Andrew V. Jones (Ed.)

Imperial College Computing Student Workshop

Proceedings of ICCSW '11
September 29th–30th 2011, London UK

Imperial College
London

Volume Editor

Andrew V. Jones
Verification of Autonomous Systems Group
Department of Computing
Imperial College London
180 Queen's Gate, London, SW7 2AZ
United Kingdom
Email: andrewj@doc.ic.ac.uk

Preface

This volume contains the proceedings of the inaugural Imperial College Student Workshop (ICCSW '11). The workshop took place on 29th–30th September 2011 in London UK. The event was hosted by Imperial College London.

ICCSW '11 aimed to be a truly student-run workshop: all of the organisation was lead by students and the majority of the submitting authors acted as reviewers for other papers. Both the quality and number of the submissions, as well as the high standard of reviews, exceeded the committee's initial expectations.

These proceedings contain 16 original contributions in various fields from across computer science, including both theoretical and applied papers.

Both days of the workshop featured a keynote talk on a facet of computer science. The talks were titled:

- *Artificial Intelligence and Human Thinking*, by Robert Kowalski (Imperial College London); and
- *Building and Operating Products at Google-scale*, by Ollie Cook (Google Inc.)

The workshop also featured two discussion panels. These panel sessions discussed various areas of computer science from a provocative angle, to allow for stimulating and spontaneous discourse on a given topic. The discussion panels focused on two areas:

- *The Future of Machine Learning*, featuring Aldo Faisal (Imperial College London), Artur Garcez (City University London), Stephen Muggleton (Imperial College London), Daniel Roy (University of Cambridge) and Alessandra Russo (Imperial College London); and
- *Software Engineering in the Age of Multi-Scale Computing*, featuring Ollie Cook (Google Inc.), Sam Dutton (Google Inc.) and Juan Silveira (Google Inc.)

ICCSW '11 proved to be an international event, attracting both submissions and attendees from the UK and the rest of Europe. At the time of writing we are pleased to host over 60 participants from the following countries: Cyprus, France, Greece, Italy, the Netherlands, Romania and the United Kingdom.

The organisation of ICCSW '11 would not have been possible without the resources and financial support provided by Imperial College London and Google Inc. The latter provided travel bursaries for students and committee members coming from institutions outside of London. Google Inc. also provided one of the keynote talks and members for one of the discussion panels. For this support we are truly grateful!

September 2011
London

Andrew V. Jones

Conference Organisation

Steering Committee

Xiuyi Fan	Department of Computing, Imperial College London
Petr Hosek	Department of Computing, Imperial College London
Andrew V. Jones	Department of Computing, Imperial College London
Nicholas Ng	Department of Computing, Imperial College London
Sören Preibusch	University of Cambridge Computer Laboratory
Reuben Rowe	Department of Computing, Imperial College London
Malte Schwarzkopf	University of Cambridge Computer Laboratory
Pia-Ramona Wojtinnek	Oxford University Computing Laboratory

Advisory Committee

Krysia Broda	Department of Computing, Imperial College London
Naranker Dulay	Department of Computing, Imperial College London

Sponsors

Google Inc.
Imperial College London

Additional Reviewers

Ekaterina Abramova	Department of Computing, Imperial College London
Konstantinos Barlas	National Technical University of Athens
Ping-Lin Chang	Department of Computing, Imperial College London
Leo De Penning	TNO Behaviour and Societal Sciences
Marco Diciolla	University of Oxford
Qinquan Gao	Department of Computing, Imperial College London
Marcel C. Guenther	Department of Computing, Imperial College London
Evgenios Hadjisoteriou	University of Cyprus
Bihan Jiang	Department of Computing, Imperial College London
Spyros Komninos	National Technical University of Athens
Katerina Ksystra	National Technical University of Athens
Bjoern Lellmann	Department of Computing, Imperial College London
Stefan Minica	University of Amsterdam
Alex Muscar	Department of Computer Science, University of Craiova
Filipa Peleja	Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa

Table of Contents

Keynote Talks

Accepted Papers

Building and Operating Products at Google-scale

Ollie Cook

Google Inc.

Abstract. Building products for Google-scale presents some unique challenges in the software development and operational spheres. Through exploring techniques relating to design, development, configuration, deployment and monitoring the talk will describe how Google delivers high availability products with low latency to millions of customers worldwide.

Profile. Ollie Cook is a site reliability manager at Google, managing the operational teams supporting Google Calendar and real-time communications. His teams focus on product and system availability, monitoring, capacity planning and deployment. Prior to joining Google in 2011, Ollie was the technology services manager at Betfair, an online gambling provider, specialising in low-latency trading systems, and deployment and configuration management of their global computing footprint.

Andrew V. Jones (Ed.): Proceedings of the 1st Imperial College Student Workshop (ICCSW '11), pp. 1–1. September 29th–30th 2011, London UK.

Artificial Intelligence and Human Thinking

Robert Kowalski

Imperial College London, Department of Computing,
Exhibition Road, London SW7 2AZ, UK

Abstract. Research in AI has built upon the tools and techniques of many different disciplines, including formal logic, probability theory, decision theory, management science, linguistics and philosophy. However, the application of these disciplines in AI has necessitated the development of many enhancements and extensions. Among the most powerful of these are the methods of computational logic.

I will argue that computational logic, embedded in an agent cycle, combines and improves upon both traditional logic and classical decision theory. I will also argue that many of its methods can be used, not only in AI, but also in ordinary life, to help people improve their own human intelligence without the assistance of computers.

Profile. Professor Robert Anthony Kowalski is a Senior Research Investigator and Emeritus Professor of Computational Logic at the Department of Computing of the Imperial College London. Professor Kowalski is recognised for his contributions to logic for knowledge representation and problem solving, including his pioneering work on automated theorem proving and logic programming.

Andrew V. Jones (Ed.): Proceedings of the 1st Imperial College Student Workshop (ICCSW '11), pp. 2–2. September 29th–30th 2011, London UK.

Combining Markov Decision Processes with Linear Optimal Controllers

Ekaterina Abramova, Daniel Kuhn, and Aldo Faisal

Imperial College London, Department of Computing,
Exhibition Road, London SW7 2AZ, UK

Abstract. *Linear Quadratic Gaussian (LQG) control has a known analytical solution [1] but non-linear problems do not [2]. The state of the art method used to find approximate solutions to non-linear control problems (iterative LQG) [3] carries a large computational cost associated with iterative calculations [4]. We propose a novel approach for solving non-linear Optimal Control (OC) problems which combines Reinforcement Learning (RL) with OC. The new algorithm, RLOC, uses a small set of localized optimal linear controllers and applies a Monte Carlo algorithm that learns the mapping from the state space to controllers. We illustrate our approach by solving a non-linear OC problem of the 2-joint arm operating in a plane with two point masses. We show that controlling the arm with the RLOC is less costly than using the Linear Quadratic Regulator (LQR). This finding shows that non-linear optimal control problems can be solved using a novel approach of adaptive RL.*

Keywords: Reinforcement Learning, non-linear Optimal Control, locally linear approximations, Linear Quadratic Regulator, robotic arm.

1 Introduction

Optimal Control (OC) theory aims to find a control law which would manipulate a dynamical system while minimizing a cost associated with that system. Non-linear OC deals with systems involving non-linear dynamics and is known to be the most difficult area of the control theory [5]. Difficulties in solution of Hamilton-Jacobi-Bellman partial differential equation for this class of problems [2] resulted in use of iterative methods which yield approximate solutions.

Main methods include: Receding Horizon Control (RHC) [6], Control Lyapunov Functions (CLFs) [7], Differential Dynamic Programming (DDP) [8], [9], Iterative Linear Quadratic Regulator (iLQR) [10] and Iterative Linear Quadratic Gaussian control (iLQG) [3]. The CLF method can lack stability unless it fits closely with the value function and RHC can act suboptimally. The DDP, iLQR and iLQG involve iterative calculations and are computationally costly.

We propose an alternative general method where the adaptive properties of Reinforcement Learning (RL) are combined with the power of OC. This method, RLOC, would be applicable in many different fields as long as the system dynamics and costs can be formally stated or approximated.

Andrew V. Jones (Ed.): Proceedings of the 1st Imperial College Student Workshop (ICCSW '11), pp. 3–9. September 29th–30th 2011, London UK.

RL involves an agent that interacts with the world through actions and receives corresponding rewards [11] thus improving the agent's behaviour. RL problems that satisfy the Markov Property are called Markov Decision Processes (MDPs). An MDP represents a framework used to define control problems.

2 Problem Formulation

2.1 Linear Optimal Control

The Linear Quadratic Regulator (LQR) control has a closed form solution [12]. Its dynamics are linear in $\mathbf{x}$ and costs are assumed to be quadratic. The deterministic linear optimal control problem has the following form

$$\mathbf{x}_{k+1} = A\mathbf{x}_k + B\mathbf{u}_k \tag{1}$$

where $\mathbf{x}$ is the state vector, $\mathbf{u}$ is the control vector, A is the system matrix acting on the state vector and B is the control gain matrix acting on the control vector.

The continuous system dynamics $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$, where $\dot{\mathbf{x}}$ is the first derivative of the state vector with respect to time, can be represented in discrete form yielding the update rule (Forward Euler)

$$\begin{aligned} \frac{\mathbf{x}_{k+1} - \mathbf{x}_k}{\Delta t} &= \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + \Delta t \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) \end{aligned}$$

where k is the step number.

The total cost, J, is the overall cost, accumulated over finite horizon through incremental (J_k) and final (J_n) cost rates

$$J = \frac{1}{2}{\mathbf{x}_n}^{\mathrm{T}} Q^f \mathbf{x}_n + \sum_{k=0}^{n-1} (\frac{1}{2}{\mathbf{u}_k}^{\mathrm{T}} R \mathbf{u}_k + \frac{1}{2}{\mathbf{x}_k}^{\mathrm{T}} Q \mathbf{x}_k) \tag{2}$$

where n is the number of steps, R is the control cost matrix, Q is the state cost matrix and Q^f is the final state cost matrix.

The control update is linear in $\mathbf{x}$ and is calculated using

$$\mathbf{u}_k = -L_k \mathbf{x}_k \tag{3}$$

where $L_k = ((R + B^{\mathrm{T}} V_{k+1} B)^{-1} B^{\mathrm{T}} V_{k+1} A)$ is the feedback gain matrix and $V_k = Q + A^{\mathrm{T}} V_{k+1} A - A^{\mathrm{T}} V_{k+1} B (R + B^{\mathrm{T}} V_{k+1} B)^{-1} B^{\mathrm{T}} V_{k+1} A$ is the cost to go function. The L matrix does not depend on $\mathbf{x}$ and therefore can be computed offline.

2.2 Non-Linear Optimal Control

The deterministic non-linear optimal control problem we choose to study has the following general form

$$\mathbf{x}_{k+1} = A(\mathbf{x}_k)\mathbf{x}_k + B(\mathbf{x}_k)\mathbf{u}_k \tag{4}$$

It does not have an analytical solution [2] and approximate solutions are computed using iterative methods.

2.3 Proposition

Traditional OC methods used for solving non-linear problems need as many locally linear controllers as there are incremental number of steps. We propose approximating non-linear OC systems by using a reduced number of linear optimal controllers which are joined together for global use with a RL algorithm.

Our problem formulation operates in two spaces: Formulation Space (φ_1) and Process Space (φ_2). The φ_1 describes every aspect of the OC problem and incorporates the Formulation States (FS). The φ_2 describes every aspect of the Markov Decision Process and incorporates the Process States (PS).

The non-linear control problem is converted into a finite MDP problem by:

1 Discretizing the continuous FS (i.e. variable $\mathbf{x}$) into a finite number of equal discrete states that correspond to the PS $\{s_1, s_2, ..., s_n\}$.
2 Using a Linear Quadratic Regulator (LQR) to obtain a small set of localized optimal linear controllers, specifically the feedback gain matrices $L_1, L_2, ..., L_n$.
3 Using a RL algorithm to learn the mapping from the PS to controllers (i.e. optimal way of combining localized linear controllers).

3 Application: Modelling Human Arm

Human motor coordination is well predicted by optimal control [13] and the approximations to non-linear human arm dynamics have been extensively studied [3], [10], [14], [15]. We illustrate RLOC by solving a non-linear optimal control problem of a simplified human arm model (Fig. 1).

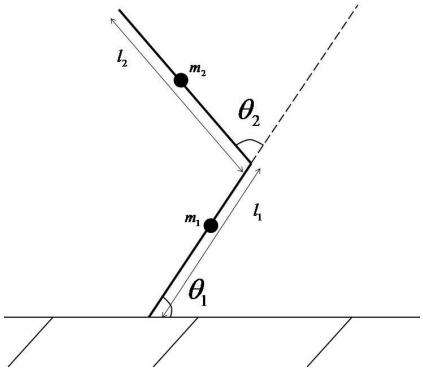

Fig. 1. Simulated arm representation where θ_1 is the angle of the shoulder, θ_2 is the angle of the elbow, both defined on the $[0°, 180°]$ continuous interval, l_1 is the length of the first link, l_2 is the length of the second link, m_1 and m_2 are the positions of the weight on each respective link. Arm is allowed to move in a horizontal plane.

The forward arm dynamics are described by the equation

$$\ddot{\theta} = \mathcal{M}(\theta)^{-1}(\tau - \mathcal{C}(\theta, \dot{\theta}) - \mathcal{B}\dot{\theta}) \tag{5}$$

where $\theta \in \mathbb{R}^2$ is the joint angle vector (shoulder: θ_1, elbow θ_2), $\mathcal{M}(\theta)$ is a PD symmetric inertia matrix, $\mathcal{C}(\theta, \dot{\theta}) \in \mathbb{R}^2$ is a vector centripetal and Coriolis forces, $\mathcal{B} \in \mathbb{R}^{2\times 2}$ is the joint friction matrix and $\tau \in \mathbb{R}^2$ is the joint torque (defined to be the control $\mathbf{u} = \tau$) [14].

The FS is defined as a 4D vector containing joint angles and their velocities

$$\mathbf{x} = (\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2)^{\mathrm{T}} \tag{6}$$

The state dynamics are described as the first derivative of the state vector, $\dot{\mathbf{x}}$

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) = (\dot{\theta}_1, \dot{\theta}_2, \ddot{\theta}_1, \ddot{\theta}_2)^{\mathrm{T}} \tag{7}$$

3.1 Simulation

The arm joint angles are discretized into 36 PS states. Six feedback gain matrices, corresponding to RL actions, are obtained by linearizing the arm dynamics using LQR approach. The linearizations are performed around equally spaced points in the φ_1 and vary in elbow angle only. This is due to the fact that non-linearity of the arm depends only on the elbow angle (Fig. 2).

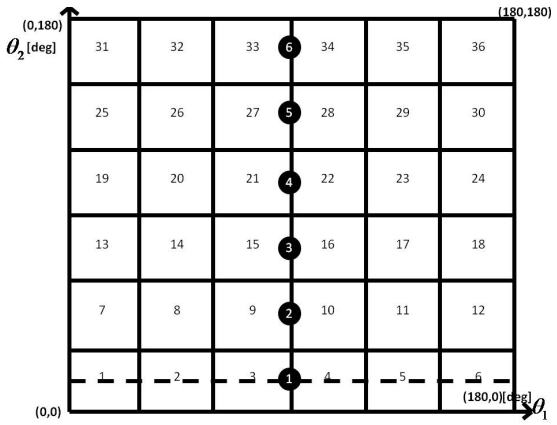

Fig. 2. The $x - axis$ is the shoulder angle, $y - axis$ is the elbow angle. The squares show PS and the black circles point to locations of feedback gain matrices obtained by linearization of arm dynamics. The dashed line shows the φ_1 along which the L matrices are equal, this is true for any such line parallel to the $x - axis$.

We choose an epsilon-greedy on-policy Monte Carlo algorithm [11]. It learns from sampling sequences of states s, actions a and immediate rewards (costs) r. A full sample path from a starting to an absorbing state is called a trace, Γ, we use 20000 traces. Optimal policy is learnt for the trajectory of the arm from the center of any of the 36 states to a target.

Optimal Policy is learnt using the following steps:

1 Initialize a deterministic policy having equal probability of picking any action (linearized controller).
2 For each trace: a) pick a random starting Process State and b) chose current action corresponding to that Process State (determined by the policy).
3 Get a trace: a) start controlling the arm using LQR, b) once enter a new Process State record the {s,a,r} triplet, c) pick a new controller at random, d) repeat until reach the target (i.e. end of obtaining a trace).
4 Examine {s,a,r} triplets to calculate Action-Value Function Q. Improve the Policy by altering probabilities of picking each action in each state. If the Policy yields lower trace cost, store it as the Optimal Policy.
5 Repeat steps 2. to 4. until obtain a specified number of traces (in our case 20000).

3.2 Results and Discussion

The results show that using RLOC algorithm to optimally control the arm from any of the 36 states to an arbitrary target of $(40°, 10°)$ results in either equal to or better performance than the LQR (Fig. 3). The results are presented as 'relative differences' in cost using the following formula $\frac{(J_{LQR} - J_{RLOC})}{|J_{LQR}|}100\%$. This is necessary since some starting states are further away from the target and hence the acquired cost would be higher simply due to the distance travelled. Negative values indicate that RLOC is less costly.

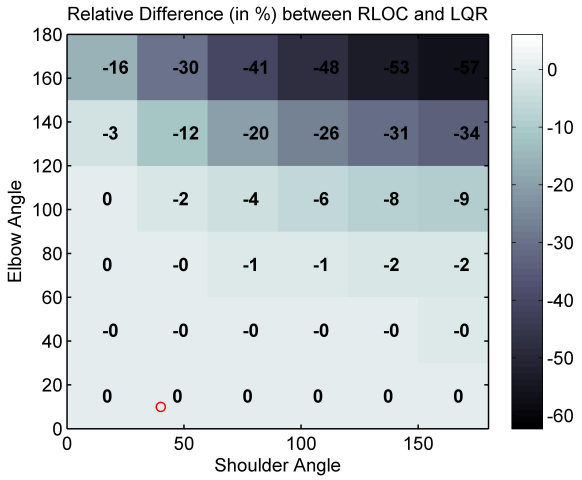

Fig. 3. Relative cost differences in % between the cost of RLOC and LQR (both calculated using J in equation (2). Each square corresponds to a PS, numbered as in (Fig 2) and the target is marked by a circle. The following cost matrices were used: $R = diag[1\ 1]$, $Q = diag[1.5\ 1.5\ 0\ 0]$ and $Q_f = diag[3000\ 3000\ 300\ 300]$.

The total costs accumulated during the MC simulation are shown in (Fig. 4). The algorithm 'learns' throughout the simulation. This can be seen by the decrease in the cost size experienced at the end of each trace for each of the starting states (picked at random at the beginning of a trace). The agent learns a better policy with each sampled trace which results in decreasing cost size as the simulation progresses. Note that the graph has distinctive reduction in the worst cost experienced for each starting state and these transform into lines as the algorithm learns. Each line represents minimal possible cost that could be incurred by controlling the arm from each starting state to the target under RLOC algorithm.

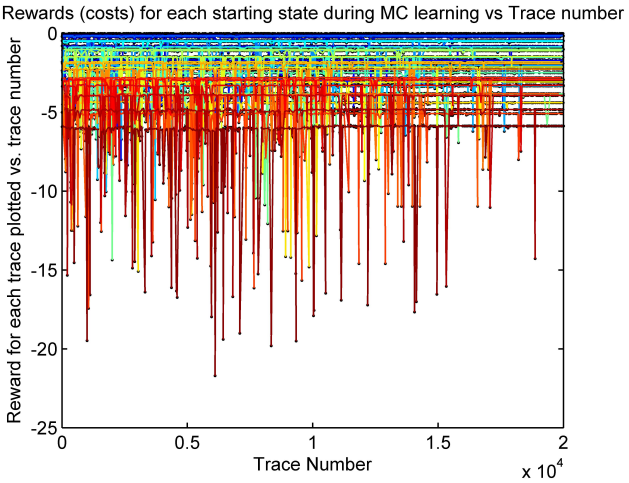

Fig. 4. Plot of the total cost of each trace vs. the trace number (20000 traces obtained). Each starting Process State cost variability is marked with a different line. As the simulation progresses, the variability of the maximum cost encountered by the learner is reduced. Therefore this figure demonstrates that the algorithm learns to use better (less costly) controllers for each Process State.

3.3 Conclusions

We presented the theoretical formulation, supported by a practical example, for the novel approach of solving non-linear optimal control problems by combining RL with OC to produce a new algorithm RLOC. The use of the RL agent is advantageous since the learner is able to explore a large amount of states, experiencing various state-action scenarios. This allows the learner to pick the control policy which provides it with the most overall reward (least cost).

We illustrated the proposed algorithm with a model of the human arm movement, where a combination of LQR control and epsilon greedy on policy Monte

Carlo method was used. The algorithm proved to be able to control the arm from the moment of initialization. This allows a trade off between speed and optimality, which may be desirable in practice for on-line computations. The algorithm was able to reach the desired target in a smooth manner with less accumulated cost than the LQR. This finding is significant because it is the first time it has been shown that RL can be combined with OC to produce better results than using LQR or RL alone. We have therefore taken a step closer to improving our current ability to control complicated non-linear systems.

An important aspect of this research is that it can be applied to many real life problems (such as drug delivery, space exploration, algorithmic trading, air traffic control and robotic control).

References

1. E. Todorov. Optimal control theory. *Bayesian brain: probabilistic approaches to neural coding*, pages 269–298, 2006.
2. A.E. Bryson and Y.C. Ho. *Applied optimal control: optimization, estimation, and control*. Hemisphere Pub, 1975.
3. E. Todorov and W. Li. A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *American Control Conference, 2005. Proceedings of the 2005*, pages 300–306. IEEE, 2005.
4. D. Mitrovic, S. Klanke, and S. Vijayakumar. Adaptive optimal feedback control with learned internal dynamics models. *From Motor Learning to Interaction Learning in Robots*, pages 65–84, 2010.
5. J.A. Primbs, V. Nevistić, and J.C. Doyle. Nonlinear optimal control: A control lyapunov function and receding horizon perspective. *Asian Journal of Control*, 1(1):14–24, 1999.
6. WH Kwon, AM Bruckstein, and T. Kailath. Stabilizing state-feedback design via the moving horizon method. *International Journal of Control*, 37(3):631–643, 1983.
7. E.D. Sontag. Lyapunov-like characterization of asymptotic controllability. *SIAM J. CONTR. OPTIMIZ.*, 21(3):462–471, 1983.
8. D.H. Jacobson. New second-order and first-order algorithms for determining optimal control: A differential dynamic programming approach. *Journal of Optimization Theory and Applications*, 2(6):411–440, 1968.
9. D. Jacobson. Differential dynamic programming methods for solving bang-bang control problems. *Automatic Control, IEEE Transactions on*, 13(6):661–675, 1968.
10. W. Li and E. Todorov. Iterative linear-quadratic regulator design for nonlinear biological movement systems. In *Proceedings of the First International Conference on Informatics in Control, Automation, and Robotics*, pages 222–229. Citeseer, 2004.
11. R.S. Sutton and A.G. Barto. *Reinforcement learning*, volume 9. MIT Press, 1998.
12. B.D.O. Anderson and J.B. Moore. *Optimal control: linear quadratic methods*. Prentice-Hall Englewood Cliffs (NJ):, 1989.
13. C.M. Harris and D.M. Wolpert. Signal-dependent noise determines motor planning. *Nature*, 394(6695):780–784, 1998.
14. W. Li and E. Todorov. Iterative linearization methods for approximately optimal control and estimation of non-linear stochastic system. *International Journal of Control*, 80(9):1439–1453, 2007.
15. M.T. Mason. *Mechanics of robotic manipulation*. The MIT Press, 2001.

Neural-Symbolic Cognitive Agents: Architecture and Theory[1]

H.L.H. de Penning, A.S. d'Avila Garcez, L.C. Lamb, J-J.C. Meyer

TNO Behaviour and Societal Sciences, Soesterberg, The Netherlands
Department of Computing, City University, London, UK
Instituto de Informática, UFRGS, Porto Alegre, Brazil
Department of Information and Computing Sciences, Utrecht University, The Netherlands
leo.depenning@tno.nl, aag@soi.city.ac.uk, lamb@inf.ufrgs.br, jj@cs.uu.nl

Abstract. In real-world applications, the effective integration of learning and reasoning in a cognitive agent model is a difficult task. However, such integration may lead to a better understanding, use and construction of more realistic models. Unfortunately, existing models are either oversimplified or require much processing time, which is unsuitable for online learning and reasoning. Currently, controlled environments like training simulators do not effectively integrate learning and reasoning. In particular, higher-order concepts and cognitive abilities have many unknown temporal relations with the data, making it impossible to represent such relationships by hand. We introduce a novel cognitive agent model and architecture for online learning and reasoning that seeks to effectively represent, learn and reason in complex real-world applications. The agent architecture of the model combines neural learning with symbolic knowledge representation. It is capable of learning new hypotheses from observed data, and inferring new beliefs based on these hypotheses. Furthermore, it deals with uncertainty and errors in the data using a Bayesian inference model. The model has successfully been applied in real-time simulation and visual intelligence systems.

Keywords: Neural-Symbolic, Cognitive Agent, Restricted Boltzmann Machine (RBM), Temporal Logic.

1 World Problem

The effective integration of automated learning and cognitive reasoning in real-world applications is a difficult task [1]. Usually, most applications deal with large amounts of data observed in the real-world containing errors, missing values and inconsistencies. Even in controlled environments, like training simulators, integrated learning and reasoning is not very successful [2], [3]. Although the use of training

[1] This paper summarizes and clarifies previous work on Neural-Symbolic Cognitive Agents appeared in the proceedings of IJCAI [12] and NeSy [17]. Also it includes a proof of soundness of the NSCA model.

Andrew V. Jones (Ed.): Proceedings of the 1st Imperial College Student Workshop (ICCSW '11), pp. 10–16. September 29th–30th 2011, London UK.

simulators simplifies the data and knowledge acquisition, it is still very difficult to construct a cognitive model of an (intelligent) agent that is able to deal with the many complex relations in the observed data. When it comes to the assessment and training of high-order cognitive abilities (e.g. leadership, tactical manoeuvring, safe driving, etc.) training is still guided or done by human experts [4]. The reason is that expert behaviour on high-level cognition is too complex to model, elicit and represent in an automated system. There can be many temporal relations between low and high-order aspects of a training task. Human behaviour is often non-deterministic and subjective (i.e. biased by personal experience and other factors like stress or fatigue) and what is known is often described vaguely and limited to explicit (i.e. "explainable") behaviour.

2 Knowledge Problem

Several attempts have been made to tackle the problems described in section 1. For instance [5] describes a number of systems that use machine learning to learn the complex relations from observation of experts and trainees during task execution. Although these systems are successful in learning and generalization, they lack the expressive power of logic-based (symbolic) systems and are therefore difficult to understand and validate [6]. Alternatively, one could add probabilistic reasoning to logic-based systems [3]. These systems perform better in expressing their internal knowledge as they are logic based and are able to deal with inconsistencies in the data because they reason with probabilities. Unfortunately, when it comes to knowledge representation and modelling these systems still require either statistical analysis of large amounts of data or knowledge representation by hand. Therefore, both approaches are time expensive and are not appropriate for use in real-time applications, which demand online learning and reasoning.

In this paper, we present a new cognitive agent model that is able to: (i) learn complex temporal relations from real-world observations, (ii) reason probabilistically about the knowledge that has been learned and/or encoded, and (iii) represent the agent's knowledge in symbolic form for explanation and validation.

3 Theoretical Relevance

The construction of effective cognitive agent models is a long standing research endeavour in artificial intelligence, cognitive science, and multi-agent systems [1], [7]. One of the main challenges toward achieving such models is the provision of integrated cognitive abilities, such as learning, reasoning and knowledge representation. Recently, cognitive computational models based on artificial neural networks have integrated inductive learning and deductive reasoning, see e.g. [8], [9]. In such models, neural networks are used to learn and reason about (an agent's) knowledge about the world, represented by symbolic logic. In order to do so, algorithms map logical theories (or knowledge about the world) T into a neural network N which computes the logical consequences of T. This provides also a learning system in the network that can be trained by examples using T as background

knowledge. In agents endowed with neural computation, induction is typically seen as the process of changing the weights of a network in ways that reflect the statistical properties of a dataset, allowing for generalizations over unseen examples. In the same setting, deduction is the neural computation of output values as a response to input values *(stimuli* from the environment) given a particular set of weights. Such network computations have been shown equivalent to a range of temporal logic formalisms [10]. Based on this approach the agent architecture of our model can be seen as a *Neural Symbolic Cognitive Agent* (NSCA). In our model, the agent architecture uses temporal logic as theory T and a Restricted Boltzmann Machine (RBM) as neural network N. A RBM is a partially connected neural network with two layers, a visible V and a hidden layer H, and symmetric connections W between these layers [11].

A RBM defines a probability distribution $P(V=v, H=h)$ over pairs of vectors v and h encoded in these layers, where v encodes the input data in binary or real values and h encodes the posterior probability $P(H \mid v)$. Such a network can be used to infer or reconstruct complete data vectors based on incomplete or inconsistent input data and therefore implement an auto-associative memory. It does so by combining the posterior probability distributions generated by each unit in the hidden layer with a conditional probability distribution for each unit in the visible layer. Each hidden unit constrains a different subset of the dimensions in the high-dimensional data presented at the visible layer and is therefore called an expert on some feature in the input data. Together, the hidden units form a so-called "Products of Experts" model that constrains all the dimensions in the input data.

4 The Cognitive Model and Agent Architecture

The Neural-Symbolic Cognitive Agent (NSCA), depicted in figure 1, uses a Recurrent Temporal Restricted Boltzmann Machine (RTRBM) to encode prior knowledge, reason with this knowledge (deduction), infer beliefs about observations (abduction) and learn new knowledge from observations (induction) [12]. In this paper we will proof that the model can encode symbolic rules R, in the form of temporal logic clauses, as a joint probability distribution on hypotheses H (represented by the hidden units) and beliefs B (represented by the visible units), and that the model is able to encode temporal relations between hypotheses. The latter is possible due to recurrent connections between hidden unit activations at time t and the activations at time t-1, see [13].

Deduction in the RTRBM is similar to Bayesian inference, where for all hypotheses H the probability is calculated that the hypotheses are true given the observed beliefs b and the previously applied hypotheses H^{t-1} (i.e. $P(H|B=b,H^{t-1})$). From this posterior probability distribution the RTRBM selects the most likely hypotheses h using random Gaussian sampling, i.e. $h \sim P(H|B=b,H^{t-1})$. Via **abduction** the RTRBM then infers the most likely beliefs based on h by calculating the conditional probability (i.e. $P(B|H=h)$). The differences between the observed and inferred beliefs are then used by the NSCA to determine the implications of the applied hypotheses.

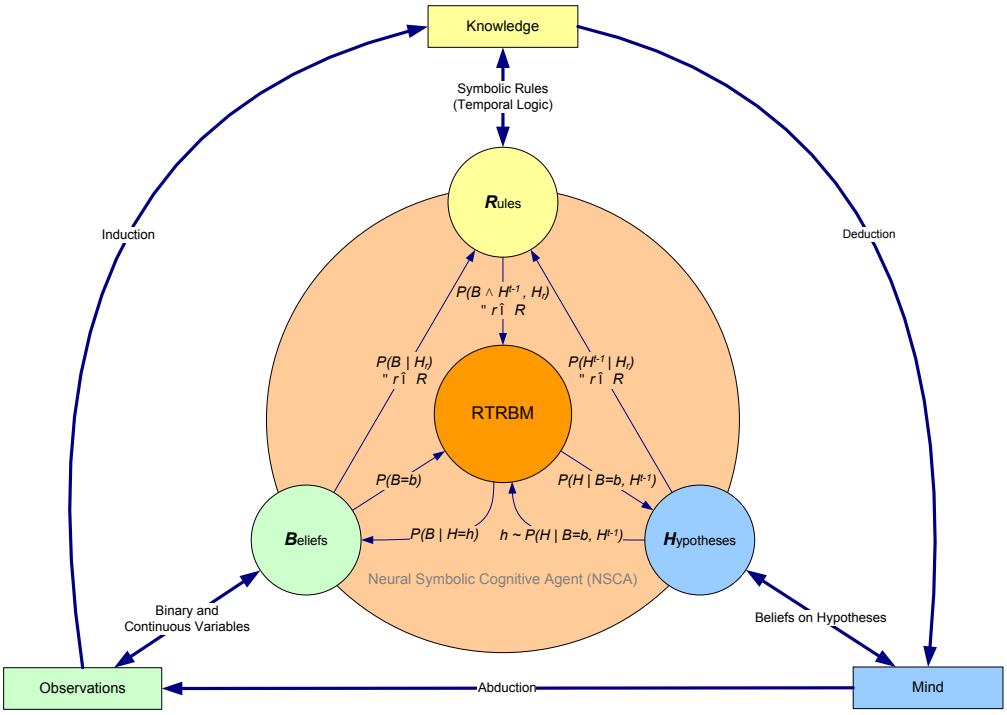

Fig. 1. Neural-Symbolic Cognitive Agent Architecture.

Induction of new knowledge can be obtained by using the difference to improve the hypotheses about the observed beliefs. It does so by updating the weights in the RTRBM using Contrastive Divergence and Backpropagation-Through-Time [13].

The NSCA architecture also enables the modelling of higher-order temporal relations using the probabilities on hypotheses (depicted as the current state of 'mind' in figure 1) of lower-level NSCAs as observations. Such a layered network of NSCAs is called a Deep Belief Network (or Deep Boltzmann Machine when RBMs are used) and are in theory capable of learning and reasoning with first-order logic [14].

5 Temporal Knowledge Representation

The symbolic rules *R,* encoded in the RTRBM, are typically in the form of temporal logic clauses that describe equivalences between hypotheses and beliefs over time. For example, $H_1 \leftrightarrow B_1 \wedge B_3 \wedge \bullet H_1$ denotes that hypothesis H_1 holds at time t if and only if beliefs B_1 and B_3 hold at time t and hypothesis H_1 holds at time t-1, where we use the *previous time* temporal logic operator $\bullet$ to denote t-1. We consider a broad set of past and future temporal logic operators as described in [10], that also describes a set of translations that relate a range of temporal logic formula having both past and future operators to a form having only the *previous time* operator. This enables a range of temporal logic formula to be encoded in and extracted from a RTRBM as described in the following algorithms and theorem.

Extraction Algorithm: Based on [15] we can extract a temporal logic program for R from a RTRBM N by finding the states of N that lower the total energy in its energy function. This means finding the states that maximize the likelihood of each clause r in R encoded in N. Assuming N is stable we can extract these states by assuming the hypothesis related to r, denoted by H_r, is true and then infer the related beliefs b and previous time formula h^{t-1} from the RTRBM using random Gaussian sampling of the conditional probability distribution (i.e. $\forall r \in R$: $b_r \sim P(B \mid H_r)$ and $h_r^{t-1} \sim P(H^{t-1} \mid H_r)$).

Similar to Pinkas, we calculate a confidence parameter c_r to denote the strength of the equivalence in each clause r. This confidence parameter is based on the notion of Bayesian credibility [16] and calculated in a similar way (see Eq. 4).

If we do this for all clauses, we can construct a temporal logic formula P using the following equations (where k is the number of beliefs, m the number of hypotheses and w_{ij} is the weight of the connection between the related visible units and hidden units in the RTRBM):

$$P = \left\{ \left\langle c_r : H_r \leftrightarrow \bigwedge_{i=1}^{k} \varphi_r^{(i)} \wedge \bigwedge_{l=1}^{m} \rho_r^{(l)} \right\rangle, \forall r \in R \right\} \tag{1}$$

$$\varphi_j^{(i)} = \begin{cases} B_i \le b_j(i) & ,if \quad w_{ij} < 0 \\ B_i \ge b_j(i) & ,if \quad w_{ij} > 0 \\ \varnothing & ,if \quad w_{ij} = 0 \end{cases} \tag{2}$$

$$\rho_j^{(l)} = \begin{cases} \bullet H_l & ,if \quad h_j^{t-1}(l) = 1 \\ \bullet \neg H_l & ,if \quad h_j^{t-1}(l) = 0 \end{cases} \tag{3}$$

$$c_r = P(H_r \mid b_r, h_r^{t-1}) \tag{4}$$

The literals for the beliefs, denoted by $\varphi_j^{(i)}$, are calculated using Eq. 2 and depend on the weight w_{ij} of the connection between the hidden unit that represents hypothesis H_j and the visible unit that represents belief B_i. A negative weight will increase the probability of H_j when we decrease the value of B_i. So all values for belief B_i less or equal to $b_j(i)$ will increase the probability of hypothesis H_j. The inverse applies to a positive weight. When the weight is zero a belief has no influence on the hypothesis and can be left out. The previous time literals for the hypotheses, denoted by $\rho_j^{(i)}$, are calculated using Eq. 3 and use the temporal operator $\bullet$. Notice that the previous time literals do not use equality operators, since h_r^{t-1} is always sampled from the binary stochastic hidden units, whereas, beliefs b_r are sampled from the continues stochastic visible units and therefore use equality operators to describe restrictions in the continuous data for which the clause applies.

Encoding Algorithm: The extraction algorithm above shows that temporal logic clauses can be extracted from the RTRBM efficiently. Encoding these clauses is the dual of the extraction algorithm, i.e. for each clause r in R; (i) add a hidden unit to the RTRBM to represent the hypothesis H_r in the clause and for each belief literal B_i in the clause, add a visible unit, (ii) randomize the weights connecting the visible and hidden units, and (iii) minimize the difference between $P(H_r \mid B=b_r, H^{t-1}=h_r^{t-1})$ and

confidence c_r of the clause, and the differences between b_r and $P(B \mid H_r=c_r)$ and h_r^{t-1} and $P(H^{t-1} \mid H_r=c_r)$ by applying the contrastive divergence algorithm [13].

Theorem: For any temporal logic program P there exists a RTRBM N such that N computes P.

Proof: The soundness of the encoding of temporal formulas w.r.t. a temporal logic programming fixed-point semantics is shown in [10]. For each rule of the form in Eq. 1, assume that a first time point t=0 exists without loss of generality. Given arbitrary initial values for the $\bullet\alpha$ formulas, we have that the computation of P in the recurrent network converges to a least fixed point [8]. Inductive step: at time point t, either N is stable with α activated in H_r or a value for α is inferred from B and H_r^{t-1}. At time point $t+1$, from the encoding algorithm, $\bullet\alpha$ will be activated in H_r^{t-1} with arbitrary confidence level c assuming minimization of the contrastive divergence [13]. This completes the proof. ■

6 Experiments and Results

Several experiments have been conducted with the NSCA in various real-world applications. For example, the NSCA has been used to learn relations between observed data from a driving simulator (e.g. positions and orientations of vehicles, gear, steering wheel angle, etc.) and high-order driving skills (e.g. safe, social and economic driving) [12]. Another application was the recognition of human behaviour (e.g. fall, chase, exchange, jump, etc.) in video based on low-level visual features (e.g. bounding box properties of detected objects) [17]. Results of these experiments have shown that the NSCA is capable of learning meaningful temporal relations from observation and extract these relations in symbolic form.

7 Conclusions and Future Work

The cognitive model and agent architecture presented in this paper offer an effective approach that integrates symbolic reasoning and neural learning in a unified model and has been successfully applied in several real-world applications. The approach allows the modelled agent to learn rules about observed data in complex, real-world environments. Learned behaviour can be extracted to update existing knowledge for validation, reporting and feedback. Furthermore the approach allows prior knowledge to be encoded in the model and deals with uncertainty in real-world data.

Future work includes research on using Deep Belief Networks [14] to deal with first-order logic and “Direction of Fit” to perform various forms of action planning and selection.

In summary, we believe that our work provides an integrated model for knowledge representation, learning and reasoning which may indeed lead to realistic computational cognitive agent models, thus answering the challenges put forward in [1], [7].

8 References

[1] L. G. Valiant, "Three problems in computer science," *Journal of the ACM (JACM)*, vol. 50, no. 1, pp. 96–99, 2003.

[2] J. Sandercock, *Lessons for the construction of military simulators: a comparison of artificial intelligence with human-controlled actors. Technical Report, DSTO TR-1614*. Adelaide, South Australia: , 2004.

[3] A. Heuvelink, "Cognitive Models for Training Simulations," Dutch Graduate School for Information and Knowledge Systems (SIKS), PhD. diss. no. 2009-24, 2009.

[4] K. van den Bosch and J. B. J. Riemersma, "Reflections on scenario-based training in tactical command," *Scaled Worlds: Development, validation and applications*, pp. 1–21, 2004.

[5] H. K. G. Fernlund, A. J. Gonzalez, M. Georgiopoulos, and R. F. DeMara, "Learning tactical human behavior through observation of human performance.," *IEEE transactions on systems, man, and cybernetics, Part B.*, vol. 36, pp. 128-140, Feb. 2006.

[6] E. Smith and S. Kosslyn, *Cognitive Psychology: Mind and Brain*. Prentice-Hall, 2006.

[7] M. Wooldridge, *An introduction to multiagent systems*, 2nd ed. Wiley, 2009.

[8] A. S. d'Avila Garcez, L. C. Lamb, and D. M. Gabbay, *Neural-Symbolic Cognitive Reasoning*. Springer-Verlag New York Inc, 2009.

[9] J. Lehmann, S. Bader, and P. Hitzler, "Extracting reduced logic programs from artificial neural networks," *Applied Intelligence*, vol. 32, no. 3, pp. 249–266, 2010.

[10] L. C. Lamb, R. V. Borges, and A. S. d'Avila Garcez, "A connectionist cognitive model for temporal synchronisation and learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2007, vol. 22, no. 1, pp. 827-832.

[11] P. Smolensky, "Information processing in dynamical systems: Foundations of harmony theory," in *Parallel Distributed Processing: Volume 1: Foundations*, vol. 1, D. E. Rumelhart and J. L. McClelland, Eds. Cambridge, MA: MIT Press, 1986, pp. 194-281.

[12] L. de Penning, A. S. d'Avila Garcez, L. C. Lamb, and J.-J. C. Meyer, "A Neural-Symbolic Cognitive Agent for Online Learning and Reasoning," in *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2011.

[13] I. Sutskever, G. E. Hinton, and G. Taylor, "The recurrent temporal restricted boltzmann machine," in *Neural Information Processing Systems (NIPS)*, 2008, vol. 21.

[14] R. Salakhutdinov and G. E. Hinton, "Deep Boltzmann machines," in *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2009, vol. 5, no. 2, pp. 448–455.

[15] G. Pinkas, "Artificial Intelligence Reasoning , nonmonotonicity and learning in connectionist networks that capture propositional knowledge," *Artificial Intelligence*, vol. 77, pp. 203-247, 1995.

[16] P. M. Lee, *Bayesian Statistics: An Introduction*, Third Ed. Arnold Publication, 1997.

[17] L. de Penning, "Visual Intelligence using Neural-Symbolic Learning and Reasoning," in *Workshop proc. on Neural Symbolic Learning and Reasoning at IJCAI 2011*, 2011.

Time-Bounded Verification of CTMCs against MTL specifications ⋆ ⋆⋆

Taolue Chen, Marco Diciolla, Marta Kwiatkowska, and Alexandru Mereacre

Department of Computer Science, Oxford University,
Wolfson Building, Parks Road, Oxford, OX1 3QD, United Kingdom

Abstract. In this paper we study time-bounded verification of a finite continuous-time Markov chain (CTMC) $\mathcal{C}$ against a real-time specification, provided as a metric temporal logic (MTL) property φ. The key question is: what is the probability of the set of timed paths of $\mathcal{C}$ that satisfy φ over a time interval of fixed, bounded length? We provide approximation algorithms to solve these problems. We first derive a bound N such that timed paths of $\mathcal{C}$ with at most N discrete jumps are sufficient to approximate the desired probability up to ε. Then, for each discrete (untimed) path σ of length at most N, we generate timed constraints over variables determining the residence time of each state along σ, depending on the real-time specification under consideration. The probability of the set of timed paths, determined by the discrete path and the associated timed constraints, can thus be formulated as a multidimensional integral. Summing up all such probabilities yields the result.

1 Introduction

Verification of *continuous-time Markov chains* (CTMCs) has received much attention in recent years [5]. Thanks to considerable improvements of algorithms, (symbolic) data structures and abstraction techniques, CTMC model checking has emerged as a valuable analysis technique. Aided by powerful software tools, it has been adopted by researchers from, e.g., systems biology, queuing networks and dependability. To mention just a few practical applications, these models have been used to quantify the throughput of production lines, to determine the mean time between failure in safety-critical systems, and to identify bottlenecks in high-speed communication networks.

The focus of CTMC model checking [4] has primarily been on checking stochastic versions of the *branching-time* temporal logic CTL, such as *continuous stochastic logic* CSL [4]. The verification of *linear temporal logic* (LTL) properties reduces to applying well-known algorithms [14,10] to embedded discrete-time Markov chains (DTMCs). Linear-time properties equipped with timing constraints have only recently been considered. In particular, [7,8] treat linear *real-time* specifications that are given as *deterministic timed automata* (DTA). These include properties of the form, "what is the probability to reach a given target state within the deadline, while avoiding unsafe states and not staying too long in any of the dangerous states on the way?". Such properties

⋆ This work is supported by the ERC Advanced Grant VERIWARE.

⋆⋆ A longer version of this paper appeared in the proceedings of FORMATS11 [9]

Andrew V. Jones (Ed.): Proceedings of the 1st Imperial College Student Workshop (ICCSW '11),
pp. 17–23. September 29th–30th 2011, London UK.

cannot be expressed in CSL nor in its dialects [3,11]. Model checking DTA properties can be done by a reduction to computing the reachability probability in a *piecewise deterministic Markov process*, based on the product construction between the CTMC and DTA [8,6]. It remains a challenge to tackle more general real-time specifications like *Metric Temporal Logics* ([1,12], MTL).

For this reason, we study the time-bounded verification problem of a CTMC $\mathcal{C}$, against a real-time specification provided as an MTL formula φ. The key question is: what is the probability of the set of timed paths of $\mathcal{C}$ that satisfy φ over a fixed time interval $[0, T]$ where $T \in \mathbb{R}_{>0}$? We provide approximation algorithms to solve these problems. Given any $\varepsilon > 0$ a priori, we first derive a bound N such that it is sufficient only to consider timed paths of $\mathcal{C}$ with at most N discrete jumps to approximate the desired probability up to ε. Then, for each *discrete* (untimed) path σ of $\mathcal{C}$ of length at most N, we generate a family of linear constraints, $\mathcal{S}$, over variables determining the residence time of each state in σ. The discrete path σ, together with the associated timing constraints $\mathcal{S}$, determines a set of *timed* paths of $\mathcal{C}$, each of which satisfies φ. The probability of this set of timed paths can be formulated as a multidimensional integral, which can be calculated by Laplace transforms. Summing up all such probabilities yields the desired result. We believe these results are of independent interest, as they have potential usage in domains such as runtime verification.

The reader should notice that even though MTL is generally undecidable [2] (if we include singular intervals), this does not affect our algorithm. In fact, informally we can state that in any CTMC $\mathcal{C}$, the probability of an event happening in a specific singular time instant is zero.

2 Preliminaries

2.1 Continuous-time Markov chains

Given a set $\mathcal{H}$, let Pr: $\mathcal{F}(\mathcal{H}) \rightarrow [0, 1]$ be a *probability measure* on the measurable space $(\mathcal{H}, \mathcal{F}(\mathcal{H}))$, where $\mathcal{F}(\mathcal{H})$ is a σ-algebra over $\mathcal{H}$. Let $Distr(\mathcal{H})$ denote the set of probability measures on this measurable space.

Definition 1 (CTMC). *A (labeled) continuous-time Markov chain* (CTMC) *is a tuple* $\mathcal{C} = (S, \mathrm{AP}, L, \alpha, \mathbf{P}, E)$ *where*

- S *is a finite set of states;*
- AP *is a finite set of atomic propositions;*
- $L : S \rightarrow 2^{\mathrm{AP}}$ *is the labeling function;*
- $\alpha \in Distr(S)$ *is the initial distribution;*
- $\mathbf{P} : S \times S \rightarrow [0, 1]$ *is a stochastic matrix; and*
- $E : S \rightarrow \mathbb{R}_{\geq 0}$ *is the exit rate function.*

Example 1. An example CTMC is illustrated in Fig. 1, where $\mathrm{AP} = \{a, b, c\}$ and s_0 is the initial state, i.e., $\alpha(s_0) = 1$ and $\alpha(s) = 0$ for any $s \neq s_0$. The exit rates are indicated at the states, whereas the transition probabilities are attached to the transitions.

In a CTMC $\mathcal{C}$, state residence times are *exponentially* distributed. More precisely, the residence time X of a state $s \in S$ is a random variable governed by a nonnegative

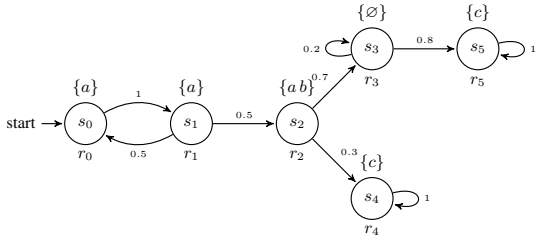

Fig. 1. An example CTMC

exponential distribution with parameter $E(s)$ (written as $X \sim \text{Exp}(E(s))$). Hence, the probability to exit state s in t time units (t.u. for short) is given by $\int_0^t E(s) \cdot e^{-E(s)\tau} d\tau$. Furthermore, the probability to take the transition from s to s' in t t.u. equals $\mathbf{P}(s, s') \cdot \int_0^t E(s) \cdot e^{-E(s)\tau} d\tau$.

Definition 2. *Given a* CTMC $\mathcal{C} = (S, \text{AP}, L, \alpha, \mathbf{P}, E)$*, we define the following notions.*

- *A (finite)* discrete *path $\sigma = s_0 \to s_1 \to s_2 \to \ldots$ is a (finite) sequence of states; we define σ_i to be the state s_i, and σ^i to be the prefix of length i of σ.*
- *A (finite)* timed *path $\rho = s_0 \xrightarrow{x_0} s_1 \xrightarrow{x_1} s_2 \xrightarrow{x_2} \ldots$, where $x_i \in \mathbb{R}_{>0}$ for each $i \geq 0$, is a sequence starting in state s_0; we define $|\rho|$ to be the length of a* finite *timed path ρ; $\rho[n] := s_n$ is the n-th state of ρ and $\rho\langle n \rangle := x_n$ is the time spent in state s_n; let $\rho@t$ be the state occupied in ρ at time $t \in \mathbb{R}_{\geq 0}$, i.e. $\rho@t := \rho[n]$, where n is the smallest index such that $\sum_{i=0}^{n} \rho\langle i \rangle \geq t$.*

Intuitively, a timed path $\rho = s_0 \xrightarrow{x_0} s_1 \xrightarrow{x_1} s_2 \xrightarrow{x_2} \ldots$ suggests that the CTMC $\mathcal{C}$ starts in state s_0 and stays in this state for x_0 t.u., and then jumps to state s_1, staying there for x_1 t.u., and then jumps to s_2 and so on. An example timed path is $\rho = s_0 \xrightarrow{3} s_1 \xrightarrow{2} s_0 \xrightarrow{1.5} s_1 \xrightarrow{3.4} s_2 \ldots$ with $\rho[2] = s_0$ and $\rho@4 = \rho[1] = s_1$.

Let $Paths^{\mathcal{C}}$ denote the set of infinite timed paths in the CTMC $\mathcal{C}$, and $Paths^{\mathcal{C}}(s)$ the set of infinite timed paths in $\mathcal{C}$ that start in s. Given a time bound $T \in \mathbb{R}_{\geq 0}$ and $N \in \mathbb{N} \cup \{\infty\}$, we define $Paths^{\mathcal{C}}_{T,<N}(s)$, to be the set of all timed paths with *at most* $N - 1$ discrete jumps in time interval $[0, T]$; and $Paths^{\mathcal{C}}_{T,\geq N}(s)$, to be the set of all timed paths with *at least* N jumps in $[0, T]$.

For notational simplicity we will omit the superscript $\mathcal{C}$ when appropriate and also we write $Paths^{\mathcal{C}}_{T}$ instead of $Paths^{\mathcal{C}}_{T,\leq\infty}$ for the set of all timed paths with an arbitrary number of jumps in $[0, T]$.

In general, computing the probability of a cylinder set with k intervals $I_0 \ldots I_{k-1}$ (i.e. k discrete jumps) reduces to calculating k integrals over $I_0 \ldots I_{k-1}$.

2.2 Metric Temporal Logic

Definition 3 (Syntax of MTL). *Let* AP *be an arbitrary nonempty, finite set of atomic propositions. Let $I = [a, b]$ be an interval such that $a, b \in \mathbb{N} \cup \{\infty\}$. The* Metric

Temporal Logic *is inductively defined as:*

$$\varphi ::= p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathcal{U}_I \varphi_2 \ ,$$

where $p \in$ AP *and* φ_1, φ_2 *are* MTL *formulas.*

We introduce the time-bounded semantics for MTL, as follows.

Definition 4 (Semantics of MTL). *Given an* MTL *formula* φ*, a time bound* T*, a timed path* ρ *and a variable* $t \in \mathbb{R}_{\geq 0}$*, the satisfaction relation* $(\rho, t) \models_T \varphi$ *is inductively defined as follows:*

$$\begin{array}{ll}
(\rho,t) \models_T p & \Leftrightarrow p \in L(\rho@t) \wedge t \leq T \\
(\rho,t) \models_T \neg\varphi_1 & \Leftrightarrow (\rho,t) \not\models_T \varphi_1 \\
(\rho,t) \models_T \varphi_1 \wedge \varphi_2 & \Leftrightarrow (\rho,t) \models_T \varphi_1 \wedge (\rho,t) \models_T \varphi_2 \\
(\rho,t) \models_T \varphi_1 \mathcal{U}_I \varphi_2 & \Leftrightarrow \exists t'.\ t \leq t' \leq T \text{ s.t. } t' - t \in I \wedge (\rho,t') \models_T \varphi_2 \wedge \\
 & \quad\ \forall t''.\ t \leq t'' < t' \Rightarrow (\rho,t'') \models_T \varphi_1
\end{array}$$

where $p \in$ AP *and* φ_1, φ_2 *are* MTL *formulas.*

3 MTL Specifications

In this section we study the problem of model checking CTMCs against MTL properties. Let $\Pr^{\mathcal{C}}_T(\varphi) := \Pr^{\mathcal{C}}(\{\rho \in Paths^{\mathcal{C}}_T \mid (\rho, 0) \models_T \varphi\})$ denote the probability that the CTMC $\mathcal{C}$ satisfies the MTL formula φ, for a given time bound T. Instead of computing $\Pr^{\mathcal{C}}_T(\varphi)$, we give a procedure to compute $\Pr^{\mathcal{C}}_{T,<N}(\varphi) := \Pr^{\mathcal{C}}(Paths^{\mathcal{C}}_{T,<N}(\varphi))$ for sufficiently large N which ensures that $\Pr^{\mathcal{C}}_T(\varphi) - \Pr^{\mathcal{C}}_{T,<N}(\varphi) < \varepsilon$ for arbitrarily small $\varepsilon \in \mathbb{R}_{>0}$. This yields an approximation algorithm. Below we present an algorithm to compute $\Pr^{\mathcal{C}}_{T,<N}(\varphi)$. We first give a sketch, and provide the crucial sub-procedures in Sec. 3.1 and Sec. 3.2.

Choose N *to get the desired error bound* ε. The first step of the algorithm is to choose the smallest N such that we get the desired error bound ε.

Compute the product $\mathcal{C} \otimes \mathcal{A}_{\widetilde{\varphi}}$. The basic idea of this step is to exclude those CTMC timed paths which definitely fail φ in order to reduce the number of paths to be analyzed. To this end, we define an LTL formula $\widetilde{\varphi}$ such that, if a discrete path of $\mathcal{C}$ fails $\widetilde{\varphi}$, then any timed path with the discrete path as the skeleton must fail φ. We then construct an NFA out of $\widetilde{\varphi}$ such that only those finite discrete CTMC paths which are accepted by the NFA are the prefixes of the potential skeletons of timed paths satisfying φ. Then we apply the standard product construction, which suffices to identify those CTMC finite discrete paths analyzed in the next step.

Any MTL formula φ can be transformed into a *positive normal form* containing only two temporal operators: $\mathcal{U}_{[a,b]}$ and $\Box_{[a,b]}$, where $(\rho, t) \models_T \Box_{[a,b]}\varphi$ iff $\forall t' \in [a, b] \Rightarrow (\rho, t + t') \models_T \varphi$.

Given any MTL φ *in positive normal form*, we define an (untimed) LTL formula $\widetilde{\varphi}$ as follows:

$$\begin{array}{ll}
\varphi = p & \Rightarrow \widetilde{\varphi} = p \\
\varphi = \neg p & \Rightarrow \widetilde{\varphi} = \neg p \\
\varphi = \varphi_1 \vee \varphi_2 & \Rightarrow \widetilde{\varphi} = \widetilde{\varphi}_1 \vee \widetilde{\varphi}_2 \\
\varphi = \varphi_1 \wedge \varphi_2 & \Rightarrow \widetilde{\varphi} = \widetilde{\varphi}_1 \wedge \widetilde{\varphi}_2 \\
\varphi = \varphi_1 \mathcal{U}_I \varphi_2 & \Rightarrow \widetilde{\varphi} = \widetilde{\varphi}_1 \mathcal{U} \widetilde{\varphi}_2 \\
\varphi = \Box_I \varphi_1 & \Rightarrow \widetilde{\varphi} = \text{TRUE}\, \mathcal{U} \widetilde{\varphi}_1
\end{array}$$

where φ_1 and φ_2 are MTL formulas and $\widetilde{\varphi}_1$ and $\widetilde{\varphi}_2$ are LTL formulas.

As the next step, we construct a *nondeterministic finite automaton* (NFA) $\mathcal{A}_{\widetilde{\varphi}}$ which accepts all the prefixes of infinite paths satisfying the formula $\widetilde{\varphi}$. The NFA can be obtained by a minor adaptation of the well-known Vardi-Wolper construction. We then build the product of $\mathcal{C}$ and $\mathcal{A}_{\widetilde{\varphi}}$ ($\mathcal{C} \otimes \mathcal{A}_{\widetilde{\varphi}}$).

Compute all the discrete paths of $\mathcal{C} \otimes \mathcal{A}_{\widetilde{\varphi}}$ of length at most N and calculate the probabilities.

1. Search the graph $\mathcal{C} \otimes \mathcal{A}_{\widetilde{\varphi}}$ to get all the discrete accepting paths σ of $\mathcal{C}$ of length at most N;
2. Run Alg. 1 on each discrete path σ of length $n \leq N$ to obtain the system of linear inequalities $\mathcal{S}$;
3. Compute the probability of $\sigma[\mathcal{S}]$ (cf. Sec. 3.2);
4. Sum up all the probabilities for each discrete path to obtain $\Pr^{\mathcal{C}}_{T,<N}(\varphi)$.

3.1 Constraints Generation

We describe the Alg. 1 that takes as input a discrete path σ of length n and an MTL formula φ and returns a family of linear constraints $\mathcal{S} = \bigvee_{i \in I} \bigwedge_{j \in J_i} c_{ij}$ where c_{ij} is a linear inequality over the set of variables $t_0, \ldots, t_{n-1}$.

Algorithm 1 Constraints generation

Require: A finite discrete path σ of length $n > 0$, an MTL formula φ and a time bound T
Ensure: Family of linear inequalities $\mathcal{S}$ over $t_0, \ldots, t_{n-1}$

$\mathcal{S}'$:=`Constr_Gen(`σ`,0,`φ`)`
$\mathcal{S}$:=`Fourier_Motzkin(`$\mathcal{S}'$`,`t_0`,...,`t_{n-1}`)`
return $\mathcal{S}$

Function `Constr_Gen(`σ`,`t`,`φ`)`
case(φ) :

$\varphi = p$	:	**return** $\left(\bigvee_{k=0}^{n} p \in L(\sigma_k) \wedge \sum_{i=0}^{k} t_i \geq t \wedge \sum_{i=0}^{k-1} t_i < t\right) \wedge t < T$
$\varphi = \neg\varphi_1$	:	$\mathcal{S}' :=$ `¬Constr_Gen(`σ`,`t`,`φ_1`)`
$\varphi = \varphi_1 \wedge \varphi_2$	:	$\mathcal{S}' :=$`Constr_Gen(`σ`,`t`,`φ_1`)` $\wedge$ `Constr_Gen(`σ`,`t`,`φ_2`)`
$\varphi = \varphi_1 \mathcal{U}_{[a,b]} \varphi_2$	:	$\mathcal{S}' := \exists t'.\big(t \leq t' < T \wedge t' - t \geq a \wedge t' - t < b \wedge$ `Constr_Gen(`σ`,`t'`,`φ_2`)` $\wedge\, \forall t''.\, t \leq t'' < t' \Rightarrow$ `Constr_Gen(`σ`,`t''`,`φ_1`)`$\big)$

return $\mathcal{S}'$

3.2 Computing Probabilities

Given a CTMC $\mathcal{C}$, a discrete path σ of length N and the family of linear constraints $\mathcal{S}(t_0, \ldots, t_{N-1})$ obtained from Alg. 1, the main task of this section is to compute the probability of $\sigma[\mathcal{S}]$, i.e., $\Pr^{\mathcal{C}}(\sigma[\mathcal{S}])$. The value of the joint probability can be computed through the following multidimensional integration:

$$\Pr^{\mathcal{C}}(\sigma[\mathcal{S}]) = \underbrace{\int \cdots \int}_{N}{}_{\mathcal{S}(t_0,\ldots,t_{N-1})} \prod_{i=0}^{N-1} E(s_i) \cdot \mathbf{P}(s_i, s_{i+1}) \times e^{-E(s_i)\tau_i} d\tau_i. \quad (1)$$

We use the algorithm of [13] (Sec. 5) to compute efficiently the multidimensional integral based on the Laplace transform.

3.3 Main Algorithm

We summarize the time-bounded verification algorithm for a CTMC $\mathcal{C}$ against an MTL formula φ in Alg. 2. Recall that Λ is the maximal exit rate appearing in $\mathcal{C}$.

Algorithm 2 Time-bounded verification of a CTMC $\mathcal{C}$ against an MTL formula φ

Require: $\mathcal{C}$, φ, T and ε
Ensure: $\Pr^{\mathcal{C}}_{T,<N}(\varphi)$
  Choose an integer $N \geq \Lambda T e^2 + \ln(\frac{1}{\varepsilon})$
  Transform φ into $\widetilde{\varphi}$ and generate NFA $\mathcal{A}_{\widetilde{\varphi}}$ out of $\widetilde{\varphi}$
  Compute the product $\mathcal{C} \otimes \mathcal{A}_{\widetilde{\varphi}}$
  for each discrete path σ of $(\mathcal{C} \otimes \mathcal{A}_{\widetilde{\varphi}}) \downarrow_1$ of length $n < N$ **do**
    Generate the family of linear constraints $\mathcal{S}(t_0, \ldots, t_{n-1})$
    Calculate the probability p of $\sigma[\mathcal{S}]$
    $\Pr^{\mathcal{C}}_{T,<N}(\varphi) := \Pr^{\mathcal{C}}_{T,<N}(\varphi) + p$
  end for
  return $\Pr^{\mathcal{C}}_{T,<N}(\varphi)$

4 Conclusion

In this paper we have studied time-bounded verification of CTMCs against real-time specifications. In particular, we presented effective procedures to approximate the probability of the set of timed paths of CTMCs that satisfy real-time specifications over a time interval of fixed bounded length, arbitrarily closely. Model checking CTMCs against linear real-time specifications has received scant attention so far. To our knowledge, this issue has only been (partially) addressed in [7,3,11].

A natural question is how to tackle the traditional (time-unbounded) verification. The scheme introduced in this paper still works. However, one cannot guarantee an approximation to stay within the given error bound ε, which means that the resulting procedure is *not* an approximation algorithm any more. We leave this as future work.

References

1. R. Alur and T. A. Henzinger. Real-time logics: Complexity and expressiveness. In *LICS*, pages 390–401, 1990.
2. R. Alur, T. Feder, and T. A. Henzinger. The benefits of relaxing punctuality. *J. ACM*, 43(1):116–146, 1996.
3. C. Baier, L. Cloth, B. R. Haverkort, M. Kuntz, and M. Siegle. Model checking Markov chains with actions and state labels. *IEEE Trans. Software Eng.*, 33(4):209–224, 2007.
4. C. Baier, B. R. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE Trans. Software Eng.*, 29(6):524–541, 2003.
5. C. Baier, B. R. Haverkort, H. Hermanns, and J.-P. Katoen. Performance evaluation and model checking join forces. *Commun. ACM*, 53(9):76–85, 2010.
6. B. Barbot, T. Chen, T. Han, J.-P. Katoen, and A. Mereacre. Efficient CTMC model checking of linear real-time objectives. In P. A. Abdulla and K. R. M. Leino, editors, *TACAS*, volume 6605 of *Lecture Notes in Computer Science*, pages 128–142. Springer, 2011.
7. T. Chen, T. Han, J.-P. Katoen, and A. Mereacre. Quantitative model checking of continuous-time Markov chains against timed automata specifications. In *LICS*, pages 309–318, 2009.
8. T. Chen, T. Han, J.-P. Katoen, and A. Mereacre. Model checking of continuous-time Markov chains against timed automata specifications. *Logical Methods in Computer Science*, 7(1–2):1–34, 2011.
9. T. Chen, M. Diciolla, M. Kwiatkowska, and A. Mereacre. Time-Bounded Verification of CTMCs against Real-Time Specifications. 9th International Conference on Formal Modeling and Analysis of Timed Systems, FORMATS 2011, Aalborg, Denmark. *Lecture Notes in Computer Science*, 6919, 2011.
10. C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *J. ACM)*, 42(4):857–907, 1995.
11. S. Donatelli, S. Haddad, and J. Sproston. Model checking timed and stochastic properties with CSL^{TA}. *IEEE Trans. Software Eng.*, 35(2):224–240, 2009.
12. R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.
13. J. B. Lasserre and E. S. Zeron. A Laplace transform algorithm for the volume of a convex polytope. *J. ACM*, 48(6):1126–1140, 2001.
14. M. Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *FOCS*, pages 327–338, 1985.

MASSPA-Modeller: A Spatial Stochastic Process Algebra modelling tool

Marcel C. Guenther and Jeremy T. Bradley

Imperial College London, 180 Queen's Gate,
London SW7 2AZ, United Kingdom,
Email: {mcg05,jb}@doc.ic.ac.uk

Abstract. We introduce *MASSPA–Modeller*, a visual modelling tool for the recently developed spatial stochastic process algebra MASSPA which describes Markovian Agent Models (MAM)s. The major advantage of using a visual editor to generate MASSPA models is that the laborious task of modelling the communication between agents is partially automated by the tool. Furthermore the tool can separately check the correctness of local and spatial aspects of the model and thereby help users to find mistakes. For the analysis of the resulting models MASSPA–Modeller uses the powerful *GPA–analyser* engine. Additionally we briefly summarise the latest developments in spatial stochastic process algebras.

Keywords: Performance Analysis, Higher Moment Analysis, Spatial Stochastic Process Algebra, Spatial Modelling, MAM, MASSPA

1 Introduction

Performance analysis studies modelling techniques for predicting performance of computer and telecommunication systems. Moreover, it is also applied in other areas such as crowd modelling, systems biology and various other fields. Common analysis targets are the average time to complete a job or the probability distribution of client service times. Most models are described in high-level languages such as PEPA[1], a Stochastic Process Algebra (SPA), which translate to Continuous Time Markov Chains (CTMC)s. Alternatively Stochastic Petri nets can be used to describe CTMCs, but in this paper we will focus on process algebras. All CTMC models assume that any delay between two events in the underlying stochastic process is exponentially distributed.

Recent developments in fluid analysis methods [2,3] for CTMCs derived from high-level model descriptions such as SPAs have given modellers means to analyse extremely large models. Previously such models could only be analysed using stochastic simulation [4], as traditional analysis techniques, which are based on linear equation solvers, can only solve small models due to the well-known state space explosion problem. Fluid techniques exploit the fact that moments

Andrew V. Jones (Ed.): Proceedings of the 1st Imperial College Student Workshop (ICCSW '11), pp. 24–30. September 29th–30th 2011, London UK.

of stochastic processes, which describe state populations in lumped CTMCs, can be approximated by Ordinary Differential Equations (ODE)s. Today, there is a large number of formalisms for which the mapping from the high-level model description to ODEs has been defined, e.g. GPEPA [3], SCCP [5] and stochastic π-calculus [6]. More recent research has shown that it is even possible to approximate measures such as passage time distributions and reward vectors using ODEs [7,8].

Spatial modelling languages are useful extensions to their non-spatial counterparts, for instance when investigating crowd movements, disaster propagation or network topologies. Evaluation techniques for spatial models are similar to those for non-spatial models, however, spatial models tend to have larger CTMCs and allow analysis techniques that take into account the spatial nature of the model. Fluid analysis techniques have been applied to CTMCs that arise from spatial models described in Bio-PEPA [9] and the MAM formalism [10] for which we defined MASSPA, a Markovian Agent Spatial Stochastic Process Algebra [11]. The MAM formalism is a particularly interesting spatial modelling framework as it has been applied in a large number of different areas such as wireless sensor networks [10], fire propagation [12] and traffic modelling [13] to name but a few. Research in [11] shows that there is a generic mapping from MASSPA to a mass-action type reaction system. As a consequence any MAM model expressed in MASSPA grammar can now be analysed using the powerful *GPA–analyser*, which was originally developed for the analysis of massive GPEPA models [14].

Despite the existence of spatial SPAs (SSPA)s, complex spatial models can still be too large to be expressed in SSPAs by hand. To facilitate the creation of spatial models, visual editors such as DrawNet [15] and SeSam [16] have been developed for stochastic Petri nets and agent based simulations. Moreover, a recent extension to DrawNet enables users to describe the spatial aspects of MAMs, but it currently does not allow users to define sequential Markovian Agents [17]. MASSPA–Modeller, which is described in this paper, is the first spatial modelling tool for MAMs that allows users to define all aspects of MA models. It further enables users to perform higher moment and continuous reward vector analysis on spatial models. Moreover, it is the first hybrid tool that allows users to express local agent behaviour using SPA and spatial behaviour in a visual editor. This paper is organised as follows. First, we briefly describe the MAM formalism and MASSPA in Sect. 2. We then demonstrate the MASSPA-Modeller work-flow in Sect. 3 and finally present conclusions in Sect. 4.

2 MAM and MASSPA

The MAM formalism was first described by Gribaudo *et al.* in [10]. Each MAM consists of two parts, a definition for local agent behaviour and a model describing their distribution and interactions in space. Agents can evolve through local and message induced transitions. While local transitions are no different from

those in other formalisms, message induced transitions are novel. Every time an agent changes state it can emit a message of type M. At the same time another agent can listen for messages of type M and act on incoming messages. The perception function $u(\cdot)$, which is part of the model, defines all combinations of agents in all locations that can exchange messages (cf. [10,11]). In other words $u(\cdot)$ can be thought of as a directed graph where vertices are pairs of agent state populations and edges are message channels between them. Note that the MAM message exchange paradigm is an asynchronous form of communication, as receiving agents can decide to discard messages without blocking sending agents. Therefore Markovian Agents (MA)s are said to be autonomous. MASSPA [11], the Markovian Agent SSPA, simplifies the definition and evaluation of MAMs and makes comparisons with other formalisms easier. A MASSPA model consists of sequential agent, topology, agent population size and $Channel(\cdot)$ definitions (e.g. Fig. 1).

```
// Agent definitions
Agent OnOff {
  On  = !(2.0,M,1.0).Off;   Off = ?(M,1.0).On + (1.5).On;
};
// Spatial model definition
Locations = {(0),(1),(2)};
// Initial agent distributions
On@(0) = 100;   Off@(1) = 200;   Off@(2) = 150;
// Channel definition
Channel(On@(1),Off@(0),M) = 1/100;
Channel(On@(0),Off@(1),M) = 1/150;
Channel(On@(2),Off@(1),M) = 1/150;
Channel(On@(1),Off@(2),M) = 1/200;
```

Fig. 1. Simple MASSPA model for an `On`/`Off` agent model generated using MASSPA-Modeller. In this model agents in the `On` state emit `!(...)` messages of type `M` and agents in the `Off` state can turn `On` if they receive `?(...)` a message of this type or alternatively simply turn `On` at rate `1.5`. The perception function describes all message channels as well a scaling rate for each channel that will modulate the rate at which messages can be sent on a particular channel.

3 The MASSPA–Modeller

The example in Fig. 1 shows a fairly simple MASSPA model. However, as the number of agents states, locations and message types increases, the definition of the perception function can become large. Moreover, it becomes difficult to visualise the directions of channels if a model has hundreds or thousands of

channels. One of the main features of MASSPA–Modeller (`http://www.doc.ic.ac.uk/~mcg05`) is its ability to mitigate this by allowing users to define high-level channels such as: *Any state in location (0) can send any kind of message to any state in location (1).* Later this high-level channel description is used to auto-generate actual MASSPA channels.

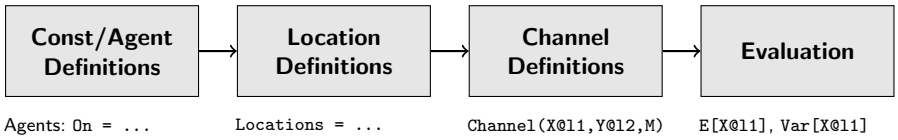

Fig. 2. MASSPA-Modeller work-flow

Having illustrated why a visual editor is necessary for the definition of complex MASSPA models, we now give a brief description of the MASSPA–Modeller work-flow, which is shown in Fig. 2. As a running example we will describe how the MASSPA model shown in Fig. 1 can be created in MASSPA–Modeller. The first step is to define constants, variables, functions and sequential agent definitions in the *Agents & Variables* tab. In our simple model we only define the latter, i.e. `Agent OnOff {On = !(2.0,M,1.0).Off; Off = ?(M,1.0).On + (1.5).On;}`, but in more elaborate models each transition rate could be represented as a expression of constants, variables and functions. Once this definition has been entered, the user can compile the agent definition. Errors and warnings will be displayed in the console below the editor tab.

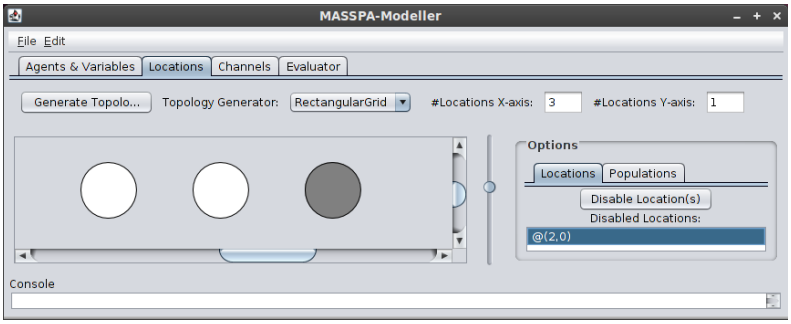

Fig. 3. Location definition in MASSPA–Modeller with one disabled location.

Having defined our agent(s), the *Locations* tab (see Fig. 3) can be used to generate different topologies such as rectangular or radial location grids. In our case we have a simple line consisting of 3 locations. Having generated a topology we can disable any unwanted location and define initial populations for every agent

state in each location. In our example we use this feature to create the following three initial populations `On@(0) = 100; Off@(1) = 200; Off@(2) = 150;`.

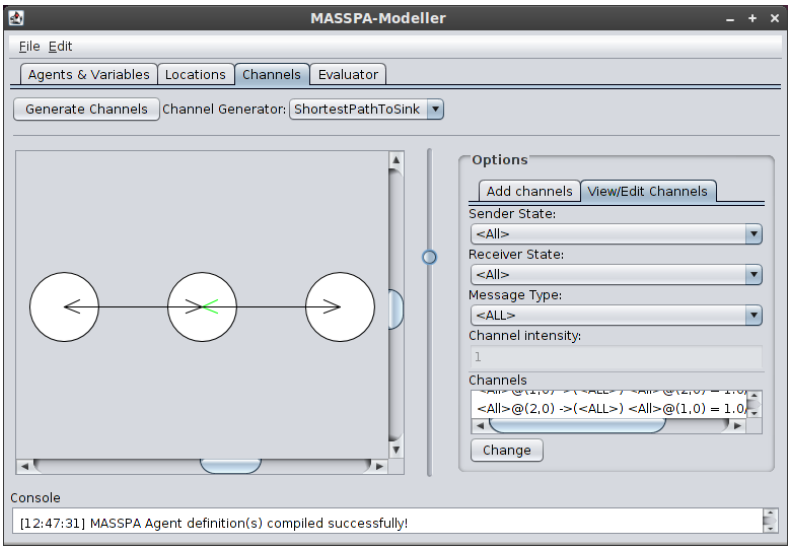

Fig. 4. Channel definition in MASSPA–Modeller.

Once users have create the topology and defined the initial populations, they can define the communication channels between locations or, if more fine grained communication control is needed, between specific sender and receiver populations (see Fig. 4). Channels can be generated using a predefined channel generator or by adding channels manually. Channel generators are useful for models with many locations, for instance when we want to create a source to sink style communication pattern with constraints on the maximum length of a single hop. In our example it is easiest to create the four channels manually in the editor.

The final tab allows users to generate the MASSPA model and to specify the evaluation method. Having generated the model, the evaluation method needs to be defined in GPA syntax [14], e.g. `ODEs(stopTime=60.0,stepSize=0.1, density=10,closure=MASSPA_infty){E[On@(2)],Var[On@(2)];}` uses fluid analysis to determine the mean and variance of `On` states in location `(2)` from time `0` to `60`. Pressing the *Evaluate* button will compile the model and use the GPA–analyser engine to perform the specified analysis and generate Fig. 5.

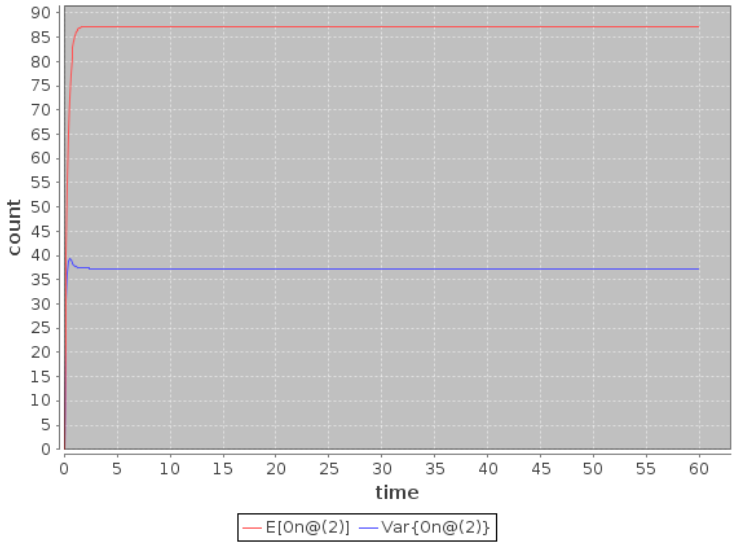

Fig. 5. Mean and variance for population `On@(2)`.

4 Conclusions

We have presented a new spatial modelling tool for MASSPA, which allows users to define all aspects of Markovian Agent models. Users can specify local agent behaviour using process algebra, while any spatial aspects of the model can be modelled using visual editors, which is more convenient than defining communication channels by hand. This combination between letting modellers define agents using process algebra and spatial aspects using an editor should be especially appealing to modellers who are familiar with non-spatial process algebras such as PEPA or GPEPA. In the future we might add this hybrid modelling approach to DrawNet, as this tool provides a much richer user interface for composing spatial models than MASSPA–Modeller. A particularly interesting challenge would be to create a WYSIWYG editor in DrawNet that allows users to define local agent behaviour as a labelled transition diagram or alternatively using MASSPA.

References

1. J. Hillston, "A Compositional Approach to Performance Modelling," *Cambridge University Press*, p. 158, 1996.
2. J. Hillston, "Fluid flow approximation of PEPA models," *Second International Conference on the Quantitative Evaluation of Systems QEST05*, pp. 33–42, 2005.

3. R. A. Hayden and J. T. Bradley, "A fluid analysis framework for a Markovian process algebra," *Theoretical Computer Science*, vol. 411, no. 22-24, pp. 2260–2297, 2010.
4. D. T. Gillespie, "Exact stochastic simulation of coupled chemical reactions," *Journal of Physical Chemistry*, vol. 81, no. 25, pp. 2340–2361, 1977.
5. L. Bortolussi and A. Policriti, "Modeling Biological Systems in Stochastic Concurrent Constraint Programming," *Constraints*, vol. 13, no. 1-2, pp. 66–90, 2008.
6. A. Stefanek, *Continuous and spatial extension of stochastic pi-calculus*. Master thesis, Department of Computing, Imperial College London, 2009.
7. R. A. Hayden, A. Stefanek, and J. T. Bradley, "Fluid computation of passage time distributions in large Markov models," *submitted to Theoretical Computer Science*, 2010.
8. A. Stefanek, R. A. Hayden, and J. T. Bradley, "Fluid analysis of energy consumption using rewards in massively parallel Markov models," in *2nd ACMSPEC International Conference on Performance Engineering ICPE*, pp. 121–132, 2011.
9. V. Galpin, "Towards a spatial stochastic process algebra," in *Proceedings of the 7th Workshop on Process Algebra and Stochastically Timed Activities (PASTA)*, (Edinburgh), 2008.
10. M. Gribaudo, D. Cerotti, and A. Bobbio, "Analysis of On-off policies in Sensor Networks Using Interacting Markovian Agents," *6th IEEE International Conference on Pervasive Computing and Communications PerCom (2008)*, pp. 300–305, 2008.
11. M. C. Guenther and J. T. Bradley, "Higher moment analysis of a spatial stochastic process algebra.," in *8th European Performance Engineering Workshop - EPEW 2011*, 2011.
12. D. Cerotti, M. Gribaudo, A. Bobbio, C. Calafate, and P. Manzoni, "A Markovian Agent Model for Fire Propagation in Outdoor Environments," in *7th European Performance Engineering Workshop EPEW* (A. Aldini, M. Bernardo, L. Bononi, and V. Cortellessa, eds.), vol. 6342 of *Lecture Notes in Computer Science*, pp. 131–146, Springer Berlin Heidelberg, 2010.
13. D. Cerotti, M. Gribaudo, and A. Bobbio, "Presenting Dynamic Markovian Agents with a road tunnel application," in *IEEE International Symposium on Modeling Analysis Simulation of Computer and Telecommunication Systems MASCOTS*, pp. 1–4, IEEE, 2009.
14. A. Stefanek, R. Hayden, and J. Bradley, "A new tool for the performance analysis of massively parallel computer systems," *Eighth Workshop on Quantitative Aspects of Programming Languages QAPL 2010 March 2728 2010 Paphos Cyprus*, 2010.
15. A. Baravalle, G. Franceschinisy, M. Gribaudo, V. Lanfranchi, M. Iaconoth, N. Mazzoccath, and V. Vittoriniu, "DrawNET Xe: GUI and Formalism Definition Language," *Informatica*, 2003.
16. F. Klügl, R. Herrler, and M. Fehler, "SeSAm: Implementation of Agent-Based Simulation Using Visual Programming," *Components*, no. May, pp. 1439–1440, 2006.
17. D. Cerotti, E. Barbierato, and M. Gribaudo, "A tool suite for modelling spatial interdependencies of distributed systems with Markovian Agents," tech. rep., 2011.

Argumentation and Temporal Persistence

E. Hadjisoteriou* and A. Kakas

University of Cyprus, Dept. of Computer Science,
Kallipoleos 75, 1678 Nicosia, Cyprus
{csp7he2,antonis}@cs.ucy.ac.cy
http://www.cs.ucy.ac.cy/

Abstract. We study how the problem of temporal projection can be formalized in terms of argumentation. In particular, we extend earlier work of translating the language $\mathcal{E}$ for Reasoning about Actions and Change into a Logic Programming argumentation framework, by introducing new types of arguments for (i) backward persistence and (ii) persistence from observations. This forms a conservative extension of the language $\mathcal{E}$ that gives semantic meaning to domains that cannot be interpreted in the language $\mathcal{E}$.

Keywords: Argumentation, narrative information, observations, backwards and forwards persistence

1 Introduction and Motivation

Given some narrative information we can use argumentation to capture temporal projection from this and general knowledge about the causal laws of our problem domain. As shown in [4], where the language $\mathcal{E}$ [3] for reasoning about actions and change was formalized in terms of argumentation, default persistence over time is captured by assigning higher priority to arguments that are based on later events over the arguments based on earlier events.

In this paper we extend this argumentation based formulation of language $\mathcal{E}$ by introducing also arguments based on property observations. Thus, we approach the qualification problem[6]. We review how temporal persistence is captured and introduce new arguments for backward persistence. This will allow us to recover and also extend language $\mathcal{E}$, giving a semantic meaning to domains that cannot be interpreted in the language $\mathcal{E}$. With this form of backward persistence the extended interpretation of the language $\mathcal{E}$ comes closer to the original Event calculus [5] which also include notions for backward temporal conclusions.

As an example of how language $\mathcal{E}$ is extended consider a parking domain, with action constant $ParkingCar$ and property fluent $CarInParkingSpace$ and the narrative that we park the car at time 4 and that later at time 8 we observed that the car is not where it was parked:
$ParkingCar$ initiates $CarInParkingSpace$ (Δ_1)

* An earlier version of this paper appeared in the proceedings of the 7th PLS.

Andrew V. Jones (Ed.): Proceedings of the 1st Imperial College Student Workshop (ICCSW '11), pp. 31–38. September 29th–30th 2011, London UK.

$ParkingCar$ happens-at 4 (Δ_2)
$\neg CarInParkingSpace$ holds at time 8 (Δ_3)

For domains like this, where a fluent (e.g. $CarInParkingSpace$) changes its truth value without any known causal explanation, language $\mathcal{E}$ does not give a model. On the other hand, our extended argumentation framework of the language $\mathcal{E}$ that includes arguments for observations and for backwards persistence as well allows arguments for both truth values of the fluent within this time interval. Forwards persistence from the action $ParkingCar$ (Δ_2) that indicates $CarInParkingSpace$ for every time point $t > 4$ (Δ_1) come in conflict with backwards persistence from the observation argument $\neg CarInParkingSpace$ (Δ_3). Allowing same priority to conflicting forward persistence over backwards persistence will give the natural interpretation of unknown value for the fluent $CarInParkingSpace$ for every $t \in (4, 8)$.

By introducing backwards persistence in our argumentation framework and assigning suitable priorities we can fully recover and also extend language $\mathcal{E}$. In our extended version we get models to domains that language $\mathcal{E}$ can not interpret. Furthermore, language $\mathcal{E}$ [4] handles domains without observations. We allow observations as part of our argumentation framework and assign priorities against all the other already existing arguments. As models must comply to all observations we treat observations as indisputable arguments. The reason we can do this is because of backwards persistence arguments and the priority assigned over forward persistence arguments.

The rest of the paper is organized as follows. Section 2 gives a brief review of the language $\mathcal{E}$. In section 3 we give the extended argumentation framework of $\mathcal{E}$. Section 4 presents our formal results and section 5 contains our conclusions.

2 A Brief Review of Language $\mathcal{E}$

Language $\mathcal{E}$ [3] is an action language that uses three kinds of propositions: **c-propositions**, of the form "A initiates F when C" or "A terminates F when C", **h-propositions** of the form "A happens-at T" and **t-propositions** of the form "L holds-at T", where A is an action constant, F is a fluent constant, T is a time point, L is a fluent literal and C is a set of fluent literals. Computational complexity is not the main concern of this paper. Lets note that the number of such models is exponentially high.

Models of the language $\mathcal{E}$ assign a truth value, $\{true\, or\, false\}$ at every fluent and every time point in the domain such that within any time interval the truth value assigned by a model to any fluent remains the same or **persists**, changing from false to true (resp. from true to false) at an initiation (resp. termination) time point. A time point T is an initiation (resp. termination) point when the problem domain description contains a combination of a c-proposition "A initiates (resp. terminates) F when C" and an h-proposition "A happens-at T", such that the model satisfies C at T. Furthermore, a model must confirm all the t-propositions given in the problem domain description resulting from fluent observations of the state of the world at various time points. Entailment and

consistency of formulae of the form "L holds-at T", where L is a fluent literal are then defined in the usual way. For formal definitions and results the reader is referred to [3].

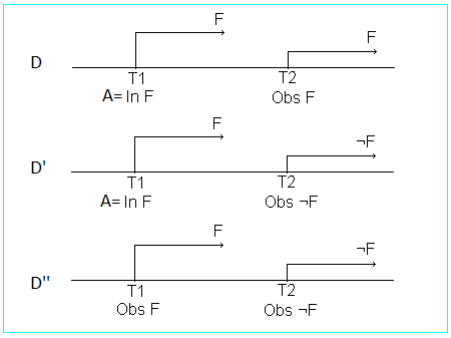

Fig. 1. Example Domains

As examples let us consider the domain descriptions $D, D^{'}$ and $D^{''}$ illustrated in (Figure 1), where A is an action and $T_1 < T_2$ are two time points. In domain D we have an initiation point at time T_1 and at time T_2 we observe F. Models of language $\mathcal{E}$, for domain D, require F to be true for all $T > T_1$ whereas, for $T^{'} \leq T_1$ a model can assign F to be either true or false at all such time points. In the domain $D^{'}$, where we have an initiation point at time T_1 and observation $\neg F$ at time T_2, and the domain $D^{''}$, where we have an observation F at time T_1 and an observation $\neg F$ at time T_2, the language $\mathcal{E}$ is inconsistent and has no models. The persistence of the F holding onwards from T_1 cannot be reconciled with the observation of $\neg F$ at T_2. Lets note that domain $D^{'}$ is similar to the parking domain example.

3 Argumentation Formulation

Language $\mathcal{E}$ has been reformulated in terms of argumentation [4]. In this the information from t-propositions (observations) is imposed as a-posteriori constraints on the argumentation formulation. We will extend this reformulation so that t-propositions are taken into account directly within the argumentation. To do so we will generalize the original formulation by allowing backward temporal persistence arguments as well as forward ones.

Following the earlier approach in [4], we define an argumentation logic program with priorities corresponding to a given domain description as follows.

Definition 1. *[argumentation program of D] The argumentation program corresponding to a domain D is $\Delta \equiv (B(D), A, <)$ where:*

- *The background knowledge, $B(D)$, contains the rule definitions of $Initiation(F,t)$ and $Termination(F,t)$ from c-propositions in D, facts of the form $Observation(L,T)$ for every t-proposition "L holds at T" in D and actions of the form A for every h-propositions "A happens-at T" in D.*
- *A consists of the following argument rules: For all time points t_1, t_2 and t such that $t_1 < t < t_2$,*

  Persistence:

$HoldsAt(f,t_2) \leftarrow HoldsAt(f,t)$	$PFP[f,t_2;t]$
$HoldsAt(f,t_1) \leftarrow HoldsAt(f,t)$	$PBP[f,t_1;t]$
$\neg HoldsAt(f,t_2) \leftarrow \neg HoldsAt(f,t)$	$NFP[f,t_2;t]$
$\neg HoldsAt(f,t_1) \leftarrow \neg HoldsAt(f,t)$	$NBP[f,t_1;t]$

  Local Generation Arguments:

$HoldsAt(f,t+1) \leftarrow Initiation(f,t)$	$PG_F[f,t]$
$\neg HoldsAt(f,t) \leftarrow Initiation(f,t)$	$PG_B[f,t]$
$\neg HoldsAt(f,t+1) \leftarrow Termination(f,t)$	$NG_F[f,t]$
$HoldsAt(f,t) \leftarrow Termination(f,t)$	$NG_B[f,t]$

  Local Observation Arguments:

$HoldsAt(f,t) \leftarrow Observation(f,t)$	$PO[f,t]$
$\neg HoldsAt(f,t) \leftarrow Observation(\neg f,t)$	$NO[f,t]$

  Assumption at 0:

$HoldsAt(f,0)$	$PA[f,0]$
$\neg HoldsAt(f,0)$	$NA[f,0]$

- *The priority (or strength of argument) relation, $<$, between these arguments is given below (t, t^*, t_1 and t_2 are time points):*

  If $t_1 < t_2$
  $PFP[f,t^*;t_1] < NFP[f,t^*;t_2]$, $NFP[f,t^*;t_1] < PFP[f,t^*;t_2]$,
  $PBP[f,t^*;t_2] < NBP[f,t^*;t_1]$, $NBP[f,t^*;t_2] < PBP[f,t^*;t_1]$,
  $NFP[f,t_2;t_1] < PO[f,t_2]$, $PFP[f,t_2;t_1] < NO[f,t_2]$,
  $NBP[f,t_1;t_2] < PO[f,t_1]$ *and* $PBP[f,t_1;t_2] < NO[f,t_1]$.
  At 0,
  $PA[f,0] < NO[f,0]$ *and* $NA[f,0] < PO[f,0]$.
  At t,
  $PG_B[f,t] < PO[f,t]$ *and* $NG_B[f,t] < NO[f,t]$.
  At $t+1$,
  $PG_F[f,t] < NO[f,t+1]$ *and* $NG_F[f,t] < PO[f,t+1]$.

Informally, the above priority makes forward persistence arguments that are based on later narrative information stronger and similarly for backward persistence arguments that are based on earlier narrative information. Also we assign higher priority to t-propositions over forward and backwards persistence. With this assignment observations become part of the argumentation rules and are treated as constrains that must be satisfied. However, note that there is no priority between conflicting forward and backward arguments. Such priorities can be additionally set when we wish to impose further properties on the temporal reasoning.

The semantics of these programs is given through the standard argumentation notion (see [1, 2]) of maximally admissible subsets of the given argumentation program, called **admissible extensions**. A subset of arguments is admissible if it does not derive $HoldsAt(f, t)$ and $\neg HoldsAt(f, t)$ for any fluent and time point and it can counter-attack any subset of arguments that attacks it. This attacking relation is defined such that a set of arguments would attack another if it derives a contrary conclusion and its argument rules in doing so are not weaker than the opposing argument rules. For the formal details please refer to [2, 4].

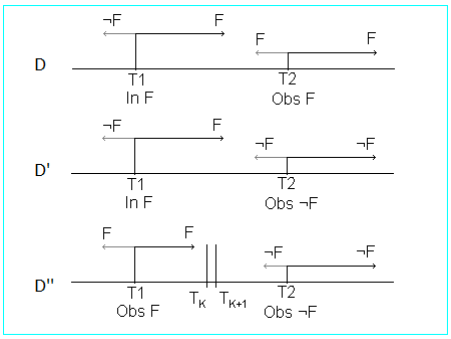

Fig. 2. Example Domains and Arguments

Comparing our earlier example domains D, D' and D'' (see Figure 1) within this new argumentation framework (Figure 2) we see that in the new domain D, for all $T > T_1$ the strongest (and hence admissible) argument is for F to hold. For $T' \leq T_1$ we can have admissible arguments for F or its negation $\neg F$ depending on the assumption we make at the initial time point. For the domain D'' the strongest argument for all time points $T \geq T_2$ is for $\neg F$. For times between T_1 and T_2 we have admissible arguments for either F or $\neg F$: at some time point T_k, $T_1 \leq T_k$, the fluent F changes from true to false at T_{k+1}. This indicates that the given narrative has some missing information within this time interval that would explain the change in F. Similar results hold for D' where also in this case there exists an admissible extension where $\neg F$ holds for all times T, such that $T_1 \leq T \leq T_2$. This captures the possibility that the generation of F at T_1 has failed.

4 Formal Results

In this section we present a set of formal results that show how our proposal for an argumentation semantics gives a meaning to any theory even when domains have t-propositions. By allowing forward persistence to be non comparable to

conflicting backwards persistence we can recover and also extend language $\mathcal{E}$ when this can not give a semantic meaning to a domain.

Property 1. Let D be a domain description and E an admissible extension of D. E is consistent (i.e. there does not exist a t-proposition "holds-at(f,t)" in D such that $D \models \neg$holds-at(f,t)).

Theorem 1. *Let D be a language $\mathcal{E}$ domain description and a countable number of h-propositions. Then:*

- *For every language $\mathcal{E}$ model, M, of D there exists an admissible extension, E, of the corresponding argumentation program $\Delta \equiv (B(D), A, <)$ such that E corresponds to M, i.e. $E \models$ holds-at(f,T) if and only if $M(f,T) = true$ and $E \models \neg$holds-at(f,T) if and only if $M(f,T) = false$.*
- *There exists a complete admissible extension D of the corresponding argumentation program $\Delta \equiv (B(D), A, <)$.*

For example, consider domain $D^{'}$ and $D^{''}$. With the new argumentation framework all maximally admissible extensions are consistent while in language $\mathcal{E}$ maximally admissible extensions are inconsistent.

Theorem 2 gives an interpretation of the extended semantic of the argumentation formulation in terms of the original language $\mathcal{E}$. We first need the following two lemmas:

Lemma 1. *Let D be a consistent domain and E a complete admissible extension of D. Let f be a fluent and $t_n < t_m$ two time points. If there does not exist a generation point for the fluent f in E at $t_1 \in [t_n, t_m)$ nor an observation point for the fluent f in E at $t_2 \in (t_n, t_m)$ and if*
$E \models$ holds-at(f,t_n) and $E \models$ holds-at(f,t_m) or
$E \models \neg$holds-at(f,t_n) and $E \models \neg$holds-at(f,t_m) then, there does not exist a time point $T \in [t_n, t_m)$ where the given fluent f changes its truth value in E, i.e. $E \models$ holds-at(f,T), for every $T \in [t_n, t_m]$ or $E \models \neg$holds-at(f,T), for every $T \in [t_n, t_m]$.

Informally, when no information is given in the narratives between two time periods that assign the same truth value for every fluent then a complete admissible extension gives a constant truth value for every fluent over this time period.

Lemma 2. *Let D be a consistent domain and E a complete admissible extension of D. Let f be a fluent and $t_n < t_m$ two time points. If there does not exists a generation point for the fluent f in E at $t_1 \in [t_n, t_m)$ nor an observation point for the fluent f in E at $t_2 \in (t_n, t_m)$ and if*
$E \models$ holds-at(f,t_n) and $E \models \neg$holds-at(f,t_m) or
$E \models \neg$holds-at(f,t_n) and $E \models$ holds-at(f,t_m) then, there exist at least one time points $T \in [t_n, t_m)$ where f can change its truth value in E, i.e.
$E \models$ holds-at(f,T) and $E \models \neg$holds-at$(f,T+1)$ or $E \models \neg$holds-at(f,T) and $E \models$ holds-at$(f,T+1)$. If the number of such time points is $k > 1$ then k is an odd number.

When two time periods assign opposite values for a fluent f and no information is given in the narratives then in a complete admissible extension there must exist k many (k is an odd number) time points between these two time points that change the truth value of f.

Theorem 2. *Let D be a domain description. For every maximally admissible extension E there exist a domain $D^{'}$ obtained from D by adding new events such that there exist a language $\mathcal{E}$ model, M, of $D^{'}$ that corresponds to E (i.e. $E \models$ holds-at(f,t) if and only if $M \models$ holds-at(f,t)).*

For example consider domain $D^{''}$ where from F at time T_1 we jump to $\neg F$ at time T_2. Let time point $T_k \in (T_1, T_2)$. By accepting that time T_k is a termination point for F we explain the semantic meaning given by argumentation to the domain.

To recover exactly the language $\mathcal{E}$ semantics we need to add extra priorities and specifically, to give preference to forward arguments over conflicting backwards arguments. The formal result for this is given in theorem 3.

Theorem 3. *In addition to the priorities given in definition 1 let also the following when $t_1 < t_2$:*
$PFP[f,t;t_1] > NBP[f,t;t_2]$ *and* $NFP[f,t;t_1] > PBP[f,t;t_2]$.

Then, every maximally admissible extension E, for any domain D corresponds to a model M of the language $\mathcal{E}$, of D.

5 Conclusions and Further Work

We have reexamined the argumentation reformulation of language $\mathcal{E}$ and introduced backwards persistence as well as forward persistence arguments. This enabled us to extend in a meaningful way domains that language $\mathcal{E}$ could not interpret. When language $\mathcal{E}$ is inconsistent within two time points, the argumentation interpretation corresponds to the unknown occurrences of events that could resolve this inconsistency.

As a future work we recommend a planning for more complicated domains with action A, fluents F and G such that A causes $\neg F$ and $\neg G$. In addition, even though there are many ways to deal with ramification problems (F_1 causes F_2 when L) we leave an open door that one can work on this issue further. Finally, all sets in this theory are countable. We are very interested to learn if there a way to extend this theory to uncountable sets and variables.

References

1. Phan Minh Dung, *On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games*, Artif. Intell. **77** (1995), no. 2, 321–357.
2. Antonis C. Kakas, Paolo Mancarella, and Phan Minh Dung, *The acceptability semantics for logic programs*, ICLP, 1994, pp. 504–519.

3. Antonis C. Kakas and Rob Miller, *A simple declarative language for describing narratives with actions*, J. Log. Program. **31** (1997), no. 1-3, 157–200.
4. Antonis C. Kakas, Rob Miller, and Francesca Toni, *An argumentation framework of reasoning about actions and change*, LPNMR, 1999, pp. 78–91.
5. R Kowalski and M Sergot, *A logic-based calculus of events*, New Gen. Comput. **4** (1986), no. 1, 67–95.
6. M. Thielscher, *The Qualification Problem: A Solution to the Problem of Anomalous Models*, AIJ **131** (2001), no. 1–2, 1–37.

Facial Action Recognition using sparse appearance descriptors and their pyramid representations

Bihan Jiang[1], Michel F. Valstar[1], and Maja Pantic[1,2]

[1] Department of Computing, Imperial College London, UK
{bi.jiang09, michel.valstar, m.pantic}@imperial.ac.uk
[2] EEMCS, University of Twente, Netherlands
PanticM@cs.utwente.nl

Abstract. Most existing work on automatic analysis of facial expressions has focused on a small set of prototypic emotional facial expressions such as fear, happiness, and surprise. The system proposed here enables detection of a much larger range of facial behaviour by detecting facial muscle actions (action units, AUs). It automatically detect all 9 upper face AUs using local apperance descriptors. Meanwhile, the merits of the family of local binary pattern descriptors are investigated. We compare Local Binary Patterns, Local Phase Quantisation, Pyramid Local Binary Pattern, as well as our proposed descriptors Block-based Pyramid Local Binary Pattern and Block-based Pyramid Local Phase Quantisation for AU detection. Results show that our proposed descriptor Block-based pyramid Local Binary Pattern outperforms all other tested methods for the problem of FACS Action Unit analysis and the systems that utilise pyramid representation outperform those that use basic appearance descriptors.

1 Introduction

One limitation of the majority of existing facial expression recognition methods is that they focus on a small set of prototypic emotional facial expressions, specifically fear, sadness, happiness, anger, disgust, and surprise. Yet, these six basic emotion categories form only a subset of the total range of possible facial displays and the categorisation of facial expressions can therefore be forced and unnatural. The Facial Action Coding System (FACS) is the best known and most commonly used system developed for human observers to describe facial activities. The coding system defines atomic facial muscle actions called Action Units (AUs). With FACS, every possible facial expression (emotional or otherwise) can be described as a combination of AUs. For instance, the expression of happiness contains AU6 and AU12, while the expression of sadness contains AU1, AU4 and AU15.

Deriving an effective facial representation from images is an essential step for successful facial expression recognition. Traditionally the feature extraction

Andrew V. Jones (Ed.): Proceedings of the 1st Imperial College Student Workshop (ICCSW '11), pp. 39–45. September 29th–30th 2011, London UK.

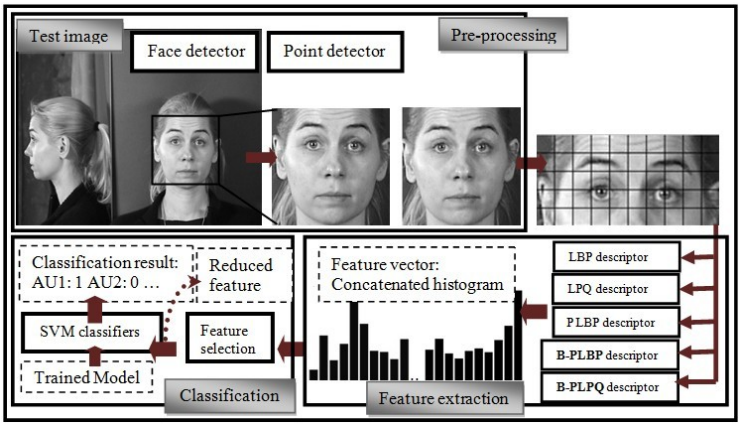

Fig. 1. The outline of our proposed system

approaches may be divided into two streams: geometric feature-based methods and appearance-based methods. Geometric feature based methods employ the geometrical properties of a face such as the positions of facial points relative to each other, the distances between pairs of points or the velocities of separate facial points. For a method using appearance features, the changes in image texture such as those created by wrinkles, bulges, and changes in feature shapes are captured.

Our key contributions are three-fold. First, we propose the novel appearance feature descriptors Block-based Pyramid Local Binary Pattern (B-PLBP) and Block-based Pyramid Local Phase Quantisation (P-BLPQ). Secondly, the proposed appearance descriptor B-PLBP and B-PLPQ are applied to the problem of FACS AU analysis. Finally, the applicability of different SVM kernels for histogram-based features has been studied. The experimental results show that our novel descriptor B-PLBP outperforms the three other methods for FACS AU analysis in terms of recognition accuracy.

The remainder of this paper is organised as follows. Section 2 briefly describes the methodologies used in this work. It introduced the basic principle of static appearance descriptors LBP, LPQ, PLBP and our proposed extensions B-PLBP and B-PLPQ, the training datasets used in our experiments, the classification technique used in this work and the different kernels tested. The evaluation procedures and test results are given in Section 3. Section 4 provides the conclusions of our research.

2 Methodologies

Fig. 1 shows an overview of the proposed system. In order to detect the upper face AUs, we use 9 SVM classifiers, one for each AU, which are trained on a

subset of the most informative spatiotemporal features selected by GentleBoost. To extract these appearance features, we first find the face in the input static image using an adapted version of the Viola and Jones face detector. Next the detected face images are registered to remove head rotations and scale variations by using the OpenCV implementation of an object detector to locate the eyes. Based on that, the face image is scaled to make the distance between the eye locations 100 pixels, and then cropped to be 200 by 200 pixels. After that, the registered image is divided into small blocks and the LBP, LPQ, PLBP, B-PLBP and B-PLPQ features are extracted. The histograms from all blocks are concatenated as a feature vector to represent the corresponding face image.

2.1 Local Appearance Descriptors

Method 1. Local Binary Patterns (LBP) were first introduced by Ojala et al. in [4], and proved to be a powerful means of texture description. By thresholding a 3×3 neighbourhood of each pixel with respect to the centre value, the operator labels each pixel. Considering the 8-bit result to be the binary representation of a decimal number, a 256-bin histogram of the LBP labels computed over a region is used as a texture descriptor. This has been successfully applied to face recognition by Ahonen et al.[1]. They proposed to divide face images into m local regions, from which LBP histograms can be extracted, and then concatenate them into a single,spatially enhanced feature histogram. The resulting histogram encodes both the local texture and global shape of face images. This version is what we adopted in our work. Readers are kindly asked to refer to [4, 1] for details.

Method 2. Local Phase Quantisation (LPQ) was originally proposed by Ojansivu and Heikkila as a texture descriptor that is robust to image blurring [5]. The descriptor uses local phase information extracted using the 2-D DFT or, more precisely, a short-term Fourier transform (STFT) computed over a rectangular M-by-M neighbourhood N_x at each pixel position $\mathbf{x}$ of the image $f(\mathbf{x})$ defined by

$$F(\mathbf{u}, \mathbf{x}) = \sum_{\mathbf{y} \in N_x} f(\mathbf{x}\text{-}\mathbf{y}) e^{-j2\pi \mathbf{u}^T \mathbf{y}} = \mathbf{w}_\mathbf{u}^T \mathbf{f}_\mathbf{x} \tag{1}$$

where $\mathbf{w_u}$ is the basis vector of the 2-D DFT at frequency $\mathbf{u}$, and $\mathbf{f_x}$ is the vector containing all M^2 samples from N_x.

The phase information in the Fourier coefficients is recorded by examining the signs of the real and imaginary parts of each component in F_x. The resulting eight bit binary coefficients $g_j(x)$ are represented as integers using binary coding. As a result, a histogram of these values from all positions is composed and used as a 256-dimensional feature vector in classification. Similar to LBP, we use a block version of LPQ which has shown promising performance in [3]. For more details, please refer to [5, 3].

Method 3. Qian et.al [6] extended the conventional LBP to the pyramid transform domain named **Pyramid Local Binary Pattern (PLBP)**. By cascading the LBP information of hierarchical spatial pyramids, PLBP takes texture resolution variations into account. They comprehensively compared PLBP with other LBP extensions for texture classification and claimed that PLBP is with satisfactory performances and with low computational cost. However, a histogram computed over the whole image represents only the global distribution of the patterns thus the local information has been ignored. On the other hand, some researchers are critical of Ahonen's approach, suggesting that the subregions are not necessarily well aligned with facial features and the resulting facial description depends on the chosen window size and the position of these subregions [2]. These problems were reflected in our results (see Fig.5).

Motivated by these ideas, we propose two novel descriptors **B-PLBP** and **B-PLPQ** which capture pixel-level, region-level and structure-level information for face representation. The face image is represented in an image pyramid by different spatial resolutions. Each pixel in the higher spatial pyramid levels is obtained by down sampling from its adjacent low-pass filtered high resolution image. Hence in the low resolution images, a pixel corresponds to a region in its high-resolution equivalently. For each pyramid level, the image are divided into regions. The region sizes remain constant. The dense appearance descriptor features extracted from each region, and each level of the pyramid, are concatenated into a single, spatially enhanced feature histogram. As shown in Fig.2, the blocks in each level encodes different spatial information. In our experiments, a three level pyramid model and a region size of 25×25 pixels is used.

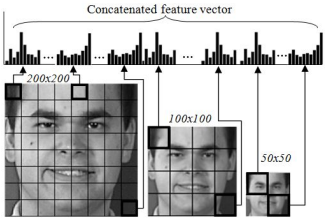

Fig. 2. The block-based pyramid representation

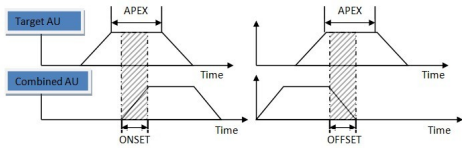

Fig. 3. The criterion of static data selection. The shaded areas are included in the dataset

2.2 Data Collection

In this work, the efficiency of the discussed descriptors are tested based on dataset collected from the MMI Facial Expression Database (MMI database [8]). The MMI database is a fully web-searchable collection of visual and audio-visual recordings of subjects displaying facial expressions which are FACS annotated. It includes 69 different subjects of both sexes (44 female), ranging in age from 19

to 62, having either a European, African, Asian, Caribbean or South American ethnic background. All fully FACS-coded recordings show facial expressions that are posed, and it is these data which will be used in this work.

In [3], the authors proposed a **heuristic approach** to select data for training. It is noted that when more than one AU is activated, facial actions can appear very different from when they occur in isolation. For example, AU1 and AU2 pull the brow up, whereas AU4 pulls the brows together and down using primarily the corrugators muscle at the bridge of the nose. The appearance of AU4 changes dramatically depending on whether it occurs alone or in combination with AU1 and AU2. In order to capture the appearance of each action unit as fully as possible and thus build a richer data space, the heuristic approach takes in every video the first apex frames of each target AU, and all the apex frames where any other upper face AUs are in onset or offset (see Fig. 3). The shaded parts are the frames selected. However, AU combinations are not treated differently by the classifiers. In other words, each AU is recognised independently of all the others.

2.3 Classification

A previous successful technique to facial expression classification is Support Vector Machine (SVM). In this work, we adopted SVM as classifiers for AU detection. Given a training set of labelled examples $\{(x_i, y_i), i = 1, ..., l\}$, where $x_i \in R^n$ and $y_i \in \{1, -1\}$, a new test example x is classified by the following function:

$$f(x) = sgn(\sum_{i=1}^{l} \alpha_i y_i K(\mathbf{x_i}, \mathbf{x}) + b) \tag{2}$$

where sgn function returns the sign of y, i.e. either 1 or -1, α_i are Lagrange multipliers of a dual optimisation problem that describe the separating hyperplane, $K()$ is a kernel function, and b is the threshold parameter of the hyperplane. Performing an implicit mapping of data into a higher dimensional feature space, which is defined by the kernel function, the training process is achieved by finding a linear separating hyper-plane with the maximal margin (M) to separate data in this higher dimensional space. The most popular kernels are linear, polynomial and Radial Basis Function (RBF). Recently, Maji et.al [7] proposed a histogram intersection kernel SVMs (IKSVMs). They also introduced a more efficient way to compute it. It is shown that IKSVM gives comparable accuracy while being $50\times$ faster and require $200\times$ less memory than the standard SVM implementation in their experiments. In this work, we evaluate the efficiency of these four kernels in our application.

3 Evaluation

3.1 Comparison Setup

We evaluated the four appearance descriptors on 442 videos taken from the MMI database. In order to compare different approaches, the same evaluation process

is performed. As this is a user independent system for FACS Action Unit detection, the evaluation is done in a subject independent manner. Generalisation to new subjects is tested using 10-fold cross validation.

The performance measure used in this work is the area under the ROC curve. By using the signed distance of each sample to the SVM hyper-plane and varying a decision threshold, we plot the hit rate (true positives) against the false alarm rate (false positives). The area under this curve is equivalent to percent correct in a 2-alternative forced task (2AFC), which can be computed more efficiently.

3.2 Results

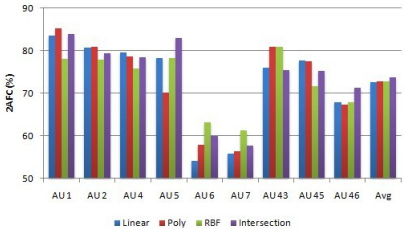

Fig. 4. Average 2AFC (%) based on different kernels for SVM

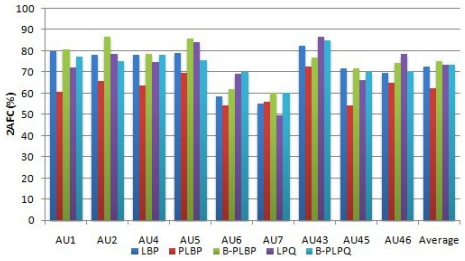

Fig. 5. The 2AFC (%) using LBP, PLBP, B-PLBP, LPQ and B-PLPQ based on MMI

Experiment 1. *Kernel functions:* Fig.4 shows the average 2AFC scores performed with B-PLBP based on different SVM kernel as we discussed in section 2.3. The LBP features and the proved best training data selection method, the heuristic approach, has been employed. For all the kernels, the parameters are optimised before training (refer to 3-A). In general, the best results are reached with the histogram intersection kernel. This is expected as all the features used in this work are histogram-based. For AU6 and AU7, which our detector poorly performed, RBF kernel gives the best result. This probably result from the fact that features that which capture subtle appearance changes, are non-linear separable.

Experiment 2. *Appearance descriptors:* Figure 5 presents the 10-fold cross-validation results using LBP, LPQ, PLBP, B-PLBP and B-PLPQ for 9 upper face AUs. Note that LBP and LPQ used here are block-based. To report the best performance of all systems, the heuristic approach and the histogram intersection kernel SVM are adopted in these experiments. In general, the block-based pyramid extension outperform their original version (LBP and LPQ). The importance is more clear for P-PLBP. We can see that, overall speaking, B-PLBP produces best results among these four descriptors and the PLBP performs worst.

The average 2AFC scores from B-PLBP is 12.8% higher than that for PLBP. The importance of local shape information for AU detection is again shown by our results. Compared to B-PLBP, the improvement of B-PLPQ is less obvious. This can probably be explained by the blur-invariant characteristic of the LPQ descriptor, which effectively negates the effect of the image pyramid.

4 Conclusions

We successfully implemented a robust and real-time AU detection system. We compared the appearance descriptors LBP, LPQ and their block-based pyramid extension B-PLBP and B-PLPQ. Results show that the systems based on LPQ generally achieve higher accuracy rate than LBP system, and that the systems that utilise pyramid representation outperform those that use basic appearance descriptors. Although the family of block-based pyramid descriptors are more computationally expensive than the basic ones, they attain a higher recognition performance. All in all, the experimental results clearly show that our proposed descriptor B-PLBP outperforms all other tested methods for the problem of FACS Action Unit analysis. Note that although we only applied the method to upper face AUs, the method can be readily used for all other AUs.

References

1. T. Ahonen, A. Hadid, and M. Pietikainen. Face recognition with local binary patterns. In *European Conference on Computer Vision*, pages 469–481, 2004.
2. D. Huang, C. Shan, and M. Ardabilian. Local binary pattern and its application to facial image analysis: A survey. *IEEE Trans. Systems, Man and Cybernetics, Part C*, 41:1–17, 2011.
3. B. Jiang, M. Valstar, and M. Pantic. Action unit detection using sparse appearance descriptors in space-time video volumes. In *Proceedings of IEEE International Conference on Automatic Face and Gesture Recognition (FG'11)*, Santa Barbara, CA, USA, March 2011.
4. T. Ojala, M. Pietikainen, and D. Harwood. A comparative study of texture measures with classification based on featured distribution. *Pattern Recognition*, 29(1):51–59, 1996.
5. V. Ojansivu and J. Heikkila. Blur insensitive texture classification using local phase quantization. In *In Proc. Int. Conf. on Image and Signal Processing*, volume 5099, pages 236–243, 2008.
6. X. Qian, X. Hua, P. Chen, and L.Ke. Plbp: An effective local binary patterns texture descriptor with pyramid representation. *Semi-Supervised Learning for Visual Content Analysis and Understanding*, 44(10):2502–2515, 2011.
7. J. M. S. Maji, A.C. Berg. Classication using intersection kernel support vector machines is efcient. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 2008.
8. M. Valstar and M. Pantic. Induced disgust, happiness and surprise: an addition to the mmi facial expression database. In *Proc. Int'l Conf. Language Resources and Evaluation, W'shop on EMOTION*, pages 65–70, 2010.

Note on a Dichotomy for the Classes $W[P](\mathcal{C})$ Defined via Symmetric Connectives

Björn Lellmann

Imperial College London, Department of Computing
bl610@doc.ic.ac.uk

Abstract. Adopting a generalised notion of connectives as ptime-computable symmetric boolean functions, for finite sets $\mathcal{C}$ of such connectives the classes $W[P](\mathcal{C})$ are defined via the parameterised weighted satisfiability problem for circuits with $\mathcal{C}$-gates. This note will prove the following dichotomy result: for all finite sets $\mathcal{C}$ of connectives $W[P](\mathcal{C}) = \mathrm{FPT}$ or $W[P](\mathcal{C}) = W[P]$.

Keywords: Parameterised Complexity; Symmetric Gates; Majority

1 Introduction

One of the most important parameterised complexity classes is the class $W[P]$. It can be considered a natural parameterised analogue of the class NP and was originally defined in [3] via the parameterised weighted satisfiability problem for boolean circuits. What happens, if we change the computational power of the underlying circuits? One possibility is to allow other than the boolean gates, such as majority gates, which output 1 (true) if the majority of the inputs is 1 (true), and 0 (false) otherwise, or parity gates, which output 1 if the number of inputs set to 1 is odd, and 0 otherwise. Notice that these gates are symmetric in the sense that their output is invariant under permutations of the inputs. In the parameterised setting of [3] this amounts to the question of how difficult it is to solve the parameterised weighted satisfiability problem for circuits with symmetric gates of unbounded fan-in. In the following we will call such a family of symmetric gates which in addition is *ptime*-computable a connective. For a finite set $\mathcal{C}$ of such connectives the class of problems, which are *fpt*-reducible to the parameterised weighted satisfiability problem for $\mathcal{C}$-circuits, will be called $W[P](\mathcal{C})$. Of course now an interesting question is the relationship between these classes and $W[P]$. This note proves that a full dichotomy holds, i.e. that for every finite set $\mathcal{C}$ of connectives $W[P](\mathcal{C}) = \mathrm{FPT}$ or $W[P](\mathcal{C}) = W[P]$. The proof will also give characterisations of the connectives for both alternatives.

Yet, the study of $W[P](\mathcal{C})$ is not just interesting in its own right. These classes were first defined in [5], where the authors also defined the $W(\mathcal{C})$-hierarchies via the notion of weft and showed that for sufficiently strong bounded $\mathcal{C}$ the levels of the $W(\mathcal{C})$-hierarchy and the W-hierarchy coincide. Unfortunately for unbounded $\mathcal{C}$ there are only a few exemplary (although very interesting) results. When trying

Andrew V. Jones (Ed.): Proceedings of the 1st Imperial College Student Workshop (ICCSW '11), pp. 46–51. September 29th–30th 2011, London UK.

to show such results for unbounded connectives it is vital to first have a look at the class $W[P](\mathcal{C})$, which contains the $W(\mathcal{C})$-hierarchy. Using the dichotomy, the connectives with $W[P](\mathcal{C}) = \text{FPT}$ can be omitted from such an analysis straight away.

2 Preliminaries

In the following $\mathbb{N}$ denotes the set of natural numbers. As usual for a finite alphabet Σ the set Σ^* consists of the finite strings over Σ. If $\bar{x}$ is a string in Σ^* then $|\bar{x}|$ denotes the length of $\bar{x}$, and we write $\bar{x}^n$ for the concatenation of n copies of $\bar{x}$. The *weight* of a binary string $\bar{x} \in \{0,1\}^*$ is the number of 1's in $\bar{x}$.

A *parameterised problem* P is a subset of $\Sigma^* \times \mathbb{N}$ for a finite alphabet Σ. Here $\mathbb{N}$ is encoded in unary. Given an instance $(\bar{x}, k) \in \Sigma^* \times \mathbb{N}$ of the problem, $\bar{x}$ is the *input*, and k is the *parameter*. The problem P is *fixed-parameter-tractable*, if there are a computable function $h : \mathbb{N} \to \mathbb{N}$ and a polynomial p, such that for every instance $(\bar{x}, k)$ membership in P can be decided in time $h(k) \cdot p(|\bar{x}|)$. The class of fixed-parameter-tractable problems is denoted by FPT. Given two parameterised problems $P \subseteq \Sigma^* \times \mathbb{N}$ and $Q \subseteq (\Sigma')^* \times \mathbb{N}$ a mapping $f : \Sigma^* \times \mathbb{N} \to (\Sigma')^* \times \mathbb{N}$ is an *fpt-reduction* of P to Q, if the following holds:

- for all $(\bar{x}, k) \in \Sigma^* \times \mathbb{N}$: $(\bar{x}, k) \in P$ iff $f(\bar{x}, k) \in Q$
- there are a computable function $h : \mathbb{N} \to \mathbb{N}$ and a polynomial p such that f is computable in time $h(k) \cdot p(|\bar{x}|)$
- there is a computable function $g : \mathbb{N} \to \mathbb{N}$ such that for all $(\bar{x}, k)$ and $(\bar{x}', k')$ with $f(\bar{x}, k) = (\bar{x}', k')$ we have $k' \leq g(k)$.

If there is such an *fpt*-reduction, then P is *fpt-reducible* to Q and we write $P \leq^{fpt} Q$. For detailed information on parameterised complexity see [4, 7, 8].

3 The general framework

Arguably the most important property of boolean connectives is that they are symmetric, that is that the output depends only on the number of the inputs, which are set to '1' (true) respective '0' (false). Thus we generalise the notion of a boolean connective in the following way (following [5]).

Definition 1. *A* connective C *is a function* $C : \mathbb{N} \times \mathbb{N} \longrightarrow \{0,1\}$ *computable in time polynomial in the sum of its arguments.*

For $m, n \in \mathbb{N}$ we interpret the value $C(m, n)$ as the truth value of the connective C when its input consists of m ones and n zeros. For $n \in \mathbb{N}$ and $\bar{x} \in \{0,1\}^n$ with weight k we write $C[\bar{x}]$ for $C(k, n-k)$. For $\ell \in \mathbb{N}$ it will be convenient to write $C \restriction_\ell$ for the connective C restricted to tuples (m, n) with $m + n = \ell$.

Example 1. The standard connectives $\bigwedge, \bigvee, Maj, \oplus$ fit in this framework via:

$$\begin{aligned} \textstyle\bigwedge(m,n) = 1 &\iff n = 0 \\ \textstyle\bigvee(m,n) = 1 &\iff m \neq 0 \\ Maj(m,n) = 1 &\iff m > n \\ \oplus(m,n) = 1 &\iff m \text{ is odd} \end{aligned}$$

Substituting boolean gates in a boolean circuit by gates labelled with a connective in $\mathcal{C}$, we get the obvious notion of a $\mathcal{C}$-circuit as follows: Let $\mathcal{C}$ be a finite set of connectives. A *$\mathcal{C}$-circuit* is a finite directed acyclic graph with multiple edges and with labelled vertices, which we will call *gates*. There are two kinds of gates with in-degree 0: the *input-gates*, which are labelled with distinct consecutive natural numbers starting with 1, and the *constant-gates*, which are labelled with one of the constants 0 and 1, or a connective $C \in \mathcal{C}$. Gates with in-degree > 0 are labelled with a connective $C \in \mathcal{C}$ each. The gates with out-degree 0 are called the *output-gates* of the circuit. In the following all circuits are assumed to have exactly one output-gate. A $\mathcal{C}$-circuit D with ℓ input gates and one output gate computes a function $f_D : \{0,1\}^\ell \to \{0,1\}$ in the obvious way. As usual we say that the $\mathcal{C}$-circuit D is *satisfiable*, if there is a $\bar{x} \in \{0,1\}^\ell$, such that $f_D(\bar{x}) = 1$. The circuit is *k-satisfiable*, if there is a $\bar{x} \in \{0,1\}^\ell$ with weight k, such that $f_D(\bar{x}) = 1$.

Notice that boolean circuits are a special case of $\mathcal{C}$-circuits for $\mathcal{C} = \{\neg, \bigvee, \bigwedge\}$, where $\neg$ is the connective defined by $\neg(m,n) = 1 \Leftrightarrow m = 0$. Analogously to the parameterised weighted satisfiability problem for boolean circuits (see [4, 7]), the parameterised weighted satisfiability problem for $\mathcal{C}$-circuits is defined as:

p-WSat(Circ($\mathcal{C}$))
Input: a $\mathcal{C}$-circuit and $k \in \mathbb{N}$
Question: Is the circuit k-satisfiable?

Definition 2. *Let $\mathcal{C}$ be a set of connectives. A parameterised problem is in the class $W[P](\mathcal{C})$ iff it is fpt-reducible to the parameterised problem* p-WSat(Circ($\mathcal{C}$)).

If the set $\mathcal{C}$ contains only one connective C we write $W[P](C)$ instead of $W[P](\{C\})$. Since every ptime-computable boolean function with domain $\{0,1\}^*$ is computed by a family of boolean circuits computable in polynomial time in the length of the input [9, Theorem 11.5], we immediately get

Proposition 1. *$W[P](\mathcal{C}) \subseteq W[P]$ for all finite sets $\mathcal{C}$ of connectives.*

Proof. Replacing the $\mathcal{C}$-gates with the corresponding subcircuits yields an fpt-reduction to $W[P]$. □

4 The Dichotomy Theorem

Lemma 1. *Let C be one of the connectives $\bigvee, \bigwedge, \oplus$. Then $W[P](C) \subseteq \mathrm{FPT}$.*

Proof. Let C be one of the connectives $\oplus, \bigvee, \bigwedge$. Then for all $\bar{x}, \bar{y} \in \{0,1\}^*$ we have $C[C[\bar{x}]\,\bar{y}] = C[\bar{x}\,\bar{y}]$, and thus every C-circuit is equivalent to a C-circuit consisting of a single C-gate, which receives (possibly multiple) edges from the old input-gates and possibly from constant-gates. If $C \in \{\bigvee, \bigwedge\}$ we further simplify the circuit by substituting every multiple edge by a single edge. If $C = \oplus$, we substitute every multiple edge of odd cardinality by a single edge and delete every multiple edge of even cardinality. The resulting circuit is equivalent to the original circuit. As this clearly can be done in *fpt*-time, and as the k-satisfiability of the simplified circuit easily can be checked in polynomial time, we have p-$\textsc{WSat}(\textsc{Circ}(C)) \in \mathrm{FPT}$ and thus $W[P](C) \subseteq \mathrm{FPT}$. □

Theorem 1 (Dichotomy Theorem). *Let $\mathcal{C}$ be a finite set of connectives. Then $W[P](\mathcal{C}) = W[P]$ or $W[P](\mathcal{C}) = \mathrm{FPT}$.*

Proof. By Proposition 1, closure of $W[P](\mathcal{C})$ under *fpt*-reductions and availability of constant gates we have $\mathrm{FPT} \subseteq W[P](\mathcal{C}) \subseteq W[P]$. Call a connective C

- $\vee$-*closed* if there are $n, m \in \mathbb{N}$ with $C(m, n+2) = 0$ and $C(m+1, n+1) = C(m+2, n) = 1$
- $\wedge$-*closed* if there are $n, m \in \mathbb{N}$ with $C(m, n+2) = C(m+1, n+1) = 0$ and $C(m+2, n) = 1$
- *monotone*, if for all $m, n \in \mathbb{N}$ we have $C(m, n+1) \leq C(m+1, n)$.

If the connectives C_1, C_2, C_3 are $\vee$- and $\wedge$-closed and not monotone respectively, they simulate the boolean connectives via

$$x \vee y = C_1[1^{m_1}\, x\, y\, 0^{n_1}] \tag{1}$$
$$x \wedge y = C_2[1^{m_2}\, x\, y\, 0^{n_2}] \tag{2}$$
$$\neg x = C_3[\,1^{m_3}\, x\, 0^{n_3}] \tag{3}$$

A set $\mathcal{C}$ of connectives is monotone, if every connective in $\mathcal{C}$ is monotone, and $\vee$-closed (respective $\wedge$-closed), if there is a $\vee$-closed ($\wedge$-closed) connective in $\mathcal{C}$. For monotone $\mathcal{C}$ we have four cases.

Case 1: $\mathcal{C}$ is $\vee$-closed and $\wedge$-closed. As then $\mathcal{C}$ is able to simulate small disjunctions and conjunctions according to equivalences (1) and (2) above, the parameterised weighted satisfiability problem for boolean circuits without negation gates p-$\textsc{WSat}(\textsc{Circ}^+)$ is *fpt*-reducible to p-$\textsc{WSat}(\textsc{Circ}(\mathcal{C}))$. Since p-$\textsc{WSat}(\textsc{Circ}^+)$ is $W[P]$-complete [1] we have $W[P] \subseteq W[P](\mathcal{C})$.

Case 2: $\mathcal{C}$ is $\vee$-closed, but not $\wedge$-closed. For all $C \in \mathcal{C}$ and for all $\ell \in \mathbb{N}$ we then have $C \restriction_\ell = \mathit{const}$ or $C \restriction_\ell = \bigvee \restriction_\ell$, because otherwise the connective C would be $\wedge$-closed as well. But then we have p-$\textsc{WSat}(\textsc{Circ}(\mathcal{C})) \leq^{fpt}$ p-$\textsc{WSat}(\textsc{Circ}(\bigvee))$ via the following reduction: Given a $\mathcal{C}$-circuit we compute for every gate ν with ℓ_ν inputs and label $C \in \mathcal{C}$ the values $C[0^{\ell_\nu}]$ and $C[1\,0^{\ell_\nu - 1}]$. This can be done in

polynomial time as a connective is *ptime*-computable. If both values are the same we replace the gate by a gate labelled with the respective constant 0 or 1 and delete the incoming edges. If the values differ, we know that $C \restriction_{\ell_\nu} = \bigvee \restriction_{\ell_\nu}$ and thus replace the gate by a gate labelled with $\bigvee$. Then we delete all gates from which there is no path to the output-gate. The resulting circuit is equivalent to the original circuit. By Lemma 1 we have $W[P](\bigvee) \subseteq \mathrm{FPT}$ and thus $W[P](\mathcal{C}) \subseteq \mathrm{FPT}$

Case 3: $\mathcal{C}$ is $\wedge$-closed, but not $\vee$-closed. Similar to Case 2 we get a reduction p-WSat(Circ($\mathcal{C}$)) $\leq^{fpt}$ p-WSat(Circ($\bigwedge$)) and therefore $W[P](\mathcal{C}) \subseteq \mathrm{FPT}$.

Case 4: $\mathcal{C}$ is neither $\vee$-closed nor $\wedge$-closed. Then every connective in $\mathcal{C}$ must be constant on inputs of length $\ell > 1$, and on inputs of length 1 either constant or identity. But then the problem p-WSat(Circ($\mathcal{C}$)) is in FPT: for every gate ν in the $\mathcal{C}$-circuit with $\ell_\nu > 1$ inputs and label $C \in \mathcal{C}$ we compute the value $C[1^{\ell_\nu}]$ and replace the gate by the according constant gate. For C-gates with one input we compute $C[1]$ and $C[0]$, and delete the gate if it is equivalent to identity, or replace it with an apropriate constant-gate otherwise. The parameterised weighted satisfiability problem for the resulting circuit clearly is in FPT.

If $\mathcal{C}$ on the other hand is not monotone, we know by equivalence (3), that $\mathcal{C}$ is able to simulate negation. Now if there is a $C \in \mathcal{C}$ and $m, n \in \mathbb{N}$, such that $C(m, n+2) \neq C(m+1, n+1) = C(m+2, n)$ or $C(m, n+2) = C(m+1, n+1) \neq C(m+2, n)$, then either $C(m, n+2) = 0$ and C is $\vee$- respectively $\wedge$-closed, or $C(m, n+2) = 1$ and $\neg C$ is $\wedge$- respectively $\vee$-closed. Thus substituting the gates in a boolean circuit by constant size $\mathcal{C}$-subcircuits, we get $W[P] \subseteq W[P](\mathcal{C})$.

If there are no such m, n as above, then for all $\ell \in \mathbb{N}$ and for all $C \in \mathcal{C}$ we have that $C \restriction_\ell$ is constant or $C(\ell - t, t) \neq C(\ell - (t+1), t+1)$ for all $t < \ell$. In the non-constant case the values of $C \restriction_\ell$ alternate between 0 and 1. Thus $C \restriction_\ell = \oplus \restriction_\ell$ or $C \restriction_\ell = (\neg\oplus) \restriction_\ell$. But then we have p-WSat(Circ($\mathcal{C}$)) $\leq^{fpt}$ p-WSat(Circ($\oplus$)) via the following reduction: For every gate ν with ℓ_ν inputs and label C compute the values $C(0, \ell_\nu)$ and $C(1, \ell_\nu - 1)$ and replace the gate by

- a gate labelled with c, if $C(0, \ell_\nu) = C(1, \ell_\nu - 1) = c$
- a gate labelled with $\oplus$, if $C(0, \ell_\nu) = 0$ and $C(1, \ell_\nu - 1) = 1$
- two gates computing $\oplus[1 \oplus [X_1 \ldots X_{\ell_\nu}]]$, if $C(0, \ell_\nu) = 1$ and $C(1, \ell_\nu - 1) = 0$

Thus with Lemma 1 we have $W[P](\mathcal{C}) \subseteq \mathrm{FPT}$. □

Corollary 1 (Characterisation). *Let $\mathcal{C}$ be a finite set of connectives. If $\mathcal{C}$ has at least two of the properties of being $\vee$-closed, $\wedge$-closed, or not monotone, then $W[P](\mathcal{C}) = W[P]$. Otherwise $W[P](\mathcal{C}) = \mathrm{FPT}$.* □

5 Conclusion

We saw that for every finite set $\mathcal{C}$ of connectives the class $W[P](\mathcal{C})$ is equal to $W[P]$ or collapses to FPT. The proof also yielded characterisations of the connectives for both alternatives. A question left open is whether we get a similar dichotomy for the classes $W[\mathrm{Sat}](\mathcal{C})$, which are defined via the parameterised

weighted satisfiability problems for $\mathcal{C}$-formulas ($\mathcal{C}$-circuits where all gates except for the input-gates have fan-out ≤ 1). Although there are results which settle the issue for connectives with bounded fan-in [2, Theorem 5.2.2], it is unclear how to handle connectives with unbounded fan-in. It is also not clear, whether for the dichotomy theorem to hold it is necessary to presuppose the constant gates 0 and 1.

References

1. Abrahamson, A.A., Downey, R.G, and Fellows, M.R.: Fixed-Parameter Tractability and Completeness IV: On Completeness for $W[P]$ and PSPACE analogues. Annals of Pure and Applied Logic, 73(3):235-276, 1995
2. Clote, P., and Kranakis, E.: Boolean Functions and Computation Models. Springer-Verlag, 2002
3. Downey, R.G., Fellows, M.R.: Fixed-Parameter Tractability and Completeness I: Basic Results. SIAM Journal on Computing, 24:873-921, 1995
4. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer-Verlag, 1999
5. Fellows, M., Flum, J., Hermelin, D., Müller, M., Rosamond, F.: W-hierarchies defined by symmetric gates. Theory Comput Syst, 46(2):311-339, 2010
6. Fellows, M., Hermelin, D., Müller, M., Rosamond, F.: A purely democratic characterization of $W[1]$. In: Proceedings of the 3rd IWPEC'08, LNCS 5018, pages 103–114, 2008
7. Flum, J, Grohe, M.: Parameterized Complexity Theory. Springer-Verlag, 2006
8. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press, 2006
9. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley, 1994

Agent Oriented Programming: from Revolution to Evolution (Position Paper)

Alex Muscar

University of Craiova, Romania
amuscar@software.ucv.ro

Abstract. Despite being around for quite some time, agents have failed to gain wide acceptance. Their AI heritage has forced them into a niche from which they cannot seem to escape: being the vehicle of AI experimentation. Even though the premises of Agent Oriented Programming (AOP) is a revolutionary departure from Object Oriented Programming (OOP) the vision has not materialized.
In this paper we propose taking a step back and looking at AOP as an evolution from OOP. Rather than viewing agents as specialized AI tools, we adopt a view on agents as a generic metaphor for building complex software systems. The guiding lines of our approach are: (i) overall conceptual simplicity; and (ii) the use of programming languages as the main means of expression. We will explore some paradigms and approaches that we think can greatly benefit the view of agents as an evolution from OOP.

1 Introduction

When introduced by Shoham almost two decades ago [16], Agent Oriented Programming (AOP) was intended as a higher level means of developing complex software systems – when compared to Object Oriented Programming (OOP). Soon after its introduction is was regarded by many as a "revolution in software" [7]. But the promises of the agent community have failed to materialize and agents haven't gained wide acceptance. They are mostly regarded as experimentation tools for the AI community instead of a technology for developing practical applications. Motivated by the agent paradigm's lack of success there are voices that predict its downfall [8]. At the same time some members of the agent community are trying to raise awareness to the lack of coherence and perspectives in the community [3,15].

We believe that the very fact of looking at AOP as a *revolutionary* step from OOP instead of an *evolutionary* one has been one of the main reasons that led to the current state of affairs. Given the AI heritage of the agent community it is no wonder that most of its efforts went into models of rational agents. While this has led to a series of interesting results it also meant that other aspects of agent systems such as the organizational ones or the ones regarding the environment in which agents operate have not been given the attention they deserve [6]. This

Andrew V. Jones (Ed.): Proceedings of the 1st Imperial College Student Workshop (ICCSW '11), pp. 52–58. September 29th–30th 2011, London UK.

was most unfortunate as those two aspects are essential to using agents as a generic metaphor for building complex software systems.

The purpose of this paper is twofold. First, it takes a conceptual step back, and looks at agents as an evolution of the OOP and the Actors model [1]. Second, it it tries to distill the core concepts that will go into the making of a new agent language, which we plan to develop in the near future. We have chosen two basic concepts which we plan to use as the basis of our language: *reactive objects* [11] and *dynamic delegation* as featured in *prototypical* languages such as SELF [17,4,18]. Based on this two concepts we believe that more advanced topics such as concurrency, autonomy and security can be tackled.

In [6], Dastani identifies the development of programming languages that combine the three key abstractions of multi-agent system (MAS) – agents, organizations and environments – as one of the main challenges of the agent community. After a brief survey of some popular agent languages in sec. 2 we focus on each of the aforementioned aspects in sec. 3. We conclude in sec. 4.

2 Motivation and Background

As stated above we adopt the view that languages profoundly influence the way programmers design systems. Thus, we think that a language needs to provide: i) a *uniform* model (i.e. everything is an agent); ii) *self-sufficiency* (i.e. the capacity to extend the language from within itself); and iii) *reified* concepts from the problem domain (e.g. agents, environments, organizations). Given this requirements, in table 1 we make a succinct overview of three of the most popular agent languages: *JASON*, *GOAL* and *MetateM*.

	JASON	**GOAL**	**MetateM**
Uniformity	no	yes	yes
Self-sufficiency	no	no	no
Reified concepts	none	none	agents

Table 1. Comparing agent languages

Each of the languages considered in our overview fails for one for more of our requirements. JASON is neither uniform – agents and environments are separate entities, nor does it feature reified entities. If we consider belief and knowledge bases as being part of the agent we can consider that GOAL is a uniform language since it only operates with agents, but this does not prove to be that useful since agent operations are limited and environments are declared implicitly. MetateM is the closest language to our desired model. It is uniform in the sense that agents are used to represent other agent's environments reifying the concept of agents in the process. All three languages fail when it comes to self-sufficiency since they can only be extended in Java.

3 An evolutionary approach

We will address each of the three concepts identified by Dastani in [6] – agents, organizations and environments – while trying to provide uniformity, self-sufficiency and problem domain reification.

3.1 Individual agents

In their simplest definition agents are *autonomous and interacting entities* [6]. Agents are autonomous in the sense that they can *decide* which action to perform next in order to reach their objective. This is indeed a very lose definition of agents. Instead, for the rest of this paper, we shall operate with the following definition: an agent is *a computational entity that (i) has its own thread of control and can decide autonomously if and when to perform a given action; (ii) communicates with other agents by asynchronous message passing*[1]. This definition implies *concurrently* executing agents with *reactive* behaviors. This is in line with the definition of AOP systems provided by Shoham in [16] where AOP is a specialization of OOP in the sense of the Actor model.

We begin by looking at three possible abstractions for single agent entities: objects, actors and *reactive objects* [11]. We have chose the former two since our purpose is going back to the origins of the agent metaphor, and the latter as an evolution of objects which borrows from actors.

	Objects	**Actors**	**Reactive Objects**
Granularity	object	actor	object
Execution model	sequential	concurrent	concurrent
Communication	sync	async	sync and async
Message ordering	strict order	no restriction	strict order

Table 2. Comparing agent abstractions

It is obvious from table 2 that reactive objects combine the best features of OOP and the Agent model. They are autonomous units of execution that are either executing the *sequential* code of exactly one method, or passively maintaining their state [11]. While this almost fits our definition of agents, there is still one aspect that we haven't fully addressed: autonomy. Reactive objects are autonomous in the sense of having their own thread of control there is still the issue of *decision making*, but in order to be fully autonomous an agent needs a decision making component [6]. A popular choice in the agent community is the *BDI model* which can be easily mapped on the concepts already introduced above. This view of agents gives us a higher level view which is especially useful when tackling the inherent concurrency in the execution model that we have

[1] We consider asynchronous programming as being characterized by many simultaneously pending reactions to internal or external events

defined for our agents. An interesting approach to dealing with concurrency by focusing on the execution of plans instead of programs or processes has been proposed for the E language: *communicating event loops* [9].

The communicating events loops model can be seen as a refinement of the reactive object semantics. When a thread of control[2] needs to send an asynchronous message send it ads an entry to a queue of *pending deliveries*. During a *turn* a pending delivery is dequeued, the message is sent, and all the resulting synchronous calls are executed. When a turn finishes, another entry from the pending queue is dequeued and the process starts all over again.

To address distributed scenarios a further refinement is introduced: objects running in the same event loops can be invoked both synchronously and asynchronously while objects running in remote event loops can only be invoked asynchronously. This offers isolation for plans executing in different event loops.

The communicating events loops model offers two key properties for concurrent systems:

Asynchrony messages between two event loops are sent asynchronously and the event loop controls when they are sent the risk of deadlocks is eliminated because an event loop can never interrupt its currently executing plan to wait for another event loop to execute its plan; and

Serializability an event loop reacts to incoming events serially the risk of race conditions is eliminated.

For these reasons this model has also been adopted by AmbientTalk [5] which deals with ad hoc networks, highly distributed and concurrent systems.

3.2 Organization

In order for a multi-agent system to achieve its purpose the behavior of individual agents has to be organized. Furthermore, the system needs to maintain some global invariants. This can be done endogenously, by making the organizational and regulatory aspects part of the agent. We think that the endogenous approach is cumbersome and that it obscures the development of individual agents. We find the exogenous approach much more appealing: agent's actions are externally controlled. This leads to a development style that's more modularized and decoupled.

Let's once again look at the OOP community for inspiration: dynamic delegation as featured in prototypical languages, such as SELF, seems to offer the flexibility we are looking for when it comes to structuring adaptive distributed systems, and we have chosen it for its elegance and conceptual simplicity. In [18], the authors introduce the concept of *traits* for organizing large prototypical systems. Traits are akin to abstract types: they are used to factor out common functionality shared multiple objects. Thus, altering the behavior of a trait also alters the behavior of all the objects that share it.

[2] For the purpose of this article we can think of *thread of control* and *event loop* as interchangeable terms.

Some of the research in agent organization has focused on using *coordination artifacts* – from the *Agents and Artifacts* (A&A) theory [12]. According to [13] coordination artifacts are entities designed to provide some kind of functionality or service, they have a well-defined interface, providing operations that can be invoked by agents. Indeed, coordination artifacts have much in common with traits. Another benefit of traits is that hey naturally handle dynamically changing behavior either by altering the trait as stated before, or by changing an object's traits on the fly.

Another advantage of adopting traits has to do with security. Since traits act as abstract types defining reusable behavior they are suitable for a system implementing object capabilities [10,2]. In this model there is no *ambient authority*. All actions are performed via unforgettable references to objects. The only component of the system which has full authority is the *powerbox* which, based on some security criteria, can hand references to various capabilities to the requesting objects. This system is extremely flexible since instead handing a reference to the actual capability, the powerbox can pass a reference to a stripped down version of the capability (e.g. in the case of a file system access capability it can pass a version that can only read files, but not delete them). Traits are especially suitable for this model since new traits can be defined in terms of existing one by *refining* their behavior [18].

3.3 Environment

One of the main models proposed for representing environments for MAS is A&A [14]. As stated in sec. 3.2, there is a close resemblance between coordination artifacts and traits. But this resemblance is not limited to coordination artifacts. Artifacts are generic bundles of behavior that provide a usage interface. They can be co-constructed and co-used by agents and they can be grouped in *namespaces*. This view of artifacts maps directly on prototypical objects, which thanks to their generality can act as traits, regular objects and namespaces. This opens up new ways of organizing systems.

3.4 A coherent view

By combining the communicating event loops model with the delegation semantics of prototypical languages we gain uniformity. We can define agents, artifacts and environments as event loops. Furthermore, since we can change delegation links at runtime, we can also reify agent behavior as traits[3]. This buys us flexibility and adaptability: an agent can adapt to varying environmental conditions by delegating to different traits. The advantages span to the distributed aspects of the system as well. By introducing a distinction between local and remote event loops we can take advantage of uniformity further: an agent can delegate to either a local or remote counterpart. In turn this can benefit agent mobility

[3] Which are also agents.

since agents are self contained entities. As long as their delegation links are appropriately adapted, agents can continue to function the same way on the new host as they did on the original one.

While this features could be implemented as an Domain Specific Language (DSL) – or even Embedded Domain Specific Language (EDSL) – we think that the approach would benefit most by being implemented as a full fledged language. Together, the features from the previous paragraph, make up for a language that conforms to our initial criteria: uniformity, self-sufficiency and reification of concepts from the problem domain. It also exposes the three main features identified by Dastani in [6] to the programmer.

4 Conclusion and Future Work

In this paper we have presented our view on agent languages. We have addressed the three main abstractions of such languages: agents, organizations and environments. In doing so we took a step back and tried to gear agent development toward a more generic audience while still keeping its high level view on computation.

We also looked at what hints the agent paradigm can take from OOP, and especially from the prototype-based model. We showed how the flexibility of prototypical objects can benefit the organization of MAS. We also briefly addressed how representing environments for agents can use the same model thus leading to a uniform representation.

In the near future we will closely focus on the ideas proposed in this paper, especially on implementing an agent language based on communicating event loops and we will investigate the use of traits in providing security, organization and the reification of environments. We will also address some open issues that we glossed over in this paper because of the lack of space: i) the interaction of the inherent dynamism in prototypical systems like SELF with the object capability model and event loops; ii) re-introducing some static checks in the context of such a highly dynamic system; and iii) language extensibility both at the syntactic and semantic level.

Some additional problems are related to runtime system of such an agent language. In order to offer concurrency, most of the current agent languages, use one Operating System (OS) thread per agent. But OS threads are expensive so such a solution does not scale well for systems with many agents. Using a *thread pool* seems like a good solution, but have yet to study the interaction of event loops and thread pools. To complicate things further, choosing a concurrency model (e.g. futures, promises) has a deep impact on the design of the language (e.g. using promises with callbacks for when they are resolved leads to the well known problem of inversion of control).

These are all serious issues that need to be settled, but we feel that the basic concepts (reactive objects with dynamic delegation) make for a solid foundation for an agent language targeted at MAS.

References

1. Agha, G.: Actors: a model of concurrent computation in distributed systems. MIT Press, Cambridge, MA, USA (1986)
2. Bracha, G., von der Ahé, P., Bykov, V., Kashai, Y., Maddox, W., Miranda, E.: Modules as objects in newspeak. In: Proceedings of the 24th European conference on Object-oriented programming. pp. 405–428. ECOOP'10, Springer-Verlag, Berlin, Heidelberg (2010)
3. Castelfranchi, C.: Bye-bye agents? not. IEEE Internet Computing 14, 93–96 (March 2010)
4. Chambers, C., Ungar, D., Chang, B.W., Hölzle, U.: Parents are shared parts of objects: inheritance and encapsulation in self. Lisp Symb. Comput. 4, 207–222 (July 1991)
5. Cutsem, T.V., Mostinckx, S., Boix, E.G., Dedecker, J., Meuter, W.D.: Ambienttalk: Object-oriented event-driven programming in mobile ad hoc networks. In: Proceedings of the XXVI International Conference of the Chilean Society of Computer Science. pp. 3–12. IEEE Computer Society, Washington, DC, USA (2007)
6. Dastani, M.: Programming multi-agent systems (May 2011)
7. Guilfoyle, C., Warner, E.: Intelligent Agents: the new revolution in software. Ovum (1994), incomplete ref
8. Hewitt, C.: Perfect disruption: The paradigm shift from mental agents to orgs. IEEE Internet Computing 13, 90–93 (January 2009)
9. Miller, M.S., Tribble, E.D., Shapiro, J.: Concurrency among strangers: programming in e as plan coordination. In: Proceedings of the 1st international conference on Trustworthy global computing. pp. 195–229. TGC'05, Springer-Verlag, Berlin, Heidelberg (2005)
10. Miller, M.S.: Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control. Ph.D. thesis, Johns Hopkins University, Baltimore, Maryland, USA (May 2006)
11. Nordlander, J., Jones, M.P., Carlsson, M., Kieburtz, R.B., Black, A.P.: Reactive objects. In: Symposium on Object-Oriented Real-Time Distributed Computing. pp. 155–158 (2002)
12. Omicini, A.: Formal respect in the a&a perspective. Electron. Notes Theor. Comput. Sci. 175, 97–117 (June 2007)
13. Ricci, A., Viroli, M.: Coordination artifacts: a unifying abstraction for engineering environment-mediated coordination in MAS. Informatica 29(4), 433–443 (2005)
14. Ricci, A., Piunti, M., Viroli, M., Omicini, A.: Environment programming in cartago. In: El Fallah Seghrouchni, A., Dix, J., Dastani, M., Bordini, R.H. (eds.) Multi-Agent Programming:, pp. 259–288. Springer US (2009)
15. Santi, A.: From objects to agents: Rebooting agent-oriented programming for software development
16. Shoham, Y.: Agent-oriented programming. Artif. Intell. 60, 51–92 (March 1993)
17. Smith, R.B., Ungar, D.: Self: The power of simplicity. Tech. rep., Mountain View, CA, USA (1994)
18. Ungar, D., Chambers, C., Chang, B.W., Hölzle, U.: Organizing programs without classes. Lisp Symb. Comput. 4, 223–242 (July 1991)

Conditional Labelling for Abstract Argumentation ⋆

Guido Boella[1], Dov M. Gabbay[2], Alan Perotti[1], Leendert van der Torre[3], and Serena Villata[4]

[1] Dipartimento di Informatica, Università di Torino {guido,perotti}@di.unito.it
[2] King's College London dov.gabbay@kcl.ac.uk
[3] ICR, University of Luxembourg leon.vandertorre@uni.lu
[4] INRIA, Sophia Antipolis serena.villata@inria.fr

Abstract. Agents engage in dialogues having as goals to make some arguments acceptable or unacceptable. To do so they may put forward arguments, adding them to the argumentation framework. Argumentation semantics can relate a change in the framework to the resulting extensions but it is not clear, given an argumentation framework and a desired acceptance state for a given set of arguments, which further arguments should be added in order to achieve those justification statuses. Our methodology, called *conditional labelling*, is based on argument labelling and assigns to each argument three propositional formulae. These formulae describe which arguments should be attacked by the agent in order to get a particular argument *in*, *out*, or *undecided*, respectively. Given a conditional labelling, the agents have a full knowledge about the consequences of their attacks on the acceptability of each arguments, without having to recompute the overall labelling of the framework for each possible set of attacks they may raise.

1 Introduction

Agents engage in dialogues having as goals to make some arguments acceptable or unacceptable: for instance, *agent A wins the auction* or *agent B is proven guilty*. At each turn, an agent owns a set of possible arguments she can add to the framework: each addition of further arguments to the framework is called a *move*. Argumentation semantics allow us to relate the introduction of a new argument (a *move*) to the resulting justification status of an argument (the *goal*): for instance, *if you defeat argument α then argument β will be labeled undec*. What is missing is a mechanism for making inferences from goals to moves: suppose an agent wants to make an argument β *undec*. How can she compute which arguments to add in order to achieve this goal? What she can do is to try and simulate the introduction of every possible argument she owns in the framework and then compute β's resulting label, comparing it to her goal. Beside this exhaustive approach there is no way, so far, for an agent to know which move to make in order to achieve her goal. Since reaching a goal may require the insertion

⋆ A longer version of this paper appeared in the proceedings of TAFA11 [2]

Andrew V. Jones (Ed.): Proceedings of the 1st Imperial College Student Workshop (ICCSW '11), pp. 59–65. September 29th–30th 2011, London UK.

of several arguments, the complexity of the exhaustive approach is exponential (wrt cardinality of the powerset) over the number of arguments an agent can add to the framework.

The research questions of the paper are:

1. What kind of information can we associate to each argument concerning its possible justification statuses depending on the acceptability of other arguments in the framework?
2. How to compute this information in an efficient way?

We deal with abstract argumentation frameworks [4], where the internal structure of the arguments is left unspecified. We are inspired by Caminada's labelling [3], which assigns to each argument a label *in*, *out*, *undec*, and we extend this idea by assigning a triple of propositional formulae, called *conditional lables*, to every argument in the framework. These formulae are a guide in the dialogic process and suggest which move should be made next. Note that these formulae (and the algorithmic process to compute them) are in no way related to the number of agents: our approach does not depend on the number of argumenting agents and we apply it to a two-agent scenario for the sake of explanation. In this paper we focus on the *grounded* semantics, since it always allows to compute one single labelling. Our approach can be extended to deal with different semantics, but semantics with multiple or no extensions must be handled with care, in particular when investigating about credulous approaches to multiple extensions semantics.

2 Conditional labels

Our goal is to enrich each argument with some information about his *vulnerability*, i.e., we want to know how this argument could be successfully (even if indirectly) attacked, defended or made undecided. We purposely restrict our attention to argument defeating, due to two considerations: first of all, attacks are not resources but consequences of the insertion of the arguments and given a couple of arguments the existence of attacks between them is determined and not subject to strategic moves of agents. In second place, the building of an argumentation framework is a monotonic process and arguments can be defeated with new arguments rather than removed from the framework. Hence our proposal is to attach **three** formulae to each argument, meaning respectively

- Which arguments should be attacked in order to have this argument labelled *in*?
- Which arguments should be attacked in order to have this argument labelled *out*?
- Which arguments should be attacked in order to have this argument labelled *undec*?

Given an argumentation framework $\langle A, R \rangle$ (as defined in [4]), we associate to each argument α three formulae: α^+, α^-, $\alpha^?$. We indicate a generic formula associated to argument α as α^*. The language of the formulae is the same:

Definition 1. *(Language of conditional labels)*

- *if* $\beta \in A$, β° *is a formula.*
- $\top$ *and* $\bot$ *are formulae*
- *if* α_1^* *and* α_2^* *are formulae, also* $\alpha_1^* \wedge \alpha_2^*$ *and* $\alpha_1^* \vee \alpha_2^*$ *are.*

The interpretation of the formulae is: a formula α^+, if satisfied, guarantees that the related argument α is accepted (labelled *in*). The same holds for α^- formulae for *out* labels and $\alpha^?$ formulae for *undec* labels respectively. The atoms of those formulae are argument names β° or the special values $\top, \bot$.

- β° means *the agent has to defeat argument* β (to reach her goal)
- $\top$ means *the agent does not need to do anything* (to reach her goal)
- $\bot$ means *the agent can not do anything* (to reach her goal)

Due to space constraints, formal definition of the use of conditional labels is omitted; the main intuition will be just introduced informally. For techincal details see [2].

Each conditional label is composed by a head and a body; given an argument α, its three conditional labels are $\alpha^+ : body_\alpha^+$, $\alpha^- : body_\alpha^-$, $\alpha^? : body_\alpha^?$. A *targetset* for a label is a minimal set of arguments such that the arguments names are a solution for the label.

When we modify a framework via a move M we can defeat a set of arguments $defeat(M)$. If this set is one of the allowed target sets for the conditional label *lab* of an argument α, then the labelling of α in the resulting framework will be the one expressed by the head of the label *lab*: for instance, if $defeat(M)$ is a target set for $body_\alpha^+$, after M α's label will be *in* (same for $body_\alpha^-$ and *out* and $body_\alpha^?$ and *undec* respectively).

From a practical point of view, suppose that an agent wants to defend argument α: she has to compute the label α^+ and the target sets of the formula (that is, the minimal sets of solutions that satisfy the body of the label) are the arguments that have to be defeated in order to defend α.

3 Computing conditional labels

Our approach can be decomposed in four phases:

1. associate each argument to three base (or local) labels,
2. compute conditional labels by substitution,
3. find target sets (for instance, by dnf-normalizing the formulae),
4. find a move such that it satisfies a target set of the goal formula.

The local labels correspond to:

$$a^+ = \bigwedge_{b \ s.t. \ (b,a) \in R} b^-$$

The meaning of this formula is: *in order to ensure a's acceptance, all of a's attackers must be out.*

$$a^- = a^\circ \vee \bigvee_{b \ s.t. \ (b,a)\in R} b^+$$

The meaning of this formula is: *in order to ensure a's rejection, either a is defeated or one of a's attackers is accepted.*

$$a^? = \left(\bigvee_{b \ s.t. \ (b,a)\in R} b^? \right) \wedge \left(\bigwedge_{b \ s.t. \ (b,a)\in R} b^- \vee b^? \right)$$

The meaning of this formula is: *in order to have an argument a undecided, at least one of a's attackers has to be undecided and all of a's attackers must be out or undecided.*
Note that this definition of grounded semantics mirrors Dung's original formulation ([4]).
The a° in the second formula means *a has to be defeated* and no substitution is required; b^+, b^- and $b^?$ refer to other formulae and have to be substituted to the actual formulae they refer to.

After this initial definition, the substitution process (phase two) takes place. It consists in substituting the references to other labels to those labels' actual values.

Simplifications need to be specified as follows:

- $\top \vee \alpha \rightsquigarrow \top$ (you either do nothing or do α: doing nothing is more convenient)
- $\bot \vee \alpha \rightsquigarrow \alpha$ (you can either fail or do α: in order to succeed you have to do α)
- $\top \wedge \alpha \rightsquigarrow \alpha$ (you have to both do nothing and α, therefore α)
- $\bot \wedge \alpha \rightsquigarrow \bot$ (you fail and you have to do α: you still fail)
- $\alpha \wedge \alpha \rightsquigarrow \alpha$
- $\alpha \vee \alpha \rightsquigarrow \alpha$
- $\alpha \vee (\alpha \wedge \beta) \rightsquigarrow \alpha$
- $\alpha \wedge (\alpha \vee \beta) \rightsquigarrow \alpha$

Let $i, j \in \{+, -, ?\}$. If α^i appears in the body of α^j:

- if $i = j = ?$, $\alpha^i \rightsquigarrow \top$
- else, $\alpha^i \rightsquigarrow \bot$

We express our termination conditions as simplification rules. The meaning is the following: if, substituting in the body of a conditional formula for an argument α, a conditional formula over the same argument is reached, the argument α belongs to a loop. So in this case the $a^?$ label is satisfied while a^+, a^- are not: if there is no way to give this argument an *in-out* label navigating the whole loop, it is pointless to go through the whole loop again.

4 Example

We now present some examples of conditional labelling.

Consider the example visualized in Figure 1.1. The basic labels are:

- $a^+ : b^-,\ \ a^- : b^+ \vee a^\circ,\ \ a^? : b^?$
- $b^+ : a^-,\ \ b^- : a^+ \vee b^\circ,\ \ b^? : a^?$

Solving the labels, for a we get $a^+ : b^\circ$, $a^- : a^\circ$, $a^? : \top$.

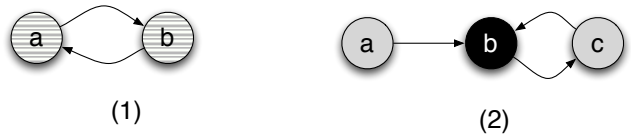

Fig. 1: Basic frameworks. Plain grey nodes represent *in* arguments and black nodes represent *out* arguments. *undec* arguments are depicted as dashed grey nodes.

Consider the example visualized in Figure 1.2. The basic labels are:

- $a^+ : \top,\ \ a^- : a^\circ,\ \ a^? : \bot$
- $b^+ : a^- \wedge c^-,\ \ b^- : a^+ \vee c^+ \vee b^\circ,\ \ b^? : (a^? \vee c^?) \wedge (a^- \vee a^?) \wedge (c^- \vee c^?)$
- $c^+ : b^-,\ \ c^- : b^+ \vee c^\circ,\ \ c^? : b^?$

Consider argument b: it is *out*, but can be labelled *in* if we attack both a and c or *undec* if we attack a (thus activating the $b-c$ loop). We compute the conditional labels in the following way:

$b^+ : a^- \wedge c^-$
$= a \wedge (b^+ \vee c^\circ)$
$= a^\circ \wedge (\bot \vee c^\circ)$
$\rightsquigarrow a^\circ \wedge c^\circ$ (b can be labelled *in* by defeating a and c)

$b^- : a^+ \vee c^+ \vee b^\circ$
$= \top \vee b^- \vee b^\circ$
$= \top \vee \bot \vee b^\circ$
$\rightsquigarrow \top$ (no move is required in order to label b *out*)

$b^? : (a^? \vee c^?) \wedge (a^- \vee a^?) \wedge (c^- \vee c^?)$
$= (\bot \vee b^?) \wedge (a^\circ \vee \bot) \wedge ((b^+ \vee c^\circ) \vee b^?)$
$\rightsquigarrow (b^?) \wedge (a^\circ) \wedge ((b^+ \vee c^\circ) \vee b^?))$
$= (b^?) \wedge (a^\circ) \wedge ((\bot \vee c^\circ) \vee \top)$
$\rightsquigarrow (b^?) \wedge (a^\circ) \wedge (\top)$
$= (\top) \wedge (a^\circ) \wedge (\top)$
$\rightsquigarrow a^\circ$ (b can be labelled *undec* by defeating a)

The conditional labels computed mirror the intuitive description of the framework we gave and model it in a formal way.

5 Related Work

Conditional labelling is closely related to the dialogues games [5, 1]. Among others, Prakken [5] presents a formal framework for a class of argumentation dialogues, where each dialogue move either attacks or surrenders to a preceding move of the other participant. Amgoud and Hameurlain [1] argue that a strategy is a two steps decision process: i) to select the type of act to utter at a given step of a dialogue, and ii) to select the content which will accompany the act. Roth et al. [6] start from two principles: i) the outcome of a dispute depends on the strategies actually adopted by parties, but ii) this does not mean that the outcome can never be predicted because by using game theoretical solution concepts, the actions themselves can often be found. In comparison with this kind of frameworks, we share the idea that the first step consists in choosing the next move depending on the strategies of the agents. The differences are that we are not interested in providing the complete framework for argumentation dialogues games, we aim at providing a tool which can be used in those systems and which can be integrated with strategies. We do not restrict our framework to deal with two agents, and we extend the well-known argumentation labelling in order to provide a complete information about the argumentation framework on which it is applied.

6 Summary

In this paper we present a new kind of argument labelling, called conditional labelling. Conditional labelling allows to associate to each argument the information concerning its possible justification statuses, depending on the changes in the framework. In particular, we express this information by means of propositional formulae which express which arguments should be attacked in order to get the desired argument accepted, not accepted, or undecided. While it is quite straightforward to assign those conditional labels in argumentation frameworks without cycles and multiple attacks, it is rather complicated in the general case. When an argumentation framework with cycles is considered, it is possible to have in the conditional label α^* of an argument another α^* because the conditional labelling algorithm, using substitution, looks for all the attackers of the node until it finds the node itself. The conditional labelling allows the agents to avoid the exhaustive search of all the possible combinations in adding new arguments, and decreases the exponential complexity this search requires.

Future work addresses several issues: first of all, a deeper investigation on the complexity results related to the computation of the new labellings is necessary. From a purely argumentative perspective, we want to find out how conditional labels can be useful *after* a move: that is, if the previous information can be used to compute new conditional labels after the framework has been modified. Associating a cost concept to moves, our labelling lets agents link action costs to goals' outcomes, and can therefore be used as an underlying mechanism to develop strategies in a game theoretical context.

References

1. L. Amgoud and N. Hameurlain. A formal model for designing dialogue strategies. In H. Nakashima, M. P. Wellman, G. Weiss, and P. Stone, editors, *Autonomous Agents and Multiagent Systems (AAMAS)*, pages 414–416. ACM, 2006.
2. G. Boella, D. Gabbay, A. Perotti, L. van der Torre, and S. Villata. Conditional labelling for abstract argumentation. In *Procs. of the 1st Workshop on Theory and Applications of Formal Argumentation (TAFA), 2011.*
3. M. Caminada. On the issue of reinstatement in argumentation. In M. Fisher, W. van der Hoek, B. Konev, and A. Lisitsa, editors, *Journées Européennes sur la Logique en Intelligence Artificielle (JELIA)*, volume 4160 of *Lecture Notes in Computer Science*, pages 111–123. Springer, 2006.
4. P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.*, 77(2):321–358, 1995.
5. H. Prakken. Coherence and flexibility in dialogue games for argumentation. *J. Log. Comput.*, 15(6):1009–1040, 2005.
6. B. Roth, R. Riveret, A. Rotolo, and G. Governatori. Strategic argumentation: a game theoretical investigation. In *International Conference on AI and Law (ICAIL)*, pages 81–90. ACM, 2007.

A Persuasive Dialogue Game for Coalition Formation

Luke Riley[1]

Department of Computer Science,
University of Liverpool,
L69 3BX, UK
{L.J.Riley@liverpool.ac.uk}

Abstract. In this paper, I propose a formal dialogue framework that enables autonomous agents to engage in a process of practical reasoning, in which they can propose to form coalitions that perform joint actions, using argumentation. An argumentation scheme is used to drive this coalition formation process that results in qualitative payoffs. This paper builds on existing work that uses value-based argumentation in the context of a dialogue system, which has been empirically verified. This framework is designed explicitly for closed cooperative systems where agents hold different preferences.

Keywords

Multi-Agent Systems, Dialogue Games, Argumentation, Coalition Formation.

1 Introduction

Coalition formation is a major area within multi-agent systems research where agents form groups to achieve mutually shared goals to receive some payoff, often characterised in quantitative terms. However, not all situations allow for an obvious quantitative payoff and can be defined more explicitly in terms of goals that can be achieved or not. In qualitative coalition games an agent is satisfied if its goal is achieved or dissatisfied otherwise [14]. When forming teams, problems may occur, as it is not guaranteed all the agents of the system share the same views of the world and so disputes on what teams to form and why could arise.

Argumentation is a process where agents can reason about different beliefs to come to some logical conclusions. Recent work in argumentation suggests some agent systems can be more richly described with the inclusion of social values [2, 12] as opposed to just describing systems with goals. These values can be used to describe a social interest an agent has (for example, lowering taxes promotes entrepreneurship), which will be increased/decreased by moving from one state to another. In this work, the values (matched with an ordering over these values) will be used as the qualitative reasons for why agents form teams and prefer some teams to others.

Agents can communicate their arguments to each other through the use of dialogue games. Dialogue games are rule-governed interactions where each player moves by making utterances [9]. Dialogues frameworks have been previously used to form teams [7] but not from the approach of using agent argumentation from a persuasive context. Persuasion is one of the 6 main dialogue types defined by Walton and Krabbe in

Andrew V. Jones (Ed.): Proceedings of the 1st Imperial College Student Workshop (ICCSW '11),
pp. 66–72. September 29th–30th 2011, London UK.

their influential model of human dialogues. It is described as one participant seeking to persuade another about something not currently accepted [13].

This paper shows how an argumentation scheme for practical reasoning (that consists of defeasible premises matched to a defeasible conclusion for a joint action) [2] paired with its associated critical questions (CQs) will drive the coalition formation process. The CQs can challenge the premises or conclusion of the scheme and so become collaborative learning aids for the agents to find the best coalitions. If a CQ is left unanswered then the instantiation of the argumentation scheme it attacks fails to hold [10]. Using argumentation schemes and critical questions has previously been shown to be a valid extension to dialogue games (e.g. [10, 4, 11]) but no work has been completed on using this method to form coalitions.

Agents join the coalitions using a pro-active approach. This pro-active approach requires the agents to volunteer for a coalition by making the appropriate utterance (See Table 1, Section 3). The overall aim of the dialogue game will be to partition agents into appropriate coalitions that take into account all the preferences of the agents in the system.

The paper is structured as follows. Section 2 recapitulates some elements of the dialogue system from [4] and gives an overview of the modifications to its argumentation model and dialogue framework, which were empirically evaluated in [11]. Section 3 details the dialogue framework proposed. Section 4 gives a dialogue example and shows how the new system proposed evaluates the arguments to reach a conclusion on the coalition structure (the collection of coalitions) to recommend. Section 5 concludes the paper.

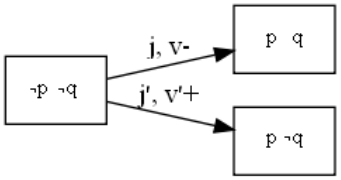

Fig. 1: Illustration of an agent's VATS (See Definition 1). j and j' are the joint-actions needed to move to another state, while v and v' are the values that are associated with these state changes. In this example v is demoted and v' is promoted.

2 Argumentation Model Used

For handling reasoning about the effects of actions, the following argumentation scheme for practical reasoning is used, modified from [4]. It is used to allow the agents to form arguments for coalitions, termed **coalition arguments**.

In the current circumstances R, joint action J should be performed, by coalition C, which will result in the new circumstances S, which will promote/demote the value V.

Circumstances *R* and *S* are represented as tuples of propositions, visualised in Figure 1. Joint action *J* is a tuple of single actions denoted $\langle \alpha_m, ..., \alpha_n \rangle$, Coalition *C* is a tuple of agent and single action pairs, where a pair is denoted (x, α), with the intended interpretation that if this coalition is agreed upon then each agent in *C* will perform the single action it is paired to, denoted C^x_α. If no agent has yet been assigned the single action α, this is denoted $C^?_\alpha$.

An agent may propose a joint-action including its justification, by instantiating this scheme. Agents can initially only add themselves to the coalition variable, until enough information is acquired about the other agents of the system so that accurate assumptions can be made. Other agents can then challenge instantiations by posing CQs associated with the scheme. The questions associated with the above scheme raise potential issues with: the validity of the elements instantiated in the scheme; the connections between the elements of the scheme and the side effects of the joint-actions [4]. Example CQs here are: *does doing the the joint action have a side effect which demotes another value?* and *assuming the circumstances does the joint action have the stated consequences?*.

A formally instantiated version of this scheme is denoted $\mathcal{A} = \langle q_x, j, c, q_y, v, s \rangle$ where q_x is the current state, j is the joint action, c is the coalition of agents paired to single actions, q_y is the new state, v is the value associated with this state transition and s (where $s = \{+, -, =\}$) is the sign indicating whether the value is promoted/demoted/not affected respectively. The coalition variable does not have to be completely instantiated upon the first utterance. This is to allow agents flexibility in their arguments, with the semantic meaning coming from the utterance that the instantiation is associated with (see Table 1, Section 3 for the full utterance list). An $\mathcal{A}$ will represent a proposal if $|c| < |j|$ as the coalition does not yet have a sufficient amount of members to carry out the joint action and so requires others to complete the proposal. $\mathcal{A}$ will represent an assertion if $|c| = |j|$ as the coalition now has enough members to carry out the joint action and is therefore ready to form. $\mathcal{A}$ will represent an objection if it is in the form of a CQ. A formalised CQ is instantiated as a modified version of $\mathcal{A}$ intended to reflect the question it represents in a logical form. The complete formalised CQ list is cut for space, but reflects the work of [11], expanded to incorporated the inclusion of the coalition and joint action variable.

To represent the agents' environment and help the agents create instantiations of the argumentation scheme a Value-Based Alternating Transition System (VATS) is used. It is a modified version of an Action-Based Transition System (AATS) [15], which is grounded in Alternating-time Temporal Logic (ATL). An example VATS diagram can be seen in Figure 1. Every agent in the system is assigned a VATS, which is a modified version of the one outlined in [4] and is summarised below:

Definition 1: *The VATS formalism is as follows: A* **VATS** *for an agent x, denoted S^x, is a 12-tuple*
$\langle Q^x, q_0^x, Ac^x, Av^x, Ja^x, Ag^x, \rho^x, \tau^x, \Phi^x, \pi^x, \delta^x, \xi^x \rangle$ *s.t.:*

- Q^x *is a finite set of* states*;*
- $q_0^x \in Q^x$ *is the designated* initial state*;*
- Ac^x *is a finite set of* single actions*;*
- Av^x *is a finite set of* values*;*
- Ja^x *is a finite set of* joint actions*, where each joint action is composed of m single actions where $m \in \mathbb{N}$;*
- Ag^x *is a finite set of* agents*;*
- $\rho^x : Ja^x \mapsto 2^{Q^x}$ *is an* action precondition function*, which for each joint action $j \in Ja^x$ defines the set of states $\rho(j)$ from which j may be executed;*

- $\tau^x : Q^x \times Ja^x \mapsto Q^x$ *is a partial* system transition function*, which defines the state* $\tau^x(q, j)$ *that would result by the performance of* j *from state* q*. As this function is partial, not all joint actions are possible in all states;*
- Φ^x *is a finite set of* atomic propositions*;*
- $\pi^x : Q^x \mapsto 2^{\Phi^x}$ *is an interpretation function, which gives the set of primitive propositions satisfied in each state: if* $p \in \pi^x(q)$*, then this means that the propositional variable* p *is satisfied (equivalently, true) in state* q*;*
- $\delta^x : Q^x \times Q^x \times Av^x \mapsto \{+, -, =\}$ *is a* valuation function *which defines the* status *(promoted (*$+$*), demoted (*$-$*), or neutral (*$=$*)) of a value* $v \in Av^x$ *ascribed by the agent to the transition between two states.*
- $\xi^x : Ag^x \times Ac^x \mapsto \{\top, \bot\}$ *is a partial agent capability function which defines if an agent can perform the single action (*$\top$*) or not (*$\bot$*). This function is partial as not all agents can perform all single actions.*

Note, $Q^x = \emptyset \leftrightarrow Ac^x = \emptyset \leftrightarrow Av^x = \emptyset \leftrightarrow \Phi^x = \emptyset$.

Each agent also has a preference order over its *values*, of the form $v_1 \succ ... \succ v_n$ where $n = |Av^x|$, that ranks the values into an order where v_1 is the most preferred and v_n the least (termed an 'audience' in [3]). The set of all arguments that can be created from S^x is denoted $A(S^x)$. Ψ is a subset of all the possible arguments all the agents in the system can construct, denoted $\Psi \subseteq \bigcup_{\forall x_i \in \{x_1,...,x_n\}} A(S^{x_i})$ and represents the arguments x believes to be true for the current state.

The coalition arguments and CQs uttered in the dialogue will be evaluated to determine their acceptability by placing them in a *Value-Based Argumentation Framework* (VAF) [3], which is an extended version of Dung's abstract Argumentation Framework (AF) [8]. An AF is defined as follows:

Definition 2: *Dung's Argumentation Framework is a tuple AF = (*$Args$*, R) where* $Args$ *is a set of arguments and* R *is a binary attack relation* $R \subseteq Args \times Args$*.*

A VAF extends an AF in the following manner:

Definition 3: *A VAF is a 5-tuple:* $\langle Args, R, V, val, P \rangle$ *where* $Args$ *and* R *remain the same as Definition 1,* V *is a set of non-empty values,* val *is a function mapping elements of* V *to* $Args$ *and* P *is a set of possible audiences.*

In a VAF an attack $arg_1 R arg_2$ only succeeds (arg_1 **defeats** arg_2) for an audience p iff argument arg_1 is associated with the same or a higher value than argument arg_2 in audience p's preference order, and arg_1 has not been defeated by another argument in the VAF.

A set of arguments S is **acceptable** to an audience p iff $\forall arg_x \in Args$ if arg_x attacks an argument arg_y where $arg_y \in S$ there $\exists arg_z \in S$ where arg_z defeats arg_x according to p's preference order. S is a **preferred extension** (PE) of a VAF for audience p if S is the maximal acceptable set of arguments for p.

3 Dialogue Framework

The underlying assumptions of this proposed dialogue framework are that agents in this system occupy a benevolent environment and are correctly aware of their starting state. When a dialogue commences, the number of agents in the system must remain fixed, but in between dialogues this number can change. Dialogues commence when an event triggers one agent to desire to move to another state. This agent should perform the **open** move detailed below and then start the dialogue protocol.

The protocol is similar to the one in [4] but cut for space. It finds all the legal moves that an agent can utter for its turn, given the current dialogue and the agent identifier. All these moves will then be uttered in unison. The moves available to the agents allow them to **object** to an argument with a critical question, **propose** an incomplete coalition, **assert** a complete coalition or attempt to **close** the dialogue.

Agents interact to find the best way to partition themselves into coalitions (which is a process known as coalition structure generation). Every agent asserts all the arguments and critical questions it can construct, given the union of their VATS and the other agents' utterances. Dialogues consist of a sequence of moves referred to as $[m_r, ..., m_t]$, collectively formalised to $\mathcal{D}_r^t$ where $r, t \in \mathbb{N}$ [4]. The first move of the dialogue is always the **open** move.

All agents' proposals, assertions and objections are stored in a publicly readable commitment store that grows monotonically over time until a new dialogue resets the commitment stores. For a dialogue, $\mathcal{D}_r^t$, with participants $\{x_1, \ldots, x_n\}$, for all $x \in \{x_1, \ldots, x_n\}$, the commitment store is denoted CS_x^t.

The moves that the agents in this framework can make are detailed further in Table 1 below, modified from [4] to allow for coalitions to be formed and CQs to be seperated from coalition arguments:

Move	**Format**	**Pre-conditions**	**Post-conditions**
$open$	$\langle x, open, \Lambda \rangle$	No Dialogue open. $\Lambda = [x_1, ..., x_n]$ where Λ is the available system agents.	Dialogue commenced. All agents in Λ are committed to follow the dialogue protocol.
$propose$	$\langle x, propose, \Psi \rangle$	$\forall \mathcal{A} \in \Psi, \lvert c \rvert < \lvert j \rvert$ where $c, j \in \mathcal{A}$.	Commitment store updated.
$assert$	$\langle x, assert, \Psi \rangle$	$\forall \mathcal{A} \in \Psi, \lvert c \rvert = \lvert j \rvert$ where $c, j \in \mathcal{A}$.	Commitment store updated. All agents in c are committed to perform the single action they are paired to.
$object$	$\langle x, object, \Psi \rangle$	S^x conflicts with another argument.	Objection is stored in the commitment store.
$close$	$\langle x, close \rangle$	A Dialogue is open.	Dialogue closed **iff** all agents have performed a close move in a row (without another move inbetween).

Table 1. The moves available to the agents

4 Dialogue Example and Argument Evaluation

Here is an abstract example for an agent system with 4 agents $(x_1, ..., x_4)$, 7 arguments $(arg_1, ..., arg_7)$, 5 states $(q_1, ..., q_5)$, 3 joint actions (j_1, j_2, j_3), 2 values (v_1 and v_2) and 3 completely formed coalitions $(C1, C2, C3)$. The purpose of the dialogue is to partition agents into coalitions that achieve the agents social values.

x_1 **- (proposes an instantiation of the argument scheme) -** arg_1**:** As we are in state q_1, joint action j_1, where $j_1 = \langle \alpha_1, \alpha_2 \rangle$, will result in state q_2. $\text{C1}_{\alpha_1}^{x_1}$ is proposed but $\text{C1}_{\alpha_2}^{?}$ remains. This transition will promote value v_2.

x_2 **- (asserts an instantiation of the argument scheme that extends** arg_1**) -** arg_2**:** As we are in state q_1, joint action j_1, where $j_1 = \langle \alpha_1, \alpha_2 \rangle$, will result in state q_2. $\text{C1}^{x_1}_{\alpha_1}$ and $\text{C1}^{x_2}_{\alpha_2}$ are asserted. This transition will promote value v_2.
x_3 **- (proposes an instantiation of the argument scheme) -** arg_3**:** As we are in state q_1, joint action j_2, where $j_2 = \langle \alpha_1, \alpha_3 \rangle$, will result in state q_4. $\text{C2}^{x_3}_{\alpha_3}$ is proposed but $\text{C2}^{?}_{\alpha_1}$ remains. This transition will promote value v_2.
x_4 **- (asserts an instantiation of the argument scheme that extends** arg_3**) -** arg_4**:** As we are in state q_1, joint action j_2, where $j_2 = \langle \alpha_1, \alpha_3 \rangle$, will result in state q_4. $\text{C2}^{x_4}_{\alpha_1}$ and $\text{C2}^{x_3}_{\alpha_3}$ are asserted. This transition will promote value v_2.
x_1 **- (objects to** arg_3 **and** arg_4 **with a critical question) -** arg_5**:** As we are in state q_1, performing joint action j_2, where $j_2 = \langle \alpha_1, \alpha_3 \rangle$, will demote v_1.
x_1 **- (proposes an instantiation of the argument scheme) -** arg_6**:** As we are in state q_1, joint action j_3, where $j_3 = \langle \alpha_4, \alpha_5 \rangle$, will result in state q_5. $\text{C3}^{x_1}_{\alpha_5}$ is proposed but $\text{C3}^{?}_{\alpha_4}$ remains. This transition will promote value v_1.
x_2 **- (asserts an instantiation of the argument scheme that extends** arg_6**) -** arg_7**:** As we are in state q_1, joint action j_3, where $j_3 = \langle \alpha_4, \alpha_5 \rangle$, will result in state q_5. $\text{C3}^{x_1}_{\alpha_5}$ and $\text{C3}^{x_2}_{\alpha_4}$ are asserted. This transition will promote value v_1.

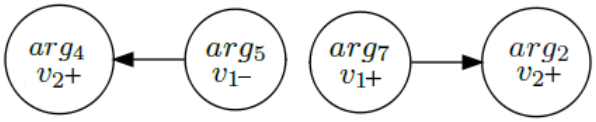

Fig. 2: Illustration of the VAF produced by the example dialogue.

From this dialogue the VAF is created. In the VAF are all the arguments uttered in an assert or objection move. The arguments uttered in a proposal move are not included as they hold incomplete coalitions that are not ready to form. The CQ arguments uttered in objection moves are in the VAF to determine the best coalitions to form. The attacks in the VAF come from coalition arguments that share an agent, coalition arguments that finish in conflicting states or conflicts that arise from the CQs.

To find the most preferred coalitions out of the remaining arguments one method that could be used is based on the borda count. Using this voting method all the agents of the dialogue have to summit their preference order to a centralised evaluating system which will then assign these preferences a score. The scoring method for a borda count is as follows: if there are k total system values, the most preferred in each preference order will be assigned the score $k - 1$, the second most preferred assigned the score $k - 2$ continuing until the least preferred gets zero. Using these borda count scores the system will be able to find one overall value order that will be used to find the overall system's preferred extension. Once all the attacks have been analysed and the preferred extension found, the arguments remaining that recommend a coalition will form the coalition structure.

The VAF created by the example dialogue can be seen in Figure 2. With a value order $v_1 \succ v_2$, arg_5 and arg_7 is the PE of the example VAF and so only C3 will form (as arg_5 doesn't recommend a coalition). A value order of $v_2 \succ v_1$ will mean the PE will contain arg_2 and arg_4 and so C1 and C2 will form. In this instance two coalitions are recommended as they are not conflicting. They do not conflict as they do not share

an agent and the shared propositions of q_4 and q_5 have the same truth value. This can happen in systems where agents do not share all the same propositions to describe the system.

5 Conclusions, Related and Future Work

Forming coalitions via argumentation has been proposed previously (e.g. [1, 6, 5]) but no persuasive dialogue game and protocol has been defined that produces a coalition structure. The dialogue game outlined here differs from the one of Amgoud [1] as her dialogue game is only used to find out if a coalition is in the set of acceptable coalitions and it is not used to form them.

This paper details preliminary work produced to formalise a dialogue game for coalition structure generation that could be modified for environments that are dynamic and open. In future work, the dialogue framework will be implemented, different methods for determining the overall value order will be considered and situations where agents are not satisfied with the final recommended coalitions will be explored.

References

1. L. Amgoud. An argumentation-based model for reasoning about coalition structures. *In ArgMAS*, pages 217–228, 2005.
2. K. Atkinson and T. J. M. Bench-Capon. Practical reasoning as presumptive argumentation using action based alternating transition systems. *Artificial Intelligence*, 171(10–15):855–874, 2007.
3. T. J. M. Bench-Capon. Persuasion in practical argument using value based argumentation frameworks. *J. of Logic and Computation*, 13(3):429–48, 2003.
4. E. Black and K. Atkinson. Dialogues that account for different perspectives in collaborative argumentation. In *Proceedings of the Eighth International Conference on Autonomous Agents and Multi-Agent Systems*, pages 867–874, 2009.
5. N. Bulling and J. Dix. Modelling and verifying coalitions using argumentation and atl. *Inteligencia Artificial*, pages 45–73, 2010.
6. C. Cayrol and M. Lagasquie-Schiex. Coalitions of arguments: A tool for handling bipolar argumentation frameworks. *Intelligent Systems*, pages 83–109, 2010.
7. F. Dignum, B. Dunin-Keplicz, and R. Verbrugge. Agent theory for team formation by dialogue. In *Proceedings of the Seventh International Workshop on Agent Theories, Architectures and Languages*, pages 141 –156, 2000.
8. P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77:321–357, 1995.
9. P. McBurney and S. Parsons. Dialogue games for agent argumentation. *Argumentation in Artificial Intelligence*, pages 261–280, 2009.
10. C. Reed and D. Walton. Argumentation schemes in dialogue. In *Proceedings of Ontario Society for the Study of Argumentation*, 2007.
11. L. Riley, K. Atkinson, T. Payne, and E. Black. An implemented dialogue system for inquiry and persuasion. *appeared at the 1st Int. Workshop on the Theory and Applications of Formal Argumentation (TAFA-11)*, 2011.
12. T. van der Weide, F. Dignum, J.-J. Meyer, H. Prakken, and G. Vreeswijk. Practical reasoning using values. In *Proceedings of the 6th int. Workshop on Argumentation in Multi-Agent Systems (ArgMAS 2009)*, pages 225–240, 2009.
13. D. Walton and E. Krabbe. *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*. State University of New York Press, Albany NY, 1995.
14. M. Wooldridge and P. Dunne. On the computational complexity of qualitative coalition games. *Artificial Intelligence*, pages 27–73, 2004.
15. M. Wooldridge and W. van der Hoek. On obligations and normative ability: Towards a logical analysis of the social contract. *J. of Applied Logic*, 3:396–420, 2005.

Model-based Self-Adaptive Components: A Preliminary Approach

Pedro Rodrigues and Emil Lupu

Department of Computing, Imperial College London, UK

Abstract. Due to the increasing scale, complexity, dynamicity and heterogeneity of modern software systems, it is not feasible to solely rely upon human management to guarantee a good service level with such availability demand. Self-managing systems are needed as an effective approach to deal with those issues by exploiting adaptive techniques to adjust a system. On top of that, model-based adaptation improves reliability, hence enhancing trust in self-managing systems. However, a centralised approach can be too complex to manage thus compromising system dependability. This paper presents a preliminary decentralised approach on model-based self-adaptive components.

1 Introduction

Self-management can be decomposed in various functions, as identified by IBM [1] as the MAPE-K loop: Monitoring, Analysis, Planning and Execution, all underpinned by system knowledge. The Monitoring service supervises the system and notifies system changes. The Analysis service receives these notifications and analyses system consistency as well as optimality, and sends a request to the Planning service to change the system when the system is not behaving as expected. The Planning service decides which changes have to be made and passes them to the Execution service to apply them.

To the best of our knowledge, most proposals in self-adaptive system are based on centralised management and a centralised model of the system. Centralised control of the details of all components in a system implies great complexity since adaptation concerning different levels of a system is dealt at the centralised manager; does not allow components to be completely autonomous and limits the reusability of management specifications concerning one component in other systems. Moreover, adaptation actions mainly focus on structural modifications, such as replacing components, hence not dealing with behavioural modifications.

In this paper we present a preliminary approach for the design of self-managing system composed of autonomous components for pervasive environments. The assumptions of systems administrators regarding adaptation action effects may not be verified at runtime. However, little work has been conducted on online reasoning about adaptation repercussions in terms of its outcome and costs.

This paper is organised as follows. Section 2 presents some of the relevant proposals regarding the services of the MAPE loop. Section 3 discusses our

Andrew V. Jones (Ed.): Proceedings of the 1st Imperial College Student Workshop (ICCSW '11), pp. 73–79. September 29th–30th 2011, London UK.

preliminary proposal for the design of composite autonomous components based on model-based adaptation. Section 4 ends this paper with closing remarks.

2 Background

This section discusses some proposals for the main concepts regarding self-managing systems. Monitoring is the foundation service of these systems as it provides online knowledge on system state. Relying upon that knowledge, decision-making services verify the need for adaptation and decide the most suitable adaptation action. Component models provide means for the specification of component structure and behaviour, which may support the decision-making service. Finally, self-adaptation frameworks combine those concepts to add self-managing properties to software systems.

2.1 Monitoring

A self-adaptive monitoring service should control the detail of collected monitoring data from sensors in the following manner: supervising main component metrics while the component fulfils its requirements and switching to more detailed monitoring when the system deviates from its normal behaviour. In this regard, an automated method for selecting a subset of metrics to be collected in the context of correlation-based monitoring was proposed in [11], resulting in detecting on average 66% of faults in case of all metrics were being connecting, though collecting only 30% of them. Alternatively, in the approach presented in [12] when an anomaly is detected the monitoring level is progressively increased until a fault root is found or the monitoring level achieved its maximum. These techniques allow to reduce power consumption while not compromising fault diagnosis accuracy.

2.2 Decision-Making

The decision-making service is responsible for choosing the most suitable adaptation action to be executed in face of a context or state change. Policies have been a successful way of expressing automated management of distributed systems and changes in the system behaviour at runtime [14]. While Event-Condition-Actions (ECA) policies express reactive actions based on the current system state, Goal and Utility-function policies express desirable system states.

The Stitch language [5] proposes a modification to ECA policies based on two constructions: *tactics* and *strategies*. Tactics implement the Condition-Action part and introduce a construct to indicate the expected behaviour of the tactic actions. Strategies are defined as a tree of Condition-Tactics nodes which define a condition for the tactic to be applied, an estimation of the time it needs to adapt the system and a list of conditional branches that define the following steps in the tree.

Alternatively, the application of genetic algorithms is proposed in [13] to determine the best configuration of a system in face of state or context change. However, it takes a considerable amount of time to compute the most fitting system configuration, as a considerable number of possible configurations is explored.

2.3 Component Models

The component model in Darwin [10] specifies component structural view in terms of required and provided services (ports) and component interactions through bindings between provided and required ports. A composite component defines bindings among internal components ports as well as binding the composite component ports to the ones of internal components. Although, a FRACTAL component [3] is also based on the principle of provided and required interfaces, each component is involved in a *membrane* that provides external control interfaces to introspect and reconfigure the component internal details, and a *content* that consists in a set of sub-components. The membrane control interfaces normally correspond to several controller and interceptor objects.

The above component models mainly focus on providing means for structural adaptation. Modes [9] extend the Darwin component model with a representation of the expected interaction behaviour between required and provided services. Each of the identifiable component states is defined as a behaviour type that is characterised by an interaction process, constrains and properties. The interaction process is represented by Finite State Processes that define a set of scenarios in which the component can operate. This representation can be used to construct a Labelled Transition System, which can then be passed to the LTSA toolset to detect the presence of deadlocks and other properties analysis. On the other hand, the MOCAS model [2] only focuses on behavioural adaptation. Each component sets a UML state machine at runtime to characterise and realise its behaviour. This state machine consists in a set of states which are connected through transitions, each one being designated by an input signal, a guard (a boolean expression) and effects. A state also includes invariants, that together with guards designate business properties.

2.4 Self-Adaptation Framework

The Rainbow framework [7] relies upon Acme ADL [8] as a generic architectural model to manage a given system. Each component can be annotated with functional and non-functional properties, expected interactions with other components and specific architectural constraints. Furthermore, the framework uses the Stitch language [5] to express adaptation policies, which are triggered by reasoning over the architectural model. However, the adaptation actions are directly applied to the underlying software system.

The GRAF framework [6] proposes using a runtime abstract model between the adaptable software and the adaptation manager, where the adaptation manager does not directly control the adaptable software. The Runtime Model Man-

ager evaluates the pre-conditions of the adaptation policy before applying adaptation actions on the runtime model. Thereafter, the conducted modifications are validated using the policy post-conditions as well as the model invariants. If they do not conform with such constraints the Runtime Model Manager rollbacks the alterations performed on the Runtime Model.

Both frameworks rely upon a centralised representation of the system as well as centralised management, which can result in significant management complexity.

2.5 Summary

Centralised representation and management of a software system can become too complex that may overwhelm the benefits of having a self-managing system, while restricting the design of autonomous components. Dealing with adaptation concerning different levels of the software system at a centralised management may compromise system dependability. On the other hand, minimising the set of metrics of online monitoring and analysis reduces the complexity of system management while minimising power consumption. Furthermore, work on online reasoning on behavioural and runtime changes is insubstantial, limiting the trust in self-managing systems as well as their autonomy.

3 Model-based Self-Adaptive Component

We propose that each component has self-managing capabilities in order to reduce the complexity of specifying the underlying mechanisms for autonomous system management. A system is structured as a hierarchy of component compositions, *i.e.* single components can be used to construct a composite one which in turn can be used in other composite relationship. The resulting composite component is responsible for the management of its sub-components, though their internal details are managed by themselves. Each component defines the level of management details a parent component is allowed to control. The hierarchical structure provides means to handle adaptation concerns at different levels. The underlying mechanisms and models of self-adaptive components will be presented in the next sub-sections.

3.1 Runtime Model

Similar to the GRAF framework [6] we propose that a component comprises a Runtime Model which incorporates a structural and a behavioural view. The structural view consists in a ADL specification of its provided and required services, annotated with some properties, *e.g.* requirements and capabilities. For instance, a given service provides an average response time of 500ms; a service client requires a service provider with 10Mbps of available bandwidth for a file transfer service, etc. The behavioural view represents the behavioural model of its provided services and internal details. Moreover, the set of provided services

can be dynamically evolved in order to accommodate new functionality or replace existing one in face of a change in system requirements.

Furthermore, each component provides a list of relevant metrics to be monitored along with the dependencies among them. Such information can be exploited by the aforementioned techniques to reduce monitoring complexity when the component is behaving properly. In addition, based on the values of those metrics, utility functions can be specified to evaluate the system. The domain of each metric can be discretised in order to reduce the complexity of specifying utility functions, *e.g.* response time $\in [0, 100]ms \rightarrow$ `low`, $]100, 500]ms \rightarrow$ `medium`, $]500, \infty[\rightarrow$ `high`.

Finally, the Runtime Model includes a set of functional and non-functional requirements, structural and behavioural invariants and goals, that guide the decision-making service when choosing a suitable adaptation action.

3.2 Adaptive Monitoring

For each of the relevant metrics specified in the component's description a configuration determining the type of monitoring, interval or period based, and the correspondent parameters is specified. For period-based monitoring, the metric's value is periodically propagated based on a specified interval. When using interval-based monitoring, the metric value is propagated when it falls outside a specified numeric interval. Such parameters as well as the set of currently monitored metrics are dynamically adapted using reactive policies based on the monitored values and the dependencies among metrics to reduce power consumption. For example, when the number of clients exceeds a given threshold, increase the monitoring rate of system response time; stop monitoring available bandwidth when the system response time is below 100ms. Moreover, the monitoring component is also responsible for updating the aforementioned annotations on the structural model.

3.3 Decision-Making

Each component applies its adaptation actions relying upon the current view of its behavioural and structural models. Therefore, the monitoring component updates those two models instead of directly propagating metrics values to the decision-making service. Based on the Stitch language [5], we suggest adaptation actions to be specified using ECA policies, each one including an expected outcome of its adaptation actions in terms of structural and behavioural modifications as well as metric variations and an estimation of the cost of applying those actions.

By reasoning over the runtime behavioural model, the decision-making service verifies which ECA policies need to be activated. If two or more policies are triggered, the decision-making service applies an utility function to the expected outcome and the cost estimation; the one with the highest utility value is selected. However, the adaptation actions are firstly executed in the runtime

behaviour model in order for the decision-making service to verify if their execution violates components goals, invariants or requirements. If the simulation of the adaptation actions does not lead the runtime model to an inconsistent state, the modifications are applied in the component; otherwise the runtime model is rollbacked to the previous state before simulating the ECA policy execution. Moreover, after the execution of an ECA policy the expected outcome and cost estimation are updated using a suitable statistic method based on the metrics values captured by the monitoring service. Alternatively, statistical models can be used to predict the outcome of a given adaptation action, using collected values to improve prediction [4].

When a goal, an invariant, a functional or non-functional requirement is invalidated and there is not an ECA policy to fix the identified problem, a plan is generated using the estimation, evaluation or prediction of the outcome of actions in ECA policies. The Runtime Model is used to validate the generated plan, *i.e.* if the plan does not violate the aforesaid components requirements, goals and invariants. The planning algorithm can be parameterised to generate a plan as quick as possible or an optimal one using utility functions. Additionally, utility-functions can also be used to periodically improve system configuration parameters or structural configuration, *i.e.* sub-components replacement on a composite relationship. Since adaptation can imply a significant cost in terms of system availability, optimisation can be conducted only when the system utility is below a given threshold.

4 Final Remarks

Self-managing systems should be designed as a hierarchy of self-adaptive composite components to ease their specification and to provide means for management at different levels while allowing to reuse components and their managing features in similar systems. Online reasoning on structural and behavioural adaptation repercussions can improve the reliability of self-managing systems, as well as increase the confidence of systems administrators towards such systems.

In this paper we have presented a preliminary design to approach the aforementioned drawbacks of current self-managing systems. We intend to elaborate on each one of the presented services and combine them to design systems of self-managing components for pervasive scenarios. The main goal of deploying self-managing systems is to decrease managing costs by reducing human intervention. One one hand, such goal cannot be achieved if the underlying mechanisms of self-managing systems do not have the level of reliability so that system administrators can truly trust them to perform their job. On the other hand, self-managing systems do not completely replace the role of system administrators, *i.e.* leading to completely autonomous management, as the actions performed by self-managing systems are governed by high-level goals specified by system administrators. Consequently, system administrators still have the control of software systems, only delegating system management to the self-managing frameworks.

Acknowledgements

This work was supported by Fundação para a Ciência e Tecnologia under the grant SFRH/BD/73967/2010.

References

1. An Architectural Blueprint for Autonomic Computing. June 2006.
2. C. Ballagny, N. Hameurlain, and F. Barbier. Mocas: A state-based component model for self-adaptation. In *Proceedings of the 2009 Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, SASO '09, pages 206–215, Washington, DC, USA, 2009. IEEE Computer Society.
3. E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.-B. Stefani. The fractal component model and its support in java: Experiences with auto-adaptive and reconfigurable systems. *Softw. Pract. Exper.*, 36:1257–1284, September 2006.
4. O. Celiku, D. Garlan, and B. Schmerl. Augmenting architectural modeling to cope with uncertainty. In *Proceedings of the International Workshop on Living with Uncertainties (IWLU'07), co-located with the 22nd International Conference on Automated Software Engineering (ASE'07)*, 5 November 2007.
5. S.-W. Cheng, D. Garlan, and B. Schmerl. Stitch: A language for architecture-based self-adaptation, 2011. Submitted for Publication.
6. M. Derakhshanmanesh, M. Amoui, G. O'Grady, J. Ebert, and L. Tahvildari. Graf: graph-based runtime adaptation framework. In *Proceeding of the 6th international symposium on Software engineering for adaptive and self-managing systems*, SEAMS '11, pages 128–137, New York, NY, USA, 2011. ACM.
7. D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, 37:46–54, October 2004.
8. D. Garlan, R. T. Monroe, and D. Wile. Foundations of component-based systems. chapter Acme: architectural description of component-based systems, pages 47–67. Cambridge University Press, New York, NY, USA, 2000.
9. D. Hirsch, J. Kramer, J. Magee, and S. Uchitel. Modes for software architectures. In *of Lecture Notes in Computer Science*, pages 113–126. Springer, 2006.
10. J. Magee, N. Dulay, S. Eisenbach, and J. Kramer. Specifying distributed software architectures. In *Proceedings of the 5th European Software Engineering Conference*, pages 137–153, London, UK, 1995. Springer-Verlag.
11. M. A. Munawar, M. Jiang, T. Reidemeister, and P. A. S. Ward. Filtering system metrics for minimal correlation-based self-monitoring. In *Proceedings of the 2009 Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, SASO '09, pages 233–242, Washington, DC, USA, 2009. IEEE Computer Society.
12. M. A. Munawar and P. A. S. Ward. Leveraging many simple statistical models to adaptively monitor software systems. *Int. J. High Perform. Comput. Netw.*, 7:29–39, February 2011.
13. A. J. Ramirez, B. H. Cheng, P. K. McKinley, and B. E. Beckmann. Automatically generating adaptive logic to balance non-functional tradeoffs during reconfiguration. In *Proceeding of the 7th international conference on Autonomic computing*, ICAC '10, pages 225–234, New York, NY, USA, 2010. ACM.
14. M. Sloman. Policy driven management for distributed systems. *Journal of Network and Systems Management*, 2:333–360, 1994. 10.1007/BF02283186.

Safe, Flexible Recursive Types for Featherweight Java

Reuben N. S. Rowe

Imperial College London

Abstract. This paper presents a type assignment system with recursive types for Featherweight Java, inspired by the work of Nakano. Nakano's innovation consists in adding a modal type constructor which acts to control the folding of recursive types, resulting in a head-normalisation guarantee. We build on this approach by introducing a second modal type constructor which prevents the unfolding of types in contexts where doing so results in non-termination. Moreover our system inherits the flexibility of Nakano's approach, allowing object-oriented features (such as binary methods) to be typed in a safe and intuitive way. The work described in this paper is preliminary, and no formal results are claimed. However, we conjecture that our type system enjoys strong normalisation and we motivate this by working through some apposite examples.

1 Introduction

Recursive types can be viewed as finite representations of infinite (but *regular*) types [8, Chapter 20]. For example, the recursive type $T = \mu X.(A \to X)$ represents the infinite type satisfying the (recursive) equation $T = A \to T$. Alternatively, T can be understood to be the type obtained from 'unfolding' $\mu X.(A \to X)$ to $A \to (\mu X.(A \to X))$ an infinite number of times. The folded and unfolded form, denoting the same (infinite) type, are considered to be equivalent, and it is usual to freely exchange one for another during type assignment.

These types naturally capture the behaviour of entities which are (potentially) infinite, or structures of indeterminate size such as lists or streams. Among such entities are the *objects* of object-oriented (OO) programming systems. For example, consider objects which are instances of the following Java classes:

```
class C {
      C m() { return new C(); }
}

class Suc implements Nat {
      Nat pred;
      Nat add(Nat x) { return new Suc(this.pred.add(x)); }
}
```

Andrew V. Jones (Ed.): Proceedings of the 1st Imperial College Student Workshop (ICCSW '11), pp. 80–86. September 29th–30th 2011, London UK.

In the first example, instances of class `C` have a method `m` which returns another instance of `C`; thus given a `C` object, `m` may be safely invoked any arbitrary (and indeterminate) number of times. A natural (recursive) type describing this behaviour could be, for instance, $\mu X.\langle \texttt{m}{:}() \to X \rangle$. The second example gives a class containing a method `add`, representing addition on (positive) natural numbers. The `add` method creates a new `Suc` object and thus, as in the previous example, it may be invoked any arbitrary number of times.

Recursive types, then, provide an ideal mechanism for reasoning about object-oriented programs. Indeed, much work (see e.g. [3, 1, 4, 2]) has already been done on the type-theoretic relationship between recursive types and OO. The drawback to recursive types however is that in their unrestricted form they are *logically inconsistent* - that is to say, they allow for the typing of non-convergent (non-terminating) programs. This is not always a problem from a program analysis point-of-view, provided one is only interested in ensuring the *partial* correctness of programs, but poses problems when constructing type-based semantics, and also when reasoning about termination properties.

It is known that placing syntactic restrictions on recursive types – specifically, disallowing *negative* occurrences of recursively bound type variables (as in the type $\mu X.X \to A$) – restores convergence and logical consistency [6]. However, this approach poses a unique problem within the setting of OO. The `Suc` example illustrates another feature of object-oriented programming: *binary methods*. These are methods which take an argument of the same kind as the object containing it - in the case of our example, the `add` method (belonging to `Suc` objects) takes another `Suc` object as input (besides also returning one as output). This is exactly the behaviour captured by recursive types containing negative self references, and one might expect to be able to assign a type such as $\mu X.\langle \texttt{m}{:}X \to X \rangle$ to instances of `Suc`. Thus, such a restriction on types is unsatisfactory for object-oriented programming.

Nakano has developed a system of recursive types for the λ-calculus, which goes some way to addressing these issues [7]. In his system, there is no restriction on the (negative) occurrence of recursively bound type variables, and by introducing an additional type constructor $\bullet$, a convergent system (up to head-normalisation) is obtained. In previous work [9] the author studied semantics for object-oriented programming based upon intersection types. The current work is motivated by a failure of that work to provide fully decidable type inference in the presence of recursively defined classes, such as those given in the example above.

In this paper, we describe a variation on Nakano's theme which we believe is capable of providing a logically consistent and flexible foundation for OO type theory. We chose to first focus on a system without intersections for simplicity – such a system is easier to formulate and reason about (both formally and informally), and since type systems without intersections are simpy special cases of systems with intersections, the recursive- and intersection-based aspects of the system can be dealt with orthogonally. We point out that the work is at a preliminary stage, so we have no formal results to present. Rather, the aim

is simply to give evidence in support of our thesis in the form of typed (non) examples. Due to space restrictions, we are unable to provide a comprehensive explanation of the relevant background material, so we will assume that the reader is familiar with the basics of type theory and type assignment, and the author's previous cited work.

2 FJ∘μ: Safe Recursive Types for OO

In this section we describe how we apply Nakano's approach [7] to OO. The system we describe is a variation on our previous work [9], which in turn is based on Featherweight Java (FJ) [5], a formal model of the core operational semantics of Java. We refer the reader to those papers for details elided here.

Definition 1 (FJ∘μ Predicates). *The predicates (types) of FJ∘μ are formed according to the following grammar (where X, like φ, ranges over predicate variables):*

$$\sigma \quad ::= \quad C \quad | \quad \varphi \quad | \quad \bullet\sigma \quad | \quad \circ\sigma \quad | \quad \langle f{:}\sigma\rangle \quad | \quad \mu X.\langle m{:}(\overrightarrow{\sigma_n}) \to \sigma\rangle$$

with the restriction that any bound recursive predicate variables must be within the scope of either the $\bullet$ or $\circ$ type constructors (under their respective μ binders). The notation $\bullet^r\,\sigma$ ($\circ^r\,\sigma$) is a shorthand for σ preceded by r occurrences of $\bullet$ ($\circ$), and $\bullet^{r+}\,\sigma$ ($\circ^{r+}\,\sigma$) denotes the same thing, but indicates a strictly positive number of occurrences of $\bullet$ or $\circ$.

The basic idea is that method invocations are allowed by types of the form $\bullet^r\,\mu X.\langle m{:}(\vec{\sigma}) \to \sigma\rangle$, but *disallowed* at types of the form $\circ^{r+}\,\mu X.\langle m{:}(\vec{\sigma}) \to \sigma\rangle$. Thus the $\circ$ constructor serves to control the unfolding of recursive types, and can therefore be seen in some respect as the *dual*[1] of the $\bullet$ constructor in Nakano's system where it is used to control the *folding* of recursive types.

We define a *coercion* relation on predicates which permits $\bullet$ types to turn into $\circ$ types, and is used to determine when a method predicate can be safely assigned to a new object instance.

Definition 2 (Coercion). *The* coercion *relation $\lhd$ is the smallest preorder on predicates satisfying:*

$$\sigma \lhd \sigma' \Rightarrow \begin{cases} \bullet\,\sigma \lhd \bullet\,\sigma' \\ \bullet\,\sigma \lhd \circ\,\sigma' \\ \circ\,\sigma \lhd \circ\,\sigma' \end{cases}$$

The type assignment system for FJ∘μ is given by the rules in Figure 1. Γ is a typing environment for variables, and Σ is a typing environment for *classes* (also called a *self* environment), used to type `new` expressions and the self reference variable `this` within the bodies of methods. These class environments contain a

[1] Here we use this word in an informal sense, rather than its formal category-theoretic sense.

$$(\textsc{var}): \frac{}{\Sigma; \Gamma, x{:}\sigma \vdash x{:}\sigma}\ (x \neq \texttt{this}) \qquad (\bullet): \frac{\Sigma; \Gamma \vdash e{:}\sigma}{\Sigma; \Gamma \vdash e{:}\bullet\sigma} \qquad (\textsc{coerce}): \frac{\Sigma; \Gamma \vdash e{:}\sigma}{\Sigma; \Gamma \vdash e{:}\sigma'}\ (\sigma \lhd \sigma')$$

$$(\textsc{fld}): \frac{\Sigma; \Gamma \vdash e{:}\bullet^r\langle f{:}\sigma\rangle}{\Sigma; \Gamma \vdash e.f{:}\bullet^r\sigma} \qquad (\textsc{invk}): \frac{\Sigma; \Gamma \vdash e_0{:}\bullet^r \mu X.\langle m{:}(\overrightarrow{\sigma_n}) \to \sigma\rangle \quad \Sigma; \Gamma \vdash e_1{:}\bullet^r \sigma_1' \quad \ldots \quad \Sigma; \Gamma \vdash e_n{:}\bullet^r \sigma_n'}{\Sigma; \Gamma \vdash e_0.m(\overrightarrow{e_n}){:}\bullet^r(\sigma[\mu X.\langle m{:}(\overrightarrow{\sigma_n}) \to \sigma\rangle/X])\ (\sigma_i' = \sigma_i[\mu X.\langle m{:}(\overrightarrow{\sigma_n}) \to \sigma\rangle/X] \text{ for each } i \in \overline{n})}$$

$$(\textsc{self-obj}): \frac{}{\Sigma, \widehat{C}{:}(\overrightarrow{\sigma_n}) \to \sigma; \Gamma \vdash \texttt{this}{:}C} \qquad (\textsc{self-fld}): \frac{}{\Sigma, \widehat{C}{:}(\overrightarrow{\sigma_n}) \to \sigma; \Gamma \vdash \texttt{this}{:}\langle f_i{:}\sigma_i\rangle}\ (\mathcal{F}(C) = \overrightarrow{f_n},\ i \in \overline{n})$$

$$(\textsc{self-meth}): \frac{}{\Sigma, \widehat{C}{:}(\overrightarrow{\sigma_n}) \to \sigma; \Gamma \vdash \texttt{this}{:}\circ\sigma} \qquad (\textsc{inst-obj}): \frac{\Sigma; \Gamma \vdash e_1{:}\sigma_1 \quad \ldots \quad \Sigma; \Gamma \vdash e_n{:}\sigma_n}{\Sigma; \Gamma \vdash \texttt{new}\ C(\overrightarrow{e_n}){:}C}\ (\mathcal{F}(C) = \overrightarrow{f_n})$$

$$(\textsc{inst-fld}_1): \frac{\ldots \quad \Sigma; \Gamma \vdash e_i{:}\bullet^r\sigma \quad \ldots}{\Sigma; \Gamma \vdash \texttt{new}\ C(\overrightarrow{e_n}){:}\bullet^r\langle f_i{:}\sigma\rangle}\ (\mathcal{F}(C) = \overrightarrow{f_n},\ i \in \overline{n}) \qquad (\textsc{inst-fld}_2): \frac{\ldots \quad \Sigma; \Gamma \vdash e_i{:}\circ^r\sigma \quad \ldots}{\Sigma; \Gamma \vdash \texttt{new}\ C(\overrightarrow{e_n}){:}\circ^r\langle f_i{:}\sigma\rangle}\ (\mathcal{F}(C) = \overrightarrow{f_n},\ i \in \overline{n})$$

$$(\textsc{inst-rec}): \frac{\Sigma; \Gamma \vdash e_1{:}\circ^{r+}\sigma_1 \quad \ldots \quad \Sigma; \Gamma \vdash e_n{:}\circ^{r+}\sigma_n}{\Sigma; \Gamma \vdash \texttt{new}\ C(\overrightarrow{e_n}){:}\circ^{r+}\sigma}\ (C{:}(\overrightarrow{\sigma_n}) \to \sigma \in \Sigma)$$

$$(\textsc{inst-meth}_1): \frac{\overline{\Sigma}, \widehat{C}{:}(\overrightarrow{\sigma_n}) \to \mu X.\langle m{:}(\overrightarrow{\sigma'_{n'}}) \to \sigma'\rangle; x_1{:}\sigma_1'', \ldots, x_{n'}{:}\sigma_{n'}'' \vdash e_b{:}\sigma'' \quad \Sigma; \Gamma \vdash e_i{:}\bullet^r\sigma_i\ (\forall i \in \overline{n})}{\Sigma; \Gamma \vdash \texttt{new}\ C(\overrightarrow{e_n}){:}\bullet^r \mu X.\langle m{:}(\overrightarrow{\sigma'_{n'}}) \to \sigma'\rangle}\ (*)$$

$$(\textsc{inst-meth}_2): \frac{\overline{\Sigma}, \widehat{C}{:}(\overrightarrow{\sigma_n}) \to \mu X.\langle m{:}(\overrightarrow{\sigma'_{n'}}) \to \sigma'\rangle; x_1{:}\sigma_1'', \ldots, x_{n'}{:}\sigma_{n'}'' \vdash e_b{:}\sigma'' \quad \Sigma; \Gamma \vdash e_i{:}\circ^{r+}\sigma_i\ (\forall i \in \overline{n})}{\Sigma; \Gamma \vdash \texttt{new}\ C(\overrightarrow{e_n}){:}\circ^{r+} \mu X.\langle m{:}(\overrightarrow{\sigma'_{n'}}) \to \sigma'\rangle}\ (*)$$

$$* \quad (\mathcal{M}(C, m) = (\overrightarrow{x_{n'}}, e_b), \sigma'[\mu X.\langle m{:}(\overrightarrow{\sigma'_{n'}}) \to \sigma'\rangle/X] \lhd \sigma'', \sigma_i'' = \sigma_i'[\mu X.\langle m{:}(\overrightarrow{\sigma'_{n'}}) \to \sigma'\rangle/X] \text{ for each } i \in \overline{n'})$$

Fig. 1. Predicate Assignment for FJ$\circ\mu$

unique *marked* class, indicated by $\widehat{C}$ and used to keep track of which class the method body currently being typed appears in. The notation $\overline{\Sigma}$ represents the self environment identical to Σ, except that no class is marked. Valid environments may only contain a *single* type statement for each variable or class. The notation $\sigma_1[\sigma_2/X]$ stands for the type obtained from σ_1 by replacing all (free) occurrences of X with σ_2.

The key inference rules of the type system are the two (INST-METH) rules, which assign a (recursive) *method* predicate to a new object (instance). Their operation can be understood by viewing the `new` keyword as representing a function that constructs objects from class definitions. Since classes may themselves create new objects according to their own definition (i.e. call their own `new` function), these functions are recursively defined. Thus the rule takes the familiar form for typing a recursively defined term, in which the body of the term is typed using an environment where recursive calls must be typed with the *same* type as the body itself. There is a subtle twist however - since the type scheme for fixed point operators in $\lambda\bullet\mu$ is $(\bullet A \to A) \to A$, recursively created objects must now be typed not with $\mu X.\langle m{:}(\overrightarrow{\sigma_n}) \to \sigma\rangle$, but with a *bulleted* version of this type. Nakano's approach would suggest using a $\bullet$ type, however in our system this would permit recursive method invocations resulting in non-termination, and so

instead we use the type $\circ\,\mu X.\langle m{:}(\overrightarrow{\sigma_n}) \to \sigma\rangle$, preventing such invocations. This is enforced by the $\circ^{r+}$ in the (INST-REC) and (SELF-METH) rules.

Using this type system, the examples from the introduction can be given their expected types:

$$\dfrac{\dfrac{}{\widehat{\texttt{C}}{:}() \to \mu X.\langle \texttt{m}{:}() \to \bullet\, X\rangle \vdash \texttt{new C()}{:}\circ\,\mu X.\langle \texttt{m}{:}() \to \bullet\, X\rangle}\,(\textsc{inst-rec})}{\vdash \texttt{new C()}{:}\mu X.\langle \texttt{m}{:}() \to \bullet\, X\rangle}\,(\textsc{inst-meth}_1)$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\widehat{\texttt{Suc}}{:}(\sigma) \to \sigma; \texttt{x}{:}\bullet\,\sigma \vdash \texttt{this}{:}\langle \texttt{pred}{:}\sigma\rangle}\,(\textsc{self-fld})}{\widehat{\texttt{Suc}}{:}(\sigma) \to S; \texttt{x}{:}\bullet\,\sigma \vdash \texttt{this.pred}{:}\sigma}\,(\textsc{fld}) \quad \dfrac{}{\widehat{\texttt{Suc}}{:}(\sigma) \to \sigma; \texttt{x}{:}\bullet\,\sigma \vdash \texttt{x}{:}\bullet\,\sigma}\,(\textsc{var})}{\widehat{\texttt{Suc}}{:}(\sigma) \to \sigma; \texttt{x}{:}\bullet\,\sigma \vdash \texttt{this.pred.add(x)}{:}\bullet\,\sigma}\,(\textsc{invk})}{\widehat{\texttt{Suc}}{:}(\sigma) \to \sigma; \texttt{x}{:}\bullet\,\sigma \vdash \texttt{this.pred.add(x)}{:}\circ\,\sigma}\,(\textsc{coerce})}{\widehat{\texttt{Suc}}{:}(\sigma) \to \sigma; \texttt{x}{:}\bullet\,\sigma \vdash \texttt{new Suc(this.pred.add(x))}{:}\circ\,\sigma}\,(\textsc{inst-rec}) \quad \dfrac{}{\texttt{y}{:}\sigma \vdash \texttt{y}{:}\sigma}\,(\textsc{var})}{\texttt{y}{:}\sigma \vdash \texttt{new Suc(y)}{:}\sigma}\,(\textsc{inst-meth}_1)$$

where $\sigma = \mu X.\langle \texttt{add}{:}(\bullet\, X) \to \bullet\, X\rangle$.

It also prevents the typing of non-terminating programs. Consider the following classes (where the `App` interface declares the method `app`):

```
class D { D m() { return new D().m(); } }

class Y implements App {
      App app(App x) { return x.app(new Y().app(x)); }
}
```

The methods in both these classes make recursive calls leading to non-terminating behaviour: the expression `new D().m()` is *unsolvable*, as it reduces only to itself; and the method invocation `new Y().app(z)`, although it reduces in one step to a *head* normal form, has the infinite reduction sequence:

$$\begin{aligned}\texttt{new Y().app(z)} &\to \texttt{z.app(new Y().app(z))}\\ &\to \texttt{z.app(z.app(new Y().app(z)))} \to \ldots\end{aligned}$$

Both of these expressions are *untypable* in FJ$\circ\mu$, since the presence of the $\circ$ type constructor prevents the typing of the recursive method invocations which lead to the non-termination.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\widehat{\texttt{D}}{:}() \to \mu X.\langle \texttt{m}{:}() \to \varphi\rangle \vdash \texttt{new D()}{:}\circ\,\mu X.\langle \texttt{m}{:}() \to \varphi\rangle}\,(\textsc{inst-rec})}{\widehat{\texttt{D}}{:}() \to \mu X.\langle \texttt{m}{:}() \to \varphi\rangle \nvdash \texttt{new D().m()}}\,(\textsc{invk})}{\nvdash \texttt{new D()}{:}\mu X.\langle \texttt{m}{:}() \to \varphi\rangle}\,(\textsc{inst-meth}_1)}{\nvdash \texttt{new D().m()}}\,(\textsc{invk})$$

$$\dfrac{\dfrac{\dfrac{}{\widehat{\texttt{Y}}{:}() \to \sigma; \texttt{x}{:}\tau \vdash \texttt{x}{:}\tau}\,(\textsc{var}) \quad \dfrac{\dfrac{}{\widehat{\texttt{Y}}{:}() \to \sigma; \texttt{x}{:}\tau \vdash \texttt{new Y()}{:}\circ\sigma}\,(\textsc{inst-rec})}{\widehat{\texttt{Y}}{:}() \to \sigma; \texttt{x}{:}\tau \nvdash \texttt{new Y().app(x)}}\,(\textsc{invk})}{\dfrac{\widehat{\texttt{Y}}{:}() \to \sigma; \texttt{x}{:}\tau \nvdash \texttt{x.app(new Y().app(x))}}{\nvdash \texttt{new Y()}{:}\sigma}\,(\textsc{inst-meth}_1) \qquad \dfrac{}{\texttt{z}{:}\tau \vdash \texttt{z}{:}\tau}\,(\textsc{var})}\,(\textsc{invk})}{\texttt{z}{:}\tau \nvdash \texttt{new Y().app(z)}}\,(\textsc{invk})$$

$\sigma = \mu X.\langle \texttt{app}{:}(\mu Y.\langle \texttt{app}{:}(\bullet\bullet X) \to \bullet X\rangle) \to \bullet X\rangle$, $\tau = \mu Y.\langle \texttt{app}{:}(\bullet\bullet\sigma) \to \bullet\sigma\rangle$.

3 Conclusions

We have presented a type system for a variant of Featherweight Java which assigns recursive types to class-based object-oriented programs. It is inspired by previous work on head normalising recursive types for Lambda Calculus, and we give several examples which show that our type system (a) types (persistently) normalising terms with intuitive recursive types; and (b) does *not* type non-terminating programs. Our contribution consists in showing how Nakano's approach can be applied to OO and, more importantly, in its extension in the form of the second type constructor $\circ$. The latter in particular is a novel contribution of this paper. We conjecture that our system types only strongly normalising (i.e. terminating) terms, and proving this is an important task of future research.

A principal type for a term is a type from which all other types assignable to that term can be generated. If a type assignment system has the principal type property, then a principal type exists for all typeable terms. Such a property is the key component of any algorithm for deciding type assignment, and thus typeability. Our motivation in carrying out this research was to obtain a type system for Featherweight Java of the same flavour as our previous work, but with *finite* (and thus *decidable*) principal types for objects. We feel the system presented in this paper is a good candidate. Take our first example from the introduction: if the set of *principal* types for this program in FJ$\circ\mu$ is $\{\texttt{C}, \mu X.\langle \texttt{m}{:}() \to \bullet X\rangle\}$, then by 'unfolding' this set in a similar way to that described in the introduction (i.e. by replacing the recursive components, $\bullet X$, by other types in the set) we obtain the infinite set $\{\texttt{C}, \langle \texttt{m}{:}() \to \texttt{C}\rangle, \langle \texttt{m}{:}() \to \langle \texttt{m}{:}() \to \texttt{C}\rangle\rangle, \ldots\}$, which is the set of principal types for this example in our system without recursive types.

The next step of future research, after showing normalisation and (finite) principal typings for this system, will be to add *intersections* to the type system as in our previous work in order to build a fully abstract semantics based on our recursive types. It is our ultimate aim to then construct decidable type inference systems for this augmented type assignment resulting in expressive, powerful and practical type based analysis of class-based OO programs.

References

1. M. Abadi and L. Cardelli. *A Theory Of Objects*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996.
2. C. Anderson, P. Giannini, and S. Drossopoulou. Towards type inference for javascript. In *ECOOP*, pages 428–452, 2005.
3. K. Fisher, F. Honsell, and J. C. Mitchell. A Lambda Calculus of Objects and Method Specialization. *Nordic J. of Computing*, 1(1):3–37, 1994.
4. N. Glew. An Efficient Class and Object Encoding. In *Proceedings of OOPSLA00*, pages 311–324, 2000.
5. A. Igarashi, B. Pierce, and P. Wadler. Featherweight Java: A Minimal Core Calculus for Java and GJ. In *OOPSLA*, pages 132–146, 1999.
6. N.P. Mendler. Recursive Types and Type Constraints in Second Order Lambda Calculus. In *Proceedings of LICS'87*, pages 30–36. IEEE, 1987.
7. H. Nakano. A Modality for Recursion. In *LICS*, pages 255–266, 2000.
8. Benjamin C. Pierce. *Types and Programming Languages*. MIT Press, 2002.
9. R. Rowe and S. van Bakel. Approximation Semantics and Expressive Predicate Assignment for Object-Oriented Programming. In *TLCA*, pages 229–244, 2011.

Measuring minimal change in argument premise revision

Mark Snaith and Chris Reed

Argumentation Research Group, School of Computing, University of Dundee, Dundee, UK, DD1 4HN
marksnaith@computing.dundee.ac.uk

Abstract. The field of belief revision studies how information can be given up in the face of new, conflicting information, while argumentation provides methods through which conflict can be modelled and the resultant acceptability of arguments evaluated. Prominent theories of belief revision depend on the notion of minimal change, measured in terms of epistemic entrenchment, to determine what beliefs to give up. In this paper, we take an initial look at the effects of removing an argument from a system of structured argumentation, in terms of both argument construction and acceptability, and how these can be used in the determination of minimal change.

1 Introduction

If a software agent is forced to accept information that conflicts with information that it currently possesses, it may be forced into giving up the original information. Conflict is a key area in argumentation, with the highly influential work of Dung [3] abstracting the nature of arguments and attacks between them. Dung's theory has been built on and expanded since the seminal paper; one recent development has been to instantiate the abstract approach by providing the arguments with structure, through the application of strict and defeasible inference rules to a knowledge base [7].

The process of removing an argument in a Dung-style framework is relatively straightforward, due to arguments being represented as single, abstract entities with no consideration for structure. However, when the arguments are given structure, a greater degree of flexibility is provided, in that giving up an entire argument can be done by, for instance, giving up a single premise. But with this flexibility comes a problem — complex arguments will contain multiple premises: exactly what premise(s) should be given up in order to remove the argument?

The field of belief revision aims to answer a more general version of this question in terms of belief sets — when an agent is required to give up a belief, and faces a choice as to exactly which belief, how does it make the choice? One of the most influential theories in belief revision is the AGM theory, which provides a set of postulates that describe valid *revisions*, *contractions* and *expansions* of belief sets [1]. These three processes are additionally guided by the concept of

Andrew V. Jones (Ed.): Proceedings of the 1st Imperial College Student Workshop (ICCSW '11), pp. 87–93. September 29th–30th 2011, London UK.

minimal change, with "minimal" being measured in terms of epistemic entrenchment — those beliefs with the lowest degree of entrenchment are more willingly given up [4, 5].

Connections between argumentation and belief revision have recently found new momentum. The work of [8, 9, 6] on Argument Theory Change sees belief revision techniques employed to revise an argumentation system when a new argument is added, such that the argument becomes warranted. We wish to take a different approach to connecting argumentation and belief revision, by considering the application of belief revision techniques to the removal of arguments from a system of structured argumentation.

In this paper, we take an initial look at effects of removing an argument from an ASPIC$^+$ argumentation system, in terms of both argument construction and acceptability, and how these can be used the determination of minimal change.

The paper proceeds as follows: in section 2 we provide a brief introduction to the system of [7]; in section 3 we identify the effects of a change to argument premises and show how these can be used to realise an entrenchment ordering; in section 4 we provide an example to demonstrate the concepts that we presented and in section 5 we outline our conclusions and areas for possible future work.

2 Preliminaries

The ASPIC$^+$ framework [7] further developed the work of [2] and instantiates the abstract approach to argumentation in [3]. The basic notion of the framework is an argumentation system, $AS = \langle \mathcal{L}, ^-, \mathcal{R}, \leq \rangle$ where $\mathcal{L}$ is a logical language, $^-$ is a contrariness function from $\mathcal{L}$ to $2^{\mathcal{L}}$, $\mathcal{R} = \mathcal{R}_s \cup \mathcal{R}_d$ is a set of strict ($\mathcal{R}_s$) and defeasible ($\mathcal{R}_d$) inference rules such that $\mathcal{R}_s \cap \mathcal{R}_d = \emptyset$ and $\leq$ is a partial preorder on $\mathcal{R}_d$.

An argumentation system contains a knowledge base, $\langle \mathcal{K}, \leq' \rangle$ where $K \subseteq \mathcal{L}$ and $\leq'$ is a partial preorder on $\mathcal{K}/\mathcal{K}_n$. $\mathcal{K} = \mathcal{K}_n \cup \mathcal{K}_p \cup \mathcal{K}_a \cup \mathcal{K}_i$ where $\mathcal{K}_n$ is a set of (necessary) axioms, $\mathcal{K}_p$ is a set of ordinary premises, $\mathcal{K}_a$ is a set of assumptions and $\mathcal{K}_i$ is a set of issues.

From the knowledge base ($\mathcal{K}$) and rules ($\mathcal{R}$) arguments are constructed. For an argument A, $Prem(A)$ is a function that returns all premises in A; $Conc(A)$ is a function that returns the conclusion of A; $Sub(A)$ is a function that returns all sub-arguments of A; $DefRules(A)$ is a function that returns all defeasible rules in A; and $TopRule(A)$ is a function that returns the last inference rule used in A.

On the basis of these functions, A is:

1. p if $p \in \mathcal{K}$ with: $Prem(A) = \{p\}$, $Conc(A) = p$, $Sub(A) = p$, $DefRules(A) = \emptyset$,$TopRule(A) =$ undefined
2. $A_1, \cdots, A_n \rightarrow \psi$ if $A_1, \cdots, A_n$ are arguments such that there exists a strict rule $Conc(A_1), \cdots, Conc(A_n) \rightarrow \psi$ in R_s; $Prem(A) = Prem(A_1) \cup \cdots \cup Prem(A_n)$, $Conc(A) = \psi$, $Sub(A) = Sub(A_1) \cup \cdots \cup Sub(A_n) \cup \{A\}$, $DefRules(A) = DefRules(A_1) \cup \cdots \cup DefRules(A_n)$,$TopRule(A) = Conc(A_1), \cdots, Conc(A_n) \rightarrow \psi$

3. $A_1, \cdots, A_n \rightarrow \psi$ if $A_1, \cdots, A_n$ are arguments such that there exists a defeasible rule $Conc(A_1), \cdots, Conc(A_n) \Rightarrow \psi$ in R_s; $Prem(A) = Prem(A_1) \cup \cdots \cup Prem(A_n)$, $Conc(A) = \psi$, $Sub(A) = Sub(A_1) \cup \cdots \cup Sub(A_n) \cup \{A\}$, $DefRules(A) = DefRules(A_1) \cup \cdots \cup DefRules(A_n) \cup \{Conc(A_1), \cdots, Conc(A_n) \Rightarrow \psi\}$, $TopRule(A) = Conc(A_1), \cdots, Conc(A_n) \Rightarrow \psi$

An argument can be attacked in three ways: on a (non-axiom) premise (undermine), on a defeasible inference rule (undercut) or on a conclusion (rebuttal).

Given an argumentation system $\mathcal{AS}$ and a knowledge base $KB = \langle \mathcal{K}, \leq' \rangle$, an argumentation theory is $AT = \langle \mathcal{AS}, KB \leq \rangle$, where $\leq$ is an argument ordering on the set of all arguments that can be constructed from KB in $\mathcal{AS}$.

In this paper, we will use the following notations: $Args(\mathcal{AS})$ is the set of all arguments in $\mathcal{AS}$; when considering the acceptability of arguments in an argumentation theory, AT, on the basis of $\mathcal{AS}$ ($AT_{\mathcal{AS}}$), we will leave the semantics unspecified and instead refer to a (possibly empty or unit) set of S-extensions $E(AT_{\mathcal{AS}})$; $\mathcal{K}(\mathcal{AS})$ is the knowledge base in an argumentation system $\mathcal{AS}$ and $\mathcal{AS} \backslash D$ is an argumentation system such that $\mathcal{K}(\mathcal{AS} \backslash D) = \mathcal{K}(\mathcal{AS}) \backslash D$.

3 Measuring minimal change

In classic theories of belief revision, the process is guided by *minimal change*, which is measured not just in terms of the logical consequences of removing a belief, but by an *entrenchment ordering* placed on beliefs — those beliefs with a lower degree of entrenchment will be more willingly given up [4, 5]. Nevertheless, logical consequences play an important part in arriving at this ordering — intuitively, an agent will be less likely to give up a belief that is fundamental to a significant number of its other beliefs.

The process of removing an argument from a system of structured argumentation involves making some modification to the system such that the argument can no longer be constructed. One of these modifications is to remove elements from the knowledge base, such that at least one of the premises required to construct the argument are no longer present. In the same way that removing beliefs from a belief set can have an impact on other beliefs, removing elements from the knowledge base of an argumentation system can have an impact on other arguments, aside from the one that is actively being removed.

This impact, however, is not solely structural when using the ASPIC$^+$ framework. Being built on Dung's abstract theory, the framework evaluates the acceptability of arguments using various sceptical and credulous semantics. In broad terms, an argument is acceptable if it is not defeated by other arguments and an argument is not acceptable if it is. Arguments can defend other arguments by defeating defeaters (for instance, an argument A defends an argument C if A defeats B, which in turn defeats C).

Thus, we must consider at least three effects when removing premises from a knowledge base in order to remove an argument from an argumentation system — **Structural**: those previously acceptable arguments that can no longer be

constructed in the system; **acceptability loss**: those arguments that remain in the system, but have become unacceptable; and **acceptability gain**: those arguments that remain in the system and have gained acceptability. It is possible to formally define these effects, and we do so now in the form of three functions.

Our first function, the *argument drop function*, considers the structural effects on an argumentation system of removing a set of propositions, $D \subseteq \mathcal{K}$:

Definition 1. *The* ***argument drop function*** Δ_A *of* $D \subseteq \mathcal{K}$*:*

$\Delta_A \colon 2^{\mathcal{K}} \rightarrow 2^{Args}$,
$\Delta_A(D) = \{A \mid A \in \bigcup E(AT_{\mathcal{AS}}), A \notin \mathcal{AS}\backslash D\}$

Our next two possible changes relate to argument acceptability, with currently acceptable arguments losing their acceptability (but remaining in $\mathcal{AS}$ as defeated arguments) and currently unacceptable arguments gaining acceptability.

We define two functions to capture these changes; first, the *acceptability drop function*, which identifies all acceptable arguments in $\mathcal{AS}$ that, while still capable of being constructed in $\mathcal{AS}\backslash D$, are no longer acceptable:

Definition 2. *The* ***acceptability drop function****,* Δ_S *of* $D \subseteq \mathcal{K}$*:*

$\Delta_S \colon 2^{\mathcal{K}} \rightarrow 2^{Args}$,
$\Delta_S(D) = \{A \mid A \in \bigcup E(AT_{\mathcal{AS}}), A \notin \bigcup E(\mathcal{AS}\backslash D), A \in \mathcal{AS}\backslash D\}$

Secondly, the *acceptability gain function*, which identifies those arguments that are not acceptable in $\mathcal{AS}$, but are acceptable in $\mathcal{AS}\backslash D$:

Definition 3. *The* ***acceptability gain function*** Λ_S *of* $D \subseteq \mathcal{K}$*:*

$\Lambda_S \colon 2^{\mathcal{K}} \rightarrow 2^{Args}$,
$\Lambda_S(D) = \{A \mid A \notin \bigcup E(AT_{\mathcal{AS}}), A \in \bigcup E(\mathcal{AS}\backslash D)\}$

There is no "argument gain" function, because we assume an open world, and thus do not consider it possible for an argumentation system to gain arguments when removing an argument. We are already considering all arguments (acceptable or otherwise) and thus the removal of an argument cannot cause new arguments to be constructed (but can influence acceptability, as captured by the acceptability drop and gain functions).

The outputs of these three functions can now be used in realising an entrenchment ordering over $2^{\mathcal{K}}$. The different functions are measuring different effects of a change and to simply combine them would be to remove this context. We therefore keep the components separate by representing them as a vector, Υ, with Υ' being a numeric vector, with the sizes of the functions as its components:

$$\Upsilon(D) = \begin{pmatrix} \Delta_A(D) \\ \Delta_S(D) \\ \Lambda_S(D) \end{pmatrix} \qquad \Upsilon'(D) = \begin{pmatrix} \mid \Delta_A(D) \mid \\ \mid \Delta_S(D) \mid \\ \mid \Lambda_S(D) \mid \end{pmatrix}$$

We arrive at an entrenchment ordering over $2^{\mathcal{K}}$ by considering the size of Υ', computed using the standard formula for the length of a vector (the square root of the sum of the squares of the components). If for some $D_1 \subseteq \mathcal{K}$ and $D_2 \subseteq \mathcal{K}$, $\mid \Upsilon'(D_1) \mid< \mid \Upsilon'(D_2) \mid$, then we have an entrenchment ordering, $<_e$ where $D_1 <_e D_2$ (that is, the set D_2 is *more* entrenched than the set D_1).

4 Example

Consider an argumentation system $\mathcal{AS}$ with knowledge base $\mathcal{K} = \{p, q, t, v, x\}$ such that $t < q$ and $v < s$; defeasible rule set $\mathcal{R}_d = \{p, q \Rightarrow r; p \Rightarrow s; t \Rightarrow u; v \Rightarrow w\}$; and contrariness relations $q \in \overline{t}$, $s \in \overline{v}$ and $w \in \overline{x}$.

In addition to atomic arguments on the basis of $\mathcal{K}$, the following arguments can be constructed in $\mathcal{AS}$: $\langle \{p, q\}; p, q \Rightarrow r; r \rangle$, $\langle \{p\}; p \Rightarrow s; s \rangle$, $\langle \{t\}; t \Rightarrow u; u \rangle$, $\langle \{v\}; v \Rightarrow w; w \rangle$, and there exists only one complete extension in $AT_{\mathcal{AS}}$: $\{p, q, r, s, x\}$.

Assume that we must remove the argument for r. This can be done by removing one of two premises: p or q. Consider the outputs of the functions for each premise:

	Δ_A	Δ_S	Λ_S
p	$\{r, s\}$	$\{x\}$	$\{v, w\}$
q	$\{r\}$	$\{\}$	$\{t, u\}$

These yield the following vectors for p and q:

$$\Upsilon(\{p\}) = \begin{pmatrix} \{r, s\} \\ \{x\} \\ \{v, w\} \end{pmatrix} \qquad \Upsilon'(\{p\}) = \begin{pmatrix} 2 \\ 1 \\ 2 \end{pmatrix}$$

$$\Upsilon(\{q\}) = \begin{pmatrix} \{r\} \\ \{\} \\ \{t, u\} \end{pmatrix} \qquad \Upsilon'(\{q\}) = \begin{pmatrix} 1 \\ 0 \\ 2 \end{pmatrix}$$

By using the sizes of $\Upsilon'(\{p\})$ and $\Upsilon'(\{q\})$, we can determine the entrenchment ordering:

$$\mid \Upsilon'(\{p\}) \mid = \sqrt{2^2 + 1^2 + 2^2} = \sqrt{9}$$

$$\mid \Upsilon'(\{q\}) \mid = \sqrt{1^2 + 0^2 + 2^2} = \sqrt{5}$$

Since $\mid \Upsilon'(\{q\}) \mid< \mid \Upsilon'(\{p\}) \mid$, our entrenchment ordering is $\{q\} <_e \{p\}$; that is, the agent, when using structural and semantic considerations, would choose to remove q instead of p in order to remove the argument for r from $\mathcal{AS}$.

5 Conclusions & future work

We have in this paper explored the concept of minimal change when removing an argument from a system of structured argumentation. We identified that an argument can be removed by removing one or more of its premises, which in turn will have an effect on other arguments.

Other arguments can be affected in one of three ways: through their removal from the system (thanks to sharing premises with the originally removed argument); through losing acceptability (but remaining constructable in the system); or gaining acceptability (thanks to a defeater either being removed, or losing acceptability).

The work presented here is an initial step towards appreciating the effects of removing an argument from an argumentation system, and forms only a small part of a larger study into the connection between belief revision and argumentation. In future work, we aim to further refine our notion of "minimal change" by incorporating preferences between arguments, and exploring the role of acceptability semantics. In terms of preferences, we currently consider all arguments identified by the drop and gain functions to be of equal weight. However, ASPIC$^+$ incorporates a preference ordering over arguments, which intuitively should influence an agent's choice when deciding what argument to sacrifice in a revision process. Acceptability semantics are divided into two broad groups: sceptical and credulous. An argument that is sceptically accepted has gone through a more rigorous process in order to determine its acceptability, and thus could be considered more important to an agent than argument that is only credulously accepted.

Beyond measures of minimal change, it is also our intention to develop a set of postulates that describe valid expansions, contractions and revisions of an argumentation system, similar in principle to the AGM postulates of [1]. We envisage these postulates to capture not only the concepts described by the AGM postulates, but also features that are unique to systems of structured argumentation.

Acknowledgements

This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) of the UK government under grant number EP/G060347/1.

References

[1] C.E. Alchourrón, P. Gärdenfors, and D. Makinson. On the logic of theory change: Partial meet contraction and revision functions. *The Journal of Symbolic Logic*, 50:2:510–530, 1985.

[2] L. Amgoud, L. Bodenstaff, M. Caminada, P. McBurney, S. Parsons, H. Prakken, J. van Veenen, and G.A.W. Vreeswijk. Final review and report on formal argumentation system. Deliverable D2.6, ASPIC IST-FP6-002307, 2006.

[3] P. M. Dung. On the acceptability of arguments and its fundemental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77:321–357, 1995.
[4] P. Gärdenfors. *Knowledge in Flux*. MIT Press, 1988.
[5] P. Gärdenfors. Belief revision: An introduction. In Peter Gärdenfors, editor, *Belief Revision*. Cambridge University Press, 1992.
[6] M.O. Moguillansky, N.D. Rotstein, M.A. Falappa, A.J. Garcia, and G.R. Simari. Argument theory change through defeater activation. In *Proceedings of the 3rd International Conference on Computational Models of Argument (COMMA 2010)*, 2010.
[7] H. Prakken. An abstract framework for argumentation with structured arguments. *Argument and Computation*, 1:2:93–124, 2010.
[8] N.D. Rotstein, M.O. Moguillansky, M.A. Falappa, A.J. Garcia, and G.R. Simari. Argument theory change: Revision upon warrant. In *Proceedings of the 2nd International Conference on Computational Models of Argument (COMMA 2008)*, 2008.
[9] N.D. Rotstein, M.O. Moguillansky, A.J. Garcia, and G.R. Simari. A dynamic argumentation framework. In *Proceedings of the third international conference on Computational Models of Argument (COMMA 2010)*, 2010.

Applying Algebraic Specifications on Digital Right Management Systems

Nikolaos Triantafyllou[1], Katerina Ksystra[1], Petros Stefaneas[2] and Panayiotis Frangos[1]

[1] School of Electrical and Computer Engineering, National Technical University of Athens, Heroon Polytechniou 9, 15780 Zografou, Athens, Greece
{nitriant, katksy, pfrangos}@central.ntua.gr
[2] School of Applied Mathematical and Physical Sciences, National Technical University of Athens, Heroon Polytechniou 9, 15780 Zografou, Athens, Greece
petros@math.ntua.gr

Abstract. Digital Right Management (DRM) Systems have been created to meet the need for digital content protection and distribution. In this survey paper we present some of the directions of our ongoing research on the applications of the algebraic specification techniques on mobile DRM systems.

Keywords: DRM, Right Expression Languages, CafeOBJ, Institutions

1 Introduction

Digital Rights Management systems (DRMs) control many aspects of the life cycle of digital contents including consumption, management and distribution. Key component of such a system is the language in which the permissions on contents and constraints are expressed. Such languages are called Right Expression Languages (RELs). In this survey paper we present some of our ongoing research directions aiming to address some of the open problems of the DRM systems [1][2], by using algebraic specifications. Our research has been focused on Open Mobile Alliance [3], a well-known DRM standard.

Our paper is organized as follows: Section 2 gives a brief introduction to the concepts needed. Section 3 gives the outline of an abstract syntax and its specification for OMA REL [4]. OMA presents an algorithm that deals with multiple licenses referring to the same content. In section 4 we refer to the formal specification of this algorithm in the algebraic specification language CafeOBJ, and to the formal verification of a safety property. This algorithm is not the optimal to use as it explained in [2]. In section 5 we suggest a redesign of this algorithm based on Order Sorted Algebra [5] and hint at a formal proof that this algorithm is correct using the methodology presented in [6]. Finally we present some of our future goals.

Andrew V. Jones (Ed.): Proceedings of the 1st Imperial College Student Workshop (ICCSW '11), pp. 94–100. September 29th–30th 2011, London UK.

2 Prerequisites

2.1 Order Sorted Algebra

An Order Sorted Algebra (OSA) is a partial ordering $\leq$ on a set of sorts [5], where by sorts we usually mean a set of names for data types. This subsort relation imposes a restriction on an S-sorted algebra A, by s-sorted algebra we mean a mapping between the sort names and sub sets from the set A called the carries of sort s, that if $s \leq s'$ then $A_s \subseteq A_{s'}$ where A_s denotes the elements of sort s in A. Order sorted algebra (OSA) provides a way for several forms of polymorphism and overloading, error definition, detection and recovery, multiple inheritance, selectors when there are multiple constructors, retracts, partial operations made total on equationally defined sub-sorts, an operational semantics that executes equations as left-to-right rewrite rules and many more applications [5].

2.2 Observation Transition Systems, CafeOBJ, Specification and Verification

An Observation Transition System (OTS) is a transition system that can be written in terms of equations. We assume there exists a universal state space, say Y. Formally, an OTS S is a triplet $S = <O, I, T>$ where I is a subset of Y, the set of initial states of the machine and O is a set of observation operators. Each observer in O is a function that takes a state of the system and possibly a series of other data type values (visible sorts) and returns a value of a data type that is characteristic to that state of the system. Finally, T is the set of transition (or action) conditional functions. Each transition takes as input a state of the system and again possibly a series of data-type values and returns a new state of the system.

CafeOBJ [8] is an executable algebraic specification language, implementing equational logic by rewriting. Equations are treated as left to right rewrite rules. It can also be used as a powerful interactive theorem prover with the proof scores method [11]. With CafeOBJ each module defines a sort. A visible sort is the specification of an abstract data type. Hidden sorts are used to specify state machines. Sort ordering is simply declared using <. Concerning hidden sorts there are two kinds of operators; *action operators*, which change the state of a machine, and *observation operators*, which observe (and return) a specific value in a particular state of the machine. Equations are denoted using the keyword *eq* and conditional equations using the keyword *ceq*. Finally modules can be imported to other modules by either protecting them or extending them. An OTS can be specified in CafeOBJ, in a natural way. The state space corresponds to the values of a hidden sort. The initial states are denoted by a set of constants of the hidden sort. Observation operators are denoted as observers and transitions as action operators.

After creating the specification of a system in the OTS/CafeOBJ approach it is possible to verify that it holds several kinds of properties such as invariant and liveness. The former consists of properties that hold in any reachable state of the system and is the most explored in the bibliography. The latter consists of properties expressing that something will eventually happen in the system. There are few applications of this methodology in CafeOBJ to our knowledge. Finally we should

mention that it is possible to conduct falsification in this approach, meaning that the CafeOBJ system can guide you to find a counter example of the property you desired to verify.

Here we discuss the procedure of verifying invariant properties, liveness properties are discussed further in section 5. To verify an invariant property you first need to express it as a predicate in CafeOBJ terms. Next, show that this predicate holds in any initial state. This is done by asking CafeOBJ to reduce the predicate term in an arbitrary initial state. Then show that the property holds for any transition, the inductive step. Assuming that the predicate holds for an arbitrary state we ask CafeOBJ to reduce whether this implies that it holds for its successor state. The successor state is obtained by applying the transition rules to the above arbitrary state. CafeOBJ will either return true, false or an expression. If it returns true then the predicate holds on that step. When an expression is returned, this means that the machine cannot continue with the reductions. We must then assist CafeOBJ by case splitting the transition providing additional equations. If false is returned then we might need to find a lemma to discard this case, showing that it is not. If however, the state is reachable then the property does not hold and we have a counterexample.

3 Formal Semantics for OMA REL

OMA REL [4] is an XML based language. The part of the language that is responsible for the expression of rights is called the agreement model. Inside this model the constraints and permission of the language are defined. We have given algebraic semantics to the OMA REL component dealing with expressing the permissions and constraints on the contents. To achieve this, we first created an abstract syntax for the language. Then we translated this syntax to the CafeOBJ specification language in order to use its rewriting as a tool for validation.

A longer version of this part of our paper appeared in the proceedings of WiMob 2009 where we proposed the abstract syntax, its specification and some case studies [9]. Here we will just present one example. Assume that Alice has purchased the following license: *Display content named contentID1 as many times as you like, and Display or Print the content named contentID2 as many times as you like*. Having specified the above abstract syntax as rewriting rules in CafeOBJ we can validate sets of licenses. The first step is to specify in a script the license of interest. In our specification this is done by declaring the permission set as; `eq ps1=add (True==>contentID2 print, add(True==> contentID2 display, add(True ==> contentID1 display, em-permset))).`

The `add` operator adds a permission element to a set of permissions. A permission element is a triplet; *constraint* on *content* allows *action.* `em-permset` is a constant denoting an empty permission set. After the permission sets and licenses are created, it is easy to perform e-validation by simply asking the CafeOBJ compiler if the desired permission belongs to the permissions allowed by this license, using the following reduction `red Permitted(print,ebook, contentID2) in permissionSET` . Where `red` is a CafeOBJ command for term rewriting the given

expression and `permissionSET` is an operator denoting the above license, which contains `ps1`.

4 Verifying the OMA Rights Choice Algorithm

We study here the algorithm that comes together with the specification of the OMA REL and is responsible for choosing the most appropriate license to use, when there exist multiple licenses referring to a specific content. This algorithm has been formally specified and in addition, it has been proved to satisfy a minimal safety property in [11]. A longer version of this specification and verification appeared in the proceedings of WINSYS 2010 [10].

The property we proved can be seen in table 1. On the left column L and S are variables representing an arbitrary license and an arbitrary state of the system respectively. bestLic(S) is an observer that returns the best license to use in S and valid(S,L) an operator that checks if licenses L constraints hold in S. The proof of such properties follows the methodology presented in section 2.2. It required four extra lemmas: two were used to discard unreachable states of the OTS and the other two where lemmas on data-types that helped CafeOBJ with the reductions on these visible sorts.

5 Proposing a New Algorithm and its Verification

There exist some cases where we end up losing execution rights by using the algorithm currently in use [2]. Indeed, let us consider the set of licenses seen on table 2. If the user decides to use his right "*listen to song A*", using the above algorithm the DRM agent will choose License 1. But by doing so, License 1 will become depleted since it contains the count constraint denoted by "once". This results in the user losing the right to ever listen to song B with this set of licenses. This would not occur if the agent had decided to use License 2 to execute the right to listen to song A.

This loss has been characterized by monotonicity of licenses in [2] and is proven that any algorithm attempting to solve this problem as is, will be NP-complete. Our approach is based on Order Sorted Algebra [5]. We point out that licenses, as data types, can be represented by ordered sorts [12]. Next we identified that this loss of rights can only occur in some special cases.

Safety property for OMA Rights Choice Algorithm euationally and informally	
eq inv1(S,L) = ((L=bestLic(S)) and not (L= nil)) implies valid(S,L) .	*When a license is chosen, then the license is valid at that specific time.*

Table 1. Minimal Safety property to verify in the original OMA Rights Choice Algorithm, in natural language and CafeOBJ equational notation

Installed Licenses on a DRM agent	
License1: "you may listen to songs A or B once before the end of the month".	*License2: "you may listen to songs A or D ten times."*

Table 2. A set of installed license that can cause a loss of rights

Liveness Property for the OMA Rights Choice Algorithm	
eq lto(S, P) = ((color(S,P) = white) ∧ (P /in allowed(S))) |--> (color(S, P) = black) .	*If a right belongs to the installed licenses and is colored white leads to it being colored black.*

Table 3. Liveness property describing the no loss of rights in CafeOBJ notation and natural language

To capture this we inserted *Labels* on licenses that denote the following three things; Firstly, if the license contains one or more permissions, secondly the dominant constraint based on the original algorithm and finally if the license only allows one more execution. These labels allow us to provide an ordering on licenses that is used to determine what license to choose so that no loss will occur, while respecting the ordering on constraints in the original algorithm. A longer version of this paper, which contains the full algorithm together with case studies and Java implementation, can be found in [12].

5.1 Verification of the New Algorithm

We have proved that our new algorithm does not cause the same loss of rights as the algorithm currently in use. The full proof will be presented elsewhere. In this section we will only sketch our proof. The proving procedure has been broken down into the following steps. First we created a specification of our algorithm as an OTS in CafeOBJ. Next we constructed an OTS, modeling the behavior of installed licenses on a DRM agent, meaning how they evolve when the user executes rights. The two OTSs where composed behaviorally as described in [13] yielding a new OTS. In to order describe and prove the desired property we added to the OTS a coloring on rights via an observer. Initially all rights are white (unused). A right is colored black (used) in two cases. Firstly, if the right corresponds to user request and the algorithm chooses the license containing this right as the optimal. Secondly, a right, say B, should be colored black if the user makes a request, say A different then B, but A only belongs to the license that contains B and that license becomes depleted after the execution of the request A.

At the property describing the no loss of rights condition (table 3), `S,P` are variables denoting an arbitrary state and a permission respectively. `color(S,P)` is an observer that returns the color of permission `P` in state `S` and `|-->` is an operator we used to denotes *leads- to*. The deduction rules for `|-->, ensure` and `unless` are provided in a separate module called OTSLogic. This is a *Liveness* property and particularly a *leads-to* property [6]. The proof followed the methodology of [6]. The *lead-to* predicate was broken down into two *ensure* predicates of the form *p ensure q*, with p and q predicates. These types of properties require proving the "*unless case*; *p unless q"* and the "*eventually case*; *p eventually q"*. For the first we

need to prove that all of the transitions preserve the predicate; *(p(s) and* $\Box$*q(s))* $\Box$ *(p(s') or q(s'))*. While for the second we need to show that there exists an instance of a transition where; *(p(s) and* $\Box$*q(s))* $\Box$*q(s')* holds. Where *s* a state of the OTS and *s'* is derived from *s* by applying a transition rule.

6 Conclusions

We have presented some of our ongoing work on the applications of algebraic specifications to mobile DRM systems. Also, we have shown how various techniques, from rewriting to theorem proving, can help solve some of the open problems of the field and also provide insights that can lead to the development of novel applications to DRMs as already shown with the proposed algorithm.

One of the main concerns with DRM is interoperability. There exist many different REL and DRM systems that cannot work together, so at the moment it is usually not possible to transfer licenses from one environment (mobile) to another (media player).We have started to address this problem by defining an Institution for OMA REL ([7]). Using the well-known abstract model theory tools of Institutions we intend to create a mechanism for translating licenses from one system to another via Institution morphisms in such a way so the meaning of the license is preserved by using semantic techniques.

References

1. Pucella, R., Weissman, V.: A Logic for Reasoning about Digital Rights. In: Proc. 15th IEEE Workshop on Computer Security Foundations (CSFW '02), pp. 282 (2002)
2. Barth A., Mitchell J. C., Managing digital rights using linear logic, 21st Symposium on Logic in Computer Science, pp. 127 – 136, 2006
3. Open Mobile Alliance Digital Rights Management Version 2.1, approved in 2008.
4. Open Mobile Alliance, DRM Rights Expression Language, approved version 2.1, 2010.
5. Goguen, J,A., Meseguer J.: Order-sorted algebra I: equational deduction for multiple inheritance, overloading, exceptions and partial operations, Theoretical Computer Science Volume 105, Issue 2, pp. 217-273 (1992)
6. Ogata, K., Futatsugi, K.: Proof Score Approach to Verification of Liveness Properties. IEICE Transactions on Information and Systems, Volume E91.D, Issue 12, pp. 2804-2817 (2008).
7. Goguen, J,A., Burstall, R., M.: Institutions: abstract model theory for specification and programming
8. Diaconescu, R., Futatsugi, K.: CafeOBJ Report: the Language, Proof Techniques, and Methodologies for Object-Oriented Algebraic Specification. AMAST Series in Computing, Vol. 6, World Scientific, Singapore, 1998
9. Triantafyllou, N., Ouranos I., Stefaneas, P.: Algebraic Specifications for OMA REL Licenses. IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), pp. 376-381 (2009)
10. Tiantafyllou, N., Ouranos, I., Stefaneas, P., Frangos, P.: Formal Specification and Verification of the OMA License Choice Algorithm in the OTS/CafeOBJ Method. Wireless Information Networks and Systems (WINSYS) 2010, pp. 173-180 (2010)

11. Futatsugi, K., Ogata, K.: Proof Scores in the OTS/CafeOBJ Method. In: Najm, E., Nestmann, U., Stevens, P. (eds.) Formal Methods for Open Object-Based Distributed Systems. LNCS, vol. 2884, pp. 170--184, Springer, Berlin, (2003)
12. Tiantafyllou, N., Ouranos, I., Stefaneas, Ksystra, K., P., Frangos, P.: Redesigning OMA Choice Algorithm, submitted for publication - available at CoRR abs/1102.1547: (2011)
13. Iida, S., Futatsugi, K., Diaconescu, R.: Component-based algebraic specification: behavioural specification for component-based software engineering. Seventh OOPSLA Workshop on Behavioral Semantics of OO Business and System Specification (1999)

Reduction of Variability in Split–Merge Systems

Iryna Tsimashenka and William Knottenbelt

Imperial College London, 180 Queen's Gate,
London SW7 2AZ, United Kingdom,
Email: {it09,wjk}@doc.ic.ac.uk

Abstract. We consider an optimisation problem applicable to systems that can be represented as split–merge queueing networks with a limited buffer space for processed subtasks. We assume Poisson arrivals and generally distributed service times. The proposition is to reduce variability in terms of the difference in the times of arrival of the first and last subtasks in systems where the release times of the subtasks can be controlled. This stands in contrast to the overwhelming majority of research which is focused on reduction of mean response time or percentiles of response time. We formally define our notion of variability in split–merge systems and construct an associated cost function and optimisation problem. For two case studies we use simulation to explore the optimisation landscape and to solve the associated optimisation problem.

1 Introduction

Performance analysis has acquired increased importance due to the growing complexity of automated systems. Performance modelling enables an understanding of the relationships between system workload, control parameters and key metrics such as customer response time, system utilisation and buffer occupancy. For systems that involve the flow and processing of customers and resources, queueing models are an appropriate formalism. Optimisation of control parameters allows to minimise, for example, mean response time within given constraints [2].

It has been observed in a recent paper related to the scheduling of Map-Reduce jobs in clusters that delayed scheduling of jobs can counterintuitively lead to greater fairness and a higher level of data locality [8]. In another research, delay scheduling was applied in the context of Quality of Service in networks [6]. Specifically, adding delays to input packets results in shaping the traffic such that packet interarrival times follow an exponential distribution. This construction permits the analysis and optimisation of the network with mathematically tractable Markovian models.

In this paper we show that adding judiciously chosen deterministic delays to subtask processing in split–merge systems can result in a reduction of variability in terms of time difference between the completion of the first and last subtasks in a job. At the same time, corresponding beneficial effects on output buffer occupancy are observed.

A major application area of our approach is automated warehouse systems [7], where partially completed subtasks need to be held in a physical buffer

Andrew V. Jones (Ed.): Proceedings of the 1st Imperial College Student Workshop (ICCSW '11), pp. 101–107. September 29th–30th 2011, London UK.

space. Another application field of this technique is parallel computing where it is sometimes desirable to minimise mean synchronisation time between tasks [5]. In healthcare systems, we can minimise the time patients wait for results following treatment [1]. Lastly, in project scheduling we can reduce mean slack time [9].

The remainder of this paper is organised as follows. Section 2 presents background material relating to split-merge systems. Section 3 presents a formal definition of variability, an associated cost function, and a simple simulation-based optimisation methodology. Section 4 illustrates the application of the methodology in the context of two case studies. Section 5 concludes and considers avenues for future work.

2 Background

As shown in Fig. 1, a split–merge system consists of a queue of waiting tasks (assumed to arrive according to a Poisson process with mean rate λ), a *split* point at which tasks split into subtasks, several (potentially) heterogeneous servers (assumed to process subtasks according to a general service time distribution $F_i(t)$ with mean service time $1/\mu_i$), a buffer for completed subtasks (the merge buffer) and a *merge* point.

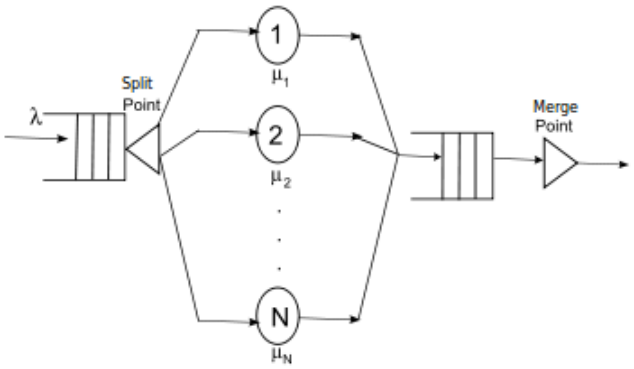

Fig. 1. Split–Merge queueing model.

When all subtask servers are idle and the task queue is not empty, a task is taken from the head of the task queue. This task splits into N subtasks at the *split* point. Each subtask server then processes its allocated subtask. Outgoing subtasks join the merge buffer. When all subtasks belonging a task are present in the merge buffer, the task exits the system via the merge point.

3 Variability in Split–Merge systems

We define the variability of a split–merge system as the mean difference in time between the arrival of the first and last subtasks (belonging to each task) in the merge buffer. Our challenge is to control this variability via the introduction of a vector of delays:

$$\mathbf{d} = (d_1, d_2, ..., d_i, ..., d_{n-1}, d_n) \tag{1}$$

Here element d_i of the vector represents the deterministic delay that will be applied before a subtask is sent to server i for processing.

We further define the *cost function* of a split-merge system for a given delay vector $\mathbf{d}$ as:

$$C(\mathbf{d}) = E(X) - E(Y) \tag{2}$$

where X is the random variable denoting the maximum completion time across all subtasks (arising from a particular task), and Y is the random variable denoting the minimum completion time across all subtasks.

Assuming that subtasks at server i are served independently with service time sampled from a distribution function $F_i(t)$, then, taking into account the delay that is applied before each subtask begins processing, X will have cumulative distribution function:

$$F_X(t) \sim \prod_{i=1}^{n} F_i(t - d_i) \tag{3}$$

Here it is assumed, for all i, that $F_i(t - d_i) = 0$ for all $t < d_i$. Similarly, Y has cumulative distribution function:

$$F_Y(t) \sim \overline{\prod_{i=1}^{n} \overline{F_i(t - d_i)}} \tag{4}$$

For a given split-merge system, our challenge is to find that vector $\mathbf{d}$ which minimises $C(\mathbf{d})$. To constrain the solution space while avoiding unnecessary delays to overall mean task processing time, we set $d_i = 0$ for the subtask server(s) with the largest mean service time. We will denote the resulting *vector of optimal delays* as:

$$\widetilde{\mathbf{d}} = (\widetilde{d_1}, \widetilde{d_2}, ..., \widetilde{d_{i-1}}, 0, \widetilde{d_{i+1}}, ..., \widetilde{d_{n-1}}, \widetilde{d_n}) \tag{5}$$

We note that minimising C results in minimum merge buffer utilisation in the split–merge system. This property is particularly relevant in physical systems (e.g. warehouses of major online retailers), which are often constrained in terms of the amount of physical output buffer space available.

Although it is our ultimate goal to establish an efficient analytical procedure for determining $\widetilde{\mathbf{d}}$, in the present paper we apply a simple simulation-based methodology – based on extensions to the JINQS queueing network simulation package [3] – to explore the shape of the cost function landscape and hence to find (near-)optimal solutions for $\widetilde{\mathbf{d}}$.

4 Numerical Results

Case Study 1

Consider a split–merge system with 3 service nodes having the following service time distributions: Uniform(2,3), Pareto(3,1) and Det(5). The latter has the highest expectation, so its optimal delay is set to 0 in $\widetilde{\mathbf{d}}$ from Eq. 5. By the simulation-based algorithm outlined at the end of the previous section, we find the *vector of optimal delays* to be:

$$\widetilde{\mathbf{d}} = (2.5317, 3.7154, 0.0) \tag{6}$$

Fig. 2 displays the CDFs of the service times of the servers before and after application of the optimal delays. Fig. 3 shows how $C(\mathbf{d})$ depends on *delay1* added to the Uniform distribution and *delay2* added to the Pareto distribution.

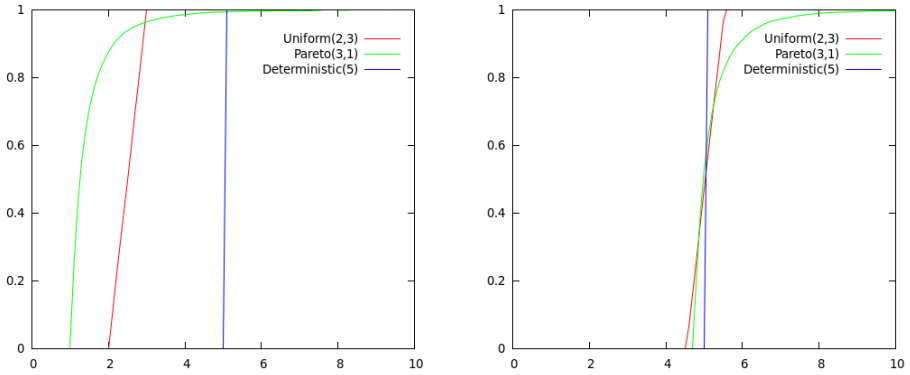

Fig. 2. CDFs of server service times before and after adding the optimal delays.

Case Study 2

Similarly, we show results for optimal delays in a split–merge system with service time distribution functions: Det(5), Erlang(6,1/3), Exp(2). Here the *vector of optimal delays* is as follows:

$$\widetilde{\mathbf{d}} = (11.9386, 0.0, 16.5880) \tag{7}$$

Fig. 4 displays the CDFs of the service times of the servers before and after application of the optimal delays.

Fig. 5 shows how $C(\mathbf{d})$ depends on *delay1* added to the Deterministic distribution and *delay2* added to the Exponential distribution.

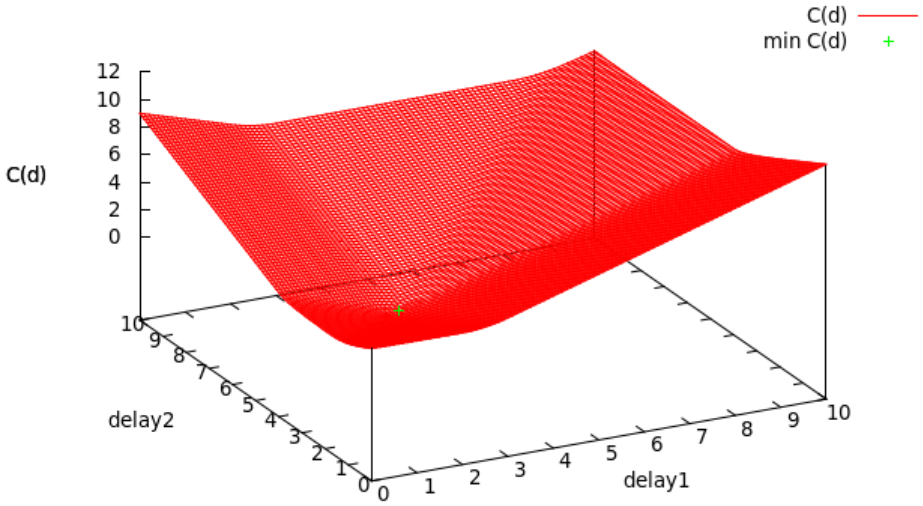

Fig. 3. Surface plot of cost function against delays.

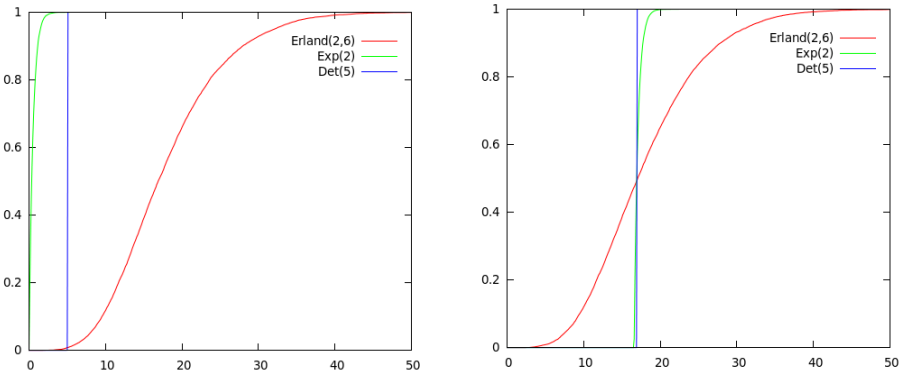

Fig. 4. CDFs of server service times before and after adding the optimal delays.

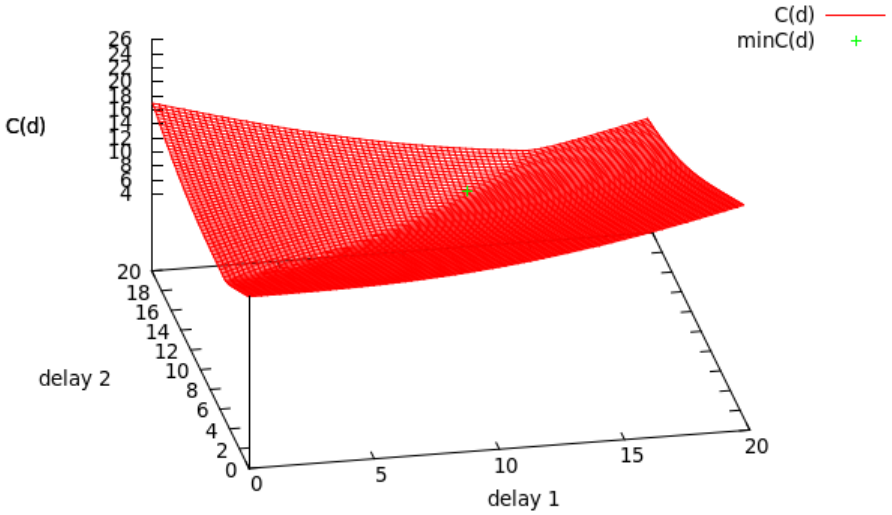

Fig. 5. Surface plot of cost function against delays.

In both cases we note the optimal delays take on different values to those one might intuitively expect (e.g. by subtracting the mean response time of each server from the maximum mean service time).

5 Conclusions and Future Work

This paper has considered the problem of introducing delays into the processing of subtasks in split-merge systems in order to reduce the variability of overall task processing time (and thereby merge buffer occupancy). We have illustrated the use of a simple simulation-based methodology for finding optimal delays in the context of two case studies.

There are several directions in which this work can be extended. Firstly, concurrent parallel generation of the cost function landscape across several computers – with a good corresponding speedup – should be simple to achieve given the embarrassingly parallel nature of the problem. Secondly, it would be interesting to explore if it is possible to find an efficient analytical procedure for determining the vector of optimal delays. It may be that it is necessary to restrict the form of service time distribution functions that can be supported (e.g. requiring that they be continuous). Finally, we intend to investigate the analogous optimisation of a type of less synchronised parallel processing system, namely fork-join systems [4]. The latter have ready application to the modelling of RAID and other computing systems.

References

1. S. W. M. Au-Yeung. *Response Times in Healthcare Systems.* PhD thesis, Imperial College London, January 2008.
2. G. Bolch. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications.* John Wiley & Sons, Inc., 2006.
3. A. J. Field. *JINQS: An Extensible Library for Simulating Multiclass Queueing Networks.* Imperial College London, August 2006.
4. P. G. Harrison and S. Zertal. Queueing Models with Maxima of Service Times. In *Computer Performance Evaluations, Modelling Techniques and Tools. 13th International Conference, TOOLS 2003, Urbana, IL, USA, September 2-5, 2003*, volume 2794, pages 152–168, September 2003.
5. J. C. S. Lui, R. R. Muntz, and D. Towsley. Computing performance bounds of fork-join parallel programs under a multiprocessing environment. *IEEE Transactions on Parallel and Distributed Systems*, 9(3):295 –311, 1998.
6. D. C. Reeve. *A New Blueprint for Network QoS.* PhD thesis, Computing Laboratory, University of Kent, Canterbury, Kent, UK, August 2003.
7. R. Serfozo. *Basics of Applied Stochastic Processes.* Springer, 2009.
8. M. Zaharia, D. Borthakur, S. Sen, K. Elmeleegy, S. Shenker, and I. Stoica. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *Proceedings of the 5th European Conference on Computer Systems*, EuroSys '10, pages 265–278, New York, NY, USA, 2010. ACM.
9. J. Zhao, M. M. Dessouky, and S. T. S. Bukkapatnam. Optimal slack time for schedule-based transit operations. *Transportation Science*, 40(4):529–539, 2006.

Real-Time Detection of Process Change using Process Mining

Phil Weber, Behzad Bordbar, and Peter Tiňo

School of Computer Science, University of Birmingham, B15 2TT, UK.
{p.weber,b.bordbar,p.tino}@cs.bham.ac.uk

Abstract. Process Mining is the discovery of business processes from log files. One application is ensuring conformance to prescribed processes or business rules. Since businesses operate in real time, needing to quickly react to change, processes change; but how can such changes be detected? We consider requirements for process mining to support this: a notion of real time, and methods to compare processes and detect significant change. We present initial results confirming the validity of the approach.

Keywords: Process mining, machine learning, real time, distributions.

1 Introduction

Business processes describe related activities which are carried out to fulfil a business function. Fig.1 shows an example process, depicted as a probabilistic automaton. Each directed arc is labelled with a symbol representing an activity, and the conditional probability of that activity taking place next. As the process is executed, the systems involved will record information in log files. Abstracting from detail, the 'trace' of a single enactment of this process might be recorded as a string of symbols $iabdefgo$. Process mining [1, 10] algorithms use logs of such traces to discover and analyse process models.

Business processes are used to manage business operations, which today take place in real time, under pressures of time, cost and competition. Processes may also ensure adherence to business rules or regulatory requirements. Divergence from these processes may therefore indicate a business problem, or have legal ramifications, and so such changes need to be detected in a timely manner.

To enable detection of process change in real-time, several requirements need to be addressed. Firstly, a definition of real time and its application to process mining; secondly, a method to measure accurately the difference between two processes; thirdly, a method to detect change in a process; and finally a notion of statistical significance of the change. We briefly address these points in section 4. First we discuss related work, then introduce a probabilistic view of business processes and process mining, which underpins the subsequent ideas.

2 Related Work

Process Mining [1, 10] has been an active area of research since the early 1990s. Typically, non-probabilistic representations are used, such as Workflow nets [11]

Andrew V. Jones (Ed.): Proceedings of the 1st Imperial College Student Workshop (ICCSW '11), pp. 108–114. September 29th–30th 2011, London UK.

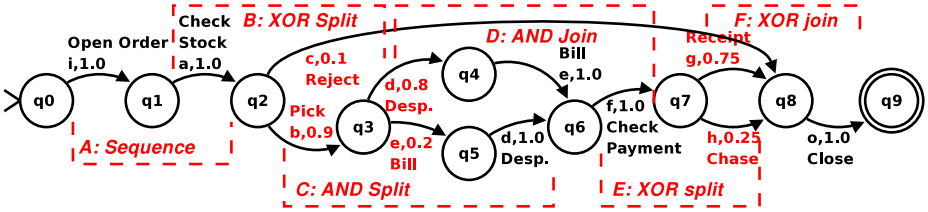

Fig. 1. PDFA showing a simplified business process for fulfilling an order.

or BPMN. Probabilistic approaches are the exception, e.g. [6]. There are three main sub-disciplines: *Process discovery* is the mining of models; *process conformance* the evaluation of results; and *extension* of models, for instance adding frequencies, mining decision rules, or predicting outcomes. We are concerned with conformance and discovery. Various algorithms have been proposed for process discovery, such as the 'formal' Alpha Algorithm [11]; others using heuristics, or various theoretical foundations such as Neural or Genetic [17].

'Real-time' is used informally in Business Process research in regard to the need for flexibility and process change to respond to a changing environment [9]. In [13, 15] process mining is part of a lifecycle of implementing and monitoring processes, finding discrepancies and resolving them, changing the model, or recommending a course of action. However, these do not discuss how much data is needed, nor how to identify when the underlying process has changed.

Other fields of research are partially related to, and may be able to inform, the work in this paper. Time series analysis [16] deals with changes to series of individual variables, whereas we are concerned with varying probability distributions over sequences. Stream mining [7] looks for patterns in data streams, and real-time data mining [5] investigates time constraints. Concept drift is the detection of change in machine learning. In [2] this is discussed in a process mining context, but with focus on model structure rather than probability.

3 A Probabilistic View of Business Processes

We model *activities* as symbols from a finite alphabet Σ, *traces* as strings $x \in \Sigma^+$, and a *process* as a probability distribution $P_{\mathcal{M}}$ over traces. Probability of trace x is $P_{\mathcal{M}}(x) : \sum_{x \in \Sigma^+} P_{\mathcal{M}}(x) = 1$. The task of a process mining algorithm is to learn a distribution $P_{\mathcal{M}'}$, to approximate $P_{\mathcal{M}}$, from the finite log W drawn *i.i.d.* from $P_{\mathcal{M}}$. This differs from existing views of process mining, which focus on discovery of a model structure in a specific representation such as Petri nets.

We use probabilistic deterministic finite automata (PDFA) [12] (Fig.1) to represent the probability distributions generated by process models, as a common denominator to which processes in other representations can be converted. A PDFA is a five-tuple $A = (Q_A, \Sigma, \delta_A, q_0, q_F)$, where Q_A is a finite set of states; Σ an alphabet of symbols; $q_0, q_F \in Q_A$ the single start and end states; and $\delta_A : Q_A \times \Sigma \times Q_A \rightarrow [0, 1]$ is a mapping defining the conditional transition probability function between states. $\delta(q_1, a, q_2)$ is the probability that given we

are in state q_1, we parse a and arrive in state q_2. Given a current state and symbol, the next state is certain. Probabilities on arcs from a state sum to 1.

PDFA A generates a probability distribution P_A on Σ^+. The probability of string x, $P_A(x)$, is found by multiplying the probabilities of the arcs followed to parse x on its unique path from the single start state q_0 to unique end state q_F.

4 Overview of Approach

4.1 Real-time Process Mining

The term 'real time' is used subjectively of systems which appear to process information 'fast'. Formally, real-time systems 'must react within precise time constraints to events in the environment' [3]. The key is predictability and results guaranteed in a specified time, rather than speed. For us this means identifying process change as soon as possible, but with confidence that change is significant.

We consider two main constraints: accuracy and time. The mining algorithm must produce a model 'close' to the 'true' model using some notion of distance between distributions. We expect accuracy to increase with the amount of data, but for this to increase mining time. So these two constraints act in tension. We desire to minimise mining time, but characteristics of the ground truth distribution will determine the minimum data needed for confidence in mining accuracy.

This lower bound on data ensures we use the correct baseline, against which to measure change. Although an upper bound can be set on the mining time, this will be constrained by the overhead of the algorithm, the time taken to process each trace, and by the desired accuracy. The Alpha algorithm [11] which we use here is quite efficient (linear in the size of the log, exponential in the number of tasks, which is typically very restricted), so the upper bound is of less import. Other algorithms such as the Genetic Miner [17] have much higher time complexity. Factors such as these must affect the choice of algorithm.

There are other issues which we do not consider, such as from the type or magnitude of change, predicting the time to detect it; or environmental issues which may affect the real time behaviour of the system [3].

4.2 Determining the Amount of Data Needed for Mining

One way to determine the amount of data needed is to consider the structures in a process (highlighted in Fig.1), and the probability of an algorithm discovering these structures. In [14] we discuss this approach and apply it to the Alpha algorithm [11], which uses heuristics about the relations seen between pairs of tasks in the log, to construct a Petri net. To compare this non-probabilistic model against the ground truth distribution, we convert the net to a PDFA by labelling its reachability graph (RG) with maximum likelihood probabilities obtained from the mining log. This allows us to satisfy the accuracy constraint.

We do not address the time constraint, since Alpha has low complexity and although the time to generate the RG is exponential in the number of states, we use only simple acyclic models. Business process models are in general relatively simple, but further work is needed to validate the efficiency of our approach.

4.3 Methods to Detect Process Change

We mine repeatedly from sublogs, using a 'sliding window', and compare the distribution generated by the mined model with the ground truth distribution. There are many measures of difference between probability distributions, such as Euclidean distance, Kullback-Leibler Divergence. Some can be efficiently calculated from PDFA, but it is not clear what distance is statistically significant. Instead, we use statistical tests for detecting that the mined distribution, or its PDFA representation, has changed significantly from the ground truth.

The count of each unique trace x in the log can be modelled as a Binomially-distributed random variable, since any trace in the log will either be x, or not. The same is true of the number of times each arc in the PDFA is used in generating the log: each trace will either use that arc, or not (at present we assume acyclic models). If the number of traces is large enough relative to the trace/arc probabilities, the Binomial can be approximated by the Normal distribution.

Goodness of Fit Test on the Distribution: The sum of k Normally distributed random variables follows a Chi^2 distribution with $k-1$ degrees of freedom. Thus we can use the Chi^2 test to determine whether the difference between the count of each unique trace found in the sample, and the expected count, is likely under the assumption that the log was drawn from the ground truth distribution. The so-called p-value gives the probability that the Chi^2 distribution would exceed the measured value, indicating that with probability $1-p$, the process has changed.

Bounds and Hypothesis Tests on the PDFA: We expect the PDFA from the mining result to have the same state structure as the ground truth PDFA (making assumptions about the ground truth PDFA and the mining algorithm). The Hoeffding inequality upper bounds the probability of a sum of random variables deviating from its expected value. As [4], we use this to compare the probability of each arc from equivalent states in the models, by comparing the sum of the Bernouilli variables that each trace involves use of that arc.

Secondly, as [8] we use a hypothesis test to test how likely it is that an arc would be used the number of times indicated by its probability in the mined model, to generate the log, assuming the ground truth probability. Here the count is modelled as Binomial or Normal variable.

Bounds and Hypothesis Tests on Traces: The methods described for testing PDFA arcs can similarly be applied to process traces, so that we can use Hoeffding bounds or hypothesis tests to determine whether a process trace is likely to occur with the observed frequency, under the ground truth distribution.

5 Experimentation and Analysis

We used the example process of Fig.1. Using our method [14] the Alpha algorithm needs 44 traces to, with 99% probability, correctly mine a (non-probabilistic) Petri net with the correct structure. We randomly simulated the PDFA to produce an MXML[1] format log file of this size, and regularly updated it by simu-

[1] Mining eXtensible Markup Language, see `www.processmining.org`.

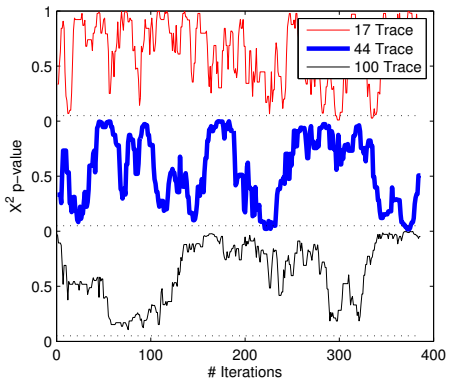

Fig. 2. Fluctuations in X^2 p-value over time, from unchanged source process.

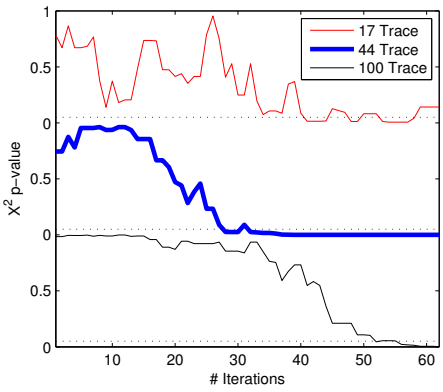

Fig. 3. Detection of XOR probability change using X^2 p-value.

lating one new trace and removing the oldest. This simulates a 'sliding window' onto a log file being updated in real time by a live process. Changes were introduced to the probabilities or structures in this PDFA. At each iteration, we used the Alpha algorithm[2] to mine a Petri Net from the current log and converted to a PDFA (section 4.2). We recorded distances between the distribution generated by this PDFA and the ground truth, and results of the tests in section 4.3.

We ran three experiments to test the hypotheses that (i) change is detectable using a variety of methods, (ii) more significant change is detected in fewer traces, and (iii) the predicted number of traces for mining the model is the optimum to use for detecting change, thus allowing detection in real time.

Since the Alpha algorithm mines only a Petri net structure (no probabilities), it needs a relatively small sample of traces, which exhibits high variance from the ground truth (Fig.2), resulting in high risk of false positives (incorrectly detecting change) or false negatives (not detecting true change). We did not take this into account beyond ensuring no false positives occurred before change was introduced, but it would affect the detection point. These initial results were also based on one test only of each sample. The main results seem clear, but are not statistically valid without averaging over multiple tests.

Varying Probabilities We varied probabilities in the XOR split B, and parallel split C. Small variations (< 0.1) were not detectable, although the distance measures increased. For the XOR split, change to $p(ab) = 0.7$ was discovered in 28 iterations, reducing to 9 for $p(ab) = 0.1$. Detection was first by X^2 (Fig.3), then by hypothesis test on strings (Fig.4) or arcs, and last by the Hoeffding tests. The looseness of the Hoeffding bound allows the string/arc frequencies to be more readily accepted as within confidence bounds given the ground truth.

The variation of AND probabilities was tested with probability of the structure in the model being 0.9 and then 0.1. The latter change was detected first

[2] implemented in the process mining tool ProM (`www.processmining.org`).

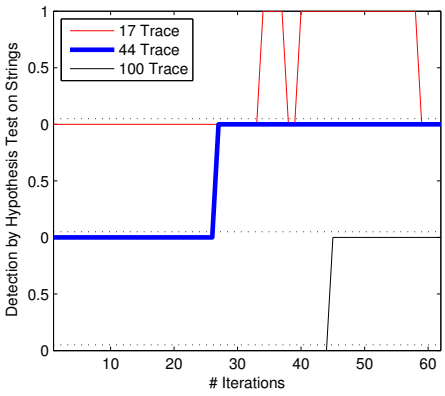

Fig. 4. Detection of XOR probability change using hypothesis test on strings.

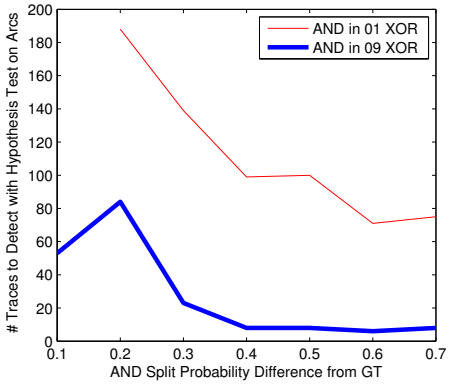

Fig. 5. AND change detection using hypothesis test on arcs, varying probabilities.

by arc differences (Fig.5), the string difference methods not detecting it at all. This is explained by the probability of traces passing through the AND structure being too low to detect significant changes, but for those that do, changes to arc usage are local and not affected by the global probability of the structure.

Varying amount of data We varied the amount of data in the 'sliding window'. With 44 traces we see high variance in the probability distribution, seen in the large fluctuations in X^2 p-value in the centre graph of Fig.2. The lower graph shows that the frequency and amplitude of these changes is reduced with 100 traces, with no significant (0.05) p-values. The cost is slower detection of change (Fig.3 and 4). Conversely, reducing the number of traces to 17, change can be detected sooner, but with higher risk of false positive or false negative.

6 Conclusion and Future Work

We examined various methods for detecting change in a running process, with initial results showing that using the optimal amount of data to be confident that the mined process is correct, various statistical methods can be used to efficiently detect change in real time. The Chi^2 test allows earliest detection of change, except where the change is in a low probability part of the model, when hypothesis testing the arc frequencies is a better choice.

Further work is needed to determine how to choose the optimal method to detect change, to understand the effect of variation in the underlying distribution and the risk of falsely identifying or missing change, and to predict the time to detect change. Some distances between distributions can be efficiently calculated from PDFA, so understanding of the significance of distance measures, would lead to more efficient methods for detecting change. More work is also needed to ensure the efficiency of the proposed method. Finally, the question remains, is process mining a better approach than simply analysing the log distributions?

References

1. van der Aalst, W. M. P., and Weijters, T. Process mining: a research agenda. *Comput. Ind.*, 53(3):231–244, 2004.
2. Jagadeesh Chandra Bose, R. P., van der Aalst, W. M. P., Zliobaite, I., and Pechenizkiy, M.. Handling concept drift in process mining. In Mouratidis, H. and Rolland, C. (eds.), *CAiSE*, LNCS, vol. 6741, pp. 391–405. Springer, 2011.
3. Giorgio C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, Norwell, USA, 1997.
4. Carrasco, R. C. and Oncina, J. Learning stochastic regular grammars by means of a state merging method. In Rafael C. Carrasco and José Oncina (eds.), *ICGI*, LNCS, vol. 862, pp. 139–152. Springer, 1994.
5. Xiong Deng, Ghanem, M., and Guo, Y. Real-time data mining methodology and a supporting framework. In Yang Xiang, Lopez, J., Haining Wang, and Wanlei Zhou. (eds.), *NSS 2009*, pp. 522–527. IEEE Computer Society, 2009.
6. Ferreira, D. R. and Gillblad, D. Discovering Process Models from Unlabelled Event Logs. In Dayal, U., Eder, J., Koehler, J., and Reijers, H. A. (eds.), *BPM 2009*. LNCS, vol. 5701, pp. 143–158. Springer, 2009.
7. Gaber, M. M., Zaslavsky, A. B/, and Krishnaswamy, S. Mining data streams: a review. *SIGMOD Record*, 34(2):18–26, 2005.
8. Jacquemont, J., Jacquenet, J., and Sebban, M.. Mining probabilistic automata: A statistical view of sequential pattern mining. *Machine Learning*, 75(1):91–127, 2009.
9. Rinderle, S., Reichert, R., and Dadam, P.. Correctness criteria for dynamic changes in workflow systems - a survey. *Data Knowl. Eng.*, 50(1):9–34, 2004.
10. Tiwari, A., Turner, C. J., and Majeed, B. A Review of Business Process Mining: State-of-the-Art and Future Trends. *Bus. Process Manage. J.*, 14(1):5 – 22, 2008.
11. van der Aalst, W. M. P., Weijters, T., and Maruster, L. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Trans. Knowl. Data Eng.*, 16(9):1128–42, 2004.
12. Vidal, E., Thollard, F., de la Higuera, F., Casacuberta, F., and Carrasco, R. C. Probabilistic Finite-State Machines - Part I. *IEEE Trans. Pattern Anal.*, 27(7):1013 – 25, 2005.
13. Weber, B., Sadiq, S. W., and Reichert, M.. Beyond rigidity - dynamic process lifecycle support. *Computer Science - R&D*, 23(2):47–65, 2009.
14. P. Weber, B. Bordbar, and P. Tiňo. A principled approach to the analysis of process mining algorithms. In *Proceedings of the 12th International Conference on Intelligent Data Engineering and Automated Learning (to appear)*, 2011.
15. Schonenberg, H., Weber, B., van Dongen, B. F., and van der Aalst, W. M. P. Supporting flexible processes through recommendations based on history. In Dumas, M., Reichert, M., and Ming-Chien Shan. (eds.), *BPM 2008*. LNCS, vol. 5240, pp. 51–66. Springer, 2008.
16. Hamilton, J.D. *Time series analysis*. Princeton University Press, 1994.
17. de Medeiros, A. K. A., Weijters, T., and van der Aalst, W. M. P. Genetic process mining: an experimental evaluation. *Data Min. Knowl. Discov.*, 14(2):245–304, 2007.

Author Index