



# **COMPUTATION TOOLS 2016**

The Seventh International Conference on Computational Logics, Algebras,  
Programming, Tools, and Benchmarking

ISBN: 978-1-61208-466-4

March 20 - 24, 2016

Rome, Italy

## **COMPUTATION TOOLS 2016 Editors**

Claus-Peter Rückemann, Westfälische Wilhelms-Universität Münster / Leibniz  
Universität Hannover / North-German Supercomputing Alliance, Germany

Lorenzo Bettini, University of Torino, Italy

# COMPUTATION TOOLS 2016

## Forward

The Seventh International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking (COMPUTATION TOOLS 2016), held between March 20-24, 2016 in Rome, Italy, continued a series of events dealing with logics, algebras, advanced computation techniques, specialized programming languages, and tools for distributed computation. Mainly, the event targeted those aspects supporting context-oriented systems, adaptive systems, service computing, patterns and content-oriented features, temporal and ubiquitous aspects, and many facets of computational benchmarking.

The conference had the following tracks:

- Advanced computation techniques
- Tools for distributed computation
- Logics

Similar to the previous edition, this event attracted excellent contributions and active participation from all over the world. We were very pleased to receive top quality contributions.

We take here the opportunity to warmly thank all the members of the COMPUTATION TOOLS 2016 technical program committee, as well as the numerous reviewers. The creation of such a high quality conference program would not have been possible without their involvement. We also kindly thank all the authors that dedicated much of their time and effort to contribute to COMPUTATION TOOLS 2016. We truly believe that, thanks to all these efforts, the final conference program consisted of top quality contributions.

Also, this event could not have been a reality without the support of many individuals, organizations and sponsors. We also gratefully thank the members of the COMPUTATION TOOLS 2016 organizing committee for their help in handling the logistics and for their work that made this professional meeting a success.

We hope COMPUTATION TOOLS 2016 was a successful international forum for the exchange of ideas and results between academia and industry and to promote further progress in the area of computational logics, algebras, programming, tools, and benchmarking. We also hope that Rome provided a pleasant environment during the conference and everyone saved some time for exploring this beautiful city.

## **COMPUTATION TOOLS 2016 Chairs**

### **COMPUTATION TOOLS Advisory Chairs**

Kenneth Scerri, University of Malta, Malta

Alexander Gegov, University of Portsmouth, UK

Ahmed Khedr, University of Sharjah, UAE

### **COMPUTATIONAL TOOLS Industry/Research Chairs**

Torsten Ullrich, Fraunhofer Austria Research GmbH - Graz, Austria

Zhiming Liu, UNU-IIST, Macao

### **COMPUTATION TOOLS Publicity Chair**

Lev Naiman, University of Toronto, Canada

Ingram Bondin, University of Malta, Malta

Tomáš Bublík, Czech Technical University in Prague, Czech Republic

# COMPUTATION TOOLS 2016

## Committee

### COMPUTATION TOOLS Advisory Committee

Kenneth Scerri, University of Malta, Malta  
Alexander Gegov, University of Portsmouth, UK  
Ahmed Khedr, University of Sharjah, UAE

### COMPUTATIONAL TOOLS Industry/Research Chairs

Torsten Ullrich, Fraunhofer Austria Research GmbH - Graz, Austria  
Zhiming Liu, UNU-IIST, Macao

### COMPUTATION TOOLS Publicity Chair

Lev Naiman, University of Toronto, Canada  
Ingram Bondin, University of Malta, Malta  
Tomáš Bublík, Czech Technical University in Prague, Czech Republic

### COMPUTATION TOOLS 2016 Technical Program Committee

Yas Aksultanny, Arabian Gulf University, Bahrain  
Youssif B. Al-Nashif, Old Dominion University, USA  
Adel Alimi, University of Sfax, Tunisia  
François Anton, Technical University of Denmark, Denmark  
Henri Basson, University of Lille North of France (Littoral), France  
Steffen Bernhard, TU-Dortmund, Germany  
Ateet Bhalla, Independent Consultant, India  
Paul-Antoine Bisgambiglia, Université de Corse, France  
Narhimene Boustia, Saad Dahlab University - Blida, Algeria  
Azahara Camacho, Complutense University of Madrid, Spain  
Luca Cassano, University of Pisa, Italy  
Emanuele Covino, Università degli Studi di Bari Aldo Moro, Italy  
Hepu Deng, RMIT University - Melbourne, Australia  
Rene de Souza Pinto, University of Sao Paulo, Brazil  
Craig C. Douglas, University of Wyoming / Yale University, USA  
António Dourado, University of Coimbra, Portugal  
Eugene Feinberg, Stony Brook University, USA  
Tommaso Flaminio, University of Insubria, Italy  
Janos Fodor, Obuda University, Hungary  
Alexander Gegov, University of Portsmouth, UK  
Victor Gergel, University of Nizhni Novgorod (UNN), Russia

Veronica Gil-Costa, University of San Luis, Argentina  
Luis Gomes, Universidade Nova de Lisboa, Portugal  
Yuriy Gorbachev, Geolink Technologies LLC, Russia  
George A. Gravvanis, Democritus University of Thrace, Greece  
Rajiv Gupta, University of California - Riverside, USA  
Fikret Gurgen, Bogazici University - Istanbul, Turkey  
Hani Hamdan, École Supérieure d'Électricité (SUPÉLEC), France  
Said Jabbour, CRIL - CNRS | University of Artois, France  
Nawaz Khan, Middlesex University London, UK  
Cornel Klein, Siemens AG - Munich, Germany  
Stano Krajci, Safarik University - Kosice, Slovakia  
Isaac Lera, University of the Balearic Islands, Spain  
Tsung-Chih Lin, Feng-Chia University, Taichung, Taiwan  
Glenn R. Luecke, Iowa State University, USA  
Reza Madankan, University of Texas MD Anderson Cancer Center, USA  
Elisa Marengo, Free University of Bozen-Bolzano, Italy  
Roderick Melnik, Wilfrid Laurier University, Canada  
Rob Miller, University College London, UK  
Julian Molina, University of Malaga, Spain  
Susana Muñoz Hernández, Universidad Politécnica de Madrid, Spain  
Adam Naumowicz, University of Bialystok, Poland  
Gianina Alina Negoita, Iowa State University, USA  
Cecilia E. Nugraheni, Parahyangan Catholic University - Bandung, Indonesia  
Flavio Oquendo, European University of Brittany/IRISA-UBS, France  
Javier Panadero, Universitat Autònoma de Barcelona, Spain  
Juan Jose Pardo, University of Castilla-la Mancha, Spain  
Vangelis Paschos, LAMSADE - University Paris-Dauphine, France  
Mario Pavone, University of Catania, Italy  
Mikhail Peretyat'kin, Institute of mathematics and mathematical modeling, Kazakhstan  
Alexandre Pinto, ISG - Royal Holloway University of London, UK / Instituto Superior da Maia, Portugal  
Enrico Pontelli, New Mexico State University, USA  
Corrado Priami, CoSBI & University of Trento, Italy  
Marcus Randall, Bond University, Australia  
Dolores Rexachs, University Autònoma of Barcelona (UAB), Spain  
Morris Riedel, University of Iceland, Iceland / Juelich Supercomputing Centre, Germany  
Ricardo Rocha, University of Porto, Portugal  
Lakhdar Saïs , CRIL - CNRS, University of Artois, France  
Guido Sciavicco, University of Murcia, Spain  
Evgenia Smirni, College of William and Mary - Williamsburg, USA  
Patrick Siarry, Université de Paris 12, France  
Hooman Tahayori, Ryerson University, Canada  
James Tan, SIM University, Singapore  
Torsten Ullrich, Fraunhofer Austria Research GmbH, Austria  
Miroslav Velez, Aries Design Automation, USA  
Chao-Tung Yang, Tunghai University, Taiwan  
Marek Zaremba, Université du Québec en Outaouais - Gatineau, Canada  
Naijun Zhan, Institute of Software/Chinese Academy of Sciences - Beijing, China

## Copyright Information

For your reference, this is the text governing the copyright release for material published by IARIA.

The copyright release is a transfer of publication rights, which allows IARIA and its partners to drive the dissemination of the published material. This allows IARIA to give articles increased visibility via distribution, inclusion in libraries, and arrangements for submission to indexes.

I, the undersigned, declare that the article is original, and that I represent the authors of this article in the copyright release matters. If this work has been done as work-for-hire, I have obtained all necessary clearances to execute a copyright release. I hereby irrevocably transfer exclusive copyright for this material to IARIA. I give IARIA permission to reproduce the work in any media format such as, but not limited to, print, digital, or electronic. I give IARIA permission to distribute the materials without restriction to any institutions or individuals. I give IARIA permission to submit the work for inclusion in article repositories as IARIA sees fit.

I, the undersigned, declare that to the best of my knowledge, the article does not contain libelous or otherwise unlawful contents or invading the right of privacy or infringing on a proprietary right.

Following the copyright release, any circulated version of the article must bear the copyright notice and any header and footer information that IARIA applies to the published article.

IARIA grants royalty-free permission to the authors to disseminate the work, under the above provisions, for any academic, commercial, or industrial use. IARIA grants royalty-free permission to any individuals or institutions to make the article available electronically, online, or in print.

IARIA acknowledges that rights to any algorithm, process, procedure, apparatus, or articles of manufacture remain with the authors and their employers.

I, the undersigned, understand that IARIA will not be liable, in contract, tort (including, without limitation, negligence), pre-contract or other representations (other than fraudulent misrepresentations) or otherwise in connection with the publication of my work.

Exception to the above is made for work-for-hire performed while employed by the government. In that case, copyright to the material remains with the said government. The rightful owners (authors and government entity) grant unlimited and unrestricted permission to IARIA, IARIA's contractors, and IARIA's partners to further distribute the work.

## Table of Contents

A Perceptron-Based Task Predictor for Multi-Core Processor Architectures <i>Jongbok Lee</i>	1
Implementing the Type System for a Typed Javascript and its IDE <i>Lorenzo Bettini, Jens von Pilgrim, and Mark-Oliver Reiser</i>	6
Logical Characterization and Complexity of Weighted Branching Preorders and Distances <i>Louise Foshammer, Kim Guldstrand Larsen, Radu Mardare, and Bingtian Xue</i>	12
Towards an Astrophysical-oriented Computational multi-Architectural Framework <i>Dzmitry Razmyslovich, Guillermo Marcus, and Reinhard Manner</i>	19

# A Perceptron-Based Task Predictor for Multi-Core Processor Architectures

Jongbok Lee

Dept. of Information and Communications Engineering  
Hansung University  
Seoul, Republic of Korea  
Email: jblee@hansung.ac.kr

**Abstract**—In order to increase the performance of multi-core system processors, the task predictor which speculatively fetches and allocates tasks to each core should be highly accurate. In this paper, a perceptron-based task predictor is proposed for the multi-core processor architectures. Using SPEC 2000 benchmarks as input, the trace-driven simulation has been performed for the dual-core to octa-core processors employing perceptron-based task predictor extensively. Its performance is compared with the architecture which utilizes the conventional two-level adaptive task predictor.

**Keywords**—multi-core processor, perceptron

## I. INTRODUCTION

Currently, multi-core processors are widely used for the high performance of the computer system, such as smart phones, tablet PCs, notebook computers, and desk top computers, etc [1]–[6]. By utilizing a task predictor, a program is partitioned into speculative multiple tasks which are assigned to the processing core units. Hence, the task predictor should be very accurate in order to effectively take advantage of a multi-core processor architecture. Recently, neural networks such as perceptrons are widely used in the digital systems, which can take advantage of learning. In this paper, a perceptron-based task predictor for multi-core processor is proposed. The SPEC2000 integer benchmark programs are used for estimating the performance of multi-core processors using perceptrons. The result is compared with the performance of the multi-core processor with the conventional scheme.

This paper is organized as follows. In the Section 2, the perceptron-based task predictor will be discussed. The simulation environment will be described in the Section 3. In the Section 4, the simulation results will be analyzed. Finally, the Section 5 concludes our paper.

## II. RELATED STUDIES

Perceptron is the neural network capable of learning by producing outputs combined with the inputs and the associated weight values. In the past studies, it has been adopted for predicting branches in the computer systems [7]–[9]. Figure 1 describes the graphic model of a perceptron. A perceptron is represented by weight vectors, which are composed of positive or negative integers. The output is the dot product of the weight vector  $w_{0..N}$  and the input vector  $x_{0..N}$ . The first element  $x_0$  is always set to 1 to serve as a bias input.

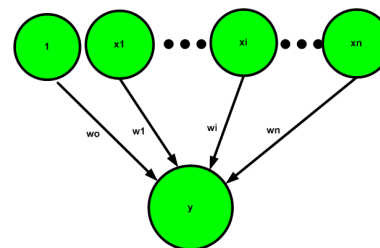


Figure 1. The perceptron.

$$y = w_0 + \sum_{i=1}^n x_i w_i \quad (1)$$

Therefore, before adapting to the previous branch results, the biased weight  $w_0$  always enables the perceptron to be biased in the initial stage. The output of a perceptron is represented as (1). The input to a perceptron is bipolar, which means that the branch is not taken if  $x_i$  is -1, and taken if  $x_i$  is 1. When the output is negative, the branch is predicted as not taken; When the output is positive, it is predicted as taken. When a branch is met, the branch address is used to generate an index between 0 and N-1 to access the perceptron table. After obtaining the weight vector  $P_{0..N}$  by fetching the  $i_{th}$  perceptron, the dot product of the weight vector and the global history register is generated to output  $y$ . The direction of the next branch is predicted upon the sign of the output. When the actual direction of the branch is available, the result is used to update the weight value of the vector P. Then, the vector P is recorded to the  $i_{th}$  entry of the perceptron table.

After the perceptron output  $y$  has been computed, the following algorithm is used to train the perceptron. Let  $t$  be -1 if the branch was not taken, or 1 if it was taken, and let  $\theta$  be the threshold, a parameter to the training algorithm used to decide when enough training has been done. Since  $t$  and  $x_i$  are always either -1 or 1, this algorithm increments the  $i_{th}$  weight when the branch outcome agrees with  $x_i$ , and decrements when it disagrees.

## III. THE PERCEPTRON-BASED TASK PREDICTOR

The task prediction of multi-core processors using perceptrons can be implemented in the similar mechanism as the branches are predicted. Figure 3 illustrates the mechanism



```

if  $sign(y_{out}) \neq t$  or  $|y_{out}| \leq \theta$ 
then
  for  $i = 0$  to  $n$  do
     $w_i = w_i + tx_i$ 
  end for
end if
    
```

Figure 2. The perceptron algorithm

of the perceptron-based predictor. The predictor records the finite length of task history results to the task history register and accesses the weight vector table to make a prediction. A temporary task history register is utilized, and at the beginning of each multiple task prediction, the contents of the task history register is transferred to the temporary task history register. In order to predict multiple tasks, the task's starting address

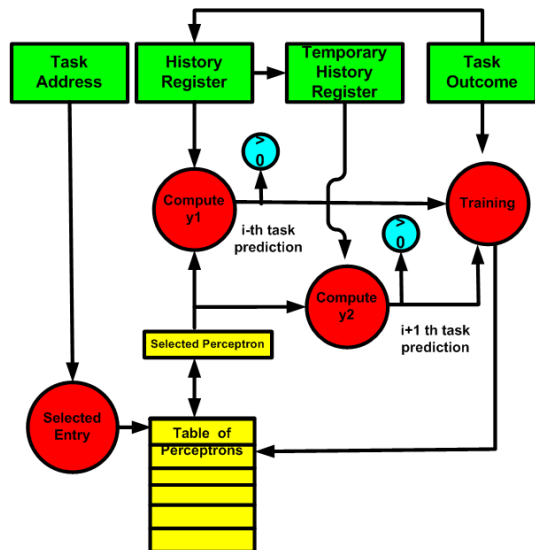


Figure 3. The perceptron-based task predictor

is hashed by the  $k$ -bits of history register which is used to index the weight vector table and to predict the  $i_{th}$  task. The  $i + 1_{th}$  task is predicted on the assumption that the first task prediction is correct. For this purpose, the rightmost 1-bit of the temporary task history register is updated and multiplied to an indexed weight vector to make the  $i + 1_{th}$  task prediction. In this way, two tasks can be predicted per cycle. Later, when the task outcomes are known, the task history register is updated according to the results. Similarly, to predict the  $i + 2_{th}$  task, the rightmost 2 bits of the temporary task history register is updated based on the first and the second task predictions.

#### IV. THE SIMULATION ENVIRONMENT

##### A. The multi-core processor architecture

Figure 4 shows the multi-core processor with  $N$  cores. Each core is an out-of-order superscalar processor which can execute

instructions in a task [10]. In addition, it has a L1 instruction

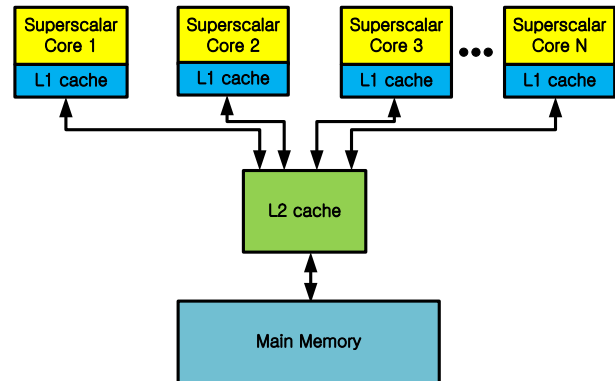


Figure 4. The multicore processor

cache and a L1 data cache. For the cache coherency of the L1 data cache, MESI protocol is utilized. If the data in the L1 data cache associated with a core is over-written by another core, it is invalidated. The L2 cache is shared among the cores, which is connected with the main memory.

The superscalar processor core is allocated with tasks which consist of a number of instructions. The fetched instructions in the task are decoded, renamed, executed, and written back. When all the instructions in the task are retired and becomes empty, new instructions of task are fetched. If the task is mispredicted, the fetch is aborted, and all the remained instructions in the task are squashed. Since the instructions are renamed, the instructions can be issued and executed out-of-order as long as there is no true-dependency. Although the instructions can be retired out-of-order, the instructions are inserted into the reorder buffer and committed in-order as to preserve the original program order.

The detailed architecture configurations and cache parameters for each core are listed in Table I. The number of simulated cores are 1, 2, 4, and 8. Each core is assigned with the maximum of two tasks respectively. Since the small task size cannot take the benefit of the instruction level parallelism, the task sizes are set to 4, 8, and 16. The functional unit of each

TABLE I. ARCHITECTURE CONFIGURATION FOR EACH CORE.

Item	Value	
number of cores	1,2,4,8	
number of tasks per core	1,2	
task length	4,8,16	
fetch,issue,retire rate	2,4,8	
functional unit	integer ALU	2,4,8
	load/store	1,2,4
L1-instruction cache	64 KB, 2-way set assoc., 16 B block, 10 cycles miss penalty	
L1-data cache	64 KB, 2-way set assoc., 32 B block, 10 cycles miss penalty	
task address cache	2K entry	
task predictor	two-level adaptive	14-bit global history
	perceptron	8-bit global history, 4096 Pattern History Table

core consists of a number of ALUs, load/store units according to each configuration. For the memory disambiguation, load-store and store-store pairs are inhibited from the speculative

execution when the effective addresses are matched, within or among the cores. The L1 instruction cache and L1 data cache for each core is 64 KB, and it is designed as 2-way set associative. This is because the data cache hit ratio can be degraded by using MESI protocol among multiple cores. For the reference, tasks are predicted using the two-level adaptive prediction scheme. The two-level adaptive task predictor is similar to the two-level adaptive branch prediction scheme where the branch address simply corresponds to the task starting address [11] [12]. In correspondence with the perceptron-based task predictor, the two-level adaptive task predictor employs a 14-bits of global history register and 16,384 items for the pattern history table. For the perceptron-based task predictor, the length of task history register is 8-bits, and the number of pattern history table is set to 4096. The threshold value for the perceptron learning is shown as 2, where  $TN$  is the length of the task history register. For both predictors, the task address cache has the size of 2048 entries. Since we do not model the main memory, the hit ratio of L2 cache is assumed to be as 100 %.

$$\theta = 2 \times TN + 14 \quad (2)$$

### B. The multi-core processor simulator

Figure 5 depicts how the developed simulator works [13]. *Initialize* function initializes all the associated variables, and *Grouping*, *Create\_Window*, and *Fetch\_One\_Instr* function fetches new instructions to fill core tasks every cycle. The

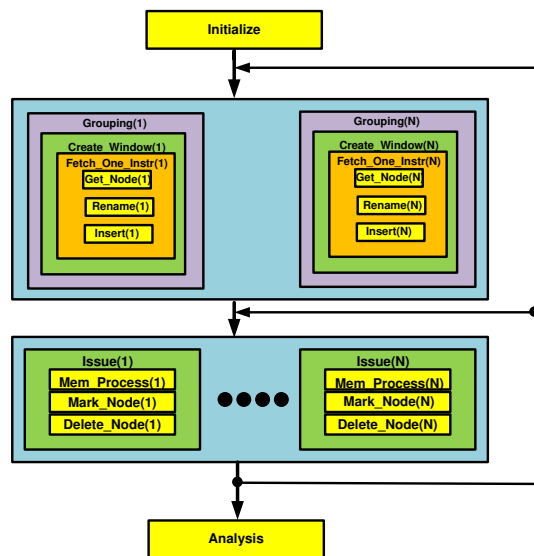


Figure 5. The flow chart of the multi-core processor simulator

instruction fetched by *Get\_Node* function is renamed at *Rename* function by receiving timestamps. After the instruction is renamed, it is inserted into a core task by *Insert* function. At the *Issue* function, the instruction in the core task can be retired so long as the corresponding functional unit is available and its time stamp is less than or equal to the current cycle. For implementing the multi-core simulation, *Grouping* function fills instructions of n-core tasks, and *Issue* function deletes instructions according to their timestamps. This process is repeated until all the fetched instructions in the core tasks are deleted to become empty. Then, the core tasks are filled

again with instructions by *Grouping* function. Since the cycle is incremented for each process, the core which spends the longest cycles determines the global cycle. If the total number of executed instruction is divided by the number of global cycles spent, then Instruction per Cycle (IPC) can be obtained. The eight SPEC 2000 integer benchmark programs that is used for the input are *bzip2*, *crafty*, *gap*, *gcc*, *gzip*, *mcf*, *parser*, and *twolf* as shown in Table II. The programs are compiled by

TABLE II. SPEC 2000 BENCHMARK PROGRAMS

benchmark	description
bzip2	compression
crafty	chess game
gap	group theory interpreter
gcc	C programming language compiler
gzip	compression
mcf	optimization of combination
parser	word processor
twolf	placement and global routing

SimpleScalar cross C compiler to obtain executables under Linux 3.3.4 [14]. The execution files are again run with SimpleScalar to obtain 100 million MIPS IV instruction traces, which are used as input for the multi-core processors. The task-level parallelism is mapped onto each core, and the trace-driven simulation is performed to get performance [15].

### V. THE SIMULATION RESULTS

Figure 6 presents the simulation results of running SPEC 2000 integer programs on the three different task lengths for the single-core, dual-core, quad-core, and octa-core processors. The performance results obtained by the two-level adaptive task predictor and the perceptron-based task predictor are compared in parallel. Figure 6a and 6b are the result of the multi-core processors with the maximum task length of four. Across the number of different cores, *bzip2* and *mcf* scores the highest performance owing to the relatively high parallelism and the low cache miss rates. However, *gcc* results in the lowest performance due to the severe losses from the low instruction and data cache hit rates. For the dual-core processors, the two-level adaptive task predictor brings the geometrical mean of 2.60 IPC, whereas the perceptron-based task predictor results in 2.63 IPC. For the octa-core processors, the two-level adaptive task predictor and the perceptron-based predictor results in 7.64 IPC and 7.73 IPC, respectively. With the perceptron-based task predictor, the performance is 1.7 times enhanced as the number of cores doubles. Therefore, when the performance of the octa-core processor is compared with the single-core, it is 5.4 times higher. With the maximum task length of four, the perceptron-based task predictor performs 1.1 % higher than the two-level adaptive task predictor.

Figure 6c and 6d show the results with the maximum task length of eight. Still, *bzip2* and *gcc* show the best and the poorest performance, respectively. For the quad-core processor, the two-level adaptive task predictor brings 7.53 IPC, whereas the perceptron-based task predictor scores 7.75 IPC. The respective performance for the octa-core processor are 12.3 IPC and 12.5 IPC. With the task length of eight, the octa-core processor brings 5.3 times higher performance than the single-core processor, which is slightly lower than the task length of four. However, the task length of eight performs 1.6 times better than the task length of four. Hence, the proposed

scheme scores higher performance than the two-level adaptive scheme by 2.8 %.

Finally, Figure 6e and 6f present the comparison result when the maximum task length is sixteen. *Parser* outperforms *bzip2* by the enlargement of the task length, whereas *gcc* still maintains low performance. For the dual-core processors, the two-level adaptive task predictor brings 6.5 IPC, whereas the perceptron-based task predictor results in 6.9 IPC. For the quad-cores, the respective values are 11.24 IPC and 11.67 IPC. And for the octa-cores, they are increased to 17.8 IPC and 18.3 IPC, respectively. With the perceptron, the performance has increased 1.8 times higher as the single-core goes to the dual-core processor. However, it is slightly decreased to 1.6 times when the quad-cores go to the octa-cores. The octa-core results in the 4.9 times higher performance than the single-core with the maximum task length of sixteen. Although the increase rate has been slowed down, the maximum task length of sixteen gives 1.5 times and 2.4 times higher performances than the maximum task length of eight and four, respectively. When the maximum task length is sixteen, the perceptron-based task predictor prevails the two-level adaptive task predictor by 5.1 %.

## VI. CONCLUSIONS

In this paper, a perceptron-based task predictor for multi-core processors has been proposed. The single-core to octa-core processors using perceptron with different task lengths have been simulated. As the result shows, the performance of multi-core processors with the perceptron-based task predictor scores higher performance than the two-level adaptive task predictor. When the task lengths are 4, 8, and 16, the respective performance increase over the two-level adaptive scheme are 1.1 %, 2.8 %, and 5.1 %.

For the future research, we will apply the perceptron-based task predictor to the asymmetric multi-core processor to further improve the efficiency, as well as expanding our scope to the multi-core embedded and multi-core digital signal processor architectures.

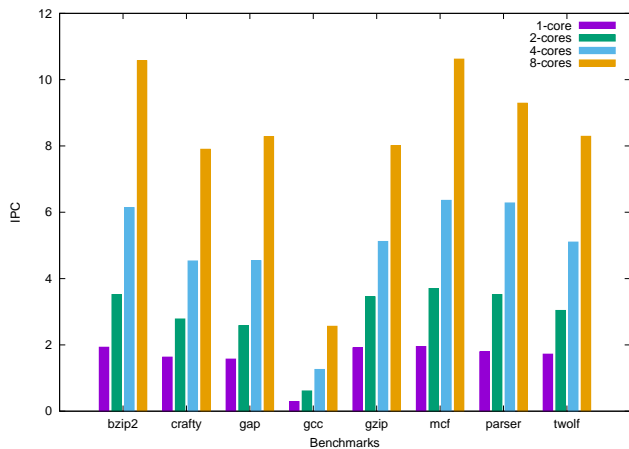
## ACKNOWLEDGMENT

The author would like to thank Hansung University for the financial support of this research.

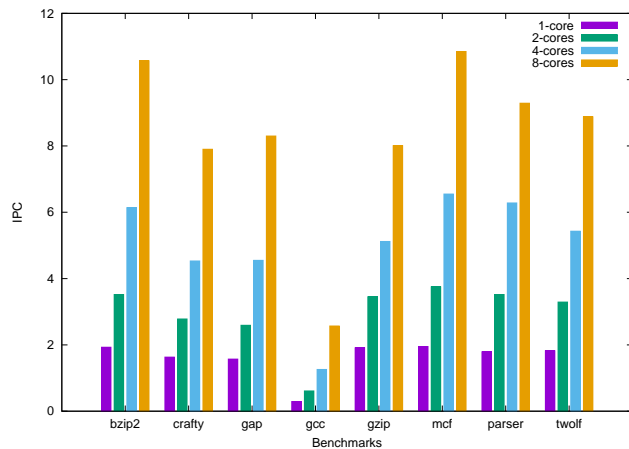
## REFERENCES

- [1] D. E. Culler and J. P. Singh, *Parallel Computer Architecture*. Morgan Kaufmann Publishers Inc., Aug. 1998.
- [2] T. Ungerer, B. Robic, and J. Silk, "Multithreaded Processors," *The Computer Journal*, vol. 45, no. 3, 2002.
- [3] S. W. Keckler, K. Olukotun, and H. P. Hofsee, *Multicore Processors and Systems*. Springer, 2009.
- [4] M. Monchiero, "How to simulate 1000 cores," *ACM SIGARCH Computer Architecture News archive*, vol. 37, no. 2, May 2009, pp. 10–19.
- [5] S. Biswas, D. Franklin, A. Savage, R. Dixon, T. Sherwood, and F. T. Chong, "Multi-execution : Multicore caching for data-similar executions," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, 2009, pp. 164–173.
- [6] D. Genbrugge and L. Eckhout, "Chip multiprocessor design space exploration through statistical simulation," *IEEE Transactions on Computers*, vol. 58, no. 12, Dec. 2009, pp. 1668–1681.
- [7] D. A. Jimenez and C. Lin, "Neural methods for dynamic branch prediction," *ACM Transactions on Computer Systems*, vol. 40, no. 2, Mar 1999, pp. 24–36.

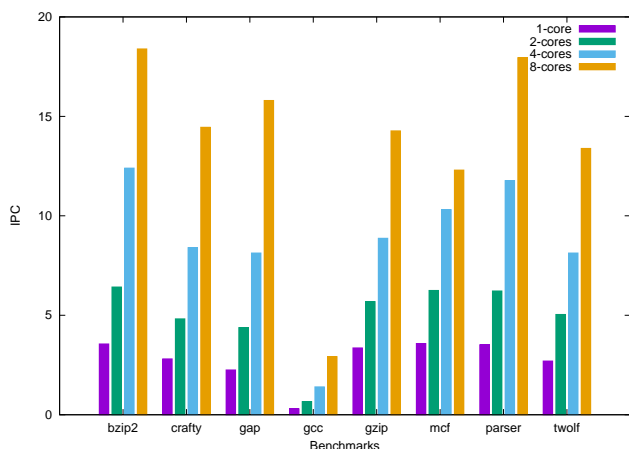
- [8] —, "Dynamic branch prediction with perceptrons," in *High Performance Computer Architecture*, Jun 2001, pp. 197–206.
- [9] D. A. Jimenez, "Fast path-based neural branch prediction," in *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, Dec 2003, pp. 243–252.
- [10] T. N. Vijaykumar and G. S. Sohi, "Task selection for a multiscalar processor," in *31st International Symposium on Microarchitecture*, Dec 1998, pp. 81–92.
- [11] T.-Y. Yeh and Y. Patt, "Alternative Implementations of Two-Level Adaptive Branch Prediction," in *Proceedings of the 19th International Symposium on Computer Architecture*, May. 1992, pp. 124–134.
- [12] J. Gummaraju and M. Franklin, "Branch prediction in multi-threaded processors," in *Parallel Architectures and Compilation Techniques*, Oct 2000, pp. 179–188.
- [13] J. Lee, "The study of statistical simulation for multicore processor architectures," in *The Sixth International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking*, Mar 2015, pp. 27–30.
- [14] T. Austin, E. Larson, and D. Ernest, "SimpleScalar : An Infrastructure for Computer System Modeling," *Computer*, vol. 35, no. 2, Feb. 2002, pp. 59–67.
- [15] A. Rico, A. Duran, F. Cabarcas, A. Ramirex, and M. Valero, "Trace-driven simulation of multithreaded applications," in *ISPASS*, Apr 2011, pp. 87–96.



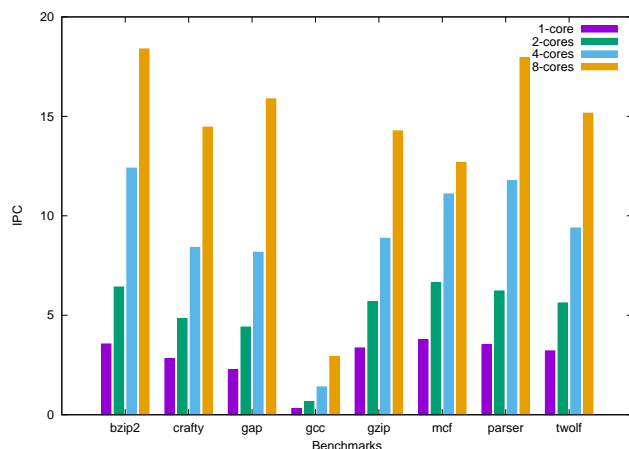
(a) two-level adaptive, maximum task length of 4



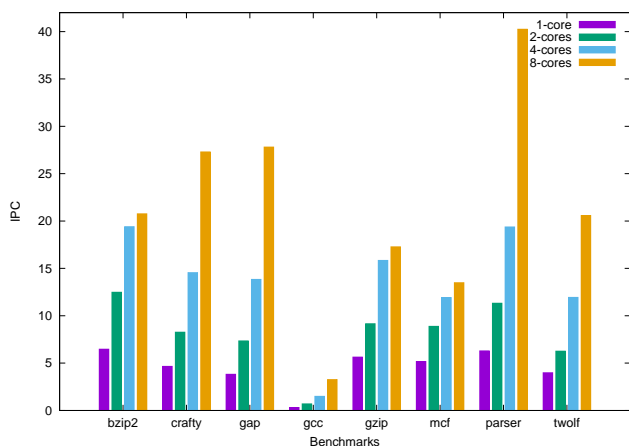
(b) perceptron, maximum task length of 4



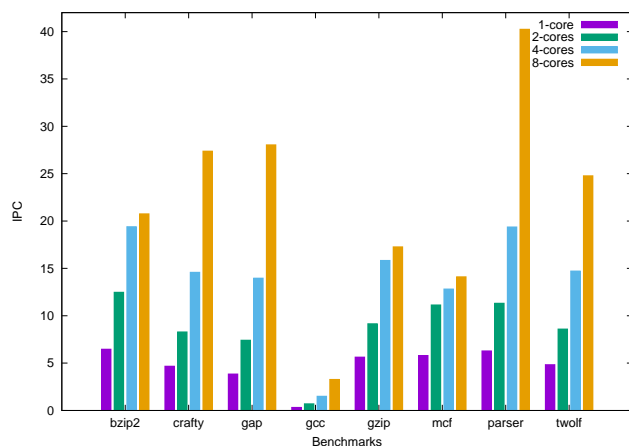
(c) two-level adaptive, maximum task length of 8



(d) perceptron, maximum task length of 8



(e) two-level adaptive, maximum task length of 16



(f) perceptron, maximum task length of 16

Figure 6. Performance results of the two-level adaptive and the perceptron-based task predictor

# Implementing the Type System for a Typed Javascript and its IDE

Lorenzo Bettini

Jens von Pilgrim, Mark-Oliver Reiser

Dip. Statistica, Informatica, Applicazioni  
Università di Firenze, Italy

NumberFour AG,  
Berlin, Germany

Email: [bettini@disia.unifi.it](mailto:bettini@disia.unifi.it) Email: [{jens.von.pilgrim,mark-oliver.reiser}@numberfour.eu](mailto:{jens.von.pilgrim,mark-oliver.reiser}@numberfour.eu)

**Abstract**—Implementing a programming language with IDE tooling features poses several challenges even when using language workbenches like Xtext that provides Eclipse integration. A complex type system with powerful type inference mechanisms requires focusing carefully on performance issues that might undermine the effective usability of the IDE: the editor must be responsive even when type inference takes place in the background, otherwise the programmer will experience too many lags. In this paper, we will present a real-world case study: N4JS, a JavaScript dialect with a full-featured Java-like static type system (including generics) and present some evaluation results. We will concentrate on techniques to make the type system implementation of N4JS integrate efficiently with Eclipse. For the implementation of such a type system we use Xsemantics, a DSL for writing type systems, reduction rules and in general relation rules for languages implemented in Xtext. Xsemantics uses a syntax that resembles formal type system specifications, so that the implementation of formally defined type rules can be implemented easier and more directly than in Java.

**Keywords**—DSL; Type System; Implementation; Eclipse.

## I. INTRODUCTION

Xtext [1] is a popular Eclipse framework for the development of Domain-Specific Languages (DSLs) and their Integrated Development Environments (IDEs). The type system and interpreter for a language implemented in Xtext are usually implemented in Java. While this works for languages with a simple type system, it becomes a problem for an advanced type system. Since the latter is often formalized, a DSL enabling the implementation of a type system similar to the formalization would be useful. Besides functional aspects, implementing a complex type system with powerful type inference mechanisms poses several challenges due to performance issues. At the same time, modern statically-typed languages tend to reduce the verbosity of the syntax with respect to types by implementing type inference systems that relieve the programmer from the burden of declaring types when these can be inferred from the context. In order to be able to cope with these high demands on both type inference and performance, efficiently implemented type systems are required.

In [2], Xsemantics [3] was introduced. Xsemantics is a DSL for writing rules for languages implemented in Xtext, e.g., the type system, the operational semantics and the subtyping. Given the type system specification, Xsemantics generates Java code that can be used in the Xtext implementation. Xsemantics specifications have a declarative flavor that resembles formal systems, while keeping the Java-like shape. This makes it usable both by formal theory people and by Java programmers. Xsemantics has improved a lot in order to make it usable

for modern full-featured languages and real-world performance requirements. The new and advanced features of Xsemantics are presented in [4].

In this paper, we present the implementation in Xsemantics of the type system of N4JS, a version of JavaScript implemented with Xtext, with powerful type inference mechanisms (including Java-like generics). The implementation of the type system of N4JS focuses both on the performance of the type system and on its integration in the Eclipse IDE. This is the first real-world example of the applicability of Xsemantics for a complex type system with involved type inference.

The paper is structured as follows. We provide a small introduction to Xtext and Xsemantics in Section II. In Section III, we present our main case study: the implementation of the type system of N4JS with Xsemantics, with some performance benchmarks related to the type system. Section IV concludes the paper and discusses some related works.

## II. XTEXT AND XSEMANTICS

In this section, we will briefly recall the main features of Xtext and Xsemantics.

Xtext [1] is a *language workbench* and it deals not only with the compiler mechanisms but also with Eclipse-based tooling: Xtext generates the Eclipse editor for the language that we are implementing with syntax highlighting, background parsing with error markers, outline view and code completion. In the following we describe the two complementary mechanisms of Xtext that the programmer has to implement for the type checking. Xsemantics aims at generating code for both mechanisms. *Scoping* is the mechanism for binding the symbols (i.e., references). Xtext supports the customization of binding with the abstract concept of *scope*, i.e., all declarations that are available (visible) in the current context of a reference. The programmer provides a `ScopeProvider` to customize the scoping. In Java-like languages the scoping will have to deal with types and inheritance relations, thus, it is strictly connected with the type system. All the other checks that do not deal with symbol resolutions, have to be implemented through a *validator*. In a Java-like language most validation checks typically consist in checking that the program is correct with respect to types. The validation takes place in background while the user is writing in the editor, so that an immediate feedback is available.

A system definition in Xsemantics is a set of *judgments* (formally, assertions about the properties of programs) and a set of *rules* (formally, implications between judgments). Rules have a conclusion and a set of premises. Rules can act on any

```

judgments {
  type |- Expression expression : output Type
  error "cannot type " + expression
  subtype |- Type left <: Type right
  error left + " is not a subtype of " + right
}

```

Figure 1. Judgment definitions in Xsemantics.

Java object. Typically, rules act on the objects representing the Abstract Syntax Tree (AST). Starting from the definitions of judgments and rules, Xsemantics generates Java code that can be used in a language implemented in Xtext for scoping and validation.

An Xsemantics judgment consists of a name, a *judgment symbol* (which can be chosen from some predefined symbols) and the *parameters* of the judgment. Parameters are separated by *relation symbols* (which can be chosen from some predefined symbols). Two judgments must differ for the judgment symbol or for at least one relation symbol. The parameters can be either input parameters (using the same syntax for parameter declarations as in Java) or output parameters (using the keyword `output` followed by the Java type). For example, the judgment definitions for an hypothetical Java-like language are shown in Figure 1: the judgment `type` takes an `Expression` as input parameter and provides a `Type` as output parameter. The judgment `subtype` does not have output parameters, thus its output result is implicitly boolean. Judgment definitions can include `error` specifications which are useful for generating informative error information.

Rules implement judgments. Each rule consists of a name, a *rule conclusion* and the *premises* of the rule. The conclusion consists of the name of the *environment* of the rule, a *judgment symbol* and the *parameters* of the rules, which are separated by *relation symbols*. To enable better IDE tooling and a more “programming”-like style, Xsemantics rules are written in the opposite direction of standard deduction rules, i.e., the conclusion comes before the premises (similar to other frameworks like [5], [6]).

The elements that make a rule belong to a specific judgment are the judgment symbol and the relation symbols that separate the parameters. Moreover, the types of the parameters of a rule must be Java subtypes of the corresponding types of the judgment. Two rules belonging to the same judgment must differ for at least one input parameter’s type. This is a sketched example of a rule, for a Java-like method invocation expression, of the judgment `type` shown in Figure 1:

```

rule MyRule
  G |- MethodSelection exp : Type type
  from {
    // premises
    type = ... // assignment to output parameter
  }

```

The rule *environment* (in formal systems it is usually denoted by  $\Gamma$  and, in the example it is named `G`) is useful for passing additional arguments to rules (e.g., contextual information, bindings for specific keywords, like **this** in a Java-like language). An empty environment can be passed using the keyword **empty**. The environment can be accessed with the predefined function **env**.

Xsemantics uses Xbase [7] to provide a rich Java-like syntax for defining rules. Xbase is a reusable expression language that integrates tightly with Java, its type system

and Eclipse Java Development Tools (JDT). The syntax of Xbase is similar to Java with less “syntactic noise” and some advanced linguistic constructs. Xbase provides *extension methods*, a syntactic sugar mechanism: instead of passing the first argument inside the parentheses of a method invocation, the method can be called with the first argument as its receiver. Xbase also provides *lambda expressions*, which have the shape `[ param1, param2, ... | body ]`. Xbase’s lambda expressions together with extension methods allow to easily write statements and expressions which are not only more readable than in Java, but they are also very close to formal specifications.

The premises of a rule, which are specified in a **from** block, can be any Xbase expression, or a *rule invocation*. The premises of an Xsemantics rule are considered to be in *logical and* relation and are verified in the same order they are specified in the block. If one needs premises in *logical or* relation, the operator **or** must be used to separate blocks of premises. If a rule does not require any premise, we can use a special kind of rule, called *axiom*, which only has the conclusion. In the premises, one can assign values to the output parameters. When another rule is invoked, upon return, the output arguments will have the values assigned in the invoked rule. If one of the premises fails, then the whole rule will fail, and in turn the stack of rule invocations will fail. In particular, if the premise is a boolean expression, it will fail if the expression evaluates to false. If the premise is a rule invocation, it will fail if the invoked rule fails. An explicit failure can be triggered using the keyword **fail**. At runtime, upon rule invocation, the generated Java system will select the most appropriate rule according to the runtime types of the passed arguments. Note that, besides this strategy for selecting a specific rules, Xsemantics itself does not implement, neither it defines, any other strategy.

Besides judgments and rules, one can write *auxiliary functions*. In type systems, such functions are typically used as a support for writing rules in a more compact form, delegating some tasks to such functions. *Predicates* can be seen as a special form of auxiliary functions. In an Xsemantics system, we can specify some special rules, *checkrule*, which do not belong to any judgment. They are used by Xsemantics to generate a Java validator for the Xtext language. A checkrule has a name, a single parameter (which is the AST object to be validated) and the premises (but no rule environment). The syntax of the premises of a checkrule is the same as in the standard rules. In an Xsemantics system, fields can be defined, which will be available to all the rules, checkrules and auxiliary functions, just like Java fields in a class are available to all methods of the class. This makes it easier to reuse external Java utility classes from an Xsemantics system. This is useful when some mechanisms are easier to implement in Java than in Xsemantics. Custom error information can be specified on judgments, rules and auxiliary functions. This can be used for providing useful error information. Moreover, when using the explicit failure keyword **fail**, a custom error information can be specified as well. This use of **fail** is useful together with *or* blocks to provide more information about the error. Moreover, in *or* blocks, the implicit variable `previousFailure` is available. This allows us to build informative error messages as shown in Section III-B.

In a language implemented with Xtext, types are used in

many places by the framework, e.g., in the scope provider, in the validator and in the content assist. For the above reasons, the results of type computations should be cached to improve the performance of the compiler and, most of all, the responsiveness of the Eclipse editor. However, caching usually introduces a few levels of complexity in implementations, and, in the context of an IDE that performs background parsing and checking, we also need to keep track of changes that should invalidate the cached values. Xsemantics provides automatic caching mechanisms that can be enabled in a system specification. The cached values will be automatically discarded as soon as the contents of the program changes.

### A. Related Work

Xsemantics can be considered the successor of Xtypes [8]. With this respect, Xsemantics provides a much richer syntax for rules that can access any existing Java library. In Xsemantics, a system can extend an existing one (adding and overriding rules). However, these extensibility and compositionality features are not as powerful as the ones of other frameworks such as, e.g., [9], [10], [11].

There are other tools for implementing DSLs and IDE tooling (see [12], [13] for a wider comparison). Spoofox [10], another language workbench which targets Eclipse, relies on Stratego [14] for rule-based specifications. Xtext Type System (XTS) [15] is a DSL for specifying type systems for DSLs built with Xtext. The main difference with respect to Xsemantics is that XTS aims at expression based languages, not at general purpose languages. EriLex [16] supports specifying syntax, type rules, and dynamic semantics but no IDE tooling.

An Xsemantics specification can access any Java type, not only the ones representing the AST. Thus, Xsemantics might also be used to validate any model, independently from Xtext itself, and possibly be used also with other language frameworks like EMFText [17]. Other approaches, such as, e.g., [11], [16], [18], [19], [20], [21], [22], instead require the programmer to use the framework also for defining the syntax of the language.

The importance of targeting IDE tooling was recognized also in older frameworks, such as *Synthesizer* [23] and *Centaur* [18] (the latter was using several formalisms [24], [25], [26]). Finally, we just mention other tools for the implementation of DSLs that are different from Xtext and Xsemantics for the main goal and programming context, such as, [20], [21], [22], [27], [28], [29], [30], [31].

## III. CASE STUDY

In this section we will describe our real-world case study: the implementation of the type system for a JavaScript dialect with a full-featured static type system implemented with Xsemantics. We will also describe some performance benchmarks related to the type system and draw some evaluations.

### A. N4JS—Typed JavaScript

We have used Xsemantics to implement the type system of a real-world language called N4JS. N4JS is a super set of JavaScript also known as ECMAScript with modules and classes as proposed in [32]. Most importantly N4JS adds a full-featured static type system on top of JavaScript, similar to TypeScript [33] or Dart [34]. N4JS is still under development, but it is already being used internally at NumberFour AG.

```
function f(A p) {
  var pNotNull = p || {name: "default", age: 42};
  ...
}
```

Figure 2. Typical usage of union types in N4JS

```
class A {
  f(union{B,C} p) {
    if (p instanceof B) { f_B(p) }
    else { f_C(p) }
  }
}
```

Figure 3. Union types used for emulated method overloading in N4JS

Moreover, it will be made available as an open source project in the near future.

Roughly speaking, N4JS' type system could be described as a combination of the type systems provided by Java, TypeScript and Dart. Besides primitive types, already present in ECMAScript, it provides declared types such as classes and interfaces, also supporting default methods (i.e., mixins), and combined types such as union types [35]. N4JS supports generics similar to Java or TypeScript, that is, it supports generic types and generic methods (which are supported by TypeScript but not by Dart) including wildcards, requiring the notion of existential types (see [36]). The syntax of type expressions is similar to Java's type expressions as far as possible.

Union types are an important feature in typed ECMAScript-related languages. For example, logical operators do not return a single boolean value in ECMAScript. This is often used in JavaScript programs in order to avoid null checks as demonstrated in Figure 2. The type of `pNotNull` is to be inferred as the union type of `A` and the object literal with a property "name" of type `string` and a property "age" of type `number`. Union types can also be used as a technique to emulate method overloading, which is not directly supported by ECMAScript, as shown in Figure 3.

Functions are first-class citizens in the ECMAScript language, which is reflected by the notion of *function types* in N4JS. In combination with generic methods, function expressions and type inference, this becomes a convenient and powerful feature, as shown in Figure 4. Note that the method call requires a lot of type inference capabilities: firstly, the type variable `T` has to be substituted correctly with `A`; secondly, the signature of the function expression has to be inferred from the formal parameter's type, taking the type variable substitution into account.

In Figure 5, we show the N4JS Eclipse editor in action. Note that the type system correctly inferred the type of the invoked method and instantiated the type parameter according

```
function <T> exists(Array<T> list,
  {function(T p):boolean} predicate): boolean { ... }

function existsJohn (Array<A> list): boolean {
  return exists ( list ,
    function(p) { return p.name == "John" }
  );
}
```

Figure 4. Generic method call with function expression and type inference

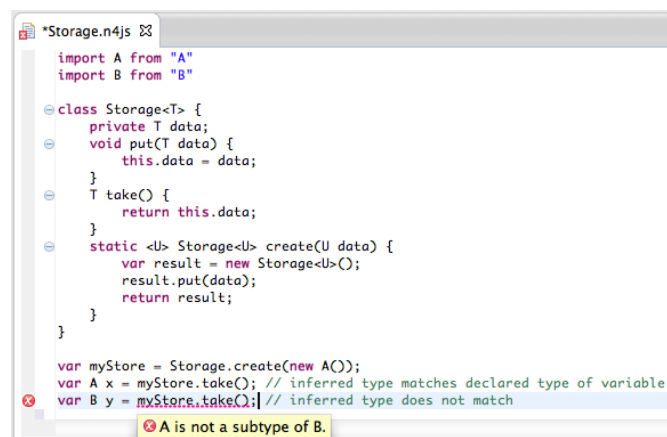


Figure 5. The N4JS editor and its type inference in action.

```

rule subtypeRefUnionOther
  G |- UnionTypeExpression U <: TypeRef S
from {
  U.typeRefs.forall[T | G |- T <: S]
}

rule subtypeRefOtherUnion
  G |- TypeRef S <: UnionTypeExpression U
from {
  U.typeRefs.exists[T | G |- S <: T]
}

```

Figure 6. N4JS union types implemented with Xsemantics.

to the argument passed to the generic method `create`.

### B. Type System

The Xsemantics based type system is not only used for validation purposes, but also for implementing the scoping (see Section II), e.g., in order to find the correct method in case of overridden methods. The whole type system of N4JS is modeled by means of Xsemantics judgments, implemented by approximately 30 axioms and 80 rules. Since type inference rules can be implemented almost 1:1 with Xsemantics, many rules are simple adaptations of rules described in the aforementioned papers. For example, the subtype relation for union types is implemented with the rules shown in Figure 6. Note that we use many Xbase features, e.g., lambda expressions and extension methods (described in Section II).

In the implementation of the N4JS type system in Xsemantics we made a heavy use of the rule environment. We are using it not only to pass contextual information to the rules, but also to store basic types that have to be globally available to all the rules of the type system (e.g., boolean, integer, etc.). This way, we can safely make the assumption that such type instances are singletons in our type system, and can be checked using the standard Java object equality. To make the type system more readable, we implemented some static methods in a separate Java class `RuleEnvironmentExtensions`, and we imported such methods as extension methods in the Xsemantics system:

```
import static extension RuleEnvironmentExtensions.*
```

These methods are used to easily access global type instances from the rule environment, as it is shown, for example, in the rule of Figure 7.

```

rule typeUnaryExpression
  G |- UnaryExpression e : TypeRef T
from {
  switch (e.op) {
  case UnaryOperator.DELETE: T= G.booleanTypeRef ()
  case UnaryOperator.VOID: T= G.undefinedTypeRef ()
  case UnaryOperator.TYPEOF: T= G.stringTypeRef ()
  case UnaryOperator.NOT: T= G.booleanTypeRef ()
  default: // INC, DEC, POS, NEG, INV
    T = G.numberTypeRef ()
  }
}

```

Figure 7. Typing of unary expression.

```

rule typeConditionalExpression
  G |- ConditionalExpression expr : TypeRef T
from {
  G |- expr.trueExpression : var TypeRef left
  G |- expr.falseExpression : var TypeRef right
  T = G.createUnionType (left, right)
}

```

Figure 8. Typing of conditional expression.

Other examples are shown in Figure 8 and 9. In particular, these examples also show how Xsemantics rules are close to the formal specifications. We believe they are also easy to read and thus to maintain.

Since the type system of N4JS is quite involved, creating useful and informative error messages is crucial to make the language usable, especially in the IDE. We have 3 main levels of error messages in the implementation: 1) default error messages defined on judgment declaration, 2) custom error messages using `fail`, 3) customized error messages due to failed nested judgments using `previousFailure` (described in Section II). Custom error messages are important especially when checking subtyping relations. For example, consider checking something like `A<string> <: A<number>`. The declared types are identical (i.e., A), so the type arguments have to be checked. If we would not catch and change the error message produced by the nested subtype checks `string <: number` and `number <: string`, then the error message would be very confusing for the user, because it only refers to the type arguments. In cases where the type arguments are explicitly given, this might be rather obvious, but that is not the case when the type arguments are only defined through type variable bindings or can change due to considering the upper/lower bound. Some examples of error messages due to subtyping are shown in Figure 10.

### C. Performance

N4JS is used to develop large scale ECMAScript applications. For this purpose, N4JS comes with a compiler, performing all validations and eventually transpiling the code to plain ECMAScript. We have implemented a test suite in

```

rule typeArrayLiteral
  G |- ArrayLiteral al : TypeRef T
from {
  val elementTypes = al.elements.map[
  elem |
  G |- elem : var TypeRef elementType;
  elementType;
  ]
  T = G.arrayType.createTypeRef (G.createUnionType (elementTypes) )
}

```

Figure 9. Typing of array literal expression.



```

class A {
}

class B {
}

class List<T> {}

class Triple<S,T,U> {}

// nested error is *not* used at all
var List<A> l = new List<B>();
// nested error is used and adjusted
var Triple<A,A,A> t = new Triple<A,B,A>();
    
```

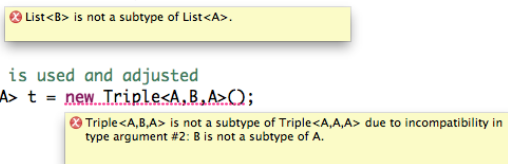


Figure 10. The N4JS IDE and error reporting.

```

// Scenario 1: function expression
function f ({function (C): A} func) { ... };
f( function (C p): A { return p.getA() || new A(); } ) // typed
f( function (p) { return p.getA() || new A(); } ) // inferred

// Scenario 2: generic method call
function <T> g (T p): T { ... }
var s1 = <string>g(""); // typed
var s2 = g(""); // inferred

// Scenario 3: variable declarations and references
var number y1 = 1; // typed
var number y2 = y1; ...
var x1 = 1; // inferred
var x2 = x1; var x3 = x2; ...
    
```

Figure 11. Scenario snippets used in performance tests

order to measure the performance of the type system. Since we want to be able to measure the effect on performance of specific constructs, we use synthetic tests with configured scenarios. In spite of being artificial, these scenarios mimic typical situations in Javascript programming. There are several constructs and features which are performance critical, as they require a lot of type inference (which means a lot of rules are to be called). We want to discuss three scenarios in detail, Figure 11 summarizes the important code snippets used in these scenarios.

*Function Expression:* Although it is possible to specify the types of the formal parameters and the return type of functions, this is very inconvenient for function expressions. The function definition  $f$  (Figure 11) is called in the lines below the definition. Function  $f$  takes a function as argument, which itself requires a parameter of type  $C$  and returns an  $A$  element. Both calls (below the definition) use function expressions. The first call uses a fully typed function expression, while the second one relies on type inference. *Generic Method Calls:* As in Java, it is possible to explicitly specify type arguments in a call of a generic function. Similar to type expressions, it is more convenient to let the type system infer the type arguments, which actually is a typical constraint resolution problem. The generic function  $g$  (Figure 11) is called one time with explicitly specified type argument, and one time without type arguments. *Variable Declarations:* The type of a variable can either be explicitly declared, or it is inferred from the type of the expression used in an assignment. This scenario

TABLE I. PERFORMANCE MEASUREMENTS (RUNTIME IN MS)

Scenario	size	without caching		with caching	
		typed	inferred	typed	inferred
Function Expressions					
	250	875	865	772	804
	500	1,860	1,797	1,608	1,676
	1000	4,046	3,993	3,106	3,222
	2000	9,252	9,544	8,143	8,204
Generic Method Calls					
	250	219	273	223	280
	500	566	644	548	654
	1000	1,570	1,751	1,935	1,703
	2000	6,143	6,436	6,146	6,427
Variable Declarations					
	50	19	580	18	39
	100	27	3,848	26	102
	200	44	31,143	36	252

demonstrates why caching is so important: without caching, the type of  $x1$  would be inferred three times. Of course, this is not the case if the type of the variable is declared explicitly.

Table I shows some performance measurements, using the described scenarios to set up larger tests. That is, test files are generated with 250 or more usages of function expressions, or with up to 200 variables initialized following the pattern described above. In all cases, we run the tests with and without caching enabled. Also, for all scenarios we used two variants: with and without declared types. We measure the time required to execute the JUnit tests.

There are several conclusions, which could be drawn from the measurement results. First of all, caching is only worth in some cases, but these cases can make all the difference. The first two scenarios do not gain much from caching, actually the overhead for managing the cache even slightly decreases performance in case of generic methods calls. In many cases, types are to be computed only once. In our example, the types of the type arguments in the method call are only used for that particular call. Thus, caching the arguments there does not make any sense. Things are different for variable declarations. As described above, caching the type of a variable, which is used many times, makes a lot of sense. Increasing the performance by the factor of more than 100 is not only about speeding up the system a little bit—it is about making it work at all for larger programs. Even if all types are declared, type inference is still required in order to ensure that the inferred type is compatible with the declared type. This is why in some cases the fully typed scenario is even slower than the scenario which uses only inferred types. While in some cases (scenario 1 and 3) the performance increases linearly with the size, this is not true for scenario 2, the generic method call. This demonstrates a general problem with interpreting absolute performance measurements: it is very hard to pinpoint the exact location in case of performance problems, as many parts, such as the parser, the scoping system and the type system are involved. Therefore, we concentrate on relative performance between slightly modified versions of the type system implementation (while leaving all other subsystems unchanged).

Summarizing, we learned that different scenarios must be taken into account when working on performance optimization,

in order to make the right decision about whether using caching or not. Surely, when type information is reused in other parts of the program over and over again, like in the variable scenario, caching optimization is crucial. Combining the type system with control flow analysis, leading to effect systems, may make caching dispensable in many cases. Further investigation in this direction is ongoing work.

#### IV. CONCLUSIONS

In this paper, we presented the implementation in Xsemantics of the type system of N4JS, a statically typed JavaScript, with powerful type inference mechanisms, focusing both on the performance of the type system and on its integration in the Eclipse IDE. The N4JS case study proved that Xsemantics is mature and powerful enough to implement a complex type system of a real-world language.

Thanks to Xtext, Xsemantics offers a rich Eclipse tooling, including the debugger for Xsemantics rule definitions. These features are extremely important for the effective usability of Xsemantics, especially in complex type systems like N4JS' one. With respect to manual implementations of type systems in Java, Xsemantics specifications are more compact and closer to formal systems. We also refer to [37] for a wider discussion about the importance of having a DSL for type systems in language frameworks. In particular, Xsemantics integration with Java allows the developers to incrementally migrate existing type systems implemented in Java to Xsemantics [38].

#### ACKNOWLEDGMENT

The first author was partially supported by itemis Schweiz, MIUR (proj. CINA), Ateneo/CSP (proj. SALT), and ICT COST Action IC1201 BETTY. We also want to thank Sebastian Zarnekow, Jörg Reichert and the colleagues at NumberFour, in particular Jakob Siberski and Torsten Krämer, for feedback and for implementing N4JS with us.

#### REFERENCES

- [1] L. Bettini, *Implementing Domain-Specific Languages with Xtext and Xtend*. Packt Publishing, 2013.
- [2] —, “Implementing Java-like languages in Xtext with Xsemantics,” in *OOPS (SAC)*. ACM, 2013, pp. 1559–1564.
- [3] —, “Xsemantics,” <http://xsemantics.sf.net>, 2016, accessed: 2016-01-07.
- [4] —, “Implementing Type Systems for the IDE with Xsemantics,” *Journal of Logical and Algebraic Methods in Programming*, 2016, to Appear.
- [5] E. Visser et al, “A Language Designer’s Workbench: A One-Stop-Shop for Implementation and Verification of Language Designs,” in *Onward!* ACM, 2014, pp. 95–111.
- [6] V. A. Vergu, P. Neron, and E. Visser, “DynSem: A DSL for Dynamic Semantics Specification,” in *RTA*, ser. LIPIcs, vol. 36. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015, pp. 365–378.
- [7] S. Efftinge et al, “Xbase: Implementing Domain-Specific Languages for Java,” in *GPCE*. ACM, 2012, pp. 112–121.
- [8] L. Bettini, “A DSL for Writing Type Systems for Xtext Languages,” in *PPPJ*. ACM, 2011, pp. 31–40.
- [9] T. Ekman and G. Hedin, “The JastAdd system – modular extensible compiler construction,” *Science of Computer Programming*, vol. 69, no. 1-3, 2007, pp. 14 – 26.
- [10] L. C. L. Kats and E. Visser, “The Spoofox language workbench. Rules for declarative specification of languages and IDEs,” in *OOPSLA*. ACM, 2010, pp. 444–463.
- [11] E. Vacchi and W. Cazzola, “Neverlang: A Framework for Feature-Oriented Language Development,” *Computer Languages, Systems & Structures*, vol. 43, no. 3, 2015, pp. 1–40.
- [12] M. Voelter et al, *DSL Engineering - Designing, Implementing and Using Domain-Specific Languages*, 2013.
- [13] M. Pfeiffer and J. Pichler, “A comparison of tool support for textual domain-specific languages,” in *Proc. DSM*, 2008, pp. 1–7.
- [14] M. Bravenboer, K. T. Kalleberg, R. Vermaas, and E. Visser, “Stratego/XT 0.17. A language and toolset for program transformation,” *Science of Computer Programming*, vol. 72, no. 1–2, 2008, pp. 52–70.
- [15] M. Voelter, “Xtext/TS - A Typesystem Framework for Xtext,” 2011.
- [16] H. Xu, “EriLex: An Embedded Domain Specific Language Generator,” in *TOOLS*, ser. LNCS, vol. 6141. Springer, 2010, pp. 192–212.
- [17] F. Heidenreich, J. Johannes, S. Karol, M. Seifert, and C. Wende, “Derivation and Refinement of Textual Syntax for Models,” in *ECMDA-FA*, ser. LNCS, vol. 5562. Springer, 2009, pp. 114–129.
- [18] P. Borras et al, “CENTAUR: the system,” in *Software Engineering Symposium on Practical Software Development Environments*, ser. SIGPLAN. ACM, 1988, vol. 24, no. 2, pp. 14–24.
- [19] M. Fowler, “A Language Workbench in Action - MPS,” <http://martinfowler.com/articles/mpsAgree.html>, 2008, accessed: 2016-01-07.
- [20] M. G. J. V. D. Brand, J. Heering, P. Klint, and P. A. Olivier, “Compiling language definitions: the ASF+SDF compiler,” *ACM TOPLAS*, vol. 24, no. 4, 2002, pp. 334–368.
- [21] A. Dijkstra and S. D. Swierstra, “Ruler: Programming Type Rules,” in *FLOPS*, ser. LNCS, vol. 3945. Springer, 2006, pp. 30–46.
- [22] M. Felleisen, R. B. Findler, and M. Flatt, *Semantics Engineering with PLT Redex*. The MIT Press, 2009.
- [23] T. Reps and T. Teitelbaum, “The Synthesizer Generator,” in *Software Engineering Symposium on Practical Software Development Environments*. ACM, 1984, pp. 42–48.
- [24] G. Kahn, B. Lang, B. Melese, and E. Morcos, “Metal: A formalism to specify formalisms,” *Science of Computer Programming*, vol. 3, no. 2, 1983, pp. 151–188.
- [25] E. Morcos-Chounet and A. Conchon, “PPML: A general formalism to specify prettyprinting,” in *IFIP Congress*, 1986, pp. 583–590.
- [26] T. Despeyroux, “Typol: a formalism to implement natural semantics,” *INRIA*, Tech. Rep. 94, Mar. 1988.
- [27] D. Batory, B. Lofaso, and Y. Smaragdakis, “JTS: Tools for Implementing Domain-Specific Languages,” in *ICSR*. IEEE, 1998, pp. 143–153.
- [28] M. Bravenboer, R. de Groot, and E. Visser, “MetaBorg in Action: Examples of Domain-Specific Language Embedding and Assimilation Using Stratego/XT,” in *GTTSE*, ser. LNCS, vol. 4143. Springer, 2006, pp. 297–311.
- [29] H. Krahn, B. Rumpe, and S. Völkel, “Monticore: a framework for compositional development of domain specific languages,” *STTT*, vol. 12, no. 5, 2010, pp. 353–372.
- [30] T. Clark, P. Sammut, and J. Willans, *Superlanguages, Developing Languages and Applications with XMF*, 1st ed. Ceteva, 2008.
- [31] L. Renggli, M. Denker, and O. Nierstrasz, “Language Boxes: Bending the Host Language with Modular Language Changes,” in *SLE*, ser. LNCS, vol. 5969. Springer, 2009, pp. 274–293.
- [32] “Draft ECMA Script Language Specification,” ISO/IEC, Working Draft ECMA-262, 6th Edition, Apr. 2014.
- [33] A. Hejlsberg and S. Lucco, *TypeScript Language Specification*, 1st ed., Microsoft, Apr. 2014.
- [34] Dart Team, *Dart Programming Language Specification*, 1st ed., Mar. 2014.
- [35] A. Igarashi and H. Nagira, “Union types for object-oriented programming,” *Journal of Object Technology*, vol. 6, no. 2, 2007, pp. 47–68.
- [36] N. Cameron, E. Ernst, and S. Drossopoulou, “Towards an Existential Types Model for Java Wildcards,” in *Formal Techniques for Java-like Programs (FTfJP)*, July 2007, pp. 1–17.
- [37] L. Bettini, D. Stoll, M. Völter, and S. Colameo, “Approaches and Tools for Implementing Type Systems in Xtext,” in *SLE*, ser. LNCS, vol. 7745. Springer, 2012, pp. 392–412.
- [38] A. Heiduk and S. Skatulla, “From Spaghetti to Xsemantics - Practical experiences migrating typesystems for 12 languages,” *XtextCon*, <http://xtextcon.org>, 2015.

# Weighted Branching Preorders and Distances: Logical Characterization and Complexity

Louise Foshammer\*, Kim Guldstrand Larsen\*, Radu Mardare\* and Bingtian Xue\*

\*Department of Computer Science, Aalborg University, Denmark

Email: {foshammer,kgl,mardare,bingt}@cs.aau.dk

**Abstract**—We investigate branching bisimulation for weighted transition systems. It is known that branching bisimulation is characterized by computational tree logic without the next operator in the non-deterministic case. We demonstrate that the weighted version of this logic characterizes a weighted version of branching bisimulation, for which the decidability is NP-complete. This leads us to investigating two fragments of this logic allowing only upper bounds and either existential or universal quantification. The resulting existential and universal simulation relations are decidable in polynomial time. We consider distance-based analogues of weighted branching bisimulation and existential simulation and characterize these using fragments of the aforementioned logic.

**Keywords**—Weighted transition systems; Weighted computational tree logic; Characterization; Weighted branching bisimulation.

## I. INTRODUCTION

Classical process algebras, such as CCS, CSP and ACP [1]–[3] provide formalisms for describing the behavior of concurrent and interacting systems essential in terms of labeled transition systems. To capture the semantic equality of processes several behavioral preorders and equivalences have been considered, including the by now classical notion of bisimulation equivalence introduced by Milner [4] and Park [5]. Alongside the development of behavioral equivalences, an overall quest has been identification of corresponding temporal or modal logics, in the sense that the behavioral equivalence between two processes is in complete agreement with equality between the sets of logical properties they satisfy [6], [7].

Another important issue has been identification of behavioral preorders and equivalences that permit internal activities of processes to be abstracted away. The original notion of observational equivalence by Milner [1] serve this precise purpose, as does the later notion of branching bisimulation introduced by Weijland and Van Glabbeek [8]. Branching bisimulation equivalence has the remarkable additional property of being completely characterized by several different and natural modal logics, one of which is computational tree logic (CTL) without the next operator [9].

Whereas labeled transition systems suffice for describing the reactive and functional behavior of processes, they lack information about quantitative and non-functional aspects such as time or resource consumption. This has motivated the introduction and study of weighted transition systems, where transitions are labeled with quantities [10], [11], e.g., real, rational or integer values, allowing for the modeling of consumption or production of resources.

In this paper, we revisit weighted transition systems to identify useful behavioral relationships that are sensitive to quantities while permitting abstraction from internal activities.

As a motivational example consider the following processes  $s$  and  $t$  both ending in the inactive process 0:

$$s \rightarrow_5 0 \text{ and } t \rightarrow_3 t' \rightarrow_2 0$$

Assuming that the states  $s, t, t'$  have the same atomic propositions, the intermediate state  $t'$  may be considered unobservable, and consequently  $s$  and  $t$  may be considered behaviorally equivalent as they end up in 0 with the same overall weight. To capture this situation in more generality, we extend in various ways the idea of branching bisimulation and simulation with weights. Aiming at extending [9], we consider a weighted version of CTL [12] without the next operator with the purpose of identifying interesting fragments and the various weighted versions of branching bisimulation and simulation they characterize. We allow for the systems to have reals as their weights, but the logic can only have rationals in the parameters, it is notable that we are, however, still able to capture the entire behavior of the systems. The study of those fragments are of importance because their weighted simulations are decidable in polynomial time (in contrast to the NP-complete decidability of the full logic) while the logics can still specify interesting properties. This is essential for developing efficient tools.

Finally, we consider weighted branching bisimulation distances. Consider that the process  $s$  has a slightly perturbed weighted transition, e.g.,  $s \rightarrow_{5+\epsilon} 0$ . Then,  $s$  and  $t$  are expected no longer to be weighted branching bisimilar. However, following the recent trends in replacing equivalences with metrics, and boolean answers with quantities [13]–[16], we shall introduce and logically characterize notions of weighted branching distance such that the distance between  $s$  and  $t$  will decrease for decreasing values of  $\epsilon$ .

The structure of this paper will be as follows; in Section II, we introduce the preliminaries, Sections III–V each introduce a weighted branching (bi)simulation variant, identify a characterizing fragment of WCTL (without next) and determine its complexity. Section VI introduces distances and finally Section VII concludes the paper and describes future work.

## II. PRELIMINARIES

A weighted Kripke structure (WKS) is a straightforward extension of Kripke structures, where weights, in the form of non-negative reals, are added to each transition.

**Definition 1** (Weighted Kripke Structure). A *weighted Kripke structure* is a tuple  $\mathcal{K} = (S, \mathcal{AP}, \mathcal{V}, \rightarrow)$ , where  $S$  is a set of states,  $\mathcal{AP}$  is a set of atomic propositions,  $\mathcal{V} : S \rightarrow \mathcal{P}(\mathcal{AP})$  is a mapping from states to sets of atomic propositions and  $\rightarrow \subseteq S \times \mathbb{R}_{\geq 0} \times S$  is a labeled transition relation.  $\star$

For simplicity, transitions are denoted by  $s \rightarrow_w s'$  instead of  $(s, w, s') \in \rightarrow$ .

We say that a WKS  $\mathcal{K} = (S, \mathcal{AP}, \mathcal{V}, \rightarrow)$  is *finite*, if  $S$  and  $\rightarrow$  are finite; *non-blocking*, if for all states  $s \in S$  there is at least one transition  $(s, w, s') \in \rightarrow$  that starts in  $s$  and *rational* if all weights on transitions belong to  $\mathbb{Q}_{\geq 0}$ .

*Example 1.* In Figure 1(a), we show the WKS  $\mathcal{K} = (S, \mathcal{AP}, \mathcal{V}, \rightarrow)$ , where  $S = \{s^1, s_1^1, s_2^1\}$ ,  $\mathcal{AP} = \{p, q\}$ ,  $\rightarrow$  is defined as  $s^1 \rightarrow_3 s^1, s^1 \rightarrow_5 s_1^1, s^1 \rightarrow_2 s_2^1$  and  $\mathcal{V}(s^1) = \{p\}$ ,  $\mathcal{V}(s_1^1) = \{q\}$  and  $\mathcal{V}(s_2^1) = \{q\}$ .

Note that  $\mathcal{K}$  is finite and rational, since it has a finite number of states, a finite number of transitions and all weights are rational. It is, however, not non-blocking, since there are no transitions from either of the states  $s_1^1$  and  $s_2^1$ .  $\diamond$

To specify properties of WKSs, we introduce a weighted extension of CTL (WCTL), where intervals are introduced on the next and the until operators.

*Definition 2* (Syntax of WCTL). Let  $\mathcal{AP}$  be a set of atomic propositions. The syntax of WCTL is given by

$$\phi ::= p \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid EX_I\phi \mid AX_I\phi \\ \mid E(\phi_1 U_I \phi_2) \mid A(\phi_1 U_I \phi_2),$$

where  $p \in \mathcal{AP}$  and  $I = [l, u]$ , where  $l, u \in \mathbb{Q}_{\geq 0}$ ,  $l \leq u$ .  $\star$

Note that  $I$  can be any type of interval and that we allow that  $l = u$ , such that the interval can be a single point.

The semantics of WCTL are given by the satisfiability relation, defined inductively for an arbitrary non-blocking WKS  $\mathcal{K} = (S, \mathcal{AP}, \mathcal{V}, \rightarrow)$  and an arbitrary state  $s \in S$ , as follows

- $\mathcal{K}, s \models p$  iff  $p \in \mathcal{V}(s)$ ,
- $\mathcal{K}, s \models \neg\phi$  iff  $s \not\models \phi$
- $\mathcal{K}, s \models \phi_1 \wedge \phi_2$  iff  $s \models \phi_1$  and  $s \models \phi_2$ ,
- $\mathcal{K}, s \models EX_I\phi$  iff there exists  $s \rightarrow_w s'$ , such that  $w \in I$  and  $s' \models \phi$ ,
- $\mathcal{K}, s \models AX_I\phi$  iff for all  $s \rightarrow_w s'$  such that  $w \in I$ ,  $s' \models \phi$ ,
- $\mathcal{K}, s \models E(\phi_1 U_I \phi_2)$  iff there exists a trace  $s \rightarrow_{w_1} s_1 \rightarrow_{w_2} \dots \rightarrow_{w_k} s_k \rightarrow_{w_{k+1}} \dots$ , such that there exists a state  $s_k$  such that  $s_k \models \phi_2$ , for all  $i < k$ ,  $s_i \models \phi_1$  and  $\sum_{i=1}^k w_i \in I$ ,
- $\mathcal{K}, s \models A(\phi_1 U_I \phi_2)$  iff for all traces  $s \rightarrow_{w_1} s_1 \rightarrow_{w_2} \dots \rightarrow_{w_k} s_k \rightarrow_{w_{k+1}} \dots$ , there exists a state  $s_k$  such that  $s_k \models \phi_2$ , for all  $i < k$ ,  $s_i \models \phi_1$  and  $\sum_{i=1}^k w_i \in I$ .

We use the other Boolean operators with their usual semantics. Consider the following example.

*Example 2.* Return to the WKS  $\mathcal{K} = (S, \mathcal{AP}, \mathcal{V}, \rightarrow)$  in Figure 1(a). The state  $s^1$  satisfies, among others, the following formulae:  $AX_{[1,3]}p$  and  $E(pU_{[4,9]}q)$ , but not for instance  $A(pU_{[0,7]}q)$ , because of the self-loop.  $\diamond$

A well-known way of comparing WKSs is with *weighted bisimulation* [4], [5], which is defined in a way that ensures complete matching of behavior where transitions are matched one-to-one. Weighted bisimulation is completely characterized by WCTL as proven by [17].

Note that in the models, we allow weights to be reals, while in the logic, we only allow rationals. We describe the models as general as possible, but as the logic has to be countable, we will have to restrict ourselves to rationals. The logic is however

still able to encompass the behavior of the model, since we can approximate the reals by rationals.

### III. WCTL WITHOUT NEXT

Let us now look at a fragment of WCTL for which we have removed the next operator. As discussed in the introduction we do not wish to reason about single transition steps in our models, which explains the need to remove the next operator. With the remaining operators we are able to reason about a bound of the cost of arriving at some behavior, while preserving behavior along the way.

*Definition 3* (Syntax of  $WCTL_{-X}$ ). Let  $\mathcal{AP}$  be a set of propositions. The syntax of  $WCTL_{-X}$  is given by

$$\phi ::= p \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid A(\phi_1 U_I \phi_2) \mid E(\phi_1 U_I \phi_2),$$

where  $p \in \mathcal{AP}$  and  $I = [l, u]$ , where  $l, u \in \mathbb{Q}_{\geq 0}$ .  $\star$

The semantics of  $WCTL_{-X}$  is given by the same satisfiability relation as for WCTL.

We will now introduce a notion of weighted branching bisimulation and observe that the bisimulation and the logic induces the same relation on finite WKSs. We allow for a weighted transition to be matched by a sequence of transitions with identical accumulated weight, behavior is preserved in each intermediate state and the end behavior is the same.

*Definition 4.* Given a WKS  $\mathcal{K} = (S, \mathcal{AP}, \mathcal{V}, \rightarrow)$  a *weighted branching bisimulation* (WBB) is a relation  $R \subseteq S \times S$ , such that whenever  $(s, t) \in R$

- $\mathcal{V}(s) = \mathcal{V}(t)$
- for all  $s \rightarrow_w s'$  there exists  $t \rightarrow_{v_1} t_1 \rightarrow_{v_2} \dots \rightarrow_{v_k} t_k$ , such that  $\sum_{i=1}^k v_i = w$ ,  $(s', t_k) \in R$  and for all  $i < k$ ,  $(s, t_i) \in R$
- for all  $t \rightarrow_v t'$  there exists  $s \rightarrow_{w_1} s_1 \rightarrow_{w_2} \dots \rightarrow_{w_k} s_k$ , such that  $\sum_{i=1}^k w_i = v$ ,  $(t', s_k) \in R$  and for all  $i < k$ ,  $(t, s_i) \in R$

If there exists a weighted branching bisimulation relating  $s$  and  $t$ , we say that  $s$  and  $t$  are weighted branching bisimilar and denote it by  $s \approx_{\mathbf{I}} t$ . The relation  $\approx_{\mathbf{I}}$  will henceforth be referred to as weighted branching bisimilarity (WBB).  $\star$

The following theorem shows that  $WCTL_{-X}$  characterizes WKS up to WBB.

*Theorem 1.* Let  $\mathcal{K} = (S, \mathcal{AP}, \mathcal{V}, \rightarrow)$  be a finite WKS. Then for all  $s, t \in S$

$$s \approx_{\mathbf{I}} t \text{ iff } [\forall \phi \in WCTL_{-X} \ s \models \phi \Leftrightarrow t \models \phi].$$

*Proof.* ( $\Rightarrow$ ) Suppose  $s \approx_{\mathbf{I}} t$  and  $s \models \phi$ . Induction on the structure of  $\phi$ .

**The case  $\phi = E(\phi_1 U_I \phi_2)$ :** By definition  $s \models \phi$  iff there exists  $s \rightarrow_{w_1} s_1 \rightarrow_{w_2} \dots \rightarrow_{w_k} s_k \rightarrow \dots$  s.t.  $s_k \models \phi_2$ ,  $\forall i < k$ ,  $s_i \models \phi_1$  and  $\sum_{i=1}^k w_i \in I$ . As  $s \approx_{\mathbf{I}} t$ , we have that for every step  $s_i \rightarrow_{w_{i+1}} s_{i+1}$  there exists  $t^i \rightarrow_{v_1^{i+1}} t_1^{i+1} \rightarrow_{v_2^{i+1}} \dots \rightarrow_{v_{h^{i+1}}^{i+1}} t^{i+1}$  such that  $t^{i+1} \approx_{\mathbf{I}} s_{i+1}$ ,

$\forall j < h^{i+1}$ ,  $t_j^{i+1} \approx_{\mathbf{I}} s_i$  and  $\sum_{j=1}^{h^{i+1}} v_j^{i+1} = w_{i+1}$ . By induction, each state  $t_j^{i+1} \models \phi_1$  for  $j < k$ , the final state  $t^k \models \phi_2$  and  $\sum_{i=1}^k \sum_{j=1}^{h^i} v_j^i = \sum_{i=1}^k w_i$  so by definition,  $t \models \phi$ .

**The case  $\phi = A(\phi_1 U_I \phi_2)$ :** If  $t \not\models \phi$  trivially  $t \not\models \phi$ . Otherwise choose a trace  $\pi_1 = t \rightarrow_{v_1} t_1 \rightarrow_{v_2} \dots$ . For

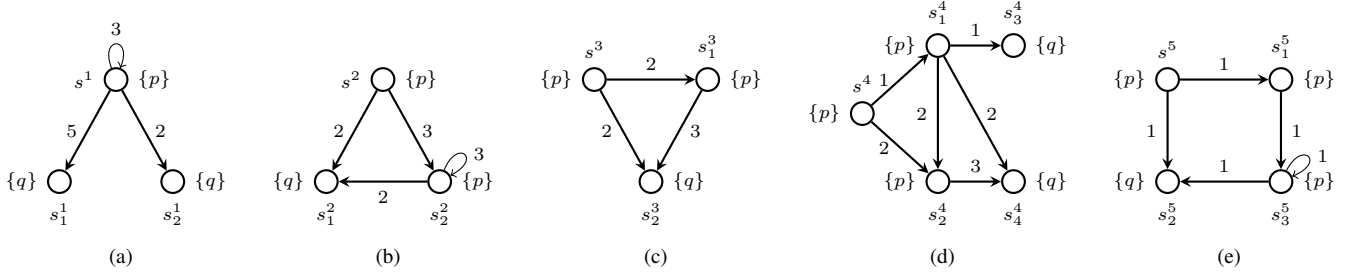


Figure 1. Five WKSs. The following relations are true  $s^1 \approx_1 s^2$ ,  $s^3 \not\approx_1 s^4$ ,  $s^3 \leq_E s^5$ ,  $s^4 \leq_E s^3$ ,  $s^3 \leq_A s^5$ ,  $s^4 \approx_0^1 s^3$  and  $s^3 \approx_1^1 s^4$ .

each step  $t_i \rightarrow_{v_{i+1}} t_{i+1}$  by relation, we have that there exist  $s^i \rightarrow_{w_1^i} s_1^i \rightarrow_{w_2^i} \dots \rightarrow_{w_{k^i}^i} s^{i+1}$ , such that  $t_{i+1} \approx_1 s^{i+1}$ , for all  $j < k^i$   $t_i \approx_1 s_j^i$  and  $\sum_{j=1}^{k^i} w_j^i = v_{i+1}$ . We therefore know that there exists a trace  $\pi_2 = s \rightarrow_{w_0^0} s_1^0 \rightarrow_{w_2^0} \dots \rightarrow_{w_{k_0}^0} s^1 \rightarrow_{w_1^1} \dots \rightarrow_{w_{k^{h-1}}^1} s^h \rightarrow_{w_1^h} \dots$  such that for all  $i$ , we have  $t_i \approx_1 s^i$ , for all  $j < k^i$   $t_i \approx_1 s_j^i$  and  $\sum_{j=1}^{k^i} w_j^i = v_{i+1}$ . Since  $s \models A(\phi_1 U_I \phi_2)$ , we know that for  $\pi_2$  there must exist some  $s^h$  such that  $s^h \models \phi_2$ , for all  $i < h$  and all  $j$ , we have  $s_j^i \models \phi_1$  and  $\sum_{i=0}^{h-1} \sum_{j=1}^{k^i} w_j^i \in I$ . By construction of  $\pi_2$  this means that for  $\pi_1$  there must exist a  $t_h$  such that  $t_h \models \phi_2$ , for all  $i < h$   $t_i \models \phi_1$  and  $\sum_{i=1}^h v_i = \sum_{i=0}^{h-1} \sum_{j=1}^{k^i} w_j^i \in I$ . By definition  $t \models A(\phi_1 U_I \phi_2)$ .

( $\Leftarrow$ ) Define  $(s, t) \in R$  iff  $[\forall \phi \in \text{WCTL}_{-X} s \models \phi \Leftrightarrow t \models \phi]$ . We show that  $R$  is a WBB. Suppose  $s \rightarrow_w s'$  and let  $\pi_i = t \rightarrow_{v_1^i} t_1^i \rightarrow_{v_2^i} \dots \rightarrow_{v_{k^i}^i} t_{k^i}^i$

such that  $\sum_{j=1}^{k^i} v_j^i = w$  be traces out of  $t$  of weight equal to  $w$ . Without loss of generality we can skip all traces with zero-cycles, which means that since the Kripke structure is finite there is only a finite number of traces  $\pi_i$  of weight  $w$ ,  $i = 1, \dots, n$ . Assume that none of these traces match  $s \rightarrow_w s'$ , which means that for each trace  $\pi_i$  either  $(s', t_{k^i}^i) \notin R$  or there exist a  $j < k^i$  such that  $(s, t_j^i) \notin R$ . For each trace  $\pi_i$  such that  $(s', t_{k^i}^i) \notin R$ ,  $i = 1, \dots, k$ ,  $k \leq n$ , there exists a formula  $\psi_i$  such that  $s' \models \psi_i$  and  $t_{k^i}^i \not\models \psi_i$  and for each trace  $\pi_i$  such that  $(s, t_j^i) \notin R$ ,  $i = k+1, \dots, n$ , there exists a formula  $\phi_i$  such that  $s \models \phi_i$  and  $t_j^i \not\models \phi_i$ .

For a decreasing series of rationals  $w^j$  such that  $\lim_{j \rightarrow \infty} w^j = w$  and an increasing series of rationals  $y^j$  such that  $\lim_{j \rightarrow \infty} y^j = w$ , we can create a series of formulae  $\phi^j = E \left( \bigwedge_{i \in [1, k]} \phi_i U_{[y^j, w^j]} \bigwedge_{i \in [k, n]} \psi_i \right)$  for which  $s \models \bigwedge_j \phi^j$ , but  $t \not\models \bigwedge_j \phi^j$ , contradicting  $(s, t) \in R$ . ■

Consider a couple of examples of the use of this theorem.

*Example 3.* Consider Figure 1(a) and 1(b). Let us verify that  $s^1$  and  $s^2$  are WBB. Transition  $s^1 \rightarrow_3 s^1$  and  $s^2 \rightarrow_3 s_2^2$  have to match each other, which means, we have to prove that also  $s^1 \approx_1 s_2^2$ . This is trivially proved, since  $s^1$  and  $s_2^2$  are the same except for the transition  $s^1 \rightarrow_5 s_1^1$ , which is matched by  $s_2^2 \rightarrow_3 s_2^2 \rightarrow_2 s_1^1$ , since  $s_1^1 \approx_1 s_2^2 \approx_1 s_1^1$  trivially. Transitions  $s^1 \rightarrow_2 s_2^1$  and  $s^2 \rightarrow_2 s_1^2$  match each other and transition  $s^1 \rightarrow_5 s_1^1$  is matched by  $s_2^2 \rightarrow_3 s_2^2 \rightarrow_2 s_1^1$ , as we know  $s^1 \approx_1 s_2^2$ .

We therefore know that any formula that is satisfied by either  $s^1$  or  $s^2$  is also satisfied by the other. ◇

*Example 4.* Consider Figure 1(c) and 1(d). We see that  $s^3$  and  $s^4$  are not weighted branching bisimilar, since the transition  $s^4 \rightarrow_1 s_1^4$  cannot be matched from  $s^3$ . This means, we can find a distinguishing formula between these two states. We see that for  $\phi = A(p U_{[2,5]} q)$ ,  $s^3 \models \phi$ , but  $s^4 \not\models \phi$ , since there is a trace  $s^4 \rightarrow_1 s_1^4 \rightarrow_2 s_2^4 \rightarrow_3 s_3^4$  for which the accumulated weight is 6 before it reaches a state that satisfies  $q$ . ◇

We can prove that WBB is a maximal fixed point of a suitable function  $\mathcal{F}_1$  defined as follows.

*Definition 5.* Given a WKS  $\mathcal{K} = (S, \mathcal{AP}, \mathcal{V}, \rightarrow)$ , define a function  $\mathcal{F}_1 : 2^{S \times S} \rightarrow 2^{S \times S}$  such that given a relation  $R \subseteq S \times S$  then  $(s, t) \in \mathcal{F}_1(R)$  if and only if

- $\mathcal{V}(s) = \mathcal{V}(t)$
- for all  $s \rightarrow_w s'$  there exists  $t \rightarrow_{v_1} t_1 \rightarrow_{v_2} \dots \rightarrow_{v_k} t_k$ , such that  $\sum_{i=1}^k v_i = w$ ,  $(s', t_k) \in R$  and for all  $i < k$ ,  $(s, t_i) \in R$
- for all  $t \rightarrow_v t'$  there exists  $s \rightarrow_{w_1} s_1 \rightarrow_{w_2} \dots \rightarrow_{w_k} s_k$ , such that  $\sum_{i=1}^k w_i = v$ ,  $(t', s_k) \in R$  and for all  $i < k$ ,  $(t, s_i) \in R$  ★

The function  $\mathcal{F}_1$  takes in a relation  $R$  that identifies pairs of states believed to be bisimilar and removes all pairs that fail to be bisimilar in one step, when assuming that the states of  $R$  are bisimilar. This means that the resulting relation is not necessarily a bisimulation relation, since the assumption can be wrong. When starting from the assumption that all states are bisimilar, however, and applying  $\mathcal{F}_1$  until stability, i.e. finding a maximal fixed point, we are sure to have WBB.

We now determine the complexity of deciding WBB.

*Theorem 2.* Deciding WBB on finite rational WKSs is NP-hard.

*Proof.* We use the integer knapsack problem, which is well-known to be NP-complete [18], and show that it is polynomial time reducible to the problem of deciding WBB.

In the integer knapsack problem, we are given a finite set  $E$  of elements which each have a value  $v_i \in \mathbb{Z}_{\geq 0}$  and a weight  $w_i \in \mathbb{Z}_{\geq 0}$ . We are further given positive integers  $p$  and  $c$ . Is there an assignment of positive integers  $a_i$  such that  $\sum_i a_i v_i \geq p$  and  $\sum_i a_i w_i \leq c$ ? The integer knapsack problem is still NP-complete if for all  $i$   $v_i = w_i$ , so we want to assign positive integers  $a_i$  such that  $\sum_i a_i w_i = c$ . Let us assume that we have a set  $E = \{e_1, \dots, e_n\}$  of elements with weights (and values)  $\{w_1, \dots, w_n\}$ , and our capacity is  $c$ .

The reduction generates two WKSs, as seen in Figure 2, such that  $\mathcal{K}_1 = (S, \mathcal{AP}, \mathcal{V}, \rightarrow)$ , where  $S = \{s\}$ ,  $\mathcal{AP} = \emptyset$ ,  $\mathcal{V}(s) = \emptyset$  and  $\rightarrow = \{(s, w_1, s), \dots, (s, w_n, s)\}$  and  $\mathcal{K}_2 = (S, \mathcal{AP}, \mathcal{V}, \rightarrow)$ , where  $S = \{t, t'\}$ ,  $\mathcal{AP} = \emptyset$ ,  $\mathcal{V}(t) = \mathcal{V}(t') = \emptyset$  and  $\rightarrow =$

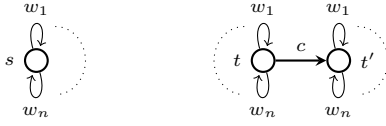


Figure 2. Two WKSs  $\mathcal{K}_1$  and  $\mathcal{K}_2$  that are generated by reduction.

$\{(t, w_1, t), \dots, (t, w_n, t), (t', w_1, t'), \dots, (t', w_n, t'), (t, c, t')\}$ . We demonstrate that there is a solution to the integer knapsack problem if and only if the two states  $s$  and  $t$  are WBB.

( $\Rightarrow$ ) Suppose there exists a set of integers  $\{k_1, \dots, k_n\}$  such that  $\sum k_i w_i = c$ . If we ignore the transition  $t \rightarrow_c t'$  the three states  $s, t$  and  $t'$  are WBB. Hence the only reason why  $s$  and  $t$  should not be WBB is if  $s$  cannot match the transition  $t \rightarrow_c t'$ . But since, there exists a set  $\{k_1, \dots, k_n\}$  such that  $\sum k_i w_i = c$ ,  $s$  can do a series of transitions accordingly and match transition  $t \rightarrow_c t'$ .

( $\Leftarrow$ ) Suppose the transition  $t \rightarrow_c t'$  can be matched by  $s \rightarrow_{w_1} s \rightarrow_{w_2} \dots \rightarrow_{w_k} s$  such that  $\sum w_i = c$ , note that each  $\rightarrow_{w_i}$  can be repeated as many times as necessary. This is equivalent to the existence of a set  $\{k_1, \dots, k_n\}$  such that  $\sum k_i w_i = c$  providing a solution to the integer knapsack problem. ■

We have demonstrated that deciding WBB is at least NP-hard, now let us prove that the problem is contained in NP. We will need the following theorem, which is a reformulated instance of a theorem proved by Nykänen and Ukkonen [19].

**Theorem 3.** Given a WKS  $\mathcal{K} = (S, \mathcal{AP}, \mathcal{V}, \rightarrow)$ ,  $s, t \in S$  and a target cost  $k$ . The problem of deciding whether there exists a trace in  $\mathcal{K}$  from  $s$  to  $t$  of cost exactly  $k$  is NP-complete.

**Theorem 4.** Deciding WBB on a finite rational WKSs is contained in NP.

*Proof.* WBB is the maximal fixed point of  $\mathcal{F}_1$ , so we apply  $\mathcal{F}_1$  repeatedly on  $S \times S$  until the set of bisimilar states stabilizes. Each time  $\mathcal{F}_1^k(S \times S) \neq \mathcal{F}_1^{k-1}(S \times S)$ , meaning at least one pair of states has been removed, we will apply  $\mathcal{F}_1$  once more. This can be done a maximum of  $n = |S|$  times, before the resulting set is empty. In each iteration, we have to check for both state in each pair of bisimilar states if every transition is matched. This means that for each state  $s$  we will for each other state  $t$  (if  $(s, t) \in \mathcal{F}_1^i$ ) check each transition  $s \rightarrow_w s'$  and see if it can be matched by a trace from  $t$  as required. This means that we will try to find a path of cost  $w$  from  $t$  to each of the states  $t'$ , where  $(s', t') \in \mathcal{F}_1^i$ . This means applying Theorem 3 at most  $n$  times for each transition  $s \rightarrow_w s'$  for each  $t$ . Since the problem can be solved by applying an NP-complete problem a polynomial number of times the problem is in NP. ■

By Theorem 2 and Theorem 4, we have the following result.

**Theorem 5.** Deciding WBB on a finite rational WKSs is NP-complete.

#### IV. WCTL WITHOUT NEXT, UNIVERSALITY AND LOWER BOUNDS

Using the entire WCTL without the next operator leads to a definition of a bisimulation relation that is NP-complete. We therefore limit ourselves further, to see if we can find a relation

which is decidable in polynomial time, while still preserving interesting properties in the logic. We limit ourselves to a fragment of WCTL called  $\text{EWCTL}_{\leq X}^{\leq}$  in which we only concentrate on upper bounds and the existential operator, while still leaving out the next operator. This will enable us to reason about a maximal bound on the cost of arriving to some behavior, while preserving original behavior on the way.

**Definition 6** (Syntax of  $\text{EWCTL}_{\leq X}^{\leq}$ ). Let  $\mathcal{AP}$  be a set of atomic propositions. The syntax of  $\text{EWCTL}_{\leq X}^{\leq}$  is given by

$$\phi ::= p \mid \neg p \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid E(\phi_1 U_{\leq u} \phi_2),$$

where  $p \in \mathcal{AP}$  and  $u \in \mathbb{Q}_{\geq 0}$ . ★

The semantics of  $\text{EWCTL}_{\leq X}^{\leq}$  is given by the same satisfiability relation as the semantics for WCTL, since we can substitute any upper bound  $\leq u$  with an interval  $[0, u]$ .

Let us now develop a notion of branching simulation, such that the simulation and the logic induces the same relation on finite WKSs. We define the simulation from the same idea as before, but now demand the cost of the matching trace to be lower than the cost of the transition.

**Definition 7.** Given a WKS  $\mathcal{K} = (S, \mathcal{AP}, \mathcal{V}, \rightarrow)$  an *existential bounded simulation* (EBS) is a relation  $R \subseteq S \times S$ , such that whenever  $(s, t) \in R$

- $\mathcal{V}(s) = \mathcal{V}(t)$ ,
- for all  $s \rightarrow_w s'$  there exists  $t \rightarrow_{v_1} t_1 \rightarrow_{v_2} \dots \rightarrow_{v_k} t_k$  such that  $\sum_{i=1}^k v_i \leq w$ ,  $(s', t_k) \in R$  and  $\forall i < k$ ,  $(s, t_i) \in R$ .

If there exists an existential bounded simulation relating  $s$  and  $t$ , we say that  $s$  and  $t$  are existential bounded similar and denote it by  $s \leq_E t$ . The relation  $\leq_E$  will henceforth be referred to as existential bounded similarity (EBS). ★

The following theorem shows that indeed  $\text{EWCTL}_{\leq X}^{\leq}$  and EBS induce the same relation on finite WKSs. The proof of the following theorem is as the proof of Theorem 1.

**Theorem 6.** Let  $\mathcal{K} = (S, \mathcal{AP}, \mathcal{V}, \rightarrow)$  be a finite WKS. Then for all  $s, t \in S$

$$s \leq_E t \text{ iff } [\forall \phi \in \text{EWCTL}_{\leq X}^{\leq} s \models \phi \Rightarrow t \models \phi].$$

Consider two examples of the use of this theorem.

**Example 5.** Consider Figure 1(c) and 1(e). We can verify that  $s^3$  and  $s^5$  are EBS. Transition  $s^3 \rightarrow_2 s_2^3$  can be matched by  $s^5 \rightarrow_1 s_2^5$ , since clearly  $s_2^3 \leq_E s_2^5$ . Transition  $s^3 \rightarrow_2 s_1^3$  can be matched by  $s^5 \rightarrow_1 s_1^5$  if  $s_1^3 \leq_E s_1^5$ . We only have one transition  $s_1^3 \rightarrow_3 s_2^3$  which can be matched by  $s_1^5 \rightarrow_1 s_3^5 \rightarrow_1 s_2^5$  if  $s_1^3 \leq_E s_3^5$ . There is the same transition  $s_1^3 \rightarrow_3 s_2^3$  which can be matched by  $s_3^5 \rightarrow_1 s_2^5$ . ◇

**Example 6.** Consider Figure 1(c) and 1(d),  $s^4$  and  $s^3$  are not EBS, since the transition  $s^4 \rightarrow_1 s_1^4$  can only be matched from  $s^3$  by doing nothing. Then, we need  $s_1^4 \leq_E s^3$ , but the transition  $s_1^4 \rightarrow_1 s_3^4$  cannot be matched from  $s^3$ . This means, we can find a distinguishing formula, such that  $s^4 \not\models \phi$ , but  $s^3 \models \phi$ . We choose  $\phi = E(pU_{\leq 1}E(pU_{\leq 1}q))$  to exemplify this. ◇

To compute EBS, we need to compute the maximal fixed point over a suitable function, defined in the canonical way. We can now find all simulating states as  $\leq_E = \mathcal{F}_E^M(S \times S)$  for some

natural number  $M$ . This gives us a way of computing  $\leq_{\mathbf{E}}$  by simply computing the non-increasing sequence  $\mathcal{F}_{\mathbf{E}}^0(S \times S) \supseteq \mathcal{F}_{\mathbf{E}}^1(S \times S) \supseteq \mathcal{F}_{\mathbf{E}}^2(S \times S) \supseteq \dots$  until it stabilizes.

Let us sketch an algorithm based upon this principle.

*Algorithm 1.* In a WKS  $\mathcal{K} = (S, \mathcal{AP}, \mathcal{V}, \rightarrow)$  let  $n$  be the number of states and  $m$  the number of transitions. We compute  $\leq_{\mathbf{E}}$  iteratively as described. The algorithm will stop after at most  $n^2$  iterations, since we will remove at least one pair for each iteration, starting from  $\mathcal{F}_{\mathbf{E}}^0(S \times S) = S \times S$ .

Let  $\mathcal{F}_{\mathbf{E}}^n(S \times S)$  be given. For each state  $s$  let  $\mathcal{K}_s^n$  be the projection of  $\mathcal{K}$  to the set  $\{t \mid (s, t) \in \mathcal{F}_{\mathbf{E}}^n(S \times S)\}$ . In  $\mathcal{K}_s^n$ , we may by classic algorithm in  $O(n^3)$  compute the shortest path between all pairs of states. We write  $t \rightarrow_w^{n,s} t'$  if  $w$  is the length of the shortest such path between  $t$  and  $t'$ .

We can now calculate our next iteration; for each  $(s, t) \in \mathcal{F}_{\mathbf{E}}^n(S \times S)$ , for each  $s \rightarrow_w s'$ , we check whether there exists a  $t'$  such that  $t \rightarrow_v^{s,n} t'$  and  $t' \rightarrow_{v'} t''$  where  $v + v' \leq w$  and  $(s', t'') \in \mathcal{F}_{\mathbf{E}}^n(S \times S)$ . If there exist such  $t'$  then  $(s, t) \in \mathcal{F}_{\mathbf{E}}^{n+1}(S \times S)$  otherwise  $(s, t) \notin \mathcal{F}_{\mathbf{E}}^{n+1}(S \times S)$ . The check can be performed in  $O(n^2)$ . Thus, overall,  $\leq_{\mathbf{E}}$  can be computed in  $O(n^2 \cdot (n \cdot n^3 + m \cdot n \cdot n^2)) \leq O(n^7)$ . We can assume  $m \leq n^2$ , since we are only interested in the cheapest transition between two states, so if two or more exist we will disregard everyone but the cheapest.

## V. WCTL WITHOUT NEXT, EXISTENTIALITY AND LOWER BOUNDS

We have now established a relation for a fragment of WCTL with upper bounds and only the existential quantifier over paths, but we would like to be able to reason about the universal quantifier, which will allow us to model safety properties, e.g., cost-bounded liveness properties. Let us look at a fragment with this quantifier instead.

*Definition 8* (Syntax of  $\text{AWCTL}_{\leq X}^{\leq}$ ). Let  $\mathcal{AP}$  be a set of atomic propositions. The syntax of  $\text{AWCTL}_{\leq X}^{\leq}$  is given by

$$\phi ::= p \mid \neg p \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid A(\phi_1 U_{\leq u} \phi_2),$$

where  $p \in \mathcal{AP}$  and  $u \in \mathbb{Q}_{\geq 0}$ .  $\star$

The semantics of  $\text{AWCTL}_{\leq X}^{\leq}$  is given by the same satisfiability relation as the semantics for WCTL, since we can substitute any upper bound  $\leq u$  with an interval  $[0, u]$ .

Let us now develop a notion of weighted branching simulation, such that whatever relation is induced by the simulation is also induced by the logic. The idea of the simulation is basically the same as for EBS except the simulating state has to have a trace that is more expensive than the transition it matches.

*Definition 9.* Given a WKS  $\mathcal{K} = (S, \mathcal{AP}, \mathcal{V}, \rightarrow)$  a *universal bounded simulation* (UBS) is a relation  $R \subseteq S \times S$ , such that whenever  $(s, t) \in R$

- $\mathcal{V}(s) = \mathcal{V}(t)$ ,
- for all  $s \rightarrow_w s'$  there exists  $t \rightarrow_{v_1} t_1 \rightarrow_{v_2} \dots \rightarrow_{v_k} t_k$  such that  $\sum_{i=1}^k v_i \geq w$ ,  $(s', t_k) \in R$  and  $\forall i < k$ ,  $(s, t_i) \in R$ .

If there exists a universal bounded simulation relating  $s$  and  $t$ , we say that  $s$  and  $t$  are universal bounded similar and denote it by  $s \leq_{\mathbf{A}} t$ . The relation  $\leq_{\mathbf{A}}$  will henceforth be referred to as universal bounded similarity (UBS).  $\star$

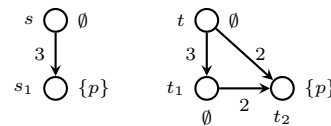


Figure 3. A WKS  $\mathcal{K}$ . The states  $s$  and  $t$  are not universally bounded similar, but every formula that  $t$  satisfies is also satisfied by  $s$ .

The following theorem shows that we can go from UBS to a relation induced by  $\text{AWCTL}_{\leq X}^{\leq}$  on finite WKSs. The structure of the proof is as the proof for Theorem 1.

*Theorem 7.* Let  $\mathcal{K} = (S, \mathcal{AP}, \mathcal{V}, \rightarrow)$  be a finite WKS. Then for all  $s, t \in S$

$$\text{if } s \leq_{\mathbf{A}} t \text{ then } [\forall \phi \in \text{AWCTL}_{\leq X}^{\leq} t \models \phi \Rightarrow s \models \phi].$$

Note that we cannot go from a logic induced relation to our simulation. This is clear from the following counterexample.

*Example 7.* Consider the WKS  $\mathcal{K}$  in Figure 3. We will prove that  $[\forall \phi \in \text{AWCTL}_{\leq X}^{\leq} \text{ if } t \models \phi \text{ then } s \models \phi, \text{ but } s \not\leq_{\mathbf{A}} t]$ . Look at all possible traces from  $s$  and  $t$ ;

$$\begin{aligned} \pi^s &= s \rightarrow_3 s_1 \\ \pi_1^t &= t \rightarrow_2 t_2 \\ \pi_2^t &= t \rightarrow_3 t_1 \rightarrow_2 t_2. \end{aligned}$$

We prove that  $s \not\leq_{\mathbf{A}} t$ . If  $s \leq_{\mathbf{A}} t$  the transition  $s \rightarrow_3 s_1$  has to be matched from  $t$ , this can only be done by  $\pi_1^t$ , since  $\pi_1^t$  has too low a cost. Then either  $s_1 \leq_{\mathbf{A}} t_1$  or both  $s \leq_{\mathbf{A}} t_1$  and  $s_1 \leq_{\mathbf{A}} t_2$ . The first case is impossible since the atomic propositions are different and the second is impossible, since  $t_1$  does not have a transition of enough cost to match  $s \rightarrow_3 s_1$ . Now, we prove that  $[\forall \phi \in \text{AWCTL}_{\leq X}^{\leq} \text{ if } t \models \phi \text{ then } s \models \phi]$ . Induction on the structure of  $\phi$ .

**The case  $t \models A(\phi_1 U_{\leq u} \phi_2)$ :** Split in cases  $u < 3$  and  $u \geq 3$ . If  $u < 3$  then  $t \models A(\phi_1 U_{\leq u} \phi_2)$  only if  $t \models \phi_2$ . By induction  $s \models \phi_2$  and therefore  $s \models A(\phi_1 U_{\leq u} \phi_2)$ .

If  $u \geq 3$  then  $t \models A(\phi_1 U_{\leq u} \phi_2)$  if either  $t \models \phi_2$  or ( $t \models \phi_1$ ,  $t_1 \models \phi_1 \wedge \phi_2$  and  $t_2 \models \phi_2$ ). In the first case by induction  $s \models \phi_2$ , hence  $s \models A(\phi_1 U_{\leq u} \phi_2)$ . In the second case by induction  $s \models \phi_1$  and since  $s_1 \leq_{\mathbf{A}} t_2$  by Theorem 7  $s_1 \models \phi_2$ , therefore  $s \models A(\phi_1 U_{\leq u} \phi_2)$ .  $\diamond$

Consider an example of how we can use Theorem 7.

*Example 8.* Consider Figure 1(c) and 1(e). We show that  $s^3$  and  $s^5$  are UBS. Transition  $s^3 \rightarrow_2 s_2^3$  can be matched by  $s^5 \rightarrow_1 s_1^5 \rightarrow_1 s_3^5 \rightarrow_1 s_2^5$ , if  $s^3 \leq_{\mathbf{E}} s_1^5$  and  $s^3 \leq_{\mathbf{E}} s_3^5$ . Let us prove that  $s^3 \leq_{\mathbf{E}} s_3^5$ . Transition  $s^3 \rightarrow_2 s_2^3$  can be matched by  $s_3^5 \rightarrow_1 s_3^5 \rightarrow_1 s_3^5 \rightarrow_1 s_2^5$ . Transition  $s^3 \rightarrow_2 s_1^3$  can be matched by  $s_3^5 \rightarrow_1 s_3^5 \rightarrow_1 s_3^5 \rightarrow_1 s_3^5$ . Let us prove that  $s^3 \leq_{\mathbf{E}} s_1^5$ . Transition  $s^3 \rightarrow_2 s_2^3$  can be matched by  $s_1^5 \rightarrow_1 s_3^5 \rightarrow_1 s_3^5 \rightarrow_1 s_2^5$ , since  $s^3 \leq_{\mathbf{E}} s_3^5$ . Transition  $s^3 \rightarrow_2 s_1^3$  can be matched by  $s_1^5 \rightarrow_1 s_3^5 \rightarrow_1 s_3^5 \rightarrow_1 s_3^5$ , since  $s^3 \leq_{\mathbf{E}} s_3^5$ . Let us return to the main problem  $s^3 \leq_{\mathbf{E}} s^5$ . We only need to prove that transition  $s^3 \rightarrow_2 s_1^3$  can be matched. We can match with  $s^5 \rightarrow_1 s_1^5 \rightarrow_1 s_3^5 \rightarrow_1 s_3^5$ . Therefore  $s^3$  and  $s^5$  satisfy the same formulae.  $\diamond$

As before, we can define a suitable function  $\mathcal{F}_{\mathbf{A}}$  in the canonical way such that the UBS is a maximal fixed point over this function. Again, we can find all simulating states as  $\leq_{\mathbf{A}} = \mathcal{F}_{\mathbf{A}}^M(S \times S)$  for some natural number  $M$ . This again gives us a way of computing  $\leq_{\mathbf{A}}$  by simply computing the non-increasing sequence  $\mathcal{F}_{\mathbf{A}}^0(S \times S) \supseteq \mathcal{F}_{\mathbf{A}}^1(S \times S) \supseteq \mathcal{F}_{\mathbf{A}}^2(S \times S) \supseteq \dots$

... until it stabilizes. To find  $\leq_A$ , we can use Algorithm 1, only we have to find the longest path between states instead of the shortest.

## VI. DISTANCES

Since the notion of simulation and bisimulation is rather restrictive, we turn our attention to distances between systems. When working with weighted systems, it can often be beneficial to describe how well one system approximates another instead of only reasoning about systems that are behaviorally equivalent, since miniscule differences in weight will render two systems not equivalent. The distances are based upon the idea of two systems being the same to a certain degree, but deviating by a percentage  $\varepsilon$ . To compare behavior for a system at distance  $\varepsilon$  from another system, we introduce an  $\varepsilon$ -expansion of formulae, which is defined in the following recursive way.

*Definition 10* ( $\varepsilon$ -expansion). The recursive  $\varepsilon$ -expansion of formulae is given for an arbitrary  $\varepsilon \in \mathbb{Q}_{\geq 0}$  by the following

- If  $\phi = x$  then  $\phi^\varepsilon = x$ , where  $x$  is a literal
- If  $\phi = \phi_1 \wedge \phi_2$  then  $\phi^\varepsilon = \phi_1^\varepsilon \wedge \phi_2^\varepsilon$
- If  $\phi = E(\phi_1 U_I \phi_2)$  then  $\phi^\varepsilon = E(\phi_1^\varepsilon U_{I^\varepsilon} \phi_2^\varepsilon)$
- If  $\phi = A(\phi_1 U_I \phi_2)$  then  $\phi^\varepsilon = A(\phi_1^\varepsilon U_{I^\varepsilon} \phi_2^\varepsilon)$ ,

where  $I^\varepsilon$  is defined as  $I^\varepsilon = [l, u]^\varepsilon = [l(1 - \varepsilon), u(1 + \varepsilon)]$ .  $\star$

Consider an  $\varepsilon$ -relation that takes WBB as a starting point.

*Definition 11.* Given a WKS  $\mathcal{K} = (S, \mathcal{AP}, \mathcal{V}, \rightarrow)$  and an  $\varepsilon \in \mathbb{R}_{\geq 0}$  define a relation  $R^\varepsilon \subseteq S \times S$  such that whenever  $(s, t) \in R^\varepsilon$  then

- $\mathcal{V}(s) = \mathcal{V}(t)$
- for all  $s \rightarrow_w s'$  there exists  $t \rightarrow_{v_1} t_1 \rightarrow_{v_2} \dots \rightarrow_{v_k} t_k$ , such that  $\sum_{i=1}^k v_i \in [w(1 - \varepsilon), w(1 + \varepsilon)]$ ,  $(s', t_k) \in R^\varepsilon$  and for all  $i < k$ ,  $(s, t_i) \in R^\varepsilon$

If  $s$  and  $t$  are in this relation, we denote it by  $s \approx_1^\varepsilon t$ .  $\star$

We can now define a weighted branching distance.

*Definition 12.* Given a WKS  $\mathcal{K} = (S, \mathcal{AP}, \mathcal{V}, \rightarrow)$  and a relation  $R^\varepsilon$  as described in Definition 11, the *weighted branching distance* (WBD) between two states  $s, t \in S$  is given by

$$d^I(s, t) = \inf_{\varepsilon} \{ (s, t) \in R^\varepsilon \} \quad \star$$

Let us restrict  $\text{WCTL}_{-X}$  to only encompass the existential quantifier, which also leads to the removal of negation on formulae, and thereby get the following logic.

*Definition 13* (Syntax of  $\text{EWCTL}_{-X}$ ). Let  $\mathcal{AP}$  be a set of atomic propositions. The syntax of  $\text{EWCTL}_{-X}$  is given by

$$\phi ::= p \mid \neg p \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid E(\phi_1 U_I \phi_2),$$

where  $p \in \mathcal{AP}$  and  $I = [l, u]$ , where  $l, u \in \mathbb{Q}_{\geq 0}$ .  $\star$

The semantics of  $\text{EWCTL}_{-X}$  is given by the same satisfiability relation as the semantics for  $\text{WCTL}_{-X}$ .

We prove that with the  $\varepsilon$ -expansion of formulae in  $\text{EWCTL}_{-X}$ , we can characterize the properties of WKSs up to WBD.

*Theorem 8.* Let  $\mathcal{K} = (S, \mathcal{AP}, \mathcal{V}, \rightarrow)$  be a finite WKS. Then for all  $s, t \in S$

$$s \approx_1^\varepsilon t \text{ iff } \forall \varepsilon' \in \mathbb{Q}_{\geq 0}, \varepsilon \leq \varepsilon' \\ [\forall \phi \in \text{EWCTL}_{-X} s \models \phi \Rightarrow t \models \phi^{\varepsilon'}].$$

*Proof.* ( $\Rightarrow$ ) Suppose  $s \approx_1^\varepsilon t$ . Induction on the structure of  $\phi$ .

**The case  $\phi = E(\phi_1 U_I \phi_2)$ :** Suppose  $s \models \phi$ . By definition  $s \models \phi$  iff there exists  $s \rightarrow_{w_1} s_1 \rightarrow_{w_2} \dots \rightarrow_{w_k} s_k \rightarrow \dots$ , such that  $s_k \models \phi_2$ ,  $\forall i < k, s_i \models \phi_1$  and  $\sum_{i=1}^k w_i \in I$ . As  $s \approx_1^\varepsilon t$ , we have that for every step  $s_i \rightarrow_{w_{i+1}} s_{i+1}$  there exists  $t^i \rightarrow_{v_1^i} t_1^i \rightarrow_{v_2^i} \dots \rightarrow_{v_{h^{i+1}}^i} t^{i+1}$  such that  $t^{i+1} \approx_1^\varepsilon s_{i+1}$ ,  $\forall j <$

$h^{i+1}, t_j^i \approx_1^\varepsilon s_i$  and  $\sum_{j=1}^{h^{i+1}} v_j^{i+1} \in [w_{i+1}(1 - \varepsilon), w_{i+1}(1 + \varepsilon)]$ .

By induction, each state  $t_j^i \models \phi_1^{\varepsilon'}$ , the state  $t^k \models \phi_2^{\varepsilon'}$

and  $\sum_{i=1}^k \sum_{j=1}^{h^i} v_j^i \in [\sum_{i=1}^k w_i(1 - \varepsilon), \sum_{i=1}^k w_i(1 + \varepsilon)] \subseteq [\sum_{i=1}^k w_i(1 - \varepsilon'), \sum_{i=1}^k w_i(1 + \varepsilon')]$  so by definition  $t \models \phi^{\varepsilon'}$ .

**The case  $\phi = A(\phi_1 U_I \phi_2)$ :** Suppose  $s \models \phi$ . If  $t \not\models$  then trivially  $t \models \phi^{\varepsilon'}$ . Otherwise suppose  $t \rightarrow_{v_1} t_1 \rightarrow_{v_2} \dots$ . As  $s \approx_{wbb} t$ , we have that for every step  $t_i \rightarrow_{v_{i+1}} t_{i+1}$  there exists  $s^i \rightarrow_{w_1^{i+1}} s_1^i \rightarrow_{w_2^{i+1}} \dots \rightarrow_{w_{k^i}^{i+1}} s^{i+1}$  such that  $s^{i+1} \approx_{wbb}$

$t_{i+1}$ ,  $\forall j < k^i, s_j^i \approx_{wbb} t_i$  and  $v_{i+1} \in [\sum_{j=1}^{k^i} w_j^i(1 - \varepsilon), \sum_{j=1}^{k^i} w_j^i(1 + \varepsilon)] \subseteq [\sum_{j=1}^{k^i} w_j^i(1 - \varepsilon'), \sum_{j=1}^{k^i} w_j^i(1 + \varepsilon')]$ .

Since  $s \models A(\phi_1 U_I \phi_2)$ , by definition for all  $s \rightarrow_{w_1} s_1 \rightarrow_{w_2} \dots \rightarrow_{w_k} s_k \rightarrow \dots$  exists  $s_k \models \phi_2$ , such that  $\forall j < k, s_j \models \phi_1$  and  $\sum_{j=1}^k w_j \in I$ . Therefore there exists an  $h$  such that

$s_k \approx_1 t_h$  and  $\forall i < h, \exists j < k, t_i \approx_1 s_j$  and  $\sum_{i=1}^h v_i \in [\sum_{i=1}^k \sum_{j=1}^{k^i} w_j^i(1 - \varepsilon), \sum_{i=0}^{k-1} \sum_{j=1}^{k^i} w_j^i(1 + \varepsilon)]$ . By induction, for all  $t \rightarrow_{v_1} t_1 \rightarrow_{v_2} \dots \rightarrow_{v_h} t_h \rightarrow \dots$ , there exists  $t_h$ , such that  $t_h \models \phi_2^{\varepsilon'}$ ,  $\forall i < h, t_i \models \phi_1^{\varepsilon'}$  and  $\sum_{i=1}^h v_i \in I^{\varepsilon'}$  so by definition  $t \models \phi^{\varepsilon'}$ .

( $\Leftarrow$ ) Define  $(s, t) \in R$  iff  $[\forall \phi \in \text{EWCTL}_{-X} \text{ if } s \models \phi \text{ then } t \models \phi^\varepsilon]$ . We show that  $R$  is a WBD.

Suppose  $s \rightarrow_w s'$  and let  $\pi_i = t \rightarrow_{v_1^i} t_1^i \rightarrow_{v_2^i} \dots \rightarrow_{v_{k^i}^i} t_{k^i}^i$

such that  $\sum_{j=1}^{k^i} v_j^i \in [w(1 - \varepsilon), w(1 + \varepsilon)]$  be traces from  $t$  of weight within  $[w(1 - \varepsilon), w(1 + \varepsilon)]$ . Without loss of generality,

we can skip traces with zero-cycles, which means that since the WKS is finite there is a finite number of traces  $\pi_i$  of weight within  $[w(1 - \varepsilon), w(1 + \varepsilon)]$ ,  $i = 1, \dots, n$ . Assume none of these traces match  $s \rightarrow_w s'$ , which means that for each  $\pi_i$  either

$(s', t_{k^i}^i) \notin R$  or there exists a  $j < k^i$  such that  $(s, t_j^i) \notin R$ . For each  $\pi_i$  such that  $(s', t_{k^i}^i) \notin R$ ,  $i = 1, \dots, k$ ,  $k \leq n$ , there exists a formula  $\psi_i$  such that  $s' \models \psi_i$  and  $t_{k^i}^i \not\models \psi_i^{\varepsilon'}$  and for each  $\pi_i$  such that  $(s, t_j^i) \notin R$ ,  $i = k + 1, \dots, n$ , there exists a formula  $\phi_i$  such that  $s \models \phi_i$  and  $t_j^i \not\models \phi_i^{\varepsilon'}$ .

This means that for a decreasing series of rationals  $w^j$  such that  $\lim_{j \rightarrow \infty} w^j = w$  and an increasing series of rationals  $y^j$  such that  $\lim_{j \rightarrow \infty} y^j = w$ , we can create a series of formulae  $\phi^j = E(\bigwedge_{i \in [1, k]} \phi_i U_{[y^j, w^j]} \bigwedge_{i \in [k, n]} \psi_i)$  for which  $s \models \bigwedge_j \phi^j$ , but  $t \not\models \bigwedge_j (\phi^j)^\varepsilon$  contradicting  $(s, t) \in R$ .  $\blacksquare$

Consider the following example, illustrating the use of Theorem 8, and the rational of letting the distance be asymmetric.

*Example 9.* Consider Figure 1(c) and 1(d). In Example 4 we showed that  $s^3$  and  $s^4$  was not WBB. When looking at the distance between them, we see that  $d(s^3, s^4) = 0$ , as  $s^4$  can exactly match  $s^3$ . Notice, however, that  $d(s^4, s^3) = 1$ . We trivially match  $s^4 \rightarrow_2 s_2^4 \rightarrow_3 s_4^4$  with  $s^3 \rightarrow_2 s_1^3 \rightarrow_3 s_2^3$ .

Transition  $s^4 \rightarrow_1 s_1^4$  has to be matched with a transition of weight within  $[1(1 - 1), 1(1 + 1)] = [0, 2]$ , which is done by matching with no transition from  $s^3$  (cost 0). We have to show that  $s_1^4 \approx_1^1 s^3$ . The only transition that we cannot trivially match is  $s_1^4 \rightarrow_1 s_3^4$ , which has to be matched by a transition



of weight within  $[0, 2]$ , it can be matched by  $s^3 \rightarrow_2 s_2^3$ . This means that whichever formulae in the logic that  $s^3$  satisfies is also satisfied by  $s^4$ , but the formulae satisfied by  $s^4$  are only guaranteed to be satisfied by  $s^3$  in their  $\varepsilon$ -extensions. For instance  $s^4$  satisfies  $E(pU_{[0,1]}E(pU_{[0,1]}q))$ , but  $s^3$  does not. It does however satisfy the  $\varepsilon$ -extension of the formula  $E(pU_{[0,2]}E(pU_{[0,2]}q))$ .  $\diamond$

Consider EBS as our starting point to define an  $\varepsilon$ -relation.

**Definition 14.** Given a WKS  $\mathcal{K} = (S, \mathcal{AP}, \mathcal{V}, \rightarrow)$  and an  $\varepsilon \in \mathbb{R}_{\geq 0}$  define a relation  $R^\varepsilon \subseteq S \times S$  such that whenever  $(s, t) \in R^\varepsilon$  then

- $\mathcal{V}(s) = \mathcal{V}(t)$
- for all  $s \rightarrow_w s'$  there exists  $t \rightarrow_{v_1} t_1 \rightarrow_{v_2} \dots \rightarrow_{v_k} t_k$ , such that  $\sum_{i=1}^k v_i \leq w(1 + \varepsilon)$ ,  $(s', t_k) \in R^\varepsilon$  and for all  $i < k$ ,  $(s, t_i) \in R^\varepsilon$

If  $s$  and  $t$  are in this relation, we denote it by  $s \leq_{\mathbb{E}}^\varepsilon t$ .  $\star$

We can now define an existential bounded distance.

**Definition 15.** Given a WKS  $\mathcal{K} = (S, \mathcal{AP}, \mathcal{V}, \rightarrow)$  and a relation  $R^\varepsilon$  as described in Definition 14, the *existential bounded distance* (EBD) between states  $s, t \in S$  is given by

$$d^{\leq}(s, t) = \inf_{\varepsilon} \{ (s, t) \in R^\varepsilon \} \quad \star$$

Let us return to the logic  $\text{EWCTL}_{\leq}^X$  and prove that with  $\varepsilon$ -expansions the logic characterizes EBD. The proof of the theorem has the same structure as the proof of Theorem 8.

**Theorem 9.** Let  $\mathcal{K} = (S, \mathcal{AP}, \mathcal{V}, \rightarrow)$  be a finite WKS. Then for all  $s, t \in S$

$$s \leq_{\mathbb{E}}^\varepsilon t \text{ iff } \forall \varepsilon' \in \mathbb{Q}_{\geq 0}, \varepsilon \leq \varepsilon' \\ [\forall \phi \in \text{EWCTL}_{\leq}^X s \models \phi \Rightarrow t \models \phi^{\varepsilon'}].$$

It should be noted that neither WBD nor EBD are (hemi)metrics as is usually the case, which stems from the relativism in the definition of the distances. Future work should include a classification of these.

## VII. CONCLUSION AND FUTURE WORK

We have extended the idea of branching bisimulation with weights in three distinct ways, relating to different fragments of WCTL. We initially removed the next operator from the logic, to allow for systems to be related even though they performed specific behavior with different number of transitions. The weighted branching bisimulation relation that was characterized by this logic, turned out to be NP-complete, which prompted us to look into other fragments of the logic. We proved that for fragments allowing only upper bounds and either the existential or the universal quantifier we could decide the resulting simulation relations in polynomial time. We furthermore expanded these concepts into distance-like relations. The distances build upon the ideas of the different relations and were also characterized by fragments of WCTL, when we introduced a relative expansion on formulae. Even if the distances are not (hemi)metrics, they can however be meaningfully interpreted as relative distances.

Notably, this work demonstrates that the real-valued weights in the models can be described by only involving rational parameters in the logics. The approximation of reals by rationals are enough to describe this more general behavior.

This research opens a few promising future work directions. On one hand, designing a simulation relation which characterizes the same relations as the logic with the universal quantifier is problematic and whether such a relation can be defined at all is an open problem. On the other hand, the distance-like relations inspired by our semantics fail to satisfy the triangle inequality. The characterization of such relations is a promising research direction. Furthermore, computability and complexity results related to these distances are open.

## REFERENCES

- [1] R. Milner, *A Calculus of Communicating Systems*, ser. Lecture Notes in Computer Science. Springer, 1980, vol. 92.
- [2] C. A. R. Hoare, “Communicating sequential processes,” *Commun. ACM*, vol. 21, no. 8, pp. 666–677, 1978.
- [3] J. A. Bergstra and J. W. Klop, “Algebra of communicating processes with abstraction,” *Theor. Comput. Sci.*, vol. 37, pp. 77–121, 1985.
- [4] R. Milner, *Communication and concurrency*. Prentice hall New York etc., 1989, vol. 84.
- [5] D. Park, “Concurrency and automata on infinite sequences,” in *Theoretical Computer Science*, ser. Lecture Notes in Computer Science, P. Deussen, Ed. Springer Berlin Heidelberg, 1981, vol. 104, pp. 167–183.
- [6] M. Hennessy and R. Milner, “Algebraic laws for nondeterminism and concurrency,” *J. ACM*, vol. 32, no. 1, pp. 137–161, Jan. 1985.
- [7] M. Browne, E. Clarke, and O. Grmberg, “Characterizing finite kripke structures in propositional temporal logic,” *Theoretical Computer Science*, vol. 59, no. 1, pp. 115 – 131, 1988.
- [8] R. J. van Glabbeek and W. P. Weijland, “Branching time and abstraction in bisimulation semantics,” *J. ACM*, vol. 43, no. 3, pp. 555–600, 1996.
- [9] R. De Nicola and F. Vaandrager, “Three logics for branching bisimulation,” *J. ACM*, vol. 42, no. 2, pp. 458–487, Mar. 1995.
- [10] K. G. Larsen and R. Mardare, “Complete proof systems for weighted modal logic,” *Theor. Comput. Sci.*, vol. 546, pp. 164–175, 2014.
- [11] K. G. Larsen, R. Mardare, and B. Xue, “Decidability and expressiveness of recursive weighted logic,” in *Perspectives of System Informatics - 9th International Ershov Informatics Conference, PSI 2014, St. Petersburg, Russia, June 24-27, 2014. Revised Selected Papers*, 2014, pp. 216–231.
- [12] P. Buchholz and P. Kemper, “Model checking for a class of weighted automata,” *Discrete Event Dynamic Systems*, vol. 20, no. 1, pp. 103–137, 2010.
- [13] C. Thrane, U. Fahrenberg, and K. G. Larsen, “Quantitative analysis of weighted transition systems,” *The Journal of Logic and Algebraic Programming*, vol. 79, no. 7, pp. 689 – 703, 2010, the 20th Nordic Workshop on Programming Theory (NWPT 2008).
- [14] K. G. Larsen, U. Fahrenberg, and C. R. Thrane, “Metrics for weighted transition systems: Axiomatization and complexity,” *Theor. Comput. Sci.*, vol. 412, no. 28, pp. 3358–3369, 2011.
- [15] U. Fahrenberg, C. Thrane, and K. G. Larsen, “Distances for weighted transition systems: Games and properties,” *arXiv preprint arXiv:1107.1205*, 2011.
- [16] L. de Alfaro, M. Faella, and M. Stoelinga, “Linear and branching system metrics,” *IEEE Trans. Software Eng.*, vol. 35, no. 2, pp. 258–273, 2009.
- [17] U. Fahrenberg, K. G. Larsen, and C. Thrane, “A quantitative characterization of weighted kripke structures in temporal logic,” *Computing and Informatics*, vol. 29, no. 6+, pp. 1311–1324, 2012.
- [18] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.
- [19] M. Nykänen and E. Ukkonen, “The exact path length problem,” *J. Algorithms*, vol. 42, no. 1, pp. 41–53, Jan. 2002.

# Towards an Astrophysical-oriented Computational multi-Architectural Framework

Dzmitry Razmyslovich\*, Reinhard Männer†  
 Institute for Computer Engineering (ZITI),  
 University of Heidelberg,  
 Mannheim, Germany

Email: \*dzmitry.razmyslovich@ziti.uni-heidelberg.de,

†reinhard.maenner@ziti.uni-heidelberg.de

Guillermo Marcus  
 NVIDIA Corporation,  
 Berlin, Germany

Email: gmarcus@nvidia.com

**Abstract**—In this exploratory paper, we present a framework for simplifying software development in the astrophysical simulations branch - Astrophysical-oriented Computational multi-Architectural Framework (ACAF). The ACAF is designed to provide a user with the set of objects and functions covering some aspects of application development for astrophysical problems. The target data to be processed with the ACAF is a set of states of a particle system. Being designed as a C++ framework, the ACAF decreases the expertise needs required to implement such programs preserving the extension flexibility and the possibility to use the existing libraries. The ACAF abstracts the accelerating device itself, the usage of it, the data distribution and usage. Also, the ACAF incorporates the different kernel implementations into a single object.

**Keywords**—Astrophysics; Heterogeneous; Framework; Cluster; GPGPU.

## I. INTRODUCTION

Astrophysical simulation tasks have usually high computational density, therefore it's common to use hardware accelerators for solving them [1]. Also, the astrophysical simulations have a huge amount of data to calculate, which makes it reasonable to use computer clusters. But the data dependencies of the simulation algorithms limit the usage of big clusters because of high data communication rate. Therefore, the astrophysical simulations tasks are normally solved using heterogeneous clusters [2][3][4]. According to TOP500, the top-rated heterogeneous clusters use Graphics Processing Units (GPU) or Field Programmable Gate Arrays (FPGA) as computational accelerators.

The most important computational astrophysical problems include N-Body simulations, Smoothed Particle Hydrodynamics (SPH), Particle-Mesh and Radiative Transfer. All of them are usually approximated for the calculation purposes with respective particle physics problems. Where particle physics is a branch of physics which deals with existence and interactions of particles, that refer to some matter or radiation. Therefore, computational astrophysics data represents a collection of particles - a particle system. Each particle contains a number of parameters like position in 3D space, speed, direction, mass, etc. A collection of certain values for all parameters of all particles is named a state of a particle system. While the computational tasks embrace numerical solving of a number of equations, which evaluate the state of a particle system [5].

Developing astrophysical simulation applications for heterogeneous clusters without usage of specialized frameworks and libraries requires the following knowledge:

- knowledge of astrophysics, since the problem consists of simulating the astrophysical objects;

- knowledge of network programming for cluster utilizing;
- knowledge of parallel programming and hardware accelerators programming including usage of specific interfaces and languages;
- knowledge of micro-electronics for designing FPGA boards.

This means much time and expertise for astrophysicists, what restricts the scientists to perform calculation experiments easily on clusters and distracts them from the main goal. So the aim of our research is to **simplify software development for astrophysical simulations implementation reducing programming knowledge requirement**. The solution we suggest is the ACAF. ACAF stands for Astrophysical-oriented Computational multi-Architectural Framework. The ACAF is a toolkit for development of astrophysical simulation applications. The target data to be processed with the ACAF is a set of states of a particle system. In this exploratory paper, we present the current state of art and the results of some experiments with the ACAF.

Technically, developing of a distributed multi-architectural application could be divided into a set of the following aspects:

- balance loading;
- data communication between nodes;
- data communication between the devices inside of each node;
- computational interfaces for different architectures;
- programming languages for different interfaces (like Open Multi-Processing (OpenMP) for Central Processing Unit (CPU); Open Computing Language (OpenCL), Compute Unified Device Architecture (CUDA), Open Accelerators (OpenACC) for GPU and Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL) for FPGAs).

All these aspects should be taken into account in order to develop an application and all of them should be examined for the current system in order to reach high computational performance. Hence, it makes sense to have the ACAF, which facilitates astrophysical research by providing a user with a set of objects and functions fulfilling the following requirements:

- the structure of an object and the semantics of a function should be plain and similar to the objects often used by scientists in other programming environments and in theoretical problem descriptions;

- the objects and functions should cover most of programming aspects mentioned above;
- in the same time, there should be a possibility to extend the tools in use as well as to provide the alternative implementations of existing tools;
- finally, it would be an additional advantage to preserve a possibility to reuse the existing computational libraries, when it makes sense.

The rest of the paper is divided into 4 sections. The Section II highlights the currently existing standards, frameworks and languages for the software development targeted heterogeneous systems. The Section III consists of 3 subsections, each of them presenting some design motivations and solutions we have used to reach the goal. In the Section IV, the usage example of the current framework implementation is given. Finally, the Section V concludes the paper with an outlook to the most important advantages of the ACAF.

## II. CURRENT STATE OF ART

This section covers mostly used and important frameworks, libraries, languages and standards, which can optimize or simplify development of the specific astrophysical cluster applications.

### A. Standards

- OpenMP [6] is a standard Application Programming Interface (API) for shared-memory programming, which enables easy and efficient CPU utilization on a single node using compiler directives. The latest version 4.0 includes the compiler directives for hardware accelerators, which were previously presented in another branch OpenACC [7]. OpenMP API model definitely reduces the requirements in parallel programming skills abstracting the numerous function calls in easy-readable pragmas. Still, it doesn't hide a lot of implementation details, which are out-of-interests for scientific programmers: device data allocation, data transferring, runtime synchronization, etc. Also, API doesn't cover at all any kind of network communications and is designed solely for a single node.
- Message Passing Interface (MPI) [8] is a standardized message-passing system designed to function on a wide variety of parallel computers. MPI is widely used on many computer clusters for parallel computations on several machines. MPI can also be used for parallel computations on a single node by running multiple instances of the program. Using the extension MVAPICH2 [9], it is also possible to integrate CUDA-enabled GPU data movement transparently into MPI calls.
- CUDA [10] stands for Compute Unified Device Architecture and is a parallel computing platform and API model created by Nvidia and is only used for Nvidia GPUs. CUDA enables a user to utilize Nvidia GPUs for general-purpose computations on a single node.
- OpenCL [11] is an open standard for general purpose parallel programming across different heterogeneous

processing platforms: CPUs, GPUs and others. Choosing OpenCL, a user can utilize some hardware accelerators on a single node.

- SyCL [11] is a new C++ single-source heterogeneous programming model for OpenCL. SyCL benefits from C++11 features like lambda functions and templates. SyCL provides a high level programming abstraction for OpenCL 1.2.

### B. Frameworks and Languages

- Cactus [12] is an open-source modular environment, which enables parallel computation across different architectures due to its modularity. As separate modules, Cactus code also provides CUDA and MPI utilization. Additionally, there is an extension of Cactus code - CaCUDA, which is able to utilize Nvidia GPUs across cluster nodes by converting CaCUDA source code into CUDA kernels. No other hardware accelerators are supported so far. As of 2015, CaCUDA looks like to be not developed any more.
- Charm++ [13] is a message-driven parallel language implemented as a C++ library. The usual Charm++ program consists of a set of objects called "chares". A chare is an atomic function, which performs some calculations. Charm++ library is responsible for distributing chares between the processing units and establishes the communication between them. Charm++ provides also an additional library - Charm++ GPU Manager, which enables the user to utilize GPU directly from Charm++. In order to run some code on GPU, a user should define a work request for GPU Manager providing CUDA kernel, input and output arguments to be transferred to GPU. GPU Manager ensures the overlapping of transfers and executions on GPU and runs GPU kernel asynchronously.
- Chapel [14] is a parallel programming language. Chapel provides a user with a high-level parallel programming model which supports data parallelism, task parallelism and nested parallelism. Chapel is a very powerful language, which enables the user to write the parallel programs with several lines of code. While being designed as a new standalone language, Chapel has limited possibilities for extending the functionality - since Chapel is an open source project, everybody can have the source code and change Low Level Virtual Machine (LLVM) grammar for having new commands. But this means, even reusing the existing computational libraries could be only done with new language features.
- Flash Code [15][16] is a modular Fortran90 framework, which uses MPI to distribute the calculations over the cluster nodes. The Flash system has no built-in support for any hardware accelerators and relies on the particular modules to optimize the calculations as much as possible. A module in the Flash system is some atomic algorithmic routine performing mathematical evaluation of the particle system. This means that a module can be implemented using any accelerating techniques and libraries, but anyway requires the proficiency in parallel programming (hardware accelerators utilization; MPI usage; data distribution and

synchronization using MPI and hardware accelerators, etc). The Flash system is deployed with a big number of modules.

- Swarm [17] is a CUDA library for parallel n-body integrations with focus on simulations of planetary systems. The Swarm framework targets single machines with Nvidia GPUs as hardware accelerators. The framework provides a user with a possibility to extend the calculations algorithm, but the final system isn't scalable and cannot utilize the power of a cluster. It is designed to solve some specific problems.
- AMUSE [18] is Python framework designed to couple existing libraries for performing astrophysical simulations involving different physical domains and scales. The framework uses MPI to involve cluster nodes. While the utilization of any hardware accelerators should be a part of libraries coupled in a particular configuration.
- The Enzo [19] project is a community-developed adaptive mesh refinement simulation code. The code is modular and can be extended by users. Enzo doesn't support network communication. Still, it contains several modules developed to utilize Nvidia GPUs using CUDA.
- Some other languages, which aren't that widely used: Julia [20] language, X10 [21] language, Fortress language [22]. All these languages were initially designed for CPU clusters. Some of them provide ports or extensions for hardware accelerators, which usually have no abstraction for the accelerator memory space communications.
- And other widely used domain-specific libraries: WaLBerla [23], RooFit [24], MLFit [25].

### III. ACAF DESIGN AND STRUCTURE

The design of the ACAF should be both user-friendly for astrophysicists and easily extendable for computer scientists. Therefore, we've designed the ACAF basing on 3 concepts:

- 1) The **computational concept** describes the principal algorithm used for calculating. In other words, the computational concept is a mathematical, physical and astrophysical background of the problem solution and the environment necessary to execute the solution of the problem on some particular device. This concept bases on a set of efficient high-parallel multi-architectural algorithms. So that each function in the space of user tools has an efficient implementation for each architecture and device in use. And all implementations for the same function can work together on different platforms.
- 2) The **communication concept** describes data transfers and synchronization points between computing units or storages. The concept lies in efficient data-distribution mechanisms, which warranty presence of the necessary data in the required memory space and in the required order. This means that the communication concept is responsible for transferring data from one memory space to another and transforming it according to the user-defined, architecture-defined or device-defined rules.

- 3) The **data concept** describes logical and physical representation of the data used in a solution, as well as distribution of this data between different storages. This concept lies both in a set of data-structures providing an efficient way of managing the data of the astrophysical objects; and a set of functions for manipulating these structures.

Design of the computational concept is a technical problem lying in the space of a properly implemented set of programming interfaces to access the necessary functions on the necessary platforms.

While the design of the data concept and the communication concept can be coupled into a special distributed database. Here and further, we understand under the database its basic definition: a database is an organized collection of data. This database should provide the user with an interface for managing data. Besides, it should manipulate the data according to the requirements and properties of computational units and algorithms. Hence, the database should fulfill the following requirements:

- operating with a set of structures efficient for representing astrophysical data: tuples, trees (oct-trees, k-d trees), arrays;
- operating with huge amount of data;
- the native support of hardware accelerators like GPUs and FPGAs;
- the data should be efficiently distributed between both cluster nodes and the calculating devices inside of each node;
- the database should be programmatically scalable: a user should be able to extend the number of features in use - architectures; devices; data-structures; data manipulation schemes and functions; communication protocols;
- the database should store the data according to the function, device and platform requirements.

This means that this special database can be seen as a partitioned global address space (PGAS), which is already addressed in several existing solutions like Chapel and X10. But in our approach, we incorporate into the database not only partitioning of the address space, also other properties specified above.

Hence in this work, we address only the communication and data concepts - **the design and implementation of a distributed database**. The computational concept is designed to contain only the algorithms and functions, necessary to present the capabilities of the database.

#### A. Database Design

The target data for the ACAF database is a set of states of some particle system. According to the definition of a particle system (see Section I), there is no need for our database to store various data of various types. All parameters of a particle are some physical properties of it. So in computer representation, the parameters are usually either integer, float or double (integral) values. Hence in our database, these types of data are only considered. A state of some particle system can be represented in some computer memory space

as an array of structures, where members of a structure are particle parameters, e.g., integral data types. Therefore, the ACAF database is only targeted to store arrays of integral data elements.

As soon as a particle system usually includes some millions of particles, it's common and necessary to use computer clusters and accelerators to simulate its states. So the aim of the ACAF is to simplify implementing the simulations tasks targeted to be run on heterogeneous computer clusters utilizing as much computational power as possible. The efficient utilization of any computational device (e.g., processing unit) becomes possible only when all the parameters necessary for computation reside in the cheapest memory space in terms of access latency. The efficient use of low-level memory spaces (processing registers and near by caches of a unit) is a part of both compiler implementation and the operating system scheduler. While the programmer's task is to ensure the presence of data in the nearest high-level memory space (usually device Random-access memory (RAM)). Moreover, it's necessary to store data in high-level memory spaces in the format acceptable with computational algorithms. Hence, raw arrays are preserved in our database. This provides the direct access to the parameters of a particle.

The ability of the ACAF database to distribute data between cluster nodes and devices enables the scalability of data amount. So the amount of data to be processed is only limited to the mutual storage capabilities of cluster nodes and devices.

Distributing data between cluster nodes and devices implies division and synchronization of data according to the implementation of the computational concept particular to a certain problem. While data synchronization in heterogeneous computer clusters context implies interoperability of different programming technologies used on different computational devices. Since the ACAF database is targeted to utilize GPUs, CPUs, FPGAs and a network, the technologies we've used include: threads and OpenCL for CPUs; OpenCL and CUDA for GPUs; OpenCL for FPGAs; MPI for a network.

Interoperability of the technologies mentioned above means the following functionality of the ACAF database: copying and/or converting of memory buffers from one technology to another; synchronizing the memory buffer content distributed between different technologies.

## B. Database Implementation

The suggested database is implemented as a part of the framework - the ACAF. The implementation is done in C++ language and is organized as a collection of classes. Some of them are template classes. We concentrate on the key classes used in the ACAF in this section. The current framework implementation is targeted to be built and executed on machines running Unix operating system.

1) *Device*: A *device* instance represents some computational device, which can be used for simulation calculations on the current node. A device instance is always described by some *architecture* instance, the vendor name, the vendor identifier, the device name, the device identifier and a set of *technology* instances. The format and type of the identifiers are always architecture-dependent. All device instances are created automatically by ACAF during framework initialization according to the devices found in the operating system. No manual instantiation of a device class is possible.

2) *Architecture*: *Architecture* class is an interface class for any *device* type supported by ACAF. The instance of each ancestor architecture type is a singleton in any ACAF process. This instance provides the functionality to identify all computational devices of the desired type in the current cluster node. The predefined architecture ancestor types are:

- CPUArchitecture - identifies all CPU devices presented in the current node by parsing */proc/cpuinfo* file;
- GPUArchitecture - identifies all GPU devices presented in the current node by scanning all Peripheral Component Interconnect (PCI) devices of Video Graphics Array (VGA) type.

3) *Technology*: *Technology* class is an interface class for describing some programming technology, which can be used for some devices presented in the current node. The instance of each ancestor technology type is a singleton in any ACAF process. The ancestor class describes how to utilize a device for computational purposes: which devices are supported; which programming language (if any) should be used; how to set the parameters before executing the code; which *storage* class should be used for storing buffers; etc. Assigning the correct set of technology instances for each device is also done automatically during the initialization of ACAF. The predefined technology ancestor types are:

- pthreadTechnology - includes the functionality to run native functions in several threads using Unix pthreads library;
- OpenCLTechnology - includes the functionality to run OpenCL kernels on the supported devices;
- CUDATechnology - includes the functionality to run precompiled CUDA kernels on Nvidia devices.

4) *Network*: *Network* class is an interface class for describing some network protocol to utilize network-based computer clusters. The instance of a network type is a singleton in any ACAF process. The ancestor class describes:

- the topology of the current process instances distributed over the network;
- the communication protocol between the processes of different cluster nodes (implemented as a *storage* class type);
- the synchronization mechanisms between nodes.

The selection of the particular ancestor network type is done according to the configuration provided by the user. ACAF predefines only one network ancestor type: MPINetwork which includes the functionality to utilize MPI library.

5) *Context*: The *context* instance is a set of devices and the programming *technologies* to be utilized for executing the simulation.

6) *Database*: The *database* instance is a part of *context*, which manages the data used in the scope of the parent context. The database object has a set of *storage* instances and a set of *content* instances. The database instance is the main user interface to manipulate the data: to list all available storages in the system; to create new distributed content instances; to list the existing content instances.

7) *Storage*: The *storage* class is the interface for allocating/reading/writing/synchronizing data in the associated memory space. The storage classes implemented in ACAF are divided into the following categories:

- RAMStorage serves the functionality to operate with on-board RAM of the current node. This object is a singleton for a database.
- DeviceStorage serves the functionality to operate with some built-in device high-level memory (usually device RAM), like GPU or FPGA. For example, CUDA storage or OpenCL storage are typical device storage instances. Usually, the implementation of a particular device storage type is *technology*-dependent. Therefore, any device storage is instantiated by the *technology* instance used in the current context for the particular device.
- FileStorage serves the functionality to operate with the files, available in the current operating system.
- NetworkStorage serves the functionality to operate with the remote content. The network storage implementation is *network*-dependent. The network storage instances are created automatically by the ACAF during the initialization according to the *network* ancestor class currently used.

Each storage instance has a collection of buffers, which are physical or abstract regions in some memory space. A storage instance doesn't reflect the logical organization of the data and only operates with its representation in the memory (some sequence of bytes).

8) *Content*: The *content* class is the interface for logical organization of the data stored in the *storage* instances of the database. An instance of the *content* class reflects the particular representation of data in some memory buffer. Additionally, the ancestor content classes provide the user with some data manipulation functions, like initializing, dumping, synchronizing data. The ACAF predefines the following content types:

- Array class is a template class, which represents a distributed array with elements of the template type: each calculation device in the context comprises some part of the array. The parts are completely independent. A typical example of an array is masses of particles.
- SyncedArray class is a template class, which represents a synchronized distributed array with elements of the template type: each calculation device in the context comprises the full array, but owns only some part of it. This means that the device should modify only its own part, while the rest array will be synchronized time-to-time according to the algorithm. A typical example of a synced array is positions of particles.

9) *Kernel*: A *kernel* represents some atomic function, which is run on the computational devices of the context. Each kernel instance contains a collection of its implementations, where each *kernel implementation* represents some binary-coded *technology*-dependent executable function. The instances of the kernel class are created by the user according to the computational algorithm.

10) *Extending the ACAF*: The user has an opportunity to extend the functionality of the ACAF by implementing the other ancestor classes of the following entities:

- *Architecture* - to support other device types;
- *Technology* - to support other programming technologies;
- *Network* - to support other network protocols;
- *Content* - to support other logical data organizations.

#### IV. USAGE EXAMPLE

A running example of ACAF usage is represented with several parts: the configuration, the mathematical algorithm implementation and the environmental host code. The provided example represents the code necessary for running distributed NBody simulation on a cluster using MPI for network communication, pthread technology for CPU code and OpenCL technology for GPU code. Any changes in the resource utilization can be made by modifying the configuration file without any need to recompile the program.

##### A. Configuration File

A configuration file contains the network protocol, the *context* specification and possible distribution descriptions (see Figure 1).

```

1 network="MPI";
2 context: { skip = true; CPU = "pthread"; GPU = "OpenCL"; };
3 distribution: {
4   default = (
5     { architecture = "GPU"; size = [1024]; block = [256]; },
6     { architecture = "CPU"; size = [256]; block = [4]; }
7   );
8 };

```

Figure 1. The configuration example.

The current configuration file consists of 2 sections:

- The first section specifies which devices and nodes should be used for running the calculation (parameters *network* and *context*). Particularly, the example file above specifies that the calculation is going to be distributed over the network with a help of MPI interface and that on each node of the network all CPUs are going to be utilized by pthread technology and all GPUs are going to be utilized by OpenCL technology. An additional flag *skip* identifies that all devices unsupported by the specified technologies should be skipped.
- The second section specifies the distribution of the data inside of a single node. The user can specify as many different distributions as it's necessary using the distinct names. In our example, we have a single distribution with the name *default*.

##### B. Algorithm Code (OpenCL and pthread)

According to the technologies specified in the configuration file and the host code initialization routine, the mathematical algorithm should be implemented for one or several technologies. In our example, the algorithm is implemented for OpenCL (see Figure 2) and pthread (see Figure 3) technologies, using respectively OpenCL C language and C++

language. The code of OpenCL implementation is represented as a separate file, while pthread implementation code is a part of the environmental host code and passed to ACAF as a pointer to the function.

```

1 #pragma OPENCL EXTENSION cl_khr_fp64 : enable
2 #pragma OPENCL EXTENSION cl_amd_fp64 : enable
3
4 #define SOFTENING 0.001
5
6 __kernel void force (
7   __global double * mass, __global double4 * position,
8   __global double4 * velocity, double time_step )
9 {
10  __local double4 shared_position[ITEMS_PER_GROUP];
11  size_t lid = get_local_id(0);
12
13  shared_position[lid] = position[get_global_id(0)];
14  double4 this_acc;
15  this_acc.x = this_acc.y = this_acc.z = this_acc.w = .0;
16  for ( size_t i = 0; i < get_global_offset(0) + get_global_size(0); ++i )
17  {
18    double4 dist = shared_position[lid] - position[i];
19    this_acc += mass[i] * dist / powr(length(dist) + SOFTENING, 3.);
20  }
21
22  size_t gid = get_global_id(0);
23  velocity[gid] += this_acc * time_step;
24  position[gid] += velocity[gid] * time_step;
25 }

```

Figure 2. The OpenCL algorithm example.

```

1 #define SOFTENING 0.001
2
3 status force (
4   const acaf::uint4 & jid, const acaf::uint4 & jtotal,
5   const acaf::variant_vector & args
6 )
7 {
8   double * mass = reinterpret_cast<double *>(*(args[0].get<void *>()));
9   double4 * position = reinterpret_cast<double4 *>(*(args[1].get<void *>()));
10  double4 * velocity = reinterpret_cast<double4 *>(*(args[2].get<void *>()));
11  double time_step = *(args[3].get<double>());
12
13  double4 this_pos = position[jid[0]];
14  double4 this_acc (0.);
15  for (size_t i = 0; i < jtotal[0]; ++i)
16  {
17    double4 dist = this_pos - position[i];
18    this_acc += mass[i] * dist / pow(dist.length() + SOFTENING, 3.);
19  }
20
21  velocity[jid[0]] += this_acc * time_step;
22  position[jid[0]] += velocity[jid[0]] * time_step;
23
24  return error::Success;
25 }

```

Figure 3. The pthread algorithm example.

### C. Environmental Host Code

Finally, the environmental host code represents the main function with initialization instructions, content creations, kernel instantiations and kernel running calls written in C++ programming language with the usage of the classes described in Section III-B (see Figure 4).

## V. CONCLUSION AND FUTURE WORK

In this exploratory paper, we presented the current state of art and some results for the Astrophysical-oriented Computational multi-Architectural Framework. The ACAF is targeted to simplify the software development for astrophysical simulations implementation by providing a user with the set of objects and functions covering some aspects of application developing.

In the current work, we focused on the communication and the data concepts of software development problem designing the special distributed database. The database is aimed to process particle systems with float and/or double (integral) parameters. The database aims to store data in high-level

```

1 int main(int argc, char ** argv)
2 {
3   MPI_Init(&argc, &argv);
4   status s = error::Success;
5   do
6   {
7     // initialize the framework, the configuration file will be read
8     s = acaf::initialize(argc, argv);
9     if (s.fail()) break;
10
11    Handle < DataBase > db = Context::getContext()->getDB();
12    LinearParticles distr(Context::getContext(), acaf_string("default"));
13    Handle<Content> mass, pos, velo;
14
15    {
16      // create arrays for masses, positions and velocities
17      acaf::pair<Handle<Content>, status> tmp;
18      tmp = db->create< Array<double, 1> >("mass",
19        Content::ACAF_CONTENT_GLOBAL, distr.units(1));
20      if (tmp.second.fail()) cout << tmp.second;
21      mass = tmp.first;
22      mass->fill(acaf::variant(1.));
23      tmp = db->create< SyncedArray<double4, 1> >("position",
24        Content::ACAF_CONTENT_GLOBAL, distr.units(1));
25      if (tmp.second.fail()) cout << tmp.second;
26      pos = tmp.first;
27      pos->random( acaf::variant(double4{-1., -1., -1., 0.}),
28        acaf::variant(double4{2., 2., 2., 0.}));
29      tmp = db->create< Array<double4, 1> >("velocity",
30        Content::ACAF_CONTENT_GLOBAL, distr.units(1));
31      if (tmp.second.fail()) cout << tmp.second;
32      velo = tmp.first;
33      velo->fill(acaf::variant(double4(0.)));
34    }
35
36    double current_time = 0.;
37    double end_time = 1.;
38    double time_step = 0.01;
39    // instantiate kernel
40    Kernel force("force", Context::getContext());
41    // add kernel implementations - OpenCL and pthreads
42    s = force.add("OpenCL", "gravity.cl", "-cl-mad-enable", true);
43    if (s.fail()) cout << s << endl;
44    s = force.add("pthread", &::force);
45    if (s.fail()) cout << s << endl;
46    // add kernel arguments, which will be forwarder later to
47    // the implementations code
48    s = force.set(0, mass);
49    if (s.fail()) cout << "Adding mass failed:" << s << endl;
50    s = force.set(1, "position");
51    if (s.fail()) cout << "Adding position failed:" << s << endl;
52    s = force.set(2, "velocity");
53    if (s.fail()) cout << "Adding velocity failed:" << s << endl;
54    s = force.set(3, variant(time_step));
55    if (s.fail()) cout << "Adding timestep failed:" << s << endl;
56
57    // evaluate the particle system
58    while (current_time < end_time)
59    {
60      // run the kernel
61      s = force.start(distr.units(1));
62      if (s.fail()) break;
63      // synchronize positions all-to-all
64      pos->synchronize();
65      current_time += time_step;
66    }
67  } while (false);
68
69  acaf::finalize();
70  MPI_Finalize();
71  if (s.fail())
72    printf("An error %d (%s) occurred. Failed!\n", s.code(), s.name());
73  else
74    printf("Success!\n");
75  return s.code();
76 }

```

Figure 4. The main function example.

memory spaces in the format acceptable with computational algorithms.

The current database implementation utilizes pthreads, OpenCL and CUDA technologies to run the calculation on CPU and GPU devices and MPI interface to distribute and exchange data over the network. The implementation uses 2 types of content: array and synced array. Extending of the database functionality can be easily done by implementing the certain program interfaces.

We can conclude that the current ACAF implementation facilitates the development of network-enabled heterogeneous NBody force simulation program. With the help of ACAF, the user is able to write an application without the expertise neither in the network programming nor in the parallel programming

of some devices (CPU, GPU). ACAF requires the user to do the following tasks:

- Write a configuration file, which specifies the devices and nodes to be used and defines the distribution of the data.
- Implement the mathematical, physical part of the program.
- Write some environmental code, which does the initialization, data definition, data initialization, kernel instantiation and defines the main particle system evaluation loop.

We performed the comparison tests of the ACAF-based implementation of the Nbody forces simulation (see Section IV) against the bare OpenCL/MPI implementation. The Figure 5 represents the percent overhead of the execution time of the ACAF-based implementation to the execution time of the bare implementation scaled over the particles number in the example system. According to this chart, we see, that the time overhead of using ACAF drops to less than 1 percent for the bigger particle systems, which equivalents to 97 seconds for the case of 327680 particles.

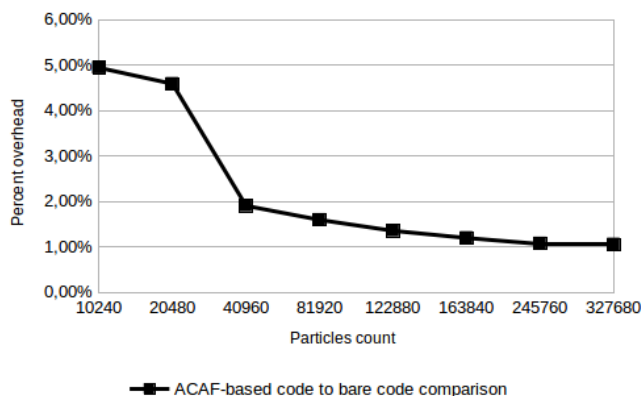


Figure 5. The ACAF-based implementation to bare implementation comparison chart.

The tests were carried out on the following test platform: the 7-nodes cluster with 4 processing nodes, each of them has the NVIDIA GeForce GTX 285 GPU with 2GB of RAM, the Intel Xeon E5504 CPU and 6GB of RAM. The nodes run Linux OS. For each test the calculation was equally distributed over all 4 processing nodes. The calculation was performed only on the GPUs using OpenCL and the positions of the particles were synchronized after each iteration using MPI.

In comparison with the other approaches mentioned in Section II-B, the following advantages of our approach can be mentioned:

- 1) ACAF is designed as a C++ framework in the first place. This implies that a lot of different other existing libraries and tools can be reused when necessary. So the user has a choice either to reimplement the algorithm using the framework tools or reuse the existing solution.
- 2) ACAF is designed to be domain specific for astrophysical (particle) problems, therefore it can have lighter structures as the generic tools.

- 3) ACAF abstracts the device itself and the usage of the device as well. This allows to create one's own usage schemas (*Technologies*) as well as extend the devices supported (by implementing *Architecture* interface).
- 4) ACAF abstracts the data distribution and usage, while providing still the flexibility for the user to implement other *Storage* classes and other *Content* classes. The *Storage* classes can also represent files, which abstracts input-output operations in the same manner.
- 5) ACAF incorporates the different kernel implementation into one object. And the object "knows" where and how to execute the code.

While the current implementation of the ACAF has the following limitations:

- 1) ACAF requires the usage of the extended C languages, like OpenCL and CUDA for utilizing GPUs.
- 2) The user should be aware that the computational code will be run simultaneously on different data and therefore the code should be reentrant.
- 3) ACAF provides only arrays as the content objects.

In the future, it's necessary to improve ACAF by extending it with the following features:

- Implement tree-structure content, which can be directly utilized for advanced SPH and NBody simulations.
- Implement the support of astrophysical-native file formats: Hierarchical Data Format version 5 (HDF5), Flexible Image Transport System (FITS), etc.
- Move ACAF implementation forward by introducing the domain specific language, which will eliminate the separate implementations for each technology.
- Implement partially synchronized arrays, enabling so even bigger data ranges.

## REFERENCES

- [1] R. Spurzem et al., "Accelerating astrophysical particle simulations with programmable hardware (FPGA and GPU)," *Computer Science - Research and Development*, vol. 23, no. 3-4, May 2009, pp. 231-239. [Online]. Available: <http://www.springerlink.com/index/10.1007/s00450-009-0081-9>
- [2] N. Nakasato, G. Ogiya, Y. Miki, M. Mori, and K. Nomoto. *Astrophysical Particle Simulations on Heterogeneous CPU-GPU Systems*. [Online]. Available: <http://arxiv.org/abs/1206.1199> [retrieved: Feb., 2016]
- [3] R. Spurzem et al., "Astrophysical particle simulations with large custom GPU clusters on three continents," *Computer Science - Research and Development*, vol. 26, no. 3-4, Apr. 2011, pp. 145-151. [Online]. Available: <http://www.springerlink.com/index/10.1007/s00450-011-0173-1>
- [4] T. Hamada and K. Nitadori, "190 TFlops Astrophysical N-body Simulation on a Cluster of GPUs," in *2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, no. November. IEEE, Nov. 2010, pp. 1-9.
- [5] S. Braibant, G. Giacomelli, and M. Spurio, *Particles and fundamental interactions: an introduction to particle physics*, 2nd ed. Springer, 2011.
- [6] L. Dagum and R. Menon, "Openmp: An industry-standard api for shared-memory programming," *IEEE Comput. Sci. Eng.*, vol. 5, no. 1, Jan. 1998, pp. 46-55. [Online]. Available: <http://dx.doi.org/10.1109/99.660313>
- [7] OpenACC. [Online]. Available: <http://www.openacc.org/> [retrieved: Feb., 2016]



- [8] W. Gropp, E. Lusk, and A. Skjellum, Using MPI (2Nd Ed.): Portable Parallel Programming with the Message-passing Interface. Cambridge, MA, USA: MIT Press, 1999.
- [9] H. Wang et al., “MVAPICH2-GPU: optimized gpu to gpu communication for infiniband clusters,” *Computer Science - Research and Development*, vol. 26, no. 3-4, 2011, pp. 257–266. [Online]. Available: <http://dx.doi.org/10.1007/s00450-011-0171-3>
- [10] Nvidia CUDA. [Online]. Available: [http://www.nvidia.com/object/cuda\\_home\\_new.html](http://www.nvidia.com/object/cuda_home_new.html) [retrieved: Feb., 2016]
- [11] Khronos Group. [Online]. Available: <http://www.khronos.org> [retrieved: Feb., 2016]
- [12] T. Goodale et al., “The Cactus framework and toolkit: Design and applications,” in *Vector and Parallel Processing – VECPAR’2002, 5th International Conference, Lecture Notes in Computer Science*. Berlin: Springer, 2003, pp. 197–227. [Online]. Available: <http://edoc.mpg.de/3341>
- [13] P. Jetley, L. Wesolowski, F. Gioachin, L. V. Kalé, and T. R. Quinn, “Scaling hierarchical n-body simulations on gpu clusters.” in *SC. IEEE*, 2010, pp. 1–11. [Online]. Available: <http://dblp.uni-trier.de/db/conf/sc/sc2010.html>
- [14] B. L. Chamberlain, “Chapel (cray inc. hpcs language).” in *Encyclopedia of Parallel Computing*, D. A. Padua, Ed. Springer, 2011, pp. 249–256. [Online]. Available: <http://dblp.uni-trier.de/db/reference/parallel/parallel2011.html>
- [15] A. Dubey et al., “The software development process of flash, a multiphysics simulation code.” in *SE-CSE@ICSE*, J. Carver, Ed. IEEE, 2013, pp. 1–8. [Online]. Available: <http://dblp.uni-trier.de/db/conf/icse/secse2013.html>
- [16] B. Fryxell et al., “FLASH: An adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes,” *Astrophys. J. Supp.*, vol. 131, Nov. 2000, pp. 273–334. [Online]. Available: <http://dx.doi.org/10.1086/317361>
- [17] S. Dindar et al., “Swarm-NG: a cuda library for parallel n-body integrations with focus on simulations of planetary systems,” *CoRR*, vol. abs/1208.1157, 2012, pp. 6–18. [Online]. Available: <http://dblp.uni-trier.de/db/journals/corr/corr1208.html>
- [18] S. P. Zwart, “The astronomical multipurpose software environment and the ecology of star clusters.” in *CCGRID. IEEE Computer Society*, 2013, p. 202. [Online]. Available: <http://dblp.uni-trier.de/db/conf/ccgrid/ccgrid2013.html>
- [19] The Enzo project. [Online]. Available: <http://enzo-project.org/> [retrieved: Feb., 2016]
- [20] J. Bezanon, A. Edelman, S. Karpinski, and V. B. Shah, “Julia: A fresh approach to numerical computing,” *CoRR*, vol. abs/1411.1607, 2014, p. 517. [Online]. Available: <http://arxiv.org/abs/1411.1607>
- [21] P. Charles et al., “X10: An object-oriented approach to non-uniform cluster computing,” *SIGPLAN Not.*, vol. 40, no. 10, Oct. 2005, pp. 519–538. [Online]. Available: <http://doi.acm.org/10.1145/1103845.1094852>
- [22] Fortress project. [Online]. Available: <http://projectfortress.java.net> [retrieved: Feb., 2016]
- [23] C. Feichtinger, S. Donath, H. Köstler, J. Götz, and U. Rüdé, “WaLBerla: HPC software design for computational engineering simulations,” *Journal of Computational Science*, vol. 2, no. 2, May 2011, pp. 105–112. [Online]. Available: <http://dx.doi.org/10.1016/j.jocs.2011.01.004>
- [24] I. Antcheva et al., “ROOT — a c++ framework for petabyte data storage, statistical analysis and visualization,” *Computer Physics Communications*, vol. 180, no. 12, 2009, pp. 2499 – 2512, 40 YEARS OF CPC: A celebratory issue focused on quality software for high performance, grid and novel computing architectures. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0010465509002550>
- [25] A. Lazzaro, S. Jarp, J. Leduc, A. Nowak, and L. Valsan, “Report on the parallelization of the MLfit benchmark using OpenMP and MPI,” CERN, Geneva, Tech. Rep. CERN-OPEN-2014-030, Jul 2012. [Online]. Available: <https://cds.cern.ch/record/1696947>