COMPUTATION TOOLS 2013

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

ISBN: 978-1-61208-277-6

May 27- June 1, 2013

Valencia, Spain

COMPUTATION TOOLS 2013 Editors

Torsten Ullrich, Fraunhofer Austria Research GmbH - Graz, Austria

Sandra Sendra, Polytechnic University of Valencia, Spain

COMPUTATION TOOLS 2013

Foreword

The Fourth International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking (COMPUTATION TOOLS 2013), held between May 27 and June 1, 2013 in Valencia, Spain, continued an event under the umbrella of ComputationWorld 2013 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.

We take here the opportunity to warmly thank all the members of the COMPUTATION TOOLS 2013 Technical Program Committee, as well as all the reviewers. We also kindly thank the authors who dedicated much of their time and efforts to contribute to COMPUTATION TOOLS 2013. 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 are grateful to the members of the COMPUTATION TOOLS 2013 organizing committee for their help in handling the logistics and for their work to make this professional meeting a success.

We hope that COMPUTATION TOOLS 2013 was a successful international forum for the exchange of ideas and results between academia and industry and for the promotion of progress in the areas of computational logics, algebras, programming, tools, and benchmarking.

We are convinced that the participants found the event useful and communications very open. We hope that Valencia, Spain provided a pleasant environment during the conference and everyone saved some time to explore this historic city.

COMPUTATION TOOLS 2013 Chairs:

COMPUTATION TOOLS General Chair

Vicente Casares-Giner, Polytechnic University of Valencia, Spain

COMPUTATION TOOLS Advisory Chairs

Kenneth Scerri, University of Malta, Malta
Jaime Lloret Mauri, Polytechnic University of Valencia, Spain
Radu-Emil Precup, "Politehnica" University of Timisoara, Romania

COMPUTATIONAL TOOLS Industry/Research Chairs

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

COMPUTATION TOOLS Publicity Chair

Sandra Sendra, Polytechnic University of Valencia, Spain

COMPUTATION TOOLS 2013

Committee

COMPUTATION TOOLS General Chair

Vicente Casares-Giner, Polytechnic University of Valencia, Spain

COMPUTATION TOOLS Advisory Chairs

Kenneth Scerri, University of Malta, Malta
Jaime Lloret Mauri, Polytechnic University of Valencia, Spain
Radu-Emil Precup, "Politehnica" University of Timisoara, Romania

COMPUTATIONAL TOOLS Industry/Research Chairs

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

COMPUTATION TOOLS Publicity Chair

Sandra Sendra, Polytechnic University of Valencia, Spain

COMPUTATION TOOLS 2013 Technical Program Committee

François Anton, Technical University of Denmark, Denmark
Henri Basson, University of Lille North of France (Littoral), France
Steffen Bernhard, TU-Dortmund, Germany
Ateet Bhalla, NRI Institute of Information Science and Technology, Bhopal, India
Paul-Antoine Bisgambiglia, Université de Corse, France
Narhimene Boustia, Saad Dahlab University - Blida, Algeria
Manfred Broy, Technical University of Munich, Germany
Luca Cassano, University of Pisa, Italy
Emanuele Covino, Università di Bari, Italy
Hepu Deng, RMIT University - Melbourne, Australia
Eugene Feinberg, Stony Brook University, USA
Tommaso Flaminio, Artificial Intelligence Research Institute (IIIA-CSIC) Spain
Janos Fodor, Obuda University, Hungary
Giuseppe Longo, Ecole Normale Supérieure Paris, France
Cynthia Vera Glodeanu, Institute of Algebra / Technische Universität Dresden, Germany
Luis Gomes, Universidade Nova de Lisboa, Portugal
Rajiv Gupta, University of California - Riverside, USA
Fikret Gurgen, Bogazici University - Istanbul, Turkey
Hani Hamdan, École Supérieure d'Électricité (SUPÉLEC), France
Cornel Klein, Siemens AG - Munich, Germany

Stano Krajci, Safarik University - Kosice, Slovakia
Giovanni Lagorio, DISI/University of Genova, Italy
Tsung-Chih Lin, Feng-Chia University, Taichung, Taiwan
Glenn R. Luecke, Iowa State University, USA
Elisa Marengo, Università degli Studi di Torino, Italy
Gianina Alina Negoita, Iowa State University, USA
Cecilia E. Nugraheni, Parahyangan Catholic University - Bandung, Indonesia
Flavio Oquendo, European University of Brittany/IRISA-UBS, France
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
Evgenia Smirni, College of William and Mary - Williamsburg, USA
James Tan, SIM University, Singapore
Torsten Ullrich, Fraunhofer Austria Research GmbH, Austria
Miroslav Velev, Aries Design Automation, USA
Zhonglei Wang, Karlsruhe Institute of Technology, Germany
Marek Zaremba, Universite du Quebec 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 or 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 is 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

Recognition of Java Source Code by Graph Matching Algorithm

Tomáš Bublík
Faculty of Nuclear Sciences and Physical Engineering
Czech Technical University in Prague
Prague, Czech Republic
e-mail: tomas.bublik@gmail.com

Miroslav Virius
Faculty of Nuclear Sciences and Physical Engineering
Czech Technical University in Prague
Prague, Czech Republic
e-mail: miroslav.virius@fjfi.cvut.cz

***Abstract*— This paper describes an option how to detect a desired Java code snippet in a large number of Java source files. The scripting language Scripthon is used to describe the desired section. Next, from this piece, an abstract tree is created, and it is compared to the other trees which are created from the Java source codes. The Java Compiler API is used to obtain the trees from the Java source codes. The final result of tree matching process is presented to a user.**

Keywords-abstract syntax tree; Java; Scripthon; trees matching; compiler API

I. INTRODUCTION

Searching source code is an easy task. Nevertheless, this applies only in the case of a simple text or simple structure names. This feature is supported in most of the current Java development environments. Some integrated development environments (hereinafter IDE) [11] [12] support an advanced searching with the regular expressions. But, what if a user wants to know, whether a program contains the singleton? Or, whether the specific method (with three concrete parameters) is somewhere in a program? It is very difficult to find such information; however, using the mathematical and programming knowledge, it is possible.

When using the Scripthon language [1], these special structures can be described very precisely. On the other hand, by using the Java Compiler application programming interface (hereinafter API), the abstract syntax trees (hereinafter AST) can be obtained and compared with Scripthon output. This paper is about using these trees for searching the desired code snippet. This task is similar to the graph matching and isomorphic sub-graphs finding in a large set of trees. An additional problem arises in the applications where an input graph needs to be matched not only to another graph, but to an entire database of graphs under a given matching paradigm. Therefore, some complexity reducing algorithms are proposed in this paper.

There are several reasons to consider graphs to be very advantageous tool for the representation of source code of some language. One reason is that there is no unnecessary material like spaces, comments, etc. Another reason is that there are many well described mathematical algorithms to work with graphs. Some of the algorithms are known for decades. Representing the code as a graph has also the disadvantage: it has large demands on a computer power and memory; especially for larger programs.

The first section compares existing similar solutions with this one. Several tools with the similar function are mentioned there. The next section introduces necessary graph theory concepts. The definitions of a graph, a sub-graph and a graph isomorphism are given. The Scripthon language is introduced briefly in fourth chapter. Because the language has been described already in another paper [1], only the important properties are mentioned here. The next two sections are about graphs generation, optimizations, and the comparison of graphs generated by the Compiler API. An algorithm for trees matching can be found in Section 6. Finally, several results are presented in the conclusion.

II. COMPARATION WITH SIMILAR SOLUTIONS

There are many approaches to the code search area. These approaches can be classified as textual, lexical, tree-based, metrics-based and graph-based. This distribution depends on how the source code is expressed. More on this topic can be found in [6]. Scripthon belongs to the tree-based solutions.

A number of similar solutions for all the mentioned tasks have been proposed in [6]; however, Scripthon is quite different tool. This tool is not supposed to detect the clones automatically. However, it is possible with the assistance of the user,. Our previous work dealt with automatic detection and removal of clones in Java source code [2]. Finally, with respect to other solutions and a complexity of this topic, we decided to try another way. In addition, we considered that the detection and removal of the so-called “non-ideal” clones is very difficult without some additional information from a user. (The “non-ideal” clones are repeated pieces of source code that are not exactly the same, but execute similar operations.) The Scripthon is primarily designed to search known patterns in source code. It means that the user must approximately know how the clone looks like. Then, he or she creates a script based on his or her ideas which finds the desired patterns.

A similar solution is described in [7]. Refactoring NG is an interesting tool which allows defining a refactoring operation programmatically; however, it cannot be used for defining the searching patterns.

In addition, Scripthon is not aimed to detect design patters. With Scripthon, it is possible to find a simple design pattern within one class (for example, the above mentioned Singleton), but it is not its main purpose. It is not possible to find a design pattern composed of multiple classes.

 ISBN: 978-1-61208-277-6

Unlike regular expressions, Scripthon offers an interesting alternative to search a shape or properties of a given Java source code.

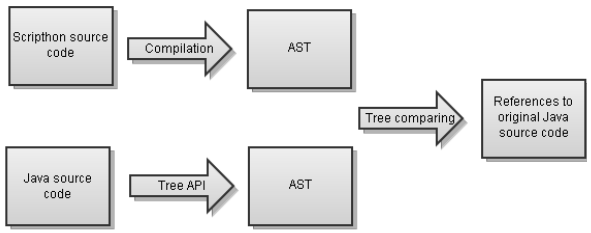

Figure 1. Complete process

Figure 1 shows the whole process of searching Java source snippets using Scripthon. The tool runs in two threads. The first one is a Scripthon compiler. Its output is a modified AST. The second thread is aimed to create a Java AST. Both trees are compared with the matching algorithm. A result of the process is the references to a given Java source code. Typically, it is the name of a Java class and the line number where the match occurred.

III. BASIC GRAPH THEORY CONCEPTS

A graph is a four-tuple $g = (V, E, \alpha, \beta)$, where V denotes a finite set of nodes, $E \subseteq V \times V$ is a finite set of edges, $\alpha : V \longrightarrow L_V$ is a node labeling function, and $\beta : E \longrightarrow L_E$ is an edge labeling function. L_V and L_E are finite of infinite sets of node and edge labels, respectively.

All the graphs in this work are considered to be directed. A subgraph $g_s = (V_S, E_S, \alpha_S, \beta_S)$ of a graph g is a subset of its nodes and edges, such that $V_S \subseteq V, E_S \subseteq E \cap (V_S \times V_S)$ Two graphs g and g' are isomorphic to each other if there exists a bijective mapping u from the nodes of g to the nodes of g, such that the structure of the edges as well as all node and edge labels are preserved under u . Similarly, an isomorphism between a graph g and a subgraph g' of a graph g' is called subgraph-isomorphism form g to g' .

A tree is a connected and undirected graph with no simple circuits. Since a tree cannot have a circuit, a tree cannot contain multiple edges or loops. Therefore, any tree must be a simple graph. An undirected graph is a tree if and only if there is a unique simple path between any two of its vertices.

The graph matching problem is actually the same as the problem of finding the isomorphism between the graphs. Moreover, matching the parts of a graph with a pattern is the same challenge as the finding the isomorphic subgraph.

IV. SCRIPHON DESCRIPTION

The Scripthon language is widely described in [1]. The following text will present only the summary of important properties of this language. Scripthon is a simple-to-learn language which is able to describe a Java source code structure. Because of its simple syntax, it is very easy to learn. The syntax of the Scripthon language is similar to the syntax of Java, and it is very intuitive. Basically, the keywords represent the structures in Java language. Thus, a Scripthon program is built only with these words and its properties. Each keyword has a special set of its own properties. There are three sets defining the usable keyword, its properties and the properties values. For example, this is the set of structural keywords (Str):

$$\mathrm{Str} = \{\mathrm{Meth}, \mathrm{Init}, \mathrm{Block}, \mathrm{Class}, \ldots\}$$

For a Class() keyword, the set of parameters (SAtr) looks like:

$$\mathrm{SAtr} = \{\mathrm{Name}, \mathrm{Rest}\}$$

For these parameters, the set of available values (AVal) is:

$$\mathrm{AVal} = \{\mathrm{public}, \mathrm{static}, \mathrm{private}, \mathrm{interface}\}$$

For example, a class is represented by a Class() keyword. The parameters of this keyword can be in the parentheses, however, if the brackets include no parameters, each class is a candidate for searching and each class of a given program corresponds to this structure. For example, the following command:

```
Class(Name = "Main"; Rest = public)
```

means that the wanted structure is a public class with the name Main. The options of the parameters are specified in the Scripthon documentation. The structure nesting it is denoted only by the line separators.

```
Meth(Rest = private; ParamsNum = 2)
  Block()
    Init(Type = int; Value = ""; Name = "sum")
```

This example means that the searched structure is a private method with two parameters. Inside the method is a block with two statements. The first statement is a variable named sum of type int. The second statement is a return statement with a parameter of the previously specified variable.

The big advance of the Scripthon language is the ability to describe the elements with a variable depth of details. This means that the searched structures can be described in a detail or very loosely. For example, this is a very detailed description:

```
Class(Name = "TestDecompile"; Rest = public)
  Meth(Name = "main"; Ret = void; Rest = public)
    Init(Name = "toPrintValue"; Type = String)
      MethCall(Name = "System.out.println")
```

The same script without details follows:

```
Class()
  Meth()
    Init()
      MethCall()
```

Therefore, a searched subject can be found on the base of a very inaccurate description. The results can be obtained with the iterative refinement of the input conditions. In the end, the user can get better results.

Furthermore, Scriphon contains a special keyword Any(). It is not a structural keyword, but it is information for the matching algorithm to act as anything. When used, it means that a searching structure could be anything (even with the sub-trees) or nothing. With respect to the previous example, the desired structure can be described even with this script:

```
Class()
  Meth()
```

However, because of the generality of this script, the number of results found will be very high. (Actually, any class corresponds to this script)

The level of detail which can be described by the current version of Scripthon is up to – but not including – the expression. In addition, Scripthon can describe a lot of Java structures, but it cannot describe the individual elements of an expression statement. For example, while describing the if statement, it is possible to address the inner block, or the else block with inner statements; however, the if-expression in the parentheses cannot be described. Moreover, Scripthon is not able to describe the mathematical operations. If a variable i is declared such as:

```
int i = a + b;
```

The most accurate Scripthon statement to find is:

```
Init(Name = "i"; Type = int)
```

In the current version of Scripthon, nothing more can be described. On the other hand, this language is designed to be extensible. The main program consists of several modules appropriate to corresponding stages of searching process.

Current version of the language cannot describe all the Java language structures. For example, annotations, generics, diamond operators, and many others are omitted, but they can be easily added in future versions. It would be necessary to introduce new structural word, define its properties, and define the rules to the searching algorithm. There is no need to change syntax, or even the compiler.

V. GRAPH GENERATION WITH JAVA COMPILER API

The Java Compiler API is used to get a graph for the searching algorithm. This API is free, and it is included in the Java distribution. Basically, the Java Compiler API serves to the advanced control of a compilation process. This API uses the AST and the visitor design pattern. Unfortunately, this design pattern is not suitable for searching purposes. This is because the Scripthon language cannot to describe such a quantity of structures, and also because the searching algorithm is not suitable for the implementation with the visitor design pattern. Therefore, the more advanced graph is created from a Java AST. This graph is based on the AST, but it has a several benefits.

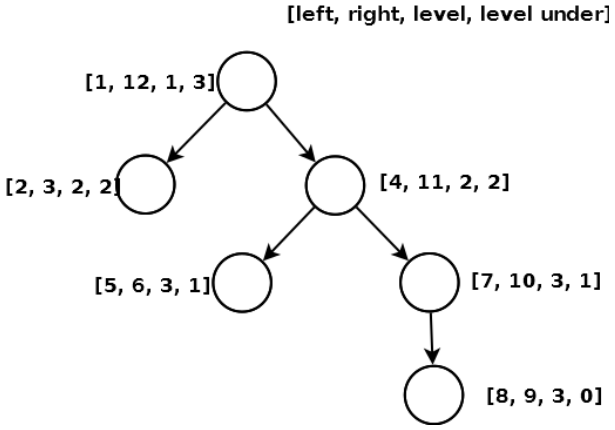

Figure 2. Tree with optimizations

The first benefit is the replacement of the visitor pattern with the classic approach. The second one is that some additional information is included, which significantly facilitates the searching.

While browsing the Java source code, the tree with the nodes enhanced by four numbers is created. These numbers are the natural numbers named left, right, level and level under. The first and the second number (left, right) denote the order index of a node in the tree preorder traversal. Therefore, an ancestor's left index is always smaller than its children left index, while the right index is always bigger than any children's right index. The level number denotes the level in a tree hierarchy of vertices, and the level under number denotes a number of levels under the current node (compare with the method described in [4]).

Suppose that x and y are two nodes from a tree; the following rules are valid for these values.

- The y node is an ancestor of x and x is a descendant of y if $\mathrm{y.left} < x.\mathrm{left} < y.right$
- The y node is a parent of x and x is a child of y if 1) $\mathrm{y.left} < x.\mathrm{left} < y.right$ and 2) $\mathrm{y.level} = \mathrm{x.level} - 1$
- The node x has $((\mathrm{x.left} - \mathrm{x.right}) - 1)$ sub-nodes.

All these data are acquired during a single pass through the tree. Obtaining this information is not a time consuming operation, because it is made during the tree production process. On the other hand, the number of comparisons can be significantly reduced with these numbers. Moreover, while comparing the trees, it is very easy to detect:

- How many elements have a given structure
- Whether a node is a leaf
- How many sub-statements are included in a given structure

The comparison of two trees is much more time consuming without this information. In summary, this information is used in cases where the shape of the given structures and its coupling is considered more than its properties.

A line reference to source code is important information which is also added to the tree as a metadata. Therefore, it is easy to link the results with the original source position and show it to the user. There are some more elements in a node metadata. For example, some of the other metadata information is a filename of the source file.

Because the number of the comparisons is a key indicator for the algorithm speed, it is necessary to keep the number of nodes as small as possible. Therefore, only the supported structures and their properties are considered while creating a tree from source code. Thus, the same Scripthon definition set is used during the tree creation process. Other elements are omitted.

VI. GRAPH MATCHING

The simple and many times described backtracking algorithm is used for the graph matching. Basically, it is the problem of finding an isomorphic tree to the given tree from a large database of trees. Comparing to the common tree matching, there are two differences. The first one is that the node properties need to be considered during the process. The second difference is that not every Scripthon node corresponds exactly to one Java structure node. For example, the already mentioned keyword Any() could correspond to more nodes.

The source trees are created from the corresponding classes. The classes and the trees are mapped one-by-one. Each tree corresponds to exactly one class. In the first step, the algorithm checks whether the shape of the structure match, and then the properties are compared. This is because the properties matching is much more time consuming operation than shape detection. Many structures are eliminated very quickly from the process in the case that the shape does not fit.

If the shape of the structure corresponds to the required shape, the structure parameters are compared. All the parameters of a given node must be met. The node properties are provided by the Java compiler.

```
1.for (Class c) //iterate over all classes from given sources
2.   match = true
3.   for (Statement s)
4.      match = compare(s, c.parentNode)
5.      if (match == false)
6.         break
7.      if (match)
8.         add it to the list of founded structures
9.
10.boolean compare(Statement s, Node n)
11.   match = true
12.   if (s.properties match n.properties)
13.      for (s.children, n.children)
14.         compare(s.child, n.child)
15.      if (match == false)
16.         return false
17.      else
18.         return true
19.   else
20.      return false
21.   return true
```

Figure 3. Simplified tree matching algorithm.

Figure 3 shows the simplified matching algorithm. It is written in Java pseudo-code. The algorithm skeleton is similar to the algorithm described in [8]. The main difference is that in our solutions are compared not two Java trees, but a Java tree and a Scriphon tree. The whole program iterates over all given Java classes (line 1in the figure 3). Instead of finding a corresponding sub-tree, the algorithm tries to exclude quickly a mismatching part. It can be seen from line 2.

At the beginning, it is assumed that the given source matches. The rest of the algorithm iterates over Scripthon statements (line 3) and tries to find a match between a statement and a node of a Java AST (line 4). A matching method (line 10) is called recursively as the sub-nodes are traversed. If a result of this method is false, the loop over Scripthon statements is interrupted (line 6), because even the first statement does not correspond to anything of a Java class. The result of the "compare" method is true (line 21) or false (line 20). A statement and a node are equal if all their corresponding properties are equal (line 12) and all the children are equal (line 13). Therefore, all children are iterated and compared recursively (line 14). If a match is found, this method returns true (line 18). Otherwise it returns false (line 16). If true, the result is added to the result list.

Many aspects are considered during properties matching process. Not only keywords and Java nodes properties are considered. According to the previous section, it is possible to exclude quickly the mismatched parts, because some additional data are known about a shape of the sub-tree.

The typical size of a class graph depends on the source size and on the number of supported structures. About 80 nodes of the graph are created from a Java class with length about 200 lines nodes in the current version of Scripthon. In future versions, when more structures will be supported, may the number of the nodes significantly increase.

 ISBN: 978-1-61208-277-6

Unfortunately, because all the Java classes with all their nodes must be compared with all the Scripthon statements, the number of complexity rapidly grows. According to [3], the sub-graph isomorphism problem has $O(N^3)$ complexity in worst case. Since the number of occurrences can be more than one, each class must be browsed more than once. Each class needs to be traversed until the number of results is 0. According to [9, 10] the graph isomorphism problem is polynomial. Therefore, even in this case, the complexity of our algorithm remains polynomial. On the other hand, with the above outlined optimizations, the number of node comparisons is significantly decreased. More on the similar graph matching techniques can be found in [5].

VII. MEASUREMENTS RESULTS

The used algorithm modifications substantially reduced the time needed to find the requested Java structures. Moreover, also the time of the tree generation procedure has been shortened. According to the measurements, the meta-information counting does not significantly affect the time of a graph creation.

The searching with optimization is much faster. The following tables show the measured time results. The small program means a program consisting of approximately 20 to 30 classes, while the larger program is a program with approximately 100 to 150 classes. There are also the results before and after the described optimizations.

TABLE I. Graph creation times

Program type	Time
Small program (no optimizations)	412 ms
Larger program (no optimizations)	4 423 ms
Small program (optimized)	132 ms
Larger program (optimized)	337 ms

TABLE II. Searching times

Program type	Time
Small program (no optimizations)	2 345 ms
Larger program (no optimizations)	11 236 ms
Small program (optimized)	753 ms
Larger program (optimized)	1 986 ms

TABLE III. Total times

Program type	Time
Small program (no optimizations)	2 757 ms
Larger program (no optimizations)	15 659 ms
Small program (optimized)	886 ms
Larger program (optimized)	2 323 ms

The measurements were performed on the quite common computer. The computer configuration was: 4GB of memory, the Intel Core I5 processor with a frequency of 2.4 GHz and Windows 7 as an operating system. The individual results represent the averages of several consecutive measurements. The first column indicates the time needed to the AST generation, while the second one represents the time required to find a piece of the sample code described by the Scripthon language. The last column is the sum of both times. The lines represent the sizes of programs on which the measurements were performed.

As it is shown in the tables, in case of the small program, the graph assembling is not significantly different. On contrary, better results can be obtained in the case of larger programs. Probably, this is because some time is needed for the overhead services related to the starting and initializing the own search.

VIII. CONCLUSION

With the described solution, we proved that the proposed concept of searching is possible. Moreover, it is also very effective. With used optimizations, the algorithm significantly improved performance of a whole process. Next, the Scripthon project is designed as a modular system. Therefore, as will the functionality requirements grow, it is not difficult to add more supported Java structures. Even the language itself could be enhanced by new syntax elements very easily. There are many possibilities of how Scripthon could be used. One of the planned usage areas is a student's work controlling task. With Scripthon, it is easy to detect whether a student's work contains prescribed programming structures.

The Scripthon language improved. The Scripthon compiler is available as a command line tool now. We suppose to develop the Scripthon plug-in for some popular integrated development environments in the future.

ACKNOWLEDGMENT

This work is supported by the SGS 11/167 grant of the Ministry of Education, Youth and Sports of the Czech Republic.

REFERENCES

[1] T. Bublík and M. Virius.: "New language for searching Java code snippets," in: ITAT 2012. Proc. of the 12th national conference ITAT. diar, Sep 17 – 21 2012. Pavol Jozef Safrik University in Kosice, pp. 35 – 40.

[2] T. Bublík and M. Virius: "Automatic detecting and removing clones in Java source code," in: Software Development 2011. Proc. of the 37th national conference Software Development. Ostrava, May 25 – 27 2011. Ostrava: Technical University of Ostrava 2011. ISBN 978-80-248-2425-3, pp. 10 – 18.

[3] I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier. 1998. "Clone Detection Using Abstract Syntax Trees," in Proceedings of the International Conference on Software Maintenance (ICSM '98). IEEE Computer Society, Washington, DC, USA, pp. 368-377.

[4] J. T. Yao and M. Zhang. 2004. "A Fast Tree Pattern Matching Algorithm for XML Query," in Proceedings of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence (WI '04). IEEE Computer Society, Washington, DC, USA, pp. 235-241.

 ISBN: 978-1-61208-277-6

[5] H. Bunke, Ch. Irniger, and M. Neuhaus. 2005. "Graph matching – challenges and potential solutions," in Proceedings of the 13th international conference on Image Analysis and Processing (ICIAP'05), Fabio Roli and Sergio Vitulano (Eds.). Springer-Verlag, Berlin, Heidelberg, pp. 1-10.

[6] Ch. K. Roy, J. R. Cordy, and R. Koschke. 2009. "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach," Sci. Comput. Program. 74, 7 (May 2009), pp. 470-495.

[7] Z. Troníček. 2012. "RefactoringNG: a flexible Java refactoring tool," in Proceedings of the 27th Annual ACM Symposium on Applied Computing (SAC '12). ACM, New York, NY, USA, pp. 1165-1170.

[8] W. Yang. 1991. "Identifying syntactic differences between two programs," Softw. Pract. Exper. 21, 7 (June 1991), pp. 739-755.

[9] J. Köbler and J. Torán. 2002. "The Complexity of Graph Isomorphism for Colored Graphs with Color Classes of Size 2 and 3," In Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science (STACS '02), Helmut Alt and Afonso Ferreira (Eds.). Springer-Verlag, London, UK, UK, pp. 121-132.

[10] I. S. Filotti and J. N. Mayer. 1980. "A polynomial-time algorithm for determining the isomorphism of graphs of fixed genus," In Proceedings of the twelfth annual ACM symposium on Theory of computing (STOC '80). ACM, New York, NY, USA, pp. 236-243.

[11] T. Boudreau, J. Glick, and V. Spurlin, "NetBeans: The Definitive Guide," Sebastopol, CA, USA: O'Reilly & Associates, Inc., 2002.

[12] S. Holzner, "Eclipse," O'Reilly Media, April 2004.

 ISBN: 978-1-61208-277-6

On an Inference System for a Hybrid Process Calculus

Zining Cao[1,2]

[1]State Key Laboratory for Civil Aircraft Flight Simulation
Shanghai Aircraft Design and Research Institute
Shanghai 201210, China
[2]Department of Computer Science and Technology
Nanjing University of Aeronautics & Astronautics
Nanjing 210016, China

***Abstract*—In this paper, we propose a hybrid process calculus. This hybrid process calculus can be used to describe hybrid properties and nondeterministic properties of software. The concrete bisimulation and symbolic bisimulation of this hybrid process calculus are proposed. We then prove the equivalence between these two bisimulation. An inference system for the symbolic bisimulation of this hybrid process calculus is given. At last, we prove the soundness and completeness of the inference system.**

***Keywords*—hybrid process calculus; symbolic bisimulation; inference system**

I. INTRODUCTION

Hybrid system is a kind of mixed discrete-continuous system. A paradigmatic example of a mixed discrete-continuous system is a digital controller of an analog plant. The discrete state of the controller is modelled by the vertices of a graph (control modes), and the discrete dynamics of the controller is modelled by the edges of the graph (control switches). The continuous state of the plant is modelled by points in $\mathbf{R}^n$, and the continuous dynamics of the plant is modelled by flow conditions such as differential equations. The behavior of the plant depends on the state of the controller: each control mode determines a flow condition, and each control switch may cause a discrete change in the state of the plant, as determined by a jump condition. Dually, the behavior of the controller depends on the state of the plant: each control mode continuously observes an invariant condition of the plant state, and by violating the invariant condition, a continuous change in the plant state will cause a control switch.

There are several works on models of hybrid systems such as [2], [3], [4], [7], and [12]. But there are seldom works on sound and complete inference systems for bisimulation of hybrid systems. For examples, some sound inference systems for bisimulation were given in [2] and [4], whereas these inference system were not proved complete. In this paper, we aim to propose a sound and complete inference system for bisimulation of hybrid systems. To this end, we firstly present a hybrid process calculus including its syntax, operational semantics and concrete bisimulation in this paper. Then the symbolic labelled transition system and symbolic bisimulation are also presented. We prove the equivalence of concrete bisimulation and symbolic bisimulation. Furthermore, we present an inference system for symbolic bisimulation. Finally, the soundness and completeness of this inference system are studied.

This paper is organized as follows: Section 2 gives a hybrid process calculus including its syntax, operational semantics and concrete bisimulation. In Section 3, we propose a symbolic theory for this hybrid process calculus including symbolic labelled transition system and symbolic bisimulation for hybrid process calculus. Furthermore, we prove the equivalence between concrete bisimulation and symbolic bisimulation. In Section 4, we give a complete inference system for this hybrid process. The soundness and completeness of the inference system are also proved. The paper is concluded in Section 5.

II. HYBRID PROCESS CALCULUS

There are many works about process algebras for hybrid systems, for example, [2], [3], [4], [7], and [12]. A comparative study of these process algebras is referred to [9]. The main aim of this paper is to propose a sound and complete inference for bisimulation of hybrid systems. To this end, we present a simple hybrid process calculus which has a relatively small number of operators and a simpler semantics. Therefore it is easier to give a complete inference system than other process algebras. The syntax, operational semantics and concrete bisimulation of this process calculus are given in this section.

A. Syntax of Hybrid Process Calculus

To give the syntax of hybrid process calculus, we first present the syntax and semantics of predication logical formulas.

Predication logical formulas are defined by the following grammar:

$\Phi, \Psi ::= x \bowtie u(x_1, ..., x_n) \mid \neg\Phi \mid \Phi \wedge \Psi \mid \forall x.\Phi$, where $\bowtie \in \{=, \neq, \geq, >, <, \leq\}$, x is a variable, $u(x_1, ..., x_n)$ is a real function with parameters $x_1, ..., x_n$, i.e., $u(r_1, ..., r_n) = r$ where $r_1, ..., r_n, r \in \mathbf{R}$ and $\mathbf{R}$ is the set of real numbers. We denote the set of variable $\{t, v_1, v_2, ..., v_n, ...\}$ as Var^0, denote the set of variable $\{t', v_1', v_2', ..., v_n', ...\}$ as Var', and denote $Var^0 \cup Var' = Var$. Informally, variables $v_1', ..., v_n'$

represent the new values taken by the variables $v_1, ..., v_n$ after a transition. Variable t represents the time variable.

The satisfiability relation $\models$ is defined between assignment θ and formula Φ as follows, where θ is a function such that the domain of θ is a subset of Var and the range of θ is $\mathbf{R}$, and free variables in Φ is in the domain of θ.

(1) $\theta \models x \bowtie u(x_1, ..., x_n)$ if $\theta(x) \bowtie u(\theta(x_1), ..., \theta(x_n))$;

(2) $\theta \models \neg\Phi$ if $\theta \not\models \Phi$;

(3) $\theta \models \Phi \wedge \Psi$ if $\theta \models \Phi$ and $\theta \models \Psi$;

(4) $\theta \models \forall x.\Phi$ if $\theta \models \Phi\{a/x\}$ for any a, where $\{s/t\}$ means replacing t by s.

We write $\Phi \models \Psi$ to mean that $\theta \models \Phi$ implies $\theta \models \Psi$ for any θ, and write $\models \Psi$ to mean that $\theta \models \Psi$ for any θ.

The formal definition of process is given as follows:

$P ::= 0 \mid X \mid P + P \mid \varepsilon(\Phi_1, \Phi_2).P \mid a(\Psi_1, \Psi_2).P \mid fixX.P$, where Φ_1 is a predication logical formulas with free variables in Var^0; Φ_2 is a predication logical formulas with free variables in Var; ε is an internal action which is invisible for observer; Ψ_1 is a predication logical formulas with free variables in Var^0; Ψ_2 is a predication logical formulas with free variables in Var; a is an external action which is visible for observer; all process variables in $fixX.P$ are guarded by action prefix. The class of processes is denoted as Pr.

Informally, 0 denotes inaction. $P_1 + P_2$ expresses nondeterministic choice of processes P_1 and P_2. $\varepsilon(\Phi_1, \Phi_2).P$ can perform an internal action ε under condition Φ_1, then continues as P, and the change of variables satisfies Φ_2. $a(\Psi_1, \Psi_2).P$ can perform an external action a under condition Ψ_1, then continues as P, and the change of variables satisfies Ψ_2. $fixX.P$ is a recursive definition of process.

B. Labelled Transition System of Hybrid Process Calculus

The operational semantics of hybrid process calculus is given in Table 1. We have omitted the symmetric rule of the nondeterministic operator.

The labelled transition system consists of a collection of relations of the form $\langle P, \rho\rangle \xrightarrow{\varepsilon(\Phi_1,\Phi_2),T} \langle Q, \sigma\rangle$ or $\langle P, \rho\rangle \xrightarrow{a(\Psi_1,\Psi_2)} \langle Q, \sigma\rangle$, where P, Q are processes, and ρ, σ are configurations. A configuration is a function ρ such that $\rho(x) \in \mathbf{R}$ for any $x \in Var^0$. A configuration represents an possible assignment of variables. The class of configurations is denoted as C. The transition $\langle P, \rho\rangle \xrightarrow{\varepsilon(\Phi_1,\Phi_2),T} \langle Q, \sigma\rangle$ means that the process P at configuration ρ can realize the action $\varepsilon(\Phi_1, \Phi_2)$, and becomes Q at configuration σ after T units of time. The transition $\langle P, \rho\rangle \xrightarrow{a(\Psi_1,\Psi_2)} \langle Q, \sigma\rangle$ means that the process P at configuration ρ can realize the action $a(\Psi_1, \Psi_2)$, and becomes Q at configuration σ. We denote by $\rho[x \leftarrow U]$ a new configuration that is the same as ρ except that $\rho[x \leftarrow U](x) = U$, and denote by $\rho[x \Leftarrow x']$ a function such that $f(x') = \rho(x)$ for any x in the domain of ρ. In the following, for function $\rho : X \to \mathbf{R}$ and function $\sigma : Y \to \mathbf{R}$ with condition $X \cap Y = \emptyset$, we use $\rho \cup \sigma$ to denote function f such that $f(x) = \rho(x)$ when $x \in X$ and $f(x) = \sigma(x)$ when $x \in Y$.

$$TAU : \langle \varepsilon(\Phi_1, \Phi_2).P, \rho\rangle \xrightarrow{\varepsilon(\Phi_1,\Phi_2),T} \langle P, \rho'\rangle, \text{ where } T \in \mathbf{R}, \forall \delta \in [0, T].\rho[t \leftarrow \rho(t) + \delta] \models \Phi_1,\ \rho'(t) = \rho(t) + T, \rho \cup (\rho'[x \Leftarrow x']) \models \Phi_2.$$

$$ACT : \langle a(\Psi_1, \Psi_2).P, \rho\rangle \xrightarrow{a(\Psi_1,\Psi_2)} \langle P, \rho'\rangle, \text{ where } \rho \models \Psi_1, \rho \cup (\rho'[x \Leftarrow x']) \models \Psi_2.$$

$$SUM : \frac{\langle P_1, \rho\rangle \xrightarrow{\alpha} \langle P_1', \rho'\rangle}{\langle P_1 + P_2, \rho\rangle \xrightarrow{\alpha} \langle P_1', \rho'\rangle}$$

$$REC : \frac{\langle P\{fixX.P/X\}, \rho\rangle \xrightarrow{\alpha} \langle P', \rho'\rangle}{\langle fixX.P, \rho\rangle \xrightarrow{\alpha} \langle P', \rho'\rangle}$$

Table 1: Operational semantics of hybrid process calculus

An example: Let process $P = \varepsilon(\Phi_1, \Phi_2).a(\Psi_1, \Psi_2).0 = \varepsilon(x_1 + x_2 + t \le 2y + 1,\ x_1' = tx_1 \wedge x_2' = 2x_2 \wedge y' \le 2x_2' - x_1).a(x_1 + x_2 \ge y,\ x_1' = 2x_1 \wedge x_2' = x_2).0$. Then at configuration ρ such that $\rho(t) = 0$, $\rho(x_1) = 1$, $\rho(x_2) = 2$, $\rho(y) = 3$,we have $\langle P, \rho\rangle \xrightarrow{\varepsilon(\Phi_1,\Phi_2),2} \langle P', \rho'\rangle$, where $P' = a(\Psi_1, \Psi_2).0 = a(x_1 + x_2 \ge y, x_1' = 2x_1 \wedge x_2' = x_2).0$, ρ' is a configuration such that $\rho'(t) = 2$, $\rho'(x_1) = 2$, $\rho'(x_2) = 4$, $\rho'(y) \le 6$. Furthermore, $\langle P', \rho'\rangle \xrightarrow{a(\Psi_1,\Psi_2)} \langle 0, \rho''\rangle$, where ρ'' is a configuration such that $\rho''(t) = 2$, $\rho''(x_1) = 4$, $\rho''(x_2) = 4$, $\rho''(y) \in \mathbf{R}$.

C. Concrete Bisimulation

Now we propose a concrete bisimulation for hybrid process calculus. Intuitively, P and Q are concrete bisimilar if whenever P can perform an action under the configuration ρ, Q can also perform the same action under the configuration ρ.

Definition 2. A symmetric relation $R \in (Pr \times C) \times (Pr \times C)$ is called a concrete bisimulation if whenever $\langle P, \rho\rangle\ R\ \langle Q, \rho\rangle$,

(1) $\langle P, \rho\rangle \xrightarrow{\varepsilon(\Phi_1^P,\Phi_2^P),T} \langle P', \rho'\rangle$ implies that there exists Q' such that $\langle Q, \rho\rangle \xrightarrow{\varepsilon(\Phi_1^Q,\Phi_2^Q),T} \langle Q', \rho'\rangle$ and $\langle P', \rho'\rangle\ R\ \langle Q', \rho'\rangle$;

(2) $\langle P, \rho\rangle \xrightarrow{a(\Psi_1^P,\Psi_2^P)} \langle P', \rho'\rangle$ with $a \neq \varepsilon$ implies that there exists Q' such that $\langle Q, \rho\rangle \xrightarrow{a(\Psi_1^Q,\Psi_2^Q)} \langle Q', \rho'\rangle$ and $\langle P', \rho'\rangle\ R\ \langle Q', \rho'\rangle$.

We write $\langle P, \rho\rangle \sim \langle Q, \rho\rangle$ if there is a concrete bisimulation R such that $\langle P, \rho\rangle\ R\ \langle Q, \rho\rangle$.

We write $P \sim_C Q$ if $\langle P, \rho\rangle \sim \langle Q, \rho\rangle$ for any ρ.

Remark: In the above definition, we do not require that Φ_1^P and Φ_1^Q (Φ_2^P and Φ_2^Q, or Ψ_1^P and Ψ_1^Q, or Ψ_2^P and Ψ_2^Q) are logical equivalent since by the operational semantics of hybrid process calculus, $\langle P, \rho\rangle \xrightarrow{\varepsilon(\Phi_1^P,\Phi_2^P),T} \langle P', \rho'\rangle$ is permitted if $\forall \delta \in [0, T].\rho[t \leftarrow \rho(t) + \delta] \models \Phi_1^P$, and $\langle Q, \rho\rangle \xrightarrow{\varepsilon(\Phi_1^Q,\Phi_2^Q),T} \langle Q', \rho'\rangle$ is permitted if $\forall \delta \in [0, T].\rho[t \leftarrow \rho(t) + \delta] \models \Phi_1^Q$, which means $\forall \delta \in [0, T].\rho[t \leftarrow \rho(t) + \delta] \models \Phi_1^P \leftrightarrow \Phi_1^Q$. Therefore the logical equivalent relation between Φ_1^P and Φ_1^Q is implied by the side condition of operational semantics of hybrid process calculus. The cases of Φ_2^P and Φ_2^Q, Ψ_1^P and Ψ_1^Q, Ψ_2^P and Ψ_2^Q are similar.

 ISBN: 978-1-61208-277-6

III. A SYMBOLIC THEORY FOR HYBRID PROCESS CALCULUS

In this section, a symbolic labelled transition system and a symbolic bisimulation equivalence are presented. The full abstraction property, i.e., the equivalence between this symbolic bisimulation and the concrete bisimulation, is shown. The symbolic semantics is necessary for an efficient implementation of the calculus in automated tools exploring state spaces, and the full abstraction property means processes are bisimilar in the symbolic setting if they are bisimilar in the original semantics.

A. Symbolic Labelled Transition System

The symbolic operational semantics of hybrid process calculus is given in Table 2. We have omitted the symmetric of the nondeterministic. The labelled transition system consists of a collection of relations of the form $P \xrightarrow{\Gamma,(\varepsilon(\Phi_1,\Phi_2),T),\Gamma'} Q$ or $P \xrightarrow{\Gamma,a(\Psi_1,\Psi_2),\Gamma'} Q$. The transition $P \xrightarrow{\Gamma,(\varepsilon(\Phi_1,\Phi_2),T),\Gamma'} Q$ means that the process P can realize the action $\varepsilon(\Phi_1,\Phi_2)$ if condition Γ is true, and becomes Q where Γ' is true after T units of time. The transition $P \xrightarrow{\Gamma,a(\Psi_1,\Psi_2),\Gamma'} Q$ means that the process P can realize the action $a(\Psi_1,\Psi_2)$ if condition Γ is true, and becomes Q where Γ' is true.

In the following, we use $(\exists \vec{X} .\Phi)\{\vec{X} / \vec{X'}\}$ to abbreviate $(\exists x_1...\exists x_m.\Phi)\{x_1,...,x_m/x'_1,...,x'_m\}$, where the set of free variables in Φ is $\{x_1,...,x_m,x'_1,...,x'_m\}$.

$$TAU : \varepsilon(\Phi_1,\Phi_2).P \xrightarrow{\Gamma,(\varepsilon(\Phi_1,\Phi_2),T),\Gamma'} P, \text{ where } T \in \mathbf{R},$$
$$\models \forall \delta \in [0,T].\Gamma\{t+\delta/t\} \rightarrow \Phi_1\{t+\delta/t\},$$
$$\models (\exists \vec{X} .(\Gamma \wedge \Phi_2 \wedge t' = t+T))\{\vec{X} / \vec{X'}\} \rightarrow \Gamma'.$$

$$ACT : a(\Psi_1,\Psi_2).P \xrightarrow{\Gamma,a(\Psi_1,\Psi_2),\Gamma'} P, \text{ where } \models \Gamma \rightarrow \Psi_1,$$
$$\models (\exists \vec{X} .(\Gamma \wedge \Psi_2))\{\vec{X} / \vec{X'}\} \rightarrow \Gamma'.$$

$$SUM : \frac{P_1 \xrightarrow{\Gamma,\alpha,\Gamma'} P'_1}{P_1 + P_2 \xrightarrow{\Gamma,\alpha,\Gamma'} P'_1}$$

$$REC : \frac{P\{fixX.P/X\} \xrightarrow{\Gamma,\alpha,\Gamma'} P'}{fixX.P \xrightarrow{\Gamma,\alpha,\Gamma'} P'}$$

Table 2: Symbolic operational semantics of hybrid process calculus

An example: Let process $P = \varepsilon(\Phi_1,\Phi_2).Q = \varepsilon(x_1+x_2 \leq 2y+1,\ y' \leq 2x'_2 - x_1).Q$, formula $\Gamma = (t = 0 \wedge x_1 \geq 0 \wedge x_1 + x_2 \leq 3 \wedge y = 3)$. Then we have $P \xrightarrow{\Gamma,(\varepsilon(\Phi_1,\Phi_2),2),\Gamma'} Q$, where $\Gamma' = (t \geq 2 \wedge y \leq 2x_2)$.

B. Symbolic Bisimulation

In this section we define a symbolic version of concrete bisimulation for hybrid process calculus. Symbolic bisimulation is defined as a family of binary relations indexed by a predication logical formula which expresses variable constraints.

Definition 3. A collection of formulas Σ is a partition of Φ if for any θ it holds that $\theta \models \Phi$ implies $\theta \models \Psi$ for some $\Psi \in \Sigma$. A finite partition of Φ is a finite collection of formulas which is a partition of Φ.

Definition 4. A symmetric relation $R \in Pr \times Pr$ with respect to the formula Γ is called a symbolic bisimulation if whenever $P\ R^{\Gamma}\ Q$,

(1) $P \xrightarrow{\Gamma,(\varepsilon(\Phi_1^P,\Phi_2^P),T),\Gamma'_P} P'$ implies that there exists a finite partition $\Sigma = \{\phi_i \mid i \in I\}$, $\models \Gamma \wedge \Phi_1^P \leftrightarrow \vee_{i\in I}\phi_i$, for any ϕ_i there exists a finite partition $\Pi = \{\chi_j \mid j \in J\}$ such that $\models \phi_i \wedge \Phi_2^P \leftrightarrow \vee_{j\in J}\chi_j$, for any χ_j there exists Q' such that $Q \xrightarrow{\Gamma,(\varepsilon(\Phi_1^Q,\Phi_2^Q),T),\Gamma'_Q} Q'$ and $\phi_i \models \Gamma \wedge \Phi_1^Q$, $\chi_j \models \Gamma \wedge \Phi_2^Q$ and $P'\ R^{(\exists \vec{X}.\chi_j \wedge t' = t+T)\{\vec{X}/\vec{X'}\}}\ Q'$;

(2) $P \xrightarrow{\Gamma,a(\Psi_1^P,\Psi_2^P),\Gamma'_P} P'$ with $a \neq \varepsilon$ implies that there exists a finite partition $\Sigma = \{\phi_i \mid i \in I\}$, $\models \Gamma \wedge \Psi_1^P \leftrightarrow \vee_{i\in I}\phi_i$, for any ϕ_i there exists a finite partition $\Pi = \{\chi_j \mid j \in J\}$ such that $\models \phi_i \wedge \Psi_2^P \leftrightarrow \vee_{j\in J}\chi_j$, for any χ_j there exists Q' such that $Q \xrightarrow{\Gamma,.a(\Psi_1^Q,\Psi_2^Q),\Gamma'_Q} Q'$ and $\phi_i \models \Gamma \wedge \Psi_1^Q$, $\chi_j \models \Gamma \wedge \Psi_2^Q$ and $P'\ R^{(\exists \vec{X}.\chi_j)\{\vec{X}/\vec{X'}\}}\ Q'$.

We write $P \sim_S^{\Gamma} Q$ if there is a symbolic bisimulation R such that $P\ R^{\Gamma}\ Q$.

C. Equivalence Between Concrete Bisimulation and Symbolic Bisimulation

In this section, we will prove the equivalence between concrete bisimulation and symbolic bisimulation. Thus to give a complete inference system for concrete bisimulation, it is enough to give a complete inference system for symbolic bisimulation.

To prove Proposition 1 which states the equivalence between concrete bisimulation and symbolic bisimulation, we need some lemmas.

Lemma 1. There exists ρ such that $\rho \cup \rho' \models \Gamma \Leftrightarrow \rho' \models (\exists x_1...\exists x_m.\Gamma)$, where the domain of ρ is $\{x_1,...,x_m\}$, the domain of ρ' is $\{y_1,...,y_n\}$, $\{x_1,...,x_m\} \cap \{y_1,...,y_n\} = \emptyset$.

Proof. See Appendix A. ■

The following lemma gives the corresponding relation between symbolic transition and concrete transition.

Lemma 2. (1) Given Γ and Γ', if for any $\rho \models \Gamma$, there is ρ', such that $\rho' \models \Gamma'$, $\rho \cup (\rho'[x \Leftarrow x']) \models \Gamma \wedge \Phi_2$, and $\langle P,\rho\rangle \xrightarrow{\varepsilon(\Phi_1,\Phi_2),T} \langle P',\rho'\rangle$, then $P \xrightarrow{\Gamma,(\varepsilon(\Phi_1,\Phi_2),T),\Gamma'} P'$, where $\models \forall \delta \in [0,T].\Gamma\{t+\delta/t\} \rightarrow \Phi_1\{t+\delta/t\}$, $\models (\exists \vec{X} .(\Gamma \wedge \Phi_2 \wedge t' = t+T))\{\vec{X} / \vec{X'}\} \rightarrow \Gamma'$;

(2) Given Γ and Γ', if for any $\rho \models \Gamma$, there is ρ', such that $\rho' \models \Gamma'$, $\rho \cup (\rho'[x \Leftarrow x']) \models \Gamma \wedge \Psi_2$, and $\langle P,\rho\rangle \xrightarrow{a(\Psi_1,\Psi_2)} \langle P',\rho'\rangle$, then $P \xrightarrow{\Gamma,a(\Psi_1,\Psi_2),\Gamma'} P'$, where $\models \Gamma \rightarrow \Psi_1$, $\models (\exists \vec{X} .(\Gamma \wedge \Psi_2))\{\vec{X} / \vec{X'}\} \rightarrow \Gamma'$;

(3) $P \xrightarrow{\Gamma,(\varepsilon(\Phi_1,\Phi_2),T),\Gamma'} P'$ implies for any $\rho \models \Gamma$, there is ρ', such that $\rho' \models \Gamma'$ and $\rho \cup (\rho'[x \Leftarrow x']) \models \Gamma \wedge \Phi_2$, $\langle P,\rho\rangle \xrightarrow{\varepsilon(\Phi_1,\Phi_2),T} \langle P',\rho'\rangle$, where $\models \forall \delta \in [0,T].\Gamma\{t+\delta/t\} \rightarrow \Phi_1\{t+\delta/t\}$, $\models (\exists \vec{X} .(\Gamma \wedge \Phi_2 \wedge t' = t+T))\{\vec{X} / \vec{X'}\} \rightarrow \Gamma'$;

(4) $P \xrightarrow{\Gamma, a(\Psi_1,\Psi_2), \Gamma'} P'$ implies for any $\rho \models \Gamma$, there is ρ', such that $\rho' \models \Gamma'$ and $\rho \cup (\rho'[x \Leftarrow x']) \models \Gamma \wedge \Psi_2$, $\langle P, \rho\rangle \xrightarrow{a(\Psi_1,\Psi_2)} \langle P', \rho'\rangle$, where $\models \Gamma \rightarrow \Psi_1$, $\models (\exists \vec{X} .(\Gamma \wedge \Psi_2))\{\vec{X} / \vec{X'}\} \rightarrow \Gamma'$.

Proof. See Appendix B. ■

The following lemma shows the image-finite property of symbolic transition.

Lemma 3. (1) For any P and Γ, there are finitely many $\varepsilon(\Phi_1^P, \Phi_2^P)$, such that $P \xrightarrow{\Gamma, (\varepsilon(\Phi_1^P,\Phi_2^P), T), \Gamma'_P} P'$;

(2) For any P and Γ, there are finitely many $a(\Psi_1^P, \Psi_2^P)$, such that $P \xrightarrow{\Gamma, a(\Psi_1^P,\Psi_2^P), \Gamma'_P} P'$.

Proof. By induction on the inference length of $P \xrightarrow{\Gamma, (\varepsilon(\Phi_1^P,\Phi_2^P), T), \Gamma'_P} P'$ or $P \xrightarrow{\Gamma, a(\Psi_1^P,\Psi_2^P), \Gamma'_P} P'$. ■

In the following, we show that any process is symbolic bisimilar to a "normal process".

Lemma 4. For any P and Γ, there exists a process in the form of $\Sigma_{l\in L}\varepsilon(\Phi_{l1}, \Phi_{l2}).P_l + \Sigma_{m\in M} a_m(\Psi_{m1}, \Psi_{m2}).P_m$ such that $P \sim_S^\Gamma \Sigma_{l\in L}\varepsilon(\Phi_{l1}, \Phi_{l2}).P_l + \Sigma_{m\in M} a_m(\Psi_{m1}, \Psi_{m2}).P_m$.

Proof. By Lemma 3 and by induction on the structure of P. ■

The equivalence between concrete bisimulation and symbolic bisimulation is given in the following proposition.

Proposition 1. For any $\rho \models \Gamma$, $\langle P, \rho\rangle \sim \langle Q, \rho\rangle \Leftrightarrow P \sim_S^\Gamma Q$.

Proof. See Appendix C. ■

Remark: For the symbolic bisimulation, for a transition from P, there should be a finite partition from Q. In the proof of Proposition 1, we show the existence of such finite partition.

IV. A Complete Inference System for Hybrid Process Calculus

In this section, we give an inference system for symbolic bisimulation. The soundness and completeness of this inference system are also studied.

A. An Inference System for Bisimulation of Hybrid Process Calculus

An inference system for symbolic bisimulation consists of the following rules. The rules are in the form of $\frac{A_1,\ ...,\ A_n}{B}$, which means B is true if $A_1,\ ...,\ A_n$ are all true. In these rules, the notation $\Gamma \triangleright P = Q$ means process P is equivalent to process Q if formula Γ is true.

(1) $$\frac{\begin{array}{c}\Gamma \models \Phi_1 \leftrightarrow \Phi_3,\\ \models \exists t.(t' \geq t \wedge \Phi_1 \wedge \Gamma \wedge \Phi_2) \leftrightarrow \exists t.(t' \geq t \wedge \Phi_3 \wedge \\ \Gamma \wedge \Phi_4)\end{array}}{\Gamma \triangleright \varepsilon(\Phi_1, \Phi_2).P = \varepsilon(\Phi_3, \Phi_4).P}$$

(2) $$\frac{\begin{array}{c}\Gamma \models \Phi_1,\\ (\exists \vec{X} .(\Phi_1 \wedge \Gamma \wedge \Phi_2 \wedge t' \geq t)\{\vec{X} / \vec{X'}\} \triangleright P = Q\end{array}}{\Gamma \triangleright \varepsilon(\Phi_1, \Phi_2).P = \varepsilon(\Phi_1, \Phi_2).Q}$$

(3) $$\frac{\Gamma \models \Psi_1 \leftrightarrow \Psi_3,\ \models (\Psi_1 \wedge \Gamma \wedge \Psi_2) \leftrightarrow (\Psi_3 \wedge \Gamma \wedge \Psi_4)}{\Gamma \triangleright a(\Psi_1, \Psi_2).P = a(\Psi_3, \Psi_4).P}$$

(4) $$\frac{\Gamma \models \Psi_1,\ (\exists \vec{X} .(\Psi_1 \wedge \Gamma \wedge \Psi_2))\{\vec{X} / \vec{X'}\} \triangleright P = Q}{\Gamma \triangleright a(\Psi_1, \Psi_2).P = a(\Psi_1, \Psi_2).Q}$$

(5) $$\frac{\Gamma \models \neg\Phi_1}{\Gamma \triangleright \varepsilon(\Phi_1, \Phi_2).P = 0}$$

(6) $$\frac{\Gamma \models \neg\Psi_1}{\Gamma \triangleright a(\Psi_1, \Psi_2).P = 0}$$

(7) $$\frac{\Gamma \models \neg\Phi_2}{\Gamma \triangleright \varepsilon(\Phi_1, \Phi_2).P = 0}$$

(8) $$\frac{\Gamma \models \neg\Psi_2}{\Gamma \triangleright a(\Psi_1, \Psi_2).P = 0}$$

(9) $$\frac{\Gamma \triangleright P = Q}{\Gamma \triangleright P + R = Q + R}$$

(10) $$\frac{}{\Gamma \triangleright fixX.P = P\{fixX.P/X\}}$$

(11) $$\frac{\Gamma \triangleright P = Q\{P/X\}}{\Gamma \triangleright P = fixX.Q}$$

(12) $$\frac{\Gamma \triangleright P = Q}{\Gamma \triangleright fixX.P = fixX.Q}$$

(13) $$\frac{}{\Gamma \triangleright P = P}$$

(14) $$\frac{\Gamma \triangleright P = Q}{\Gamma \triangleright Q = P}$$

(15) $$\frac{\Gamma \triangleright P = Q,\ \Gamma \triangleright Q = R}{\Gamma \triangleright P = R}$$

(16) $$\frac{}{\mathbf{F} \triangleright P = Q}$$ where $\mathbf{F}$ denotes the constant false formula.

(17) $$\frac{\Gamma_1 \triangleright P = Q,\ \Gamma_2 \triangleright P = Q,\ \Gamma \models \Gamma_1 \vee \Gamma_2}{\Gamma \triangleright P = Q}$$

(18) $$\frac{}{\Gamma \triangleright \varepsilon(\Phi_1 \vee \Phi_2, \Phi_3).P = \varepsilon(\Phi_1, \Phi_3).P + \varepsilon(\Phi_2, \Phi_3).P}$$

(19) $$\frac{}{\Gamma \triangleright \varepsilon(\Phi_1, \Phi_2 \vee \Phi_3).P = \varepsilon(\Phi_1, \Phi_2).P + \varepsilon(\Phi_1, \Phi_3).P}$$

(20) $$\frac{}{\Gamma \triangleright a(\Phi_1 \vee \Phi_2, \Phi_3).P = a(\Phi_1, \Phi_3).P + a(\Phi_2, \Phi_3).P}$$

(21) $$\frac{}{\Gamma \triangleright a(\Phi_1, \Phi_2 \vee \Phi_3).P = a(\Phi_1, \Phi_2).P + a(\Phi_1, \Phi_3).P}$$

Remark: A special case of Rule (17) is the Rule CONS: $\frac{\Gamma' \triangleright P = Q,\ \Gamma \models \Gamma'}{\Gamma \triangleright P = R}$

We write $\vdash \Gamma \triangleright P = Q$ to mean that $\Gamma \triangleright P = Q$ can be derived from this proof system.

B. Soundness and Completeness of Inference System

In this section, we study the soundness and completeness of inference system.

We firstly give the soundness of inference system.

Proposition 2. $\vdash \Gamma \triangleright P = Q \Rightarrow P \sim_S^\Gamma Q$.

Proof. By induction on the length of inference. The base case when the length is 0 is straightforward. For the induction step we do case analysis on the last rule applied. ■

Now we turn to completeness. To prove the completeness of inference system, we give the following definitions and lemmas.

Definition 5. A standard equation set

$E : \{X_i = \Sigma_{l \in L} \varepsilon(\Phi_{l1}, \Phi_{l2}).X_l + \Sigma_{m \in M} a_m(\Psi_{m1}, \Psi_{m2}).X_m + \Sigma_{n \in N} W_n \mid i \in I\}$

is an equation set with formal process variables in $\{X_i\}$ and free process variables in $\{W_j \mid j \in J\}$. E is closed if $\{W_j \mid j \in J\} = \emptyset$.

Definition 6. A process P provably $\Gamma-$satisfy an equation set E ($\{X_i = Q_i \mid i \in I\}$) if there exist a vector of processes $\{P_i \mid i \in I\}$ and a condition Γ such that $\vdash \Gamma \triangleright P_1 = P$, and $\vdash \Gamma \triangleright P_i = Q_i\{P_j/X_j\}$ for each $i \in I$. We will simply say "provably satisfies E" when $\Gamma = \mathbf{T}$, where $\mathbf{T}$ denotes the constant true formula.

The following lemma states that any process can be represented as a standard equation set.

Lemma 5. For any process P with free process variables W there exists a standard equation set E, with free process variables in W, which is provably satisfied by P. In particular, if P is closed then E is also closed.

$Proof$. See Appendix D. ■

The following lemma shows that two bisimilar processes can be represented as same standard equation set.

Lemma 6. For closed processes P and Q, if $P \sim_S^\Gamma Q$ then there exist a standard, closed equation set E, which is provably $\Gamma-$satisfied by both P and Q.

$Proof$. See Appendix E. ■

The following lemma states that two processes can be proved to be equivalent if they can be represented as same standard equation set.

Lemma 7. If both P and Q provably $\Gamma-$satisfy an equation set E then $\vdash \Gamma \triangleright P = Q$.

$Proof$. See Appendix F. ■

Now we prove the completeness of inference system.

Proposition 3. For closed processes P and Q, $P \sim_S^\Gamma Q \Rightarrow \vdash \Gamma \triangleright P = Q$.

$Proof$. By Lemma 6, there is a standard equation set E such that which are $\Gamma'-$satisfied by both P and Q for some Γ' such that $\Gamma' \Rightarrow \Gamma$. By Lemma 6, $\vdash \Gamma' \triangleright P = Q$. Finally, by Rule CONS, $\vdash \Gamma' \triangleright P = Q$. ■

The soundness and completeness of inference system is given as follows.

Proposition 4. For closed processes P and Q, $P \sim_S^\Gamma Q \Leftrightarrow \vdash \Gamma \triangleright P = Q$.

$Proof$. By Proposition 2 and Proposition 3. ■

Since concrete bisimulation is equivalent to symbolic bisimulation, the inference system is also sound and complete for concrete bisimulation.

Proposition 5. For any $\rho \models \Gamma$, $\langle P, \rho \rangle \sim \langle Q, \rho \rangle \Leftrightarrow \vdash \Gamma \triangleright P = Q$.

$Proof$. By Proposition 1 and Proposition 4. ■

V. CONCLUSIONS

There are many works on hybrid systems such as [2], [3], [4], [7], and [12]. But as far as we know, there are seldom works on sound and complete inference systems for bisimulation of hybrid systems. However, the sound and complete inference systems for some special kind of hybrid system, such as real timed system, have been proposed. In [11], a timed process calculus where processes denotes timed automata was proposed. Then a complete inference system for such a timed process calculus was presented.

The main aim of this paper is to present a sound and complete inference system for bisimulation of hybrid systems. This paper proposed a hybrid process calculus firstly. Then the concrete bisimulation and symbolic bisimulation for this hybrid process calculus were presented and the equivalence between the two bisimulations were proved. We proposed an inference system for symbolic bisimulation. Furthermore, the soundness and completeness of the inference system were also proved.

ACKNOWLEDGMENT

This work was supported by the Aviation Science Fund of China under Grant No. 20128052064 and the National Natural Science Foundation of China under Grant No. 60873025.

REFERENCES

[1] R. Alur and D. L. Dill. A theory of timed automata. Theoretical Computer Science, 126: 1994, pp. 183-235.

[2] J.A.Bergstra and C.A.Middelburg. Process Algebra for Hybrid Systems, Theoretical Computer Science 335, 2005, pp. 215-280.

[3] D.A. van Beek, K.L. Man, M.A. Reniers, J.E. Rooda, and R. Schiffelers. Syntax and Consistent Equation Semantics of Hybrid Chi, Journal of Logic and Algebraic Programming 68, 2006, pp. 129-210.

[4] P.J. Cuijpers and M.A. Reniers. Hybrid Process Algebra, Journal of Logic and Algebraic Programming 62, 2005, pp. 191-245.

[5] Jan Friso Groote and Alban Ponse. The Syntax and Semantics of μCRL, Report, Stichting Mathematisch Centrum, 1990, 35 pages.

[6] Jifeng H. From CSP to hybrid Systems, in A.W.Roscoe(Ed.), A Classical Mind: Essays in honour of C.A.R. Hoare, Prentice hall, Englewood Cliffs, NJ, 1994, pp. 171-189.

[7] Thomas A. Henzinger. The Theory of Hybrid Automata. In the Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS 96), 1996, pp. 278-292.

[8] M. Hennessy, H. Lin. Symbolic bisimulations. Theoretical Computer Science, 138(2), 1995, pp. 353-389.

[9] Uzma Khadim. A Comparative Study of Process Algebras for Hybrid Systems. Report, Technische Universiteit Eindhoven, 2006. 108 pages.

[10] N. Lynch, R. Segala, and F.W. Vaandrager, Hybrid I/O automata, Information and Computation 185 (1), 2003, pp.105-157.

[11] H. Lin and W. Yi. Axiomatising Timed Automata. In FST&TCS 2000, LNCS 1974, 2000, pp. 277-289.

[12] W. Rounds and H. Song. The ϕ-calculus: a language for distributed con- trol of reconfigurable embedded systems, In the Proceedings of HSCC 2003, LNCS 2623, 2003, pp. 435-449, Springer-Verlag.

Appendix A. Proof of Lemma 1

Lemma 1. There exists ρ such that $\rho \cup \rho' \models \Gamma \Leftrightarrow \rho' \models (\exists x_1 ... \exists x_m.\Gamma)$, where the domain of ρ is $\{x_1, ..., x_m\}$, the domain of ρ' is $\{y_1, ..., y_n\}$, $\{x_1, ..., x_m\} \cap \{y_1, ..., y_n\} = \emptyset$.

$Proof$. $\Rightarrow$: Suppose there exists ρ such that $\rho \cup \rho' \models \Gamma$, where the domain of ρ is $\{x_1, ..., x_m\}$, the domain of ρ' is

$\{y_1, ..., y_n\}$, $\{x_1, ..., x_m\} \cap \{y_1, ..., y_n\} = \emptyset$. It is immediately that $\rho' \models (\exists x_1 ... \exists x_m.\Gamma)$.

$\Leftarrow$: Suppose $\rho' \models (\exists x_1 ... \exists x_m.\Gamma)$, where the domain of ρ' is $\{y_1, ..., y_n\}$, $\{x_1, ..., x_m\} \cap \{y_1, ..., y_n\} = \emptyset$. Then there is ρ such that the domain of ρ is $\{x_1, ..., x_m\}$, $\rho \cup \rho' \models \Gamma$. ■

Appendix B. Proof of Lemma 2

Lemma 2. (1) Given Γ and Γ', if for any $\rho \models \Gamma$, there is ρ', such that $\rho' \models \Gamma'$, $\rho \cup (\rho'[x \Leftarrow x']) \models \Gamma \wedge \Phi_2$, and $\langle P, \rho\rangle \overset{\varepsilon(\Phi_1,\Phi_2),T}{\longrightarrow} \langle P', \rho'\rangle$, then $P \overset{\Gamma,(\varepsilon(\Phi_1,\Phi_2),T),\Gamma'}{\longrightarrow} P'$, where $\models \forall \delta \in [0,T].\Gamma\{t+\delta/t\} \rightarrow \Phi_1\{t+\delta/t\}$, $\models (\exists \vec{X}.(\Gamma \wedge \Phi_2 \wedge t' = t+T))\{\vec{X} / \vec{X'}\} \rightarrow \Gamma'$;

(2) Given Γ and Γ', if for any $\rho \models \Gamma$, there is ρ', such that $\rho' \models \Gamma'$, $\rho \cup (\rho'[x \Leftarrow x']) \models \Gamma \wedge \Psi_2$, and $\langle P, \rho\rangle \overset{a(\Psi_1,\Psi_2)}{\longrightarrow} \langle P', \rho'\rangle$, then $P \overset{\Gamma,a(\Psi_1,\Psi_2),\Gamma'}{\longrightarrow} P'$, where $\models \Gamma \rightarrow \Psi_1$, $\models (\exists \vec{X}.(\Gamma \wedge \Psi_2))\{\vec{X} / \vec{X'}\} \rightarrow \Gamma'$;

(3) $P \overset{\Gamma,(\varepsilon(\Phi_1,\Phi_2),T),\Gamma'}{\longrightarrow} P'$ implies for any $\rho \models \Gamma$, there is ρ', such that $\rho' \models \Gamma'$ and $\rho \cup (\rho'[x \Leftarrow x']) \models \Gamma \wedge \Phi_2$, $\langle P, \rho\rangle \overset{\varepsilon(\Phi_1,\Phi_2),T}{\longrightarrow} \langle P', \rho'\rangle$, where $\models \forall \delta \in [0,T].\Gamma\{t+\delta/t\} \rightarrow \Phi_1\{t+\delta/t\}$, $\models (\exists \vec{X}.(\Gamma \wedge \Phi_2 \wedge t' = t+T))\{\vec{X} / \vec{X'}\} \rightarrow \Gamma'$;

(4) $P \overset{\Gamma,a(\Psi_1,\Psi_2),\Gamma'}{\longrightarrow} P'$ implies for any $\rho \models \Gamma$, there is ρ', such that $\rho' \models \Gamma'$ and $\rho \cup (\rho'[x \Leftarrow x']) \models \Gamma \wedge \Psi_2$, $\langle P, \rho\rangle \overset{a(\Psi_1,\Psi_2)}{\longrightarrow} \langle P', \rho'\rangle$, where $\models \Gamma \rightarrow \Psi_1$, $\models (\exists \vec{X}.(\Gamma \wedge \Psi_2))\{\vec{X} / \vec{X'}\} \rightarrow \Gamma'$.

Proof. By induction on the inference length.

(1) Suppose for any $\rho \models \Gamma$, there is ρ', $\rho' \models \Gamma'$ and $\rho \cup (\rho'[x \Leftarrow x']) \models \Gamma \wedge \Phi_2$, $\langle P, \rho\rangle \overset{\varepsilon(\Phi_1,\Phi_2),T}{\longrightarrow} \langle P', \rho'\rangle$. We only discuss the case $\langle \varepsilon(\Phi_1, \Phi_2).P, \rho\rangle \overset{\varepsilon(\Phi_1,\Phi_2),T}{\longrightarrow} \langle P, \rho'\rangle$. Other cases are similar or trivial.

Suppose for any $\rho \models \Gamma$, there is ρ', $\rho' \models \Gamma'$ and $\rho \cup (\rho'[x \Leftarrow x']) \models \Gamma \wedge \Phi_2$, $\langle \varepsilon(\Phi_1, \Phi_2).P, \rho\rangle \overset{\varepsilon(\Phi_1,\Phi_2),T}{\longrightarrow} \langle P, \rho'\rangle$. We have $\forall \delta \in [0,T].\rho[t \leftarrow \rho(t)+\delta] \models \Phi_1$, $\rho'(t) = \rho(t)+T$, $\rho \cup (\rho'[x \Leftarrow x']) \models \Phi_2$. Therefore $\varepsilon(\Phi_1, \Phi_2).P \overset{\Gamma,(\varepsilon(\Phi_1,\Phi_2),T),\Gamma'}{\longrightarrow} P$, where $\models \forall \delta \in [0,T].\Gamma\{t+\delta/t\} \rightarrow \Phi_1\{t+\delta/t\}$, $\models (\exists \vec{X}.(\Gamma \wedge \Phi_2 \wedge t' = t+T))\{\vec{X} / \vec{X'}\} \rightarrow \Gamma'$.

(2) Suppose for any $\rho \models \Gamma$, there is ρ', $\rho' \models \Gamma'$ and $\rho \cup (\rho'[x \Leftarrow x']) \models \Gamma \wedge \Psi_2$, $\langle P, \rho\rangle \overset{a(\Psi_1,\Psi_2)}{\longrightarrow} \langle P', \rho'\rangle$. Similar to Case (1).

(3) Suppose $P \overset{\Gamma,(\varepsilon(\Phi_1,\Phi_2),T),\Gamma'}{\longrightarrow} P'$. We only discuss the case $\varepsilon(\Phi_1, \Phi_2).P \overset{\Gamma,(\varepsilon(\Phi_1,\Phi_2),T),\Gamma'}{\longrightarrow} P$. Other cases are similar or trivial.

Suppose $\varepsilon(\Phi_1, \Phi_2).P \overset{\Gamma,(\varepsilon(\Phi_1,\Phi_2),T),\Gamma'}{\longrightarrow} P$. We have $\models \forall \delta \in [0,T].\Gamma\{t+\delta/t\} \rightarrow \Phi_1\{t+\delta/t\}$, $\models (\exists \vec{X}.(\Gamma \wedge \Phi_2 \wedge t' = t+T))\{\vec{X} / \vec{X'}\} \rightarrow \Gamma'$. Therefore $\langle \varepsilon(\Phi_1, \Phi_2).P, \rho\rangle \overset{\varepsilon(\Phi_1,\Phi_2),T}{\longrightarrow} \langle P, \rho'\rangle$, where $\forall \delta \in [0,T].\rho[t \leftarrow \rho(t)+\delta] \models \Phi_1$, $\rho'(t) = \rho(t)+T$, $\rho \cup (\rho'[x \Leftarrow x']) \models \Phi_2$.

(4) Suppose $P \overset{\Gamma,a(\Psi_1,\Psi_2),\Gamma'}{\longrightarrow} P'$. Similar to Case (3). ■

Appendix C. Proof of Proposition 1

Proposition 1. For any $\rho \models \Gamma$, $\langle P, \rho\rangle \sim \langle Q, \rho\rangle \Leftrightarrow P \sim_S^\Gamma Q$.

Proof. $\Rightarrow$: Let $R = \{(P, Q) \mid \langle P, \rho\rangle \sim \langle Q, \rho\rangle$ for any $\rho \models \Gamma\}$. It is enough to prove that $R \subseteq \sim_S^\Gamma$.

It holds that $P \sim_S^\Gamma \Sigma_{l\in L}\varepsilon(\Phi_{l1}, \Phi_{l2}).P_l + \Sigma_{m\in M}a_m(\Psi_{m1}, \Psi_{m2}).P_m$ and $Q \sim_S^\Gamma \Sigma_{o\in O}\varepsilon(\Phi_{o1}, \Phi_{o2}).Q_o + \Sigma_{p\in P}a_p(\Psi_{p1}, \Psi_{p2}).Q_p$ by Lemma 3 and Lemma 4.

(1) Since $\langle P, \rho\rangle \sim \langle Q, \rho\rangle$, we have that $\Gamma \models (\vee_{l\in L}\Phi_{l1}) \leftrightarrow (\vee_{o\in O}\Phi_{o1})$, $\Gamma \models (\vee_{l\in L}\Phi_{l2}) \leftrightarrow (\vee_{o\in O}\Phi_{o2})$, $\Gamma \models (\vee_{m\in M}\Psi_{m1}) \leftrightarrow (\vee_{p\in P}\Psi_{p1})$, and $\Gamma \models (\vee_{m\in M}\Psi_{m2}) \leftrightarrow (\vee_{p\in P}\Psi_{p2})$, otherwise P can perform some action that Q can not, and that is a contradiction. Therefore, $\Gamma \models \vee_{l\in L,o\in O}(\Phi_{l1} \wedge \Phi_{o1}) \leftrightarrow \vee_{l\in L}\Phi_{l1} \leftrightarrow \vee_{o\in O}\Phi_{o1}$ and $\{\Phi_{l1} \wedge \Phi_{o1} \mid l \in L, o \in O\}$ is a finite partition of $\vee_{l\in L}\Phi_{l1}$ and $\vee_{o\in O}\Phi_{o1}$. Similarly, there is a finite partition of $(\vee_{l\in L}\Phi_{l2})$ and $(\vee_{o\in O}\Phi_{o2})$, a finite partition of $(\vee_{m\in M}\Psi_{m1})$ and $(\vee_{p\in P}\Psi_{p1})$, and a finite partition of $(\vee_{m\in M}\Psi_{m2})$ and $(\vee_{p\in P}\Psi_{p2})$.

Suppose $\langle P, \rho\rangle \overset{\varepsilon(\Phi_{l1},\Phi_{l2}),T}{\longrightarrow} \langle P', \rho'\rangle$ for any $\rho \models \Gamma$. By Lemma 2, $P \overset{\Gamma,(\varepsilon(\Phi_{l1},\Phi_{l2}),T),\Gamma'_P}{\longrightarrow} P'$, where $\models \forall \delta \in [0,T].\Gamma\{t+\delta/t\} \rightarrow \Phi_{l1}\{t+\delta/t\}$, $\models (\exists \vec{X}.(\Gamma \wedge \Phi_{l2} \wedge t' = t+T))\{\vec{X} / \vec{X'}\} \rightarrow \Gamma'_P$. Since $\langle P, \rho\rangle \sim \langle Q, \rho\rangle$, by Lemma 2 and Lemma 4, we have that there exists a finite partition $\Sigma = \{\Gamma \wedge \Phi_{l1} \wedge \Phi_{o1} \mid l \in L, o \in O\}$, $\models \Gamma \wedge \Phi_{l1} \leftrightarrow \vee_{o\in O}(\Gamma \wedge \Phi_{l1} \wedge \Phi_{o1})$, for any $\Gamma \wedge \Phi_{l1} \wedge \Phi_{o1}$, there exists a finite partition $\Pi = \{\Gamma \wedge \Phi_{l2} \wedge \Phi_{o2}\}$, $\models \Gamma \wedge \Phi_{l1} \wedge \Phi_{o1} \wedge \Phi_{l2} \leftrightarrow \vee_{o\in O}(\Gamma \wedge \Phi_{l2} \wedge \Phi_{o2})$, for any $\Gamma \wedge \Phi_{l2} \wedge \Phi_{o2}$, there exists Q' such that $Q \overset{\Gamma,(\varepsilon(\Phi_{o1},\Phi_{o2}),T),\Gamma'_Q}{\longrightarrow} Q'$ and $\Gamma \wedge \Phi_{l1} \wedge \Phi_{o1} \models \Gamma \wedge \Phi_{o1}$, $\Gamma \wedge \Phi_{l2} \wedge \Phi_{o2} \models \Gamma \wedge \Phi_{o2}$ and $P' \; R^{(\exists \vec{X}.(\Gamma \wedge \Phi_{l2} \wedge \Phi_{o2} \wedge t' = t+T))\{\vec{X} / \vec{X'}\}} \; Q'$.

(2) In the case of $P \overset{\Gamma,a_m(\Psi_{m1},\Psi_{m2}),\Gamma'_P}{\longrightarrow} P'$, proof is similar to Case (1).

$\Leftarrow$: Let $R = \{(\langle P, \rho\rangle, \langle Q, \rho\rangle) \mid P \sim_S^\Gamma Q$ where $\rho \models \Gamma\}$. It is enough to prove that $R \subseteq \sim$.

(1) Suppose $P \sim_S^\Gamma Q$. It holds that $P \sim_S^\Gamma \Sigma_{l\in L}\varepsilon(\Phi_{l1}, \Phi_{l2}).P_l + \Sigma_{m\in M}a_m(\Psi_{m1}, \Psi_{m2}).P_m$ and $Q \sim_S^\Gamma \Sigma_{o\in O}\varepsilon(\Phi_{o1}, \Phi_{o2}).Q_o + \Sigma_{p\in P}a_p(\Psi_{p1}, \Psi_{p2}).Q_p$ by Lemma 3 and Lemma 4. We have that $\Gamma \models (\vee_{l\in L}\Phi_{l1}) \leftrightarrow (\vee_{o\in O}\Phi_{o1})$, $\Gamma \models (\vee_{l\in L}\Phi_{l2}) \leftrightarrow (\vee_{o\in O}\Phi_{o2})$, $\Gamma \models (\vee_{m\in M}\Psi_{m1}) \leftrightarrow (\vee_{p\in P}\Psi_{p1})$, and $\Gamma \models (\vee_{m\in M}\Psi_{m2}) \leftrightarrow (\vee_{p\in P}\Psi_{p2})$, otherwise P can perform some action that Q can not, and that is a contradiction. Therefore, $\Gamma \models \vee_{l\in L,o\in O}(\Phi_{l1} \wedge \Phi_{o1}) \leftrightarrow \vee_{l\in L}\Phi_{l1} \leftrightarrow \vee_{o\in O}\Phi_{o1}$ and $\{\Phi_{l1} \wedge \Phi_{o1} \mid l \in L, o \in O\}$ is a finite partition of $\vee_{l\in L}\Phi_{l1}$ and $\vee_{o\in O}\Phi_{o1}$. Similarly, there is a finite partition of $(\vee_{l\in L}\Phi_{l2})$ and $(\vee_{o\in O}\Phi_{o2})$, a finite partition of $(\vee_{m\in M}\Psi_{m1})$ and $(\vee_{p\in P}\Psi_{p1})$, and a finite partition of $(\vee_{m\in M}\Psi_{m2})$ and $(\vee_{p\in P}\Psi_{p2})$.

Suppose $P \overset{\Gamma,(\varepsilon(\Phi_{l1},\Phi_{l2}),T),\Gamma'_P}{\longrightarrow} P'$. By Lemma 2, for any $\rho \models \Gamma$, there is ρ', $\rho' \models \Gamma'_P$ and $\rho \cup (\rho'[x \Leftarrow x']) \models \Gamma \wedge \Phi_{l2}$, $\langle P, \rho\rangle \overset{\varepsilon(\Phi_{l1},\Phi_{l2}),T}{\longrightarrow} \langle P', \rho'\rangle$, where $\models \forall \delta \in [0,T].\Gamma\{t+\delta/t\} \rightarrow \Phi_{l1}\{t+\delta/t\}$, $\models (\exists \vec{X}.(\Gamma \wedge \Phi_{l2} \wedge t' = t+T))\{\vec{X} / \vec{X'}\} \rightarrow \Gamma'_P$.

Since $P \sim_S^\Gamma Q$, we have that there exists a finite partition $\Sigma = \{\phi_i \mid i \in I\}$, $\models \Gamma \wedge \Phi_{l1} \leftrightarrow \vee_{i\in I}\phi_i$, for any ϕ_i, there exists a finite partition $\Pi = \{\chi_j \mid j \in J\}$, $\models \phi_i \wedge \Phi_{l2} \leftrightarrow \vee_{j\in J}\chi_j$, for any χ_j, there exists Q' such that $Q \stackrel{\Gamma,(\varepsilon(\Phi_{o1},\Phi_{o2}),T),\Gamma'_Q}{\longrightarrow} Q'$ and $\phi_i \models \Gamma \wedge \Phi_{o1}$, $\chi_j \models \Gamma \wedge \Phi_{o2}$ and $P'\ R^{(\exists \vec{X}.\chi_j \wedge t' = t+T)\{\vec{X}/\vec{X'}\}}\ Q'$. Hence $\rho \models \phi_i \models \Gamma \wedge \Phi_{o1}$ and $\rho' \models (\exists\ \vec{X}\ .\chi_j \wedge t' = t+T)\{\vec{X}\ /\ \vec{X'}\} \models (\exists\ \vec{X}\ .\Gamma \wedge \Phi_{o2} \wedge t' = t+T)\{\vec{X}\ /\ \vec{X'}\}$. Therefore by Lemma 2 we have that there exists Q' such that $\langle Q, \rho\rangle \stackrel{\varepsilon(\Phi_{o1},\Phi_{o2}),T}{\longrightarrow} \langle Q', \rho'\rangle$ and $\langle P', \rho'\rangle\ R\ \langle Q', \rho'\rangle$.

(2) In the case of $\langle P, \rho\rangle \stackrel{a_m(\Psi_{m1},\Psi_{m2})}{\longrightarrow} \langle P', \rho'\rangle$, proof is similar to Case (1). ■

Appendix D. Proof of Lemma 5

Lemma 5. For any process P with free process variables W there exists a standard equation set E, with free process variables in W, which is provably satisfied by P. In particular, if P is closed then E is also closed.

Proof. By induction on the structure of P. The only non-trivial case is recursion when $P \equiv fixX.P'$. By induction, there is a standard equation set $E' : \{X_i = U_i \mid i \in I\}$ with free process variables in $FV(P) \cup \{X\}$ and P'_i such that $\vdash P' = P'_1$ and $\vdash P'_i = U_i\{P'_j/X_j \mid j \in I\}$.

We may assume that X is different from any X_i. Let $V_i = U_i\{U_1/X\}$ for each $i \in I$. Note that since X is under an action prefixing in P', it does not occur free in U_1. Hence $V_1 = U_1$. Consider the equation set $E : \{X_i = V_i \mid i \in I\}$. Set $P_i = P'_i\{P/X\}$. Then $\vdash P = fixX.P' = fixX.P'_1 = P'_1\{fixX.P'_1/X\} = P'_1\{P/X\} = P_1$ and $\vdash P = P'_1\{P/X\} = U_1\{P'_i/X_i \mid i \in I\}\{P/X\} = U_1\{P'_i\{P/X\}/X_i \mid i \in I\} = U_1\{P_i/X_i \mid i \in I\}$. Now $\vdash P_i = P'_i\{P/X\} = U_i\{P'_j/X_j \mid j \in I\}\{P/X\} = U_i\{P, P'_j\{P/X\}/X, X_j \mid j \in I\} = U_i\{P, P_j/X, X_j \mid j \in I\} = U_i\{U_1\{P_j/X_j \mid j \in I\}, P'_j\{P/X\}/X, X_j \mid j \in I\} = U_i\{U_1/X\}\{P_j/X_j \mid j \in I\} = V_i\{P_j/X_j \mid j \in I\}$. This shows that P satisfies E. ■

Appendix E. Proof of Lemma 6

Lemma 6. For closed processes P and Q, if $P \sim_S^\Gamma Q$ then there exist a standard, closed equation set E, which is provably $\Gamma-$satisfied by both P and Q.

Proof. Let E_1 and E_2 be the standard equation sets for P and Q, respectively: $E_1 : \{X_i = \Sigma_{l\in L}\varepsilon(\Phi_{l1}, \Phi_{l2}).X_l + \Sigma_{m\in M}a_m(\Psi_{m1}, \Psi_{m2}).X_m + \Sigma_{n\in N}W_n \mid i \in I\}$, $E_2 : \{Y_j = \Sigma_{o\in O}\varepsilon(\Phi_{o1}, \Phi_{o2}).X_o + \Sigma_{p\in P}b_p(\Psi_{p1}, \Psi_{p2}).X_p + \Sigma_{q\in Q}W_q \mid j \in J\}$. So there are P_i, Q_j such that $\vdash P_1 = P$, $\vdash Q_1 = Q$, and $\vdash P_i = \Sigma_{l\in L}\varepsilon(\Phi_{l1}, \Phi_{l2}).P_l + \Sigma_{m\in M}a_m(\Psi_{m1}, \Psi_{m2}).P_m$, $\vdash Q_j = \Sigma_{o\in O}\varepsilon(\Phi_{o1}, \Phi_{o2}).Q_o + \Sigma_{p\in P}b_p(\Psi_{p1}, \Psi_{p2}).Q_p$. Without loss of generality, we may assume $a_m = b_p = a$ for all m, p.

Define $E : \{Z_{ij} = \Sigma_{l\in L,o\in O}\varepsilon(\Phi_{l1} \wedge \Phi_{o1}, \Phi_{l2} \wedge \Phi_{o2}).Z_{lo} + \Sigma_{m\in M,p\in P}a(\Psi_{m1} \wedge \Psi_{p1}, \Psi_{m2} \wedge \Psi_{p2}).Z_{mp} + \Sigma_{n\in N,q\in Q}Z_{nq} \mid i \in I, j \in J\}$.

We claim that E is provably $\Gamma-$satisfied by P when each Z_{ij} is instantiated with P_i.

We need to show, for each i, $\vdash \Gamma \triangleright P_i = \Sigma_{l\in L,o\in O}\varepsilon(\Phi_{l1} \wedge \Phi_{o1}, \Phi_{l2} \wedge \Phi_{o2}).P_l + \Sigma_{m\in M,p\in P}a(\Psi_{m1} \wedge \Psi_{p1}, \Psi_{m2} \wedge \Psi_{p2}).P_m$.

Since $P \sim_S^\Gamma Q$, we have for any l, $\Phi_{l1} \wedge \neg(\vee_{o\in O}\Phi_{o1}) = \mathbf{F}$, $\Phi_{l2} \wedge \neg(\vee_{o\in O}\Phi_{o2}) = \mathbf{F}$, and for any m, $\Psi_{m1} \wedge \neg(\vee_{p\in P}\Psi_{p1}) = \mathbf{F}$, $\Psi_{m2} \wedge \neg(\vee_{p\in P}\Psi_{p2}) = \mathbf{F}$.

Therefore, $\vdash\ \Gamma \triangleright \Sigma_{l\in L,o\in O}\varepsilon(\Phi_{l1} \wedge \Phi_{o1}, \Phi_{l2} \wedge \Phi_{o2}).P_l + \Sigma_{m\in M,p\in P}a(\Psi_{m1} \wedge \Psi_{p1}, \Psi_{m2} \wedge \Psi_{p2}).P_m$

$= \Sigma_{l\in L,o\in O}\varepsilon(\Phi_{l1} \wedge \Phi_{o1}, \Phi_{l2} \wedge \Phi_{o2}).P_l + 0 + 0 + \Sigma_{m\in M,p\in P}a(\Psi_{m1} \wedge \Psi_{p1}, \Psi_{m2} \wedge \Psi_{p2}).P_m + 0 + 0$

$= \Sigma_{l\in L,o\in O}\varepsilon(\Phi_{l1} \wedge \Phi_{o1}, \Phi_{l2} \wedge \Phi_{o2}).P_l + \Sigma_{l\in L,o\in O}\varepsilon(\Phi_{l1} \wedge \neg(\vee_{o\in O}\Phi_{o1}), \Phi_{l2} \wedge \Phi_{o2}).P_l + \Sigma_{l\in L,o\in O}\varepsilon(\Phi_{l1} \wedge \Phi_{o1}, \Phi_{l2} \wedge \neg(\vee_{o\in O}\Phi_{o2})).P_l + \Sigma_{m\in M,p\in P}a(\Psi_{m1} \wedge \Psi_{p1}, \Psi_{m2} \wedge \Psi_{p2}).P_m + \Sigma_{m\in M,p\in P}a(\Psi_{m1} \wedge \neg(\vee_{p\in P}\Psi_{p1}), \Psi_{m2} \wedge \Psi_{p2}).P_m + \Sigma_{m\in M,p\in P}a(\Psi_{m1} \wedge \Psi_{p1}, \Psi_{m2} \wedge \neg(\vee_{p\in P}\Psi_{p2})).P_m$

$= \Sigma_{l\in L}\varepsilon(\Phi_{l1}, \Phi_{l2}).P_l + \Sigma_{m\in M}a(\Psi_{m1}, \Psi_{m2}).P_m = P_i$

Symmetrically we can show that E is provably $\Gamma-$satisfied by Q when each Z_{ij} is instantiated with Q_j. ■

Appendix F. Proof of Lemma 7

Lemma 7. If both P and Q provably $\Gamma-$satisfy an equation set E then $\vdash \Gamma \triangleright P = Q$.

Proof. By induction on the size of E. For the base case when E contains only one equation $X_1 = V_1$, we have $\vdash \Gamma \triangleright P = V_1\{P/X_1\}$. Therefore $\vdash \Gamma \triangleright P = fixX_1.V_1$. Similarly, $\vdash \Gamma \triangleright Q = fixX_1.V_1$. Hence $\vdash \Gamma \triangleright P = Q$.

Assume the result for m and let E contain $m+1$ equations: $X_i = V_i$, $1 \le i \le m+1$. Since P provably $\Gamma-$satisfies E, there are P_i, $1 \le i \le m+1$, such that $\vdash \Gamma \triangleright P_1 = P$, and $\vdash \Gamma \triangleright P_i = V_i\{P_j/X_j\}$ for each $1 \le i, j \le m+1$. In particular, $\vdash \Gamma \triangleright P_{m+1} = V_{m+1}\{P_i/X_i \mid 1 \le i \le m+1\} = (V_{m+1}\{P_i/X_i \mid 1 \le j \le m\})\{P_{m+1}/X_{m+1}\}$. By Rule (11), $\vdash \Gamma \triangleright P_{m+1} = fixX_{m+1}.V_{m+1}\{P_i/X_i \mid 1 \le i \le m\}$. Writing W_{m+1} for $fixX_{m+1}.V_{m+1}$, we have $\vdash\ \Gamma \triangleright P_{m+1} = W_{m+1}\{P_i/X_i \mid 1 \le i \le m\}$. Therefore, $\vdash\ \Gamma \triangleright P_i = V_i\{P_j/X_j \mid 1 \le j \le m+1\} = V_i\{P_j/X_j \mid 1 \le j \le m\}\{P_{m+1}/X_{m+1}\} = V_i\{P_j/X_j \mid 1 \le j \le m\}\{W_{m+1}\{P_i/X_i \mid 1 \le i \le m\}/X_{m+1}\} = V_i\{W_{m+1}/X_{m+1}\}\{P_j/X_j \mid 1 \le j \le m\}$. This shows P provably $\Gamma-$satisfies the equation set $E' : X_i = V_i\{W_{m+1}/X_{m+1}\}$ for each $1 \le i \le m$. Symmetrically we can show that Q provably $\Gamma-$satisfies the equation set E'. By induction we conclude $\vdash \Gamma \triangleright P = Q$. ■

Static and Dynamic Analysis for Robustness under Slowdown

Ingram Bondin
Department of Computer Science
University of Malta
Msida, Malta
email: ingrambondin@gmail.com

Gordon J. Pace
Deparment of Computer Science
University of Malta
Msida, Malta
email: gordon.pace@um.edu.mt

***Abstract*—Robustness of embedded systems to potential changes in their environment, which may result in the inputs being affected, is crucial for reliable behaviour. One typical possible change is that the system's inputs are slowed down, altering its temporal behaviour. Algorithmic analysis of systems to be able to deduce their robustness under such environmental interference is desirable. In this paper, we present a framework for the analysis of synchronous systems to analyse their behaviour when the inputs slow down through stuttering. We identify different types of slowdown robustness constraints and present static and dynamic analysis techniques for determining whether systems written in Lustre satisfy these robustness properties.**

***Keywords*-Synchronous Languages; Lustre; Slowdown**

I. Introduction

Software is increasingly becoming more prominent as a controller for a variety of devices and processes. Embedded systems operate within an environment, by which they are affected and with which they interact — this tight interaction usually means that changes to the environment directly affect the behaviour of the embedded system. One such situation can occur when the environment slows down its provision of input to the system, possibly resulting from a variety of reasons. For example, the system producing the inputs or the communications channel on which these inputs pass to the program might be under heavy load, delaying the inputs; or the program is deployed on a faster platform, therefore making the input relatively slower.

One question which arises immediately in such scenarios is how the system behaves when its input slows down. Does it act in an expected manner, or does the slow input cause it to produce unwanted output? In this paper, we develop an approach to study whether a system continues to behave correctly under these conditions. We characterise different notions of correctness since, for instance, in some cases we may desire the output to be delayed by the same amount as the inputs, whereas in others, the values but not the actual delays on the outputs are important.

The theory we develop is applied to the synchronous language Lustre [1], which enables the static deduction of a program's resource requirements, making it ideal for the design of embedded systems. Although retiming analysis techniques for continuous time can be found in the literature [2], our approach adapts them for discrete time, the timing model used by Lustre and other synchronous languages.

Such a theory requires addressing a number of considerations. In section II we define streams [3], which are infinite sequences of values, as well as the Lustre programs which manipulate them. In the model we adopt, streams can be slowed down through the repetition of values, which is also called stuttering. Stuttering can be a valid model for slow input under several scenarios:

- If a memory's clock signal becomes slower, the memory will take more time to read new input, and thus will maintain its present output for a longer time. A program which samples the values of this memory at the same rate will then experience repetition in its input.
- The system providing the input might not be ready to provide its output, or it might experience a fault from which it needs time to recover. In these situations, some systems might keep their present output constant until they are ready once again. In this case, the receiving program will also experience repetition in its input.
- A physical process which is being sampled in order to provide input to a program might slow down. Under certain sampling conditions, the resulting input received by the program corresponds to experiencing stutter in its inputs.

Providing stuttered input to a program will cause it to react in a particular manner. A program can be said to be robust with respect to slow input if it behaves in a way which is acceptable to the scenario under consideration. Section III provides a number of robustness properties which characterise what may be acceptable in different scenarios. Given such a property, one needs some algorithmic means of checking whether it holds or not for a given program. Section IV considers a method based on the static analysis of the program's text, while section V focuses on a method based on the dynamic analysis of its state space. This is followed by a case study in section VI, in which these two approaches are applied to a number of Boolean Lustre programs. Section VII presents work related to the theory which has been developed, while section VIII provides some concluding remarks.

 ISBN: 978-1-61208-277-6

II. STREAMS, SLOWDOWN AND LUSTRE PROGRAMS

We adopt the standard view of a stream as an infinite sequence of values over a particular type, representing the value of the stream over a discrete time domain. We will write $s(t)$ to denote the value taken by stream s at time t, lifting the notation for vectors of streams. For instance, for two streams s and s', $\langle s, s' \rangle(t)$ is defined to be the tuple of values $\langle s(t), s'(t) \rangle$.

By slowing down a stream, one obtains the same sequence of values, but possibly with some of the values repeated a number of times, representing stutter. A slowdown can be characterised using a latency function — a total function which returns the number of times each value in the stream will stutter for. Given a stream s, which is slowed down according to a latency function λ, one obtains the slowed down stream s_λ:

$$s_\lambda = \underbrace{s(0), \ldots s(0)}_{\lambda(0)+1}, \underbrace{s(1), \ldots s(1)}_{\lambda(1)+1}, \ldots \underbrace{s(n), \ldots s(n)}_{\lambda(n)+1} \ldots$$

Note that s_λ is obtained from s by replacing the value of s at time t by a block of of $\lambda(t)+1$ copies of this value. We will write Start_t^λ to denote the time instant at which the t^{th} such block begins: $\sum_{i=0}^{t-1} \lambda(i)$. Similarly, End_t^λ denotes the time instant at which the block ends and is analogously defined.

Note that the constant zero latency function leaves the stream untouched. If a latency function is a constant function, we shall refer to it as *uniform*.

As before, we will overload this notation for vectors of streams, with $\langle s, s' \rangle_\lambda$ being equivalent to $\langle s_\lambda, s'_\lambda \rangle$.

Lustre [1] provides a way of symbolically specifying systems which process streams in a declarative manner. A Lustre program $P = \langle V, I, O, E \rangle$ is defined over a set of stream variables V, with two disjoint subsets I and O consisting of the input and output stream variables of the program respectively, and a set of equations E which explains how to compute the value of each output variable at every instant of time in terms of other program variables. Equations can take one of the following forms:

$$\begin{array}{lcl} y & = & \otimes(x_1, \ldots x_n) \\ y & = & \mathit{pre}\ x_1 \\ y & = & x_1\ \texttt{->}\ x_2 \\ y & = & x_1\ \mathit{fby}\ x_2 \end{array}$$

Instantaneous operators $\otimes$ are used to represent computation performed at each time instant. For instance, the equation $y = \wedge(x_1, x_2)$ would update the value of stream variable y with the value of the conjunction of the stream variables x_1 and x_2 at each time instant: $y(t) = x_1(t) \wedge x_2(t)$. The *delay* operator *pre* allows access to the previous value of a given stream variable: $(\mathit{pre}\ x)(t{+}1) = x(t)$ with the resulting stream being undefined for the initial time point, at which it is said to take the value *Nil*. In fact, *pre* behaves like an uninitialised memory. The *initialisation* operator $x_1\ \texttt{->}\ x_2$ yields a stream behaving like x_1 at the first time instant, and like x_2 elsewhere: $(x_1\ \texttt{->}\ x_2)(0) = x_1(0)$ and $(x_1\ \texttt{->}\ x_2)(t{+}1) = x_2(t{+}1)$. These last two operators are frequently combined to produce an initialised memory using the *followed-by* operator, with $x_1\ \mathit{fby}\ x_2$ being equivalent to $x_1\ \texttt{->}\ \mathit{pre}\ x_2$.

Below we illustrate two sample programs. The program TOGGLE represents a toggle switch which starts in the Boolean state *true*, and which outputs its present state if its toggle input is *false* and inverts and outputs its present state if the toggle input is *true*. On the other hand, the program SISO is a 4-bit serial in serial out register, which starts with all its memories set to true.

```
node TOGGLE(toggle : bool)
returns(out : bool);
var X, Y : bool;
let
   out = if toggle then x else y;
   x = not y;
   y = true fby out
tel;
```

```
node SISO(i1 : bool)
returns(i5 : bool);
var i2, i3, i4 : bool;
let
   i2 = true fby i1;
   i3 = true fby i2;
   i4 = true fby i3;
   i5 = true fby i4;
tel;
```

We will use the notation P_{inst}, P_{delay}, P_{init}, and P_{fby} for the primitive programs with just one equation consisting of a single application of an instantaneous, delay, initialisation or followed-by operator respectively. For each primitive program, the variable occurring on the left hand side of its equation is an output variable, those appearing on the right are inputs.

For a Lustre program P, $\mathrm{dep}_0(P) \subseteq V \times V$ relates a stream variable y to a stream variable x if y is defined in P by an equation with x appearing on the right hand side. The irreflexive transitive closure of this relation denotes the dependencies between the stream variables and is written as $\mathrm{dep}(P)$. Another important concept is that of an instantaneous dependency relation. This relation can be obtained by starting from the relation $\mathrm{inst}_0(P) \subseteq V \times V$, which relates a stream variable y to a stream variable x only if y's defining equation involves x, and x does not appear in a pre equation or on the right hand side of an fby equation. The irreflexive transitive closure of this relation, $\mathrm{inst}(P)$ denotes the instantaneous dependencies between stream variables. A Lustre program P is said to be well-formed if none of its variables instantaneously depend on themselves: $\forall s \cdot (s, s) \notin \mathrm{inst}(P)$.

Given two Lustre programs P_1 and P_2 (with inputs I_1, I_2 and outputs O_1, O_2 respectively) their composition, written $P_1 \mid P_2$, is the Lustre program whose equation set is the union of the equation sets of the respective programs. Its inputs are the inputs of either program not appearing as outputs of the other ($I = (I_1 \cup I_2) \setminus (O_1 \cup O_2)$), and vice versa for its outputs ($O = (O_1 \cup O_2) \setminus (I_1 \cup I_2)$). In particular, certain specific types of composition shall be referred to as follows:

- *Disjoint composition*, if $O_2 \cap I_1 = O_1 \cap I_2 = \emptyset$.
- *Composition without feedback*, if $O_2 \cap I_1 = \emptyset$ or $O_1 \cap I_2 = \emptyset$.
- *Fully connected composition*, if $O_2 \cap I_1 = \emptyset$ and $O_1 = I_2$, or conversely $O_1 \cap I_2 = \emptyset$ and $O_2 = I_1$.

Another important operation is that of *adding a feedback loop* to a program P by connecting an output y to an

 ISBN: 978-1-61208-277-6

input x written $P[y \rightarrow x]$, provided y does not depend on x, that is $(y, x) \notin dep(P)$. Given the Lustre program $P' = \langle\{x, y\}, \{y\}, \{x\}, \{x = y\}\rangle$, adding a feedback loop can also be defined in terms of composition as follows:

$$P[y \rightarrow x] \stackrel{\text{df}}{=} P \mid P'$$

Assuming the existence of an ordering on the program's variables, given a Lustre program P, and a vector i which assigns a stream to each of the program's input variables, $P(i)$ denotes the vector o of output streams corresponding to the output variables of P as computed by the semantics of Lustre [1].

Our goal is therefore that of identifying Lustre programs P such that upon slowing down their inputs i according to a latency function λ, will result in P still being well behaved. In the next section we will identify different forms of such well-behaviour of $P(i_\lambda)$ with respect to the unslowed behaviour $P(i)$.

Boolean Lustre programs can also be compiled into automata spanning over the state space they cover [4]. This can be defined for Lustre programs using fby (instead of delays) as follows:

Definition 1: (Lustre Automaton). Let P be a Boolean Lustre program with n input variables, m output variables, and k fby equations of the form $y = x_1$ *fby* x_2. Then, this program can be compiled into an automaton $A = \langle S, s_{init}, \tau, \delta\rangle$, where S is its set of states, s_{init} is its initial state, $\tau : \mathbb{B}^n \times S \rightarrow S$ is its transition function and $\delta : \mathbb{B}^n \times S \rightarrow \mathbb{B}^m$ is its output function. The automaton processes the input vector provided to the program one tuple at a time. During each instant, it uses its current input tuple and its present state to (i) move to a new state under the guidance of its transition function τ and (ii) output an output tuple as defined by its output function δ, which represents the values of the program's output variables at that particular time instant. The program P can be converted into automaton A using the following procedure.

External Initialisation: A program is said to be initialised externally if in at least one of its fby statements x_1 *fby* x_2, the initial variable x_1 depends on one of the program's input variables.

States: Each fby statement x_1 *fby* x_2 corresponds to a memory element in the program, whose value is determined by the variable x_1 at the first instant and by the variable x_2 at all further instants. Since each such memory can either be true or false, we create 2^k states, with each state representing one possible configuration of the program's memories. If the program is initialised externally, we also add a special initial state $init$ to the set of states.

Initial State: If the program is initialised externally, the initial state is $init$. Otherwise, the initial state is the state corresponding to the configuration obtained by evaluating the variables of the form x_1 within the program's fby statements.

Transition Function: With n input variables, there are 2^n possible input tuples. Each state therefore has 2^n transitions, with each transition labeled with the associated input tuple.

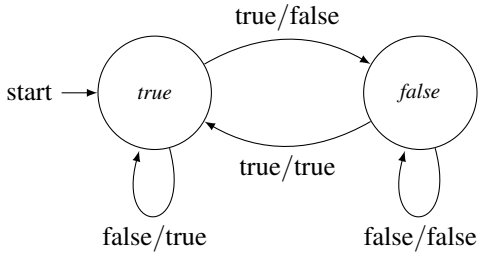

Fig. 1. Automaton obtained from toggle switch program

Given a state $s \neq init$ and input tuple a, the next state $\tau(a, s)$ is computed as follows: (i) assign the configuration represented by present state s to the respective variables of the form x_2 occurring on the right hand side of *fby* statements, (ii) assign the input values represented by tuple a to the respective input variables and (iii) simulate the Lustre program, using the defining equations of the variables of the form x_2 to determine the configuration of the memories at the next time instant, allowing the selection of the appropriate next state. The initial state $init$, if present, also has 2^n transitions. The next states are determined as follows (i) assign the input values represented by tuple a to the respective input variables, (ii) use the defining equations of variables of the form x_1 to compute the value of the initialisation variables and (iii) simulate the Lustre program using the defining equations of the variables of the form x_2 which determine the next state. Again, these values determine the configuration of the memories at the next time instant and allow the selection of the appropriate next state.

Output Function: Each transition is associated with an m-tuple, which represents the values of the output variables when the automaton finds itself in a certain state and processes a certain input tuple. The procedure for obtaining the output tuple is similar to that for obtaining the next state, except that the output tuple is constructed by simulating the program and considering the values of the output variables.

Fig. 1 shows the automaton which would be obtained by applying the above procedure to the toggle switch program TOGGLE. The two states represent the two possible configurations which the memory corresponding to the program's only *fby* equation can be in. Meanwhile, for each transition, the value on the left shows the value of the toggle input variable which causes the transition, and the value on the right shows the output value computed by the program. We shall return to this representation of the TOGGLE program at a later stage.

We now consider a number of different forms of program robustness to slow input.

III. SLOWDOWN ROBUSTNESS

Whether a program behaves in an acceptable way depends on the scenario it is operating in. In this section, the four well behaviour properties of *stretch robustness*, *stutter robustness*, *fast-enough robutness* and *immediate-at-first robustness* are introduced, characterising desirable behaviour under different circumstances.

 ISBN: 978-1-61208-277-6

A. Stretch Robustness

Stretch robustness (STR) specifies the fact that if the input of a program slows down by some amount, then the output of a program should slow down by the same amount. This property can be formalised by requiring that whenever a latency function λ is applied to a program's input, the program will respond by applying the same latency to its output.

Definition 2: (Stretch Robustness). A program P is said to be *stretch robust with respect to a latency function* λ, if for any input vector i: $P(i_\lambda) = P(i)_\lambda$. P is simply said to be *stretch robust* if it is stretch robust with respect to all latency functions.

The table below shows the relationship between a slow input vector i_λ and the required program output $P(i_\lambda)$:

i_λ	$\underbrace{i(0),\ldots i(0)}_{\lambda(0)+1}$,	$\underbrace{i(1),\ldots i(1)}_{\lambda(1)+1}$,	$\ldots$	$\underbrace{i(n),\ldots i(n)}_{\lambda(n)+1}$	$\ldots$
$P(i_\lambda)$	$\underbrace{o(0),\ldots o(0)}_{\lambda(0)+1}$,	$\underbrace{o(1),\ldots o(1)}_{\lambda(1)+1}$,	$\ldots$	$\underbrace{o(n),\ldots o(n)}_{\lambda(n)+1}$	$\ldots$

One immediate consequence of this property is that additional repetition of the program's input does not cause the program to change its output. Stretch robustness is thus useful in situations where one requires the program not to change its output when faced with additional latency. Stretch robustness is a very strong property, which can be relaxed in a number of ways to obtain weaker criteria which may be sufficient in certain circumstances. We shall now consider these criteria.

B. Stutter Robustness

Stutter robustness (STU) requires that if the input of a program slows down by some amount, the output of the program should also slow down, but possibly at a different rate. This will be modeled by requiring that whenever a latency function λ is applied to a program's input, the program will respond by applying *some* latency function λ' to its output. Unlike stretch robustness, λ and λ' need not be equal:

Definition 3: (Stutter Robustness). A program P is *stutter robust with respect to a latency function* λ if there exists a latency function λ' such that for every input vector i: $P(i_\lambda) = P(i)_{\lambda'}$. P is said to be *stutter robust* if it is stutter robust with respect to any latency function.

The relationship between a slow vector of inputs i_λ and the required program output $P(i_\lambda)$ is shown below:

i_λ	$\underbrace{i(0),\ldots i(0)}_{\lambda(0)+1}$,	$\underbrace{i(1),\ldots i(1)}_{\lambda(1)+1}$,	$\ldots$	$\underbrace{i(n),\ldots i(n)}_{\lambda(n)+1}$	$\ldots$
$P(i_\lambda)$	$\underbrace{o(0),\ldots o(0)}_{\lambda'(0)+1}$,	$\underbrace{o(1),\ldots o(1)}_{\lambda'(1)+1}$,	$\ldots$	$\underbrace{o(n),\ldots o(n)}_{\lambda'(n)+1}$	$\ldots$

Thus for a stutter robust program, the output under slow input can be obtained from the original output by adding *any* number of repetitions to the values appearing in the original output, without adding any other artifacts nor removing any values. This means that stutter robustness is useful as a well behaviour property in situations where one needs to ensure that the output under slow input has the same structure as the original output, but one is able to tolerate additional repetition in the slow output.

C. Fast-Enough and Immediate-at-First Robustness

In stretch robustness, the value of the outputs remains equal to the original value in the unslowed system. In fast-enough robustness (FE) this constraint is relaxed by requiring only that the program converge to the original output before the slowed down input ends. Formally, we shall say that a program is fast-enough robust if, when we apply a latency function λ to the program's input, the slow output has the property that its value at the end of each block of repetitions (at points of the form End_t^λ) is equal to the value taken by the original output at the points t (i.e. those points which were expanded into the blocks of repetitions).

Definition 4: (Fast-Enough Robustness). A program P is *fast-enough robust with respect to a latency function* λ if for any input vector i:

$$\forall t : \mathbb{T} \cdot P(i_\lambda)(End_t^\lambda) = P(i)(t)$$

Program P is said to be fast-enough robust if it is fast-enough robust with respect to any latency function.

Fast-enough robustness is primarily of interest for particular latency functions, since general fast-enough robustness can be proved to be equivalent to general stretch robustness.

Fast-enough robustness can be visualised as follows (using ? to indicate don't-care values):

i_λ	$\underbrace{i(0),\ldots i(0)}_{\lambda(0)+1}$,	$\underbrace{i(1),\ldots i(1)}_{\lambda(1)+1}$,	$\ldots$	$\underbrace{i(n),\ldots i(n)}_{\lambda(n)+1}$	$\ldots$
$P(i_\lambda)$	$\underbrace{?,\ldots ?,o(0)}_{\lambda(0)+1}$,	$\underbrace{?,\ldots ?,o(1)}_{\lambda(1)+1}$,	$\ldots$	$\underbrace{?,\ldots ?,o(n)}_{\lambda(n)+1}$	$\ldots$

This well behaviour property is useful in scenarios in which one can tolerate the fact that additional latency on the input might produce undesirable intermediate results as long as the original value is produced by the end of the latency period.

The dual of fast-enough robustness is *immediate-at-first robustness* (IAF) — instead of constraining the slow input to converge to the original value before a block of repetitions ends, it requires it to produce the original value as soon as a block of repetitions starts, leaving it free to assume any value until that block of repetition ends.

Definition 5: (Immediate-At-First Robustness). A program P is said to be *immediate-at-first robust with respect to latency function* λ if for any input vector i:

$$\forall t : \mathbb{T} \cdot P(i_\lambda)(Start_t^\lambda) = P(i)(t)$$

P is said to be immediate-at-first robust if it satisfies the above constraint with respect to any latency function.

Immediate-at-first robustness can be visualised as follows:

 ISBN: 978-1-61208-277-6

i_λ	$\underbrace{i(0),\ldots i(0)}_{\lambda(0)+1}$,	$\underbrace{i(1),\ldots i(1)}_{\lambda(1)+1}$,	$\ldots$	$\underbrace{i(n),\ldots i(n)}_{\lambda(n)+1}$	$\ldots$
$P(i_\lambda)$	$\underbrace{o(0),?,\ldots?}_{\lambda(0)+1}$,	$\underbrace{o(1),?,\ldots?}_{\lambda(1)+1}$,	$\ldots$	$\underbrace{o(n),?,\ldots?}_{\lambda(n)+1}$	$\ldots$

This well behaviour property is useful in scenarios in which one requires the program to react immediately as soon as the latency on a previous input value wears off, but in which further repetition of the input can be safely ignored by outputing any result.

We shall now consider algorithmic means to check Lustre programs for robustness.

IV. Detecting Robustness: Static Analysis

The first approach to checking whether a program satisfies a well behaviour property is based on a static analysis of the structure of the Lustre program. The analysis is based on two main theorems: (i) Theorem 1 which identifies which primitive programs satisfy which well behaviour properties and; (ii) Theorem 2 which identifies which well behaviour properties are preserved upon composition of two well behaved programs.

Theorem 1: Primitive Lustre programs all come with a level of guaranteed robustness: (i) instantaneous programs are robust under all four forms; (ii) delay and followed-by programs are robust under stutter and immediate-at-first robustness; and (iii) primitive initialisation programs are immediate-at-first robust.

Proof: (i) Instantaneous programs apply a pointwise operator to their input streams to obtain their output streams. Thus, the same input tuple always causes the same output tuple. Repetition of inputs through latency will therefore cause repetition of outputs, which makes the program stretch robust. (ii) The output of delay and fby programs has an additional initial value with respect to the input stream. Slowing the input stream down by a latency function, causes the program to attach this value to the slow stream. The output under slow input can therefore be obtained from the original input through a latency function, which does not repeat the attached element, and which repeats all subsequent elements accordingly. These programs are therefore stutter robust. The programs are also immediate-at-first robust as can be inferred from the depiction below, which shows how the values of the original output (first row) are associated to the corresponding blocks of the output under latency (second row). It is clear that the value at the beginning of each block is equal to the corresponding value in the original output.

$P(i)$	Nil	$x_1(0)$	$x_1(1)$
$P(i_\lambda)$	$Nil, \underbrace{x_1(0)\ldots x_1(0)}_{\lambda(0)}$	$x_1(0), \underbrace{x_1(1)\ldots x_1(1)}_{\lambda(1)}$	$x_1(1), \underbrace{x_1(2)\ldots x_1(2)}_{\lambda(2)}$

(iii) Initialisation programs take the first value of stream x_1, and attach to it the stream x_2 from its second value onwards. Below one can see how the blocks of output under slow input relate to the original output; this illustrates the fact that the value at the beginning of each block is equal to the corresponding value in the original output.

$P(i)$	$x_1(0)$	$x_2(1)$	$x_2(2)$
$P(i_\lambda)$	$x_1(0), \underbrace{x_2(0)\ldots x_2(0)}_{\lambda(0)}$	$\underbrace{x_2(1)\ldots x_2(1)}_{\lambda(1)+1}$	$\underbrace{x_2(2)\ldots x_2(2)}_{\lambda(2)+1}$

□

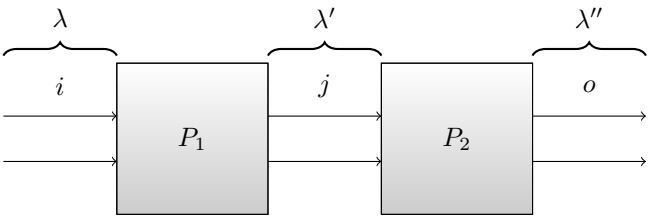

Fig. 2. Fully connected composition preserves STU

We can now consider the effect of composing robust programs.

Theorem 2: Some forms of composition of robust programs guarantee robustness of the resulting program: (i) the composition without feedback of two stretch robust programs is always stretch robust, so is adding a feedback loop to a stretch robust program; (ii) the fully connected composition of two stutter robust Lustre programs is always stutter robust; and (iii) the disjoint composition of two immediate-at-first robust programs is always immediate-at-first robust.

Proof: We provide a proof of (ii) to illustrate the proof idea. We consider two arbitrary stutter robust programs P_1 and P_2, and show that their fully connected composition is also stutter robust. Since P_1 and P_2 are being composed in a fully connected way, every output of P_1 is connected to an input of P_2, and there are no feedback connections. This is shown in Fig. 2.

Now suppose that if we pass a vector i to P_1, the program responds by outputting vector j. Also suppose that when P_2 receives vector j it outputs vector o in response. We need to show that if a latency function λ is applied to the composite program's input vector i, the composite program applies some latency function to its output vector o. Since P_1 is stutter robust, applying λ to the input vector i will make P_1 apply some latency function λ' to its output j. Hence, P_2 receives the vector j slowed down by λ' as input. Since P_2 is also stutter robust it will apply some other latency function λ'' to its output o. Thus, applying a latency function to the input of the composed program, causes the composed program to slow its output by some latency function, proving that stutter robustness is preserved by fully connected composition. □

We now consider a method which analyses the behaviour of the particular program under examination, rather than its structure.

V. Detecting Robustness: Dynamic Analysis

Theorem 2 allows us to conclude robustness of composed programs in a syntactically compositional manner. In this

section, we give richer, although more expensive, semantic analysis techniques for Lustre programs allowing for dynamic robustness analysis of their behaviour. Through the use of symbolic methods, such as with Binary Decision Diagrams (BDDs), the analysis can be applied either on whole programs or to subprograms. In the latter case, Theorem 2 can then be used to obtain results about the composition of the subprograms.

The techniques we shall discuss rely on identifying conditions on the Lustre automaton which are sufficient to guarantee that certain well behaviour properties are satisfied by that program. Two types of conditions are defined: (i) latency independent conditions, which check whether a robustness property holds in general, and (ii) latency dependent conditions, which check whether a property holds when some particular latency function is applied to the program's input.

The conditions identified can be checked using either an exhaustive analysis of the automaton's state space, or preferably using a symbolic representation of the automaton such as BDDs to ensure that the approach scales up to larger systems.

A. Latency Independent Conditions

We start by identifying properties which guarantee slow-down robustness for any latency function. The strongest condition, is that stateless[1] programs are always stretch robust.

Theorem 3: (Condition 1 — Stretch Robustness). If $\forall a, s, s' \cdot \delta(a, s) = \delta(a, s')$, then the program is stretch robust.

Proof: Under such a condition, a particular input tuple always generates the same output tuple, independently of the state the automaton finds itself in. Thus, any repetition of an input tuple caused by a latency function causes a repetition of the corresponding output tuple. This is sufficient to ensure stretch robustness. □

Under stutter robustness, slowing a program's input by a latency function λ, causes the program to slow its output by a latency function λ'. In practice, this means that the output under slow input can be obtained through the repetition of the original output tuples only. We now show that if the automaton has a certain feature, then this property cannot hold.

Theorem 4: (Condition 2 — Failure Of Stutter Robustness). Programs satisfying the following condition are not stutter robust:

$$\begin{aligned}\exists a, b, s, s', j, k, l \cdot \\ \delta(a, s) = j \wedge \tau(a, s) = s' \wedge \\ \delta(a, s') = k \wedge k \neq j \wedge \\ \delta(b, s') = l \wedge b \neq a \wedge l \neq k\end{aligned}$$

Proof: Condition 2 looks for the presence of reachable states s and s' having the following properties: (i) under input tuple a, state s outputs tuple j and passes to state s'; (ii) under input tuple a, state s' outputs $k \neq j$ and (iii) under input tuple $b \neq a$, state s' outputs tuple $l \neq k$.

We now show that if this structure is present in the automaton, there will always be some input vector and some latency function which breaks the stutter robustness property. We first construct the input vector as follows. Choose a path from the start state s_{init} to the state s. By following this path of n transitions, we obtain the first n tuples of the input vector. We also obtain the first n tuples of the output vector. To this initial segment of the input vector, one appends the input tuples a, b, which causes the resulting output vector to be augmented by the output tuples j, l. The rest of the input vector can be chosen arbitrarily.

We now choose a latency function, which when applied to the input vector above, breaks the property. The chosen latency function will insert 1 repetition for the input tuple at time instant $n+1$, and 0 repetitions elsewhere. Applying this latency function to the input vector chosen earlier yields the original initial segment followed by the tuples a, a, b. Through the presence of the regularity identified in the theorem, the resulting output will be the initial segment of the output vector followed by the output tuples j, k, which means that with respect to the original output an l tuple has been deleted. This makes it impossible to derive the output under slow input from the original output through the addition of repetitions only. □

Finally, we can also identify a sufficient condition for immediate-at-first robustness. If the automaton obtained from the program always loops with repetitions after the first occurrence of an input, then the program is guaranteed to be immediate-at-first robust.

Theorem 5: (Condition 3 — Immediate-At-First Robustness). If $\forall a, s, s' \cdot (\tau(a, s) = s') \implies (\tau(a, s') = s')$, then the program is immediate-at-first robust.

Proof: When processing an input vector i, the automaton uses the current input $i(t)$ and state $s(t)$, to compute the output $o(t)$ and next state $s(t+1)$. When input i has latency λ, the program receives consecutive blocks of constant inputs, with the n^{th} block consisting of tuples of the form $i(n)$. For the program to be immediate-at-first robust, the output at the beginning of the n^{th} block must have the form $o(n)$.

We observe that if the automaton finds itself in state $s(n)$ at the beginning of block n, the condition guarantees that (i) at the first time instant in the block the automaton moves to state $s(n+1)$; (ii) it stays in state $s(n+1)$ for the remainder of the block and (iii) the $(n+1)^{th}$ block starts in state $s(n+1)$. Noting that in block 0, the automaton starts in the initial state $s(0)$, provides the base case for an inductive argument which guarantees that the automaton finds itself in state $s(n)$ at the beginning of the n^{th} block, causing the output to be $o(n)$ as required. □

B. Latency Dependent Conditions

So far, we tried to identify programs which are robust under an input slowed down by an unknown latency. If one knows that the inputs of a program are going to slow down by some uniform latency function $\lambda(t) = c$, where c is a constant, it is possible to check whether the program is robust for that particular scenario using the following weakened conditions.

Condition 4 requires that for any state s, the state reached by the automaton after the occurrence of a specific input tuple,

[1] A program is stateless if the output depends solely on the input at that point in time.

 ISBN: 978-1-61208-277-6

$\tau(a, s)$, is the same state reached after the occurrence of $c+1$ such input tuples, which we denote by $\tau^{c+1}(a, s)$.

Theorem 6: (Condition 4 — Immediate-At-First-Robustness). If $\forall a, s \cdot \tau(a, s) = \tau^{c+1}(a, s)$ for some positive natural number $c \geq 2$, the program is immediate-at-first robust for latency functions of the form $\lambda(t) = c$

Proof: When processing an input vector i, the automaton uses the current input $i(t)$ and state $s(t)$, to compute the output $o(t)$ and next state $s(t+1)$. When input i has latency λ, the program receives consecutive blocks of constant inputs of size $c+1$, with the n^{th} block consisting of tuples of the form $i(n)$. For the program to be immediate-at-first robust, the output at the beginning of the n^{th} block must have the form $o(n)$.

Suppose that at the beginning of the n^{th} block the automaton finds itself in state $s(n)$. Then at the beginning of the $(n+1)^{th}$ block it is in state $s(n+1)$ on account of the following facts: (i) at the first time instant in the n^{th} block the automaton moves to $s(n+1)$ and (ii) the condition guarantees that after $c+1$ steps of the same input the automaton will return to $s(n+1)$. Noting that in block 0, the automaton starts in the initial state $s(0)$, provides the base case for an inductive argument which guarantees that the automaton finds itself in state $s(n)$ at the beginning of the n^{th} block, causing the output to be $o(n)$ as required. □

The final condition which will be considered requires that if an automaton is in state s, it will return to the same state s after c repetitions of the input.

Theorem 7: (Condition 5 — Immediate-At-First and Fast-Enough Robustness). If $\forall a, s \cdot \tau^c(a, s) = s$ for some positive natural number $c \geq 2$, the program is both immediate-at-first robust, as well as fast-enough robust, for latency functions of the form $\lambda(t) = c$.

Proof: When processing an input vector i, the automaton uses the current input $i(t)$ and state $s(t)$, to compute the output $o(t)$ and next state $s(t+1)$. When input i has latency λ, the program receives consecutive blocks of constant inputs of size $c+1$, with the n^{th} block consisting of tuples of the form $i(n)$. For the program to be immediate-at-first robust, the output at the beginning of the n^{th} block must have the form $o(n)$. Similarly, for a program to be fast-enough robust, the output at the end of the n^{th} block must have the form $o(n)$.

Suppose that at the beginning of the n^{th} block the automaton finds itself in state $s(n)$. Then at the end of the n^{th} block it is in state $s(n)$ on account of the fact that the automaton returns to its original state after c transitions of the same input. This state also combines with input $i(n)$ to ensure passage to state $s(n+1)$ at beginning of the $(n+1)^{th}$ block. Noting that in block 0, the automaton starts in the initial state $s(0)$, provides the base case for an inductive argument which guarantees that the automaton always finds itself in state $s(n)$ at the end of the n^{th} block, causing the output to be $o(n)$ as required for fast-enough robustness, and in state $s(n+1)$ at the beginning of the $(n+1)^{th}$ block guaranteeing that the output is $o(n+1)$ as required by immediate-at-first robustness. □

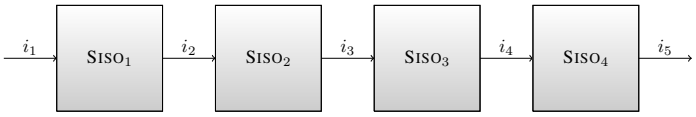

Fig. 3. SISO program broken into primitive programs

VI. CASE STUDY

The static and dynamic analysis theorems were applied to six Boolean Lustre programs to examine whether these are strong enough to deduce slowdown robustness. For comparison purposes, a manual analysis of these programs was also performed in order to discover which robustness properties each program satisfies or fails to satisfy. The programs under consideration, with the actual properties satisfied by each are listed below:

- RCA, a (stateless) ripple carry adder which satisfies stretch robustness.
- RISE, a program which receives a Boolean stream and detects the presence of rising edges, and which satisfies stutter robustness and immediate-at-first robustness.
- SWSR, a switch with a set and reset input, which satisfies stretch robustness.
- TOGGLE, a switch with a toggle input, which does not satisfy any property for every latency function.
- SISO, a serial in serial out register, which satisfies stutter robustness.
- PIPO, a parallel in parallel out register, which satisfies stutter robustness and immediate-at-first robustness.

We shall now discuss the application of the static and dynamic analysis theorems to the programs in question. To illustrate how the static analysis theorems can be employed to reason about a program, we will consider their use to prove that the SISO register program is stutter robust.

Example 1: Since the SISO program has 4 equations, we first break it down into four separate primitive programs SISO$_1$, SISO$_2$, SISO$_3$ and SISO$_4$ as shown in Fig. 3, where SISO$_j = \langle\{i_j, i_{j+1}\}, \{i_j\}, \{i_{j+1}\}, \{i_{j+1} = true\ fby\ i_j\}\rangle$

It is clear that each such program is an fby primitive program, and that these primitive programs can be composed through fully connected composition to obtain the program SISO. This can be done by starting from SISO$_1$ and sequentially composing the programs SISO$_2$, SISO$_3$ and SISO$_4$. Since SISO can be built from stutter robust primitives and through stutter robustness preserving compositions, we can conclude that it is stutter robust.

Table I illustrates the results which can be obtained in a similar manner through the static analysis of the programs in question. An entry in the table indicates whether the corresponding program can be shown to satisfy a particular robustness property or not through this technique. Within an entry, a ✓ symbol indicates that the program was found to satisfy the property. In addition, a ? symbol indicates that the static analysis yielded an inconclusive result, while a − symbol indicates that a test was unnecessary since the program was found to satisfy the stronger property of stretch robustness.

 ISBN: 978-1-61208-277-6

TABLE I
RESULTS OBTAINED THROUGH STATIC ANALYSIS

Property/Program	RCA	RISE	SWSR	TOGGLE	SISO	PIPO
STR	✓	?	?	?	?	?
STU	-	?	?	?	✓	?
IAF	-	?	?	?	?	✓

TABLE II
RESULTS OBTAINED THROUGH DYNAMIC ANALYSIS

Property/Program	RCA	RISE	SWSR	TOGGLE	SISO	PIPO
STR	✓	?	?	?	?	?
STU	-	?	?	×	?	?
IAF	-	✓	✓	$\checkmark_{c=2}$	?	✓
FE	-	?	?	$\checkmark_{c=2}$	?	?

As one can see, the static analysis reveals that the ripple carry adder is stretch robust, that the SISO register is stutter robust and that the PIPO register is immediate-at-first robust. Static analysis thus yields results when the programs have a simple structure in terms of the interconnections between the component primitive programs.

We now illustrate how dynamic analysis can be applied by means of another example. We shall show that the Toggle Switch program TOGGLE is both immediate-at-first robust as well as fast-enough robust for the latency function $\lambda(t) = 2$.

Example 2: Starting from the TOGGLE program, we first obtain the automaton representation of the program by using the construction outlined in Definition 1. This yields the automaton depicted earlier in Fig. 1. By observing the structure of the automaton, we note that from any state, taking 2 transitions with the same input tuple returns the automaton to the same state. The program thus satisfies the properties in question through the use of Theorem 7.

Table II summarises the results obtained through the dynamic analysis of the programs under consideration. In addition to the earlier conventions, an $\times$ symbol indicates that the program was found not to satisfy the property in question, while a ✓ symbol with subscript $c = 2$, indicates that the program has been proven to satisfy the property for the latency function $\lambda(t) = 2$ through the use of a latency dependent condition. In practice, BDD techniques were used to evaluate the conditions, and the evaluation was instantaneous for the programs in question.

Dynamic analysis enlarges the scope of automatically derived well behaviour results to programs which have more complex structures. The ripple carry adder is reconfirmed as strech robust and the PIPO register has been reconfirmed imemdiate-at-first robust. In addition, the rising edge program, the switch with set and reset program and the PIPO register have been shown to be immediate-at-first robust, and the toggle switch has been conclusively shown not to be stutter robust. More over, the toggle switch program has been shown to be both immediate-at-first robust and fast-enough robust for the specific latency function $\lambda(t) = 2$.

While not all of the properties satisfied by the programs have been discovered through the automated analysis, the combination of static and dynamic analysis has revealed many details about the well behaviour of the programs in question. The number of programs which have been proved immediate-at-first robust indicates that the condition which detects it might be applicable for some interesting set of programs. On the other hand, the results obtained using the latency dependent conditions are encouraging as they indicate the possibility of satisfying a property under a particular slowdown scenario even though the program might not satisfy it in general. One can also note that the two approaches complement one another; in particular, unlike dynamic analysis, static analysis can be used to reason about programs which satisfy stutter robustness.

VII. RELATED WORK

The discrete theory of slowdown considers the effect of slowing down all the input streams of a stream processing program by the same amount through the addition of stutter. There are various other models of slowdown which can be found in the literature. The theory of latency insensitive design [5] allows streams to slow down through the addition of explicit stall moves into those streams. In reaction to performing a stall move on an input stream, a program reacts by performing a procrastination effect, that is by inserting additional stall moves in its other streams to ensure that causality between the events of a program is preserved. A program is said to be patient if it knows how to perform a procrastination effect in response to any possible stall move. In other words, the program is always able to delay its operation in response to slow input without breaking. Patience is thus a form of robustness to delays in the process' streams, but which, unlike our properties, does not dictate the exact form which this robustness should take.

In the theory of polychronous processes [6], used to give a semantics to the synchronous language Signal [3], streams do not have to take a value at every time instant. Given a particular program behaviour, consisting of the input and output streams of a program, the operations of stretching and relaxation can be used to obtain a slower program behaviour. Stretching remaps the time instants at which the values occour on each stream, preserving the order of values in each stream, and the simultaneity of values between different streams. The stretching operation stretches all the streams by exactly the same amount and is similar to how a stretch robust program would behave when its inputs are slowed down. On the other hand, when relaxation slows a behaviour, it only guarantees that the order of values within each stream is preserved. The notion of relaxation which arises when all input streams are slowed down by one amount, and all output streams by another amount, is similar to how to a stutter robust program would behave under input slowdown. Signal guarantees that all its programs are stretch closed (a property analogous to stretch robustness), but this is only possible because no additional values are ever inserted as a result of slowing down a stream.

Reasoning about slowdown and speedup for continuous time behaviour has been investigated in [2]. The behaviour of a program can be slowed down by stretching these real-time signals through time by using the concept of time transforms.

 ISBN: 978-1-61208-277-6

The concept of a latency function can be seen as a discrete time version of a time transform. When one slows a behaviour through a time transform, all streams are slowed down exactly by the same amount. This manner of slowing down a behaviour corresponds to how one would expect a stretch robust system to react in our discrete theory.

Stutter invariance for Linear Temporal Logic (LTL) properties has been investigated in [7], in which a stuttered path slows down all inputs and outputs of a program by the same amount. Stutter invariant properties are ones which, if they are satisfied by a program, then they are also satisfied by all stutterings of its behaviour. If a Lustre program is stretch robust, then its inputs can be safely slowed down without the risk of breaking the constraint imposed by a stutter invariant property on the program.

The theory of stability [8] considers programs whose outputs fluctuate when their inputs are kept constant. Programs which do not exhibit such a phenomenon are said to be *stable*; when the inputs of these programs are unchanged, the outputs will converge to stable values after a finite period of time. The concept of stability relates to the concept of fast-enough robustness. An input which has stopped changing is similar to an input which is stuttering when considered over some finite horizon of time. While the theory of stability requires the output of a system to eventually converge to some particular value, fast-enough robustness requires an output to converge to an expected value before the sequence of repetitions ends.

Instead of checking whether a system exhibits certain classes of behaviour when an environment changes, it is possible to check whether a system degrades gracefully when the environment misbehaves. In [9] the authors consider a robustness approach in response to environments which fail to obey the assumptions made during system design. A system is said to be robust, if a small number of violations of the environment assumptions causes only a small number of violations of the system specification.

It is also possible to use a probabilistic approach to understand how changes in the environment are propagated through the system's components, and how the behaviour of these components under changed or missing input contributes to cause unacceptable system wide behaviour [10]. From our perspective, the general approach is interesting because it can help to isolate which components misbehave under slow input, causing a complex system to fail.

VIII. Conclusions

Since input stutter can arise in various situations, especially in systems which finely sample input, it is crucial that such systems do not change their behaviour as such transformations on their input occur. In this paper we have identified a number of different levels of robustness with respect to slowdown which one may require, and presented sound checks using static analysis of the code or using symbolic verification techniques over the system's behaviour.

Of these robustness properties, the most restrictive, stretch robustness is highly compositional, and relaxing it to obtain the weaker properties loses this compositionality property. Dynamic analysis allows for the analysis of programs on a global level, at an increased computational cost. The two approaches can, however, be combined, allowing for the analysis of more complex programs.

One major restriction of our results is that we assume that all the inputs of the system are slowed down by the same amount. In practice, this may be too strong a restriction, for instance with some nodes using a combination of external inputs and streams coming from other nodes and which may have been slowed down further. Another restriction is that we limit our dynamic analysis techniques to Boolean Lustre programs or circuits.

In the future, we plan to relax this constraint by using control graph analysis techniques to programs with numeric values, using approaches similar to [11].

References

[1] P. Caspi, D. Pilaud, N. Halbwachs, and J. Plaice, "Lustre: a declarative language for programming synchronous systems," Proc. 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages (POPL '87), ACM, Jan. 1987, pp. 178 - 188.

[2] C. Colombo, G. Pace, and G. Schneider, "Safe runtime verification of real-time properties," Proc. 7th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS '09), Springer Verlag, Sep. 2009, pp.103–111.

[3] A. Gamatié, Designing embdedded systems with the SIGNAL programming language - synchronous reactive specification. Springer, 2010.

[4] N. Halbwachs, P. Raymond, and C. Ratel, "Generating efficient code from data-flow programs," Proc. 3rd International Symposium on Programming Language Implementation and Logic Programming (PLILP '91), LNCS 528, Springer Verlag, Aug. 1991, pp. 207–218.

[5] L. Carloni, K. Mcmillan, and A. Sangiovanni-Vincentelli, "Theory of latency insensitive design," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 20(9), Sep. 2001, pp.1059–1076.

[6] P. Le Guernic, J.-P. Talpin, and J.-C. Le Lann, "Polychrony for system design," Journal of Circuits, Systems and Computers, vol. 12(3), Jun. 2003, pp.261–304.

[7] D. Peled and T. Wilke, "Stutter-invariant temporal properties are expressible without the next-time operator," Information Processing Letters, vol. 63(5), Sep. 1997 pp. 243–246.

[8] N. Halbwachs, J.-F. Héry, J.-C. Laleuf, and X. Nicollin, "Stability of discrete sampled systems," Proc. 6th International Symposium on Formal Techniques in Real-Time and Fault-Tollerant Systems (FTRTFT '00), Springer Verlag, London, Sep. 2000, pp.1-11.

[9] R. Bloem, K. Greimel, T. Henzinger, and B. Jobstmann, "Synthesizing robust systems," Proc. Formal Methods in Computer-Aided Design (FMCAD '09), IEEE Computer Society, Nov. 2009, pp. 85–92.

[10] X. Ge, R. Paige, and J. McDermid, "Probabilistic failure propagation and transformation analysis," Proc. 28th International Conference on Computer Safety, Reliability and Security (SAFECOMP '09), LNCS 5775, Springer Verlag, Sep. 2009, pp. 215–228.

[11] B. Jeannet, N. Halbwachs, and P. Raymond, "Dynamic partitioning in analyses of numerical properties," Proc. Static Analysis Symposium (SAS'99), LNCS 1694, Springer Verlag, Sep. 1999, pp. 39–50.

 ISBN: 978-1-61208-277-6

Prediction System of Larynx Cancer

Benjamín Moreno-Montiel and Carlos Hiram Moreno-Montiel
Posgrado Ciencias y Tecnologías de la Información
Universidad Autónoma Metropolitana - Iztapalapa,
México D.F., México
opelo1209@yahoo.com, hiramoreno@gmail.com

Abstract—In the task of data classification, there exist many uses and applications such as Credit assignment, Business, games Development, gene Research in public health problems, among others. In this research there is a large collection of data for treatment and prevention of some diseases, the most complex is the study of Cancer. The databases there have provided valuable knowledge useful for study of this disease that in many cases is unknown. An example of these databases is at the Centro Medico Nacional Siglo XXI, with information of human laryngeal carcinoma (LaCa). In this paper, we propose a Prediction System of Larynx Cancer (PSLC) to apply the task of classification of this type of databases to obtain novel knowledge for LaCa. The prediction system has two components, one component is the transformation and selection of data, the second component is a set of classifiers to obtain the prediction of life of sample patients with this type of cancer. With this prediction system, we found that, when there is an increase in CRBP-1 gene, it was correlated with patient survival; this allowed us to implement a Hybrid Classifier of Decision Rules (HCDR). The HCDR obtained the highest predictive value using genes CRBP-1 and provided a better degree of accuracy, with more than 90%, in comparison with different classifiers, indicating that the PSLC has a high degree of reliability.

Keywords-Data Minning; Classification; Classifier; Biochip genetic; Larynx Cancer.

I. INTRODUCTION

One of the most recent advances in genomic science is the search of genes responsible for alterations in different organisms, generating new research, in areas such as Agriculture, Medicine, Biomedical Sciences, among others. Some techniques, such as comparative genomic hybridization [6] and immunohistochemical assays on tissues [14] can determine the possible origin of different diseases such as cancer. By exhaustive search techniques and use of decision trees [2], there have been developed systems that make use of these techniques, finding relationship between genetic patterns of patients with some types of cancer, to provide prevention and early treatment.

An example of these systems is OncoTree [11] (for renal carcinoma progression), which finds relationship between genetic patterns of Renal Cancer using decision trees. Although having good results on this type of cancer, this system has two limitations: on the one hand, it can only do an analysis of renal cancer, and, on the other hand, limiting the use of this system is reserved only for some businesses due to their high prices.

There exist a small number of these and other predictive systems in only some countries; with high prices for use and analysis of results, limiting research in this area worldwide, this is a main problem with cancer prediction systems.

Because of this problem in some institutions [4], studies have been generated on different types of cancer, seeking to develop prediction systems like Onco Tree. An example of these studies was conducted in the Centro Medico Nacional Siglo XXI in Mexico City, which obtained a database (DB) with information about Larynx Cancer. This DB has the study of 21 patients with this type of cancer for over five years, which represents one of the largest repositories in Mexico.

The problem with this DB is that only specialists in the area of genetics have studies of how this may originate this type of cancer at the chromosomal level. In a study conducted with the Universidad Autónoma Metropolitana (UAM) and Instituto de Ciencias y Tecnologías de la Información (ICyTDF), we had access to this DB, for which we proposed as a way to find these patterns using Data Mining, such as those used in Onco Tree.

Data Mining [10] is the exploration and analysis to identify patterns non trivial (knowledge) within large amounts of data, which may be valid, novel, potentially, useful and understandable. With Data Mining we solve many tasks [13] as Prediction, Classification, Identification, Grouping and Association.

In this paper, we propose a Prediction System of Larynx Cancer (PSLC), with which we use the first and second Data Mining tasks. The prediction we use to find the correlations of the attributes within the DB's of Genetic Markers (dbGM). The task of classification is used to make the analysis and diagnosis of potential patients using the correlations found in the prediction task.

The PSCL is composed of two components; the first component is the transformation and selection of data, because dbGM has a special format called ISCN (An International System for Human Cytogenetic Nomenclature), which corresponds to all nomenclature that is handled in the research area of Human Genetics.

For the second component, we mix the prediction and classification tasks to get a prediction of laryngeal cancer in patients who are predisposed to have this type of cancer. At this stage we call the Engine of Operation (EM) of PSCL, which we use to obtain the experimental results of the classification of dbGM.

 ISBN: 978-1-61208-277-6

This paper is organized as follows. In Section 2, we will discuss previous work on prediction systems and the main algorithms used for constructing such systems. In Section 3, we describe how we build the PSLC. In Section 4, the tests that were performed to a DB of genetic biochips, will be discussed along with the results obtained using several Data Mining classifiers. Finally, we will present a conclusion and future work steps.

II. PREVIOUS WORK

A. Background of LaCa

Laryngeal cancer (LaCa) represents an important public health problem mainly affecting people over fifty years worldwide. Several methods are used for treatment and prevention. Comparative genomic hybridization (CGH) has been widely used in cancer research [8], [10], [14] the detection of specific patterns of chromosomal imbalances, for example, loss of chromosome 13 in all carcinomas cell carcinomas of the larynx [9], [10].

However, in the spatial resolution of CGH not much is known about the identity of specific genes that could be targeted chromosomal regional imbalances. The CGH array solves this problem by increasing the sensitivity for detection of changes in DNA copy number in specific loci (which are the specific location of a gene or DNA sequence on a chromosome) through the use of genomic DNA fragments is defined by a mapping location.

It is known that these arrays extend over a solid surface, resulting in a resolution of the imbalances in a number of copies of a single gene level [12]. To refine patterns of chromosomal imbalances present in squamous cell carcinoma of the larynx, and especially to identify the specific genes that could target the of copy number changes in this tumor type, array CGH is applied with oncogenes, tumor suppressor genes or some other genes associated with cancer.

This is achieved by determining the relation of level chromosomal patterns of samples obtained using different classification models in data mining, to determine which genes are involved, below some methods are described for classifying data. In the next section we review some data mining classifiers.

B. Several Methods for classification

In the literature, there have been a large number of classifiers that allow us to differentiate a set of samples according to the category or as usually called the class to which they belong.

There is a large number of classifiers [5] for instance the classifiers based on decision trees as adultery, C4.5 and ID3, classifiers based on decision rules as Decision Tables and Decision List, ensemble-based systems such as Bagging and Boosting, classifiers based on separating classes by hyperplanes such as Support Vector Machine (SVM) classifier, among others.

In this paper, we developed a Hybrid Classifier of Decision Rules (HCDR) [2], which was incorporated in the information on the main genetic markers within the dbGM, which allow us to obtain better performance measures. Since this classifier has the better results, in this section we focus on the previous work of this classifier.

There are three ways [13] of how to build a classifier based on decision rules, which are described below:

- **Decision Trees:** With this method we create a set of rules, each of them for each leaf of the tree, which are easy to interpret.
- **Specific algorithms for rule induction:** The language of representation of decision rules is essentially propositional. Is this method, each rule is learned one by one, so each time it selects a rule, the examples that are covered by the selected rule are removed from the training set.
  The process is repeated iteratively until a stop condition is fulfilled. To learn a rule begins with rules as general as possible, then these records are being added to maximize classification accuracy of this rule.
- **Models based on association rules:** The aim of the association rules is to find associations or correlations between items or objects in the database. Wanted the best association rules, in order to overcome some limitations of the models based on decision trees, which consider only the attributes one by one partially.

Once we review the main strategies on how to build classifiers based on decision rules, in the next section we describe how we build the PSLC we propose.

III. PREDICTION SYSTEM OF LARYNX CANCER (PSLC)

A. PSLC components

The PSLC has two components [7], the component of Transformation and Selection of Data (TSD) and the Engine of Operation (EO).

In the TSD component receiving data input, either read from a file or entered manually by people who will use the PSLC, which have a special format ISCN. The function of this component will convert this format to a matrix format for easier the processing in EO component.

In the EO component, once the data are in the matrix format we perform the classification of each patient tested. This component is implemented in the HCDR, which will allow us to make a prediction of what kind of life has one patient. The following sections explain in more detail each component of PCS_EME.

 ISBN: 978-1-61208-277-6

B. Transformation and Selection of Data

For this component, we implement techniques to apply the pre-processing and transformation phases on dbGM. For these two phases we implement two algorithms of Digitization or Labeling and Discretization, which perform the pre-processing and transformation phases. First, we show the ISCN format, taking an example of genetic biochip BD; then, later, we describe each algorithm of the TSD component.

1) ISCN Format

The DB has some special format called ISCN, which has an entire nomenclature that is handled in the area of Human Genetics, for the specific case of the dbGM.

These nomenclatures are reduced to a subset, which contains the information about possible genetic alterations associated with certain cancers. In one study of Centro Médico Nacional Siglo XXI of Instituto Mexicano Del Seguro Social, they obtained a dbGM with 19 records of patients with laryngeal cancer and two patients with cervical cancer [4], over five years.

They obtained this dbGM from progenetix (this database containing an overview of the anomalies in the number of copies of Human Cancer of Comparative Genomic Hybridization (CGH)), to generate a karyotype and a grouping of data; one record of this base is as follows:

L1 rev ish enh(1p36.22, 1p13.1, 1qtel, 2q14, 3p14.2, 3q26.3, 5q21q22, 5q33q35, 7q32q34, 8p22, 8p22q21.3, 8q24qter, 9q22.3, 14qtel, 15qtel, 16qtel, 17ptel, 17q23, 20q13.2, 20qtel, Xq12) dim enh(19ptel) amp enh(1q21, Xp22.3)

As we can see, this information is totally unknown because it is in a specialized format, such as the ISCN, but each acronym has a special meaning, which we are going to describe for each acronym that appears on this record:

- *L1:*Label assigned to the patient.
- *rev, ish, enh, dim and amp:* Techniques used to obtain genetic information, *rev* is the technique called Reverse, *ish* is the technique called In Situ Hybridization, *enh* is the technique called Enhanced, *dim* is the technique called Diminished and finally *amp* is the technique called Amplified Signal.
- *1p36.22:*It is the first record in the parentheses, the number at the start leading *1* is the number of chromosomes, *p* is the kind of arm that is in this case *p* is the short arm, if occupied *q* it would be the long arm, and finally *36.22* refers to the area of the chromosome where there is a change.

We can see that each record has a set of implicit information, which is difficult to handle for most classifiers, which is why we implement a method of transformation for these data. To achieve this objective, we implement digitization and discretization algorithms to apply on this dbGM and we obtain a format more appropriate for the incorporation of classifiers, then we show how we apply these algorithms on dbGM

2) Implementing Labeling and Discretization techniques.

In the example of the previous subsection, there are terms that are used exclusively in the ISCN format, which represent the acronym of each element that compose a record of this type of the dbGM.

With these acronyms, the labeling and discretization techniques were applied for change to integer values, which are handled by classifiers more directly, as a first element each acronym that are present in the dbGM is listed.

Techniquesfor each patient

- rev ish enh
- dim enh
- amp enh
- amp
- dim

Chromosomes

- 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, X, Y

Type of arm on chromosome

- p: Short arm.
- q: Long arm.

Region where the genetic alteration is present:

- Real values associated with each of the chromosomes that can go terminal region (ter) at the telomeric (tel).

So, when we apply the labeling and discretization techniques, we have to assign a nomenclature to each of these acronyms, the nomenclature used is as follows:

Techniques:

- rev ish enh → 101
- amp enh → 102
- dim enh → 103
- amp → 104
- dim → 105

Chromosomes:

- From 1, 2, …, 22
- X → 23
- Y → 24

Arm:

- P → 300
- Q → 301

Region:

- Real values multiplied by 100
- ter → 4000
- tel → 4001

The allocation of these values was arbitrary, having this way the transformation of all acronyms of ISCN format. In Table I, we can see an example of the application of these techniques on the record shown in Section 1).

 ISBN: 978-1-61208-277-6

The allocation of these values was arbitrary, having this way the transformation of all acronyms of ISCN format. In Table I, we can see an example of the application of these techniques on the record shown in Section 1.

Record in ISCN format:

L1, rev ish enh (1p36.22, 1p13.1, 1qtel, 2q14, 3p14.2, 3q26.3,5q21q22, 5q33q35, 7q32q34, 8p22, 8p22q21.3, 8q24qter, 9q22.314qtel, 15qtel, 16qtel, 17ptel, 17q23, 20q13.2,20qtel, Xq12) dim enh(19ptel) amp enh(1q21, Xp22.3)

TABLE I. RECORD AFTER WE APPLY THE LABELING AND DISCRTETIZATION TECHNIQUES.

1011	300	3622	1	300	1310	1
	301	4000	2	301	1400	3
	300	1420	3	301	2630	5
	301	2100	5	301	2200	5
	301	3300	5	301	3500	7
	301	3200	7	301	3400	8
	300	2200	8	300	2200	8
	301	2130	8	301	2400	8
	301	4000	9	301	2230	14
	301	4000	15	301	4000	26
	301	4000	17	300	4000	17
	301	2300	20	301	1320	20
	301	4000	23	301	1200	104
	19	300	4000	101	1	301
	2100	23	300	2230		

In the record of Table I, there are only integer data so makes it easier handling it for the classifiers, however exist a problem because the size of registration is large and each record has a different length. For this reason we propose a way of how to make the registers have the same size, we do this by separating techniques which have in a register.

To show how we apply this change in records to have the same size, we use another record of dbGM, which contains the following information:

L8T2N0 amp (19ptel, 22q11.21, 22q11.2, 22q13.3, 22qtel) dim (19ptel)

As we can see in the previous record, there are two techniques, which are the amp and dim, this example will help to illustrate each phase that must be performed to transform the format ISCN to the format that handles PSLC.

The operation of transformation has four phases, which are as follows.

Phase 1:

In the record above, there are two techniques, amp and dim, in this phase we identify the techniques, are divided, and new records are formed according to the number of techniques, the new records are creating as follows:

- amp(19ptel, 22q11.21, 22q11.2, 22q13.3, 22qtel)
- dim(19ptel)

Phase 2:

For each new record, we must separate it into each genetic alteration, retaining the order for each technique, after which we apply this phase to the new records and they change as follows:

- amp
    1. 19ptel
    2. 22q11.21
    3. 22q11.2
    4. 22q13.3
    5. 22qtel
- dim
    1. 19ptel

Phase 3:

Since they have separate genetic alterations, we will proceed to separate in chromosome number, type of arm and area of the chromosome where the alteration is present, respecting each new nomenclature established, we carry out as follows:

- amp -> 104

Order	*Chromosome*	*Arm*	*Region*
1	19	300	4001
2	22	301	1121
3	22	301	1120
4	22	301	1330
5	22	301	4001

- dim-> 105

Order	*Chromosome*	*Arm*	*Region*
1	19	300	4001

Phase 4:

Once we have coded both records, all information representing a new record in the dbGM, so now instead of

 ISBN: 978-1-61208-277-6

having two records, it will have six records, but all the same size; the example obtained from this phase can be seen in Table II:

TABLE II. FIRST CODING OF RECORD

Technique	*Chromosome*	*Arm*	*Area*	*Order*
104	19	300	4001	1
104	22	301	1121	2
104	22	301	1120	3
104	22	301	1330	4
104	22	301	4001	5
105	19	300	4001	6

We can see that in this matrix format, there are the different techniques (amp and dim), with all information in them, but we add new information of each record, for this case we add two new attributes, number of changes and sequence.

The number of changes attribute, has the techniques used in some patient. In the sequence attribute, since each technique was encoded in a range of 101 to 105, for this new attribute the last digit is used and depending on whether one or more techniques used they are strung together, returning to the previous example used the technique amp and dim which have associated values 104 and 105 respectively, so that the new attribute would be 45. Finally, adding these two attributes, the final record we can see in Table III.

With this example, we explain how we performed the transformation the format of dbGM, for having a matrix format to carry out the EO component of PSLC.

3) *Operation Engine*

When performing the task of data classification, there are two sets, training and testing. With the training set the models of classification algorithms are constructed. Using the test sets the classification is performed for each of the records in the DB, which are carried out individually, i.e. record by record.

This is the traditional way of how to perform data classification, however for the dbGM is not possible to carry out the classification in the traditional way, since as we can see in Table III; a set of six records in the matrix format is equivalent to one record in the ICSN format.

In this paper, we develop a module called Engine of Operation (EO) of PSLC. The EO implements a set of classifiers as Naive Bayes, C4.5, k-NN and the HCDR, among others to perform the task of classification. The classification performed with these classifiers is the tradition; therefore we find a way to change it, for this we implement a voting criterion as used in classifiers based on ensembles.

A classifier based on ensembles [1], joins one or more types of classifiers to obtain improvements in performance measures, which is why we need a criterion to designate the classification of each example in the test set, there have been two voting criteria the Majority and Weighted voting criterion.

TABLE III. MATRIX FORMAT OF A RECORD

Tech nique	*Number ofchanges*	*Sequence*	*Chro moso me*	*Arm*	*Area*	*Order*	*Class*
104	2	45	19	300	4001	1	Survival
104	2	45	22	301	1121	2	Survival
104	2	45	22	301	1120	3	Survival
104	2	45	22	301	1330	4	Survival
104	2	45	22	301	4001	5	Survival
105	2	45	19	300	4001	6	Survival

In Majority Voting Criterion, each of the classifiers votes to decide to which class each example belongs to for the test set, eventually counting the votes and assigning the class majority. For Weighted Voting Criterion, each classifier has a weight, so each vote has a different weight, so that in the end has a weighted voting, but the class is assigned given the most voting weight.

With these two criteria, we decided to use the majority-voting criterion [3] to adapt the classifiers of EO, to exemplify this incorporation we can see the diagram in Figure 1, which shows the traditional way to carry out the classification (Tclass) and the applying of majority voting criterion for classification (MVclass).

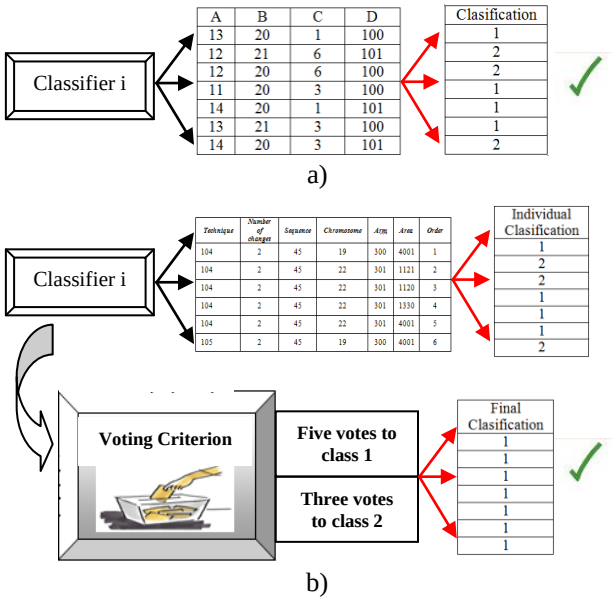

Figure 1. a) Traditional way to perform the classification. b) Classification with a majority voting criterion.

As we can see in Figure 1, the difference between Tclass and MVclass is the incorporation of the majority-voting criterion, in which the votes are counted there for the class 1

and the class 2. The class 1 has the most votes, this set of examples that represent a single example in the ISCN format, they assign the majority class, respecting the original meaning that we have for each record, we conclude this component; in the next section, we review the experiments and results obtained by the PSLC.

IV. EXPERIMENTS AND RESULTS

In this section, we show the results obtained by performing a series of tests with the BD of larynx cancer to measure the performance of the PSLC. The tests consisted in generating a number of test and training sets with different sizes; we can see these sets in Table IV.

TABLE IV. SELECTED TRAINING SETS.

Cases	Number of patients	Number of records
1	3	49
2	5	92
3	7	135
4	9	151
5	11	160
6	13	224
7	15	299
8	17	332
9	19	382
10	Full DB (21)	431

With each training set of Table IV, the MVclass were performed according to the following steps:

1. We take two sets of dbGM, a set classless and another set with the class of each record; these will be the test and training set respectively.
2. The test sets we use by some classifiers, to perform the MVclass of each example.
3. Once it gets the MVclass of each record of the test set, we compared them to their real classes.
4. At the end of this comparison, we obtained the performance measures that will allow us to evaluate each classifier.

At the end of the testing process, we choose the accuracy as the performance measure to evaluate each classifier. Accuracy provides information on the percentage of correctly classified examples, out of the entire test set; this performance measure is formally defined as follows:

$$Accuracy = \frac{(a + d)}{(a + b + c + d)} \quad (1)$$

In equation 1 has the following:

- (a, d) is the correctly classified examples and
- (c, b) is the classified incorrectly examples

It is noteworthy that for each of the tests we used cross-validation to determine the validity of each classifier we implement. Figure 2 and Table V show the final results we obtain in each of the tests that were performed.

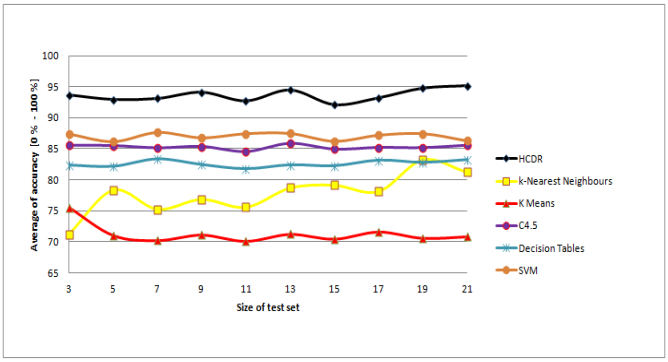

Figure 2. Graph of the final results.

TABLE V. FINAL RESULTS FOREACH CLASSIFIER

Size of test set	HCDR	k-NN	K Means	C4.5	Decision Tables	SVM
3	93.67	71.21	75.5	85.54	82.32	87.37
5	91.95	78.34	71.06	85.48	82.12	86.19
7	93.1	75.24	70.25	85.12	83.33	87.61
9	94.1	76.85	71.13	85.34	82.41	86.78
11	92.72	75.56	70.13	84.54	81.73	87.41
13	94.45	78.69	71.24	85.83	82.37	87.45
15	92.12	79.14	70.48	84.93	82.21	86.25
17	91.21	78.13	71.61	85.19	83.13	87.17
19	94.74	83.26	70.62	85.15	82.77	87.42
21	95.13	81.26	70.85	85.54	83.21	86.31
Average of accuracy	93.619	77.768	71.287	85.266	82.56	86.996

We can see in Figure 2 and Table IV that the HCDR showed better results than traditional classifiers, obtaining an average accuracy of 93,619. The HCDR was strengthened with genetic markers found in the database of the study after five years in patients with larynx cancer. By using new technologies such as scanning AXON, the targets are searched must be related to cancer of the larynx and cervical cancer.

Table VI shows the class for each patient in the study over five years. In these patients, we determined the main patterns present at the genetic level, which are present in ISCN format.

TABLE VI. PATIENTS WITH CANCER OF LARYNX, WITH THEIR RESPECTIVE CLASS AND PATTERNS.

Patient	Class	Pattern
L3	Survival	3q21q22 and 3q27q29
L4	Survival	1q25q31, 3q21q22 and 7qtel
L5	Poor survival	17q11.2

 ISBN: 978-1-61208-277-6

L6	Poor survival	5q21q22, 5q33q35, 8p22q21.3, 8q29qter and 7q32q34
L7	Poor survival	8p22q21.3 and8q29qter
L8	Survival	22q11.2 and 22q13.3
L9	Survival	3q21q22
L10	Survival	2q31q32
L11	Survival	5q33q35
L12	Survival	22q11.2 and 22q13.3
L13	Poor survival	1q25q31, 3q21q22, 3q27q29 and 7q21q22
L14	Poor survival	3q21q22 and 7p12.3p12.18p22q21.3
L15	Poor survival	1q25q31 and 3q21q22
L16	Poor survival	3q21q22 and 7p12.3p12.1
L17	Survival	3p12p13 and 17q21q22
L18	Survival	12q13q14
L19	Survival	1q25q31, 3q21q22, 3q27q29 and 7p12.3q12.1
L20	Poor survival	3q21q22, 3q27q29,7p12.3p12.1 and7q21q22
L21	Poor survival	5q33q35

With Table VI, we locate the presence of CRBP-1 and EGFR genes. CRBP-1 is a protein involved in the transport of retinol from its storage sites of the liver to peripheral tissues. Vitamin A plays an important role in a variety of cellular processes associated with epithelial tissue proliferation and differentiation. This protein could improve the condition of some form of cancer, when analyzing our database we observed that it actually improved the conditions of patients with larynx cancer.

In contrast, EGFR amplification could be associated with poor survival of patients with larynx cancer. A gain of CRBP-gene may have a protective effect and increased survival. These data suggest that alteration CRBP-1 gene and its expression in carcinomas of the larynx squamous provide prognostic information with greater potential patient survival.

Previously, Peralta et al. [4] reported the behavior of these two genes but with techniques in microarray data analysis. For this work became the chromosomal level analysis of each of the patients involved, and seek the relationship patterns of each sample to determine the genes present. In the same way, it was determined that the gene CRBP-1 was present in patients who survived the EGFR gene was present in poor survival. In this way it was shown at the chromosome level and table's decision techniques and data mining obtained the correct result.

By identifying these genetic patterns, we could implement the HCDR, which is able to incorporate decision rules that endorse the existence of these patterns in any of the branches of the decision tree constructed, which resulted an average accuracy of over 15% compared to traditional classifiers.

V. CONCLUSION AND FUTURE WORK

In this paper, we propose a Prediction System of Larynx Cancer (PSLC), for exploration and analysis of the databases of genetic markers (dbGM). The PSLC has two components, the Transformation and Selection of Data (TSD) component and the Operation Engine (EO).

With the TSD component, we perform data transformation that was in ISCN format to a matrix format for better data handling. This component was necessary since most of the classifiers are better suited to this type of data format. Otherwise if we had taken the raw ISCN format, many changes should have been made in the operation of each classifier. That is why we decided to not make these changes and implement the TSD module.

With the EO component implemented as an alternative way to perform the classification by incorporating a majority voting criterion (MVclass). With this alternative form of classification, we performed a series of tests with a set of classifiers. In Table V and Figure 2, the accuracy grew 15% with respect to the other classifiers considered for the tests; this gives us the result that the HCDR performs better than traditional classifiers.

This classifier considers the incorporation of decision rules with tree structures, together with the genetic markers, the CRBP-1 to survival of patients and EGFR to poor survival of patients, located in a series of preliminary tests.

With PSLC we proved that by incorporating two areas of knowledge (Artificial Intelligence and Genetics), different from one another, there may be generated more complex algorithms; such is the case of HCDR. With this model of classification we obtained a better accuracy for each one of the tests we made to the dbGM, comparing it with different traditional classifiers. The PCSL is an example of the new interdisciplinary projects of Science and Technology that combine more than one area of current research.

For the PSLC we consider its seminal work on the issue, since if we want to develop new versions incorporating other types of cancer, it will be directly because in tests we found that the genetic information of two different types of cancer, in this case larynx and cervical, show many similarities, so we left the base ready to scale this project.

ACKNOWLEDGMENT

Special thanks to Instituto de Ciencia y Tecnologia Del Distrito Federal - ICyTDF and Universidad Autónoma Metropolitana - UAM for their support for the realization of this system we propose in this paper. We also thank Dr. Mauricio Salcedo-Vargas, which belongs to the Centro Médico Nacional Siglo XXI of Departamento de Oncología for their valuable collaboration is aspects of genetics.

REFERENCES

[1] B. Moreno Montiel and R. Mac Kinney Romero,"A Hybrid Classifier with Genetic Weighting," in Proceedings of the Sixth International Conference on Software and Data Technologies, July 2011, vol. 2, pp. 359–364.

[2] T. Gunnar Houeland and A. Aamodt, "An efficient hybrid classification algorithm: an example from palliative care,"

Proceedings of the 6th international conference on Hybrid artificial intelligent systems, September 2011, vol. II, pp. 197–204.

[3] E. Hüllermeier and S. Vanderlooy, "Combining predictions in pairwise classification: An optimal adaptive voting strategy and its relation to weighted voting," Pattern Recognition, January 2010, vol. 43, pp. 128–142, doi:10.1016/j.patcog.2009.06.013.

[4] R. Peralta, M. Baudis, G. Vazquez, S. Juárez, H. Decanini, D. Hernandez, F. Gallegos, A. Valdivia, P. Piña, and M. Salcedo, "Increased expression of cellular retinol-binding protein 1 in laryngeal squamous cell carcinoma," Journal of Cancer Research and Clinical Oncology. January 2010, vol. 136, pp. 931–938, doi: 10.1007/s00432-009-0735-9. Epub 2010 Jan 7.

[5] X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. Yu, ZH. Zhou, M. Steinbach, D. Hand, and D. Steinberg, "Top 10 algorithms in data mining," Knowledge and Information Systems, April 2009, vol. 14, pp. 1–37, doi:10.1007/s10115-007-0114-2.

[6] M. Shinawi and SW. Cheung. "The array CGH and its clinical applications,". Drug Discov Today, September 2008, vol. 13, pp. 760–770, doi:10.1016/j.drudis.2008.06.007. PMID 18617013.

[7] I. Witten and E. Frank, "Data Mining: Practical Machine Learning Tools and Techniques," Morgan Kaufmann Publishers, January 2005, 2nd edition, pp. 560.

[8] D . Albertson, and D. Pinkel,"Genomic microarrays in human genetic disease and cancer," Hum Mol Genet, October 2003, no. 2:R, pp. 145-52, doi: 10.1093/hmg/ddg261.

.

[9] M. Kujawski, M. Rydzanics, M. Sarlom-Rikala, and K. Szyfter, "Rearrangements involving the 13q chromosome arm committed to the progression of laryngeal squamous cell carcinoma," Cancer Genet Cytogenet, August 2002, vol. 137, No 1 pp.54-58.

[10] S. Struski, M. Doco-Fenzy, and P. Cornillet-Lefebvre "Compilation of published comparative genomic hybridization studies", Cancer Genet Cytogenet, May 2002, vol. 135, No 1 pp. 63-90.

[11] F. Jiang, R. Desper, C. H. Papadimitriou, R. A. Schäffer , O. Kallioniem, J. Richter, P. Schraml, G. Sauter, M. J. Mihatsch, and H. Moch, "Construction of evolutionary tree models for renal cell carcinoma from comparative genomic hybridization data,"Cance research, November 2000, vol. 60, No22 pp. 6503-6509.

[12] D. Pinkel, R. Segraves, D. Sudar, S. Clark, I. Poole, D. Kowbel, C. Collins, W. L. Kuo, C. Chen, Y. Zhai, S. H. Dairkee, B. M. Ljung, J. W. Gray, and D. G. Albertson, "High resolution analysis of DNA copy number variation using compartive genomic hybridization to microarrays," Nature Genetics, October 1998, vol. 20, pp. 207-211,, doi: 10.1101/gr.2012304.

[13] T. M. Mitchell, "Machine Learning", McGraw-Hill Science/Engineering/Math, March 1997

[14] S. Solinas-Toldo, S. Lampel, S. Stilgenbauer, J. Nickolenko, A. Benner, H.Döhner, T. Creme, and P.Lichter, "Matrix-based comparative genomic hybridization: biochips to screen for genomic imbalances," Genes Chromosomes Cancer, December 1997, vol. 20, No 4 pp. 399-407, doi: 10.1002/(SICI)1098-2264(199712)20:4<399::AID-GCC12>3.0.CO;2-I.

 ISBN: 978-1-61208-277-6

Introduction in First-Order Combinatorics
Providing a Conceptual Framework for Computation in Predicate Logic

Mikhail G. Peretyat'kin
Institute of Mathematics and Mathematical Modeling
125 Pushkin Street, 050010 Almaty, Kazakhstan
e-mail: m.g.peretyatkin@predicate-logic.org

***Abstract*—In this work, we introduce general specifications for the concepts of finitary and infinitary first-order combinatorics as well as give preliminary definitions of semantic layers of model-theoretic properties connected with these combinatorics. We use only the simplest notions of first-order logic and algorithm theory together with elementary properties of signature reduction procedures and constructions of finitely axiomatizable theories known in common practice. The work represents an ideological basis and starting point for investigations on expressive power of first-order predicate logic.**

Keywords—first-order logic; computation; theory; computably axiomatizable theory; interpretation; signature reduction procedure; combinatorics.

I. INTRODUCTION

The principal problem concerning expressive power of first-order predicate logic was solved by W. Hanf [2][3], who proved that, for any computably axiomatizable theory T, there is a finitely axiomatizable theory F together with a computable isomorphism $\mu: \mathcal{L}(T) \to \mathcal{L}(F)$ between their Tarski–Lindenbaum algebras. Moreover, in the same work [3], Hanf gives a direct formula that presents the isomorphism type of the Tarski–Lindenbaum algebra $\mathcal{L}(PC(\sigma))$ of predicate calculus $PC(\sigma)$ of a finite rich signature σ. The works of Hanf–Myers [4] and Myers [6] introduce a method of constructing computable isomorphisms between $\mathcal{L}(PC(\sigma_1))$ and $\mathcal{L}(PC(\sigma_2))$, where σ_1 and σ_2 are arbitrary finite rich signatures.

Subsequent work of Myers [7] describes an enhanced isomorphism between the Tarski–Lindenbaum algebras $\mathcal{L}(PC(\sigma_2))$ and $\mathcal{L}(PC(\sigma_3))$, where signature σ_2 consists of a single binary predicate, while σ_3 consists of a single ternary (or n -ary, $n > 3$) predicate. He builds a computable isomorphism $\mu: \mathcal{L}(PC(\sigma_2)) \to \mathcal{L}(PC(\sigma_3))$ such that, for any complete extension T' of $PC(\sigma_2)$ and corresponding complete extension S' of $PC(\sigma_3)$, $S' = \mu(T')$, the theories T' and S' are mutually interpretable in each other via so-called tuple-quotient interpretations (using a definable set of tuples of a finite length modulo a definable equivalence relation as a domain of the interpretation). Thereby, the corresponding completions will have rather like model-theoretic properties.

Our previous work [10] represents a universal construction of finitely axiomatizable theories controlling the structure of the Tarski–Lindenbaum algebra of a theory together with a large layer of model-theoretic properties, while the works [8] and [9] describe special methods of constructing isomorphisms between the Tarski–Lindenbaum algebras of predicate calculi of different finite rich signatures. Notice that, the methods in [8] are based on the universal construction providing computable transformation of the theory, that corresponds to the term "infinitary combinatorics". Furthermore, the methods in [9] are based on finite-to-finite signature reduction procedures providing first-order definable transformation of the theory, that corresponds to the term "finitary combinatorics". Therefore, a natural idea arises to use the combinatory terminology for further works in this direction.

Probably, any exact definition is impossible for the concept of combinatorics as well as for its particular cases such as "finite combinatorics" or "infinite combinatorics". However, some specifications are possible for these concepts if to restrict ourselves to the case of the language of first-order logic. The problem to define such specifications arises just in connection with the idea to define a new approach for investigations on expressive power of first-order predicate logic. Earlier, this problem was not even posed at all while the methods of first-order combinatorics were considered as obvious constructions of model theory available in the common practice; furthermore, different specialists considered different meanings of the term "first-order combinatorics" itself.

The given work introduces some general specifications to finitary and infinitary combinatorics. They are intended to be used during investigations on the problem of characterization of the Tarski–Lindenbaum algebras of predicate calculi of finite rich signatures; these algebras should be considered as generalized, i.e., enhanced with an assignment function within the finitary or infinitary semantic layer of model-theoretic properties. At such an approach, finitary first-order methods represent the finite (one can say, combinatorial) level of computation, while infinitary first-order methods represent the algorithmic level of computation in first-order predicate logic.

II. PRELIMINARIES

We consider theories in first-order predicate logic *with equality* and use general concepts of logic, model theory, algorithm theory, and constructive models found in Rautenberg [11], Hodges [5], Rogers [12], Goncharov and Ershov [1]. Generally, *incomplete* theories are considered.

A finite signature is called *rich* if it contains at least an n-ary predicate or function symbol for $n > 1$, or two unary function symbols. In this work, the signatures are

 ISBN: 978-1-61208-277-6

considered only, which admit Gödel's numbering of the formulas. Such a signature is called *enumerable*. In writing of a signature, capital letters are used for predicates, small letters for functions and constants, and superscripts specify arities of appropriate symbols. If $\mathfrak{M}$ is a model, $|\mathfrak{M}|$ stands for the universe set of $\mathfrak{M}$. If T is a theory, by $Mod(T)$, we denote the class of all models of T. The Tarski–Lindenbaum algebra of theory T over formulas without free variables is denoted by $L(T)$, while $\mathcal{L}(T)$ stands for the Tarski–Lindenbaum algebra $L(T)$ considered together with a Gödel numbering γ such that the concept of a computable isomorphism becomes applicable to such objects. Such isomorphisms between the Tarski–Lindenbaum algebras of theories were initially considered by Hanf [2].

Let T be a theory of signature σ and $\sigma' \subseteq \sigma$. An m-ary relation P^m is called *first-order definable* in T relative to σ' if there is a formula $\varphi(x_1, \dots, x_m)$ of signature σ' such that

$$T \vdash P(x_1, \dots, x_m) \leftrightarrow \varphi(x_1, \dots, x_m).$$

Relation P is called $\exists \cap \forall$*-definable* in T relative to σ', if there are formulas $\theta(x_1, \dots, x_m)$ and $\theta'(x_1, \dots, x_m)$ of signature σ', such that $\theta(x_1, \dots, x_m)$ is an $\exists$-formula, $\theta'(x_1, \dots, x_m)$ is a $\forall$-formula, and two following conditions are satisfied:

$$T \vdash P(x_1, \dots, x_m) \leftrightarrow \theta(x_1, \dots, x_m),$$

$$T \vdash P(x_1, \dots, x_m) \leftrightarrow \theta'(x_1, \dots, x_m).$$

Particularly, the formula $(\forall x_1 \dots x_m)(\theta(x_1, \dots, x_m) \leftrightarrow \theta'(x_1, \dots, x_m))$ must be true in the theory T. Similar definitions also apply for functions and constants instead of the relation P.

Theories T and S are called *first-order equivalent* or *isomorphic*, written as $T \approx S$, if S can be obtained from T by a finite number of operations of renaming signature symbols and by adding and eliminating those signature symbols that are first-order definable in terms of other signature symbols. Theories T and S are called *first-order* $\exists \cap \forall$*-equivalent* or *algebraically isomorphic*, written as $T \approx_a S$, if S can be obtained from T by a finite number of operations of renaming signature symbols and by adding and eliminating those signature symbols that are $\exists \cap \forall$-definable in terms of other signature symbols. Obviously, we have $T \approx_a S \Rightarrow T \approx S$ for arbitrary theories T and S.

An arbitrary set $\mathfrak{p}$ of complete theories of enumerable signatures which is closed under $\approx$ is said to be a *model property*, while a set $\mathfrak{p}$ of complete theories closed under $\approx_a$ is said to be an *algebraic property*. Both types of properties are called *model-theoretic* properties. Examples of model-theoretic properties of model type: "theory has a prime model", "theory is not stable". An example of property of algebraic type: "theory is model complete". By AL, we denote the set of all model-theoretic properties of algebraic type, while ML stands for the set of all properties of model type; the inclusion $ML \subseteq AL$ is obvious. An arbitrary collection L of model-theoretic properties is said to be a *semantic layer*. A set $L \subseteq ML$ is called a *model semantic layer*, while a set $L \subseteq AL$ is called an *algebraic semantic layer*. Notice that, any model semantic layer can be regarded as an algebraic layer. In the case when there is a computable isomorphism $\mu: \mathcal{L}(T) \to \mathcal{L}(S)$ preserving any model-theoretic properties within a layer L, the theories T and S are said to be *semantically similar* over the layer L, symbolically written as $T \equiv_L S$.

A. Demonstration of the relation of semantic similarity

It is a simple exercise to construct a computably axiomatizable theory T satisfying the following properties: T is decidable, the set of all complete extensions of T, called its *Stone space*, consists of a countable sequence $T_k, k \in \mathbb{N} \cup \{\omega\}$, such that, each of the theories $T_0, T_1, T_2, \dots$ is a stable theory without prime models and is finitely axiomatizable over T, while T_ω is not finitely axiomatizable over T, it is not stable and has a prime model. Applying the universal construction, [10,Th.0.6.1], we can find a finitely axiomatizable theory F together with a computable isomorphism $\mu: \mathcal{L}(T) \to \mathcal{L}(F)$ preserving any property in the following immediately listed semantic layer of model-theoretic properties:

$$L = \{"\textit{theory is stable}", "\textit{theory has a prime model}"\}.$$

Thereby, within the layer L, this theory F has exactly the same model-theoretic properties as T did. This example demonstrates concepts of a model-theoretic property, semantic layer, computable isomorphism between the Tarski–Lindenbaum algebras, as well as a possibility of applications of the universal construction.

B. Demonstration of model versus algebraic properties

Algebraic-type properties are thinner in comparison with those of model-type. Often, model-type properties are considered, while sometimes, thinner algebraic-type properties are also needed. For instance, let T be the theory of discrete linear orders considered in signature $\sigma = \{<^2, \lhd^2\}$, where

$$x \lhd y \leftrightarrow (x < y) \,\&\, (\forall z)(x \le z \le y \to (x = z \vee z = y)).$$

Since $\lhd$ is first-order definable relative to $<$, we can omit predicate $\lhd$ obtaining another theory T_0 of discrete linear orders in smaller signature $\sigma_0 = \{<^2\}$. Theories T and T_0 are isomorphic with each other; particularly, we have $T \equiv_{ML} T_0$. On the other hand, T and T_0 are not algebraically isomorphic because T is model complete; thus, all its complete extensions are model complete as well; on the contrary, there is a complete extension of T_0 which is not model complete. Thereby, $T \equiv_{AL} T_0$ does not the case.

III. CARTESIAN EXTENSIONS OF THEORIES

Let σ be a signature and

$$\xi = \langle \varphi_1^{m_1}, \varphi_2^{m_2}, \dots, \varphi_s^{m_s} \rangle \quad (1)$$

be a finite sequence of formulas of this signature, where φ_k is a formula with m_k free variables. Starting from a tuple ξ and an arbitrary model $\mathfrak{M}$ of signature σ, we will construct some new model $\mathfrak{M}_1 = \mathfrak{M}\langle \xi \rangle$ of signature

$$\sigma_1 = \sigma \cup \{U^1, U_1^1, \dots, U_s^1\} \cup \{K_1^{m_1+1}, \dots, K_s^{m_s+1}\} \quad (2)$$

as follows. As a universe for the model, we take the following set

$$|\mathfrak{M}_1| = |\mathfrak{M}| \cup A_1 \cup A_2 \cup \dots \cup A_s ,$$

where the pointed out parts are pairwise disjoint. In the part $|\mathfrak{M}|$, all symbols of signature σ are defined exactly as they were defined in $\mathfrak{M}$; in remaining, these symbols are defined trivially; U is defined by $U(x) \Leftrightarrow x \in |\mathfrak{M}|$; U_k is defined by $U_k(x) \Leftrightarrow x \in A_k$; predicate K_k represents a one-to-one correspondence between the set of tuples $\{\bar{a} | \mathfrak{M} \vDash \varphi_k(\bar{a})\}$ and the set A_k . So defined model $\mathfrak{M}\langle \xi \rangle$ is said to be *Cartesian extension of* $\mathfrak{M}$ by means of sequence (1), denoted by $\mathfrak{M}\langle \varphi_1^{m_1}, \varphi_2^{m_2}, \dots, \varphi_s^{m_s} \rangle$, or $\mathfrak{M}\langle \xi \rangle$ for short. Now, we consider a theory T of signature σ, and fix signature (2) for extensions of models. Let us define a new theory T' as follows

$$T' = Th\{\mathfrak{M}\langle \xi \rangle | \mathfrak{M} \in Mod(T)\}.$$

It is said to be *Cartesian extension* of T by means of sequence of formulas (1), denoted by $T\langle \varphi_1^{m_1}, \varphi_2^{m_2}, \dots, \varphi_s^{m_s} \rangle$, or $T\langle \xi \rangle$ for short. According to the construction, the theory $T\langle \xi \rangle$ is defined uniquely up to an algebraic isomorphism of theories; moreover, an interpretation $I_{T,\xi}$ of the source theory T in the target theory $T\langle \xi \rangle$ is naturally defined.

Now, we consider a sequence of formulas of signature σ of a more common form

$$\varkappa = \langle \varphi_1^{m_1}/\varepsilon_1, \varphi_2^{m_2}/\varepsilon_2, \dots, \varphi_s^{m_s}/\varepsilon_s \rangle, \quad (3)$$

where $\varphi_k(\bar{x})$ is a formula with m_k free variables, while $\varepsilon_k(\bar{y}, \bar{z})$ is a formula with $2m_k$ free variables. By $Equiv(\varepsilon_k, \varphi_k)$, we denote a sentence stating that ε_k is an equivalence relation on the set of tuples distinguished by the formula $\varphi_k(\bar{x})$. Let us repeat the construction given above with the only difference that $(m_k + 1)$-ary predicate K_k represents a one-to-one correspondence between the quotient set $\{\bar{a} | \mathfrak{M} \vDash \varphi_k(\bar{a})\}/\hat{\varepsilon}_k$ and the set A_k, where $\hat{\varepsilon}_k(\bar{y}, \bar{z}) = \varepsilon_k(\bar{y}, \bar{z}) \vee \neg Equiv(\varepsilon_k, \varphi_k)$. The obtained theory $T\langle \varkappa \rangle$ is said to be *Cartesian-quotient extension* of T by means of sequence of formulas $\varkappa$. Similarly to the previous case, the theory $T\langle \varkappa \rangle$ is determined uniquely up to an algebraic isomorphism of theories; moreover, there is a natural interpretation $I_{T,\varkappa}$ of the source theory T in the target theory $T\langle \varkappa \rangle$.

The introduced operations are used in further definition of first-order combinatorics.

1. Statement: *Up to an algebraic isomorphism of theories, each finite-to-finite signature reduction procedure represents a particular case of Cartesian extension of theories*.

Proof. Immediately, by Beth's Definability Theorem, [5, Th. 5.5.4]. □

IV. First-order combinatorics

First, we introduce some *common specification* in a compact form.

By *first-order combinatorics*, we mean transformation methods of countable (specifically, computably axiomatizable) theories, which can change both signature and axiomatic of the theory preserving, as much as possible, its model-theoretic properties. The emphasis is on the methods definable in first-order predicate logic, while the principal goal is the maximality of the collection of preserved model-theoretic properties. Moreover, the main objective is naturalness of the accepted specification. Significance of the complex of definitions for combinatorics is considered as higher if these definitions adequately correspond to an available approach to logic (particularly, in set theory or model theory). In the case of ambiguity in the choice of some technical details, the preference should be directed to the variants of concepts simplifying the situation or providing more perfect appearance. As an initial basis for the concept of combinatorics we take the class of signature reduction procedures, which are considered as a particular case of combinatorial methods in first-order logic. The common problem is to generalize these particular methods to maximum wide natural approach in such a way that so serious term as "combinatorics" would become acceptable here.

With this, the common specification is complete.

Now, we turn to develop the common idea in a mathematical form.

A signature reduction procedure is normally applied, when we are going to transform a given theory T having an infinite or too large finite signature to some new theory S having a small finite signature. Moreover, the target theory S must inherit from the source theory T all model-theoretic properties within a given layer $L = \{\mathfrak{p}_0, \mathfrak{p}_1, \mathfrak{p}_2, \dots\}$. Generally, specifications for the signature reduction methods are subordinated to the purpose to pass from T to S as large collection of properties as possible; thus, any exotic methods of signature reduction distorting some evident model-theoretic properties should be rejected. Ordinarily, the signature reduction procedure is determined by an interpretation I of T in S preserving the demanded properties. It is possible to establish (for instance, with the back-and-forth Ehrenfeucht method), that generally, such an interpretation I defines an isomorphism of the Tarski–Lindenbaum algebras $\mu: \mathcal{L}(T) \to \mathcal{L}(S)$ passing from T to S both structure of extensions of theory and any model-theoretic properties within the layer L from complete extensions of T to corresponding complete extensions of S. In many cases, reviewing is limited by complete theories

 ISBN: 978-1-61208-277-6

only; in this case, the Tarski–Lindenbaum algebras $\mathcal{L}(T)$ and $\mathcal{L}(S)$ are 2-element Boolean algebras.

Notice that, in technical realization, signature reduction procedures may consist of two or more separate stages. Particularly, first, a reduction from an infinite in some finite signature could be performed, while on the second stage, the obtained finite signature is reduced to the wished small finite signature. Another remark is that the universal construction of finitely axiomatizable theories (see [10, Th.0.6.1]) is a transformation from the class of computably axiomatizable theories in the class of finitely axiomatizable theories (of finite signatures); thereby, such a transformation can be considered as an improved variant of infinite-to-finite signature reduction procedure. Moreover, for the construction, the whole transformation procedure consists of so called *main stage* (performing the actual passage from a computably axiomatizable theory to a finitely axiomatizable theory) and a few auxiliary stages performing signature reductions of certain types. Practical observation shows that, the universal construction can control the same model-theoretic properties which are under control of infinite-to-finite signature reduction procedures. This definitely shows that, both signature reduction procedures and constructions of finitely axiomatizable theories should be considered jointly as an integrated complex of transformations of theories.

V. TWO TYPES OF FIRST-ORDER COMBINATORICS

Combinatorics of a given type is characterized by a definite set of used methods and by collection of those model-theoretic properties which are controlled by application of these methods. Since we consider combinatorics in first-order logic, the concept of a method is understood as some manner $\mathfrak{m}$ of first-order transformation of a computably axiomatizable theory T in another such theory S producing a computable isomorphism of the Tarski–Lindenbaum algebras $\mu: \mathcal{L}(T) \to \mathcal{L}(S)$; moreover, as a control over a model-theoretic property $\mathfrak{p}$ we mean that the isomorphism μ passes without change this property $\mathfrak{p}$ from any complete extension of T to corresponding complete extension of S. As mentioned above, some interpretation I of T in S is meant behind the isomorphism μ. Thus, by way of constructing an input theory T, it is possible to influence on properties of the target theory S within the semantic layer L of the controlled properties.

Now, we define *finite first-order combinatorics* or shortly *finitary combinatorics* as combinatorics that is determined by finitary transformation methods between first-order theories, and *infinite computable first-order combinatorics* or shortly *infinitary combinatorics* as combinatorics that is determined by effective infinitary transformation methods between first-order theories. Finite-to-finite ($f2f$) signature reduction procedures represent transformations of theory from one finite signature in another finite signature. They are examples of finitary methods in first-order logic; at the same time, some other finitary methods in this logic exist; particularly, any Cartesian-quotient (or Cartesian) extension of a theory represents a finitary first-order method. Infinite-to-finite ($i2f$) signature reduction procedures represent transformations of theory from an infinite enumerable signature in a finite signature. They are examples of infinitary methods in first-order logic; at the same time, some other infinitary methods in this logic exist; particularly, any release of the universal construction represents infinitary first-order methods transforming computably axiomatizable theories in finitely axiomatizable theories.

There are two possibilities to compare semantic layers.

2. Rule of inverse inclusion: Any relatively smaller class of methods defines the relatively larger semantic layer, i.e., if $\mathcal{M}_1$ and $\mathcal{M}_2$ are classes of transformation methods of theories, while L_1 and L_2 are the semantic layers determined by these classes, we have $\mathcal{M}_1 \subset \mathcal{M}_2 \Rightarrow L_1 \supseteq L_2$; furthermore, the union of classes of methods $\mathcal{M}_1 \cup \mathcal{M}_2$ determines the intersection of layers $L_1 \cap L_2$; the rule is formally exact.

3. Rule of representative check: We fix a large enough list $\mathcal{R}$ of commonly known model-theoretic properties, which is agreed to be considered as representative. For two semantic layers L_1 and L_2, $L_1 \overset{\mathcal{R}}{=} L_2$ means that $\mathfrak{p} \in L_1 \Leftrightarrow \mathfrak{p} \in L_2$ for all $\mathfrak{p} \in \mathcal{R}$, and $L_1 \overset{\mathcal{R}}{\subseteq} L_2$ means that $\mathfrak{p} \in L_1 \Rightarrow \mathfrak{p} \in L_2$ for all $\mathfrak{p} \in \mathcal{R}$; this rule represents a practical method of comparison even in the case when no possibility exists for formally exact comparison of volumes of the semantic layers; for $\mathcal{R}$, it are possible to take the join of collections of model-theoretic properties immediately listed in [9, Lem.4.2] and [10, Th.0.6.1].

Let us formulate an important relation between finitary and infinitary methods.

4. Principle of subordination of finite to infinite: If a class of transformation methods $\mathcal{M}$ is intended for definition of some version of infinitary layer, we must include in $\mathcal{M}$ all finitary methods relevant to this class; this requirement prevents unacceptable situation when an infinitary semantic layer is defined by a class of infinitary methods where some finitary methods are missed.

There is an obvious possibility to introduce the concept of *abstract infinite first-order combinatorics* as a version of infinite combinatorics with omitted requirement of computability for the passage from T to S, and thus, for the isomorphism $\mu: \mathcal{L}(T) \to \mathcal{L}(S)$. Since the class of abstract infinite methods is obviously wider in comparison with the class of computable infinite methods, by the rule of inverse inclusion, the semantic layer defined by computable infinite methods extends the layer defined by abstract infinite methods. This shows minor significance of the abstract approach and establishes computable infinite first-order combinatorics as the principal player in this direction of investigations.

VI. SEMANTIC LAYERS DEFINED BY COMBINATORICS

Now, we specify semantic layers, which are actual in this problematic.

We introduce the following notations:

 ISBN: 978-1-61208-277-6

$F2f\mathcal{L}$ = the set of all model-theoretic properties of algebraic type preserved by any $f2f$ signature reduction procedure,

$I2f\mathcal{L}$ = the set of all model-theoretic properties of algebraic type preserved by any $i2f$ signature reduction procedure $\cap F2f\mathcal{L}$,

$Uni\mathcal{L}$ = the set of all model-theoretic properties of model type preserved by any transformation of theories defined by the universal construction of finitely axiomatizable theories $\cap I2f\mathcal{L} \cap F2f\mathcal{L}$,

ACL = the set of all model-theoretic properties of algebraic type preserved by any Cartesian extension of any computably axiomatizable theory,

ADL = the set of all model-theoretic properties of algebraic type preserved by any Cartesian-quotient extension of any computably axiomatizable theory,

MDL = the set of all model-theoretic properties of model type preserved by any Cartesian-quotient extension of any computably axiomatizable theory,

MQL = the set of all model-theoretic properties of model type preserved by any quasiexact interpretation from a computably axiomatizable theory to another such theory $\cap MDL$,

$Fin\mathcal{L}$ = the set of all model-theoretic properties of algebraic type preserved by any finitary method between computably axiomatizable theories (an ideal concept),

$Inf\mathcal{L}$ = the set of all model-theoretic properties of model type preserved by any infinitary method between computably axiomatizable theories (an ideal concept).

For infinitary layers, intersections with finitary layers $\cap F2f\mathcal{L}$ and $\cap MDL$ are added for the sake of realization the requirement of subordination of finite methods to infinite ones, while $\cap I2f\mathcal{L} \cap F2f\mathcal{L}$ is added because the universal construction includes intermediate stages of types $i2f$ and $f2f$. By the rule of inverse inclusion, these intersections can be equivalently realized by adding corresponding methods in the definition. The class of quasiexact interpretations, [10, Ch.5], represents a technical framework for the universal construction, while currently, an advanced definition is available for this class (in forthcoming publication).

Semantic layer $Fin\mathcal{L}$ is said to be the *truly finitary layer*, while another layer $Inf\mathcal{L}$ is the *truly infinitary layer*. Currently, these definitions are just formal (presenting some ideal concepts), since we have not provided specifications to the set of all methods for the combinatorics. Nevertheless, one can believe that these two classes of methods must exist as mathematical objects.

The scheme in Fig. 1 shows all available model-theoretic inclusions between the layers we have defined, where the relation $L_1 \equiv L_2 \Leftrightarrow_{dfn} (L_1 \subseteq L_2 \,\&\, L_1 \overset{\mathcal{R}}{=\!=} L_2)$ is used presenting so called 'inclusion-almost-coincidence' relation. Two upper rows in the scheme represent layers of algebraic types, while its lower part represents layers of model type.

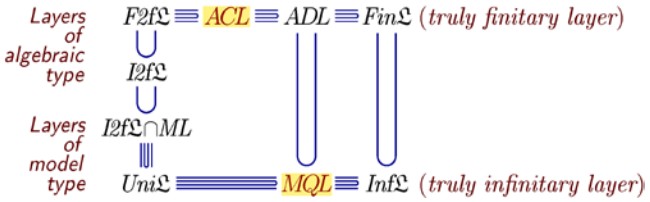

Figure 1. A dependence scheme between the semantic layers

5. Statement: *All inclusions and inclusions-almost-coincidences between the semantic layers presented in Fig. 1 take place*.

Justification. Most of the inclusions are checked immediately, using the rules of inverse inclusion and representative check. The inclusion $ACL \subseteq F2f\mathcal{L}$ is provided by Statement 1, while inclusions $MQL \subseteq ADL$ and $Inf\mathcal{L} \subset Fin\mathcal{L}$ are justified by the principle of subordination of finite layers to infinite. □

In Fig. 1, we have marked two particular semantic layers ACL and MQL. They play the role of *working* versions of the semantic layers for *finitary* and respectively *infinitary* combinatorics. It is important that the pointed out layers have complete definitions; moreover, they are most useful in applications. On the other hand, these two layers properly cover the truly finitary and respectively truly infinitary layer ensuring that practical applications with ACL and MQL are independent of investigations concerning approaches to definition of the truly semantic layers $Fin\mathcal{L}$ and $Inf\mathcal{L}$.

VII. Conclusion

Methods of finitary combinatorics represent simple and evident constructions in model theory. Methods of infinitary combinatorics are also often used. A key moment is that, each combinatorial method $\mathfrak{m}$ transforming T to S must define a computable isomorphism of the Tarski–Lindenbaum algebras $\mu: \mathcal{L}(T) \to \mathcal{L}(S)$. Operation of a Cartesian extension of the theory as well as other methods of finitary first-order combinatorics do not represent a great interest themselves, but they become an effective tool for investigations of the Tarski–Lindenbaum algebra of predicate calculi of finite rich signatures. However, the pointed out types of combinatorics were not provided with any strict definitions or even general agreements.

Regular references to the results known in the common practice are inappropriate within technically complicated fragments of reasoning; therefore, it is needed to introduce some formal basis for the concepts of finitary and infinitary combinatorics. This paper, providing a fundament to initial definitions concerning these combinatorics, represents a conceptual framework for the further investigations on expressive power of first-order predicate logic.

 ISBN: 978-1-61208-277-6

REFERENCES

[1] S. S. Goncharov and Yu. L. Ershov, Constructive models, Plenum, New York, 1999.

[2] W. Hanf, “Model-theoretic methods in the study of elementary logic,” Symposium on Theory of Models, North-Holland, Amsterdam, 1965, pp. 33-46.

[3] W. Hanf, “The Boolean algebra of Logic,” Bull. American Math. Soc., vol. 31, 1975, pp. 587-589.

[4] W. Hanf and D. Myers, “Boolean sentence algebras: Isomorphism constructions, ”J. Symbolic Logic, vol. 48, no. 2, 1983, pp. 329-338.

[5] W. Hodges, A shorter model theory, Cambridge University Press, Cambridge, 1997.

[6] D. Myers, “Lindenbaum–Tarski algebras,” Handbook of Boolean algebras, Ed: J. D. Monk, R. Bonnet, Elsevier Science Publishers, 1989, pp. 1167-1195.

[7] D. Myers, “An interpretive isomorphism between binary and ternary relations,” Structures in Logic and Computer Science: A Selection of Essays in Honor of Andrzej Ehrenfeucht, 1997, pp. 84-105.

[8] M. G. Peretyat'kin, “Semantic universal classes of models,” Algebra and Logic, 1991, vol. 30, no. 4, pp. 414-434.

[9] M. G. Peretyat'kin, “Semantic universality of theories over superlist,” Algebra and Logic, 1992, vol. 30, no. 5, pp. 517-539.

[10] M. G. Peretyat'kin, Finitely axiomatizable theories, Plenum, New York, 1997.

[11] W. Rautenberg, A concise introduction to mathematical logic, Textbook, Third Edition, Springer, 2010.

[12] H. J. Rogers. Theory of Recursive Functions and Effective Computability, McGraw-Hill Book Co., New York, 1967.

.

 ISBN: 978-1-61208-277-6

Using an Expression Interpreter to Reason With Partial Terms

Lev Naiman
Department of Computer Science
University of Toronto
Toronto, Canada
Email: naiman@cs.toronto.edu

***Abstract* — Refinements of programming specifications often include partial terms and need to be handled using formal rules. The idea of an expression interpreter over character strings is presented as a candidate solution. The interpreter allows for reasoning with partial terms without requiring a meta-logic or a logic with more than two values. We show how the interpreter can be used to create generic and compact laws, which also allow simple reasoning about expressions syntactically. We argue that it is simple to integrate the interpreter within existing theorem provers.**

Keywords* — *logic; partial-terms;expression interpreter; theorem prover; two-valued logic

I. Introduction

In programming specifications and their refinements we commonly encounter partial terms. Partial terms are defined as expressions that fail to denote a value. A term t in a theory T is partial if there are no laws in T that apply to t. An example is where a function or an operator is applied to an argument outside of its domain, such as $1/0$. We also say that a formula e is unclassified in theory T if it is neither classified as a theorem or an anti-theorem. Such expressions are present in proofs of programs due to the partial functions and operators that are often used in specifications. Borrowing an example from [1], we might implement the difference function as follows (where the domain of *diff* is integers, and the assumed theory is arithmetic and first-order two-valued logic).

$$\mathit{diff}\, i\, j = \mathbf{if}\ i = j\ \mathbf{then}\ 0\ \mathbf{else}\ (\mathit{diff}\, i\, (j+1)) + 1\ \mathbf{fi} \tag{1}$$

We would like to prove

$$\forall i, j : int \cdot i \geq j \Rightarrow (\mathit{diff}\, i\, j) = i - j \tag{2}$$

but when trying to simplify this expression instantiated with 1 and 2 respectively for i and j we get

$$\begin{aligned} 1 \geq 2 \quad &\Rightarrow \quad (\mathit{diff}\, 1\, 2) = 1 - 2 \\ = \ \mathbf{F} \quad &\Rightarrow \quad (\mathit{diff}\, 1\, 2) = -1 \end{aligned} \tag{3}$$

and we cannot apply any laws at this point to simplify it further. A law would allow simplifying the expression to true, but it requires that both operands be boolean. The expression *diff* 1 2 is a partial term because no laws apply to it. For this reason we cannot use any law to conclude that $(\mathit{diff}\, 1\, 2) = -1$ is a boolean, even though it has the form $X = Y$. Tools that reason with such expressions must be based on formal rules in order to have confidence in their proofs. We propose a character-string interpreter to solve this problem.

The rest of the paper is organized as follows: in section II we examine the existing approaches in the literature to cope with partial terms. In section III we describe the background theories we use to define the interpreter in IV. Section V shows how the interpreter can be used to cope with partial terms. Section VI describes other benefits of the interpreter when constructing theories. Section VII describes how we can extend the definition of the interpreter to be more expressive.

II. Current Approaches to Partial Terms

One approach to resolve partial terms is to make all terms denote. Formally this means that for each partial term such as $x/0$, a law must exist saying which set of values that expression is a member of. This set of values is assumed to already be defined in the logic, as opposed to newly created values. In this case there could be a law defined saying that $\forall x : int \cdot x/0 : int$. This is the approach used in the programming theory of [2]. Such laws do not explicitly say what value a partial term is equal to, and this can cause certain peculiar and possibly unwanted results such as $0/0 = 0$ being a theorem.

$$\begin{aligned} & 0 \\ &= 0 \times (1/0) \\ &= 1 \times (0/0) \\ &= 0/0 \end{aligned} \tag{4}$$

This approach can be slightly modified and the value of partial terms can be fixed. However, this might cause some unwanted properties. In the case of division by zero a choice of 42 as used in [3] cannot be allowed due to inconsistency.

In [4] the authors point out that underspecification alone may cause problems. If we allow domains of single elements then these problems can go as far as inconsistency. The semantic model of our interpreter uses underspecification, but not exclusively. In some cases, similarly to LPF, the interpreter would leave some expressions unclassified. One way of finding a model for partial functions in set theory is the standard approach of mapping any unmapped element from the domain to a special value, usually called $\perp$ [5]. The denotational semantics for a generic law for equality are extended with

 ISBN: 978-1-61208-277-6

TABLE I
THREE-VALUED BOOLEAN OPERATORS

	T	**F**	$\perp$
$\neg$	**F**	**T**	$\perp$

	TT	**TF**	**FT**	**FF**	**T**$\perp$	$\perp$**T**	$\perp$**F**	**F**$\perp$	$\perp\perp$
$\vee$	**T**	**T**	**T**	**F**	**T**	**T**	$\perp$	$\perp$	$\perp$
$\wedge$	**T**	**F**	**F**	**F**	$\perp$	$\perp$	**F**	**F**	$\perp$

this value, and in this particular model $7/0 = 5/0$ would be a theorem (assuming strict equality). However, a user of a logic that includes the interpreter would not need to perform any calculations that concern this extra value.

The logic of partial terms (LPT) [6], [7] is an example of a logic that does not include the undefined constant. It does however include a definedness operator $\downarrow$. In this theory the specialization law $(\forall x \cdot A(x)) \Rightarrow A(v)$ requires that v be defined. The basic logic of partial terms (BPT) [8] is a modification of LPT, and relaxes the previous requirement for some laws. It allows for reasoning with non-terminating functional programs. Some logics such as [9] inlcude multiple notions of equality to be used in calculations. This may complicate the laws of quantifiers.

Another approach to deal with partial terms is a non-classical logic such as LPF [3] with more than two values. In these logics the truth table of boolean operators is usually extended as in table I (where $\perp$ represents an "undefined" value, and the column heads are both of the arguments to the operator). In this logic the expression $0/0 = 1$ would not be classified to one of the boolean values, but would rather be classified as $\perp$. Undefinedness is either resolved by the boolean operators or is carried up the tree of the expression. Some three valued logics have a distinct undefined value for each value domain, such as integers and booleans.

Three and more valued logics have varied useful applications. However, a drawback of using a logic with multiple truth values is that certain useful boolean laws no longer hold. This is particularly true of the law of the excluded middle, $\forall x : bool \cdot x \vee \neg x$, which in a three value logic can be modified to $\forall x : bool \cdot x \vee \neg x \vee undefined(x)$. In the Logic of Computable Functions (LCF) [10] there is a $\perp_t$ value for each type t, requiring the modification of several laws. Another issue of multiple valued logics is that not knowing the value of an expression seems to be pushed one level up; attempting to formalize these extra values will result in a semantic gap. There are always expressions that must remain unclassified for a theory to remain consistent.

A further method of dealing with partial terms is conditional, or short-circuit operators [11]. This approach is similar to those logics with three values, since it gives special treatment to partial terms. Boolean operators have an analogous syntax $a \,\mathbf{cor}\, b$, $a \,\mathbf{cand}\, b$, $a \,\mathbf{cimp}\, b$, etc. In these expressions if the first value is undefined, then the whole expression is undefined. These conditional operators are not commutative.

III. BACKGROUND THEORIES

We introduce two theories from [12] that we will use to define the interpreter.

A. Bunch Theory

A bunch is a collection of objects. It is different from a set, which is a collection of objects in a package. A bunch is instead just those objects, and a bunch of a single element is just the element itself. Every expression is a bunch, but not all bunches are elementary. Here are two bunch operators.

$$A \,, B \qquad \text{A union B} \quad (5)$$

$$A : B \qquad \text{A in B, or A included in B} \quad (6)$$

Operators such as a comma, colon, and equality apply to whole bunches, but some operators apply to their elements instead. In other words, they distribute over bunch union. For example

$$\begin{aligned} & 1 + (4, 7) \\ &= 1 + 4 \ , \ 1 + 7 \\ &= 5 \,, 8 \end{aligned} \quad (7)$$

Bunch distribution is similar to a cross-product in set theory.

B. String Theory

A string is an indexed collection of objects. It is different from a list or ordered pair, which are indexed collections of objects in a package. A string of a single item is just that item. The simplest string is the empty string, called nil. Strings are joined together, or concatenated with the semicolon operator to form larger strings. This operator is associative but not commutative. The string $0; 1$ has zero as the first item and one as the second. For a natural number n and a string S, $n * S$ means n copies of S. Let nat be the bunch of natural numbers. The copies operator is defined as follows.

$$0 * S = nil \quad (8)$$

$$\forall\, n : nat \cdot (n + 1) * S = n * S; S \quad (9)$$

Strings can be indexed, and their length can be obtained with the length operator ($\leftrightarrow$).

$$S_n \qquad \text{S at index n} \quad (10)$$

$$\leftrightarrow S \qquad \text{length of S} \quad (11)$$

A semicolon distributes over bunch union, as so does an asterisk in its left operand. Some examples of the operators defined are

$$\begin{aligned} \leftrightarrow (7; 1; 0) &= 3 \\ (7; 1; 0)_0 &= 7 \\ 1; (5, 17); 0 &= (1; 5; 0), (1; 17; 0) \\ 3 * (0; 1) &= 0; 1; 0; 1; 0; 1 \\ (0, 1) * (0; 1) &= 0 * (0; 1), 1 * (0; 1) = nil, 0; 1 \end{aligned} \quad (12)$$

The prefix copies operator $*S$ is defined to mean $nat * S$, or informally the bunch of any number of copies of S. Finally,

 ISBN: 978-1-61208-277-6

we introduce characters, which we write with double quote marks such as "a", "b", etc. To include the open and close double-quote characters we escape them with a backslash: "$\backslash$"". Strings that contain exclusively character strings are sometimes abbreviated with a single pair of quotes: "abc" is short for "a"; "b"; "c". If the bunch of all characters is called $char$, then the bunch of all two-character strings is $char; char$.

Bunch and string theory are used because they allow for compact language definitions. For example, denoting the collection of naturals greater than zero in set theory can be done by writing $\{n : nat | n > 0\}$. In bunch theory it can be written as $nat + 1$. We can of course define an addition operator that distributes over the contents of a set, but the benefit of bunch theory (and analogously string theory) is that no such duplication is necessary.

IV. Defining the Interpreter

We would like a simple method to reason with partial terms that introduces as few new operators as possible, and which preserves the properties of existing operators. We would also like to avoid a separate meta-language, and to do all reasoning within a single logic. In the literature authors often use one set of symbols for the meta-logic operators and another for the object logic. We use character strings instead both for clarity, and in the case where we wish to use the logic to study itself. We take the idea of the character-string predicate of Hehner [13], and we expand it to be a general interpreter for any expression in our language. To maintain consistency we exclude the interpreter itself from the interpreted language. The interpreter, which we call $\mathcal{I}$ is an operator which applies to character strings and produces an expression. The interpreter can be thought of as unquoting a string. We first define our language as a bunch of character strings.

Let *char* be the bunch of all character symbols, let *alpha* be the bunch of character symbols in the English alphabet, and let *digit* be the bunch of digit character. We define our language *lang* to be the following bunch of strings.

$$alpha; *alpha; *\text{“}\text{'}\text{”} = var \tag{13}$$
$$digit; *digit = num$$
$$binops = \text{“}\wedge\text{”}, \text{“}\vee\text{”}, \text{“}=\text{”}, \text{“}\Rightarrow\text{”}, \text{“}\Leftarrow\text{”}, \text{“};\text{”}, \text{“}-\text{”}, \text{“ ”}$$
$$var, num, \text{“}\mathbf{T}\text{”}, \text{“}\mathbf{F}\text{”} : lang$$
$$\text{“}\langle\text{”}; var; \text{“}:\text{”}; lang; \text{“}\rightarrow\text{”}; lang; \text{“}\rangle\text{”} : lang$$
$$\text{“(”}; lang; \text{“)”} : lang$$
$$lang; binops; lang : lang$$
$$\text{“}\neg\text{”}, \text{“}-\text{”}, \text{“}\forall\text{”}, \text{“}\exists\text{”}; lang : lang$$
$$\text{“}\backslash\text{“”}; *char; \text{“}\backslash\text{””} : lang$$

Here we have defined a language that includes boolean algebra, numbers, logical quantifiers, functions, and strings. The language is defined similarly to how a grammar for a language would be given. Function syntax is $\langle v : D \rightarrow B\rangle$, where the angle brackets denote the scope of the function, and v is the introduced variable of type D. We treat quantifiers as operators that apply to functions. The quantifiers $\exists$ and $\forall$ give boolean results. When we use more standard notation such as $\forall v : domain \cdot body$ we mean it as an abbreviation for $\forall\langle v : D \rightarrow B\rangle$.

The interpreter is intuitively similar to a program interpreter: it turns passive data into active code. Our interpreter turns a text that represents an expression into the expression itself. The interpreter is defined very closely to how $lang$ was defined. The laws are as follows.

$$\mathcal{I}\text{“}\mathbf{T}\text{”} = \mathbf{T} \tag{14}$$
$$\mathcal{I}\text{“}\mathbf{F}\text{”} = \mathbf{F}$$
$$\forall\, s : (digit; *digit) \cdot \forall\, d : digit \cdot \mathcal{I}(s; d) = (\mathcal{I}s) \times 10 + (\mathcal{I}d)$$
$$\forall\, dom, body : lang \cdot$$
$$\mathcal{I}(\text{“}\langle a : \text{”}; dom; \text{“}\rightarrow\text{”}; body; \text{“}\rangle\text{”}) = \langle a : \mathcal{I}dom \rightarrow \mathcal{I}body\rangle$$
$$\forall\, s : lang \cdot \mathcal{I}\text{“(”}; s; \text{“)”} = \mathcal{I}s \quad \wedge$$
$$\mathcal{I}(\text{“}\neg\text{”}; \text{“(”}; s; \text{“)”}) = \neg(\mathcal{I}s) \quad \wedge$$
$$\mathcal{I}(\text{“}\forall\text{”}; \text{“(”}; s; \text{“)”}) = \forall(\mathcal{I}s) \quad \wedge$$
$$\mathcal{I}(\text{“}\exists\text{”}; \text{“(”}; s; \text{“)”}) = \exists(\mathcal{I}s) \quad \wedge$$
$$\mathcal{I}(\text{“}-\text{”}; \text{“(”}; s; \text{“)”}) = -(\mathcal{I}s)$$
$$\forall s, t : lang \cdot \mathcal{I}(\text{“(”}; s; \text{“)”}; \text{“}\wedge\text{”}; \text{“(”}; t; \text{“)”}) = (\mathcal{I}s) \wedge (\mathcal{I}t) \quad \wedge$$
$$\mathcal{I}(\text{“(”}; s; \text{“)”}; \text{“}=\text{”}; \text{“(”}; t; \text{“)”}) = (\mathcal{I}s) = (\mathcal{I}t) \quad \wedge$$
$$\mathcal{I}(\text{“(”}; s; \text{“)”}; \text{“}\Rightarrow\text{”}; \text{“(”}; t; \text{“)”}) = (\mathcal{I}s) \Rightarrow (\mathcal{I}t) \wedge$$
$$\mathcal{I}(\text{“(”}; s; \text{“)”}; \text{“}\Leftarrow\text{”}; \text{“(”}; t; \text{“)”}) = (\mathcal{I}s) \Leftarrow (\mathcal{I}t) \wedge$$
$$\mathcal{I}(\text{“(”}; s; \text{“)”}; \text{“;”}; \text{“(”}; t; \text{“)”}) = (\mathcal{I}s); (\mathcal{I}t) \quad \wedge$$
$$\mathcal{I}(\text{“(”}; s; \text{“)”}; \text{“}-\text{”}; \text{“(”}; t; \text{“)”}) = (\mathcal{I}s) - (\mathcal{I}t) \quad \wedge$$
$$\mathcal{I}(\text{“(”}; s; \text{“)”}; \text{“ ”}; \text{“(”}; t; \text{“)”}) = (\mathcal{I}s)\ \ (\mathcal{I}t)$$
$$\forall\, s : *char \cdot \mathcal{I}(\text{“}\backslash\text{“”}; s; \text{“}\backslash\text{””}) = s$$

To save space we leave out the interpretation of each digit. For scopes the introduced variable must be an identifier, and the expression $\mathcal{I}a$ in that position would not satisfy this requirement. We instead have a law for only the identifier a, and other identifiers can be obtained through an application of a renaming law. We add character brackets to these laws in order to avoid precedence issues.

Note that we defined $lang$ as a bunch of texts, and not the expressions themselves. When these texts are interpreted, the results are expressions or values in the language. The text "2" is in $lang$, but not the value 2. The interpreter is similar to a function of strings and distributes over bunch union. It is of course possible to have a logical language to parallel the texts in $lang$; all the expressions in the language which do not contain $\mathcal{I}$ can then be denoted as $\mathcal{I}\, lang$. In this paper we leave out some operators from $lang$, such as the ones in bunch theory.

The interpreter is similar to a traditional semantic valuation function, with a few differences. First, the interpreter is a way of encoding meta-logic within the logic itself; no extra meta-logic is required. Second, the interpreter does not necessarily map every string in the language to a value. Rather, we later introduce generic laws that reason with such expressions directly. Lastly, we will show how the interpreter can be included in the interpreted language without inconsistency.

 ISBN: 978-1-61208-277-6

A. Variables

One significant change that we allow in our logic is for variables. The interpreter needs to refer to an infinite collection of strings that represent variables, and there is no simple way to refer to all variables themselves. We would like the interpretation of a string representing a variable to be the variable that is represented:

$$\begin{aligned} &\mathcal{I}\text{“}a\text{”} = a \\ &\mathcal{I}\text{“}b\text{”} = b \\ &\cdots \end{aligned} \tag{15}$$

We instead say that a variable with the name a is an abbreviation for $\mathcal{I}$“a”, and similarly for all other variable names. Although in our initial definition we excluded the interpreter from the interpreted strings, we later show in VII how we can extend our language to safely include the interpreter.

There is an important consequence of making variable syntax more expressive: function application and variable instantiation is no longer a decidable procedure in general. This is because deciding whether two variable strings are equal is now as difficult as all of proving. However, this does not pose a problem for the implementation of function application along with the interpreter in a theorem prover. The simple solution is that whenever we see an interpreter in the body, we simply do not apply the function. We argue that this rarely hinders the use of the interpreter, since users can do all calculation in the sub-language that does not include the interpreter, exactly as before. In the case where reasoning with the interpreter is desired, standard proof obligations can be generated and discharged.

We finish this section by noting that we could have simplified the definition considerably if we had a fully parenthesized prefix language. All operator interpretation could be compressed to a single law, and some bracket characters removed.

V. Resolving Partial Terms With the Interpreter

Our solution to reasoning with partial terms is neither at the term or propositional level. We rather say that some operators, such as equality or bunch inclusion are generic. For example, here are two of the generic laws for equality.

$$\forall a, b : lang \cdot \mathcal{I}\,(a; \text{“}=\text{”}; b) : bool \qquad \text{Boolean Equality} \tag{16}$$

$$\forall a : \mathcal{I}\,lang \cdot a = a \qquad \text{Reflexivity} \tag{17}$$

The first law says that any equality is a boolean expression, similarly to the Excluded Fourth Law in LPF which implies an equality is either true, false or undefined [1]. The arguments can be any expressions in the interpreted language. For a simple formal example of the use of the law we continue with the difference example.

(18)

$$\begin{aligned} &\mathbf{F} \Rightarrow \mathit{diff}\,1\,2 = -1 && \text{Bool Base Law} \\ &\quad \text{Type Checking Proof Obligation} \\ &\quad (\mathit{diff}\,1\,2 = -1) : \mathit{bool} && \text{Interpreter laws} \\ &\quad = \mathcal{I}\text{“}\mathit{diff}\,1\,2 = -1\text{”} : \mathit{bool} && \text{String Assoc.} \\ &\quad = \mathcal{I}(\text{“}\mathit{diff}\,1\,2\text{”}; \text{“}=\text{”}; \text{“}-1\text{”}) : \mathit{bool} && \text{Bool Equality} \\ &\quad = \mathbf{T} \\ &= \mathbf{T} \end{aligned}$$

As we can see in the example, since the interpreter unquotes expressions, using it in proofs is usually just the reverse process.

A. Implementation

In general, implementing laws that use the interpreter in a theorem prover is non-trivial. This is because it is difficult to determine if unification alone is sufficient to check if a law applies. We deliberately wrote two equality laws differently to illustrate a couple cases where this task can be made easy. If the only place the interpreter appears in a law is the expression $\mathcal{I}\,lang$ in the domain of a variable, it can be treated as a generic type. Type checking can be done by scanning to see that the interpreter does not appear in any instantiated expression with a generic type. In the case of the second law, instantiating the variables and parsing yields a valid expression without any further computation.

VI. Metalogical Reasoning Within the Logic

There are several benefits of defining the interpreter and using it to create laws. One such benefit is the creation of generic laws, where type-checking for variables is not necessary. The removal of type-checking is not only beneficial for simplicity, partiality, and efficiency, but some operators are meant to be truly generic. For example, the left operand of the set-membership operator ($\in$) can be any expression in the language, and set brackets can be placed around any expression. By including the interpreter in the logic these laws are expressed with full formality. For sets, an example would be

$$\forall A, B : \mathcal{I}\,lang \cdot (\{A\} = \{B\}) = (A = B) \tag{19}$$

Another benefit is compact laws. For example, we wish to define a generic symmetry law for natural arithmetic in our logic. If we had a prefix notation then we could have written it like this.

$$\forall f : (+, \times, =) \cdot \forall a, b : nat \cdot = (f\,a\,b)(f\,b\,a) \tag{20}$$

Using the interpreter we can create a law in a similar fashion for non-prefix notation.

$$\begin{aligned} &\forall f : \text{“}+\text{”}, \text{“}\times\text{”}, \text{“}=\text{”} \cdot \forall a, b : lang \cdot \\ &\qquad \mathcal{I}\,(a, b) : nat \Rightarrow \mathcal{I}\,(a; f; b) = \mathcal{I}\,(b; f; a) \end{aligned} \tag{21}$$

 ISBN: 978-1-61208-277-6

This law can be made completely generic and include more than arithmetic operators. It even becomes simpler to write.

$$\forall f : \text{“}+\text{”}, \text{“}\times\text{”}, \text{“}\wedge\text{”}, \text{“}\vee\text{”}, \text{“}=\text{”} \cdot \forall a, b : lang \cdot \quad (22)$$
$$\mathcal{I}\,(a; f; b) = \mathcal{I}\,(b; f; a) \quad (23)$$

These sorts of laws allow us to capture an idea like associativity or commutativity in a compact way, and can be easily extended by concatenating to the operator text.

One of the most useful features of the interpreter is reasoning about the syntactic structure of an expression without requiring a meta-logic. These laws include function application and several programming laws. Some laws have caveats, such as requiring that in some expressions certain variables or operators do not appear. For example, there is a quantifier law for $\forall$ that says if the variable a does not appear free in P.

$$(\forall a : D \cdot P) = P \quad (24)$$

We would like to formalize this caveat. It is straight forward to write a program that checks variable or operator appearance in a string (respecting scope). We formalize a specification of the “no free variable” requirement using the interpreter. For simplicity, assume that variables are single characters, and strings are not in the interpreted language. For a string P in our language and a variable named a we specify

$$\begin{aligned}&\exists i : (0.. \leftrightarrow P) \cdot P_i = \text{“}a\text{”} \quad \wedge \quad (25)\\ &\quad \neg\exists s, t, D, pre, post : *char \cdot\\ &\qquad (pre; \text{“}\langle a : \text{”}; D; \text{“} \rightarrow \text{”}; s; P_i; t; \text{“}\rangle\text{”}; post) = P\\ &\qquad \vee\, (pre; \text{“}\langle\text{”}; P_i; D; \text{“} \rightarrow \text{”}; s; \text{“}\rangle\text{”}; post) = P\end{aligned}$$

This specification says that a is free in P. The first part says that there is an index i in P at which a appears. The second part says that a is not local. Let $free$ denote this specification parameterized for an expression and a variable; $free$ “a” P says that a is free in P. The caveat for the quantifier law is formalized as

$$\neg(free\ \text{“}a\text{”}\ P) \quad \Rightarrow \quad \mathcal{I}\,(\text{“}\forall a : \text{”}; D; \text{“} \cdot \text{”}; P) = \mathcal{I}\,P \quad (26)$$

In a similar manner we can avoid including axiom schemas in some theories and have just a single axiom. The notation allows us to refer to all variables in an expression.

VII. Including the Interpreter

So far we have excluded the interpreter itself from the interpreted language to maintain consistency. Gödel’s First Incompleteness Theorem implies that we could never define our interpreter to be both consistent and complete [14], [15]. Let the § symbol denote bunch comprehension, and be treated as a quantifier; when applied to a function it returns a bunch. Then there are expressions such as $\{\S x : \mathcal{I}\,lang \cdot \neg(x \in x)\}$ whose string representation we cannot interpret (interpreting this expression in particular causes Russell’s paradox). A simpler proof of Gödel’s theorem by [13] shows why a straight-forward inclusion of the interpreter is inconsistent. However, as [13] also suggests, any logic can be completely described by another. This point is intuitively manifested in the fact that all expressions that cannot be interpreted include the interpreter itself. In a sense, we relegate all issues of partiality in our logic to involve only the interpreter.

However, we can weaken the restriction on the interpreter being excluded from the language. The motivation for including the interpreter is to reason about languages that allow this sort of self reference. In practice, theorem provers such Coq [16] allow reflection as a proving technique. We would like to use the interpreter as a simple way of reasoning about termination and consistency of definitions. The key insight is that a mathematical function disregards computation time. The domain $xnat$ is the naturals extended with ∞. We can measure computation time recursively by defining the following timing function.

$$\begin{aligned}&\mathcal{T} = \langle s : lang \rightarrow \mathbf{result}\, r : xnat \cdot \quad (27)\\ &\quad \mathbf{var}\, f : (char \rightarrow bool) := \langle t : lang \rightarrow \text{“}\backslash\text{“”}; t; \text{“}\backslash\text{””} = s\rangle \cdot\\ &\quad \mathbf{if}\, \exists f\, \mathbf{then}\, r := 1\, \mathbf{else}\\ &\quad \mathbf{var}\, f : (lang \rightarrow bool) := \langle t : lang \rightarrow \text{“}\neg\text{”}; t = s\rangle \cdot\\ &\quad \mathbf{if}\, \exists f\, \mathbf{then}\, r := 1 + \mathcal{T}(\S f)\, \mathbf{else}\\ &\quad \mathbf{var}\, f : (lang \rightarrow lang \rightarrow bool) :=\\ &\qquad \langle t, t' : lang \rightarrow t; \text{“}\wedge\text{”}; t' = s\rangle \cdot\\ &\quad \mathbf{if}\, \exists f\, \mathbf{then}\, r := 1 + \mathcal{T}(\S t : lang \cdot \exists t' : lang \cdot f\, t\, t')\\ &\qquad + \mathcal{T}(\S t' : lang \cdot \exists t : lang \cdot f\, t\, t')\, \mathbf{else}\\ &\quad \vdots\\ &\quad \mathbf{var}\, f : (lang \rightarrow bool) := \langle t : lang \rightarrow \text{“}\mathcal{I}\text{”}; t = s\rangle \cdot\\ &\quad \mathbf{if}\, \exists f\, \mathbf{then}\, r := 1 + \mathcal{T}(\mathbf{if}\, \S f : var \wedge \mathcal{I}(\S f) : lang\\ &\qquad \mathbf{then}\, \mathcal{I}(\S f)\, \mathbf{else}\, \S f\, \mathbf{fi})\, \mathbf{else}\\ &\qquad r := \infty\\ &\quad \mathbf{fi}\\ &\quad \rangle\end{aligned}$$

This function is in a way parallel to how an interpretation works, except that it counts time. The time in question is the number of law applications needed to simplify an expression to have no interpreter symbol in it. At each if-statement the function checks for the occurrence of a certain piece of syntax, and the vertical ellipsis would include a similar check for the rest of the syntax. The special part of this function is when we see the interpreter symbol. If the interpreter was applied to a string representing a variable, and that variable’s value is a string in the language, we recurse on its value. If the interpreter is applied to any other expression, we recurse on

 ISBN: 978-1-61208-277-6

that expression's string representation. For example, if we have

$$Q = \text{“}\neg\mathcal{I}Q\text{”} \tag{28}$$

then we calculate

$$\begin{aligned} &\mathcal{T}Q \\ &= \mathcal{T}\text{“}\neg\mathcal{I}Q\text{”} \\ &= 1 + \mathcal{T}\text{“}\mathcal{I}Q\text{”} \\ &= 1 + \mathcal{T}Q \end{aligned} \tag{29}$$

and therefore $\mathcal{T}Q = \infty$ since $\mathcal{T}Q : xnat$. For any string that does not include the interpreter the time is linear; this can be proven by structural induction over *lang* if we add an induction axiom along with the construction axioms we defined earlier. We should only interpret an expression that includes the interpreter if the execution time of the interpretation is finite. If it is infinite or cannot be determined, then there is a potential for inconsistency had we decided to interpret it regardless. We can add the interpreter to the interpreted language as follows:

$$\forall s : lang \cdot \mathcal{T}s < \infty \Rightarrow \mathcal{I}\text{“}\mathcal{I}\backslash\text{“”}; s; \text{“}\backslash\text{””} = \mathcal{I}s \tag{30}$$

This also implies that interpreting variables in their unabbreviated form is also safe. If we have $s : var$ then:

$$\begin{aligned} &\mathcal{T}\text{“}\mathcal{I}\text{”}; s \\ &= 1 + \mathcal{T}s \\ &= 1 + 1 \\ &= 2 \end{aligned} \tag{31}$$

In general, proving a finite execution time is the halting problem. When reasoning about logics it may be useful to include the interpreter in the interpreted language. For many practical purposes it can be left out.

VIII. Proof of Consistency

To prove the interpreter consistent we will find a model in set theory. Characters are implemented as natural numbers, having

$$\text{“0”} = 0,\ \ldots\ \text{“9”} = 9,\ \text{“}a\text{”} = 10,\ \ldots\ \text{“}z\text{”} = 25,\ \ldots \tag{32}$$

Strings are implemented as ordered pairs in the standard way.

$$a; b = \{\{a\}, \{a, b\}\} \tag{33}$$

The interpreter is a mapping from the set of all strings in our language $lang$ to the class of all sets. $\mathcal{I} \subseteq lang \times Sets$. It is assumed that all other theories (functions, boolean algebra, numbers) are implemented in set theory in the standard way. For this reason partial functions might be implemented using another special value that all remaining domain elements will be mapped to. We will not delve into the implementation of functions and other theories, since once they are implemented in set theory, they are included in the class $Sets$.

We must prove that there exists a function $\mathcal{I}$ such that the interpreter axioms are true. The recursion theorem will be used to prove this [5]. The theorem states that given a set X, an element a of X, and a function $f : X \to X$ there exists a unique function F such that

$$F\,0 = a \tag{34}$$

$$\forall n : nat \cdot F\,(n+1) = f(F\,n) \tag{35}$$

Since $\mathcal{I} \subseteq lang \times Sets$ it is necessary to first find a function from $lang$ to the naturals; this is an enumeration of the strings in $lang$. Let $charNum$ be the total number of characters in $char$. Character string comparison for strings s, t is defined as

$$(s > t) = strNum(s) > strNum(t) \tag{36}$$

$$\begin{aligned} strNum = \langle S : *char \to \textbf{if}\ S = nil\ \textbf{then}\ 0 \\ \textbf{else}\ S_0 + charNum \times S_{1..\leftrightarrow S}\ \textbf{fi}\rangle \end{aligned} \tag{37}$$

The enumeration function $enum$ of strings in $lang$ is defined as

$$enum = (g^{-1}) \tag{38}$$

$$\begin{aligned} g = \langle n : nat \to \textbf{if}\ n = 0\ \textbf{then}\ (MIN\ s : lang \cdot s) \\ \textbf{else}\ (MIN\ s : (\S t : lang \cdot t > g(n-1)) \cdot s)\ \textbf{fi}\rangle \end{aligned} \tag{39}$$

The function $strNum$ assigns a unique number to each string. Some character strings are not in $lang$, and we desire an enumeration free from gaps. The function g assigns a unique string in $lang$ to each natural as follows: zero is mapped to the first string in the language, and each subsequent number is mapped to the next smallest string. Since g is one-to-one, we define $enum$ as its inverse. We define function F for a given state in the model with finite single-character variables as

$$F\,0 = \{0; 0\} \tag{40}$$

$$\vdots$$

$$F\,9 = \{9; 9\} \cup F\,8$$

$$\begin{aligned} &\text{For all } s : char \text{ let } m = enum(\text{“}\backslash\text{“”}; s; \text{“}\backslash\text{””}) \text{ in} \\ &F\,m = \{m; s\} \cup F\,(m-1) \end{aligned} \tag{41}$$

$$\begin{aligned} &F\,(n+1) = \{(n+1); (H\,(n+1)\,(F\,n))\} \cup F\,n \\ &(H \text{ is defined below}) \end{aligned} \tag{42}$$

At each argument n function F is a mapping of all previous numbers to their corresponding expressions, in addition to the current one. The base elements are the variables, numbers and strings. Function H constructs expressions using the operators in our language from previous expressions. It is defined as

follows.

$$
\begin{aligned}
&H\,k\,I = \qquad (43)\\
&\{S : Sets | \exists n, m : \mathbf{dom}(I) \cdot g\,k = (g\,n); \text{“}+\text{”}; (g\,m)\\
&\qquad\qquad \wedge S = I\,n + I\,m\} \cup\\
&\{S : Sets | \exists n, m : \mathbf{dom}(I) \cdot g\,k = (g\,n); \text{“}\wedge\text{”}; (g\,m)\\
&\qquad\qquad \wedge S = I\,n \wedge I\,m\} \cup\\
&\{S : Sets | \exists n : \mathbf{dom}(I) \cdot g\,k = \text{“}\neg\text{”}; (g\,n) \wedge S = \neg I\,n\} \cup\\
&\vdots
\end{aligned}
$$

Like the timing function, the vertical ellipsis represents a similar treatment for other operators and is used to save space. Since g is one-to-one, only a single set in this union will have an element in it. In other words, each number is mapped to a single expression (but not vice-versa). Finally, the interpreter is implemented as follows.

$$\mathcal{I} = \langle s : lang \to (F\ (enum\,s))\,(enum\,s)\rangle \qquad (44)$$

IX. Conclusion

We have presented the formalism of an expression interpreter for the purpose of reasoning with partial terms. Our technique requires no separate meta-logic, and we believe that our encoding of expressions as character strings is simple and transparent. The use of the interpreter allows proofs with partial terms to proceed in a fully formal fashion classically; that is, with just the standard boolean algebra. We show how the interpreter can be used to create generic and compact laws, which also allow syntactic reasoning about expressions. We also argue that the incorporation of the interpreter in theorem provers is simple, since the parsing that is required for its use is an efficient linear-time algorithm.

References

[1] C. B. Jones and C. A. Middelburg, “A typed logic of partial functions reconstructed classically,” *ACTA*, vol. 31, no. 5, pp. 399–430, 1994.

[2] C. C. Morgan, *Programming from specifications, 2nd Edition*. Upper Saddle River, NJ, USA: Prentice Hall, 1994.

[3] C. B. Jones, M. J. Lovert, and L. J. Steggles, “A semantic analysis of logics that cope with partial terms,” in *ABZ*, ser. LNCS, J. Derrick, J. A. Fitzgerald, S. Gnesi, S. Khurshid, M. Leuschel, S. Reeves, and E. Riccobene, Eds., vol. 7316. Springer, 2012, pp. 252–265.

[4] C. B. Jones, “Partial functions and logics: A warning,” *IPL*, vol. 54, no. 2, pp. 65–67, 1995.

[5] W. Just and M. Weese, *Discovering Modern Set Theory. I*. American Mathematical Society, 1996, vol. 8.

[6] M. Beeson, *Foundations of Constructive Mathematics*. New York, NY, USA: Springer-Verlag, 1985.

[7] ——, “Lambda logic,” in *Automated Reasoning: Second International Joint Conference, IJCAR 2004*. Springer, 2004, pp. 4–8.

[8] R. F. Stärk, “Why the constant ’undefined’? logics of partial terms for strict and non-strict functional programming languages,” *J. Funct. Program.*, vol. 8, no. 2, pp. 97–129, 1998.

[9] R. D. Gumb, “The lazy logic of partial terms,” *JSYML*, vol. 67, no. 3, pp. 1065–1077, 2002.

[10] M. J. C. Gordon, R. Milner, and C. P. Wadsworth, *Edinburgh LCF*, ser. Lecture Notes in Computer Science. Springer, 1979, vol. 78.

[11] D. Gries, *The Science of Programming*. New York: Springer-Verlang, 1981.

[12] E. C. R. Hehner, *A Practical Theory of Programming*. New York: Springer, 1993. [Online]. Available: http://www.cs.toronto.edu/ hehner/aPToP/

[13] ——, “Beautifying gödel,” pp. 163–172, 1990.

[14] K. Gödel, “Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme,” *Monatshefte für Mathematik und Physik*, vol. 38, no. 1, pp. 173–198, 1931.

[15] R. Zach, “Kurt gödel and computability theory,” in *Logical Approaches to Computational Barriers*, ser. Lecture Notes in Computer Science, A. Beckmann, U. Berger, B. Löwe, and J. Tucker, Eds. Springer Berlin Heidelberg, 2006, vol. 3988, pp. 575–583.

[16] T. C. D. Team, “The coq proof assistant reference manual,” 2009.

 ISBN: 978-1-61208-277-6

Reducing Higher Order π-Calculus to Spatial Logics

Zining Cao[1,2]

[1]Department of Computer Science and Technology
Nanjing University of Aeronautics & Astronautics
Nanjing 210016, China
[2]State Key Laboratory for Civil Aircraft Flight Simulation
Shanghai Aircraft Design and Research Institute
Shanghai 201210, China

***Abstract*—In this paper, we show that the theory of processes can be reduced to the theory of spatial logic. Firstly, we propose a spatial logic SL for higher order π-calculus, and give an inference system of SL. The soundness and incompleteness of SL are proved. Furthermore, we show that the structure congruence relation and one-step transition relation can be described as the logical relation of SL formulas. At last we extend all definitions and results of SL to a weak semantics version of SL, called WL.**

***Keywords*—higher order π-calculus; spatial logic; inference system**

I. INTRODUCTION

Higher order π-calculus was proposed and studied intensively in Sangiorgi's dissertation [29]. In higher order π-calculus, processes and abstractions over processes of arbitrarily high order, can be communicated. Some interesting equivalences for higher order π-calculus, such as barbed equivalence, context bisimulation and normal bisimulation, were presented in [29]. Barbed equivalence can be regarded as a uniform definition of bisimulation for a variety of concurrent calculi. Context bisimulation is a very intuitive definition of bisimulation for higher order π-calculus, but it is heavy to handle, due to the appearance of universal quantifications in its definition. In the definition of normal bisimulation, all universal quantifications disappeared, therefore normal bisimulation is a very economic characterization of bisimulation for higher order π-calculus. The coincidence among the three weak equivalences was proven [29], [28], [20]. Moreover, this proposition was generalized to the strong bisimulation case [10].

Spatial logic was presented in [12]. Spatial logic extends classical logic with connectives to reason about the structure of the processes. The additional connectives belong to two families. Intensional operators allow one to inspect the structure of the process. A formula $A_1|A_2$ is satisfied whenever we can split the process into two parts satisfying the corresponding subformula A_i, $i = 1, 2$. In the presence of restriction in the underlying model, a process P satisfies formula $n\circledR A$ if we can write P as $(\nu n)P'$ with P' satisfying A. Finally, formula 0 is only satisfied by the inaction process. Connectives $|$ and $\circledR$ come with adjunct operators, called guarantee ($\triangleright$) and hiding ($\oslash$) respectively, which allow one to extend the process being observed. In this sense, these can be called contextual operators. P satisfies $A_1 \triangleright A_2$ whenever the spatial composition (using $|$) of P with any process satisfying A_1 satisfies A_2, and P satisfies $A \oslash n$ if $(\nu n)P$ satisfies A. Some spatial logics have an operator for fresh name quantification [11].

There are lots of works of spatial logics for π-calculus and *Mobile Ambients*. In some papers, spatial logic was studied on its relations with structural congruence, bisimulation, model checking and type system of process calculi [5], [6], [9], [16], [27].

The main idea of this paper is that the theory of processes can be reduced to the theory of spatial logic.

In this paper, we present a spatial logic for higher order π-calculus, called SL, which comprises some action temporal operators such as $\langle\tau\rangle$ and $\langle a\langle A\rangle\rangle$, some spatial operators such as prefix and composition, some adjunct operators of spatial operators such as $\triangleright$ and $\oslash$, and some operators on the property of free names and bound names such as $\ominus n$ and $\widetilde{\ominus}$. We give an inference system of SL, and prove the soundness of the inference system for SL. Furthermore, we show that there is no finite complete inference system for SL. Then we study the relation between processes and SL formulas. We show that an SL formula can be viewed as a specification of processes, and conversely, a process can be viewed as a special kind of SL formulas. Therefore, SL is a generalization of processes, which extend process with specification statements. We show that the structural congruence relation and one-step transition relation can be described as the logical relation of SL formulas. We also show that bisimulations for higher order processes coincide with logical equivalence with respect to some fragment of a sublogic of SL.

Furthermore, we give a weak semantics version of SL, called WL, where the internal action is unobservable. The results of SL are extended to WL, such as an inference system for WL, the soundness of this inference system, and the non-existence of a finite complete inference system for WL.

Finally, we add μ-operator to SL. The new logic is named μSL. We show that WL is a sublogic of μSL and replication operator can be expressed in μSL. Thus μSL is a powerful logic which can express both strong semantics and weak semantics for higher order processes.

This paper is organized as follows: In Section 2, we briefly review higher order π-calculus. In Section 3, we present a spatial logic SL, including its syntax, semantics and inference system. The soundness and incompleteness of the inference

 ISBN: 978-1-61208-277-6

system of SL are proved. Furthermore, we discuss that SL can be regarded as a specification language of processes and processes can be regarded as a kind of special formulas of SL. Bisimulation in higher order π-calculus coincides with logical equivalence with respect to some fragment of a sublogic of SL. In Section 4, we give a weak semantics version of SL, called WL. We generalize concepts and results of SL to WL. The paper is concluded in Section 5.

II. Higher Order π-Calculus

A. Syntax and Labelled Transition System

In this section we briefly recall the syntax and labelled transition system of the higher order π-calculus. Similar to [28], we only focus on a second-order fragment of the higher order π-calculus, i.e., there is no abstraction in this fragment.

We assume a set N of names, ranged over by $a, b, c, ...$ and a set Var of process variables, ranged over by $X, Y, Z, U, ...$ We use $E, F, P, Q, ...$ to stand for processes. Pr denotes the set of all processes.

We first give the syntax for the higher order π-calculus processes as follows:

$P ::= 0 \mid U \mid \pi.P \mid P_1|P_2 \mid (\nu a)P$

π is called a prefix and can have one of the following forms:

$\pi ::= a(U) \mid \overline{a}\langle P\rangle$, here $a(U)$ is a higher order input prefix and $\overline{a}\langle P\rangle$ is a higher order output prefix.

In each process of the form $(\nu a)P$ the occurrence of a is bound within the scope of P. An occurrence of a in a process is said to be free iff it does not lie within the scope of a bound occurrence of a. The set of names occurring free in P is denoted $fn(P)$. An occurrence of a name in a process is said to be bound if it is not free, and we write the set of bound names as $bn(P)$. $n(P)$ denotes the set of names of P, i.e., $n(P) = fn(P) \cup bn(P)$.

Higher order input prefix $a(U).P$ binds all free occurrences of U in P. The set of variables occurring free in P is denoted $fv(P)$. We write the set of bound variables as $bv(P)$. A process is closed if it has no free variable; it is open if it may have free variables. Pr^c is the set of all closed processes.

Processes P and Q are α-convertible, $P \equiv_\alpha Q$, if Q can be obtained from P by a finite number of changes of bound names and variables. For example, $(\nu b)(\overline{a}\langle b(U).U\rangle.0) \equiv_\alpha (\nu c)(\overline{a}\langle c(U).U\rangle.0)$.

Structural congruence is the smallest congruence relation that validates the following axioms: $P|Q \equiv Q|P$; $(P|Q)|R \equiv P|(Q|R)$; $P|0 \equiv P$; $(\nu a)0 \equiv 0$; $(\nu m)(\nu n)P \equiv (\nu n)(\nu m)P$; $(\nu a)(P|Q) \equiv P|(\nu a)Q$ if $a \notin fn(P)$.

In [26], Parrow has shown that in higher order π-calculus, the replication can be defined by other operators such as higher order prefix, parallel and restriction. For example, $!P$ can be simulated by $R_P = (\nu a)(D|\overline{a}\langle P|D\rangle.0)$, where $D = a(X).(X|\overline{a}\langle X\rangle.0)$.

The operational semantics of higher order processes is given in Table 1. We have omitted the symmetric cases of the parallelism and communication rules.

$$ALP: \frac{P \xrightarrow{\alpha} P'}{Q \xrightarrow{\alpha} Q'}\ P \equiv Q, P' \equiv Q'$$

$$OUT: \overline{a}\langle E\rangle.P \xrightarrow{\overline{a}\langle E\rangle} P$$

$$IN: a(U).P \xrightarrow{a\langle E\rangle} P\{E/U\}\ bn(E) = \emptyset$$

$$PAR: \frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q}\ bn(\alpha) \cap fn(Q) = \emptyset$$

$$COM: \frac{P \xrightarrow{(\nu\widetilde{b})\overline{a}\langle E\rangle} P' \quad Q \xrightarrow{a\langle E\rangle} Q'}{P|Q \xrightarrow{\tau} (\nu\widetilde{b})(P'|Q')}\ \widetilde{b} \cap fn(Q) = \emptyset$$

$$RES: \frac{P \xrightarrow{\alpha} P'}{(\nu a)P \xrightarrow{\alpha} (\nu a)P'}\ a \notin n(\alpha)$$

$$OPEN: \frac{P \xrightarrow{(\nu\widetilde{c})\overline{a}\langle E\rangle} P'}{(\nu b)P \xrightarrow{(\nu b,\widetilde{c})\overline{a}\langle E\rangle} P'}\ a \neq b,\ b \in fn(E) - \widetilde{c}$$

$$REP: \frac{P|!P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P'}$$

Table 1. The operational semantics of higher order π-calculus

B. Bisimulations in Higher Order π-Calculus

Context bisimulation and contextual barbed bisimulation were presented in [29] to describe the behavioral equivalences for higher order π-calculus. Let us review the definition of these bisimulations. In the following, we abbreviate $P\{E/U\}$ as $P\langle E\rangle$.

Context bisimulation is an intuitive definition of bisimulation for higher order π-calculus.

Definition 1 A symmetric relation $R \subseteq Pr^c \times Pr^c$ is a strong context bisimulation if $P\ R\ Q$ implies:

(1) whenever $P \xrightarrow{\tau} P'$, there exists Q' such that $Q \xrightarrow{\tau} Q'$ and $P'\ R\ Q'$;

(2) whenever $P \xrightarrow{a\langle E\rangle} P'$, there exists Q' such that $Q \xrightarrow{a\langle E\rangle} Q'$ and $P'\ R\ Q'$;

(3) whenever $P \xrightarrow{(\nu\widetilde{b})\overline{a}\langle E\rangle} P'$, there exist Q', F, $\widetilde{c}$ such that $Q \xrightarrow{(\nu\widetilde{c})\overline{a}\langle F\rangle} Q'$ and for all $C(U)$ with $fn(C(U)) \cap \{\widetilde{b}, \widetilde{c}\} = \emptyset$, $(\nu\widetilde{b})(P'|C\langle E\rangle)\ R\ (\nu\widetilde{c})(Q'|C\langle F\rangle)$. Here $C(U)$ represents a process containing a unique free variable U.

We write $P \sim_{Ct} Q$ if P and Q are strongly context bisimilar.

Contextual barbed equivalence can be regarded as a uniform definition of bisimulation for a variety of process calculi.

Definition 2 A symmetric relation $R \subseteq Pr^c \times Pr^c$ is a strong contextual barbed bisimulation if $P\ R\ Q$ implies:

(1) $P|C\ R\ Q|C$ for any C;

(2) whenever $P \xrightarrow{\tau} P'$ then there exists Q' such that $Q \xrightarrow{\tau} Q'$ and $P'\ R\ Q'$;

(3) $P \downarrow_\mu$ implies $Q \downarrow_\mu$, where $P \downarrow_a$ if $\exists P'$, $P \xrightarrow{a\langle E\rangle} P'$, and $P \downarrow_{\overline{a}}$ if $\exists P'$, $P \xrightarrow{(\nu\widetilde{b})\overline{a}\langle E\rangle} P'$.

We write $P \sim_{Ba} Q$ if P and Q are strongly contextual barbed bisimilar.

Intuitively, a tau action represents the internal action of processes. If we just consider external actions, then we should adopt weak bisimulations to characterize the equivalence of processes.

Definition 3 A symmetric relation $R \subseteq Pr^c \times Pr^c$ is a weak context bisimulation if $P\ R\ Q$ implies:

(1) whenever $P \stackrel{\varepsilon}{\Longrightarrow} P'$, there exists Q' such that $Q \stackrel{\varepsilon}{\Longrightarrow} Q'$ and $P'\ R\ Q'$;

(2) whenever $P \stackrel{a\langle E\rangle}{\Longrightarrow} P'$, there exists Q' such that $Q \stackrel{a\langle E\rangle}{\Longrightarrow} Q'$ and $P'\ R\ Q'$;

(3) whenever $P \stackrel{(\nu\widetilde{b})\overline{a}\langle E\rangle}{\Longrightarrow} P'$, there exist Q', F, $\widetilde{c}$ such that $Q \stackrel{(\nu\widetilde{c})\overline{a}\langle F\rangle}{\Longrightarrow} Q'$ and for all $C(U)$ with $fn(C(U)) \cap \{\widetilde{b},\widetilde{c}\} = \emptyset$, $(\nu\widetilde{b})(P'|C\langle E\rangle)\ R\ (\nu\widetilde{c})(Q'|C\langle F\rangle)$. Here $C(U)$ represents a process containing a unique free variable U.

We write $P \approx_{Ct} Q$ if P and Q are weakly context bisimilar.

Definition 4 A symmetric relation $R \subseteq Pr^c \times Pr^c$ is a weak contextual barbed bisimulation if $P\ R\ Q$ implies:

(1) $P|C\ R\ Q|C$ for any C;

(2) whenever $P \stackrel{\varepsilon}{\Longrightarrow} P'$ then there exists Q' such that $Q \stackrel{\varepsilon}{\Longrightarrow} Q'$ and $P'\ R\ Q'$;

(3) $P \Downarrow_\mu$ implies $Q \Downarrow_\mu$, where $P \Downarrow_\mu$ if $\exists P'$, $P \stackrel{\varepsilon}{\Longrightarrow} P'$ and $P' \downarrow_\mu$.

We write $P \approx_{Ba} Q$ if P and Q are weakly contextual barbed bisimilar.

III. Logics for Strong Semantics

In this section, we present a logic to reason about higher order π-calculus called SL. This logic extends propositional logic with three kinds of connectives: action temporal operators, spatial operators, operators about names and variables. We give the syntax and semantics of SL. The inference system of SL is also given. We prove the soundness and incompleteness of this inference system. As far as we know, this is the first result on the completeness problem of the inference system of spatial logic. Furthermore, we show that structural congruence, one-step transition relation and bisimulation can all be characterized by this spatial logic. It is well known that structural congruence, one-step transition relation and bisimulation are the central concepts in the theory of processes, and almost all the studies of process calculi are about these concepts. Therefore, our study gives an approach of reducing theory of processes to theory of spatial logic. Moreover, since processes can be regarded as a special kind of spatial logic formulas, spatial logic can be viewed as an extension of process calculus. Based on spatial logic, it is possible to propose a refinement calculus [23] of concurrent processes.

A. Syntax and Semantics of Logic SL

Now we introduce a logic called SL, which is a spatial logic for higher order π-calculus.

Definition 5 Syntax of logic SL

$A ::= \top \mid \bot \mid \neg A \mid A_1 \wedge A_2 \mid \langle\tau\rangle A \mid \langle a\langle A_1\rangle\rangle A_2 \mid \langle a[A_1]\rangle A_2 \mid \langle\overline{a}\langle A_1\rangle\rangle A_2 \mid 0 \mid X \mid a \odot X.A \mid A \setminus a \odot X \mid \overline{a}\langle A_1\rangle.A_2 \mid A \setminus \overline{a} \mid A_1|A_2 \mid A_1 \triangleright A_2 \mid a\circledR A \mid A \oslash a \mid (\mathbf{N}x)A \mid (\mathbf{N}X)A \mid (\ominus a)A \mid (\tilde{\ominus})A \mid a \neq b \mid X \mid \mu X.A(X)$ where X occurs positively in $A(X)$, i.e., all free occurrences of X fall under an even number of negations.

In $(\mathbf{N}x)A$, $(\mathbf{N}X)A$, the variables x (and X) are bound within the scope of the formula A. We assume that the standard relation $\equiv_\alpha$ of α-conversion (safe renaming of bound variables) was defined on formulas, but we never implicitly take formulas "up to α-conversion": our manipulation of variables via α-conversion steps is always quite explicit. The set $fn(A)$ of free names in A, and the set $fpv(A)$ of free propositional variables in A, are defined in the usual way. A formula is closed if it has no free variable such as X, and it is open if it may have free variables. SL^c is the set of all closed formulas. In the following, we use $A\{b/a\}$ to denote the formula obtained by replacing all occurrences of a in A by b. Similarly, we use $A\{Y/X\}$ to denote the formula obtained by replacing all occurrences of X in A by Y. It is easy to see that a process can also be regarded as a spatial formula. For example, process $\overline{a}\langle E\rangle.P$ is also a spatial formula. In this paper, we say that such a formula is in the form of process formula.

Semantics of SL is given as following:

We write such set of processes in which A is true as $[[A]]^e_{Pr}$, where e: $Var \to 2^{Pr}$ is an environment. We denote by $e[X \leftarrow W]$ a new environment that is the same as e except that $e[X \leftarrow W](X) = W$. The set $[[A]]^e_S$ is the set of processes that satisfy A with respect to e.

Definition 6 Semantics of logic SL

$[[\top]]^e_{Pr} = Pr$;

$[[\bot]]^e_{Pr} = \emptyset$;

$[[\neg A]]^e_{Pr} = Pr - [[A]]^e_{Pr}$;

$[[A_1 \wedge A_2]]^e_{Pr} = [[A_1]]^e_{Pr} \cap [[A_2]]^e_{Pr}$;

$[[\langle\tau\rangle A]]^e_{Pr} = \{P \mid \exists Q.\ P \stackrel{\tau}{\longrightarrow} Q \text{ and } Q \in [[A]]^e_{Pr}\}$;

$[[\langle a\langle A_1\rangle\rangle A_2]]^e_{Pr} = \{P \mid \exists P_1, P_2.\ P \stackrel{a\langle P_1\rangle}{\longrightarrow} P_2,\ P_1 \in [[A_1]]^e_{Pr} \text{ and } P_2 \in [[A_2]]^e_{Pr}\}$;

$[[\langle a[A_1]\rangle A_2]]^e_{Pr} = \{P \mid \forall R, R \in [[A_1]]^e_{Pr}, \exists Q.\ P \stackrel{a\langle R\rangle}{\longrightarrow} Q \text{ and } Q \in [[A_2]]^e_{Pr}\}$;

$[[\langle\overline{a}\langle A_1\rangle\rangle A_2]]^e_{Pr} = \{P \mid \exists P_1, P_2.\ P \stackrel{(\nu\widetilde{b})\overline{a}\langle P_1\rangle}{\longrightarrow} P_2, (\nu\widetilde{b})P_1 \in [[A_1]]^e_{Pr} \text{ and } P_2 \in [[A_2]]^e_{Pr}\}$;

$[[0]]^e_{Pr} = \{P \mid P \equiv 0\}$;

$[[X]]^e_{Pr} = \{P \mid P \equiv X\}$;

$[[a \odot X.A]]^e_{Pr} = \{P \mid \exists Q.\ P \equiv a(X).Q \text{ and } Q \in [[A]]^e_{Pr}\}$;

$[[A \setminus a \odot X]]^e_{Pr} = \{P \mid a(X).P \in [[A]]^e_{Pr}\}$;

$[[\overline{a}\langle A_1\rangle.A_2]]^e_{Pr} = \{P \mid \exists P_1, P_2.\ P \equiv \overline{a}\langle P_1\rangle.P_2,\ P_1 \in [[A_1]]^e_{Pr} \text{ and } P_2 \in [[A_2]]^e_{Pr}\}$;

$[[A \setminus \overline{a}]]^e_{Pr} = \{P \mid \overline{a}\langle P\rangle.0 \in [[A]]^e_{Pr}\}$;

$[[A_1|A_2]]^e_{Pr} = \{P \mid \exists Q_1, Q_2.\ P \equiv Q_1|Q_2,\ Q_1 \in [[A_1]]^e_{Pr}$ and $Q_2 \in [[A_2]]^e_{Pr}\}$;

$[[A_1 \triangleright A_2]]^e_{Pr} = \{P \mid \forall Q.\ Q \in [[A_1]]^e_{Pr}$ implies $P|Q \in [[A_2]]^e_{Pr}\}$;

$[[a\textcircled{R}A]]^e_{Pr} = \{P \mid \exists Q.\ P \equiv (\nu a)Q$ and $Q \in [[A]]^e_{Pr}\}$;

$[[A \oslash a]]^e_{Pr} = \{P \mid (\nu a)P \in [[A]]^e_{Pr}\}$;

$[[(\mathbf{N}x)A]]^e_{Pr} = \cup_{n \notin fn((\mathbf{N}x)A)}([[A\{n/x\}]]^e_{Pr} \backslash \{P \mid n \in fn(P)\})$;

$[[(\mathbf{N}X)A]]^e_{Pr} = \cup_{V \notin fpv((\mathbf{N}X)A)}([[A\{V/X\}]]^e_{Pr} \backslash \{P \mid V \in fpv(P)\})$;

$[[(\ominus a)A]]^e_{Pr} = \{P \mid a \notin fn(P)$ and $P \in [[A]]^e_{Pr}\}$;

$[[(\tilde{\ominus})A]]^e_{Pr} = \{P \mid \exists Q.\ P \equiv Q$ and $bn(Q) = \emptyset$ and $Q \in [[A]]^e_{Pr}\}$;

$[[a \neq b]]^e_{Pr} = Pr$ if $a \neq b$;

$[[a \neq b]]^e_{Pr} = \emptyset$ if $a = b$.

$[[X]]^e_{Pr} = e(X)$

$[[\mu X.A(X)]]^e_{Pr} = \cap\{W \subseteq Pr \mid [[A(X)]]^{e[X \leftarrow W]}_{Pr} \subseteq W\}$.

In SL, formula $\langle a\langle A_1\rangle\rangle A_2$ is satisfied by the processes that can receive a process satisfying A_1 and then become a process satisfying A_2. A process satsfies formula $\langle a[A_1]\rangle A_2$ if it receive any process satisfying A_1 then it becomes a process satisfying A_2. $A \setminus a \odot X$ is an adjunct operator of $a \odot X.A$, and $A \setminus \overline{a}$ is an adjunct operator of $\overline{a}\langle A\rangle.0$. $(\ominus a)A$ is satisfied by processes that satisfy A and a is not its free name. $(\tilde{\ominus})A$ is satisfied by processes that satisfy A and have no bound names. Other operators in SL are well known in spatial logic or can be interpreted similarly as above operators.

Definition 7 $P \models_{SL} A$ if $P \in [[A]]^e_{Pr}$.

Definition 8 For a set of formulas Γ and a formula A, we write $\Gamma \models_{SL} A$, if A is valid in all processes that satisfy all formulas of Γ.

For example, the following equations hold in SL:

$\{P \mid \forall P_1.\ P_1 \in [[A_1]]^e_{Pr}$ implies $\overline{a}\langle P_1\rangle.P \in [[A_2]]^e_{Pr}\} = [[(b \odot Y.\overline{a}\langle A_1\rangle.Y \triangleright \langle\tau\rangle A_2) \setminus \overline{b}]]^e_{Pr}$.

$\{P \mid \forall P_1.\ P_1 \in [[A_1]]^e_{Pr}$ implies $\overline{a}\langle P\rangle.P_1 \in [[A_2]]^e_{Pr}\} = [[(b \odot Y.\overline{a}\langle Y\rangle.A_1 \triangleright \langle\tau\rangle A_2) \setminus \overline{b}]]^e_{Pr}$.

$\{P \mid a \in fn(P)$ and $P \in [[A]]^e_{Pr}\} = [[\neg(\ominus a)\top \wedge A]]^e_{Pr}$.

$\{P \mid X \in fv(P)$ and $P \in [[A]]^e_{Pr}\} = [[\neg(\ominus X)\top \wedge A]]^e_{Pr}$.

$\{(\nu n)P \mid P \in [[A\{n/x\}]]^e_{Pr}\} = [[(\mathbf{N}x)x\textcircled{R}A]]^e_{Pr}$.

$\{a(U).P \mid P \in [[A\{U/X\}]]^e_{Pr}\} = [[(\mathbf{N}X)a \odot X.A]]^e_{Pr}$.

$[[A|!A]]^e_{Pr} = [[!A]]^e_{Pr}$, where $!A \stackrel{def}{=} \neg\mu X.\neg(A|\neg X)$.

B. Inference System of SL

Now we list a number of valid properties of spatial logic. The combination of the complete inference system of first order logic and the following axioms and rules form the inference system S of SL.

(1) $\langle\alpha\rangle\bot \rightarrow \bot$;

(2) $a \odot X.\bot \rightarrow \bot$;

(3) $\overline{a}\langle\top\rangle.\bot \rightarrow \bot$;

(4) $\overline{a}\langle\bot\rangle.\top \rightarrow \bot$;

(5) $\bot \setminus a \odot X \rightarrow \bot$;

(6) $\bot \setminus \overline{a} \rightarrow \bot$;

(7) $A|\bot \rightarrow \bot$;

(8) $A \triangleright \bot \rightarrow \neg A$;

(9) $\bot \triangleright A \leftrightarrow \top$;

(10) $a\textcircled{R}\bot \rightarrow \bot$;

(11) $\bot \oslash a \rightarrow \bot$;

(12) $(\ominus a)\bot \rightarrow \bot$;

(13) $(\mathbf{N}x)\bot \rightarrow \bot$;

(14) $(\tilde{\ominus})\bot \rightarrow \bot$;

(15) $(\mathbf{N}X)\bot \rightarrow \bot$;

(16) $A|B \leftrightarrow B|A$;

(17) $(A|B)|C \leftrightarrow A|(B|C)$;

(18) $A|0 \leftrightarrow A$;

(19) $a\textcircled{R}0 \leftrightarrow 0$;

(20) $a\textcircled{R}b\textcircled{R}A \leftrightarrow b\textcircled{R}a\textcircled{R}A$;

(21) $a\textcircled{R}((\ominus a)A|B) \leftrightarrow (\ominus a)A|a\textcircled{R}B$;

(22) $a\textcircled{R}A \rightarrow (\mathbf{N}b)b\textcircled{R}A\{b/a\}$;

(23) $a \odot X.A \rightarrow (\mathbf{N}Y)a \odot Y.A\{Y/X\}$;

(24) $(\ominus a)0 \leftrightarrow 0$;

(25) $(\ominus a)X \leftrightarrow X$;

(26) $(\ominus a)a \odot X.A \leftrightarrow \bot$;

(27) $(\ominus a)\overline{a}\langle B\rangle.A \leftrightarrow \bot$;

(28) $a \neq b \rightarrow ((\ominus a)b \odot X.A \leftrightarrow b \odot X.(\ominus a)A)$;

(29) $a \neq b \rightarrow ((\ominus a)\overline{b}\langle B\rangle.A \leftrightarrow \overline{b}\langle(\ominus a)B\rangle.(\ominus a)A)$;

(30) $(\ominus a)A|(\ominus a)B \leftrightarrow (\ominus a)(A|B)$;

(31) $a \neq b \rightarrow ((\ominus a)(\ominus b)A \leftrightarrow (\ominus b)(\ominus a)A)$;

(32) $(\ominus a)a\textcircled{R}A \leftrightarrow a\textcircled{R}A$;

(33) $(\tilde{\ominus})0 \leftrightarrow 0$;

(34) $(\tilde{\ominus})X \leftrightarrow X$;

(35) $(\tilde{\ominus})a \odot X.A \leftrightarrow a \odot X.(\tilde{\ominus})A$;

(36) $(\tilde{\ominus})\overline{a}\langle B\rangle.A \leftrightarrow \overline{a}\langle(\tilde{\ominus})B\rangle.(\tilde{\ominus})A$;

(37) $(\tilde{\ominus})A|(\tilde{\ominus})B \leftrightarrow (\tilde{\ominus})(A|B)$;

(38) $(\tilde{\ominus})a\textcircled{R}\neg(\ominus a)\top \rightarrow \bot$;

(39) $(\mathbf{N}x)0 \leftrightarrow 0$;

(40) $(\mathbf{N}x)X \leftrightarrow X$;

(41) $(\mathbf{N}x)a \odot X.A \leftrightarrow a \odot X.(\mathbf{N}x)(x \neq a \wedge A)$;

 ISBN: 978-1-61208-277-6

(42) $(\mathbf{N}x)\overline{a}\langle B\rangle.A \rightarrow \overline{a}\langle(\mathbf{N}x)(x \neq a \wedge B)\rangle.(\mathbf{N}x)(x \neq a \wedge A)$;

(43) $(\mathbf{N}x)(A|B) \rightarrow (\mathbf{N}x)A|(\mathbf{N}x)B$;

(44) $(\mathbf{N}x)x \neq a \wedge a®A \rightarrow a®(\mathbf{N}x)A$;

(45) $(\mathbf{N}X)0 \leftrightarrow 0$;

(46) $(\mathbf{N}X)X \rightarrow Y$;

(47) $(\mathbf{N}X)a \odot Y.A \leftrightarrow a \odot Y.(\mathbf{N}X)A$;

(48) $(\mathbf{N}X)\overline{a}\langle B\rangle.A \rightarrow \overline{a}\langle(\mathbf{N}X)B\rangle.(\mathbf{N}X)A$;

(49) $(\mathbf{N}X)(A|B) \rightarrow (\mathbf{N}X)A|(\mathbf{N}X)B$;

(50) $(\mathbf{N}X)a®A \leftrightarrow a®(\mathbf{N}X)A$;

(51) $a \odot X.(A \setminus a \odot X) \rightarrow A$;

(52) $A \rightarrow (a \odot X.A) \setminus a \odot X)$;

(53) $\overline{a}\langle A \setminus \overline{a}\rangle.0 \rightarrow A$;

(54) $A \rightarrow ((\overline{a}\langle A\rangle.0) \setminus \overline{a})$;

(55) $(A|A \triangleright B) \rightarrow B$;

(56) $A \rightarrow (B \triangleright A|B)$;

(57) $a®(A \oslash a) \rightarrow A$;

(58) $A \rightarrow (a®A \oslash a)$;

(59) $\langle\alpha\rangle A, A \rightarrow B \vdash \langle\alpha\rangle B$;

(60) $a \odot X.A, A \rightarrow B \vdash a \odot X.B$;

(61) $\overline{a}\langle C\rangle.A, A \rightarrow B \vdash \overline{a}\langle C\rangle.B$;

(62) $\overline{a}\langle B\rangle.A, B \rightarrow C \vdash \overline{a}\langle C\rangle.A$;

(63) $\langle\overline{a}\langle B\rangle\rangle A, C \rightarrow B \vdash \langle\overline{a}\langle C\rangle\rangle A$;

(64) $\langle a[B]\rangle A, C \rightarrow B \vdash \langle a[C]\rangle A$;

(65) $A \setminus a \odot X, A \rightarrow B \vdash B \setminus a \odot X$;

(66) $A \setminus \overline{a}, A \rightarrow B \vdash B \setminus \overline{a}$;

(67) $A \rightarrow B \vdash A|C \rightarrow B|C$;

(68) $a®A, A \rightarrow B \vdash a®B$;

(69) $(\ominus a)A, A \rightarrow B \vdash (\ominus a)B$;

(70) $(\tilde{\ominus})A, A \rightarrow B \vdash (\tilde{\ominus})B$;

(71) $\overline{a}\langle B\rangle.A \rightarrow \langle\overline{a}\langle B\rangle\rangle A$;

(72) $(\langle\tau\rangle A)|B \rightarrow \langle\tau\rangle(A|B)$;

(73) $(\langle a\langle C\rangle\rangle A)|B \rightarrow \langle a\langle C\rangle\rangle(A|B)$;

(74) $(a \odot U.A \wedge ((\tilde{\ominus})B \leftrightarrow B)) \rightarrow \langle a[B]\rangle A\{B/U\}$;

(75) $(((\ominus b_1, ..., \ominus b_n)B \leftrightarrow B) \wedge ((\tilde{\ominus})C \leftrightarrow C)) \rightarrow (\langle\overline{a}\langle b_1®...b_n®C\rangle\rangle A)|B \rightarrow \langle\overline{a}\langle b_1®...b_n®C\rangle\rangle(A|B))$;

(76) $(((\ominus b_1, ..., \ominus b_n)B \leftrightarrow B) \wedge ((\tilde{\ominus})C \leftrightarrow C)) \rightarrow (\langle\overline{a}\langle b_1®...b_n®C\rangle\rangle A)|\langle a[C]\rangle B \rightarrow \langle\tau\rangle b_1®...b_n®(A|B))$;

(77) $(a \neq b \wedge ((\ominus a)B \leftrightarrow B) \wedge ((\tilde{\ominus})B \leftrightarrow B)) \rightarrow (a®\langle b\langle B\rangle\rangle A \rightarrow \langle b\langle B\rangle\rangle a®A)$;

(78) $(\wedge_{i=1}^{n} a \neq b_i \wedge a \neq c \wedge ((\ominus a)B \leftrightarrow B) \wedge ((\tilde{\ominus})B \leftrightarrow B)) \rightarrow (a®\langle\overline{c}\langle b_1®...b_n®B\rangle\rangle A \rightarrow \langle\overline{c}\langle b_1®...b_n®B\rangle\rangle a®A)$;

(79) $(a \neq b \wedge (\wedge_{i=1}^{n} b \neq c_i) \wedge\ (B \rightarrow \neg(\ominus b)\top) \wedge ((\tilde{\ominus})B \leftrightarrow B)) \rightarrow (b®\langle\overline{a}\langle c_1®...c_n®B\rangle\rangle A \rightarrow \langle\overline{a}\langle b®c_1®...c_n®B\rangle\rangle A)$;

(80) $\langle a[B]\rangle A \rightarrow \langle a\langle B\rangle\rangle A$;

(81) $\langle a\langle B\rangle\rangle A \rightarrow \langle a[B]\rangle A$, *where B is syntactically a valid process in the higher order $\pi-calculus$.*

(82) $A(\mu X.A(X)) \rightarrow \mu X.A(X)$;

(83) $(A(B) \rightarrow B) \rightarrow (\mu X.A(X) \rightarrow B)$.

Intuitively, axiom $a®A \rightarrow (\mathbf{N}b)b®A\{b/a\}$ means that if process P satisfies $(\nu a)A$ and b is a fresh name then P satisfies $(\nu b)A\{b/a\}$. Axiom $\overline{a}\langle B\rangle.A \rightarrow \langle\overline{a}\langle B\rangle\rangle A$ means that an output prefix process can perform an output action, which is a spatial logical version of Rule OUT in the labelled transition system of higher order π-calculus. Axiom $(a \odot U.A \wedge ((\tilde{\ominus})B \leftrightarrow B)) \rightarrow \langle a[B]\rangle A\{B/U\}$ means that an input prefix process can perform an input action, which is a spatial logical version of Rule IN in the labelled transition system of higher order π-calculus. Axiom $(((\ominus b_1, ..., \ominus b_n)B \leftrightarrow B) \wedge ((\tilde{\ominus})C \leftrightarrow C)) \rightarrow (\langle\overline{a}\langle b_1®...b_n®C\rangle\rangle A)|\langle a[C]\rangle B \rightarrow \langle\tau\rangle b_1®...b_n®(A|B))$ is a spatial logical version of Rule COM. Other axioms and rules are spatial logical version of structural congruence rules or labelled transition rules similarly.

Definition 9 If "$A_1, ..., A_n$ infer B" is an instance of an inference rule, and if the formulas $A_1, ..., A_n$ have appeared earlier in the proof, then we say that B follows from an application of an inference rule. A proof is said to be from Γ to A if the premise is Γ and the last formula is A in the proof. We say A is provable from Γ in an inference system AX, and write $\Gamma \vdash_{AX} A$, if there is a proof from Γ to A in AX.

C. Soundness of SL

Inference system of SL is said to be sound with respect to processes if every formula provable in SL is valid with respect to processes.

Now, we can prove the soundness of inference system S of SL:

Proposition 1 $\Gamma \vdash_S A \Rightarrow \Gamma \models_{SL} A$

Proof. See Appendix A. ■

D. Incompleteness of SL

The system SL is complete with respect to processes if every formula valid with respect to processes is provable in SL. For a logic, completeness is an important property. The soundness and completeness provide a tight connection between the syntactic notion of provability and the semantic notion of validity. Unfortunately, by the compactness property [18], the inference system of SL is not complete.

The depth of higher order processes in Pr, is defined as below:

Definition 10 $d(0)=0$; $d(U)=0$; $d(a(U).P)=1+d(P)$; $d(\overline{a}\langle E\rangle.P)=1+d(E)+d(P)$; $d(P_1|P_2)=d(P_1)+d(P_2)$; $d((\nu a)P)=d(P)$.

Lemma 1 For any $P \in Pr$, there exists n, such that $d(P)=n$.

$Proof$. Induction on the structure of P.

Proposition 2 There is no finite sound inference system AX such that $\Gamma \models_{SL} A \Rightarrow \Gamma \vdash_{AX} A$.

$Proof$. See Appendix B. ■

E. Spatial Logic as a Specification of Processes

In the refinement calculus [23], imperative programming languages are extended by specification statements, which specify parts of a program "yet to be developed". Then the development of a program begins with a specification statement, and ends with an executable program by refining a specification to its possible implementations. In this paper, we generalize this idea to the case of process calculi. Roughly speaking, we extend processes to spatial logic formulas which are regarded as the specification statements. One can view the intensional operators of spatial logic as the "executable program statements", for example, $\overline{a}\langle P\rangle.Q$, $P|Q$ and etc; and view the extensional operators of spatial logic as the "specification statements", for example, $A \triangleright B$, $A \backslash \overline{b}$ and etc. For example, $(b \odot Y.\overline{a}\langle Y\rangle.A_1 \triangleright \langle\tau\rangle A_2) \backslash \overline{b} | (d \odot Y.\overline{c}\langle B_1\rangle.Y \triangleright \langle\tau\rangle B_2) \backslash \overline{d}$ represents a specification statement which describes a process consisting of a parallel of two processes satisfying statements $(b \odot Y.\overline{a}\langle Y\rangle.A_1 \triangleright \langle\tau\rangle A_2) \setminus \overline{b}$ and $(d \odot Y.\overline{c}\langle B_1\rangle.Y \triangleright \langle\tau\rangle B_2) \setminus \overline{d}$ respectively. Furthermore, $(b \odot Y.\overline{a}\langle Y\rangle.A_1 \triangleright \langle\tau\rangle A_2) \setminus \overline{b}$ represents a specification which describes a process P such that $\overline{a}\langle P\rangle.Q$ satisfies A_2 for any Q satisfying A_1. Similarly, $(d \odot Y.\overline{c}\langle B_1\rangle.Y \triangleright \langle\tau\rangle B_2) \backslash \overline{d}$ represents a specification statement which describes a process M such that $\overline{c}\langle N\rangle.M$ satisfies B_2 for any N satisfying B_1. We can also define refinement relation on spatial logic formulas. Intuitively, if $\models_{SL} A \to B$, then A refines B. For example, $a\circledR(a \odot X.d.X|\overline{a}\langle c.0\rangle.e.0)$ refines $a\circledR(\langle a[c.0]\rangle d.c.0|\langle\overline{a}\langle c.0\rangle\rangle e.0)$. Based on spatial logic, one may develop a theory of refinement for concurrent processes. This will be a future research direction for us.

F. Processes as Special Formulas of Spatial Logic

Any process can be regarded as a special formula of spatial logic. For example, $(\mathbf{N}a)a\circledR(\mathbf{N}X)(a \odot X.d.X|\overline{a}\langle c.0\rangle.e.0)$ is a spatial logic formula, which represents the process which is structural congruent to $(\nu a)(a(X).d.X|\overline{a}\langle c.0\rangle.e.0)$. Furthermore, in this section, we will show that structural congruence and labelled transition relation can be reformulated as the logical relation of spatial logical formulas.

Definition 11 The translating function T^{PS} is defined inductively as follows:

$T^{PS}(P) \stackrel{def}{=} P$ for process P that has no operators of $(\nu a)\cdot$, or $a(X).\cdot$;

$T^{PS}((\nu a)P) \stackrel{def}{=} (\mathbf{H}a)T^{PS}(P)$;

$T^{PS}(a(X).P) \stackrel{def}{=} (a\mathbf{H}X)T^{PS}(P)$.

Lemma 2 $\vdash_{SL} A|!A \leftrightarrow !A$, where $!A \stackrel{def}{=} \neg\mu X.\neg(A|\neg X)$.

$Proof$: See Appendix C. ■

Proposition 3 For any $P,Q \in Pr^c$, $P \equiv Q \Leftrightarrow P \models_{SL} T^{PS}(Q)$ and $Q \models_{SL} T^{PS}(P) \Leftrightarrow T^{PS}(P) \vdash_{SL} T^{PS}(Q)$ and $T^{PS}(Q) \vdash_{SL} T^{PS}(P)$.

$Proof$. See Appendix D. ■

Proposition 4 For any $P,Q \in Pr^c$, $P \xrightarrow{\alpha} Q \Leftrightarrow P \models_{SL} \langle\alpha\rangle T^{PS}(Q) \Leftrightarrow T^{PS}(P) \vdash_{SL} \langle\alpha\rangle T^{PS}(Q)$.

$Proof$. See Appendix E. ■

Although Proposition 2 states that the inference system is not complete, Propositions 3 and 4 show that this inference system is complete with respect to structural congruence and labelled transition relation of processes.

G. Behavioral Equivalence Relation of Spatial Logic

In [9], we introduced a spatial logic called L, and proved that L gives a characterization of context bisimulation.

Definition 12 [9] Syntax of logic L

$$A ::= \neg A \mid A_1 \wedge A_2 \mid \langle a\langle\top\rangle\rangle\top \mid \langle\overline{a}\langle\top\rangle\rangle\top \mid \langle\tau\rangle A \mid A_1 \triangleright A_2.$$

It is easy to see that L is a sublogic of SL.

In [9], we proved the equivalence between $\sim_{Ct}$ and logical equivalence with respect to L.

Proposition 5 [9] For any $P,Q \in Pr^c$, $P \sim_{Ct} Q \Leftrightarrow$for any formula $A \in L$, $P \models_L A$ iff $Q \models_L A$.

Definition 13 A and B are behavioral equivalent with respect to L, written $A \sim_L B$, iff for any formula $C \in L$, $\models_{SL} A \to C$ iff $\models_{SL} B \to C$.

By Proposition 5, it is easy to get the following corollary, which characterizes $\sim_{Ct}$ by SL property.

Corollary 1 For any $P,Q \in Pr^c$, $P \sim_{Ct} Q \Leftrightarrow P \sim_L Q$.

Relation $\sim_L$ is a binary relation on spatial logical formulas. The above results show that $\sim_L$ gives a logical characterization of bisimulation when formulas are in the form of processes. Moreover, relation $\sim_L$ also gives a possibility to generialize bisimulation on processes to that on spatial logical formulas. Since we have discussed that spatial logical formulas can be regarded as specifications of processes, we may get a concept of bisimulation on specifications of processes based on $\sim_L$.

IV. Logics for Weak Semantics

In this section, we present a logic for weak semantics, named WL. Roughly speaking, in this logic, action temporal operators $\langle\tau\rangle$, $\langle a\langle A\rangle\rangle$, $\langle a[A]\rangle$ and $\langle\overline{a}\langle A\rangle\rangle$ in SL are replaced by the weak semantics version of operators $\langle\langle\varepsilon\rangle\rangle$, $\langle\langle a\langle A\rangle\rangle\rangle$, $\langle\langle a[A]\rangle\rangle$ and $\langle\langle\overline{a}\langle A\rangle\rangle\rangle$. Almost all definitions and results of SL can be generalized to WL.

A. Syntax and Semantics of Logic WL

Now we introduce a logic called WL, which is a weak semantics version of spatial logic.

Definition 14 Syntax of logic WL

$A ::= \top \mid \bot \mid \neg A \mid A_1 \wedge A_2 \mid \langle\langle\varepsilon\rangle\rangle A \mid \langle\langle a\langle A_1\rangle\rangle\rangle A_2 \mid \langle\langle a[A_1]\rangle\rangle A_2 \mid \langle\langle \overline{a}\langle A_1\rangle\rangle\rangle A_2 \mid 0 \mid X \mid a \odot X.A \mid A \setminus a \odot X \mid \overline{a}\langle A_1\rangle.A_2 \mid A \setminus \overline{a} \mid A_1 | A_2 \mid A_1 \triangleright A_2 \mid a\text{®}A \mid A \oslash a \mid (\mathbf{N}x)A \mid (\mathbf{N}X)A \mid (\ominus a)A \mid (\tilde{\ominus})A \mid a \neq b \mid X \mid \mu X.A(X)$ where X occurs positively in $A(X)$, i.e., all free occurrences of X fall under an even number of negations.

Definition 15 Semantics of logic WL

In the following, we use $\stackrel{\varepsilon}{\Longrightarrow}$ to abbreviate the reflexive and transitive closure of $\stackrel{\tau}{\longrightarrow}$, and use $\stackrel{\alpha}{\Longrightarrow}$ to abbreviate $\stackrel{\varepsilon}{\Longrightarrow}\stackrel{\alpha}{\longrightarrow}\stackrel{\varepsilon}{\Longrightarrow}$. By neglecting the tau action, we can get the weak semantics of processes. Semantics of formulas of WL can be the same as formulas of SL, except that semantics of operators $\langle\langle\varepsilon\rangle\rangle$, $\langle\langle a\langle A\rangle\rangle\rangle$, $\langle\langle a[A]\rangle\rangle$ and $\langle\langle\overline{a}\langle A\rangle\rangle\rangle$ should be defined as follows:

$[[\langle\langle\varepsilon\rangle\rangle A]]^e_{Pr} = \{P \mid \exists Q.\ P \stackrel{\varepsilon}{\Longrightarrow} Q \text{ and } Q \in [[A]]^e_{Pr}\}$;

$[[\langle\langle a\langle A_1\rangle\rangle\rangle A_2]]^e_{Pr} = \{P \mid \exists P_1, P_2.\ P \stackrel{a\langle P_1\rangle}{\Longrightarrow} P_2,\ P_1 \in [[A_1]]^e_{Pr} \text{ and } P_2 \in [[A_2]]^e_{Pr}\}$;

$[[\langle\langle a[A_1]\rangle\rangle A_2]]^e_{Pr} = \{P \mid \forall R, R \in [[A_1]]^e_{Pr}, \exists Q.\ P \stackrel{a\langle R\rangle}{\Longrightarrow} Q \text{ and } Q \in [[A_2]]^e_{Pr}\}$;

$[[\langle\langle\overline{a}\langle A_1\rangle\rangle\rangle A_2]]^e_{Pr} = \{P \mid \exists P_1, P_2.\ P \stackrel{(\nu\widetilde{b})\overline{a}\langle P_1\rangle}{\Longrightarrow} P_2, (\nu\widetilde{b})P_1 \in [[A_1]]^e_{Pr} \text{ and } P_2 \in [[A_2]]^e_{Pr}\}$.

B. Inference System of WL

The inference system of WL is similar to the inference system of SL except that any inference rule about action temporal operators $\langle\tau\rangle$, $\langle a\langle A\rangle\rangle$, $\langle a[A]\rangle$ and $\langle\overline{a}\langle A\rangle\rangle$ in SL is replaced by one of the following inference rules.

(1) $\langle\langle\alpha\rangle\rangle\bot \rightarrow \bot$;

(2) $\langle\langle\alpha\rangle\rangle A, A \rightarrow B \vdash \langle\langle\alpha\rangle\rangle B$;

(3) $\langle\langle\alpha\rangle\rangle A, A \rightarrow \langle\langle\varepsilon\rangle\rangle B \vdash \langle\langle\alpha\rangle\rangle B$;

(4) $\langle\langle\varepsilon\rangle\rangle A, A \rightarrow \langle\langle\alpha\rangle\rangle B \vdash \langle\langle\alpha\rangle\rangle B$;

(5) $\langle\langle\overline{a}\langle B\rangle\rangle\rangle A, C \rightarrow B \vdash \langle\langle\overline{a}\langle C\rangle\rangle\rangle A$;

(6) $\langle\langle a[B]\rangle\rangle A, C \rightarrow B \vdash \langle\langle a[C]\rangle\rangle A$;

(7) $\overline{a}\langle B\rangle.A \rightarrow \langle\langle\overline{a}\langle B\rangle\rangle\rangle A$;

(8) $(a \odot U.A \wedge ((\tilde{\ominus})B \leftrightarrow B)) \rightarrow \langle\langle a[B]\rangle\rangle A\{B/U\}$;

(9) $(\langle\langle\varepsilon\rangle\rangle A)|B \rightarrow \langle\langle\varepsilon\rangle\rangle(A|B)$;

(10) $(\langle\langle a\langle C\rangle\rangle\rangle A)|B \rightarrow \langle\langle a\langle C\rangle\rangle\rangle(A|B)$;

(11) $(((\ominus b_1, ..., \ominus b_n)B \leftrightarrow B) \wedge ((\tilde{\ominus})C \leftrightarrow C)) \rightarrow$
$(\langle\langle\langle\overline{a}\langle b_1\text{®}...b_n\text{®}C\rangle\rangle\rangle A)|B \rightarrow$
$\langle\langle\overline{a}\langle b_1\text{®}...b_n\text{®}C\rangle\rangle\rangle(A|B))$;

(12) $(((\ominus b_1, ..., \ominus b_n)B \leftrightarrow B) \wedge ((\tilde{\ominus})C \leftrightarrow C)) \rightarrow$
$(\langle\langle\langle\overline{a}\langle b_1\text{®}...b_n\text{®}C\rangle\rangle\rangle A)|\langle\langle a[C]\rangle\rangle B \rightarrow$
$\langle\langle\varepsilon\rangle\rangle b_1\text{®}...b_n\text{®}(A|B))$;

(13) $a\text{®}\langle\langle\varepsilon\rangle\rangle A \rightarrow \langle\langle\varepsilon\rangle\rangle a\text{®}A$;

(14) $(a \neq b \wedge (((\ominus a)B \wedge (\tilde{\ominus})B) \leftrightarrow B)) \rightarrow$
$(a\text{®}\langle\langle b\langle B\rangle\rangle\rangle A \rightarrow \langle\langle b\langle B\rangle\rangle\rangle a\text{®}A)$;

(15) $(\wedge_{i=1}^n a \neq b_i \wedge a \neq c \wedge ((\ominus a)B \leftrightarrow B) \wedge ((\tilde{\ominus})B \leftrightarrow B))$
$\rightarrow (a\text{®}\langle\langle\overline{c}\langle b_1\text{®}...b_n\text{®}B\rangle\rangle\rangle A \rightarrow$
$\langle\langle\overline{c}\langle b_1\text{®}...b_n\text{®}B\rangle\rangle\rangle a\text{®}A)$;

(16) $(a \neq b \wedge (\wedge_{i=1}^n b \neq c_i) \wedge\ (B \rightarrow \neg(\ominus b)\top) \wedge ((\tilde{\ominus})B \leftrightarrow B)) \rightarrow (b\text{®}\langle\langle\overline{a}\langle c_1\text{®}...c_n\text{®}B\rangle\rangle\rangle A \rightarrow$
$\langle\langle\overline{a}\langle b\text{®}c_1\text{®}...c_n\text{®}B\rangle\rangle\rangle A)$;

(17) $\langle\langle a[B]\rangle\rangle A \rightarrow \langle\langle a\langle B\rangle\rangle\rangle A$;

(18) $\langle\langle a\langle B\rangle\rangle\rangle A \rightarrow \langle\langle a[B]\rangle\rangle A$, *where B is syntactically a valid process in the higher order $\pi-$calculus.*

The above axioms and rules are weak semantics version of corresponding axioms and rules in SL. We name the above inference system of WL as W.

The soundness and incompleteness of inference system W of WL can be given similarly as the case of SL:

Proposition 6 $\Gamma \vdash_W A \Rightarrow \Gamma \models_{WL} A$

Proposition 7 There is no finite sound inference system AX such that $\Gamma \models_{WL} A \Rightarrow \Gamma \vdash_{AX} A$.

Similar to Proposition 4, we show that many steps transition relation $\stackrel{\alpha}{\Longrightarrow}$ is provable in WL.

Proposition 8 For any $P, Q \in Pr^c$, $P \stackrel{\alpha}{\Longrightarrow} Q \Leftrightarrow P \models_{WL} \langle\langle\alpha\rangle\rangle T^{PS}(Q) \Leftrightarrow T^{PS}(P) \vdash_{WL} \langle\langle\alpha\rangle\rangle T^{PS}(Q)$.

Since structural congruence and labelled transition relation are central concepts in the theory of processes, and they can be characterized in WL, the above propositions give a possible approach to reduce the theory of processes to the theory of spatial logic in the case of weak semantics.

V. Conclusions

Spatial logic was proposed to describe structural and behavioral properties of processes. There are many papers on spatial logic and process calculi. Spatial logic is related to some topics on process calculi, such as model checking, structural congruence, bisimulation and type system. In [16], a spatial logic for ambients calculus was studied, and a model checking algorithm was proposed. Some axioms of spatial logic were given, but the completeness of logic was not studied. Most spatial logics for concurrency are intensional [27], in the sense that they induce an equivalence that coincides with structural congruence, which is much finer than bisimilarity. In [22], Hirschkoff studied an extensional spatial logic. This logic only has spatial composition adjunct ($\triangleright$), revelation adjunct ($\oslash$), a simple temporal modality ($\langle\rangle$), and an operator for fresh name quantification. For π-calculus, this extensional spatial logic was proven to induce the same separative power as strong early bisimilarity. In [9], context bisimulation of higher order π-calculus was characterized by an extensional spatial logic.

 ISBN: 978-1-61208-277-6

In [5], a type system of processes based on spatial logic was given, where types are interpreted as formulas of spatial logic.

In this paper, we want to show that the theory of processes can be reduced to the theory of spatial logics. We firstly defined a logic SL, which comprises some temporal operators and spatial operators. We gave the inference system of SL and showed the soundness and incompleteness of SL. Furthermore, we showed that structural congruence and transition relation of higher order π-calculus can be reduced to the logical relation of SL formulas. We also showed that bisimulations in higher order π-calculus can be characterized by a sublogic of SL. At last, we propose a weak semantics version of SL, called WL. These results can be generalized to other process calculi. Since some important concepts of processes can be described in spatial logic, we think that this paper may give an approach of reducing the study of processes to the study of spatial logic. The further work for us is to develop a refinement calculus [23] for concurrent processes based on our spatial logic.

ACKNOWLEDGMENT

This work was supported by the Aviation Science Fund of China under Grant No. 20128052064 and the National Natural Science Foundation of China under Grant No. 60873025.

REFERENCES

[1] R. M. Amadio and M. Dam. Reasoning about Higher-order Processes. In TAPSOFT95, LNCS 915, 1995, pp. 202-216.

[2] R. M. Amadio. On the Reduction of CHOCS-Bisimulation to π-calculus Bisimulation. In CONCUR93, LNCS 715, 1993, pp. 112-126.

[3] A. Arnold and D. Niwinski. Rudiments of μ-calculus. Studies in Logic, Vol 146, North-Holland, 2001.

[4] M. Baldamus and J. Dingel. Modal Characterization of Weak Bisimulation for Higher-order Processes. In TAPSOFT97, LNCS 1214, 1997, pp. 285–296.

[5] L. Caires. Spatial-Behavioral Types for Concurrency and Resource Control in Distributed Systems. In Theoretical Computer Science 402(2-3), 2008, pp. 120-141.

[6] L. Caires. Logical Semantics of Types for Concurrency . In CALCO'07, LNCS, 2007, pp. 16-35.

[7] L. Caires, H. T. Vieira. Extensionality of Spatial Observations in Distributed Systems. In EXPRESS'2006, ENTCS, 2006.

[8] L. Caires. Behavioral and spatial observations in a logic for the π-calculus. In FOSSACS04, LNCS 2987, 2004, pp. 72-87.

[9] Z. Cao. A Spatial Logical Characterisation of Context Bisimulation. In Proceeding of ASIAN 2006, LNCS 4435, 2006, pp. 232-240.

[10] Z. Cao. More on bisimulations for higher-order π-calculus. In FOSSACS06, LNCS 3921, 2006, pp. 63-78.

[11] L. Caires and L. Cardelli. A Spatial Logic for Concurrency (Part II), Theoretical Computer Science, Vol 322(3), 2004, pp. 517-565.

[12] L. Caires and L. Cardelli. A Spatial Logic for Concurrency (Part I). Information and Computation, Vol 186(2), 2003, pp. 194-235.

[13] W. Charatonik, S. Dal Zilio, A. D. Gordon, S. Mukhopadhyay, and J.-M. Talbot. The complexity of model checking mobile ambients. In FoSSaCS'01, LNCS 2030, 2001, pp. 152-167.

[14] W. Charatonik, S. Dal Zilio, A. D. Gordon, S. Mukhopadhyay, J.-M. Talbot. Model Checking Mobile Ambients.

[15] L. Cardelli and A. Gordon. Logical Properties of Name Restriction. In Proc. of TLCA'01, LNCS 2044. 2001.

[16] L. Cardelli and A. Gordon. Anytime, Anywhere, Modal Logics for Mobile Ambients. In Proc. of POPL'00, 2000, pp. 365-377. ACM Press.

[17] G. Conforti and G. Ghelli. Decidability of Freshness ,Undecidability of Revelation. In : Proc. of FoSSaCS'04 , LNCS 2987. 2004.

[18] C. C. Chang. Model Theory. North-Holland, 1977.

[19] L. Caires1 and E. Lozes. Elimination of Quantifiers and Undecidability in Spatial Logics for Concurrency. In Theoretical Computer ScienceVolume 358 , Issue 2, August 2006, pp. 293 - 314.

[20] A. Jeffrey, J. Rathke. Contextual equivalence for higher-order π-calculus revisited. In Proceedings of Mathematical Foundations of Programming Semantics, Elsevier, 2003.

[21] D. Hirschkoff, E. Lozes, and D. Sangiorgi. Separability, Expressiveness and Decidability in the Ambient Logic. In Proc. of LICS'02, 2002, pp. 423-432, IEEE Computer Society.

[22] D. Hirschkoff. An Extensional Spatial Logic for Mobile Processes. CONCUR'04, LNCS 3170, 2004, pp. 325-339, Springer-Verlag.

[23] C. Morgan, P. Gardiner, K. Robision, and T. Vickers. On the Refinement Calculus. Springer-Verlag, 1994.

[24] R. Milner, J. Parrow, and D. Walker. Modal logics for mobile processes. Theoretical Computer Science, 114(1), 1993, pp.149-171.

[25] L. Gregory Meredith, Matthias Radestock: Namespace Logic: A Logic for a Reflective Higher-Order Calculus. TGC 2005, 2005, pp. 353-369.

[26] J.Parrow. An introduction to the π-calculus. In J. Bergstra, A. Ponse and S. Smolka editors, Handbook of Process Algebra, North-Holland, Amsterdam, 2001.

[27] D. Sangiorgi. Extensionality and Intensionality of the Ambient Logic. In Proc. of the 28th POPL, 2001, pp. 4-17. ACM Press,

[28] D. Sangiorgi. Bisimulation in higher-order calculi. Information and Computation, 131(2), 1996, pp. 141-178.

[29] D. Sangiorgi. Expressing mobility in process algebras: first-order and higher-order paradigms. Ph.D thesis, Department of Computer Science, University of Einburgh, 1992.

[30] C. Stirling. Modal Logics for Communicating Systems. Theoretical Computer Science, (49), 1987, pp. 311-347.

[31] B. Thomsen, Plain CHOCS: A second generation calculus for higher order processes. Acta Informatica, Vol 30, 1993, pp. 1-59.

Appendix A. Proof of Proposition 1

Proposition 1 $\Gamma \vdash_{SL} A \Rightarrow \Gamma \models_{SL} A$

$Proof$. It is enough by proving that every axiom and every inference rule of inference system is sound. We only discuss the following cases:

Case (1): Axiom $a\circledR((\ominus a)A|B) \leftrightarrow (\ominus a)A|a\circledR B$.

Suppose $P \in [[a\circledR((\ominus a)A|B)]]$, then $P \equiv (\nu a)(P_1|P_2)$, $a \notin fn(P_1)$, $P_1 \in [[A]]$ and $P_2 \in [[B]]$. Therefore we have $P \equiv (\nu a)(P_1|P_2) \equiv P_1|(\nu a)P_2$, $P \in [[(\ominus a)A|a\circledR B]]$. Hence $a\circledR((\ominus a)A|B) \leftrightarrow (\ominus a)A|a\circledR B$. The inverse case is similar.

Case (2): Axiom $a \neq b \rightarrow ((\ominus a)\overline{b}\langle B\rangle.A \leftrightarrow \overline{b}\langle(\ominus a)B\rangle.(\ominus a)A)$.

Suppose $a \neq b$ and $P \in [[(\ominus a)\overline{b}\langle B\rangle.A]]$, then $P \equiv \overline{b}\langle P_1\rangle.P_2$, $a \notin fn(P_1)$, $a \notin fn(P_2)$, $P_1 \in [[B]]$ and $P_2 \in [[A]]$. Therefore we have $P_1 \in [[(\ominus a)B]]$ and $P_2 \in [[(\ominus a)A]]$, $P \in [[\overline{b}\langle(\ominus a)B\rangle.(\ominus a)A)]]$. Hence $a \neq b \rightarrow ((\ominus a)\overline{b}\langle B\rangle.A \rightarrow \overline{b}\langle(\ominus a)B\rangle.(\ominus a)A)$. The inverse case is similar.

Case (3): Axiom $(A|A \triangleright B) \rightarrow B$.

Suppose $P \in [[A|A \triangleright B]]$, then $P \equiv P_1|P_2$, $P_1 \in [[A]]$ and $P_2 \in [[A \triangleright B]]$. Therefore, $P \equiv P_1|P_2 \in [[A|A \triangleright B]]$. Hence $(A|A \triangleright B) \rightarrow B$.

Case (4): Axiom $A \rightarrow (B \triangleright A|B)$.

Suppose $P \in [[A]]$, then for any $Q \in [[B]]$, $P|Q \in [[A|B]]$. Hence $A \rightarrow (B \triangleright A|B)$.

Case (5): Axiom $(((\ominus b_1, ..., \ominus b_n)B \leftrightarrow B) \wedge ((\tilde{\ominus})C \leftrightarrow C)) \rightarrow ((\langle\overline{a}\langle b_1\circledR ... b_n\circledR C\rangle\rangle A)|B \rightarrow \langle\overline{a}\langle b_1\circledR ... b_n\circledR C\rangle\rangle (A|B))$.

Suppose $P \in [[(\langle\overline{a}\langle b_1 ® ... b_n ® C\rangle\rangle A)|B]]$, then $P \equiv P_1|P_2$, $P_1 \xrightarrow{(\nu b_1,...,b_n)\overline{a}\langle Q\rangle} P_1'$, $P_1' \in [[A]]$, $P_2 \in [[B]]$ and $Q \in [[C]]$. Since $(\ominus b_1, ..., b_n)B \leftrightarrow B$, $\{b_1, ..., b_n\} \cap fn(P_2) = \emptyset$. Therefore we have $P_1|P_2 \xrightarrow{(\nu b_1,...,b_n)\overline{a}\langle Q\rangle} P_1'|P_2$. Hence $(((\ominus b_1, ..., \ominus b_n)B \leftrightarrow B) \wedge ((\tilde{\ominus})C \leftrightarrow C)) \rightarrow (((\langle\overline{a}\langle b_1 ® ... b_n ® C\rangle\rangle A)|B \rightarrow \langle\overline{a}\langle b_1 ® ... b_n ® C\rangle\rangle(A|B))$.

Case (6): Axiom $(((\ominus b_1, ..., \ominus b_n)B \leftrightarrow B) \wedge ((\tilde{\ominus})C \leftrightarrow C)) \rightarrow (((\langle\overline{a}\langle b_1 ® ... b_n ® C\rangle\rangle A)|\langle a[C]\rangle B \rightarrow \langle\tau\rangle b_1 ® ... b_n ® (A|B))$.

Suppose $P \in [[(\langle\overline{a}\langle b_1 ® ... b_n ® C\rangle\rangle A)|\langle a[C]\rangle B]]$, then $P \equiv P_1|P_2$, $P_1 \xrightarrow{(\nu b_1,...,b_n)\overline{a}\langle Q\rangle} P_1'$, $P_2 \xrightarrow{a\langle Q\rangle} P_2'$, $P_1' \in [[A]]$, $P_2' \in [[B]]$ and $Q \in [[C]]$. Since $(\ominus b_1, ..., b_n)B \leftrightarrow B$, $\{b_1, ..., b_n\} \cap fn(P_2') = \emptyset$. Therefore we have $P_1|P_2 \xrightarrow{\tau} (\nu b_1, ..., b_n)(P_1'|P_2')$. Hence $(((\ominus b_1, ..., \ominus b_n)B \leftrightarrow B) \wedge ((\tilde{\ominus})C \leftrightarrow C)) \rightarrow (((\langle\overline{a}\langle b_1 ® ... b_n ® C\rangle\rangle A)|\langle a[C]\rangle B \rightarrow \langle\tau\rangle b_1 ® ... b_n ® (A|B))$.

Case (7): Axiom $(\wedge_{i=1}^n a \neq b_i \wedge a \neq c \wedge ((\ominus a)B \leftrightarrow B) \wedge ((\tilde{\ominus})B \leftrightarrow B)) \rightarrow (a ® \langle\overline{c}\langle b_1 ® ... b_n ® B\rangle\rangle A \rightarrow \langle\overline{c}\langle b_1 ® ... b_n ® B\rangle\rangle a ® A)$.

Suppose $P \in [[a ® \langle\overline{c}\langle b_1 ® ... b_n ® B\rangle\rangle A]]$, then $P \equiv (\nu a)P_1$, $P_1 \xrightarrow{(\nu b_1,...,b_n)\overline{c}\langle Q\rangle} P_1'$, $Q \in [[B]]$, $P_1' \in [[A]]$. Since $\wedge_{i=1}^n a \neq b_i \wedge a \neq c \wedge ((\ominus a)B \leftrightarrow B) \wedge ((\tilde{\ominus})B \leftrightarrow B)$, $a \notin n(Q)$. Therefore we have $P \equiv (\nu a)P_1 \xrightarrow{(\nu b_1,...,b_n)\overline{c}\langle Q\rangle} (\nu a)P_1'$. Hence $(\wedge_{i=1}^n a \neq b_i \wedge a \neq c \wedge ((\ominus a)B \leftrightarrow B) \wedge ((\tilde{\ominus})B \leftrightarrow B)) \rightarrow (a ® \langle\overline{c}\langle b_1 ® ... b_n ® B\rangle\rangle A \rightarrow \langle\overline{c}\langle b_1 ® ... b_n ® B\rangle\rangle a ® A)$.

Case (8): Axiom $(a \neq b \wedge \wedge_{i=1}^n b \neq c_i \wedge (B \rightarrow \neg(\ominus b)\top) \wedge ((\tilde{\ominus})B \leftrightarrow B)) \rightarrow (b ® \langle\overline{a}\langle c_1 ® ... c_n ® B\rangle\rangle A \rightarrow \langle\overline{a}\langle b ® c_1 ® ... c_n ® B\rangle\rangle A)$.

Suppose $P \in [[b ® \langle\overline{a}\langle c_1 ® ... c_n ® B\rangle\rangle A]]$, then $P \equiv (\nu b)P_1$, $P_1 \xrightarrow{(\nu c_1,...,c_n)\overline{a}\langle Q\rangle} P_1'$, $Q \in [[B]]$, $P_1' \in [[A]]$. Since $a \neq b \wedge \wedge_{i=1}^n b \neq c_i \wedge (B \rightarrow \neg(\ominus b)\top) \wedge ((\tilde{\ominus})B \leftrightarrow B)$, $b \in fn(Q)$. Therefore we have $P \equiv (\nu b)P_1 \xrightarrow{(\nu b)(\nu c_1,...,c_n)\overline{a}\langle Q\rangle} P_1'$. Hence $(a \neq b \wedge \wedge_{i=1}^n b \neq c_i \wedge (B \rightarrow \neg(\ominus b)\top) \wedge ((\tilde{\ominus})B \leftrightarrow B)) \rightarrow (b ® \langle\overline{a}\langle c_1 ® ... c_n ® B\rangle\rangle A \rightarrow \langle\overline{a}\langle b ® c_1 ® ... c_n ® B\rangle\rangle A)$. ■

Appendix B. Proof of Proposition 2

Proposition 2 There is no finite sound inference system AX such that $\Gamma \models_{SL} A \Rightarrow \Gamma \vdash_{AX} A$.

Proof. Let $\Phi = \{\overline{a}\langle 0\rangle.\top, \overline{a}\langle 0\rangle.\overline{a}\langle b.0\rangle.\top, \overline{a}\langle 0\rangle.\overline{a}\langle b.0\rangle.\overline{a}\langle b.b.0\rangle.\top, \overline{a}\langle 0\rangle.\overline{a}\langle b.0\rangle.\overline{a}\langle b.b.0\rangle.\overline{a}\langle b.b.b.0\rangle.\top, ...\}$. It is easy to see that any finite subset of Φ can be satisfied in Pr, but Φ can not be satisfied in Pr. Suppose it is not true, let P satisfies Φ. By Lemma 1, there exists n, such that $d(P) = n$. But for any n, there exists φ_n in Φ such that for any P satisfying φ_n, $d(P) > n$. This contradicts the assumption. Therefore Φ can not be satisfied in Pr.

Suppose there is a finite inference system such that $\Gamma \models_{SL} A \Rightarrow \Gamma \vdash_{SL} A$. Since Φ can not be be satisfied in Pr, we have $\Phi \models_{SL} \bot$. By the assumption, $\Phi \vdash_{SL} \bot$. Hence there is a proof from Φ to $\bot$ in SL. Since proof is a finite formula sequence, there is finite many formulas φ_i in Φ occur in the proof. Therefore we have $\wedge\Phi_i \vdash_{SL} \bot$, where $\Phi_i = \{\varphi_i \mid \varphi_i$ is in the proof$\}$. Then by the soundness of inference system of SL, we have that Φ_i is not satisfiable. Since Φ_i is a finite subset of Φ, this contradicts the assumption. Therefore SL have no finite complete inference system. ■

Appendix C. Proof of Lemma 2

Lemma 2 $\vdash_{SL} A|!A \leftrightarrow !A$.

Proof : Since by the inference system, $\vdash_{SL} S(\mu X.S(X)) \rightarrow \mu X.S(X)$, we have $\neg\mu X.S(X) \rightarrow \neg S(\mu X.S(X))$. Let $S(X) = \neg(A|\neg X)$, then $\neg\mu X.S(X) = \neg\mu X.\neg(A|\neg X) = !A$, $\neg S(\mu X.S(X)) = A|\neg\mu X.\neg(A|\neg X) = A|!A$. Therefore we get $\vdash_{SL} !A \rightarrow A|!A$.

Since by the inference system, $\vdash_{SL} !A \rightarrow A|!A$, we have $\vdash_{SL} \neg(A|A|!A) \rightarrow \neg(A|!A)$. Let $T(X) = \neg(A|\neg X)$, then $T(\neg(A|!A)) = \neg(A|A|!A)$. Since $\vdash_{SL} T(\neg(A|!A)) \rightarrow \neg(A|!A)$, by the inference system, we have $\vdash_{SL} \mu X.T(X) \rightarrow \neg(A|!A)$. Furthermore, $\mu X.T(X) = \mu X.\neg(A|\neg X) = \neg !A$, hence $\vdash_{SL} \neg !A \rightarrow \neg(A|!A)$, we have $\vdash_{SL} A|!A \rightarrow !A$. ■

Appendix D. Proof of Proposition 3

Proposition 3 For any $P, Q \in Pr^c$, $P \equiv Q \Leftrightarrow P \models_{SL} T^{PS}(Q)$ and $Q \models_{SL} T^{PS}(P) \Leftrightarrow T^{PS}(P) \vdash_{SL} T^{PS}(Q)$ and $T^{PS}(Q) \vdash_{SL} T^{PS}(P)$.

Proof. It is trivial by the definition that $P \equiv Q \Leftrightarrow P \models_{SL} T^{PS}(Q)$ and $Q \models_{SL} T^{PS}(P)$. By the soundness, $T^{PS}(P) \vdash_{SL} T^{PS}(Q) \Rightarrow P \models_{SL} T^{PS}(Q)$. We only need to prove $P \equiv Q \Rightarrow T^{PS}(P) \vdash_{SL} T^{PS}(Q)$ and $T^{PS}(Q) \vdash_{SL} T^{PS}(P)$.

We only discuss the following cases, other cases are similar or trivial:

Case (1): $(\nu m)(\nu n)P \equiv (\nu n)(\nu m)P$: Since $m ® n ® T^{PS}(P) \leftrightarrow n ® m ® T^{PS}(P)$, we have $m ® n ® T^{PS}(P) \vdash_{SL} n ® m ® T^{PS}(P)$. The inverse case is similar.

Case (2): $(\nu a)(P|Q) \equiv P|(\nu a)Q$ if $a \notin fn(P)$: Since $a \notin fn(P)$, $(\ominus a)T^{PS}(P) \leftrightarrow T^{PS}(P)$. Furthermore, since $a ® ((\ominus a)T^{PS}(P)|T^{PS}(Q)) \leftrightarrow (\ominus a)T^{PS}(P)|a ® T^{PS}(Q)$, we have $a ® (T^{PS}(P)|T^{PS}(Q)) \vdash_{SL} T^{PS}(P)|a ® T^{PS}(Q)$. The inverse case is similar. ■

Appendix E. Proof of Proposition 4

Proposition 4 For any $P, Q \in Pr^c$, $P \xrightarrow{\alpha} Q \Leftrightarrow P \models_{SL} \langle\alpha\rangle T^{PS}(Q) \Leftrightarrow T^{PS}(P) \vdash_{SL} \langle\alpha\rangle T^{PS}(Q)$.

Proof. It is trivial by the definition that $P \xrightarrow{\alpha} Q \Leftrightarrow P \models_{SL} \langle\alpha\rangle T^{PS}(Q)$. By the soundness, $T^{PS}(P) \vdash_{SL} \langle\alpha\rangle T^{PS}(Q) \Rightarrow P \models_{SL} \langle\alpha\rangle T^{PS}(Q)$. We only need to prove $P \xrightarrow{\alpha} Q \Rightarrow P \vdash_{SL} \langle\alpha\rangle T^{PS}(P)$.

We apply the induction on the length of the inference tree of $P \xrightarrow{\alpha} Q$:

Case (1): if the length is 0, then $P \xrightarrow{\alpha} Q$ is in the form of $\overline{a}\langle E\rangle.K \xrightarrow{\overline{a}\langle E\rangle} K$ or $a(U).K \xrightarrow{a\langle E\rangle} K\{E/U\}$.

Subcase (a): $\overline{a}\langle E\rangle.K \xrightarrow{\overline{a}\langle E\rangle} K$: Since $\overline{a}\langle E\rangle.T^{PS}(K) \rightarrow \langle\overline{a}\langle E\rangle\rangle T^{PS}(K)$, we have $\overline{a}\langle E\rangle.T^{PS}(K) \vdash_{SL} \langle\overline{a}\langle E\rangle\rangle T^{PS}(K)$.

Subcase (b): $a(U).K \xrightarrow{a\langle E\rangle} K\{E/U\}$: Since $(a(U).T^{PS}(K) \wedge ((\tilde{\ominus})T^{PS}(E) \leftrightarrow T^{PS}(E))) \rightarrow \langle a[T^{PS}(E)]\rangle T^{PS}(K)\{T^{PS}(E)/U\}$, we have $a(U).T^{PS}(K) \vdash_{SL} \langle a[T^{PS}(E)]\rangle T^{PS}(K)\{T^{PS}(E)/U\}$.

Case (2): Assume the claim holds if length is n, now we discuss the case that length is $n+1$.

Subcase (a): $\dfrac{M \xrightarrow{(\nu\widetilde{b})\overline{a}\langle E\rangle} M' \quad N \xrightarrow{a\langle E\rangle} N'}{M|N \xrightarrow{\tau} (\nu\widetilde{b})(M'|N')} \widetilde{b} \cap fn(N) = \emptyset.$

Since $M \xrightarrow{(\nu\widetilde{b})\overline{a}\langle E\rangle} M'$, $N \xrightarrow{a\langle E\rangle} N'$, and $\widetilde{b} \cap fn(N) = \emptyset$, we have $T^{PS}(M) \rightarrow \langle \overline{a}\langle \widetilde{b}\circledR T^{PS}(E)\rangle\rangle T^{PS}(M')$, $T^{PS}(N) \rightarrow \langle a[T^{PS}(E)]\rangle T^{PS}(N')$ and $(\ominus b_1, ..., b_n)T^{PS}(E) \leftrightarrow T^{PS}(E)$. By the axiom: $(((\ominus b_1, ..., b_n)T^{PS}(N) \leftrightarrow T^{PS}(N)) \wedge (\tilde{\ominus})T^{PS}(E)) \rightarrow ((\langle \overline{a}\langle b_1\circledR ... b_n\circledR T^{PS}(E)\rangle\rangle T^{PS}(M))|\langle a[T^{PS}(E)]\rangle T^{PS}(N) \rightarrow \langle\tau\rangle b_1\circledR ... b_n\circledR(T^{PS}(M)|T^{PS}(N)))$, we have $P \equiv T^{PS}(M)|T^{PS}(N) \vdash_{SL} \langle\tau\rangle b_1\circledR ... b_n\circledR (T^{PS}(M')|T^{PS}(N'))$.

Subcase (b): $\dfrac{M \xrightarrow{b\langle E\rangle} M'}{(\nu a)M \xrightarrow{b\langle E\rangle} (\nu a)M'} a \notin n(\alpha).$

Since $M \xrightarrow{b\langle E\rangle} M'$ and $a \notin n(b\langle E\rangle)$, we have $T^{PS}(M) \rightarrow \langle b\langle T^{PS}(E)\rangle\rangle T^{PS}(M')$ and $((\ominus a)T^{PS}(E) \wedge (\tilde{\ominus})T^{PS}(E)) \leftrightarrow T^{PS}(E)$. By the axiom $(a \neq b \wedge ((\ominus a)T^{PS}(E) \wedge (\tilde{\ominus})T^{PS}(E)) \leftrightarrow T^{PS}(E)) \rightarrow (a\circledR \langle b\langle T^{PS}(E)\rangle\rangle T^{PS}(M) \rightarrow \langle b\langle T^{PS}(E)\rangle\rangle a\circledR T^{PS}(M))$, we have $T^{PS}(P) = a\circledR T^{PS}(M) \vdash_{SL} a\circledR\langle b\langle T^{PS}(E)\rangle\rangle T^{PS}(M) \vdash_{SL} \langle b\langle T^{PS}(E)\rangle\rangle a\circledR T^{PS}(M))$.

Subcase (c): $\dfrac{M \xrightarrow{(\nu\widetilde{c})\overline{a}\langle E\rangle} M'}{(\nu b)M \xrightarrow{(\nu b,\widetilde{c})\overline{a}\langle E\rangle} M'} a \neq b,\ b \in fn(E) - \widetilde{c}.$

Since $M \xrightarrow{(\nu\widetilde{c})\overline{a}\langle E\rangle} M'$ and $a \neq b$, $b \in fn(E) - \widetilde{c}$, we have $T^{PS}(M) \rightarrow \langle \overline{a}\langle \widetilde{c}\circledR T^{PS}(E)\rangle\rangle T^{PS}(M')$ and $a \neq b \wedge \wedge_{i=1}^{n} b \neq c_i \wedge (B \rightarrow \neg(\ominus b)\top)$. By the axiom $(a \neq b \wedge \wedge_{i=1}^{n} b \neq c_i \wedge (E \rightarrow \neg(\ominus b)\top) \wedge ((\tilde{\ominus})E \leftrightarrow E)) \rightarrow (b\circledR\langle \overline{a}\langle c_1\circledR ... c_n\circledR T^{PS}(E)\rangle\rangle T^{PS}(M') \rightarrow \langle \overline{a}\langle b\circledR c_1 \circledR ... c_n\circledR T^{PS}(E)\rangle\rangle T^{PS}(M'))$, we have $T^{PS}(P) = b\circledR T^{PS}(M) \vdash_{SL} (b\circledR\langle \overline{a}\langle c_1\circledR ... c_n\circledR T^{PS}(E)\rangle\rangle T^{PS}(M') \vdash_{SL} \langle \overline{a}\langle b\circledR c_1\circledR ... c_n\circledR T^{PS}(E)\rangle\rangle T^{PS}(M')$. ■