A Theory of Communicating Sequential Processes in Coq

PROPOSTA DE TRABALHO DE GRADUAÇÃO

Aluno: Carlos Alberto da Silva Carvalho de Freitas (casc2@cin.ufpe.br)
Orientador: Gustavo Henrique Porto de Carvalho (ghpc@cin.ufpe.br)
Área: Engenharia de Software

July 2020
Abstract

Theories of concurrency such as Communicating Sequential Processes (CSP) allow system specifications to be expressed clearly and analyzed with precision. However, the state explosion problem, common to model checkers in general, is a real constraint when attempting to verify system properties for large systems. An alternative is to ensure these properties via proof development. This work will provide an approach on how we can develop a theory of CSP in the Coq proof assistant, and evaluate how this theory compares to other theorem prover-based frameworks for the process algebra CSP. We will implement an infrastructure for declaring syntactically and semantically correct CSP specifications in Coq, along with native support for process representation through Labelled Transition Systems (LTSs), in addition to traces refinement analysis.
Resumo

Teorias de concorrência tais como *Communicating Sequential Processes* (CSP) permitem que especificações de sistemas sejam descritas com clareza e analisadas com precisão. No entanto, o problema da explosão de estados, comum aos verificadores de modelo em geral, é uma limitação real na tentativa de verificar propriedades de um sistema complexo. Uma alternativa é garantir essas propriedades através do desenvolvimento de provas. Este trabalho fornecerá uma abordagem sobre como se pode desenvolver uma teoria de CSP no assistente de provas Coq, além de compará-la com outros frameworks baseados em provadores de teoremas para a álgebra de processos CSP. Portanto, será implementada uma infraestrutura para declarar especificações sintatica e semanticamente corretas de CSP em Coq, juntamente com um suporte nativo para a representação de processos por meio de Sistemas de Transições Rotuladas (LTSs), além de análise de refinamento no modelo de *traces*. 
Introduction

Concurrency is an attribute of any system that allows multiple components to perform operations at the same time. The understanding of this property is essential in modern programming because major areas, such as distributed and real-time systems, rely on this concept to work properly. As a result, the variety of applications enabled by the concurrency feature is broad: aircraft and industrial control systems, routing algorithms, peer-to-peer networks, client-server applications and parallel computation, to name a few.

Since concurrent systems may have parts that execute in parallel, the combination of ways in which these parts can interact raises the complexity in designing such systems. Phenomena like deadlock, livelock, nondeterminism and race condition can emerge from these interactions, so these issues must be addressed in order to avoid undesired behaviour. Typically, testing cannot provide enough evidence to guarantee properties such as deadlock freedom, divergence freedom and determinism for a given system.

That being said, CSP (a theory for Communicating Sequential Processes) [1] [2] introduces a convenient notation that allows systems to be described in a clear and accurate way. More than that, it has an underlying theory that enables designs to be analysed and proven correct with respect to desired properties. The FDR (Failures-Divergence Refinement) tool is a model checker for CSP responsible for making this process algebra a practical tool for specification, analysis and verification of systems. System analysis is achieved by allowing the user to make assertions about processes and then exploring every possible behaviour, if necessary, to check the truthfulness of the assertions made.

Although it is undeniable that FDR is a useful tool in the analysis of systems described in CSP, it has a limitation common to standard model checkers in general: the state explosion problem. An alternative way for deciding whether a system meets its specification is by proof development. Examples of this different approach are CSP-Prover [4] and Isabelle/UTP [5], both frameworks based on the theorem prover Isabelle. Nevertheless, to the best of our knowledge, there is not a theory for CSP in the Coq proof assistant [3] yet. Considering that, the main research question of this work is the following: how could we develop a theory of CSP in Coq, exploiting the main advantages of this proof assistant?
Objectives

The main objective (MO) of this work is to define in Coq a theory for concurrent systems, based on a limited scope of the process algebra CSP. This objective is unfolded into the following specific objectives (SO):

- **SO1**: study CSP and frameworks based on this process algebra.
- **SO2**: define a syntax for CSP in Coq, based on a restricted version of the CSP$_M$ language (machine readable language for CSP).
- **SO3**: provide support for the LTS-based (Labelled Transition System) representation, considering the Structural Operational Semantics (SOS) of CSP.
- **SO4**: make use of the QuickChick tool to search for counterexamples of the traces refinement relation.
Methodology

Initially, we will study the process algebra CSP, as well as theorem prover-based frameworks for this algebra. Afterwards, we will define a syntax for CSP in Coq. For practical reasons, only a representative set of CSP operators will be considered for this project. In order to achieve such goal, we will introduce the fundamental entities of CSP – processes and events – as inductive types. Once the constructors for these two types are available, the next step will be to define a CSP$_M$-like notation for the defined abstract syntax, so the users can write specifications in a convenient and familiar way.

After defining in Coq the abstract and concrete syntax of CSP$_M$, we will define the language semantics. Regarding the operational semantics, the Structural Operational Semantics (SOS) will be the selected category to describe the meaning of syntactically correct CSP expressions. Moreover, we will introduce yet another way of expressing the behaviour of a process, and that is through a denotational model: the traces model. Both relational and computable definitions of the trace relation will be supplied by this project.

Additionally, the support for representing a CSP process in terms of a Labelled Transition System (LTS) will also be available in this work. To accomplish this objective, there will be a relational and a computable definition of the LTS representation based on the SOS. For viewing purposes, it will be possible to generate a graph in the Dot notation from a LTS, which later can be converted into an image using the GraphViz software.

Furthermore, we will also support refinement analysis considering the traces model. Automation of refinement analysis is going to be supported via the QuickChick library. We will implement a generator that takes a process and returns a valid random trace for it, and a checker that verifies whether the trace relation holds between a process and a list of events. Equipped with such definitions, the QuickChick library comes in handy for putting the traces refinement proposition to the test and potentially finding counterexamples if a process fails to refine another.

Therefore, the main activities of this work is summarised as follows:

- **SO1**: study CSP and frameworks based on this process algebra.
  - A1: carry out a literature review.
- **SO2**: define a reduced CSP syntax in Coq.
  - A2: define the abstract syntax for a subset of CSP operators.
  - A3: define a concrete syntax as close as possible to CSP$_M$ language.
- **SO3**: provide support for LTS representation based on the SOS of CSP.
  - A4: define the SOS via inference rules.
- A5: provide a relational and a computable definition of LTSs.
- A6: generate a graph in Dot language from a LTS.
- SO4: provide support for traces refinement analysis via QuickChick.
  - A7: define the notions of trace and trace refinement.
  - A8: implement a generator of valid traces and a checker for refinement.

In addition to the aforementioned activities, it is also part of this work writing a monograph reporting the main achievements (activity A9), besides preparing a final presentation (activity A10).
## Schedule

<table>
<thead>
<tr>
<th>Activity</th>
<th>March</th>
<th>April</th>
<th>May</th>
<th>June</th>
<th>July</th>
<th>August</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>A2</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A3</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A4</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>A5</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>A6</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>A7</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A8</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>A9</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>A10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
References

Possible Members of the Examination Committee

Prof. Juliano Iyoda

Prof. Alexandre Mota
Signatures

Recife, 07 de Julho de 2020

Carlos Alberto da Silva Carvalho de Freitas
(Student)

Gustavo Henrique Porto de Carvalho
(Supervisor)