Saarland University

Faculty of Natural Sciences and Technology I
Department of Computer Science

Encodings of Bounded Synthesis of Distributed Systems

Bachelor's Thesis

Jan Eric Baumeister

Supervisor

Prof. Bernd Finkbeiner, Ph.D.

Advisor

Leander Tentrup, M.Sc.

Reviewers

Prof. Bernd Finkbeiner, Ph.D.
Prof. Dr.-Ing. Holger Hermanns

submitted on

October 23,2017

Eidesstattliche Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement in Lieu of an Oath

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

______________________________ ______________________________

(Ort / Place, Datum / Date) (Unterschrift / Signature)

Abstract

Reactive synthesis is one approach to get correct programs from a formal specification, e.g. given in linear-time temporal logic (LTL). This approach constructs correct-by-design implementations directly from those specifications and frees the programmer from writing an implementation. To cope with the drawback of the high complexity, bounded synthesis was introduced by Finkbeiner and Schewe in 2012 [12]. This approach bounds the size of the implementation, for example in the number of states. The original implementation constructs an SMT query that is satisfiable iff a realizing implementation of that size exists. Later Faymonville et al. [8] proposed different encodings to (quantified) propositional logics.

In this thesis we explain the bounded synthesis approach for single process and distributed synthesis. Based on the encodings introduced by Faymonville et al. [8] we define three new encodings to solve the distributed bounded synthesis problem and proof their functional correctness. We conclude by comparing the different encodings in an experimental evaluation section.

Acknowledgement

I am deeply grateful to Prof. Finkbeiner for offering me this interesting thesis topic. I would like to thank my advisor Leander Tentrup. He gave me constructive answers to all my questions and helped me to complete this thesis. Another grateful thank to Prof. Finkbeiner and Prof. Hermanns for reviewing this thesis. Last but not least I would like to thank my family and friends for their ongoing support.

Contents

1 Introduction **1**

2 Preliminary **5**
2.1 LTL . . . 5
2.2 Transition System . . . 7
2.3 Automata . . . 9
2.4 Run Graph . . . 12
2.5 Quantified Boolean Formulas . . . 12
2.6 Dependency Quantified Boolean Formulas . . . 15

3 Single Process Synthesis **17**
3.1 Synthesis . . . 17
3.2 Bounded Synthesis . . . 17
3.3 Translating LTL to Universal co-Büchi Automata . . . 18
3.4 Verification . . . 19

4 Encodings for Single Processes **21**
4.1 Verification . . . 21
4.2 SAT Propositional Encoding: Explicit Encoding . . . 22
4.3 QBF Encoding: Input-symbolic Encoding . . . 25

5 Distributed Synthesis **29**
5.1 Architecture . . . 29
5.2 Distributed Synthesis . . . 31
5.3 Composition of Transition Systems . . . 31
5.4 Challenges with Distributed Systems . . . 32

6 Encodings for Distributed Synthesis **35**
6.1 Verification . . . 35
6.2 SAT Propositional Encoding: Explicit Encoding . . . 36
6.3 QBF Encoding: Input-symbolic Encoding . . . 41
6.4 DQBF Encoding: Input-symbolic Encoding . . . 47

7 Implementation & Results **53**
7.1 BoSy . . . 53
7.2 Evaluation . . . 55

8 Related Work **61**

9 Conclusion & Future Work **63**
References 65

Bibliography **65**

10 Appendix **69**
10.1 Architectures 69
10.2 Example for the SAT Propositional Constraint System 75
10.3 Example for the QBF Constraint System 76
10.4 Example for the DQBF Constraint System 77

List of Figures

1.1 Abstract Idea of Synthesis 3

2.1 A Virtual Representation of the Temporal Operators of LTL 6
2.2 Example of a Moore and Mealy Transition System 9
2.3 Example of a Universal co-Büchi Automaton 10
2.4 Example of a Run Graph 12
2.5 Two BDT Representing the Skolem Functions for the Respective QBFs below . . 14
2.6 Two BDT Representing the Skolem Functions for the Respective DQBFs below . 16

3.1 General Concept to Solve Bounded Synthesis 18
3.2 Example of a Run Graph with a Valid Annotation Function 20

4.1 General Concept of the Encoding Used for the Verification 22
4.2 Structure of the Input-Symbolic Encoding 25

5.1 Three Examples for Architectures 30
5.2 Composition of two Transition Systems 32

6.1 Example to Show the Need of the Composition Constraint 37
6.2 Three Examples for Architectures with a Total Order 46

7.1 Test Results on Architectures where the QBF Encoding is Applicable 58
7.2 Test Results on Architectures where the QBF Encoding is not Applicable 58

8.1 Representation of the Constraints Introduced by Guthoff 62

List of Tables

7.1 Results of the Evaluation in seconds 60

1 Introduction

In computer science, there are essentially two ways to relate the formal specification of a program to an actual model or implementation. The first one is verification which gets as input a logical specification, like formulas in linear-time temporal logic (LTL) and a model of an implementation, for example transition systems. We verify if the given implementation satisfies the specification. The drawback of this method is that an implementation has to be written. Therefore Church introduced in 1957 the synthesis problem [5] which describes the second way to relate a specification to an implementation. It gets as input only the specification and produces a model of an implementation, which satisfies the given specification. This frees the programmer from writing programs. As Finkbeiner [10] pointed out, the reactive synthesis problem is one of the most intriguing challenges in the theory of reactive systems. Reactive synthesis is based on reactive systems, which continuously interact with the environment representing for example the inputs of a human. At first, this problem was only a theoretical problem. However, the development of faster algorithms in recent years made practical applications possible, like in hardware, device drivers and robotics.

The synthesis problem can be seen as a game between the system, trying to fulfill the specification and the environment, trying to produce an error. If there is a winning strategy for the system, this strategy defines the implementation of the system to fulfill the specification. These games are played on a finite game area, that can be expressed in different finite formulas and the players are represented as the infinite input. Historically the development to solve the synthesis problem happened in three waves. The first wave started after Church introduced the problem in 1957. The first basic algorithms were developed, solving a system specification given by a formula in monadic second order logic (MSO) and had a nonelementary complexity. The second wave started with the introduction of temporal logic. As the translation from LTL to deterministic automata is double exponential and LTL based model checking is PSPACE, LTL specification become an industrial technique in the hardware industry. Therefore the second wave started with the development of algorithms for the common linear-time and branching-time temporal logics. The third wave was motivated by the advances in the performance of automatic verification during the 1980s and 1990s. The goal of this wave was to develop practical algorithms, for example by reducing the complexity of the synthesis problem with synthesis-friendly logics like generalized reactivity (GR) or practical restrictions of the classical synthesis problem, like bounded synthesis.

In the synthesis problem, find an implementation for a simple architecture, with one process and a given input and output set. Therefore this is also called single process synthesis. The distributed synthesis problem deals with distributed architectures, which consists of

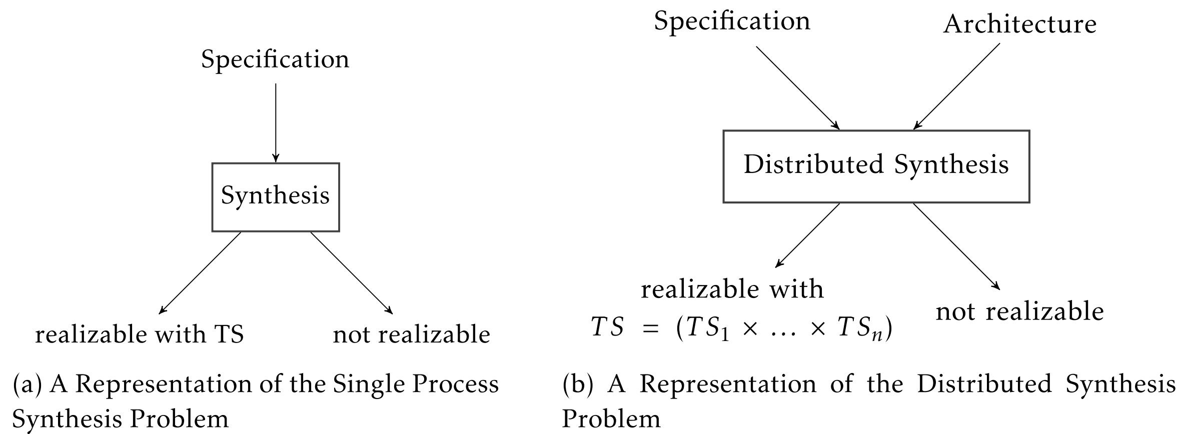
more than one process, where each process could have different information. Information is in this sense the different input and output set, where an input variable can be the output variable of the environment or of a system process. Therefore the distributed synthesis find a set of transition systems, or another model of an implementation, one for each process, such that the composition of this set satisfies the given specification. In general the distributed synthesis problem is undecidable, even for simple architectures like two independent processes. This was proven by Pnueli and Rosner [22], where the distributed synthesis problem was introduced. Distributed synthesis becomes more important with the development of hardware, which uses distributed architectures.

Finkbeiner and Schewe [12] introduce bounded synthesis. The bounded synthesis approach restricts the transition system that satisfies the given specification, by bounding the size of the transition system. If no solution can be found the state space is increased. This approach improves the performance of the synthesis problem because the smallest solution can be found. The restriction can be translated to the distributed synthesis. This approach can be transferred to the distributed synthesis, the distributed bounded synthesis. Finkbeiner and Schewe [12] construct a SMT query that is satisfiable if and only if a realizing implementation of that size exists. This original implementation employed a SMT-solver for single process and distributed synthesis.

Faymonville et al. [8] introduce new encodings as propositional formulas for the single process synthesis. The basic encoding is the SAT propositional encoding, the encoding into boolean formulas. It is also called the explicit encoding, because each transition from one state with a given input to another state, each output from a state, ... becomes a variable that is true if for example the transition from one state to another with a given input. The second encoding is represented with quantified boolean formulas (QBF) and is called the input-symbolic encoding. In this encoding the input is represented symbolically by a universal quantifier. The variables are bounded by an input and do not have written explicit in the index. The state- and input-symbolic encoding is realized with dependency quantified boolean formulas (DQBF). In this encoding inputs and states of the the transition system are symbolical represented by a universal quantifier and afterwards the states of the automaton too. The experimental evaluation of the paper shows that the SAT propositional and QBF encoding are faster than the original SMT encoding.

The goal of this Bachelor thesis is to translate the explicit and input-symbolic encoding to distributed synthesis. Afterwards there is a experimental evaluation, that is compared with the SMT encoding described in the bachelor thesis by Guthoff [17].

Therefore this thesis first explains the single process synthesis, Figure 1.1a gives an overview about the problem. The specification is the input of this problem and the output is the transition system satisfying the specification if such a solution exists. The input is described as an LTL formula and will be translated into a universal co-Büchi automaton, for reasons of simplicity. These two specification languages are part of chapter 2, where the preliminaries are explained. The output, the transition system, is also part of this chapter. In chapter 3 the single process synthesis is introduced formally, based on the models we explained in chapter 2. In the next chapter 4, the different encodings to solve the single process synthesis, based on the concept in chapter 3 and the logics explained in chapter 2, are introduced. With this chapter the explanation of single process synthesis ends and the distributed syn-



(a) A Representation of the Single Process Synthesis Problem

(b) A Representation of the Distributed Synthesis Problem

FIGURE 1.1: This figure gives an overview about the definition of the single process synthesis problems and the distributed synthesis problem to explain the structure of this thesis and how we solve the problems.

thesis starts. In figure 1.1b, we see that this problem gets as input an architecture, describing the distributed system, and produces as output the set of transition systems satisfying together the specification if such a solution exists. Chapter 5 introduces the distributed synthesis problem formally. It also introduces the model of an architecture and the composition of transition systems. In chapter 6 the new encodings to solve the distributed synthesis are introduced, based on the encodings in chapter 4. In chapter 7, we describe the implementation and present the results of the evaluation.

2 Preliminary

In this chapter the basic models like transition systems and languages like LTL formulas are introduced. We recap the definitions of section 2.1, 2.2 and 2.3 by Finkbeiner and Schewe [12].

2.1 LTL

In this thesis, the specification is always described as an linear-time temporal logic (LTL) formula. Therefore we introduce this language in the following.

Syntax

Definition 2.1 (Syntax of LTL)**.**
Let Π be a set of atomic propositions. The following grammar defines the syntax of LTL over a finite set Π of atomic propositions:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc\, \varphi \mid \varphi\, \mathcal{U}\, \varphi \ ,$$

where $p \in \Pi$ is an atomic proposition.

The operators are defined as follows:

- $\neg$ is an unary operator, called negation
- $\vee$ is a binary operator, called disjunction
- $\bigcirc$ is an unary operator, called next
- $\mathcal{U}$ is a binary operator, called until

Based on these basic operators, the same standard abbreviations as known from boolean formulas are introduced.

- $\varphi \wedge \psi := \neg(\neg\varphi \vee \neg\psi)$ is a binary operator, called conjunction
- $\varphi \rightarrow \psi := \neg\varphi \vee \psi$ is a binary operator, called implication
- $\varphi \leftrightarrow \varphi := (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$ is a binary operator, called equivalence

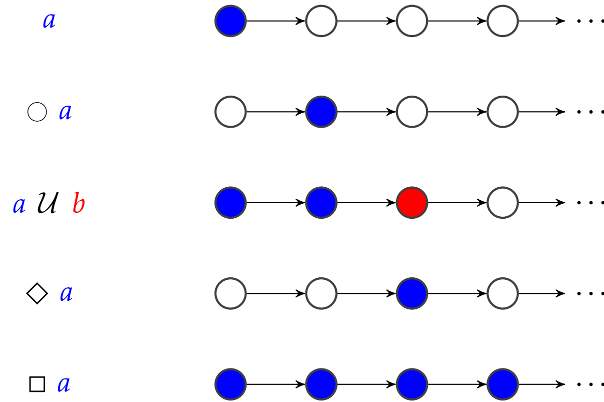


FIGURE 2.1: A virtual representation of the temporal operators [2]: The first column consists of the LTL formula, the second gives a graphical representation of an infinite word, that is accepted by the LTL formula in the same row. If a circle is labeled with a color, the atomic proposition, represented with this color, has to hold on this position. If the circle is not labeled, there is no information about this position.

The next and until operators are the basic temporal operators. They are extended by the following abbreviations:

- $\diamond\ \varphi := \text{true}\,\mathcal{U}\,\varphi$ is an unary operator, called eventually
- $\square\ \varphi := \neg(\diamond\ \neg\varphi)$ is an unary operator, called globally

Semantics

The semantics of LTL formulas is defined over infinite words. An infinite word is accepted if its word trace is accepted by the LTL formula. As in the syntax definition shown above LTL formulas have different kinds of operators: the operators we know from boolean formulas and the temporal operators. Figure 2.1 shows the intuitive semantics of the temporal operators, we look at in detail. The first LTL formula consists of the atomic proposition a. If a holds on the first position in an infinite word σ, σ is accepted by the LTL formula a. The second row, represents the LTL formula $\bigcirc\ a$. In the graphic shown the second circle is labeled blue, meaning the LTL formula accepts all infinite words where a hold on the second position. In the third row, the first and the second circle are labeled blue, the atomic proposition a holds on these positions, and the third circle is labeled red, the atomic proposition b holds on the third position. The LTL formula $a\,\mathcal{U}\,b$ accepts an infinite word σ, if there is a position i, where the atomic proposition b holds, and in each position $0 \leq i' < i$, the atomic proposition a holds. The next LTL formula $\diamond\ a$ contains the eventually operator. As already described this operator is an abbreviation and we can translate the LTL formula $\diamond\ a$ to $\text{true}\,\mathcal{U}\,a$. So all infinite words are accepted if there is a position, where the atomic proposition a holds. The last formula $\square\ a$ accepts all infinite words, where a holds on every position.

The formal definition of the semantics is shown in the following part. We use the satisfaction relation $\sigma, i \models \varphi$ where φ is the LTL formula and σ is the word beginning at position i. It

is possible that more than one atomic proposition holds in a position i, so a position can be seen as a set of atomic propositions.

Definition 2.2 (Semantics of LTL)**.**
For an infinite word $\sigma \in \omega \to 2^{\Pi}$ and a natural number $i \in \omega$, the semantics of an LTL formula is defined as follows:

- $\sigma, i \models p :\Leftrightarrow p \in \sigma(i)$
  for atomic propositions $p \in \Pi$. p has to hold on the word σ at position i
- $\sigma, i \models \neg\varphi :\Leftrightarrow \sigma, i \not\models \varphi$
  for the boolean negation, where φ is an LTL formula. φ is not allowed to hold on the word σ at position i.
- $\sigma, i \models \varphi \vee \psi :\Leftrightarrow \sigma, i \models \varphi$ or $\sigma, i \models \psi$
  for the boolean disjunction, where φ and ψ are LTL formulas. Either φ or ψ have to hold on the word σ at position i.
- $\sigma, i \models \bigcirc \varphi :\Leftrightarrow \sigma, i+1 \models \varphi$
  for the temporal path operator next, where φ is an LTL formula. φ has to hold on the word σ at position $i+1$.
- $\sigma, i \models \varphi \mathcal{U} \psi :\Leftrightarrow \exists n \geq i.\ \sigma, n \models \psi$ and $\forall m \in \{i, \ldots n-1\}.\ o, m \models \varphi$
  for the temporal path operator until, where φ and ψ are LTL formulas. There exists a position n in the word σ, such that ψ holds at position n and at all positions between i and $n-1$ φ holds on the word σ.

A sequence $\sigma \in \omega \to 2^{\Pi}$ is a *model* of an LTL formula φ denoted by $\sigma \models \varphi$, iff $\sigma, 0 \models \varphi$

Example 2.1.
During this thesis most examples will be based on the following LTL formula.

$$\varphi = \Box(r_1 \to \bigcirc \Diamond g_1) \wedge \Box(r_2 \to \bigcirc \Diamond g_3) \wedge \Box\neg(g_1 \wedge g_2)$$

This formula φ accepts all infinite words, where each occurrence of r_1 is followed by a g_1 in the future, each occurrence of r_2 is followed by a g_2 in the future and g_1 and g_2 does not hold on the same time. The idea of this specification is that each request, that is the input, has to be answered by the grant, that has to be the output of the system.

2.2 Transition System

As described in the introduction, the reactive synthesis algorithm we explain in this thesis finds a transition system for a given specification, if there is one. These transition systems represent the implementations. This section introduces the different kinds of transition systems. In general, transition systems can be formally defined as follows:

Definition 2.3 (Transition System)**.**
A transition system $\mathcal{T}$ with input set I and output set O is a tuple $\langle T, t_0, \tau \rangle$, where

- T is the set of states
- $t_0 \in T$ is the initial state
- $\tau : T \times 2^I \rightarrow 2^O \times T$ is the transition relation, with 2^I being the set of possible inputs and 2^O the set of possible outputs.

A transition system $\mathcal{T}$ is finite, iff T is finite. The transition function maps one state t with a given input i to an output o and a next state t'. We call the output o the label of the transition. We differentiate between state-labeled and transition-labeled transition systems.

Moore Transition System

A transition system $\mathcal{T}$ is called a state-labeled transition system or Moore transition system if the labelling produced by $\tau(t,i)$ is independent of i. Formally: Given a state $t \in T$ and any input $i \neq i' \in 2^I$ with $\tau(t,i) = (o,_)$ and $\tau(t,i') = (o',_)$ it holds that $o = o'$.

Example 2.2 (Moore Transition System)**.**
An example of a Moore transition system is represented in figure 2.2a. This example starts at t_0 which is the initial state. Independent of the input, the output g_1 is produced and the system ends up in t_1. t_1 ends up in the initial state independent of the input and produces the output g_2.

Mealy Transition System

A transition system $\mathcal{T}$ is called Mealy transition system if it is transition-labeled.

Example 2.3 (Mealy Transition System)**.**
This Mealy transition system in figure 2.2b has the same behavior like the Moore transition system in figure 2.2a .

Definition 2.4 (Traces)**.**
A trace σ in a transition system $\mathcal{T}$ is a sequence:

$$t_0 \xrightarrow{\tau_1} t_1 \xrightarrow{\tau_2} t_2 \xrightarrow{\tau_3} \ldots,$$

where t_0 is the initial state and $t_i \xrightarrow{\tau_i} t_{i+1}$ is in τ.

To see the difference between Mealy and Moore transition systems, we look at the different traces with the input $\{r_1, r_2\}^\omega$ on the examples 2.2a and 2.2b.

- $t_0 : g_2 \xrightarrow{r_1} t_1 : g_1 \xrightarrow{r_1} t_0 : g_2 \xrightarrow{r_1} t_1 : g_1 \xrightarrow{r_1} \ldots$
- $t_0 \xrightarrow{r_1/g_1} t_1 \xrightarrow{r_2/g_2} t_0 \xrightarrow{r_1/g_1} t_1 \xrightarrow{r_2/g_2} \ldots$

(a) Moore Representation (b) Mealy Representation

FIGURE 2.2: A virtual representation of the transition systems $\langle T, t_0, \tau \rangle$ with input set $\{r_1, r_2\}$ and output set $\{g_1, g_2\}$, with: $T = \{t_0, t_1\}$ and $\tau = \{(t_0, \top, \{g_1\}, t_1), (t_1, \top, \{g_2\}, t_0)\}$, where $\top$ accepts every input.

The first trace represents the Moore transition system. We see the state $t_0 : g_2$ goes with input r_1 to the state $t_1 : g_1$ and so on. The output is part of the state. The second trace represents the Mealy transition system. t_0 goes with the transition r_1/g_1 to the state t_1. The output is now part of the transition.

2.3 Automata

To solve the bounded synthesis approach, the LTL specification φ is translated into a universal co-Büchi automaton $\mathcal{A}_\varphi$ with $\mathcal{L}(\varphi) = \mathcal{L}(\mathcal{A}_\varphi)$. Because we use Büchi automata in the translation, in this section Büchi automata and co-Büchi automata are introduced.

Non-deterministic Büchi Automata

Definition 2.5 (Non-deterministic Büchi Automata).
A non-deterministic Büchi automaton $\mathcal{A}$ over a finite alphabet Σ is a tuple $\langle Q, q_0, \delta, F \rangle$, where

- Q is the set of states
- q_0 is the initial state
- $\delta : Q \times \Sigma \times Q$ is the transition relation.
- $F \subseteq Q$ is the set of accepting states

The semantics, defining the acceptance, is defined over infinite words

$$\sigma = \sigma_1 \sigma_2 \dots \sigma_i \dots \in (2^\Sigma)^\omega,$$

where σ_i is the letter at position i in the word σ.

Definition 2.6 (Run).
A run of an infinite word σ on a non-deterministic Büchi automaton $\mathcal{A}$ is a sequence

$$q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} q_2 \xrightarrow{\sigma_2} \dots,$$

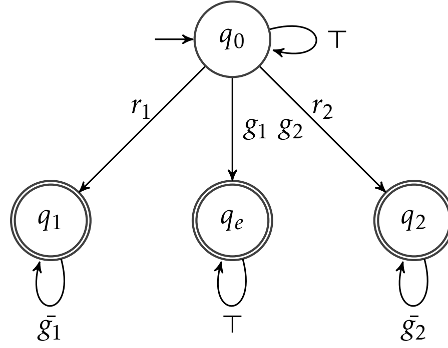


FIGURE 2.3: Visual Representation of the non-deterministic Büchi Automaton in Example 2.4 and the universal co-Büchi Automaton in Example 2.5

where q_0 is the initial state and $(q_i, \sigma_i, q_{i+1}) \in \delta$.

A run is accepted by $\mathcal{A}$ iff $q_i \in F$ holds for infinitely many i.

Definition 2.7 (Acceptance Büchi Automaton).
An infinite word σ is accepted by a non-deterministic Büchi automaton, iff there exists an accepted run.

Intuitively, an infinite word σ is accepted if there are infinitely many occurrences of accepting states in a run of $\mathcal{A}$ on σ.

Example 2.4 (Büchi Automaton).
An example of a Büchi automaton is shown in figure 2.3. It can be formally written as $\mathcal{A} = \langle Q, q_0, \delta, F \rangle$, where

- $Q = \{q_0,\ q_1,\ q_2,\ q_e\}$
- $\delta = \{(q_0, r_1, q_1),\ (q_1, \bar{g}_1, q_1),\ (q_0, g_1\ g_2, q_e),\ (q_e, \top, q_e),\ (q_0, r_2, q_2),\ (q_1, \bar{g}_2, q_1)\}$
- $F = \{q_1,\ q_2,\ q_e\}$

With $\bar{x}$ is mentioned: x is not part of the input. This Büchi automaton accepts all words that do not satisfy the LTL formula

$$\varphi = \Box(r_1 \rightarrow \bigcirc \Diamond g_1) \wedge \Box(r_2 \rightarrow \bigcirc \Diamond g_3) \wedge \Box\neg(g_1 \wedge g_2)$$

in example 2.1. Here an example: The infinite word $r_1(g_2 r_1)^\omega$ is accepted, because of the accepting run

$$q_0 \xrightarrow{r_1} q_1 \xrightarrow{g_2} q_1 \xrightarrow{r_1} q_1 \xrightarrow{g_2} \ldots,$$

that only stays in q_1.

Universal co-Büchi Automata

Definition 2.8 (Universal co-Büchi Automata).
A universal co-Büchi automaton $\mathcal{A}$ over a finite alphabet Σ is a tuple $\langle Q, q_0, \delta, F\rangle$, where

- Q is the set of states
- q_0 is the initial state
- $\delta : Q \times \Sigma \times Q$ is the transition relation
- F is the set of rejecting states

As we can see, the universal co-Büchi automaton has the same definition as the non-deterministic Büchi automaton, with one difference. F is called the set of rejecting states, whereas in a Büchi automaton, F is called the set of accepting states.

Non-deterministic Büchi automaton and a universal co-Büchi automaton differ in the accepting and branching conditions.

Definition 2.9 (Run).
A run of an infinite word σ on a universal co-Büchi automaton $\mathcal{A}$ is a sequence

$$q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} q_2 \xrightarrow{\sigma_2} \dots,$$

where q_0 is the initial state and $(q_i, \sigma_i, q_{i+1}) \in \delta$.

A run is accepted by $\mathcal{A}$ iff $q_i \in F$ holds for finitely many i.

Definition 2.10 (Acceptance co-Büchi Automaton).
An infinite word σ is accepted by a universal co-Büchi automaton, iff all possible runs are accepted.

Intuitively, an infinite word σ is accepted if there are only finitely many occurrences of rejecting states in all runs of $\mathcal{A}$ on σ.

Example 2.5 (Universal co-Büchi Automaton).
The example given in figure 2.3 can also be interpreted as a universal co-Büchi automaton $\mathcal{A}$ having the same language as the LTL formula defined in example 2.1. An example of a word which is not accepted is $r_1(g_2r_2)^\omega$. The run that stays always in q_0 is accepted, but the run leading to q_1 and staying there is not, because the rejecting state q_1 is visited infinitely often. This run does therefore not fulfill the accepting condition of the universal co-Büchi automaton. An example of a word which is accepted is $(r_1g_1)^\omega$ as the only possible run stays in q_0 and never leaves this state which is accepted by the universal co-Büchi automaton.

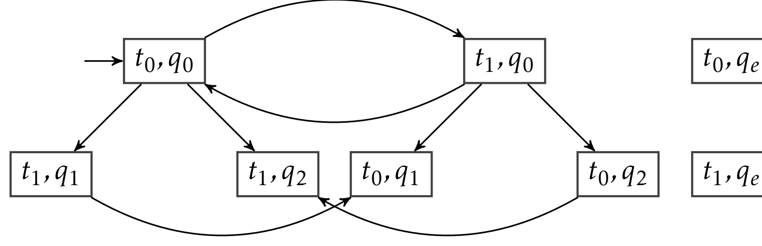


FIGURE 2.4: This figure represent the reachable fragment of the run graph between the transition system in figure 2.2b and the universal co-Büchi automaton in figure 2.3.

2.4 Run Graph

To verify, if a transition system holds on a universal co-Büchi automaton, we have to simulate all runs of the transition system on the universal co-Büchi automaton. These runs are represented by a run graph. We recap the definition by Faymonville et al. [8].

Definition 2.11 (Run Graph).
The product of a transition system $\mathcal{T} = \langle T,\ t_0, \tau\rangle$ and the universal co-Büchi automaton $\mathcal{A} = \langle Q,\ q_0,\ \delta,\ F\rangle$ is a run graph $G = \langle V,\ E\rangle$, where

- $V = T \times Q$ is the set of vertices
- $E \subseteq V \times V$ is the edge relation with
  $((t,\ q),(t',\ q')) \in E \Leftrightarrow \exists i \in 2^I.\ \exists o \in 2^O.\ \tau(t,\ i) = (o,\ t') \wedge (q,\ i \cup o, q') \in \delta$

The intuition of the edge-relation is, that there is an edge from $(t,\ q)$ to $(t',\ q')$, iff an edge from t to t' with input i and output o exists in the transition system $\mathcal{T}$ and there is an edge from q to q' with i and o as input in the universal co-Büchi automaton $\mathcal{A}$.

Example 2.6 (Run Graph).
The transition system represented in figure 2.2b and the universal co-Büchi automaton represented in figure 2.3 can be combined to a run graph. The fragment reachable from the initial state is represented in figure 2.4.

2.5 Quantified Boolean Formulas

As already mentioned, this thesis introduces different encodings for the reactive synthesis. One of these encodings is represented as a quantified boolean formula (QBF). A QBF is a SAT propositional formula, where variables can be bounded by quantifiers. We recap the definitions by Rabe and Tentrup [23].

Syntax

Definition 2.12 (Syntax of QBF)**.**
The following grammar defines the syntax of a QBF over a finite set X of variables with domain $\mathbb{B} = \{0,1\}$:

$$\varphi ::= x \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x.\ \varphi \mid \forall x.\ \varphi\ ,$$

where $x \in X$.

The same standard abbreviations for boolean formulas, defined in section 2.1, are used. For readability we later lift the quantifier over variables to a quantifier over a set of variables so $\forall x_1.\ \forall x_2.\ \ldots \forall x_n.\ \varphi$ becomes $\forall X.\varphi$ with $X = \{x_1, x_2, \ldots, x_n\}$. The same holds for the $\exists$-quantifier.

Semantics

We now want to define the semantics of QBF. Therefore we introduce additional notation.

An assignment of X is a function $\alpha : X \to \mathbb{B}$, that maps each variable $x \in X$ to true (represented by the value 1) or false (represented by the variable 0). For readability, an assignment can also be described by a set $\vec{x} \subseteq X$ including all variables that are assigned to true. $A(X)$ represents the set of assignments over the set of variables X.

A variable x is called bounded by a quantifier $Q \in \{\forall, \exists\}$ in the scope of ψ in a QBF φ, iff $Qx.\ \psi$ is part of the QBF φ. All variables that are not bounded are called free variables. A formula without free variables is called closed. We use the semantics relation $\vec{x} \models \varphi$, where φ is a QBF and $\vec{x} \subseteq X$ is an assignment to the set of free variables of φ.

Definition 2.13 (Satisfiability of QBF)**.**
For an assignment $\vec{x}$ with $\vec{x} \subseteq X$, the semantics of a QBF is defined as follows:

- $\vec{x} \models x :\Leftrightarrow x \in \vec{x}$
  for free variables $x \in X$.
- $\vec{x} \models \neg\varphi :\Leftrightarrow \vec{x} \not\models \varphi$
  for boolean negation, where φ is a QBF formula.
- $\vec{x} \models \varphi \vee \psi :\Leftrightarrow \vec{x} \models \varphi$ or $\vec{x} \models \psi$
  for boolean conjunction, where φ and ψ are QBF formulas.
- $\vec{x} \models \exists x.\ \varphi :\Leftrightarrow \vec{x} \models \varphi$ or $\vec{x} \cup \{x\} \models \varphi$
  for the existential quantifier, where φ is a QBF formula.
- $\vec{x} \models \forall x.\ \varphi :\Leftrightarrow \vec{x} \models \varphi$ and $\vec{x} \cup \{x\} \models \varphi$
  for the universal quantifier, where φ is a QBF formula.

A closed QBF φ can be checked for satisfiability by $\emptyset \models \varphi$.

(a) $\exists a.\ \forall b.\ \exists c.\ a \wedge (b \leftrightarrow c)$ (b) $\exists a.\ \forall b.\ \exists c.\ c \wedge (b \leftrightarrow a)$

Figure 2.5: Two BDT that represent the Skolem functions for the respective QBFs below.

Skolem Functions

A variable y depends on a variable x if the existential quantifier that binds y is in the scope of the universal quantifier that binds x. For example in the QBF $\forall x.\ \exists y.\ x \leftrightarrow y$ the variable y depends on the variable x. We call the dependency set of an existential quantifier variable y as $\text{dep}(y)$. With this dependency we can build a Skolem function $f_y : \mathcal{A}(\text{dep}(y)) \rightarrow \mathbb{B}$ that maps an assignment of all variables depending on the existential bounded variable y to true or false. The satisfiability of a QBF is equivalent to the existence of a set of Skolem functions for the existentially bounded variables Y such that $\{y \in Y \mid f_y(\vec{x} \cap \text{dep}(y))\} \models \varphi$ holds for every assignment $\vec{x}$ of the universally quantified variables X.

A representation of Skolem functions are binary decision trees (BDT). In BDT the assignments of the dependent variables are represented with branching and labels. The following two examples give an intuition for QBF formulas.

Example 2.7.
At first we take a look at the formula $\exists a.\ \forall b.\ \exists c.\ a \wedge (b \leftrightarrow c)$. If we want to proof that this formula is satisfiable we have to find a valid Skolem function for the variables a and c. Figure 2.5a shows a BDT representing the Skolem functions. Because a has no dependencies and a stands alone on the left side of the conjunction, a has to be assigned always to 1 (true). The variable c however depends on the variable b. If b is assigned to 0, c is set to 0 and if b is assigned to 1 c is set to 1.

Example 2.8.
The second example introduces an unsatisfiable formula: $\exists a.\forall b.\exists c.\ c \wedge (b \leftrightarrow a)$. To prove the unsatisfiability, we have to show that all possible Skolem functions are not valid. Figure 2.5b gives an example. The first Skolem function assigns a (which has no dependencies) to a value. In example 2.5a, only one assignment was possible and we could choose this. Now both assignments are possible so we guess and choose a to be 1. The second function assigns c to a value, depending on the value of b. If b is assigned to 1, c can be assigned to 1 and the formula gets true. If b is assigned to 0, we can find no assignment for c such that the formula gets true. The Skolem function that assigns a to 1 is wrong so we change the assignment to 0. If we would assign c to its value, we get the same effect. These are the possibilities to assign the Skolem functions which together proof that this QBF is unsatisfiable.

2.6 Dependency Quantified Boolean Formulas

In chapter 6 we express one of the encodings with dependency quantified boolean formulas (DQBF). A DQBF is a SAT propositional formula, where the variables can be bounded by quantifiers, similar to QBF. We recap the definition by Finkbeiner and Tentrup [13] and Fröhlich et al. [14] .

Syntax

Definition 2.14 (Syntax of DQBF).
A DQBF φ can always be described in the following structure:

$$\varphi = Q.\psi = \forall x_1.\forall x_2.\forall x_3.\ldots\exists y_1(H_1).\exists y_2(H_2).\exists y_3(H_3).\ldots.\psi,$$

where Q is the quantifier prefix and ψ is a SAT propositional formula over the variables $V = X \cup Y = \{x_1, x_2, x_3, \ldots\} \cup \{y_1, y_2, y_3, \ldots\}$ and $H_i \subseteq X$ for all i.

A DQBF $\varpi = Q.\psi$ consists of two parts. At first the variables $v \in V$ are bounded with universal and existential quantifiers. For the existentially quantified variables, we specify the dependencies explicitely with the set $H_i \subseteq X$. This is the difference to QBF, where the dependencies are specified by the order of the quantifiers. The second part is the SAT propositional formula ψ over the variables V. ψ consists of atomic propositions $v \in V$, the binary operators $\vee$ and the unary operator $\neg$. But the same standard abbreviations for boolean formulas, defined in section 2.1, can be used.

Semantics

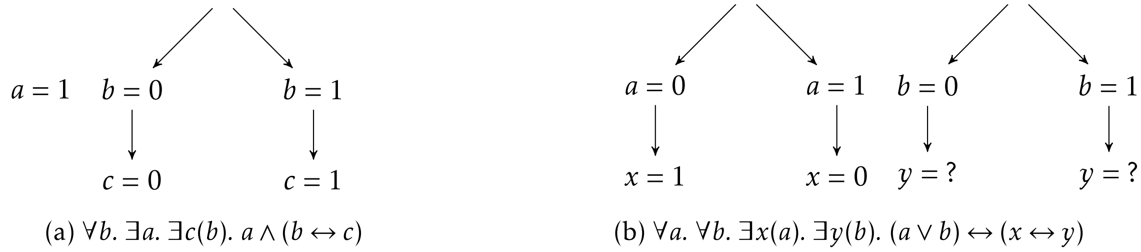
Before we can define the satisfiability of a DQBF formula we need more notation.

We call the function $\alpha : V \to \mathbb{B}$ an assignment over the variables V. The existentially quantified variables y are represented with the Skolem functions $f_y : (H_y \to \mathbb{B}) \to (\{y\} \to \mathbb{B})$. So an existential quantified variable is mapped to an assignment, under the assignment of its depending variables $h \in H_y$.

Definition 2.15 (Satisfiability of DQBF).
A DQBF $\varphi = Q.\psi$ is satisfiable, iff for all possible assignments of the universally quantified variables $x \in X$, there is a Skolem function f_y for all existentially quantified variables $y \in Y$ such that ψ is satisfiable.

The Skolem functions can be represented as BDT like the Skolem functions in QBF. To get an intuition consider the following two examples.

Example 2.9.
At first we take a look at the formula $\forall b.\ \exists a.\ \exists c(b).\ a \wedge (b \leftrightarrow c)$. To prove the satisfiability of this DQBF we have to find a valid Skolem function for the variables a and c. Figure 2.6a represents these two functions as BDTs. The first variable a has no dependencies. Because the atomic proposition is on the left side of the conjunct, we build the Skolem function, that maps a to the assignment 1. The second Skolem function gets as input the assignment



(a) $\forall b.\ \exists a.\ \exists c(b).\ a \wedge (b \leftrightarrow c)$

(b) $\forall a.\ \forall b.\ \exists x(a).\ \exists y(b).\ (a \vee b) \leftrightarrow (x \leftrightarrow y)$

Figure 2.6: Two BDT that represent the Skolem functions for the respective DQBFs below.

of b. Because the right side of the conjunct is an equivalence between b and c, the Skolem function maps the variable c to the assignment of b.

Example 2.10.
The second example introduces an unsatisfiable DQBF: $\forall a.\ \forall b.\ \exists x(a).\ \exists y(b).\ (a \vee b) \leftrightarrow (x \leftrightarrow y)$. To prove the unsatisfiability, we have to prove that all possible Skolem functions are not valid. Figure 2.6b represents a possible Skolem function. We have to find a function for the variable x dependent on the variable a. Because we cannot define a unique assignment for x, dependent on the assignment of a, we have to guess an assignment. We have to find a Skolem function for the variable y, too. This function gets as input the assignment of b. If b is assigned to 0, y must have the same value like x, iff a is assigned to 1. But y is independent of the assignment of a and we cannot fund a valid SKolem function. This independency between y and a, respectively x and b holds for all possible Skolem functions.

3 Single Process Synthesis

This chapter introduces the reactive synthesis problem followed by the bounded synthesis approach. At first both problems are defined formally and afterwards the general concept how to solve the bounded synthesis is explained. We recap the definitions by Finkbeiner and Schewe [12].

3.1 Synthesis

Definition 3.1 (Reactive Realizability from LTL Specification)**.**
Given a specification as an LTL formula φ over a set of variables, partitioned into inputs and outputs. The realizability problem is to decide, whether there is a transition system $\mathcal{T}$ that satisfies φ.

The realizability problem decides if there is a solution or not. The reactive synthesis returns a transition system that satisfies the given specification if there is such a transition system or return none if it is not possible to find one. The returned transition system describes the model for the implementation of the process and the programmer is free from writing programs.

Synthesis has a high complexity. For single processes with an LTL specification it is 2EXPTIME-complete [12]. Another disadvantage of synthesis can be the size of the returned transition system, because synthesis finds one, but not necessarily the smallest solution.

3.2 Bounded Synthesis

The bounded synthesis approach was introduced by Finkbeiner and Schewe to cope with this problem. Bounded synthesis is a restriction of the synthesis problem that bounds the size of the transition system, for example in the number of states. If no solution can be found with this size, the state space is increased.

The general concept how to solve bounded synthesis is represented in figure 3.1. On the one hand the LTL formula φ has to be translated into a universal co-Büchi automaton $\mathcal{A}$, such that $\mathcal{L}(\varphi) = \mathcal{L}(\mathcal{A})$. On the other hand a transition system $\mathcal{T}$ with bounded size b and an annotation function λ is guessed. Afterwards it is verified that the transition system holds on the universal co-Büchi automaton by building the run graph and checking if the annotation function is valid.

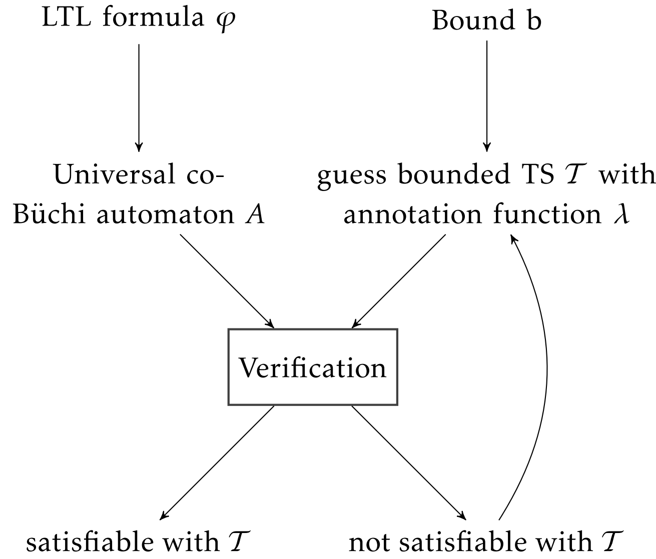


FIGURE 3.1: General Concept to Solve Bounded Synthesis

3.3 Translating LTL to Universal co-Büchi Automata

The following steps describe a possibility to translate an LTL formula φ to an equivalent universal co-Büchi automaton $\mathcal{A}$:

1. Negating the LTL formula $\neg\varphi$
2. Building a non-deterministic Büchi automaton $\mathcal{A}_{\neg\varphi}$ for $\neg\varphi$
3. Building a universal co-Büchi automaton $\mathcal{A}_{\varphi}$ from $\mathcal{A}_{\neg\varphi}$

In this section the examples are based on the formula

$$\varphi = \Box(r_1 \rightarrow \bigcirc \Diamond g_1) \wedge \Box(r_2 \rightarrow \bigcirc \Diamond g_3) \wedge \Box\neg(g_1 \wedge g_2)$$

(compare example 2.1).

1. Negating the LTL formula

At first the LTL formula φ has to be negated.

Example 3.1 (Negating φ)**.**
$\neg\varphi = \neg(\Box(r_1 \rightarrow \bigcirc \Diamond g_1) \wedge \Box(r_2 \rightarrow \bigcirc \Diamond g_3) \wedge \Box\neg(g_1 \wedge g_2))$

2. Building the non-deterministic automaton

With the negated LTL formula, we can now build a non-deterministic automaton $\mathcal{A}_{\neg\varphi}$, such that $\mathcal{L}(\mathcal{A}_{\neg\varphi}) = \mathcal{L}(\neg\varphi)$. The size of the resulting automaton is exponential in the size of the formula [16].

Example 3.2 (Non-deterministic Büchi Automaton)**.**
The resulting automaton has been shown in example 2.4.

3. Building the universal co-Büchi automaton

Let $\mathcal{A}_{\neg\varphi}$ be the non-deterministic Büchi automaton. We obtain the non-deterministic Büchi automaton $\mathcal{A}_{\varphi}$ by duplicating $\mathcal{A}_{\neg\varphi}$ and changing the branching and accepting condition. The branching condition changes in a way that all possible runs have to be accepted instead of one possible run. A word is now accepted if it leads to a finite number of occurrences of rejecting states instead of an infinite number of occurrences of accepting states (compare subsection 2.3).

Example 3.3 (Universal co-Büchi Automaton).
The example to this formula is explained in example 2.5.

3.4 Verification

As described in the concept in figure 3.1, we solve the bounded synthesis with the help of verification. We verify if a transition system holds on a universal co-Büchi automaton with the run graph by checking the validity of the given annotation function. We recap the definitions by Faymonville et al. [8].

Definition 3.2 (Annotation Function).
Given a transition system $\mathcal{T} = \langle T, t_0, \tau\rangle$ and a universal co-Büchi automaton $\mathcal{A} = \langle Q, q_0, \delta, F\rangle$. The annotation function $\lambda : T \times Q \to \{\bot\} \cup \mathbb{N}$ is a function that maps each state of the run graph to $\bot$ or a natural number.

We can verify that the transition system $\mathcal{T}$ holds on the co-Büchi automaton $\mathcal{A}$ with the help of the annotation function λ. Using this function it can be checked if the rejecting states are only visited finitely often. If this condition holds we call the annotation function λ valid. This can be checked by two conditions. First, the initial state of the run graph has not to be $\bot$, otherwise the annotation function that maps the first state to not reachable would be valid. Second, all reachable states in the run graph are labeled with greater numbers than their predecessors if they are rejecting states or with greater or equal number otherwise. The formal definition is given below.

Definition 3.3 (Validity of Annotation Functions).
Given a transition system $\mathcal{T} = \langle T, t_0, \tau\rangle$ and a universal co-Büchi automaton $\mathcal{A} = \langle Q, q_0, \delta, F\rangle$. The annotation function $\lambda : T \times Q \to \{\bot\} \cup \mathbb{N}$ is valid if it satisfies the following conditions:

- $\lambda(t_0, q_0) \neq \bot$
- $\forall t \in T, q \in Q.\ \lambda(t, q) = k \neq \bot \to \forall i \in 2^I.\ \tau(t, i) = (o, t') \wedge (q, i \cup o, q') \in \delta \to \lambda(t', q') \triangleright_{q'} k$,
  with $\triangleright_{q'} := \begin{cases} > & \text{if } q' \in F \\ \geq & \text{if } q' \notin F \end{cases}$

Theorem 3.1 (Acceptance Transition System in Universal co-Büchi Automaton [12]).
Given a transition system $\mathcal{T}$ and a universal co-Büchi automaton $\mathcal{A}$. $\mathcal{T}$ holds on $\mathcal{A}$, iff a valid $(|\mathcal{T}| \cdot |\mathcal{A}|)$-bounded annotation function can be found.

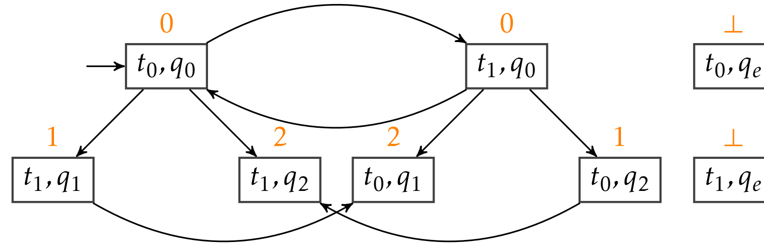


Figure 3.2: Run Graph from Figure 2.4 with a Valid Annotation Function

$(|\mathcal{T}| \cdot |\mathcal{A}|)$ gives the maximal label of the run graph. The proof is the same like the proof by Finkbeiner and Schewe[12], with the different definitions of the transition system $\mathcal{T}$ and the universal co-Büchi Automaton $\mathcal{A}$.

Example 3.4 (Verification)**.**
In figure 3.2, the run graph from figure 2.4 is represented with an annotation function. The annotation function is represented as the orange labels. You can see that the states (t_0, q_e) and (t_1, q_e) are unreachable, so they are labeled with $\perp$. The initial state (t_0, q_0) and the state (t_1, q_0) are labeled with 0, because the only input edge is the loop between these two states and they do not contain a rejecting state so they can be equal. (t_1, q_2) is labeled with 2, because (t_1, q_1) has to be strictly greater than (t_0, q_0) because of the incoming edge from (t_0, q_0) and q_1 is a rejecting state. Similar to (t_0, q_1), (t_1, q_2) has to be strictly greater than (t_1, q_1) and (t_0, q_0), because of the incoming edges and q_2 is a rejecting state. So we found a valid annotation function and can verify that the transition system $\mathcal{T}$ holds on the universal co-Büchi automaton $\mathcal{A}$.

4 Encodings for Single Processes

In this chapter we introduce two of the four new encoding by Faymonville et al. [8] to solve the bounded synthesis approach for single processes. The encodings are written as a SAT propositional, QBF and DQBF formula. As mentioned in the last chapter, we have to verify if a transition system holds on a universal co-Büchi automaton. Therefore we start to express the verification as a SAT propositional formula.

4.1 Verification

In the last chapter the concept of verification using annotation functions was introduced. As an input we receive the run graph, between the transition system and the universal co-Büchi automaton, and the annotation function. As explained in theorem 3.1, we have to verify if the given annotation function is valid. In figure 4.1 the general concept of this formula is represented. It is checked if for all reachable states in the run graph, the annotation for all successors is valid. This concept is now encoded in a SAT propositional formula. Therefore we use the following propositional representation of the input:

- λ: the annotation function λ is split into two functions:
  - $\lambda^{\mathbb{B}} : T \times Q \to \mathbb{B}$
  - $\lambda^{\#} : T \times Q \to \mathbb{N}$

  Both functions get as input a state in the run graph. The first function returns true of false, reachable or not reachable, the second returns the label as a natural number. With these two functions the variables $\lambda^{\mathbb{B}}_{t,q}$ and $\lambda^{\#}_{t,q}$ are built for all $t \in T$ and $q \in Q$, where the index can be seen as the input for the functions.

- $\mathcal{T} = \langle T, t_0, \tau \rangle$: In the SAT propositional formula, the transition relation τ is represented as variables, like the annotation function. Therefore $\tau : T \times 2^I \to 2^O \times T$ is expressed as:
  - $\tau_{t,i,t'} \leftrightarrow \tau(t,i) = (_, t')$
  - $o_{t,i} = o$ if $\tau(t,i) = (o, _)$

  The variable $\tau_{t,i,t'}$ signals if there is a transition from t with input i to t'. The variable $o_{t,i}$ produces the output.

- $\mathcal{A} = \langle Q, q_0, \delta, F \rangle$: We have to represent the transition function δ as a variable. The variable is indexed with two states in the universal co-Büchi automaton q and q', the

input: $\mathcal{T} = \langle T, t_0, \tau \rangle \quad \mathcal{A} = \langle Q, q_0, \delta, F \rangle \quad \lambda$

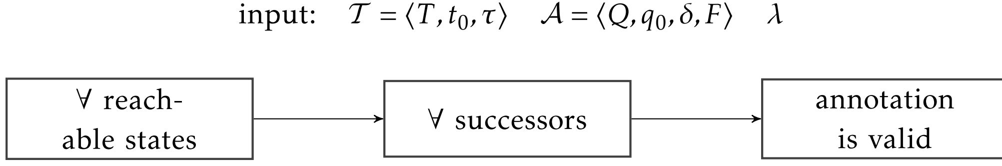


Figure 4.1: General Concept of the Encoding Used for the Verification

input i and the state of the transition system, because the input of the universal co-Büchi automaton is the input and output of the transition system. Formally $\delta : Q \times 2^{I \cup O} \to Q$ is represented by the variables: $\delta_{t,q,i,q'} \leftrightarrow \delta(q, i \cup o_{t,i}) = q'$

With these different variables, the verification formulas can now be build.

$$\bigwedge_{q \in Q} \bigwedge_{t \in T} \left(\lambda^{\mathbb{B}}_{t,q} \to \bigwedge_{q' \in Q} \bigwedge_{i \in 2^I} \left(\delta_{t,q,i,q'} \to \bigwedge_{t' \in T} \left(\tau_{t,i,t'} \to \lambda^{\mathbb{B}}_{t',q'} \wedge \lambda^{\#}_{t',q'} \rhd_{q'} \lambda^{\#}_{t,q} \right) \right) \right)$$

As you can see, the SAT propositional formula has the same structure like the pattern in figure 4.1. The formula starts with ∀ reachable states, that is represented by the annotation function part $\lambda^{\mathbb{B}}_{t,q}$. This variable is true, if the state (t,q) is reachable in the run graph. The right side of the implication starts with $\delta_{t,q,i,q'} \to \tau_{t,i,t'}$. These two variables together build the successor (t', q') in the run graph, because q' is a successor in the universal co-Büchi automaton (denoted by $\delta_{t,q,i,q'}$) and t' is a successor in the transition system, because of $\tau_{t,i,t'}$. After that it has to be checked if (t', q') is reachable in the annotation function and if $\lambda^{\#}_{t',q'} \rhd_{q'} \lambda^{\#}_{t,q}$, i.e. the annotation is valid for this transition.

With this verification formula, the different synthesis formulas can be built.

4.2 SAT Propositional Encoding: Explicit Encoding

In the last section the verification part of the SAT propositional formula was introduced. Like in figure 3.1 shown, we get as input only the specification as a universal co-Büchi automaton and we have to guess an annotation function and a transition system, so that we can verify.

Constraints

Before we guess we have to add two constraints:

- $\lambda^{\mathbb{B}}_{t_0,q_0}$. This constraint asserts that the initial state is reachable.
- $\bigwedge_{t \in T} \bigwedge_{i \in 2^I} \bigvee_{t' \in T} \tau_{t,i,t'}$. This constraint asserts that each state in the transition system has at least one outgoing edge, for each input.

Without the first constraint for every transition system and universal co-Büchi automaton, the annotation function that maps each state to $\bot$ would be accepted by the formula. This result is possible because the premise is always false. But it is obvious that this annotation function is not valid as it violates definition 3.3.

The second constraint prevents that $\tau_{t,i,t'}$ is not always false. It assures that for example the transition system that has no outgoing edge in the initial state is not accepted. This transition system produces no output and so it follows not the idea of synthesis.

Building the Formula

The guess is done with the existential quantifier.

$$\exists\{\lambda^{\mathbb{B}}_{t,q}, \lambda^{\#}_{t,q} \mid t \in T, q \in Q\}$$
$$\exists\{\tau_{t,i,t'} \mid (t,t') \in T \times T, i \in 2^I\}$$
$$\exists\{o_{t,i} \mid o \in O, t \in T, i \in 2^I\}$$

where the first line guesses all variables that describe together the annotation function and the second and third line the transition system.

The following formula describes now the SAT propositional encoding, for the synthesis:

$$\exists\{\lambda^{\mathbb{B}}_{t,q}, \lambda^{\#}_{t,q} \mid t \in T, q \in Q\}$$
$$\exists\{\tau_{t,i,t'} \mid (t,t') \in T \times T, i \in 2^I\}$$
$$\exists\{o_{t,i} \mid o \in O, t \in T, i \in 2^I\}$$
$$\lambda^{\mathbb{B}}_{t_0,q_0} \wedge \bigwedge_{t \in T} \bigwedge_{i \in 2^I} \bigvee_{t' \in T} \tau_{t,i,t'}$$
$$\bigwedge_{q \in Q} \bigwedge_{t \in T} \left(\lambda^{\mathbb{B}}_{t,q} \rightarrow \bigwedge_{q' \in Q} \bigwedge_{i \in 2^I} \left(\delta_{t,q,i,q'} \rightarrow \bigwedge_{t' \in T} \left(\tau_{t,i,t'} \rightarrow \lambda^{\mathbb{B}}_{t',q'} \wedge \lambda^{\#}_{t',q'} \triangleright_{q'} \lambda^{\#}_{t,q} \right) \right) \right)$$

The size of the formula is represented in the following theorem.

Theorem 4.1 (Size of the Explicit Encoding [8])**.**
The size of the constraint system is in $\mathcal{O}(nm^2 \cdot 2^{|I|} \cdot (|\delta_{q,q'}| + n \log(nm)))$ and the number of variables is in $\mathcal{O}(n(m \log(nm) + 2^{|I|} \cdot (|O| + n)))$, where $n = |T|$ and $m = |Q|$.

In this formula only existential quantifiers over propositional variables are used, so this formula can be solved by a SAT solver.

Building the Transition System

Like in the verification part in section 4.1, where we transform the transition system and the annotation function into variables, we can do the other way round and transform the variables into a transition system.

Example 4.1.
We want to find a transition system that is accepted by the universal co-Büchi automaton in example 3.3. If we bound the transition system to a state size of two, the formula in appendix 10.2 is produced. The result of the formula is a variable assignment. We only want to look at the variables of the transition system, that was produced. We list the variables that are assigned to true in groups, so we can produce the resulting edges. But first in general:

$\tau_{t,i,t'}$ is set to true, so in the transition system, there is an edge from t, with input i, to t'. The output that was produced is described by the variable $o_{t,i}$. $o_{t,i}$ is assigned to true, iff the state t produces the output with input i the output o. We also listed the variables, that are assigned to true.

Variables	Meaning
$\tau_{t_0,(\bar{r}_1,\bar{r}_2),t_1}, \tau_{t_0,(r_1,\bar{r}_2),t_1}, \tau_{t_0,(\bar{r}_1,r_2),t_1}, \tau_{t_0,(r_1,r_2),t_1}$	There is an edge from t_0 to t_1, with input $(\bar{r}_1, \bar{r}_2)$, one edge with input $(\bar{r}_1, r_2)$, one with input $(r_1, \bar{r}_2)$ and one with (r_1, r_2), so we can put them together to the edge from t_0 to t_1, with input $\top$.
$g1_{t_0,(\bar{r}_1,\bar{r}_2)}, g1_{t_0,(r_1,\bar{r}_2)}, g1_{t_0,(\bar{r}_1,r_2)}, g1_{t_0,(r_1,r_2)}$	The output, of the state t_0 is for all inputs the same: g_1.
$\tau_{t_1,(\bar{r}_1,\bar{r}_2),t_0}, \tau_{t_1,(r_1,\bar{r}_2),t_0}, \tau_{t_1,(\bar{r}_1,r_2),t_0}, \tau_{t_1,(r_1,r_2),t_0}$	There is an edge from t_1 to t_0, with input $\top$, because for all inputs i the variable τ_{t_1,i,t_0} is set to true.
$g2_{t_1,(\bar{r}_1,\bar{r}_2)}, g2_{t_1,(r_1,\bar{r}_2)}, g2_{t_1,(\bar{r}_1,r_2)}, g2_{t_1,(r_1,r_2)}$	The output of the state t_1 is for all inputs the same: g_2

A graphical representation of this formula is shown in figure 2.2b.

Functional Correctness

Theorem 4.2 (Correctness)**.**
Let I/O be the input/output propositions of an LTL formula φ and $b \in \mathbb{N}$ some bound. The transition system T generated by the SAT propositional encoding satisfies φ.

Proof.
We proof that the resulting transition system is accepted by the given universal co-Büchi automaton, with the resulting annotation function λ:

$$\lambda(t,q) = \begin{cases} \bot & \text{if } \lambda^{\mathbb{B}}_{t,q} = 0 \\ n & \text{if } \lambda^{\mathbb{B}}_{t,q} = 1 \wedge \lambda^{\#}_{t,q} = n \end{cases}$$

From the definition of a valid annotation function λ, we know the following conditions have to hold:

1. $\lambda(t_0, q_0) \neq \bot$
2. $\forall t \in T, q \in Q.\ \lambda(t,\ q) = k \neq \bot \rightarrow \forall i \in 2^I.\ \tau(t,\ i) = (o,\ t') \wedge (q,\ i \cup o,\ q') \in \delta \rightarrow \lambda(t', q') \triangleright_{q'} k$

We proof the conditions one by one:

1. $\lambda(t_0, q_0) \neq \bot$ is valid, because of $\lambda^{\mathbb{B}}_{t_0,q_0} = 1$
2. Be (t,q) any valid reachable state in the run graph, $\lambda(t,q) \neq \bot$. For all successors (t', q') of (t,q) it has to hold: $\lambda(t', q') \triangleright_{q'} \lambda(t,q)$. We proof the universal condition, by an arbitrary choose of (t', q'). The same holds for the arbitrary choose of (t,q).

   We have to proof:

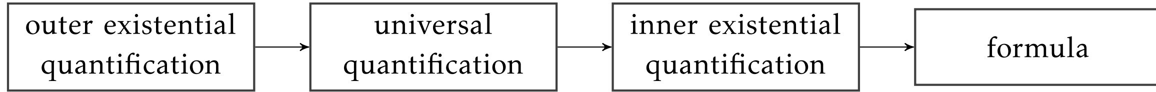


FIGURE 4.2: Structure of the Input-Symbolic Encoding

$$\forall i \in 2^I.\ \tau(t,\ i) = (o,\ t') \wedge (q,\ i \cup o,\ q') \in \delta \rightarrow \lambda(t',q') \triangleright_{q'} \lambda(t,q)$$

If we get an arbitrary input $i \in 2^I$. The premise holds, because of the definitions of the variables:

- $\tau_{t,i,t'} = 1$, because of $\tau(t,i) = (o,t')$
- $o_{t,i} = o$, because of $\tau(t,i) = (o,t')$
- $\delta_{t,q,i,q'} = 1$, because of $\delta_{t,q,i,q'} \leftrightarrow (q,i \cup o_{t,i},q') \in \delta \leftrightarrow (q,i \cup o,q') \in \delta$

If we now look at the formula, we see:

$$\lambda^{\mathbb{B}}_{t,q}\ (=\ 1) \rightarrow \delta_{t,q,i,q'}\ (=\ 1) \rightarrow \tau_{t,i,t'}\ (\ = 1) \rightarrow \lambda^{\mathbb{B}}_{t',q'} \wedge \lambda^{\#}_{t',q'} \triangleright_{q'} \lambda^{\#}_{t,q}$$

We see: $\lambda^{\mathbb{B}}_{t',q'} \wedge \lambda^{\#}_{t',q'} \triangleright_{q'} \lambda^{\#}_{t,q}$ holds and after the definition of $\lambda^{\mathbb{B}}_{t,q}$, $\lambda^{\#}_{t',q'}$ and $\lambda^{\#}_{t',q'}$ $\lambda(t',q') \triangleright_{q'}$ $\lambda(t,q)$, what has to be proven.

□

4.3 QBF Encoding: Input-symbolic Encoding

The last section introduces the explicit encoding, that can be solved by a SAT-solver. In this formula every information is contained in the indices, meaning we build a new variable. One example is the explicit handling of inputs: For every possible input $i \in 2^I$, we build own transition and output variables. The drawback of this explicit handling is the size of the constraint system and the number of variables in the constraint system, that become exponential in the number of inputs.

Building the Formula

This section avoids the problem of the exponential blow up by using quantified boolean formulas (QBF). The input variables are represented symbolically with universal quantifiers. Figure 4.2 defines the general structure of this encoding. We quantify existentially over the λ-annotation variables before the universal quantification. These variables are independent of the input variables. The inner existential quantification over the transition τ and output o variables are dependent on the input variables, so we can omit the indices from these variables.

The semantics of QBF defines the inner existential quantifier variables as boolean functions with the universal quantifier variables as input. For example the variable $\tau_{t,t'}$, representing a transition from state t to state t' is a function $\tau_{t,t'} : 2^I \rightarrow \mathbb{B}$, getting as input a set of input variables i and becomes true, if there is a transition $t \xrightarrow{i} t'$ and false if not. We can also

use another representation of $\delta : (Q \times 2^{I \cup O} \times Q)$ with the propositional formula $\delta_{t,q,q'}$ over the inputs I and output variables o_t, that depend on I. An assignment $i \cup o$ satisfies $\delta_{t,q,q'}$ iff $(q, i \cup o_t, q') \in \delta$.

The following formula represents the input-symbolic encoding. We emphasize the main changes to the explicit encoding with the blue color.

$$
\begin{aligned}
&\exists \{\lambda^{\mathbb{B}}_{t,q}, \lambda^{\#}_{t,q} \mid t \in T, q \in Q\} \\
&\forall I \\
&\exists \{\tau_{t,t'} \mid (t,t') \in T \times T\} \\
&\exists \{o_t \mid o \in O, t \in T\} \\
&\lambda^{\mathbb{B}}_{t_0,q_0} \wedge \bigwedge_{t \in T} \bigvee_{t' \in T} \tau_{t,t'} \\
&\bigwedge_{q \in Q} \bigwedge_{t \in T} \left(\lambda^{\mathbb{B}}_{t,q} \rightarrow \bigwedge_{q' \in Q} \left(\delta_{t,q,q'} \rightarrow \bigwedge_{t' \in T} \left(\tau_{t,t'} \rightarrow \lambda^{\mathbb{B}}_{t',q'} \wedge \lambda^{\#}_{t',q'} \triangleright_{q'} \lambda^{\#}_{t,q} \right)\right)\right)
\end{aligned}
$$

Theorem 4.3 (Size of the Input-Symbolic Encoding [8])**.**
Let $n = |T|$ and $m = |Q|$. The size of the input-symbolic constraint system is in $\mathcal{O}(nm^2(|\delta_{q,q'}| + n\log(nm)))$. The number of existential and universal variables is in $\mathcal{O}(n(m\log(nm) + |O| + n))$ and $\mathcal{O}(|I|)$, respectively.

This constraint system has different advantages besides the smaller size. In comparison to the explicit encoding the input-symbolic encoding does not separate the dependency between the input I and the transition function τ. With this help we can easily summarize all input assignments for one transition from state t to state t'. Another advantage of this encoding is the order of the quantifiers. If we fix the λ-annotation variables we get all possible transition systems satisfying the specification with the fixed annotation function, using the resulting 2QBF query.

Building the Transition System

To build the transition system for a formula, we use several steps:

1. At first we solve the complete formula and if the formula is satisfiable we extract the assignments for the λ-annotation variables.
2. In the next step we instantiate the λ-annotation variables and simplify the resulting formula.
3. In the last step we solve the simplified formula again with a certifying solver, which generates us the boolean functions for $\tau_{t,t'}$ and o_t, which we can translate to a transition system similar to section 4.2.

Example 4.2.
We want to solve the same problem like in example 4.1 with the input-symbolic encoding. Therefore we go through each step in detail.

- The complete formula can be found in appendix 10.3. If we extract the variables $\lambda^{\mathbb{B}}_{t,q}$ and $\lambda^{\#}_{t,q}$, we get the following assignment:

variables	assignment
$\lambda^{\mathbb{B}}_{t_0,q_0}, \lambda^{\mathbb{B}}_{t_1,q_0}$	$\bot$
$\lambda^{\mathbb{B}}_{t_0,q_0}, \lambda^{\mathbb{B}}_{t_1,q_0}, \lambda^{\mathbb{B}}_{t_0,q_1}, \lambda^{\mathbb{B}}_{t_1,q_1}, \lambda^{\mathbb{B}}_{t_0,q_2}, \lambda^{\mathbb{B}}_{t_1,q_2}$	$\top$
$\lambda^{\#}_{t_0,q_0}, \lambda^{\#}_{t_1,q_0}, \lambda^{\#}_{t_0,q_e}, \lambda^{\#}_{t_1,q_e}$	0
$\lambda^{\#}_{t_0,q_2}, \lambda^{\#}_{t_1,q_1}$	1
$\lambda^{\#}_{t_1,q_2}, \lambda^{\#}_{t_0,q_1}$	2

- The instantiation is represented in appendix 10.4.

- We solve this formula and get as result the functions:

functions	value
$\tau_{t_0,t_0}, \tau_{t_1,t_1}, g1_{t_1}, g2_{t_0}$	assign each input to $\bot$
$\tau_{t_0,t_1}, \tau_{t_1,t_0}, g1_{t_0}, g2_{t_1}$	assign each input to $\top$

  With these functions we build the transition system. There are transitions from state t_0 to t_1 and from t_1 to t_0 with every input. The output of state t_0 is g_1 for every possible input, the output of t_1 is g_2. A graphical representation is shown in figure 2.2b.

Functional Correctness

To prove the functional correctness of the QBF constraint system, we use the correctness of the SAT propositional constraint system and claim:

Theorem 4.4 (Correctness)**.**
Let I, O be the input/output propositions of an LTL formula and $b \in \mathbb{N}$ some bound. The SAT propositional and QBF encoding of bounded synthesis are equisatisfiable.

Proof.
$\Rightarrow$
Let T be the transition system generated by the SAT propositional constraint system. To prove that this transition system can also be generated by the QBF constraint system we prove that this system holds on the QBF constraint system with the variable assignment:

- $\lambda^{\mathbb{B}}_{t,q}, \lambda^{\#}_{t,q}$ has the same assignment like in the SAT propositional constraint system
- $\tau_{t,t'}$ is the function, with $\tau_{t,t'}(i) = 1$, iff $\tau_{t,i,t'} = 1$
- o_t is the function, with $o_t(i) = 1$, iff $o_{t,i} = 1$
- $\delta_{t,q,q'}$ is the function, with $\delta_{t,q,q'}(i) = 1$, iff $\delta_{t,q,i,q'} = 1$

We have to prove three conditions:

1. $\lambda^{\mathbb{B}}_{t_0,q_0}$

2. $\bigwedge_{t \in T} \bigvee_{t' \in T} \tau_{t,t'}$

3. $\bigwedge_{q \in Q} \bigwedge_{t \in T} \left(\lambda^{\mathbb{B}}_{t,q} \rightarrow \bigwedge_{q' \in Q} \left(\delta_{t,q,q'} \rightarrow \bigwedge_{t' \in T} \left(\tau_{t,t'} \rightarrow \lambda^{\mathbb{B}}_{t',q'} \wedge \lambda^{\#}_{t',q'} \rhd_{q'} \lambda^{\#}_{t,q} \right) \right) \right)$

The first condition becomes true because $\lambda^{\mathbb{B}}_{t_0,q_0}$ becomes true in the SAT propositional constraint system.

The second and the third condition become true because of the definition of the functions and the corresponding constraints in the SAT propositional constraint system.

$\Leftarrow$
This prove is similar to $\Rightarrow$, with the change that we define the variables in such a way that the functions become true. $\square$

5 Distributed Synthesis

As described in the introduction we translate the different encodings in chapter 4 to the distributed bounded synthesis approach to solve the distributed synthesis problem. Therefore this chapter introduces the distributed synthesis problem. We recap the definitions by Finkbeiner and Schewe [12].

5.1 Architecture

To solve the distributed synthesis problem, we have to define the structure of distributed systems. Distributed systems consist of different processes with different information, meaning a different input and output for each process. For example the variable x could be the output of process 1 and the input of process 2. The architecture describes the distributed system formally.

Definition 5.1 (Architecture).
An architecture A is a tuple (P, env, V, I, O) where:

- P is the set of system processes and the designated environment process
- env is the designated environment process
- V is the set of boolean system variables, with $V = \bigcup_{p \in P} O_p$
- $I = \{I_p \subseteq V \mid p \in P\}$ assigns a set I_p of input variables to each system process
- $O = \{I_p \subseteq V \mid p \in P\}$ assigns a set I_p of input variables to each system process

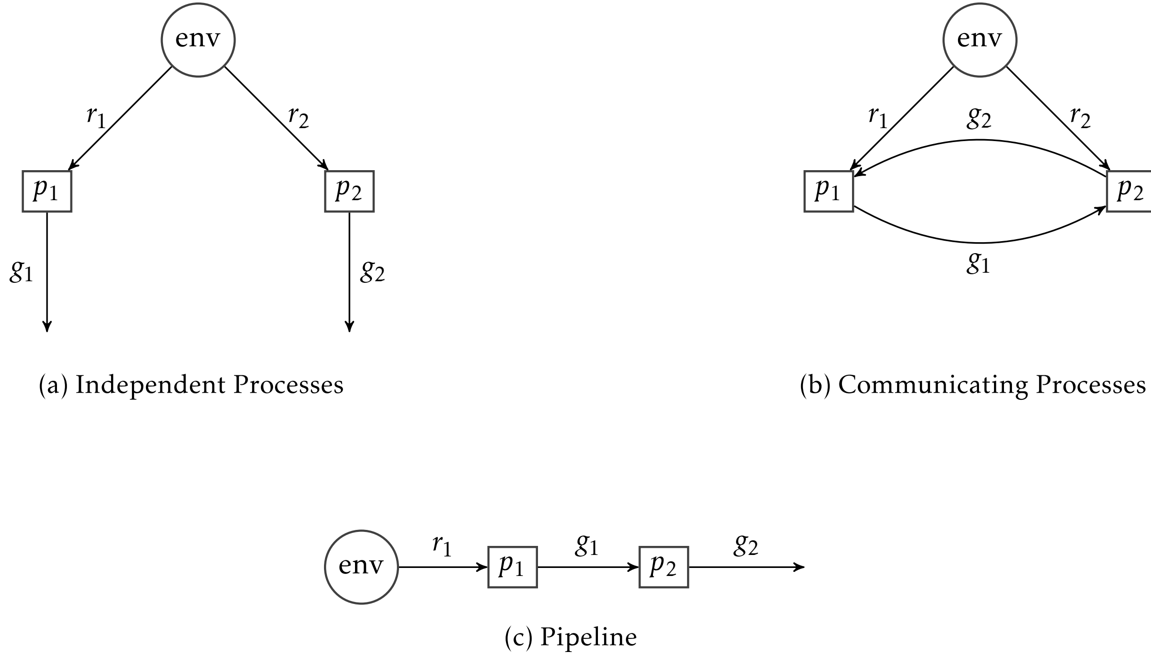
The set of system processes is defined as $P^- = P \setminus \{env\}$.

To indicate broadcasting, it is allowed that the same variable $v \in V$ occurs in multiple input sets I_p. The output sets O_p however are assumed to be pairwise disjoint.

An architecture is called fully informed, if $O_{env} \subseteq I_p$ for every system process $p \in P^-$.

The following examples introdue different architectures. The graphical representations are represented in figure 5.1.

Example 5.1 (Independent Processes).
The architecture in figure 5.1a describes two independent processes. Process p_1 gets as input only the variable r_1, that is an environment output, and produces the output g_1. Process p_2



(a) Independent Processes

(b) Communicating Processes

(c) Pipeline

FIGURE 5.1: Virtual representation of three architectures described in the examples 5.1,5.2 and 5.3

gets as input the environment output r_2 and produces the output g_2. There is no interaction between these two processes.

Formally the architecture A is described by the tuple (P, env, V, I, O), where

- $P = \{env, p_1, p_2\}$
- $V = \{r_1, r_2, g_1, g_2\}$
- $I = \{p_1 \rightarrow \{r_1\}, p_2 \rightarrow \{r_2\}\}$
- $O = \{p_1 \rightarrow \{g_1\}, p_2 \rightarrow \{g_2\}\}$

Example 5.2 (Communicating Processes)**.**
The architecture in figure 5.1b describes two communicating processes. Each process gets as input one environment output and the output of the other system process.

Formally the architecture A is described by the tuple (P, env, V, I, O), where

- $P = \{env, p_1, p_2\}$
- $V = \{r_1, r_2, g_1, g_2\}$
- $I = \{p_1 \rightarrow \{r_1, g_2\}, p_2 \rightarrow \{r_2, g_1\}\}$
- $O = \{p_1 \rightarrow \{g_1\}, p_2 \rightarrow \{g_2\}\}$

Example 5.3 (Pipeline)**.**
The architecture in figure 5.1c describes a pipeline. The first process p_1 gets as input the environment process and produces the output g_1, that is the input for the next process p_2.

Formally the architecture A is described by the tuple (P, env, V, I, O), where

- $P = \{env, p_1, p_2\}$
- $V = \{r_1, g_1, g_2\}$
- $I = \{p_1 \rightarrow \{r_1\}, p_2 \rightarrow \{g_1\}\}$
- $O = \{p_1 \rightarrow \{g_1\}, p_2 \rightarrow \{g_2\}\}$

5.2 Distributed Synthesis

Definition 5.2 (Reactive Realizability from LTL specification)**.**
Given an architecture A and a specification as an LTL formula φ over a set of variables, partitioned into inputs and outputs. The distributed synthesis problem is to decide, whether there is a set of transition systems $\mathcal{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$, one for each process in A, such that the composition $\mathcal{T}_1 \times \cdots \times \mathcal{T}_n$ realizes φ.

Like in chapter 3 the realizability problem is a decision problem. The distributed synthesis returns a set of transition system, if there is a set that realizes the specification. The transition system $\mathcal{T}_1, \ldots, \mathcal{T}_n$ has the input and output sets, as defined in the architecture A. We specify the composition in the next section in detail.

Synthesis of distributed system is in general undecidable [22], even for simple architectures like two communicating processes (compare figure 5.1b). But there are architectures, so that the distributed synthesis is decidable, like the pipeline architecture (compare figure 5.1c) [11] but it has still a non-elementary complexity.

5.3 Composition of Transition Systems

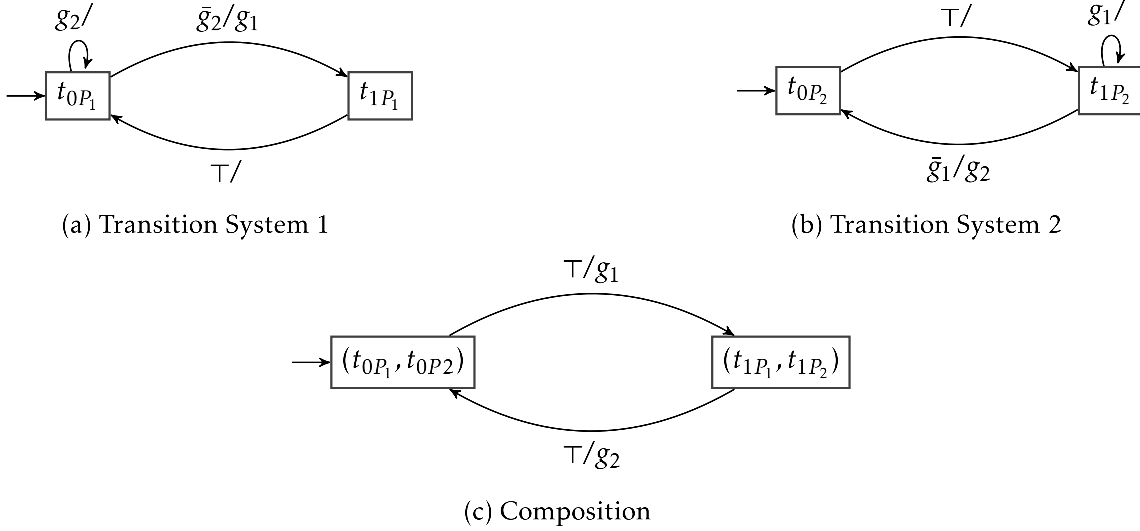
The composition of transition systems $\mathcal{T}_1, \ldots, \mathcal{T}_n$ is also a transition system $\mathcal{T}$.

Definition 5.3 (Transition System of a Distributed Architecture)**.**
Given an architecture $A = (P, env, V, I, O)$ with a set of processes describes by transition systems $\{T_1, \ldots, T_n\}$. The composition of $\{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$ is the transition system $\mathcal{T} = \langle T, t_o, \tau \rangle$ with input $I = O_{env}$ and output $O = V$, where:

- $T = T_1 \times \cdots \times T_n$ is the set of states
- $t_0 = (t_{1_0}, \ldots, t_{n_0})$ is the initial state
- $\tau : T \times 2^I \rightarrow T \times 2^O$ is the transition relation, where

$$\tau((t_1, \ldots, t_n), i) = ((t'_1, \ldots, t'_n), o),$$

iff $\tau_k(t_k, i \cap I_{pk}) = (t'_k, o_k)$ for every $k \in P^-$ and $o = o_1 \cup \cdots \cup o_n$



(a) Transition System 1

(b) Transition System 2

(c) Composition

Figure 5.2: This figure gives two transition systems and the composition between them.

Example 5.4 (Composition of two Transition systems).
We want to build the composition of the transition systems in figure 5.2a and figure 5.2b. The first transition system, with the input set $\{r_1, g_2\}$ and output set $\{g_1\}$, describes the behavior of the first process. The second transition system, with input set $\{r_2, g_1\}$ and output set $\{g_2\}$, describes the second process in architecture 5.2. To check if a specification holds on these processes, we build the composition, which is a transition system with input set $\{r_1, r_2\}$ and output set $\{g_1, g_2\}$. There is a transition from state t in the composition, if in each transition system a transition can be gone. The composition of these two transition systems is the transition system $\mathcal{T} = \langle T, t_0, \tau \rangle$, where:

- $T = \{(t_{0p_1}, t_{0p_2}), (t_{1p_1}, t_{1p_2})\}$
- $t_0 = (t_{0p_1}, t_{0p_2})$
- $\tau = \{$
  $((t_{0p_1}, t_{0p_2}), (r_1, r_2), (g_1, \bar{g}_2), (t_{1p_1}, t_{1p_2})), ((t_{0p_1}, t_{0p_2}), (r_1, \bar{r}_2), (g_1, \bar{g}_2), (t_{1p_1}, t_{1p_2})),$
  $((t_{0p_1}, t_{0p_2}), (\bar{r}_1, r_2), (g_1, \bar{g}_2), (t_{1p_1}, t_{1p_2})), ((t_{0p_1}, t_{0p_2}), (\bar{r}_1, \bar{r}_2), (g_1, \bar{g}_2), (t_{1p_1}, t_{1p_2})),$
  $((t_{1p_1}, t_{1p_2}), (r_1, r_2), (\bar{g}_1, g_2), (t_{0p_1}, t_{0p_2})), ((t_{1p_1}, t_{1p_2}), (r_1, \bar{r}_2), (\bar{g}_1, g_2), (t_{0p_1}, t_{0p_2})),$
  $((t_{1p_1}, t_{1p_2}), (\bar{r}_1, r_2), (\bar{g}_1, g_2), (t_{0p_1}, t_{0p_2})), ((t_{1p_1}, t_{1p_2}), (\bar{r}_1, \bar{r}_2), (\bar{g}_1, g_2), (t_{0p_1}, t_{0p_2}))\},$

if we only look at the reachable part.

A graphical representation can be found in figure 5.2c.

5.4 Challenges with Distributed Systems

By building the transition systems for each process, we have to guarantee that each process can only decide its behavior by the information specified in the architecture. Let us look for example on the architecture in figure 5.1a. The transition system for process p_1 gets as input the variable r_1. The different variable assignments for the variable r_2 have no effect.

There are different ways to solve this problem, Guthoff [17] for example describes two ways to add different constraints to the bounded synthesis formula. In this formula the states of the transition system $\mathcal{T}$ that represent the composition of $\mathcal{T}_1,\ldots,\mathcal{T}_n$ are first built and checked and afterwards the transition system $\mathcal{T}_p$ for each process p is built from $\mathcal{T}$. In this thesis the problem is solved in another way. The single transition systems $\mathcal{T}_1,\ldots,\mathcal{T}_n$ are built and then the composition is checked.

6 Encodings for Distributed Synthesis

In this chapter, we introduce the new encodings for the distributed synthesis problem, based on chapter 4. Like in chapter 4, we start with the verification formula, define new notation and introduce the different encodings.

6.1 Verification

The structure of the verification formula is described in figure 4.1, similar to the single process synthesis. We iterate over all reachable states of the run graph between the composition of the transition systems $\mathcal{T}$ and the states in the universal co-Büchi automaton $\mathcal{A}$ and check if the annotation function λ is valid for all successors in the run graph. The difference to the verification formula in chapter 4 is the transition system, that represents the composition of the different transition systems $\mathcal{T}_1, \ldots, \mathcal{T}_n$, one for each process. Therefore we need new notation:

- For the variables $\lambda^{\mathbb{B}}_{t,q}, \lambda^{\#}_{t,q}$ and $\delta_{q,t,\vec{i},q'}$, there is no change to chapter 4, where $t \in \mathcal{T}$ in the composition of the transition system, $q, q' \in Q$ in the universal co-Büchi automaton and $\vec{i} \in 2^I$.
- As mentioned in chapter 5, we build the transition system for each process first and check, if the composition satisfies the specification. Therefore in the verification formula we have to check if the transition and output variables for each transition system are assigned to true. There are no transition or output variables for the composition of the transition systems. However we iterate over the states of the composition of the transition systems, to check the satisfiability of the composition. We define three functions, to get the information defined in the architecture for only one process:
  - $d : P^- \times T \to T^{P^-}$: This function maps each state t in the composition of the transition systems and a system process p in the given architecture to the specific state t' in process p.
  - $d : P^- \times 2^I \to 2^I$: Simultaneous to the function above, this functions maps the input set to the input set for the system process p, specified in the architecture.
  - $p : I \to P$: In the distributed synthesis problem, we have to cope with the problem, that an input variable of one process can be the output of another process. This function maps an input variable to its producing process.

With these functions we define the variables:

- $\tau^p_{d_p(t),d_p(\vec{i}),d_p(t')}$: These variables represent the transitions from state $d_p(t)$ with input $d_p(\vec{i})$ to state $d_p(t')$ in process p
- $o_{id_{p_i}(t),d_{p_i}(\vec{i})}$: These variables represent the output i of one state $d_{p_i}(t)$ with input $d_{p_i}(\vec{i})$ in process p_i.

These two sets of variables are nearly simultaneous to chapter 4. The difference is the use of the functions defined above. The variables $\delta_{t,q,\vec{i},q'}$ do not change compared to chapter 4.

With these variables we define the verification formula, based on figure 4.1.

$$\bigwedge_{q\in Q}\bigwedge_{t\in T}\Bigg(\lambda^{\mathbb{B}}_{t,q}\rightarrow\bigwedge_{q'\in Q}\bigwedge_{\vec{i}\in 2^I}\Bigg(\Bigg(\bigwedge_{i\in\vec{i}\wedge i\notin O^{env}} i\in\vec{i}\leftrightarrow o_{id_{p_i}(t),d_{p_i}(\vec{i})}\Bigg)\rightarrow\delta_{t,q,\vec{i},q'}$$
$$\rightarrow\bigwedge_{t'\in T}\Bigg(\Bigg(\bigwedge_{p\in P^-}\tau^p_{d_p(t),d_p(\vec{i}),d_p(t')}\Bigg)\rightarrow\lambda^{\mathbb{B}}_{t',q'}\wedge\lambda^{\#}_{t',q'}\rhd_{q'}\lambda^{\#}_{t,q}\Bigg)\Bigg)\Bigg)$$

In comparison to chapter 4, in this formula we added two constraints that are highlighted blue in the formula above:

- $\bigwedge_{p\in P^-}\left(\tau^p_{d_p(t),d_p(\vec{i}),d_p(t')}\right)$
- $\bigwedge_{i\in\vec{i}\wedge i\notin O^{env}}\left(i\in\vec{i}\leftrightarrow o_{id_{p_i}(t),d_{p_i}(\vec{i})}\right)$

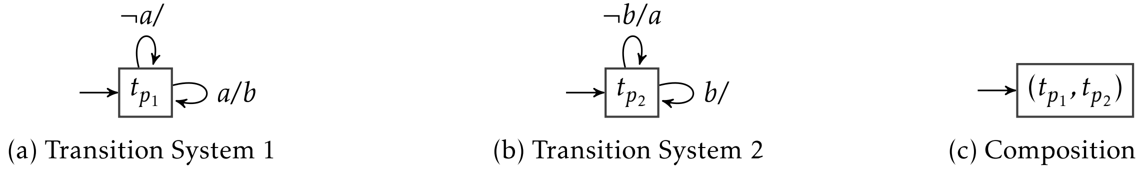
With the first constraint we check in each process if there is a transition. Only if we can go the transition in each process, $\lambda^{\mathbb{B}}_{t',q'}$ has to be reachable. With the second constraint we solve the problem, if an input variable is an output of a system process and not of the environment, we have to take sure that the output variable is set to true, if we consider an input, where this input variable is true and false otherwise. To illustrate this constraint, we look at the architecture 5.1b. Assume, we get as input $\vec{i}=\{r_1,g_2\}$ and t_1 is the state in process 1 and t_2 in process 2. The constraint is translated to:

$$\neg g_{1,t_1,\{r_1,g_2\}}\wedge g_{2,t_2,\{\bar{r}_2,\bar{g}_2\}}$$

In the following, we use the abbreviation valid$(t,\vec{i})$, for the second constraint.

6.2 SAT Propositional Encoding: Explicit Encoding

Like in chapter 4, the verification formula is the base of the synthesis formula.



(a) Transition System 1 (b) Transition System 2 (c) Composition

FIGURE 6.1: This figure gives two transition systems satisfying the constraint, that in each transition system there has to be at least one outgoing transition for every input and their composition.

Constraints

To solve the distributed synthesis with the verification formula above, we have to translate the constraints in chapter 4 to constraints for distributed systems and to expand them.

- $\lambda^{\mathbb{B}}_{t_0,q_0}$: In distributed synthesis we have to assert that the initial state is reachable too. We use the same constraint like in chapter 4, where t_0 is the composition of the initial states.

- $\bigwedge_{t\in T}\bigwedge_{i\in 2^I}\bigvee_{t'\in T}\tau_{t,i,t'}$: To assert that each state has at least one outgoing transition for each valid input i the constraint has to be translated to the composition of the different transition systems:

$$\bigwedge_{t\in T}\bigwedge_{\vec{i}\in 2^I}\text{valid}(t,\vec{i}) \rightarrow \bigvee_{t'\in T}\Big(\bigwedge_{p\in P^-}\tau^p_{d_p(t),d_p(\vec{i}),d_p(t')}\Big)$$

- With the second constraint, we have to assert that the premise $\text{valid}(t,\vec{i})$ does not become always false and preventing, that there are no transitions in the composition of the transition systems:

$$\bigwedge_{t\in T}\bigwedge_{i\in 2^{O^{env}}}\bigvee_{\vec{i}\in 2^I \wedge i\subseteq\vec{i}}\text{valid}(t,\vec{i})$$

  In this constraint, we claim that in every state for every assignment of the environment output, at least one combination of the system output assignments (that are again inputs) becomes true. Without this constraint it would be possible in cyclic architectures, like in figure 5.1b, to choose the output in such a way that the function valid becomes always false.

It is not sufficient to check if in the transition systems for each process, there are in each state at least one outgoing transition for every input. Figure 6.1 gives an counterexample. Both transition systems 6.1a and 6.1b satisfy: $\bigwedge_{t\in T}\bigwedge_{i\in 2^I}\bigvee_{t'\in T} : \tau_{t,i,t'}$. But the composition of these transition systems has no outgoing transition and we do not prevent the problem that a transition system without transitions satisfies every specification, compare section 4.2.

Building the Formula

Like in chapter 5 explained, we have to guess a set of transition systems and an annotation function. This is done like in chapter 4 with an existential quantifier:

$$\exists\{\lambda^{\mathbb{B}}_{t,q}, \lambda^{\#}_{t,q} \mid t \in T, q \in Q\}$$
$$\exists\{\tau^{p}_{d_p(t),d_p(\vec{i}),d_p(t')} \mid (t,t') \in T \times T, \vec{i} \in 2^{I}, p \in P^{-}\}$$
$$\exists\{o_{i,d_{p_i}(t),d_{p_i}(\vec{i})} \mid o_i \in O \backslash O^{env}, t \in T, \vec{i} \in 2^{I}\}$$

The SAT propositional encoding for the distributed synthesis problem can be defined by the following formula:

$$\begin{aligned}
&\exists\{\lambda^{\mathbb{B}}_{t,q}, \lambda^{\#}_{t,q} \mid t \in T, q \in Q\}\\
&\exists\{\tau^{p}_{d_p(t),d_p(\vec{i}),d_p(t')} \mid (t,t') \in T \times T, \vec{i} \in 2^{I}, p \in P^{-}\}\\
&\exists\{o_{i,d_{p_i}(t),d_{p_i}(\vec{i})} \mid o_i \in O \backslash O^{env}, t \in T, \vec{i} \in 2^{I}\}\\
&\lambda^{\mathbb{B}}_{t_0,q_0}\\
&\wedge \bigwedge_{t \in T} \bigwedge_{i \in 2^{O^{env}}} \bigvee_{\vec{i} \in 2^{I} \wedge i \subseteq \vec{i}} \text{valid}(t,\vec{i})\\
&\wedge \bigwedge_{t \in T} \bigwedge_{\vec{i} \in 2^{I}} \text{valid}(t,\vec{i}) \rightarrow \bigvee_{t' \in T} \Big(\bigwedge_{p \in P^{-}} \tau^{p}_{d_p(t),d_p(\vec{i}),d_p(t')} \Big)\\
&\wedge \bigwedge_{q \in Q} \bigwedge_{t \in T} \Bigg(\lambda^{\mathbb{B}}_{t,q} \rightarrow \bigwedge_{q' \in Q} \bigwedge_{\vec{i} \in 2^{I}} \Bigg(\text{valid}(t,\vec{i}) \rightarrow \delta_{t,q,\vec{i},q'} \rightarrow \bigwedge_{t' \in T} \Bigg(\Big(\bigwedge_{p \in P^{-}} \tau^{p}_{d_p(t),d_p(\vec{i}),d_p(t')} \Big) \rightarrow \lambda^{\mathbb{B}}_{t',q'} \wedge \lambda^{\#}_{t',q'} \rhd_{q'} \lambda^{\#}_{t,q} \Bigg) \Bigg) \Bigg)
\end{aligned}$$

Theorem 6.1 (Size of the Explicit Encoding)**.**
The size of the explicit constraint system is in $\mathcal{O}(nm^2 \cdot 2^{|I|} \cdot (|I \backslash O^{env}| + |\delta_{q,q'}| + n \cdot |P^{-}| \cdot \log(nm)))$ and the number of variables is in $\mathcal{O}(nm \log(nm) + n_p \cdot |P^{-}| \cdot 2^{|I^p|} \cdot (|O^p| + n_p))$, where $n = |T|$, $m = |Q|$, I^p the greatest input set, O^p the greatest output set and n_p the greatest size of $T_1, T_2, \ldots$.

Example 6.1 (Examples of Different Architectures with SAT Propositional Formulas)**.**

In this example, we show the SAT propositional formulas of the three architectures, given in figure 5.1. With t_1 and t_1' we define the states in p_1 and with t_2 and t_2' the states in p_2. The states in the composition between the two processes are $t = (t_1, t_2)$ and $t' = (t_1', t_2')$:

- Two independent processes 5.1a: The valid$(t,\vec{i})$ evaluates always to true, because there is no $i \in \vec{i}$ in this architecture. Process p_1 gets as input r_1, $d_{p_1}(\vec{i}) \subseteq 2^{\{r_1\}}$ and process p_2

gets as input r_2, $d_{p_2}(\vec{i}) \subseteq 2^{\{r_2\}}$:

$$
\begin{aligned}
&\exists\{\lambda^{\mathbb{B}}_{t,q}, \lambda^{\#}_{t,q} \mid t \in T, q \in Q\}\\
&\exists\{\tau^{p_1}_{t_1,d_{p_1}(\vec{i}),t_1'}, \tau^{p_2}_{t_2,d_{p_2}(\vec{i}),t_2'} \mid (t,t') \in T \times T, \vec{i} \in 2^I\}\\
&\exists\{g1_{t_1,d_{p_1}(\vec{i})}, g2_{t_2,d_{p_2}(\vec{i})} \mid t \in T, \vec{i} \in 2^I\}\\
&\lambda^{\mathbb{B}}_{t_0,q_0}\\
&\wedge \bigwedge_{t\in T} \bigwedge_{\vec{i}\in 2^I} \bigvee_{t'\in T} \left(\tau^{p_1}_{t_1,d_{p_1}(\vec{i}),t_1'} \wedge \tau^{p_2}_{t_2,d_{p_2}(\vec{i}),t_2'}\right)\\
&\wedge \bigwedge_{q\in Q} \bigwedge_{t\in T} \left(\lambda^{\mathbb{B}}_{t,q} \rightarrow \bigwedge_{q'\in Q} \bigwedge_{\vec{i}\in 2^I} \left(\delta_{t,q,\vec{i},q'} \rightarrow \bigwedge_{t'\in T} \left(\left(\tau^{p_1}_{t_1,d_{p_1}(\vec{i}),t_1'}, \tau^{p_2}_{t_2,d_{p_2}(\vec{i}),t_2'}\right) \rightarrow \lambda^{\mathbb{B}}_{t',q'} \wedge \lambda^{\#}_{t',q'} \rhd_{q'} \lambda^{\#}_{t,q} \right)\right)\right)
\end{aligned}
$$

- Pipeline 5.1c: Process p_2 gets as input g_1, which is the output of process p_1. In this case the function valid$(t,\vec{i})$ changes to $g_1 \leftrightarrow g1_{t_1,d_{p_1}(\vec{i})}$. In addition $d_{p_1}(\vec{i}) \subseteq 2^{\{r_1\}}$ and $d_{p_2}(\vec{i}) \subseteq 2^{\{g_1\}}$:

$$
\begin{aligned}
&\exists\{\lambda^{\mathbb{B}}_{t,q}, \lambda^{\#}_{t,q} \mid t \in T, q \in Q\}\\
&\exists\{\tau^{p_1}_{t_1,d_{p_1}(\vec{i}),t_1'}, \tau^{p_2}_{t_2,d_{p_2}(\vec{i}),t_2'} \mid (t,t') \in T \times T, \vec{i} \in 2^I\}\\
&\exists\{g1_{t_1,d_{p_1}(\vec{i})}, g2_{t_2,d_{p_2}(\vec{i})} \mid t \in T, \vec{i} \in 2^I\}\\
&\lambda^{\mathbb{B}}_{t_0,q_0}\\
&\wedge \bigwedge_{t\in T} \bigwedge_{\vec{i}\in 2^I} (g_1 \leftrightarrow g1_{t_1,d_{p_1}(\vec{i})}) \rightarrow \bigvee_{t'\in T} \left(\tau^{p_1}_{t_1,d_{p_1}(\vec{i}),t_1'} \wedge \tau^{p_2}_{t_2,d_{p_2}(\vec{i}),t_2'}\right)\\
&\wedge \bigwedge_{q\in Q} \bigwedge_{t\in T} \left(\lambda^{\mathbb{B}}_{t,q} \rightarrow \bigwedge_{q'\in Q} \bigwedge_{\vec{i}\in 2^I} \left((g_1 \leftrightarrow g1_{t_1,d_{p_1}(\vec{i})}) \rightarrow \delta_{t,q,\vec{i},q'} \rightarrow \bigwedge_{t'\in T} \left(\left(\tau^{p_1}_{t_1,d_{p_1}(\vec{i}),t_1'}, \tau^{p_2}_{t_2,d_{p_2}(\vec{i}),t_2'}\right) \right.\right.\right.\\
&\left.\left.\left.\rightarrow \lambda^{\mathbb{B}}_{t',q'} \wedge \lambda^{\#}_{t',q'} \rhd_{q'} \lambda^{\#}_{t,q} \right)\right)\right)
\end{aligned}
$$

- two communicating processes 5.1b: In this architecture the two processes communicate together with their output variables. Therefore valid$(t,\vec{i}) = g_1 \leftrightarrow g1_{t_1,d_{p_1}(\vec{i})} \wedge g_2 \leftrightarrow$

$g2_{t_2,d_{p_2}(\vec{i})}$. We define $d_{p_1}(\vec{i}) \subseteq 2^{\{r_1,g_2\}}$ and $d_{p_2}(\vec{i}) \subseteq 2^{\{r_2,g_1\}}$:

$$
\begin{aligned}
&\exists\{\lambda^{\mathbb{B}}_{t,q}, \lambda^{\#}_{t,q} \mid t \in T, q \in Q\}\\
&\exists\{\tau^{p_1}_{t_1,d_{p_1}(\vec{i}),t'_1}, \tau^{p_2}_{t_2,d_{p_2}(\vec{i}),t'_2} \mid (t,t') \in T \times T, \vec{i} \in 2^I\}\\
&\exists\{g1_{t_1,d_{p_1}(\vec{i})}, g2_{t_2,d_{p_2}(\vec{i})} \mid t \in T, \vec{i} \in 2^I\}\\
&\lambda^{\mathbb{B}}_{t_0,q_0}\\
&\wedge \bigwedge_{t\in T} \bigwedge_{\vec{i}\in 2^I} (g_1 \leftrightarrow g1_{t_1,d_{p_1}(\vec{i})} \wedge g_2 \leftrightarrow g2_{t_2,d_{p_2}(\vec{i})}) \rightarrow \bigvee_{t'\in T} \left(\tau^{p_1}_{t_1,d_{p_1}(\vec{i}),t'_1} \wedge \tau^{p_2}_{t_2,d_{p_2}(\vec{i}),t'_2}\right)\\
&\wedge \bigwedge_{q\in Q} \bigwedge_{t\in T} \Bigg(\lambda^{\mathbb{B}}_{t,q} \rightarrow \bigwedge_{q'\in Q} \bigwedge_{\vec{i}\in 2^I} \Bigg((g_1 \leftrightarrow g1_{t_1,d_{p_1}(\vec{i})} \wedge g_2 \leftrightarrow g2_{t_2,d_{p_2}(\vec{i})}) \rightarrow \delta_{t,q,\vec{i},q'}\\
&\rightarrow \bigwedge_{t'\in T} \left(\left(\tau^{p_1}_{t_1,d_{p_1}(\vec{i}),t'_1}, \tau^{p_2}_{t_2,d_{p_2}(\vec{i}),t'_2} \right) \rightarrow \lambda^{\mathbb{B}}_{t',q'} \wedge \lambda^{\#}_{t',q'} \triangleright_{q'} \lambda^{\#}_{t,q} \right)\Bigg)\Bigg)
\end{aligned}
$$

Building the Transition Systems

We build the transition systems $T_1, \ldots, T_n$ synchronous to chapter 4. The indexes of the variables, define the states, input and outputs. The difference to the single process synthesis, is the new index p, to indicate which transition / output is part of one process.

Correctness of Information

We have to prove, that each transition system only depends on the variables specified in the architecture. In this constraint system we define the functions:

- $d : P^- \times I \rightarrow I^p$
- $d : P^- \times T \rightarrow T^p$
- $p : I \rightarrow P$

to map the information.

Functional Correctness

Theorem 6.2 (Correctness)**.**
Let A be an architecture, I/O be the input/output propositions of an LTL formula φ and $b \in \mathbb{N}$ some bound. The set of transition systems $\{T_1, \ldots, T_n\}$ generated by the SAT propositional encoding satisfies φ.

Proof.
The functional correctness proof is similar to the functional correctness proof in section 4.2. The difference is we consider a set of transition systems and prove that the composition of this set of transition systems holds on the given universal co-Büchi automaton with the annotation function λ. We have to prove two conditions:

1. $\lambda(t_0, q_0) \neq \bot$
2. $\forall t \in T, q \in Q.\ \lambda(t,\ q) = k \neq \bot \rightarrow \forall i \in 2^{0^{env}}.\ \tau(t,\ i) = (o,\ t') \wedge (q,\ i \cup o,\ q') \in \delta \rightarrow \lambda(t', q') \triangleright_{q'} k$

with $\tau(t,i) = (o,t')$ and $t = (t_1, \ldots, t_n)$, $t' = (t'_1, \ldots, t'_n)$, $o = o_1 \cup \cdots \cup o_n$, iff $\tau^{p_1}(t_1, i_1) = (o_1, t'_1), \ldots, \tau^{p_n}(t_n, i_n) = (o_n, t'_n)$, where $i \subseteq i_1 \cup \cdots \cup i_2$.

The first condition is part of the constraint system, because the constraint $\lambda^{\mathbb{B}}_{t_0,q_0}$ has to become true. To prove that the second condition holds, we have to prove that

$$\forall i \in 2^{0^{env}}.\ \tau(t,\ i) = (o,\ t') \wedge (q,\ i \cup o,\ q') \in \delta \rightarrow \lambda(t', q') \triangleright_{q'} \lambda(t, q)$$

From the constraint system we know, that there is at least one $\vec{i} \in 2^I$, with $i \subseteq \vec{i}$. Let $i \in 2^{0^{env}}$ and $\vec{i}$, with valid$(t, \vec{i})$ be an arbitrary input and the premise holds. The variable assignment is:

- $\tau^{p_1}_{t_1,i_1,t'_1} = \cdots = \tau^{p_n}_{t_1,i_1,t'_1} = 1$, because of $\tau(t, \vec{i}) = (o, t')$
- $o_{t,i} = 1$, because of $\tau(t, i) = (o, t')$
- $\delta_{t,q,i,q'} = 1$, because of $(q, i \cup o_[t, i]) \in \delta$

From the constraint system we know

$$\lambda^{\mathbb{B}}_{t,q}\ (=\ 1) \rightarrow \text{valid}(t, \vec{i})\ (=\ 1) \rightarrow \delta_{t,q,\vec{i},q'}\ (=\ 1) \rightarrow \tau^{p_1}_{t_1,i_1,t'_1}\ (\ = 1) \wedge \cdots \wedge \tau^{p_n}_{t_n,i_n,t'_n}\ (\ = 1)$$
$$\rightarrow \lambda^{\mathbb{B}}_{t',q'} \wedge \lambda^{\#}_{t',q'} \triangleright_{q'} \lambda^{\#}_{t,q}$$

We see: $\lambda^{\mathbb{B}}_{t',q'} \wedge \lambda^{\#}_{t',q'} \triangleright_{q'} \lambda^{\#}_{t,q}$ holds and after the definition of $\lambda^{\mathbb{B}}_{t,q}$, $\lambda^{\#}_{t',q'}$ and $\lambda^{\#}_{t',q'}$ $\lambda(t', q') \triangleright_{q'} \lambda(t, q)$, what has to be proven. □

6.3 QBF Encoding: Input-symbolic Encoding

The constraint system in the previous section has the drawback that the size of the constraint system and the number of variables become exponential in the number of inputs (compare chapter 4). Therefore we want to build an input-symbolic encoding for distributed systems with QBF. In QBF the dependencies of the variables are defined by the order of the quantification, compare section 2.5. In distributed systems, we have different processes with different informations, in comparison to single processes. This section examines different architectures regarding their compatibility with QBF and their dependencies.

Correctness of Information

To express the input symbolically in the single process synthesis, we quantify universally over the input variables and quantify existentially over the transition and output variables (compare section 4.3). In communicating architectures we have the problem that an input variable of one process can be the output variable of another process. For example in the pipeline architecture, represented in figure 5.1c, the variable g_1 is the output of process 1

and the input for process 2. To express the input symbolically in communicating architectures becomes a challenge. To solve the problem for the pipeline architecture we look at different possibilities:

- We look at the case that g_1 is existentially quantified: In this case the Skolem function for the variable g_2 does not get g_1 as input and it does not describe the architecture.
- We look at the case that g_1 is universally quantified: In this case g_1 is not described by a Skolem function and we cannot describe that g_1 is dependent on r_1.

The solution for this problem is to quantify the variable existentially and universally. Therefore we introduce a new variable i_{g_1} which is universally quantified to describe g_1 as an input variable. The variable g_1 is existentially quantified to describe g_1 as an output variable like in the single process synthesis. If we now change the valid$(t,\vec{i})$ from section 6.2, to

$$\text{valid(t)} = \bigwedge_{o \in \vec{i} \wedge o \notin O^{env}} \left(i_o \leftrightarrow o_{od_{p_i}(t)} \right)$$

we require that the assignment of these variables is equal. Without this constraint we could get a wrong result. For example in the pipeline architecture if $i_{g_1} = 0$ and $g_{1,t} = 1$, the constraint system makes for the variable $g_{2,t'}$ the decision with the input $g_1 = 0$, and not with the correct input $g_1 = 1$.

We compare two processes p_i and p_k, we call p_i less or equal p_k ($p_i \leq p_k$), if and only if $d_{p_1}(I) \subseteq d_{p_2}(I)$. In the next step we examine architectures, where we cannot define a total order on the processes. An example of such an architecture are two independent processes (compare figure 5.1a).

Theorem 6.3 (Correctness of Information)**.**
Let A be an architecture. If and only if there is a total order on the set of processes in A and we introduce new variables for input variables that are system outputs in A, the order for the existentially quantified variables is linear.

Proof.
$\Rightarrow$
If we cannot define a total order on the set of processes, there are at least two processes p_1 with at least one input variable r_1 and an output variable g_1 and p_2 with at least one input variable r_2 and an output variable g_2 such that the output g_1 is independent of the input r_2 and the output g_2 is independent of the input r_1. We look at all permutations:

- $\forall r_1$ stands in the quantifier prefix not before $\exists g_1$ or $\forall r_2$ stands not before $\exists g_2$: In these cases g_1 is independent of r_1 respectively g_2 is independent on r_2 and this doesn't fulfill the claim.
- both universal quantifier are before the existential quantifier: In these cases g_1 is dependent on r_1 and r_2 and doesn't fulfill the claim.
- all permutations of the form $\forall r_1 \exists g_1 \forall r_2 \exists g_2$ respectively $\forall r_2 \exists g_2 \forall r_1 \exists g_1$: In these cases g_2 is dependent on r_1 and r_2 respectively g_1 is dependent on r_1 and r_2 and so doesn't fulfill the claim.

$\Leftarrow$
We assume the order $p_1 \leq p_2 \leq p_n$. If we define the input and output variables in the given order we get the following quantifier prefix:

$$\forall I^{p_1} \exists o^{p_1} \forall (I^{p_2} \backslash I^{p_1}) \exists o^{p_2} \ldots$$

In this case o^{p_1} only depends on I^{p_1} and o^{p_2} depends on I^{p_2}, because $(I^{p_1} \subseteq I^{p_2})$ and the universal quantification stands before the existential. In general: o^{p_k} depends on I^{p_k}. This was the claim.

The same holds for the correct information for the variables $\tau^p_{d_p(t),d_p(t)}$. $\square$

In cyclic architectures there is no total order on the set of processes. Therefore we can omit the constraint $\bigwedge_{t \in T} \bigwedge_{i \in 2^{O^{env}}} \bigvee_{\vec{i} \in 2^I \wedge i \subseteq \vec{i}} \text{valid}(t, \vec{i})$ for the explicit encoding. This constraint prevented that the valid function do not assign each input to fasle, that was only possible in cyclic architectures.

Like in chapter 4, we change the variable $\delta_{t,q,\vec{i},q'}$ to $\delta_{t,q,q'}$, $\tau^p_{t,\vec{i},t'}$ to $\tau^p_{t,t'}$ and $o_{i,d_{p_i}(t),d_{p_i}(\vec{i})}$ to $o_{i,d_{p_i}(t)}$ by removing the input index.

Building the Formula

With this restricted architectures, we get with the order $p_1 \leq \cdots \leq p_k \leq \cdots \leq p_n$ the following formula:

$\exists \{\lambda^{\mathbb{B}}_{t,q}, \lambda^{\#}_{t,q} \mid t \in T, q \in Q\}$
$\forall I^{p_1}$
$\exists \{\tau^{p_1}_{d_p(t),d_p(t')} \mid (t,t') \in T \times T\}$
$\exists \{o_{i,d_{p_i}(t)}\} \mid o_i \in O^{p_1}, t \in T\}$
$\ldots$
$\forall I^{p_k} \backslash (I^{p_1} \cup \cdots \cup I^{p_{k-1}})$
$\exists \{\tau^{p_k}_{d_p(t),d_p(t')} \mid (t,t') \in T \times T\}$
$\exists \{o_{i,d_{p_i}(t)}\} \mid o_i \in O^{p_k}, t \in T\}$
$\ldots$
$\forall I^{p_n} \backslash (I^{p_1} \cup \cdots \cup I^{p_{n-1}})$
$\exists \{\tau^{p_n}_{d_p(t),d_p(t')} \mid (t,t') \in T \times T\}$
$\exists \{o_{i,d_{p_i}(t)}\} \mid o_i \in O^{p_n}, t \in T\}$
$\lambda^{\mathbb{B}}_{t_0,q_0}$
$\wedge \bigwedge_{t \in T} \text{valid}(t) \rightarrow \bigvee_{t' \in T} \Big(\bigwedge_{p \in P^-} \tau^p_{d_p(t),d_p(t')} \Big)$
$\wedge \bigwedge_{q \in Q} \bigwedge_{t \in T} \Bigg(\lambda^{\mathbb{B}}_{t,q} \rightarrow \bigwedge_{q' \in Q} \Bigg(\text{valid}(t) \rightarrow \delta_{t,q,q'} \rightarrow \bigwedge_{t' \in T} \Bigg(\Big(\bigwedge_{p \in P^-} \tau^p_{d_p(t),d_p(t')} \Big) \rightarrow \lambda^{\mathbb{B}}_{t',q'} \wedge \lambda^{\#}_{t',q'} \rhd_{q'} \lambda^{\#}_{t,q} \Bigg) \Bigg) \Bigg)$

Theorem 6.4 (Size of the Input-Symbolic Encoding with QBF).
Let $n = |T|$, $m = |Q|$, O^p the greatest output set and n_p the greatest size of $T_1, T_2, \ldots$. The size of the input-symbolic constraint system with QBF is in $\mathcal{O}(nm^2 \cdot (|I \setminus O^{env}| + |\delta_{q,q'}| + n \cdot |P^-| \cdot \log(nm)))$, the number of existential variables is in $\mathcal{O}(nm \log(nm) + n_p \cdot |P^-| \cdot (|O^p| + n_p))$ and the number of universal variables is in $\mathcal{O}(|I|)$.

Example 6.2 (Examples of different architectures with QBF).
In this example, we show the QBF-formulas of the three architectures, given in figure 6.2, compare example 6.1:

- Two independent processes, with ascending information 6.2a: Because there is no communication in this architecture valid(t) is always true and we can omit it:

$$
\begin{aligned}
&\exists\{\lambda^{\mathbb{B}}_{(t_1,t_2),q}, \lambda^{\#}_{(t_1,t_2),q} \mid ((t_1,t_2) \in T, q \in Q\} \\
&\forall r_1 \\
&\exists\{\tau^{p_1}_{t_1,t_1'} \mid ((t_1,t_2),(t_1',t_2')) \in T \times T\} \\
&\exists\{g_{1\,t_1} \mid (t_1,t_2) \in T\} \\
&\forall r_2 \\
&\exists\{\tau^{p_2}_{t_2,t_2'} \mid ((t_1,t_2),(t_1',t_2')) \in T \times T\} \\
&\exists\{g_{2\,t_2} \mid (t_1,t_2) \in T\} \\
&\lambda^{\mathbb{B}}_{(t_1,t_2)_0,q_0} \\
&\wedge \bigwedge_{(t_1,t_2)\in T} \bigvee_{(t_1',t_2')\in T} \left(\tau^{p_1}_{t_1,t_1'} \wedge \tau^{p_2}_{t_2,t_2'}\right) \\
&\wedge \bigwedge_{q\in Q} \bigwedge_{(t_1,t_2)\in T} \Bigg(\lambda^{\mathbb{B}}_{(t_1,t_2),q} \rightarrow \bigwedge_{q'\in Q} \Bigg(\delta_{(t_1,t_2),q,q'} \rightarrow \bigwedge_{(t_1',t_2')\in T} \Bigg(\Big(\tau^{p_1}_{t_1,t_1'} \wedge \tau^{p_2}_{t_2,t_2'}\Big) \\
&\rightarrow \lambda^{\mathbb{B}}_{(t_1',t_2'),q'} \wedge \lambda^{\#}_{(t_1',t_2'),q'} \rhd_{q'} \lambda^{\#}_{(t_1,t_2),q} \Bigg)\Bigg)\Bigg)
\end{aligned}
$$

- Two independent processes, with the same information 6.2b: Because there is no com-

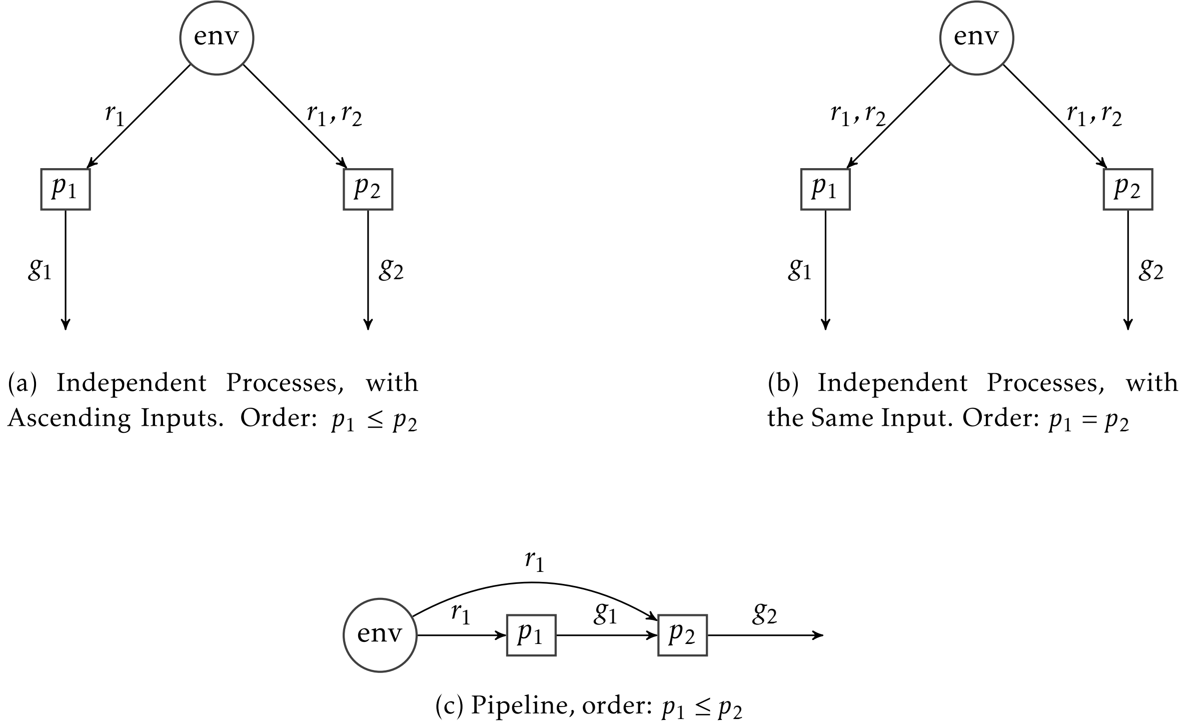
munication in this architecture valid(t) is always true and we can omit it:

$$
\begin{aligned}
&\exists\{\lambda^{\mathbb{B}}_{(t_1,t_2),q}, \lambda^{\#}_{(t_1,t_2),q} \mid ((t_1,t_2) \in T, q \in Q\} \\
&\forall r_1, r_2 \\
&\exists\{\tau^{p_1}_{t_1,t_1'} \mid ((t_1,t_2),(t_1',t_2')) \in T \times T\} \\
&\exists\{{g_1}_{t_1} \mid (t_1,t_2) \in T\} \\
&\exists\{\tau^{p_2}_{t_2,t_2'} \mid ((t_1,t_2),(t_1',t_2')) \in T \times T\} \\
&\exists\{{g_2}_{t_2} \mid (t_1,t_2) \in T\} \\
&\lambda^{\mathbb{B}}_{(t_1,t_2)_0,q_0} \\
&\wedge \bigwedge_{(t_1,t_2)\in T} \bigvee_{(t_1',t_2')\in T} \left(\tau^{p_1}_{t_1,t_1'} \wedge \tau^{p_2}_{t_2,t_2'}\right) \\
&\wedge \bigwedge_{q\in Q} \bigwedge_{(t_1,t_2)\in T} \Bigg(\lambda^{\mathbb{B}}_{(t_1,t_2),q} \rightarrow \bigwedge_{q'\in Q} \Bigg(\delta_{(t_1,t_2),q,q'} \rightarrow \bigwedge_{(t_1',t_2')\in T} \Bigg(\Big(\tau^{p_1}_{t_1,t_1'} \wedge \tau^{p_2}_{t_2,t_2'}\Big) \\
&\rightarrow \lambda^{\mathbb{B}}_{(t_1',t_2'),q'} \wedge \lambda^{\#}_{(t_1',t_2'),q'} \rhd_{q'} \lambda^{\#}_{(t_1,t_2),q} \Bigg)\Bigg)\Bigg)
\end{aligned}
$$

In the previous example p_1 gets only r_1 as input, and p_2 r_1 and r_2. The universal quantification is done after the output and the transition quantification for process p_1. In this example p_1 and p_2 have the same information and we quantify r_1 and r_2 together.

- Pipeline 6.2c: Because process p_2 gets the output g_1 of p_1 as input, we use a new input variable i_{g_1}. The function valid(t) is in this case $(i_{g_1} \leftrightarrow {g_1}_t)$. We highlighted this function blue in the formula:

$$
\begin{aligned}
&\exists\{\lambda^{\mathbb{B}}_{(t_1,t_2),q}, \lambda^{\#}_{(t_1,t_2),q} \mid ((t_1,t_2) \in T, q \in Q\} \\
&\forall r_1 \\
&\exists\{\tau^{p_1}_{t_1,t_1'} \mid ((t_1,t_2),(t_1',t_2')) \in T \times T\} \\
&\exists\{{g_1}_{t_1} \mid (t_1,t_2) \in T\} \\
&\forall i_{g_1} \\
&\exists\{\tau^{p_2}_{t_2,t_2'} \mid ((t_1,t_2),(t_1',t_2')) \in T \times T\} \\
&\exists\{{g_2}_{t_2} \mid (t_1,t_2) \in T\} \\
&\lambda^{\mathbb{B}}_{(t_1,t_2)_0,q_0} \\
&\wedge \bigwedge_{(t_1,t_2)\in T} (i_{g_1} \leftrightarrow {g_1}_{t_1}) \rightarrow \bigvee_{(t_1',t_2')\in T} \left(\tau^{p_1}_{t_1,t_1'} \wedge \tau^{p_2}_{t_2,t_2'}\right) \\
&\wedge \bigwedge_{q\in Q} \bigwedge_{(t_1,t_2)\in T} \Bigg(\lambda^{\mathbb{B}}_{t,q} \rightarrow \bigwedge_{q'\in Q} \Bigg((i_{g_1} \leftrightarrow {g_1}_{t_1}) \rightarrow \delta_{(t_1,t_2),q,q'} \rightarrow \bigwedge_{t'\in T} \Bigg(\Big(\tau^{p_1}_{t_1,t_1'} \wedge \tau^{p_2}_{t_2,t_2'}\Big) \\
&\rightarrow \lambda^{\mathbb{B}}_{(t_1',t_2'),q'} \wedge \lambda^{\#}_{(t_1',t_2'),q'} \rhd_{q'} \lambda^{\#}_{(t_1,t_2),q} \Bigg)\Bigg)\Bigg)
\end{aligned}
$$



(a) Independent Processes, with Ascending Inputs. Order: $p_1 \leq p_2$

(b) Independent Processes, with the Same Input. Order: $p_1 = p_2$

(c) Pipeline, order: $p_1 \leq p_2$

FIGURE 6.2: Three Examples for Architectures with a Total Order

Building the transition system

We build the transition systems as explained in 4.3 and in 6.2.

Functional Correctness

Theorem 6.5 (Correctness)**.**
Let A be an architecture with a total order on the processes, I/O be the input/output propositions of an LTL formula and $b \in \mathbb{N}$ some bound. The SAT propositional and QBF encodings of distributed bounded synthesis are equisatisfiable.

Proof.
$\Leftarrow$
Let $\{T_1, \ldots, T_n\}$ be the set of transition systems generated by the QBF constraint system. To prove that this set of transition systems can also be generated by the SAT propositional encoding, we prove that this set of transition systems satisfies the SAT propositional constraint system with the following variable assignment:

- $\lambda^{\mathbb{B}}_{t,q}, \lambda^{\#}_{t,q}$ has the same assignment like in the QBF constraint system
- $\tau^{p}_{d_p(t), d_p(\vec{i}), d_p(t')} = 1$, iff $\tau^{p}_{d_p(t), d_p(t')}(d_p(\vec{i})) = 1$
- $o_{i, d_{p_i}(t), d_{p_i}(\vec{i})} = 1$, iff $o_{i, d_{p_i}(t)}(d_{p_i}(\vec{i})) = 1$
- $\delta_{t,q,\vec{i},q'} = 1$, iff $\delta_{t,q,q'}(\vec{i}) = 1$

We have to prove 4 conditions:

1. $\lambda^{\mathbb{B}}_{t_0,q_0}$
2. $\bigwedge_{t\in T}\bigwedge_{i\in 2^{O^{env}}}\bigvee_{\vec{i}\in 2^I \wedge i\subseteq\vec{i}}$ valid$(t,\vec{i})$
3. $\bigwedge_{t\in T}\bigwedge_{\vec{i}\in 2^I}$ valid$(t,\vec{i}) \rightarrow \bigvee_{t'\in T}\left(\bigwedge_{p\in P^-}\tau^p_{d_p(t),d_p(\vec{i}),d_p(t')}\right)$
4. $\bigwedge_{q\in Q}\ \bigwedge_{t\in T}\left(\lambda^{\mathbb{B}}_{t,q}\rightarrow\bigwedge_{q'\in Q}\ \bigwedge_{\vec{i}\in 2^I}\left(\text{valid}(t,\vec{i})\rightarrow\delta_{t,q,\vec{i},q'}\rightarrow\bigwedge_{t'\in T}\left(\left(\bigwedge_{p\in P^-}\tau^p_{d_p(t),d_p(\vec{i}),d_p(t')}\right)\rightarrow \lambda^{\mathbb{B}}_{t',q'}\wedge\lambda^{\#}_{t',q'}\rhd_{q'}\lambda^{\#}_{t,q}\right)\right)\right)$

The first condition becomes true, because λ_{t_0,q_0} has the same assignment like in the QBF encoding, where is becomes true.

The second constraint system becomes true, because of the defined architecture. This constraint system asserts that valid$(t,\vec{i})$ does not always becomes false. This is only possible in cyclic architectures, where we cannot define a total order.

The third and the fourth condition become true because of the definition of the variables and the corresponding constraints in the QBF constraint system.

$\Rightarrow$
This proof is similar to $\Leftarrow$ with the difference that we define the functions in such a way that the variables become true. In addition the second condition is omitted in the QBF encoding. □

6.4 DQBF Encoding: Input-symbolic Encoding

The drawback of the input-symbolic encoding with QBF is that we can only solve architectures with a total order. In QBF we use this order to express the different information of the processes. In DQBF we can define these dependencies explicitly, and do not need this order. Therefore if we use DQBF we can solve the distributed synthesis problem for all distributed systems and remain symbolic in the input. To express the dependencies the variables $\delta_{t,q,q'}$ change to $\delta_{t,q,q'}(I)$, $\tau^p_{d_p(t),d_p(t')}$ to $\tau^p_{d_p(t),d_p(t')}(I^p)$ and $o_{i,d_{p_i}(t)}$ to $o_{i,d_{p_i}(t)}(I^p)$.

If we express the input symbolically with DQBF we get the same problems with communicating architectures like in the input-symbolic encoding with QBF (compare section 6.3). Therefore we use the same new variables for input variables, that are also system outputs and use the function valid(t) to check if the variables representing the input and output are set to the same value.

The input-symbolic encoding with DQBF can solve the distributed synthesis problem for all architectures, in contrast to the previous section, where no constraint system for cyclic architectures could be build. Therefore we get the same problem like in the explicit encoding. In cyclic architectures it is possible, that the valid function is always false, if we do not

prevent this. Therefore we have to transform the constraint

$$\bigwedge_{t\in T}\ \bigwedge_{i\in 2^{O^{env}}}\ \bigvee_{\vec{i}\in 2^I \wedge i\subseteq \vec{i}} \text{valid}(t,\vec{i})$$

of the explicit encoding (compare section 6.2) to an input-symbolic constraint. To express the disjunction in the new constraint we have to introduce new variables $o'_{i,t}(O^{env})$ which are existentially quantified for every $t\in T$ and every input variable that is a system output. These variables are the same variables like the output variables with the difference that they are only dependent on the environment output. With these variables we can build a constraint to prevent that valid(t) is always false. Therefore the constraint changes to

$$\bigwedge_{t\in T}\Big(\Big(\bigwedge_{i\in I\wedge i\notin O^{env}} i\leftrightarrow o'_{i,t}(O^{env})\Big)\rightarrow \text{valid}(t)\Big)$$

These new variables $o'_{i,t}(O^{env})$ are only used for modeling the architecture on the constraint system and are not translated to build the transition system or the annotation function. This is a difference to the previous encodings presented in this thesis.

Building the Formula

We use the same constraint system like in section 6.3. The changed variables are highlighted blue:

$$\exists\{\lambda^{\mathbb{B}}_{t,q},\lambda^{\#}_{t,q}\mid t\in T, q\in Q\}$$
$$\forall I$$
$$\exists\{\tau^{p}_{d_p(t),d_p(t')}(I^p)\mid (t,t')\in T\times T, p\in P^-\}$$
$$\exists\{o_{i,d_{p_i}(t)}(I^{p_i})\mid o_i\in O^{p_1}, t\in T\}$$
$$\exists\{o'_{i,t}(O^{env})\mid o_i\in (O^{p_1}\cap I), t\in T\}$$
$$\lambda^{\mathbb{B}}_{t_0,q_0}$$
$$\wedge\bigwedge_{t\in T}\Big(\Big(\bigwedge_{i\in I\wedge i\notin O^{env}} i\leftrightarrow o'_{i,t}(O^{env})\Big)\rightarrow \text{valid}(t)\Big)$$
$$\wedge\bigwedge_{t\in T}\Big(\text{valid}(t)\rightarrow \bigvee_{t'\in T}\Big(\bigwedge_{p\in P^-}\tau^{p}_{d_p(t),d_p(t')}(I^p)\Big)\Big)$$
$$\wedge\bigwedge_{q\in Q}\bigwedge_{t\in T}\Bigg(\lambda^{\mathbb{B}}_{t,q}\rightarrow\bigwedge_{q'\in Q}\Bigg(\text{valid}(t)\rightarrow\delta_{t,q,q'}(I)\rightarrow\bigwedge_{t'\in T}\Bigg(\Bigg(\bigwedge_{p\in P^-}\tau^{p}_{d_p(t),d_p(t')}(I^p)\Bigg)\rightarrow\lambda^{\mathbb{B}}_{t',q'}\wedge\lambda^{\#}_{t',q'}\rhd_{q'}\lambda^{\#}_{t,q}\Bigg)\Bigg)\Bigg)$$

Theorem 6.6 (Size of the Input-Symbolic Encoding with DQBF)**.**
Let $n=|T|$, $m=|Q|$, O^p the greatest output set and n_p the greatest size of $T_1,T_2,\ldots$. The size of the input-symbolic constraint system with DQBF is in $\mathcal{O}(nm^2\cdot(|I\backslash O^{env}|+|\delta_{q,q'}|+n\cdot|P^-|\cdot\log(nm)))$, the number of existential variables is in $\mathcal{O}(nm\log(nm)+n_p\cdot|P^-|\cdot 2^{|I^p|}\cdot(2\cdot|O^p|+n_p))$ and the number of universal variables is in $\mathcal{O}(|I|)$.

Example 6.3 (Examples of different architectures with DQBF)**.**
In this example, we show the DQBF-formulas of the three architectures, given in figure 5.1, compare example 6.1 and 6.2.

- Two independent processes 5.1a. Because there is no communication in this architecture valid(t) is always true and we can omit it. In this architecture $I \cup O = \emptyset$, so we do not introduce new variables:

$$\begin{aligned}
&\exists\{\lambda^{\mathbb{B}}_{(t_1,t_2),q}, \lambda^{\#}_{(t_1,t_2),q} \mid ((t_1,t_2) \in T, q \in Q\} \\
&\forall r_1, r_2 \\
&\exists\{\tau^{p_1}_{t_1,t_1'}(r_1), \tau^{p_2}_{t_2,t_2'}(r_2) \mid ((t_1,t_2),(t_1',t_2')) \in T \times T\} \\
&\exists\{g1_{t_1}(r_1), g2_{t_2}(r_2) \mid (t_1,t_2) \in T\} \\
&\lambda^{\mathbb{B}}_{(t_1,t_2)_0,q_0} \\
&\wedge \bigwedge_{(t_1,t_2)\in T} \bigvee_{(t_1',t_2')\in T} \left(\tau^{p_1}_{t_1,t_1'}(r_1) \wedge \tau^{p_2}_{t_2,t_2'}(r_2)\right) \\
&\wedge \bigwedge_{q\in Q} \bigwedge_{(t_1,t_2)\in T} \left(\lambda^{\mathbb{B}}_{(t_1,t_2),q} \rightarrow \bigwedge_{q'\in Q} \left(\delta_{(t_1,t_2),q,q'}(r_1,r_2) \rightarrow \bigwedge_{(t_1',t_2')\in T} \left(\left(\tau^{p_1}_{t_1,t_1'}(r_1) \wedge \tau^{p_2}_{t_2,t_2'}(r_2)\right) \right.\right.\right. \\
&\left.\left.\left.\rightarrow \lambda^{\mathbb{B}}_{(t_1',t_2'),q'} \wedge \lambda^{\#}_{(t_1',t_2'),q'} \rhd_{q'} \lambda^{\#}_{(t_1,t_2),q} \right)\right)\right)
\end{aligned}$$

- Two communicating processes 5.1b. This architecture has two communicating processes, meaning there are two variables that are output and input. The function valid(t) is evaluated to $i_{g_1} \leftrightarrow g1_{t_1}(r_1, i_{g_2}) \wedge i_{g_2} \leftrightarrow g2_{t_2}(r_2, i_{g_1})$. Because g_1 is input of process 1 and g_2 is input of process 2, we have to introduce the variables i_{g_1}, i_{g_2}, $g'_{1,t}$ and $g'_{2,t}$:

$$\begin{aligned}
&\exists\{\lambda^{\mathbb{B}}_{(t_1,t_2),q}, \lambda^{\#}_{(t_1,t_2),q} \mid ((t_1,t_2) \in T, q \in Q\} \\
&\forall r_1, r_2, i_{g_1}, i_{g_2} \\
&\exists\{\tau^{p_1}_{t_1,t_1'}(r_1, i_{g_2}), \tau^{p_2}_{t_2,t_2'}(r_2, i(g_1)) \mid ((t_1,t_2),(t_1',t_2')) \in T \times T\} \\
&\exists\{g1_{t_1}(r_1, i_{g_2}), g2_{t_2}(r_2, i_{g_1}) \mid (t_1,t_2) \in T\} \\
&\exists\{g1'_{(t_1,t_2)}(r_1,r_2), g2'_{(t_1,t_2)}(r_1,r_2) \mid (t_1,t_2) \in T\} \\
&\lambda^{\mathbb{B}}_{(t_1,t_2)_0,q_0} \\
&\wedge \bigwedge_{(t_1,t_2)\in T} \Big((i_{g_1} \leftrightarrow g1'_{(t_1,t_2)}(r_1,r_2) \wedge i_{g_2} \leftrightarrow g2'_{(t_1,t_2)}(r_1,r_2)) \rightarrow (i_{g_1} \leftrightarrow g1_{t_1}(r_1,i_{g_2}) \wedge i_{g_2} \leftrightarrow g2_{t_2}(r_2,i_{g_1}))\Big) \\
&\wedge \bigwedge_{(t_1,t_2)\in T} \Big((i_{g_1} \leftrightarrow g1_{t_1}(r_1,i_{g_2}) \wedge i_{g_2} \leftrightarrow g2_{t_2}(r_2,i_{g_1})) \rightarrow \bigvee_{(t_1',t_2')\in T} \left(\tau^{p_1}_{t_1,t_1'}(r_1,i_{g_2}) \wedge \tau^{p_2}_{t_2,t_2'}(r_2,i_{g_1})\right)\Big) \\
&\wedge \bigwedge_{q\in Q} \bigwedge_{(t_1,t_2)\in T} \left(\lambda^{\mathbb{B}}_{(t_1,t_2),q} \rightarrow \bigwedge_{q'\in Q} \left((i_{g_1} \leftrightarrow g1_{t_1}(r_1,i_{g_2}) \wedge i_{g_2} \leftrightarrow g2_{t_2}(r_2,i_{g_1})) \rightarrow \delta_{(t_1,t_2),q,q'}(r_1,r_2,i_{g_1},i_{g_2}) \right.\right. \\
&\left.\left.\rightarrow \bigwedge_{(t_1',t_2')\in T} \left(\left(\tau^{p_1}_{t_1,t_1'}(r_1,i_{g_2}) \wedge \tau^{p_2}_{t_2,t_2'}(r_2,i_{g_1})\right) \rightarrow \lambda^{\mathbb{B}}_{(t_1',t_2'),q'} \wedge \lambda^{\#}_{(t_1',t_2'),q'} \rhd_{q'} \lambda^{\#}_{(t_1,t_2),q} \right)\right)\right)
\end{aligned}$$

- Pipeline 5.1c. This architecture has a system output as input so valid(t) is $i_{g_1} \leftrightarrow g1_{t_1}(r_1)$

and we introduce the variables i_{g_1} and ${g_1}'_{(t_1,t_2)}$.

$$
\begin{aligned}
&\exists\{\lambda^{\mathbb{B}}_{(t_1,t_2),q}, \lambda^{\#}_{(t_1,t_2),q} \mid ((t_1,t_2) \in T, q \in Q\} \\
&\forall r_1, i_{g_1} \\
&\exists\{\tau^{p_1}_{t_1,t'_1}(r_1), \tau^{p_2}_{t_2,t'_2}(i_{g_1}) \mid ((t_1,t_2),(t'_1,t'_2)) \in T \times T\} \\
&\exists\{{g_1}_{t_1}(r_1), {g_2}_{t_2}(i_{g_1}) \mid (t_1,t_2) \in T\} \\
&\exists\{{g_1}'_{(t_1,t_2)}(r_1) \mid (t_1,t_2) \in T\} \\
&\lambda^{\mathbb{B}}_{(t_1,t_2)_0,q_0} \\
&\wedge \bigwedge_{(t_1,t_2)\in T} \Big((i_{g_1} \leftrightarrow {g_1}'_{(t_1,t_2)}(r_1)) \rightarrow (i_{g_1} \leftrightarrow {g_1}_{t_1}(r_1))\Big) \\
&\wedge \bigwedge_{(t_1,t_2)\in T} \Big((i_{g_1} \leftrightarrow {g_1}_{t_1}(r_1)) \rightarrow \bigvee_{(t'_1,t'_2)\in T} \big(\tau^{p_1}_{t_1,t'_1}(r_1) \wedge \tau^{p_2}_{t_2,t'_2}(i_{g_1})\big)\Big) \\
&\wedge \bigwedge_{q\in Q} \bigwedge_{(t_1,t_2)\in T} \Bigg(\lambda^{\mathbb{B}}_{(t_1,t_2),q} \rightarrow \bigwedge_{q'\in Q} \Bigg(i_{g_1} \leftrightarrow {g_1}_{t_1}(r_1) \rightarrow \delta_{(t_1,t_2),q,q'}(r_1, i_{g_1}) \\
&\rightarrow \bigwedge_{(t'_1,t'_2)\in T} \Big(\big(\tau^{p_1}_{t_1,t'_1}(r_1) \wedge \tau^{p_2}_{t_2,t'_2}(i_{g_1})\big) \rightarrow \lambda^{\mathbb{B}}_{(t'_1,t'_2),q'} \wedge \lambda^{\#}_{(t'_1,t'_2),q'} \trianglerighteq_{q'} \lambda^{\#}_{(t_1,t_2),q}\Big)\Bigg)\Bigg)
\end{aligned}
$$

Building the transition system

Building the transition system is in this constraint system, like in the QBF constraint system 6.3. With the transition functions $\tau^{p}_{t,t'}$ we build the transitions from state t to state t' in process p and with $o_{i,t}$ we build the output of the state t.

Correctness of Information

In the DQBF constraint system we can define the dependencies explicitly, so the variables get only the dependencies, defined in the architecture.

Functional Correctness

Theorem 6.7 (Correctness)**.**
Let A be an architecture, I/O be the input/output propositions of an LTL formula and $b \in \mathbb{N}$ some bound. The SAT propositional and DQBF encodings of distributed bounded synthesis are equisatisfiable.

Proof.
$\Rightarrow$
Let $\{T_1, \ldots, T_n\}$ be the set of transition systems generated by the SAT propositional constraint system. To prove that this set of transition systems can also be generated by the DQBF encoding, we prove that this set of transition systems satisfies the DQBF constraint system with the following variable assignment:

- $\lambda^{\mathbb{B}}_{t,q}, \lambda^{\#}_{t,q}$ has the same assignment like in the QBF constraint system

- $\tau^p_{d_p(t),d_p(t')}(d_p(\vec{i})) = 1$, iff $\tau^p_{d_p(t),d_p(\vec{i},d_p(t')} = 1$
- $o_{i,d_{p_i}(t)}(d_{p_i}(\vec{i})) = 1$, iff $o_{i,d_{p_i}(t),d_{p_i}(\vec{i})} = 1$
- $\delta_{t,q,\vec{i},q'} = 1$, iff $\delta_{t,q,q'}(\vec{i}) = 1$
- $o'_{i,d_{p_i}(t),d_{p_i}}(i') = 1$, iff $\bigvee_{\vec{i}\in 2^I \wedge i' \in \vec{i}} o_{i,d_{p_i}(t),d_{p_i}(\vec{i})} = 1$

We have to proof four conditions:

1. $\lambda^{\mathbb{B}}_{t_0,q_0}$
2. $\bigwedge_{t\in T}\Big(\big(\bigwedge_{i\in I\wedge i\notin O^{env}} i \leftrightarrow o'_{i,t}(O^{env})\big) \to \mathrm{valid}(t)\Big)$
3. $\bigwedge_{t\in T}\Big(\mathrm{valid}(t) \to \bigvee_{t'\in T}\big(\bigwedge_{p\in P^-} \tau^p_{d_p(t),d_p(t')}(I^p)\big)\Big)$
4. $\wedge \bigwedge_{q\in Q}\ \bigwedge_{t\in T}\ \Bigg(\lambda^{\mathbb{B}}_{t,q} \to \bigwedge_{q'\in Q}\Bigg(\mathrm{valid}(t) \to \delta_{t,q,q'}(I) \to \bigwedge_{t'\in T}\Big(\big(\bigwedge_{p\in P^-} \tau^p_{d_p(t),d_p(t')}(I^p)\big)$
   $\to \lambda^{\mathbb{B}}_{t',q'} \wedge \lambda^{\#}_{t',q'} \rhd_{q'} \lambda^{\#}_{t,q}\Big)\Bigg)\Bigg)$

The first condition becomes true, because λ_{t_0,q_0} has the same assignment like in the QBF encoding, where it becomes true.

The last three constraints become true because of the definition of the functions and the corresponding constraints in the SAT propositional constraint system.

$\Leftarrow$
This proof is similar to $\Rightarrow$ with the difference that we define the variables in such a way that the functions become true. In addition the functions $o'_{i,d_{p_i}(t)}$ are not used in the SAT propositional encoding. □

7 Implementation & Results

This chapter consist of two parts. We give a short overview about the implementation for the formulas in chapter 6. The other part is the evaluation. We have tested our implementation with different architectures and different specifications. We explain shortly the specifications and represent the results of our evaluation.

7.1 BoSy

The implementation is based on the reactive synthesis tool BoSy [9]. BoSy is a tool to solve the bounded synthesis approach [12]. It builds different constraint systems in different logic languages like the SMT encoding [12] or the SAT propositional encoding [8]. To build these encodings, at the moment two tools (ltl3ba [1] or spot [6]) are used to translate the LTL specification to a universal co-Büchi automata. These constraint systems can in the next step be solved by different solvers in the corresponding logic. We added three new constraint systems to solve the distributed synthesis, the distributed explicit encoding with SAT propositional formulas, the distributed input-symbolic encoding with QBF and the distributed input-symbolic encoding with DQBF. These encodings follow the formulas explained in chapter 6.

BoSy is implemented in Swift 3.0. Each constraint system has his own `struct` satisfying the `protocol`:

```
protocol BoSyEncoding {

    mutating func solve(forBound bound: Int) throws -> Bool
    func extractSolution() -> TransitionSystem?

}
```

The solve function gets as input the current bound and returns true, if the specification is realizable with this bound, and false if not. For each of our three cases at first we call the function `getEncoding(forBound bound: Int)`, where the constraint system is built. The return type of this function is the `protocol Logic`. This protocol is satisfied by the boolean operators, quantifiers, propositions, literals and function applications (to solve the formulas for the QBF and the DQBF constraint system). After the `getEncoding` call the `solve` function calls the solve function of the specified solver, which translates the constraint system into the input format of the solver and solves the formula. If the formula is satisfiable the solution bound is set to the current bound, the returned assignment of the solver is set and in the QBF-encoding we set the current formula.

The function `extractSolution` builds the transition systems. Its return type is `TransitionSystem` which can be returned in our case as a .dot output in the command line. The return type is an optional type, because if the transition system cannot be built, for example if the `solve` function in the QBF encoding does not return true yet and the assignment was not set, the `extractSolution` function returns nil. We look at the different `extractSolution` functions in more detail:

- If we call this function in the SAT propositional encoding we check at first if the assignment was set by the `solve` function. Afterwards we build the transitions and the outputs with the variables and their assignments. This is done as explained in section 6.2.
- In the QBF encoding this function checks at first if the assignment and the instance (the formula that was satisfiable) was set. Afterward it checks if the current solver is a QBF solver. After the check, we solve the instance with the given solver. If the result is satisfiable, the solver returns an assignment for the variables of the type $\lambda^{\mathbb{B}}_{t,q}$ and $\lambda^{\#}_{t,q}$. We evaluate the instance with the returned assignment and get a formula of the type $\forall X \exists Y.\varphi$. This formula can be solved by a certified QBF-solver, which returns the functions: $\tau^{p}_{t,t'}$ and $o_{i,t}$. These functions can (in the next step) be translated into a transition system (compare section 6.3)
- In the DQBF encoding, the `extractSolution` function is currently not implemented.

Input Format

BoSy takes a `.bosy` file as input. This file has in general the following structure:

```
1 {
2   "semantics": "mealy" | "moore",
3   "inputs": [ "r0", "r1", ... ],
4   "outputs": [ "g0", "g1", ... ],
5   "assumptions": [ ... ],
6   "guarantees": [...]
7 }
```

With the first element we define the semantics of the transition system, with the other elements we define the specification. At the moment the constraint system is only implemented for the mealy semantics. The transition system has to satisfy the specification

$$(\text{assumption}_0 \wedge \text{assumption}_1 \wedge \dots) \rightarrow (\text{guarantee}_0 \wedge \text{guarantee}_1 \wedge \dots)$$

over the input set `inputs` $\cup$ `outputs`. For the distributed synthesis we expanded the input format with a new element `architecture`:

```
1   "architecture" : {
2     "processes" : [ "p0", "p1", ...],
3     "input"     : { "p0" : [ ... ], "p1" : [ ... ], ...},
4     "output"    : { "env" : ["r_0", "r_1", ...], "p0" : [ ... ], "p1" : [ ... ]}
5     "bound"     : { "p0": ... , "p1": ... , ... }
6   }
```

This input format defines the architecture, as explained theoretically in section 5.1. With the element `processes` we define the set of system processes. The environment process is defined as a constant called `env`. The elements `input` and `output` map each process to its input / output set. The element `bound` is optional. We map each process to a natural number. With this number we define a factor how we increase the size of this process with the next bound. If this element is not defined the bound of each process is set to 1.

7.2 Evaluation

Setup

We have tested the different encodings with the EDACC framework [3] and the following configuration:

- Our machine has a 3.6 GHz quad-core Intel Xeon processor with 32 GB memory.
- We limited the evaluation with a timeout of 1 hour and a memout of 8 GB
- We use Spot 2.0 [6] to translate the LTL specification to a universal co-Büchi automaton and the following solver: CryptoMiniSat as SAT solver [25], RAReQS [18] as QBF solver, QuAbs [26] as certified solver, bloqqer [4] as preprocessor for QBF and iDQ [15] as DQBF solver.

In addition we compare the different encodings with the tool introduced by Guthoff [17]. This tool is based on an SMT encoding to solve the distributed synthesis problem.

LTL - Specifications

For the evaluation we use three types of specifications.

Simple Arbiter

A simple arbiter is defined over a set of requests $\{r_0, r_1, \ldots, r_n\}$ and a set of grants $\{g_0, g_1, \ldots, g_n\}$. In this specification we demand two guarantees and no assumption:

- Every request is answered with his corresponding grant at some point.
- There holds at most one grant at every point in time.

Formally we can describe this specification with the following LTL formula

$$\bigwedge_{0 \leq i \leq n} \square(r_i \rightarrow \Diamond g_i) \wedge \bigwedge_{0 \leq i,j \leq n \wedge i \neq j} \square \neg(g_i \wedge g_j)$$

Load Balancer

The second specification, is a load balancer. This specification was introduced by Ehlers [7]. Similar to the simple arbiter this specification is defined over a set of requests $\{r_0, r_1, \ldots, r_n\}$ and grants $\{g_0, g_1, \ldots, g_n\}$. $idle$ is added as a supplementary input variable.

On a high level this specification distributes incoming jobs, represented by the $idle$ variable to different servers $0, \ldots n$. The variables r_i represent the receiving information if the server i can get a new task and the variables g_i symbols the server task. To describe this specification formally we use a set of assumptions and a set of guarantees.

As assumptions we demand:

- There is always an incoming job.
- If there is an incoming job and in the next step no server is assigned to this job, there is again an incoming job.
- When Server 0 gets a task, we cannot receive the information if server 0 is sufficiently under-utilised to get another task and we cannot receive a job until there is an incoming job and no receiving information about server 0. This does not hold for the first step.

The following LTL formulas describe the assumptions formally:

- $\square \lozenge \, idle$
- $\square \Big(\big(idle \land \bigcirc \bigwedge_{0 \leq i \leq n} (\neg g_i) \big) \to \big(\bigcirc \, idle \big) \Big)$
- $\bigcirc \square \Big(\neg g_0 \lor \big((\neg r_0 \land \neg idle) \, \mathcal{U} \, (\neg r_0 \land idle) \big) \Big)$

As guarantees we demand:

- At every time, only one server gets the task. This only applies in the second (and following) steps.
- If a server gets a task in the next step, then the load balancer gets the information about this server in the current step.
- We can only receive the information about server 0 if every other server got a task.
- If no job is received then in the next step no server gets a task.
- There are again and again positions for each server, such that the information are received and in the next step the server gets the task

The following LTL-formulas describe the guarantees formally:

- $\bigcirc \square \Big(\bigwedge_{0 \leq i, j \leq n \land i \neq j} \neg (g_i \land g_j) \Big)$

- $\bigwedge_{0\leq i\leq n}\left(\Box\left((\bigcirc g_i)\rightarrow(r_i)\right)\right)$
- $\bigwedge_{1\leq i\leq n}\left(\Box\,(r_0\rightarrow g_i)\right)$
- $\Box\left(\neg idle\rightarrow(\bigcirc\bigwedge_{0\leq i\leq n}(\neg g_i))\right)$
- $\bigwedge_{0\leq i\leq n}\left(\Box\Diamond(\neg r_i\vee\bigcirc g_i)\right)$

Request-Answer

To define a priority on the different processes we used the following specification. This specification is similar to the simple arbiter. We added a priority, by demanding that the grant to a request has to happen in the following next steps: 2 to the power of the index of the process producing the grant. Formally: Let id be the index of the process producing the corresponding grant.

$$\bigwedge_{0\leq i\leq n}\Box(r_i\rightarrow g_i\vee\bigvee_{1\leq i<2^{id}}(\bigcirc^i g_i))\wedge\bigwedge_{0\leq i,j\leq n\wedge i\neq j}\Box\neg(g_i\wedge g_j)$$

Architectures

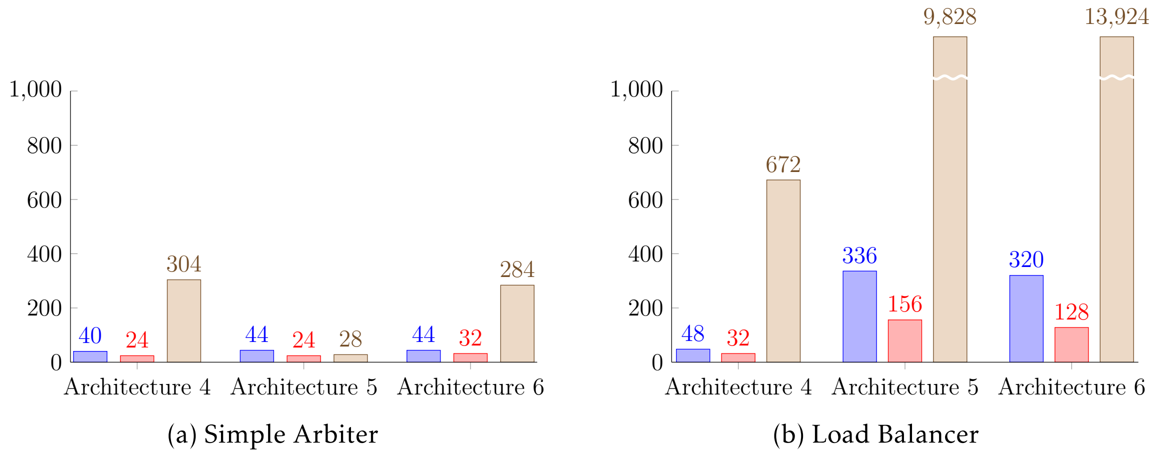
In section 10.1, we represent the architectures for our evaluation. In architectures where inputs are also system outputs, like the pipeline architecture 5.1c, we changed the specification. For all architectures we interpret a variable x, which is an input and an output, also as a new request. For the load balancer specification each process gets idle as input too. For the architecturess 1 - 12, we used the simple arbiter and load balancer specification. For architectures 13 -15, we used the request answer and load balancer specification. In addition we increased the bound in this architecture for process 1 by the factor 1 and process 2 by the factor 2.

Results

The results of our evaluations are represented in the table 7.1. In this section we compare at first only the encodings introduced in this thesis. Therefore we look at architectures where we can build the QBF encoding and at architectures where we can not build the QBF encodings. Afterwards we compare the best result with the SMT-based encodings introduced by Guthoff [17].

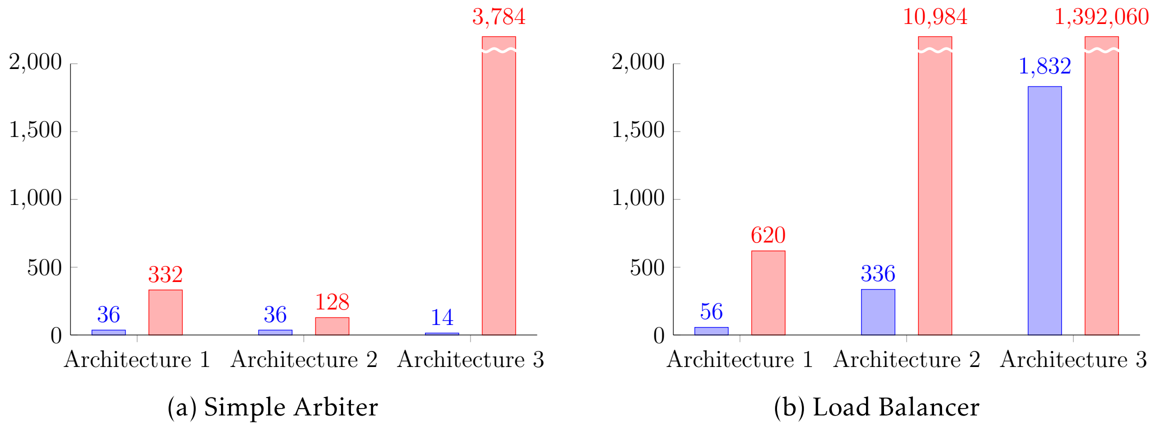
Comparison between SAT, QBF and DQBF

Figure 7.1 represents the results of architectures with a total order and two processes (Architecture 4-6). On both subfigures we can see that the explicit encoding is faster than the input-symbolic encoding with DQBF and the input-symbolic encoding with QBF is faster then the explicit encodings. The results for architectures with 3 processes (Architecture 10-12) are not represented graphically, because of the large difference. Like in the tabular 7.1 shown, the input-symbolic encoding gets in 5 of 6 cases time or memout and the explicit encoding gets in 2 cases a memout. We can also see the same results if the constraint system gets a solution, like in architectures with 2 processes.



(a) Simple Arbiter (b) Load Balancer

Figure 7.1: Results on Architecture where the QBF encoding is applicable. The blue bar represents the results of the explicit encoding, the red bar the results of the input-symbolic encoding with QBF and the brown bar the results of the input-symbolic encoding with DQBF.



(a) Simple Arbiter (b) Load Balancer

Figure 7.2: Test Results on Architecture where the QBF encoding is not applicable. The blue bar represents the results of the explicit encoding and the red bar the results of the input-symbolic encoding with DQBF.

In architectures without a total order we can see the same results like in the previous part. Figure 7.2 shows graphically that the input-symbolic encoding with DQBF is slower than the SAT propositional encoding if we get no time or memout. We can also see the effect that we get for the input encoding in architectures with 3 processes in 2 of 6 cases a memout and in 3 of 6 cases a timeout. For the architectures 13-15, where we increased the bound differently we get the same results.

As a conclusion for this part we can summarize that we get the same results like Faymonville wt al. [8]. For architectures with a total order where we can build the input-symbolic encoding with QBF, this encoding is faster than the explicit encoding. However if there is no total order on the processes and we have to use DQBF for the input-symbolic encoding, the explicit encoding is faster. This result might change in the future, if the DQBF solver get better.

Comparison with SMT

In the next step we want to compare our encodings with the SMT based encoding introduced by Guthoff [17]. We only compare the QBF constraint system with the SMT constraint system, because we could see in the previous paragraph that the constraint system with QBF was the fastest one, if it is applicable.

In architectures with 2 processes, we can see the input-symbolic encoding with QBF needs between 24 ms and 156 ms to find a valid assignment. The encoding based on SMT in comparison needs between 168ms and 372ms. We can see that the input-symbolic constraint system is faster. If we increase the number of processes we see another result. The input-symbolic encoding needs for the simple arbiter specification between 8.624s and 17.788s and for the load balancer between 72ms and 591.156s. The encoding with SMT needs less than 0.5s for the simple arbiter and less than 8s for the load balancer in all architectures.

This big difference especially in the architectures with 2 processes could be explained with the different search strategies. In the encodings introduced in this thesis, the transition systems for each process are bounded. For example in an architecture with 3 processes and bound 2, each transition system is bounded with 3 states and the composition of the transition systems has 8 states. In the constraint system introduced by Guthoff [17] the composition of the transition systems is bounded, in our example with 3 states. Due to this optimized search strategy, the solver has to solve a constraint system with 3 states instead of 8 states. If we would use this search strategy in the encoding introduced in this thesis such that the constraint system gets smaller, the results might change. If we look at the only case where the QBF encoding outperforms the SMT encoding namely the load balancer for architecture 10, we can see this result. In this case the search strategy has no impact, because we found for both encodings a solution with bound 1. This allows a direct comparison of the runtimes leading to the result that QBF is faster than SMT, which underlines the result seen above for architectures with 2 processes.

This shows that due to the different search strategies the results are difficult to compare. If we only look at specifications and architectures with a solution bound of 2 or less we get the comparable results as Faymonville et al. [8] which compared different encodings for synthesizing single processes. The symbolic encoding is faster than the explicit encoding if we use QBF and slower if not. In addition the SAT propositional and QBF-encodings are faster than an SMT based encoding.

architecture + specification	smt	explicit	input-symbolic-qbf	input-symbolic-dqbf
A1_simpleArbiter	0.208	0.036	not possible	0.332
A1_loadBalancer	0.212	0.056	not possible	0.62
A2_simpleArbiter	0.172	0.036	not possible	0.128
A2_loadBalancer	0.328	0.336	not possible	10.984
A3_simpleArbiter	0.168	0.14	not possible	3.784
A3_loadBalancer	0.384	1.832	not possible	1392.06
A4_simpleArbiter	0.168	0.04	0.024	0.304
A4_loadBalancer	0.184	0.048	0.032	0.672
A5_simpleArbiter	0.204	0.044	0.024	0.028
A5_loadBalancer	0.372	0.336	0.156	9.828
A6_simpleArbiter	0.18	0.044	0.032	0.284
A6_loadBalancer	0.328	0.32	0.128	13.924
A7_simpleArbiter	0.212	526.612	not possible	timeout
A7_loadBalancer	0.236	0.136	not possible	memout
A8_simpleArbiter	0.392	531.176	not possible	timeout
A8_loadBalancer	5.064	memout	not possible	memout
A9_simpleArbiter	0.456	5.752	not possible	1893.96
A9_loadBalancer	7.468	155.548	not possible	timeout
A10_simpleArbiter	0.216	0.148	17.788	memout
A10_loadBalancer	0.236	627.42	0.072	9.248
A11_simpleArbiter	0.444	514.556	9.144	memout
A11_loadBalancer	2.204	memout	591.156	timeout
A12_simpleArbiter	0.404	1660.68	8.624	memout
A12_loadBalancer	1.844	memout	503.036	timeout
A13_request + answer	0.396	0.476	not possible	3.208
A13_loadBalancer	0.46	0.32	not possible	25.56
A14_request + answer	0.76	14.932	not possible	4.884
A14_loadBalancer	2.264	0.364	not possible	memout
A15_request + answer	0.828	1.452	not possible	memout
A15_loadBalancer	2.184	timeout	not possible	memout

Table 7.1: Results of the Evaluation in seconds

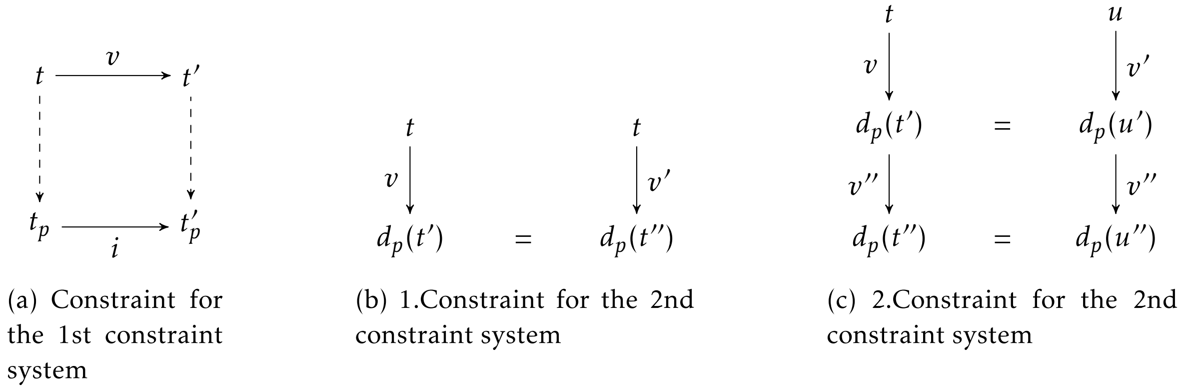
8 Related Work

A. Church introduced the synthesis problem [5] in 1957 as a theoretical problem. But in these days there are tools that solve the original problem as well as more advanced problems such as the distributed synthesis.

In 1990 Pnueli and Rosner [22] introduced the distributed synthesis problem, that transfers the synthesis problem [5] to distributed systems. They proved that this problem is in general undecidable, even for simple architectures like two communicating processes. In took 11 years until Kupfermann and Vardi [20] proved that a pipeline architecture or a ring architecture are decidable and can be synthesized by an automaton based algorithm. Two years later Walukiewicz and Mohalik [21] presented a game based construction to solve the synthesis for these architectures. In 2005 Finkbeiner and Schewe [11] divided the distributed systems in decidable and undecidable. They also introduced an algorithm to solve the synthesis for all decidable distributed architectures. To find an algorithm that can solve undecidable architectures Finkbeiner and Schewe introduced bounded synthesis [12]. Bounded synthesis is a restriction of the synthesis problem that bounds the size of the implementation and constructs an SMT query that is satisfiable iff a realizing implementation of that size exists. If no solution is found, the state space is increased. It was introduced to find small solutions for single processes and can easily be transferred to synthesizing distributed or asynchronous systems.

Bohy et al. presented in 2012 Acacia+ [24] , a tool to solve LTL synthesis. In this tool the LTL formula is translated into a universal K-co-Büchi automaton, where for each word that is accepted by the LTL formula each rejecting state in the automaton is visited at most K times. The parameter K satisfying this condition is determined incrementally. Therefore the LTL formula is at first translated into a universal co-Büchi automaton. The automaton for $K+1$ is constructed with the automaton for K using a variant of the subset construction. With a variant of safety games, the tool checks if the LTL formula is equivalent to the constructed automaton. Acacia+ uses the bounded synthesis approach to solve the synthesis problem like the encodings introduced in this thesis. In contrast to the encodings explained in this thesis Acacia+ bounds the number of visits of the rejecting states in the automaton and not the size of the transition system. In addition Acacia+ solves the single process synthesis problem whereas the encodings in this thesis solve the distributed synthesis.

Guthoff [17] introduced an implementation that is realized in Party Plus. Party Plus is based on Party [19], a to solve LTL synthesis for single processes, with the difference that it supports different SMT solver. It follows the same approach like the encodings introduced in this paper. The LTL formula is translated into a universal co-Büchi automaton by the tool ltl3ba [16]. Afterwards a constraint system is build that can be solved by a SMT solver. The



(a) Constraint for the 1st constraint system

(b) 1.Constraint for the 2nd constraint system

(c) 2.Constraint for the 2nd constraint system

FIGURE 8.1: Representation of the constraints introduced by Guthoff, to solve the challenge that the transition systems for each process have only the information specified in the architecture

two constraint systems introduced by Guthoff [17] solve the distributed synthesis problem. These constraint systems are both based on the bounded synthesis approach and guess a valid annotation function for the run graph. The difference to the encodings in this thesis is the approach to solve the challenge that each transition system should only have the information specified in the architecture. Both constraint systems are build with the transition relation of the composition of the transition systems. The difference between the encodings is the addition of new constraints to solve the challenge. Figure 8.1 gives an overview of the different solutions. In both solutions the constraints to check if the composition of the transition systems holds on the specification are checked are based on an already existing transition system of the composition, whereas this thesis constructs these constraints based on the single processes.

Faymonville et al. [8] introduced four encodings to solve the LTL synthesis for single processes with the bounded synthesis approach. The first encoding is called the explicit encoding. This encoding builds a constraint system with propositional formulas and can be solved by a SAT solver (compare section 4.2). The next encoding is called the input-symbolic encoding. In comparison to the explicit encoding the input is represented with a universal quantifier and can be solved by a QBF solver (compare section 4.3). The last two encodings can be solved by a DQBF solver. The state and input-symbolic encoding represents the states of the transition system and the input set symbolically and the full encoding represents in addition the states of the universal co-Büchi automaton symbolically. These encodings are implemented in BoSy [9] a tool to solve bounded synthesis. BoSy is an experimental frameworks that builds constraint systems in SMT formulas, SAT propositional formulas, QBF and DQBF. It compares the different encodings one below the other and the same constraint system with different solvers. This thesis is based on the encodings introduced by Faymonville et al. [8] and the implementation is based on Bosy [9]. The difference is that this thesis builds encodings to solve the distributed synthesis problem.

Conclusion & Future Work

The starting point of this thesis was the single process synthesis and the bounded synthesis approach for single processes. We looked at two of the four encodings introduced by Faymonville et al. [8] to solve this problem in detail. We describte the complexity of the size of the formula and the number of variables for each of the encodings and proved their functional correctness. In the next step we looked at the synthesis of distributed systems, their challenges and how we can formally describe these systems. To solve the distributed synthesis we constructed an explicit encoding with SAT propositional formulas, an input-symbolic encoding with QBF and an input-symbolic encoding with DQBF. We proved their functional correctness and that each variable has only the information that is specified in the architecture. In addition we described the complexity of the size of the formula and the number of variables for each encoding. In the end we evaluated and compared the different encodings one against the other and with the encoding introduced in [17].

We found out that the input-symbolic encoding is faster than the explicit encoding if we use QBF and slower if we have to use DQBF formulas. This result is similar to the result for encodings of single process synthesis [8]. The symbolic constraint systems are faster if we can use QBF and slower if we use DQBF. These results might change in the future with the development of better DQBF solver. During the evaluation we found a second result. If we test specifications with architectures consisting of 2 processes, an input set with 2 requests and an output set with 2 grants, the explicit as well as the input-symbolic encoding with QBF are faster. If we add a new process with a new request and a new grant to the architectures, the result changes. For these cases, the SMT constraint system is much faster than the encodings introduced in this thesis possibly due to the different search strategies.

For future work we could build a state- and input-symbolic encoding and a full symbolic encoding for the distributed synthesis. If DQBF solver get faster the results might change because the size of these constraint systems is smaller than the size of the explicit or the input-symbolic constraint systems. Assuming that the solvers for the different encodings are comparably fast the difference in run time between the different encodings might arise due to the different complexity of the constraint systems (compare chapter 6).

The reason for the huge difference between the encodings introduced in this thesis and the SMT encoding could be the search strategy, which bounds the composition instead of bounding the transition system for each process. If we use this search strategy in the encodings introduced in this thesis the results might change.

In addition the encoding introduced by Guthoff [17] builds another constraint system which is working on the composition of the transition systems instead of working on each single transition system. If we build the encodings for the distributed synthesis with this approach the results might change too.

Bibliography

[1] Tomás Babiak, Mojmír Kretínský, Vojtech Rehák, and Jan Strejcek. LTL to büchi automata translation: Fast and more deterministic. In Cormac Flanagan and Barbara König, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 18th International Conference, TACAS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, volume 7214 of *Lecture Notes in Computer Science*, pages 95–109. Springer, 2012.

[2] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

[3] Adrian Balint, Daniel Diepold, Daniel Gall, Simon Gerber, Gregor Kapler, and Robert Retz. EDACC - an advanced platform for the experiment design, administration and analysis of empirical algorithms. In Carlos A. Coello Coello, editor, *Learning and Intelligent Optimization - 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers*, volume 6683 of *Lecture Notes in Computer Science*, pages 586–599. Springer, 2011.

[4] Armin Biere, Florian Lonsing, and Martina Seidl. Blocked clause elimination for QBF. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *Automated Deduction - CADE-23 - 23rd International Conference on Automated Deduction, Wroclaw, Poland, July 31 - August 5, 2011. Proceedings*, volume 6803 of *Lecture Notes in Computer Science*, pages 101–115. Springer, 2011.

[5] Alonzo Church. Application of recursive arithmetic to the problem of circuit synthesis. *J. Symb. Log.*, 28(4):289–290, 1963.

[6] Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. Spot 2.0 - A framework for LTL and \omega - automata manipulation. In Cyrille Artho, Axel Legay, and Doron Peled, editors, *Automated Technology for Verification and Analysis - 14th International Symposium, ATVA 2016, Chiba, Japan, October 17-20, 2016, Proceedings*, volume 9938 of *Lecture Notes in Computer Science*, pages 122–129, 2016.

[7] Rüdiger Ehlers. Symbolic bounded synthesis. *Formal Methods in System Design*, 40(2):232–262, 2012.

[8] Peter Faymonville, Bernd Finkbeiner, Markus N. Rabe, and Leander Tentrup. Encodings of bounded synthesis. In Axel Legay and Tiziana Margaria, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference,*

TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I, volume 10205 of *Lecture Notes in Computer Science*, pages 354–370, 2017.

[9] Peter Faymonville, Bernd Finkbeiner, and Leander Tentrup. Bosy: An experimentation framework for bounded synthesis. In *Proceedings of CAV*, volume 10427 of *LNCS*, pages 325–332. Springer, 2017.

[10] Bernd Finkbeiner. Synthesis of reactive systems. In Javier Esparza, Orna Grumberg, and Salomon Sickert, editors, *Dependable Software Systems Engineering*, volume 45 of *NATO Science for Peace and Security Series, D: Information and Communication Security*, pages 72–98. IOS Press, 2016.

[11] Bernd Finkbeiner and Sven Schewe. Uniform distributed synthesis. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005), 26-29 June 2005, Chicago, IL, USA, Proceedings*, pages 321–330. IEEE Computer Society, 2005.

[12] Bernd Finkbeiner and Sven Schewe. Bounded synthesis. *STTT*, 15(5-6):519–539, 2013.

[13] Bernd Finkbeiner and Leander Tentrup. Fast DQBF refutation. In Carsten Sinz and Uwe Egly, editors, *Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8561 of *Lecture Notes in Computer Science*, pages 243–251. Springer, 2014.

[14] Andreas Fröhlich, Gergely Kovásznai, and Armin Biere. A dpll algorithm for solving dqbf. 2012.

[15] Andreas Fröhlich, Gergely Kovásznai, Armin Biere, and Helmut Veith. idq: Instantiation-based DQBF solving. In Daniel Le Berre, editor, *POS-14. Fifth Pragmatics of SAT workshop, a workshop of the SAT 2014 conference, part of FLoC 2014 during the Vienna Summer of Logic, July 13, 2014, Vienna, Austria*, volume 27 of *EPiC Series in Computing*, pages 103–116. EasyChair, 2014.

[16] Paul Gastin and Denis Oddoux. Fast LTL to büchi automata translation. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *Computer Aided Verification, 13th International Conference, CAV 2001, Paris, France, July 18-22, 2001, Proceedings*, volume 2102 of *Lecture Notes in Computer Science*, pages 53–65. Springer, 2001.

[17] Carolyn Guthoff. Bounded synthesis of distributed architectures. Bachelor thesis, Universität des Saarlandes, 2015.

[18] Mikolás Janota, William Klieber, Joao Marques-Silva, and Edmund M. Clarke. Solving QBF with counterexample guided refinement. *Artif. Intell.*, 234:1–25, 2016.

[19] Ayrat Khalimov, Swen Jacobs, and Roderick Bloem. PARTY parameterized synthesis of token rings. In Sharygina and Veith [24], pages 928–933.

[20] Orna Kupferman and Moshe Y. Vardi. Synthesizing distributed systems. In *16th Annual IEEE Symposium on Logic in Computer Science, Boston, Massachusetts, USA, June 16-19, 2001, Proceedings*, pages 389–398. IEEE Computer Society, 2001.

[21] Swarup Mohalik and Igor Walukiewicz. Distributed games. In Paritosh K. Pandya and Jaikumar Radhakrishnan, editors, *FST TCS 2003: Foundations of Software Technology and Theoretical Computer Science, 23rd Conference, Mumbai, India, December 15-17, 2003, Proceedings*, volume 2914 of *Lecture Notes in Computer Science*, pages 338–351. Springer, 2003.

[22] Amir Pnueli and Roni Rosner. Distributed reactive systems are hard to synthesize. In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume II*, pages 746–757. IEEE Computer Society, 1990.

[23] Markus N. Rabe and Leander Tentrup. CAQE: A certifying QBF solver. In Roope Kaivola and Thomas Wahl, editors, *Formal Methods in Computer-Aided Design, FMCAD 2015, Austin, Texas, USA, September 27-30, 2015.*, pages 136–143. IEEE, 2015.

[24] Natasha Sharygina and Helmut Veith, editors. *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*. Springer, 2013.

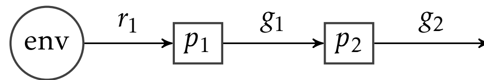
[25] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, volume 5584 of *Lecture Notes in Computer Science*, pages 244–257. Springer, 2009.

[26] Leander Tentrup. Solving QBF by abstraction. *CoRR*, abs/1604.06752, 2016.

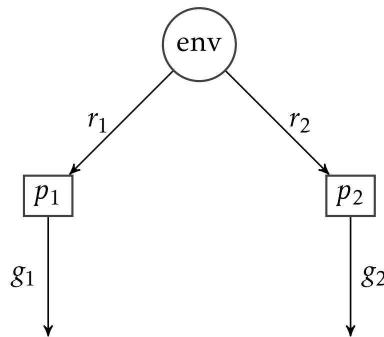
10 Appendix

10.1 Architectures

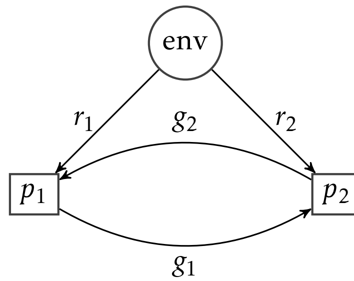
Architecture 1



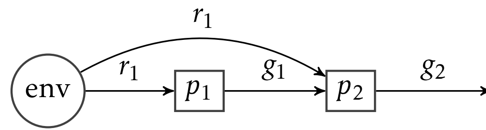
Architecture 2



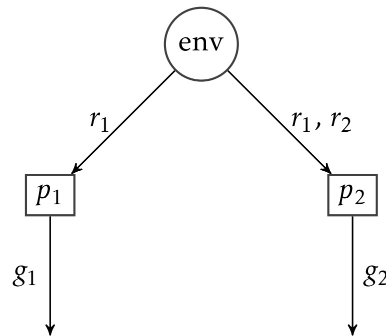
Architecture 3



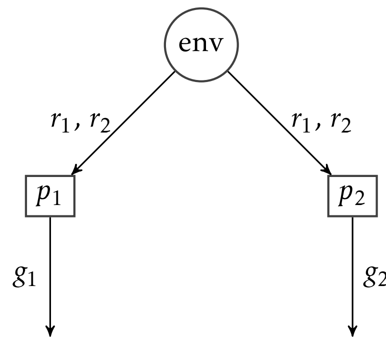
Architecture 4



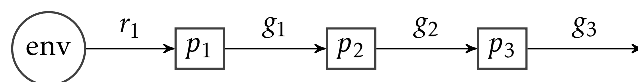
Architecture 5



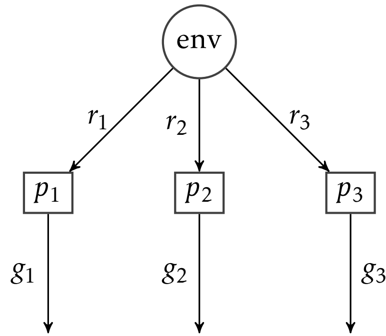
Architecture 6



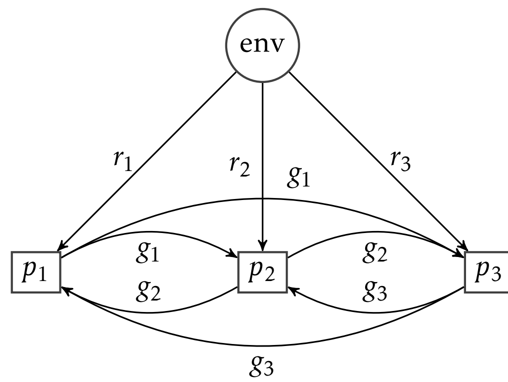
Architecture 7



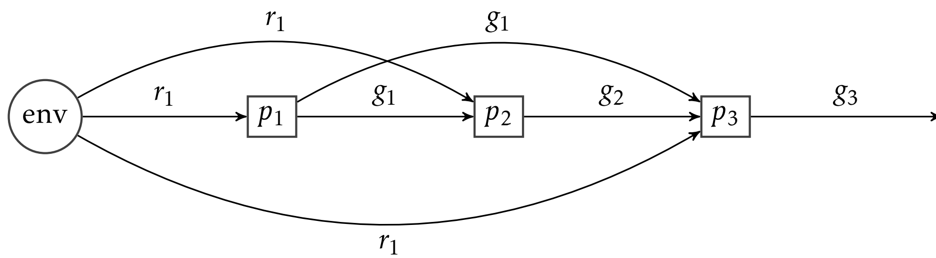
Architecture 8



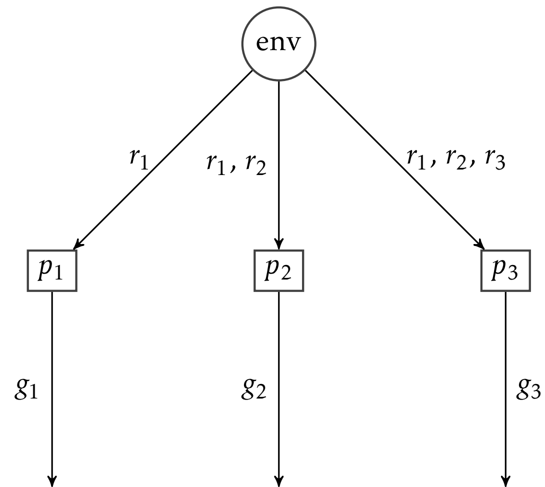
Architecture 9



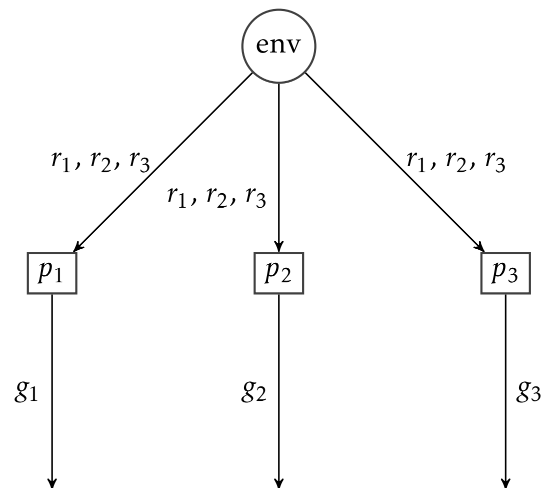
Architecture 10



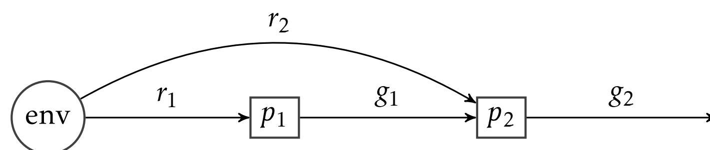
Architecture 11



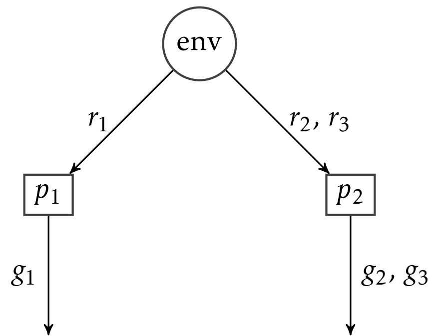
Architecture 12



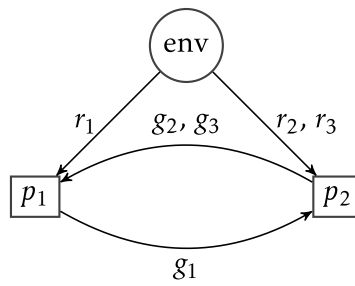
Architecture 13



Architecture 14



Architecture 15



10.2 Example for the SAT Propositional Constraint System

$\lambda^{\mathbb{B}}_{t_0,q_0}$ \\initial state

$\wedge\ (\tau_{t_0,(\bar{r}_1,\bar{r}_2),t_0} \vee \tau_{t_0,(\bar{r}_1,\bar{r}_2),t_1}) \wedge (\tau_{t_0,(\bar{r}_1,r_2),t_0} \vee \tau_{t_0,(\bar{r}_1,r_2),t_1}) \wedge (\tau_{t_0,(r_1,\bar{r}_2),t_0} \vee \tau_{t_0,(r_1,\bar{r}_2),t_1}) \wedge (\tau_{t_0,(r_1,r_2),t_0} \vee \tau_{t_0,(r_1,r_2),t_1})$ \\ $t = t_0$

$\wedge\ (\tau_{t_1,(\bar{r}_1,\bar{r}_2),t_0} \vee \tau_{t_1,(\bar{r}_1,\bar{r}_2),t_1}) \wedge (\tau_{t_1,(\bar{r}_1,r_2),t_0} \vee \tau_{t_1,(\bar{r}_1,r_2),t_1}) \wedge (\tau_{t_1,(r_1,\bar{r}_2),t_0} \vee \tau_{t_1,(r_1,\bar{r}_2),t_1}) \wedge (\tau_{t_1,(r_1,r_2),t_0} \vee \tau_{t_1,(r_1,r_2),t_1})$ \\ $t = t_1$

$\wedge\ (\lambda^{\mathbb{B}}_{t_0,q_0}$ \\ $q = q_0$, $t = t_0$

$\rightarrow (((\delta_{t_0,q_0,(\bar{r}_1,\bar{r}_2),q_0} \rightarrow ((\tau_{t_0,(\bar{r}_1,\bar{r}_2),t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_0} \wedge \lambda^{\#}_{t_0,q_0} \triangleright_{q_0} \lambda^{\#}_{t_0,q_0}) \wedge (\tau_{t_0,(\bar{r}_1,\bar{r}_2),t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_0} \wedge \lambda^{\#}_{t_1,q_0} \triangleright_{q_0} \lambda^{\#}_{t_0,q_0})))$ \\ $q' = q_0$, $i = (\bar{r}_1, \bar{r}_2)$

$\wedge\ (\delta_{t_0,q_0,(\bar{r}_1,r_2),q_0} \rightarrow ((\tau_{t_0,(\bar{r}_1,r_2),t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_0} \wedge \lambda^{\#}_{t_0,q_0} \triangleright_{q_0} \lambda^{\#}_{t_0,q_0}) \wedge (\tau_{t_0,(\bar{r}_1,r_2),t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_0} \wedge \lambda^{\#}_{t_1,q_0} \triangleright_{q_0} \lambda^{\#}_{t_0,q_0})))$ \\ $i = (\bar{r}_1, r_2)$

$\wedge\ (\delta_{t_0,q_0,(r_1,\bar{r}_2),q_0} \rightarrow ((\tau_{t_0,(r_1,\bar{r}_2),t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_0} \wedge \lambda^{\#}_{t_0,q_0} \triangleright_{q_0} \lambda^{\#}_{t_0,q_0}) \wedge (\tau_{t_0,(r_1,\bar{r}_2),t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_0} \wedge \lambda^{\#}_{t_1,q_0} \triangleright_{q_0} \lambda^{\#}_{t_0,q_0})))$ \\ $i = (r_1, \bar{r}_2)$

$\wedge\ (\delta_{t_0,q_0,(r_1,r_2),q_0} \rightarrow ((\tau_{t_0,(r_1,r_2),t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_0} \wedge \lambda^{\#}_{t_0,q_0} \triangleright_{q_0} \lambda^{\#}_{t_0,q_0}) \wedge (\tau_{t_0,(r_1,r_2),t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_0} \wedge \lambda^{\#}_{t_1,q_0} \triangleright_{q_1} \lambda^{\#}_{t_0,q_0}))))$ \\ $i = (r_1, r_2)$

$\wedge\ ((\delta_{t_0,q_0,(\bar{r}_1,\bar{r}_2),q_1} \rightarrow ((\tau_{t_0,(\bar{r}_1,\bar{r}_2),t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_1} \wedge \lambda^{\#}_{t_0,q_1} \triangleright_{q_1} \lambda^{\#}_{t_0,q_0}) \wedge (\tau_{t_0,(\bar{r}_1,\bar{r}_2),t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_1} \wedge \lambda^{\#}_{t_1,q_1} \triangleright_{q_1} \lambda^{\#}_{t_0,q_0})))$ \\ $q' = q_1$, $i = (\bar{r}_1, \bar{r}_2)$

$\wedge\ (\delta_{t_0,q_0,(\bar{r}_1,r_2),q_1} \rightarrow ((\tau_{t_0,(\bar{r}_1,r_2),t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_1} \wedge \lambda^{\#}_{t_0,q_1} \triangleright_{q_1} \lambda^{\#}_{t_0,q_0}) \wedge (\tau_{t_0,(\bar{r}_1,r_2),t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_1} \wedge \lambda^{\#}_{t_1,q_1} \triangleright_{q_1} \lambda^{\#}_{t_0,q_0})))$ \\ $i = (\bar{r}_1, r_2)$

$\wedge\ (\delta_{t_0,q_0,(r_1,\bar{r}_2),q_1} \rightarrow ((\tau_{t_0,(r_1,\bar{r}_2),t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_1} \wedge \lambda^{\#}_{t_0,q_1} \triangleright_{q_1} \lambda^{\#}_{t_0,q_0}) \wedge (\tau_{t_0,(r_1,\bar{r}_2),t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_1} \wedge \lambda^{\#}_{t_1,q_1} \triangleright_{q_1} \lambda^{\#}_{t_0,q_0})))$ \\ $i = (r_1, \bar{r}_2)$

$\wedge\ (\delta_{t_0,q_0,(r_1,r_2),q_1} \rightarrow ((\tau_{t_0,(r_1,r_2),t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_1} \wedge \lambda^{\#}_{t_0,q_1} \triangleright_{q_1} \lambda^{\#}_{t_0,q_0}) \wedge (\tau_{t_0,(r_1,r_2),t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_1} \wedge \lambda^{\#}_{t_1,q_1} \triangleright_{q_1} \lambda^{\#}_{t_0,q_0}))))$ \\ $i = (r_1, r_2)$

$\wedge\ (((\delta_{t_0,q_0,(\bar{r}_1,\bar{r}_2),q_e} \rightarrow ((\tau_{t_0,(\bar{r}_1,\bar{r}_2),t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_e} \wedge \lambda^{\#}_{t_0,q_e} \triangleright_{q_e} \lambda^{\#}_{t_0,q_0}) \wedge (\tau_{t_0,(\bar{r}_1,\bar{r}_2),t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_e} \wedge \lambda^{\#}_{t_1,q_e} \triangleright_{q_e} \lambda^{\#}_{t_0,q_0})))$ \\ $q' = q_e$, $i = (\bar{r}_1, \bar{r}_2)$

$\wedge\ (\delta_{t_0,q_0,(\bar{r}_1,r_2),q_e} \rightarrow ((\tau_{t_0,(\bar{r}_1,r_2),t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_e} \wedge \lambda^{\#}_{t_0,q_e} \triangleright_{q_e} \lambda^{\#}_{t_0,q_0}) \wedge (\tau_{t_0,(\bar{r}_1,r_2),t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_e} \wedge \lambda^{\#}_{t_1,q_e} \triangleright_{q_e} \lambda^{\#}_{t_0,q_0})))$ \\ $i = (\bar{r}_1, r_2)$

$\wedge\ (\delta_{t_0,q_0,(r_1,\bar{r}_2),q_e} \rightarrow ((\tau_{t_0,(r_1,\bar{r}_2),t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_e} \wedge \lambda^{\#}_{t_0,q_e} \triangleright_{q_e} \lambda^{\#}_{t_0,q_0}) \wedge (\tau_{t_0,(r_1,\bar{r}_2),t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_e} \wedge \lambda^{\#}_{t_1,q_e} \triangleright_{q_e} \lambda^{\#}_{t_0,q_0})))$ \\ $i = (r_1, \bar{r}_2)$

$\wedge\ (\delta_{t_0,q_0,(r_1,r_2),q_e} \rightarrow ((\tau_{t_0,(r_1,r_2),t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_e} \wedge \lambda^{\#}_{t_0,q_e} \triangleright_{q_e} \lambda^{\#}_{t_0,q_0}) \wedge (\tau_{t_0,(r_1,r_2),t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_e} \wedge \lambda^{\#}_{t_1,q_e} \triangleright_{q_e} \lambda^{\#}_{t_0,q_0}))))$ \\ $i = (r_1, r_2)$

$\wedge\ (((\delta_{t_0,q_0,(\bar{r}_1,\bar{r}_2),q_2} \rightarrow ((\tau_{t_0,(\bar{r}_1,\bar{r}_2),t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_2} \wedge \lambda^{\#}_{t_0,q_2} \triangleright_{q_2} \lambda^{\#}_{t_0,q_0}) \wedge (\tau_{t_0,(\bar{r}_1,\bar{r}_2),t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_2} \wedge \lambda^{\#}_{t_1,q_2} \triangleright_{q_2} \lambda^{\#}_{t_0,q_0})))$ \\ $q' = q_2$, $i = (\bar{r}_1, \bar{r}_2)$

$\wedge\ (\delta_{t_0,q_0,(\bar{r}_1,r_2),q_2} \rightarrow ((\tau_{t_0,(\bar{r}_1,r_2),t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_2} \wedge \lambda^{\#}_{t_0,q_2} \triangleright_{q_2} \lambda^{\#}_{t_0,q_0}) \wedge (\tau_{t_0,(\bar{r}_1,r_2),t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_2} \wedge \lambda^{\#}_{t_1,q_2} \triangleright_{q_2} \lambda^{\#}_{t_0,q_0})))$ \\ $i = (\bar{r}_1, r_2)$

$\wedge\ (\delta_{t_0,q_0,(r_1,\bar{r}_2),q_2} \rightarrow ((\tau_{t_0,(r_1,\bar{r}_2),t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_2} \wedge \lambda^{\#}_{t_0,q_2} \triangleright_{q_2} \lambda^{\#}_{t_0,q_0}) \wedge (\tau_{t_0,(r_1,\bar{r}_2),t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_2} \wedge \lambda^{\#}_{t_1,q_2} \triangleright_{q_2} \lambda^{\#}_{t_0,q_0})))$ \\ $i = (r_1, \bar{r}_2)$

$\wedge\ (\delta_{t_0,q_0,(r_1,r_2),q_2} \rightarrow ((\tau_{t_0,(r_1,r_2),t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_2} \wedge \lambda^{\#}_{t_0,q_2} \triangleright_{q_2} \lambda^{\#}_{t_0,q_0}) \wedge (\tau_{t_0,(r_1,r_2),t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_2} \wedge \lambda^{\#}_{t_1,q_2} \triangleright_{q_2} \lambda^{\#}_{t_0,q_0})))))$ \\ $i = (r_1, r_2)$

$\wedge\ (\lambda^{\mathbb{B}}_{t_1,q_0}$ \\ $q = q_0$, $t = t_1$

$\rightarrow (((\delta_{t_1,q_0,(\bar{r}_1,\bar{r}_2),q_0} \rightarrow ((\tau_{t_1,(\bar{r}_1,\bar{r}_2),t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_0} \wedge \lambda^{\#}_{t_0,q_0} \triangleright_{q_0} \lambda^{\#}_{t_1,q_0}) \wedge (\tau_{t_1,(\bar{r}_1,\bar{r}_2),t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_0} \wedge \lambda^{\#}_{t_1,q_0} \triangleright_{q_0} \lambda^{\#}_{t_1,q_0})))$ \\ $q' = q_0$, $i = (\bar{r}_1, \bar{r}_2)$

$\wedge\ (\delta_{t_1,q_0,(\bar{r}_1,r_2),q_0} \rightarrow ((\tau_{t_1,(\bar{r}_1,r_2),t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_0} \wedge \lambda^{\#}_{t_0,q_0} \triangleright_{q_0} \lambda^{\#}_{t_1,q_0}) \wedge (\tau_{t_1,(\bar{r}_1,r_2),t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_0} \wedge \lambda^{\#}_{t_1,q_0} \triangleright_{q_0} \lambda^{\#}_{t_1,q_0})))$ \\ $i = (\bar{r}_1, r_2)$

$\wedge\ (\delta_{t_1,q_0,(r_1,\bar{r}_2),q_0} \rightarrow ((\tau_{t_1,(r_1,\bar{r}_2),t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_0} \wedge \lambda^{\#}_{t_0,q_0} \triangleright_{q_0} \lambda^{\#}_{t_1,q_0}) \wedge (\tau_{t_1,(r_1,\bar{r}_2),t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_0} \wedge \lambda^{\#}_{t_1,q_0} \triangleright_{q_0} \lambda^{\#}_{t_1,q_0})))$ \\ $i = (r_1, \bar{r}_2)$

$\wedge\ (\delta_{t_1,q_0,(r_1,r_2),q_0} \rightarrow ((\tau_{t_1,(r_1,r_2),t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_0} \wedge \lambda^{\#}_{t_0,q_0} \triangleright_{q_0} \lambda^{\#}_{t_1,q_0}) \wedge (\tau_{t_1,(r_1,r_2),t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_0} \wedge \lambda^{\#}_{t_1,q_0} \triangleright_{q_1} \lambda^{\#}_{t_1,q_0}))))$ \\ $i = (r_1, r_2)$

$\wedge\ (((\delta_{t_1,q_0,(\bar{r}_1,\bar{r}_2),q_1} \rightarrow ((\tau_{t_1,(\bar{r}_1,\bar{r}_2),t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_1} \wedge \lambda^{\#}_{t_0,q_1} \triangleright_{q_1} \lambda^{\#}_{t_1,q_0}) \wedge (\tau_{t_1,(\bar{r}_1,\bar{r}_2),t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_1} \wedge \lambda^{\#}_{t_1,q_1} \triangleright_{q_1} \lambda^{\#}_{t_1,q_0})))$ \\ $q' = q_1$, $i = (\bar{r}_1, \bar{r}_2)$

$\wedge\ (\delta_{t_1,q_0,(\bar{r}_1,r_2),q_1} \rightarrow ((\tau_{t_1,(\bar{r}_1,r_2),t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_1} \wedge \lambda^{\#}_{t_0,q_1} \triangleright_{q_1} \lambda^{\#}_{t_1,q_0}) \wedge (\tau_{t_1,(\bar{r}_1,r_2),t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_1} \wedge \lambda^{\#}_{t_1,q_1} \triangleright_{q_1} \lambda^{\#}_{t_1,q_0})))$ \\ $i = (\bar{r}_1, r_2)$

$\wedge\ (\delta_{t_1,q_0,(r_1,\bar{r}_2),q_1} \rightarrow ((\tau_{t_1,(r_1,\bar{r}_2),t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_1} \wedge \lambda^{\#}_{t_0,q_1} \triangleright_{q_1} \lambda^{\#}_{t_1,q_0}) \wedge (\tau_{t_1,(r_1,\bar{r}_2),t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_1} \wedge \lambda^{\#}_{t_1,q_1} \triangleright_{q_1} \lambda^{\#}_{t_1,q_0})))$ \\ $i = (r_1, \bar{r}_2)$

$\wedge\ (\delta_{t_1,q_0,(r_1,r_2),q_1} \rightarrow ((\tau_{t_1,(r_1,r_2),t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_1} \wedge \lambda^{\#}_{t_0,q_1} \triangleright_{q_1} \lambda^{\#}_{t_1,q_0}) \wedge (\tau_{t_1,(r_1,r_2),t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_1} \wedge \lambda^{\#}_{t_1,q_1} \triangleright_{q_1} \lambda^{\#}_{t_1,q_0}))))$ \\ $i = (r_1, r_2)$

$\wedge\ (((\delta_{t_1,q_0,(\bar{r}_1,\bar{r}_2),q_e} \rightarrow ((\tau_{t_1,(\bar{r}_1,\bar{r}_2),t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_e} \wedge \lambda^{\#}_{t_0,q_e} \triangleright_{q_e} \lambda^{\#}_{t_1,q_0}) \wedge (\tau_{t_1,(\bar{r}_1,\bar{r}_2),t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_e} \wedge \lambda^{\#}_{t_1,q_e} \triangleright_{q_e} \lambda^{\#}_{t_1,q_0})))$ \\ $q' = q_e$, $i = (\bar{r}_1, \bar{r}_2)$

$\wedge\ (\delta_{t_1,q_0,(\bar{r}_1,r_2),q_e} \rightarrow ((\tau_{t_1,(\bar{r}_1,r_2),t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_e} \wedge \lambda^{\#}_{t_0,q_e} \triangleright_{q_e} \lambda^{\#}_{t_1,q_0}) \wedge (\tau_{t_1,(\bar{r}_1,r_2),t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_e} \wedge \lambda^{\#}_{t_1,q_e} \triangleright_{q_e} \lambda^{\#}_{t_1,q_0})))$ \\ $i = (\bar{r}_1, r_2)$

$\wedge\ (\delta_{t_1,q_0,(r_1,\bar{r}_2),q_e} \rightarrow ((\tau_{t_1,(r_1,\bar{r}_2),t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_e} \wedge \lambda^{\#}_{t_0,q_e} \triangleright_{q_e} \lambda^{\#}_{t_1,q_0}) \wedge (\tau_{t_1,(r_1,\bar{r}_2),t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_e} \wedge \lambda^{\#}_{t_1,q_e} \triangleright_{q_e} \lambda^{\#}_{t_1,q_0})))$ \\ $i = (r_1, \bar{r}_2)$

$\wedge\ (\delta_{t_1,q_0,(r_1,r_2),q_e} \rightarrow ((\tau_{t_1,(r_1,r_2),t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_e} \wedge \lambda^{\#}_{t_0,q_e} \triangleright_{q_e} \lambda^{\#}_{t_1,q_0}) \wedge (\tau_{t_1,(r_1,r_2),t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_e} \wedge \lambda^{\#}_{t_1,q_e} \triangleright_{q_e} \lambda^{\#}_{t_1,q_0}))))$ \\ $i = (r_1, r_2)$

$\wedge\ (((\delta_{t_1,q_0,(\bar{r}_1,\bar{r}_2),q_2} \rightarrow ((\tau_{t_1,(\bar{r}_1,\bar{r}_2),t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_2} \wedge \lambda^{\#}_{t_0,q_2} \triangleright_{q_2} \lambda^{\#}_{t_1,q_0}) \wedge (\tau_{t_1,(\bar{r}_1,\bar{r}_2),t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_2} \wedge \lambda^{\#}_{t_1,q_2} \triangleright_{q_2} \lambda^{\#}_{t_1,q_0})))$ \\ $q' = q_2$, $i = (\bar{r}_1, \bar{r}_2)$

$\wedge\ (\delta_{t_1,q_0,(\bar{r}_1,r_2),q_2} \rightarrow ((\tau_{t_1,(\bar{r}_1,r_2),t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_2} \wedge \lambda^{\#}_{t_0,q_2} \triangleright_{q_2} \lambda^{\#}_{t_1,q_0}) \wedge (\tau_{t_1,(\bar{r}_1,r_2),t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_2} \wedge \lambda^{\#}_{t_1,q_2} \triangleright_{q_2} \lambda^{\#}_{t_1,q_0})))$ \\ $i = (\bar{r}_1, r_2)$

$\wedge\ (\delta_{t_1,q_0,(r_1,\bar{r}_2),q_2} \rightarrow ((\tau_{t_1,(r_1,\bar{r}_2),t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_2} \wedge \lambda^{\#}_{t_0,q_2} \triangleright_{q_2} \lambda^{\#}_{t_1,q_0}) \wedge (\tau_{t_1,(r_1,\bar{r}_2),t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_2} \wedge \lambda^{\#}_{t_1,q_2} \triangleright_{q_2} \lambda^{\#}_{t_1,q_0})))$ \\ $i = (r_1, \bar{r}_2)$

$\wedge\ (\delta_{t_1,q_0,(r_1,r_2),q_2} \rightarrow ((\tau_{t_1,(r_1,r_2),t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_2} \wedge \lambda^{\#}_{t_0,q_2} \triangleright_{q_2} \lambda^{\#}_{t_1,q_0}) \wedge (\tau_{t_1,(r_1,r_2),t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_2} \wedge \lambda^{\#}_{t_1,q_2} \triangleright_{q_2} \lambda^{\#}_{t_1,q_0})))))$ \\ $i = (r_1, r_2)$

$\wedge\ (\lambda^{\mathcal{B}}_{t_0,q_1} \rightarrow \ldots) \wedge (\lambda^{\mathcal{B}}_{t_1,q_1} \rightarrow \ldots) \wedge (\lambda^{\mathcal{B}}_{t_0,q_e} \rightarrow \ldots) \wedge (\lambda^{\mathcal{B}}_{t_1,q_e} \rightarrow \ldots) \wedge (\lambda^{\mathcal{B}}_{t_0,q_2} \rightarrow \ldots) \wedge (\lambda^{\mathcal{B}}_{t_1,q_2} \rightarrow \ldots)$

10.3 Example for the QBF Constraint System

$$\exists \lambda^{\mathbb{B}}_{t_0,q_0}, \lambda^{\mathbb{B}}_{t_1,q_0}, \lambda^{\mathbb{B}}_{t_0,q_1}, \lambda^{\mathbb{B}}_{t_1,q_1}, \lambda^{\mathbb{B}}_{t_0,q_2}, \lambda^{\mathbb{B}}_{t_1,q_2}, \lambda^{\mathbb{B}}_{t_0,q_e}, \lambda^{\mathbb{B}}_{t_1,q_e}, \lambda^{\#}_{t_0,q_0}, \lambda^{\#}_{t_1,q_0}, \lambda^{\#}_{t_0,q_1}, \lambda^{\#}_{t_1,q_1}, \lambda^{\#}_{t_0,q_2}, \lambda^{\#}_{t_1,q_2}, \lambda^{\#}_{t_0,q_e}, \lambda^{\#}_{t_1,q_e} \quad \backslash\backslash\ \lambda\text{-annotation}$$

$$\forall r_1, r_2 \quad \backslash\backslash\ \text{inputs}$$

$$\exists \tau_{t_0,t_0}, \tau_{t_0,t_1}, \tau_{t_1,t_0}, \tau_{t_1,t_1} \quad \backslash\backslash\ \text{transitions}$$

$$\exists o_{t_0}, o_{t_1} \quad \backslash\backslash\ \text{outputs}$$

$$\lambda^{\mathbb{B}}_{t_0,q_0} \quad \backslash\backslash\text{initial state}$$

$$\wedge\ (\tau_{t_0,t_0} \vee \tau_{t_0,t_1}) \wedge (\tau_{t_1,t_0} \vee \tau_{t_1,t_1})$$

$$\wedge\ (\lambda^{\mathbb{B}}_{t_0,q_0} \quad \backslash\backslash\ q = q_0,\ t = t_0$$

$$\rightarrow (((\delta_{t_0,q_0,q_0} \rightarrow ((\tau_{t_0,t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_0} \wedge \lambda^{\#}_{t_0,q_0} \rhd_{q_0} \lambda^{\#}_{t_0,q_0}) \wedge (\tau_{t_0,t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_0} \wedge \lambda^{\#}_{t_1,q_0} \rhd_{q_0} \lambda^{\#}_{t_0,q_0}))) \quad \backslash\backslash\ q' = q_0$$

$$\wedge\ (\delta_{t_0,q_0,q_1} \rightarrow ((\tau_{t_0,t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_1} \wedge \lambda^{\#}_{t_0,q_1} \rhd_{q_1} \lambda^{\#}_{t_0,q_0}) \wedge (\tau_{t_0,t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_1} \wedge \lambda^{\#}_{t_1,q_1} \rhd_{q_1} \lambda^{\#}_{t_0,q_0}))) \quad \backslash\backslash\ q' = q_1$$

$$\wedge\ (\delta_{t_0,q_0,q_e} \rightarrow ((\tau_{t_0,t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_e} \wedge \lambda^{\#}_{t_0,q_e} \rhd_{q_e} \lambda^{\#}_{t_0,q_0}) \wedge (\tau_{t_0,t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_e} \wedge \lambda^{\#}_{t_1,q_e} \rhd_{q_e} \lambda^{\#}_{t_0,q_0}))) \quad \backslash\backslash\ q' = q_e$$

$$\wedge\ (\delta_{t_0,q_0,q_2} \rightarrow ((\tau_{t_0,t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_2} \wedge \lambda^{\#}_{t_0,q_2} \rhd_{q_2} \lambda^{\#}_{t_0,q_0}) \wedge (\tau_{t_0,t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_2} \wedge \lambda^{\#}_{t_1,q_2} \rhd_{q_2} \lambda^{\#}_{t_0,q_0}))))) \quad \backslash\backslash\ q' = q_2$$

$$\wedge\ (\lambda^{\mathbb{B}}_{t_1,q_0} \quad \backslash\backslash\ q = q_0,\ t = t_1$$

$$\rightarrow (((\delta_{t_1,q_0,q_0} \rightarrow ((\tau_{t_1,t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_0} \wedge \lambda^{\#}_{t_0,q_0} \rhd_{q_0} \lambda^{\#}_{t_1,q_0}) \wedge (\tau_{t_1,t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_0} \wedge \lambda^{\#}_{t_1,q_0} \rhd_{q_0} \lambda^{\#}_{t_1,q_0}))) \quad \backslash\backslash\ q' = q_0$$

$$\wedge\ (\delta_{t_1,q_0,q_1} \rightarrow ((\tau_{t_1,t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_1} \wedge \lambda^{\#}_{t_0,q_1} \rhd_{q_1} \lambda^{\#}_{t_1,q_0}) \wedge (\tau_{t_1,t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_1} \wedge \lambda^{\#}_{t_1,q_1} \rhd_{q_1} \lambda^{\#}_{t_1,q_0}))) \quad \backslash\backslash\ q' = q_1$$

$$\wedge\ (\delta_{t_1,q_0,q_e} \rightarrow ((\tau_{t_1,t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_e} \wedge \lambda^{\#}_{t_0,q_e} \rhd_{q_e} \lambda^{\#}_{t_1,q_0}) \wedge (\tau_{t_1,t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_e} \wedge \lambda^{\#}_{t_1,q_e} \rhd_{q_e} \lambda^{\#}_{t_1,q_0}))) \quad \backslash\backslash\ q' = q_e$$

$$\wedge\ (\delta_{t_1,q_0,q_2} \rightarrow ((\tau_{t_1,t_0} \rightarrow \lambda^{\mathbb{B}}_{t_0,q_2} \wedge \lambda^{\#}_{t_0,q_2} \rhd_{q_2} \lambda^{\#}_{t_1,q_0}) \wedge (\tau_{t_1,t_1} \rightarrow \lambda^{\mathbb{B}}_{t_1,q_2} \wedge \lambda^{\#}_{t_1,q_2} \rhd_{q_2} \lambda^{\#}_{t_1,q_0}))))) \quad \backslash\backslash\ q' = q_2$$

$$\wedge\ (\lambda^{\mathcal{B}}_{t_0,q_1} \rightarrow \ldots) \wedge (\lambda^{\mathcal{B}}_{t_1,q_1} \rightarrow \ldots) \wedge (\lambda^{\mathcal{B}}_{t_0,q_e} \rightarrow \ldots) \wedge (\lambda^{\mathcal{B}}_{t_1,q_e} \rightarrow \ldots) \wedge (\lambda^{\mathcal{B}}_{t_0,q_2} \rightarrow \ldots) \wedge (\lambda^{\mathcal{B}}_{t_1,q_2} \rightarrow \ldots)$$

10.4 Example for the DQBF Constraint System

$$
\begin{aligned}
&\forall r_1, r_2 && \backslash\backslash \text{ inputs}\\
&\exists \tau_{t_0,t_0}, \tau_{t_0,t_1}, \tau_{t_1,t_0}, \tau_{t_1,t_1} && \backslash\backslash \text{ transitions}\\
&\exists o_{t_0}, o_{t_1} && \backslash\backslash \text{ outputs}\\
&\lambda^{\mathbb{B}}_{t_0,q_0} && \backslash\backslash \text{initial state}\\
&\wedge (\tau_{t_0,t_0} \vee \tau_{t_0,t_1}) \wedge (\tau_{t_1,t_0} \vee \tau_{t_1,t_1}) &&\\
&\wedge (\lambda^{\mathbb{B}}_{t_0,q_0} && \backslash\backslash\ q = q_0, t = t_0\\
&\rightarrow ((\delta_{t_0,q_0,q_e} \rightarrow (\neg\tau_{t_0,t_0} \wedge \neg\tau_{t_0,t_1})))) && \backslash\backslash\ q' = q_e\\
&\wedge (\lambda^{\mathbb{B}}_{t_1,q_0} && \backslash\backslash\ q = q_0, t = t_1\\
&\rightarrow ((\delta_{t_1,q_0,q_e} \rightarrow (\neg\tau_{t_1,t_0} \wedge \neg\tau_{t_1,t_1})))) && \backslash\backslash\ q' = q_e\\
&\wedge (\lambda^{\mathbb{B}}_{t_0,q_1} && \backslash\backslash\ q = q_1, t = t_0\\
&\rightarrow ((\delta_{t_0,q_1,q_0} \rightarrow (\neg\tau_{t_0,t_0} \wedge \neg\tau_{t_0,t_1})) && \backslash\backslash\ q' = q_0\\
&\wedge ((\delta_{t_0,q_1,q_1} \rightarrow (\neg\tau_{t_0,t_0} \wedge \neg\tau_{t_0,t_1})) && \backslash\backslash\ q' = q_1\\
&\wedge ((\delta_{t_0,q_1,q_e} \rightarrow (\neg\tau_{t_0,t_0} \wedge \neg\tau_{t_0,t_1})) && \backslash\backslash\ q' = q_e\\
&\wedge ((\delta_{t_0,q_1,q_2} \rightarrow (\neg\tau_{t_0,t_0} \wedge \neg\tau_{t_0,t_1}))) && \backslash\backslash\ q' = q_2\\
&\wedge (\lambda^{\mathbb{B}}_{t_1,q_1} && \backslash\backslash\ q = q_1, t = t_1\\
&\rightarrow ((\delta_{t_1,q_1,q_0} \rightarrow (\neg\tau_{t_1,t_0} \wedge \neg\tau_{t_1,t_1})) && \backslash\backslash\ q' = q_0\\
&\wedge ((\delta_{t_1,q_1,q_1} \rightarrow (\neg\tau_{t_1,t_1})) && \backslash\backslash\ q' = q_1\\
&\wedge ((\delta_{t_1,q_1,q_e} \rightarrow (\neg\tau_{t_1,t_0} \wedge \neg\tau_{t_1,t_1})) && \backslash\backslash\ q' = q_e\\
&\wedge ((\delta_{t_1,q_1,q_2} \rightarrow (\neg\tau_{t_1,t_0}))) && \backslash\backslash\ q' = q_2\\
&\wedge (\lambda^{\mathbb{B}}_{t_0,q_2} && \backslash\backslash\ q = q_2, t = t_0\\
&\rightarrow ((\delta_{t_0,q_2,q_0} \rightarrow (\neg\tau_{t_0,t_0} \wedge \neg\tau_{t_0,t_1})) && \backslash\backslash\ q' = q_0\\
&\wedge ((\delta_{t_0,q_2,q_1} \rightarrow (\neg\tau_{t_0,t_1})) && \backslash\backslash\ q' = q_1\\
&\wedge ((\delta_{t_0,q_2,q_e} \rightarrow (\neg\tau_{t_0,t_0} \wedge \neg\tau_{t_0,t_1})) && \backslash\backslash\ q' = q_e\\
&\wedge ((\delta_{t_0,q_2,q_2} \rightarrow (\neg\tau_{t_0,t_0}))) && \backslash\backslash\ q' = q_2\\
&\wedge (\lambda^{\mathbb{B}}_{t_0,q_2} && \backslash\backslash\ q = q_2, t = t_0\\
&\rightarrow ((\delta_{t_1,q_2,q_0} \rightarrow (\neg\tau_{t_1,t_0} \wedge \neg\tau_{t_1,t_1})) && \backslash\backslash\ q' = q_0\\
&\wedge ((\delta_{t_1,q_2,q_1} \rightarrow (\neg\tau_{t_1,t_0} \wedge \neg\tau_{t_1,t_1})) && \backslash\backslash\ q' = q_1\\
&\wedge ((\delta_{t_1,q_2,q_e} \rightarrow (\neg\tau_{t_1,t_0} \wedge \neg\tau_{t_1,t_1})) && \backslash\backslash\ q' = q_e\\
&\wedge ((\delta_{t_1,q_2,q_2} \rightarrow (\neg\tau_{t_1,t_0} \wedge \neg\tau_{t_1,t_1}))) && \backslash\backslash\ q' = q_2
\end{aligned}
$$