Carlos Alberto da Silva Carvalho de Freitas

A Theory of Communicating Sequential Processes in Coq

Recife

2020

Carlos Alberto da Silva Carvalho de Freitas

A Theory of Communicating Sequential Processes in Coq

Trabalho apresentado como requisito parcial para a conclusão da graduação em Engenharia da Computação do Centro de Informática da Universidade Federal de Pernambuco.

Universidade Federal de Pernambuco

Centro de Informática

Graduação em Engenharia da Computação

Supervisor: Gustavo Henrique Porto de Carvalho

Recife

2020

Acknowledgements

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.

Keywords:

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.

Palavras-chave:

Contents

1 INTRODUCTION 6
1.1 Objectives 6
1.2 An overview of CSP_{Coq} 7
1.3 Main contributions 8
1.4 Document structure 8

2 BACKGROUND 10
2.1 Communicating sequential processes 10
2.1.1 Structured operational semantics 10
2.1.2 Traces refinement 10
2.2 The Coq proof assistant 10
2.2.1 Building proofs 10
2.2.2 The tactics language 10
2.3 QuickChick 10

3 A THEORY FOR CSP IN COQ 11
3.1 Syntax 11
3.2 Structured operational semantics 11
3.3 Labelled transition systems 11
3.4 Traces refinement 11

4 CONCLUSIONS 12
4.1 Related work 12
4.2 Future work 12

BIBLIOGRAPHY 13

1 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 behavior. 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) 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 behavior, 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 and Isabelle/UTP, 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 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?

1.1 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.

1.2 An overview of CSP_Coq

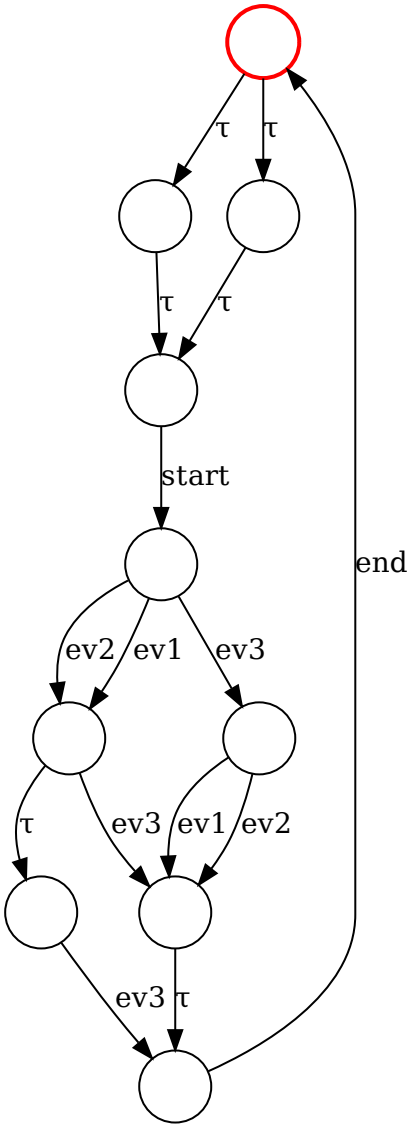
```
Require Import CSP.lts.
Require Import CSP.semantics_sos.
Require Import CSP.semantics_trace.
Require Import CSP.syntax.
Require Import Lists.List.
Require Import Lists.ListSet.
Require Import Strings.String.
Import ListNotations.

Local Open Scope string.

Definition example : specification.
Proof.
  solve_spec_ctx_rules (
    Build_Spec
    [
      Channel {{"start", "ev1", "ev2", "ev3", "end"}}
    ]
    [
      "P" ::= "start" -> ("ev1" -> SKIP [] "ev2" -> SKIP) ;; "end" -> ProcRef "P"
      ; "Q" ::= "start" -> "ev3" -> "end" -> ProcRef "Q"
      ; "R" ::= ProcRef "P" [| {{"start", "end"}} |] ProcRef "Q"
    ]
  ).
Defined.

Compute generate_dot (compute_ltsR example "R" 100).
```

Figure 1 – The "R" process LTS.



Source:

1.3 Main contributions

1.4 Document structure

Apart from this introductory chapter, in which we discuss about the motivation behind this work and its main objective, and also take a quick look at an example that illustrates what can be done using the framework developed, this monograph contains three more chapters. The content of these chapters are detailed bellow:

Chapter 2 Discusses fundamental concepts such as CSP theory, SOS approach, trace refinement and LTS representation. Moreover, this chapter introduces the Coq proof assistant and its functional language Gallina, along with an introduction to proof development (tactics) and the Ltac language inside this tool, which gives support for developing tactic macros.

Chapter 3 Provides an in-depth look at the implementation of CSP_{Coq}, including its abstract and concrete syntax, and language semantics. Furthermore, the LTS process representation support, using the GraphViz software, is also detailed in this chapter.

Chapter 4 Concludes this monograph by presenting a comparison between the infrastructure described in this work and other interactive theorem provers based on CSP. It also addresses possible topics for future work.

2 Background

2.1 Communicating sequential processes

2.1.1 Structured operational semantics

2.1.2 Traces refinement

2.2 The Coq proof assistant

2.2.1 Building proofs

2.2.2 The tactics language

2.3 QuickChick

3 A theory for CSP in Coq

3.1 Syntax

3.2 Structured operational semantics

3.3 Labelled transition systems

3.4 Traces refinement

4 Conclusions

4.1 Related work

4.2 Future work

Bibliography