MASARYK UNIVERSITY
FACULTY OF INFORMATICS

Parameter Synthesis from Hypotheses Formulable in CTL Logic

BACHELOR THESIS

Samuel Pastva

Brno, Spring 2015

Declaration

Hereby I declare, that this paper is my original authorial work, which I have worked out by my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Samuel Pastva

Advisor: John Foo, Ph.D.

Acknowledgement

I would like to thank my supervisor . . .

Abstract

The aim of the bachelor work is to provide . . .

Keywords

keyword1, keyword2 . . .

Contents

1 Introduction

Lorem Ipsum [**?**]

2 Terms and Definitions

2.1 Kripke Structure

As an input of our algorithm, we expect a parametrized Kripke Structure as defined in [CITE]. Parametrized Kripke Structure is a tuple $\mathcal{K} = (\mathcal{P}, S, S_0, \rightarrow, L)$ where

- $\mathcal{P}$ is a finite set of parameters (all possible parameter valuations)
- S is a finite set of states
- $S_0 \subseteq S$ is a set of initial states
- $\rightarrow \subseteq S \times \mathcal{P} \times S$ is a transition relation labeled by parameter valuations
- $L : S \rightarrow 2^{AP}$ is a labeling function from states to sets of atomic propositions which are true in such states

We write $s \xrightarrow{p} s'$ when $(s, p, s') \in \rightarrow$. We also write $s \rightarrow s'$ when $\exists p \in \mathcal{P}.(s, p, s') \in \rightarrow$. Note that fixing a valuation $p \in \mathcal{P}$ reduces the Parametrized Kripke Structure $\mathcal{K}$ to concrete, non-parametrized Kripke Structure $\mathcal{K}(p) = (S, S_0, \xrightarrow{p}, L)$.

2.2 CTL Logic

In order to correctly express various hypotheses in systems biology, the idea of branching time is need. Examples of such hypotheses are given in section [Case Study]. Therefore, this work uses the Computation Tree Logic(CTL) as means of hypotheses formulation.

CTL syntax is defined inductively upon finite set of atomic propositions:

$$\varphi ::= true \mid false \mid Q \mid \neg\varphi_1 \mid \varphi_1 \wedge \varphi_2 \mid \mathrm{EX}\varphi_1 \mid \mathrm{E}\varphi_1\mathrm{U}\varphi_2 \mid \mathrm{A}\varphi_1\mathrm{U}\varphi_2 \tag{2.1}$$

Sometimes, we will use parentheses to make bigger formulas easily readable, but they will in no way be used to modify the meaning of formula or priority of operators.

Note that there are also other temporal operators in standard CTL definition. We do not implement those directly in our algorithm. However, we use following equations to transform any general CTL formula prior to computation, so that it only uses operators supported in our algorithm. This way we can achieve a concise algorithm and also support whole CTL logic.

- $\mathrm{AX}\varphi = \neg\mathrm{EX}\neg\varphi$
- $\mathrm{EF}\varphi = \mathrm{E}(\mathit{true})\mathrm{U}\varphi$
- $\mathrm{EG}\varphi = \neg\mathrm{A}(\mathit{true})\mathrm{U}\neg\varphi$
- $\mathrm{AF}\varphi = \mathrm{A}(\mathit{true})\mathrm{U}\varphi$
- $\mathrm{AG}\varphi = \neg\mathrm{E}(\mathit{true})\mathrm{U}\neg\varphi$

Note that all of these transformations also preserve number of temporal operators in a formula.

Other boolean operators like implication or equivalence can also be derived using similar transformations.

Let φ be a CTL formula. We write $cl(\varphi)$ to denote the set of all sub-formulas of φ and $tcl(\varphi)$ to denote the set of all temporal sub-formulas of φ. By $|\varphi|$ we denote the size of formula φ.

We assume standard CTL semantics over non-parametrized Kripke structures as defined in [cite].

2.3 Parameter Synthesis Problem

Parameter synthesis problem is defined in following way. Suppose we are given a parametrised Kripke structure $\mathcal{K} = (\mathcal{P}, S, S_0, \to, L)$ and a CTL formula φ. For each state $s \in S$ let $P_s = \{p \in \mathcal{P} \mid s \models_{\mathcal{K}(p)} \varphi\}$,where $s \models_{\mathcal{K}(p)} \varphi$ denotes, that φ is satisfied in the state s of $\mathcal{K}(p)$. The parameter synthesis problem requires to compute the function $\mathcal{M}_\varphi^{\mathcal{K}} : S \to 2^{\mathcal{P}}$ such that $\mathcal{M}_\varphi^{\mathcal{K}}(s) = P_s$. Often we are especially interested in computing the set of all parameters for which the property

holds in some initial states $\cap_{s \in S_0} \mathcal{M}_{\varphi}^{\mathcal{K}}(s)$. We will sometimes omit the φ and $\mathcal{K}$ when they are clear from the context.

2.4 CTL logic and model approximation

In model checking, some modeling approaches suffer from over or under approximation. We say that model is over-approximated when all feasible transitions are contained in the model, but it can also contain transitions that are not feasible in the situation the model is describing. Symmetrically, we say that model is under-approximated when all transitions in the model are feasible in the modeled situation, but not all feasible transitions has to be contained in the model.

It is important to discuss this relationship between CTL and approximated models, because it is much more complicated compared to linear-time logic since CTL allows for universal and existential quantification mixing.

We say that CTL formula is *universal* or that it belongs to ACTL when it only contains universal temporal operators and no negation. Symmetrically, we say that CTL formula is *existential* or that it belongs to ECTL when it only contains existential temporal operators and no negation.

Observe that the truth of ACTL properties is preserved in over-approximated models. In other words, if an ACTL property holds in an over-approximated model, it must also hold in the original model. However, their falsity cannot be guaranteed, because the false transitions may introduce paths that falsify the property in states where it would be normally true. Similarly, the falsity of ECTL properties in over-approximated models is preserved but the truth is not. In this case, the existence of false transitions can introduce states where ECTL property holds solely due to these false paths.

Symmetrically, for under-approximated models, the falsity of ACTL and the truth of ECTL is preserved. But due to similar arguments, we can't say anything about their counterparts.

If we allow full CTL, in general, we can't make any assumptions about results obtained from either under- or over-approximated systems. This is caused by mixing of existential and universal quantification which leads to results which may be spurious and incomplete

at the same time. Therefore, no conclusions can be made without further investigation and validation of such results.

2.5 Kripke Fragments

Due to the state space explosion, given parametrised Kripke structure can be very large and therefore impossible to fit into memory of one computer. In order to solve parameter synthesis problem for such structures, we have to distribute the state space across several computational nodes. To this end, we introduce the notion of parametrised kripke fragments.

A parametrised Kripke structure $\mathcal{K}$ can be divided into several parametrised Kripke fragments $\mathcal{F}_1^{\mathcal{K}}, \mathcal{F}_2^{\mathcal{K}}, \cdots, \mathcal{F}_N^{\mathcal{K}}$. Parametrised Kripke fragment over Kripke structure $\mathcal{K} = (\mathcal{P}, S, S_0, \rightarrow, L)$ is then defined as a tuple $\mathcal{F}_i^{\mathcal{K}} = (f, \mathcal{P}, S_i, I_i, \rightarrow_i, L_i)$ where:

- $f : S \rightarrow \{1, \cdots, N\}$ is a partition function from all states to fragment indexes
- $\mathcal{P}$ is a finite set of all parameters
- $S_i = \{s \in S \mid f(s) = i \vee \exists s' \in S.((s \rightarrow s' \vee s' \rightarrow s) \wedge f(s') = i\}$ is a subset of original state space which belongs to this fragment
- $I_i = \{s \in S_0 \mid s \in S_i\}$ is a set of all fragments initial states
- $\rightarrow_i = \{(s, p, s') \in \rightarrow \mid s \in S_i \wedge s' \in S_i\}$ is a subset of original transition function reduced to only relevant states
- $L_i = \{(s, l) \in L \mid f(s) = i\}$ is a labeling subset of original labeling function relevant to this fragment

3 Algorithm

In this chapter, we describe the distributed algorithm that computes the assumption function $\mathcal{A}$.

3.1 Distributed Environment

In this section, we briefly describe the distributed environment assumed by our algorithm, in order to prevent any possible confusion.

We assume a distributed environment with fixed number of reliable processes connected by reliable, order-preserving channels (The order preservation can be relaxed to some extent). We also assume that each process has a fixed identifier and the set of all process identifiers is equal to the result set of the partition function. Each process can communicate directly (using the function SEND) with any other process (assuming it knows other process's identifier) and all messages that can't be processed directly are stored in a queue until they can be processed.

Several parts of the algorithm do not have explicit termination (they terminate by reaching deadlock - no messages are exchanged between processes). In such cases, suitable termination detection algorithm is employed to detect this deadlock and terminate computation properly. Our implementation uses Safra's algorithm [CITATION] for this purpose, but the related code has been skipped for easier readability.

The algorithm is broken into two main parts. First describes the general outline of algorithm and is similar to classic CTL model checking. Second part describes how each of the supported temporal operators is processed. This part contains more detailed description of inter-process communication and operator specific data structures. To better reflect the distributed nature of the algorithm, description of each temporal operator routine is divided into three parts: Process variables, Initialization and Message handler. First section describes data structures stored in process memory available during whole computation. Initialization section is executed exactly once and no messages can be received until it's finished. Message handler defines what should happen when message is received.

3.2 Algorithm outline

The main idea of the algorithm is described in REFERENCE and resembles other CTL model checking algorithms.

1: **procedure** CHECKCTL($\phi, \mathcal{K} = (\mathcal{P}, S, S_0, \rightarrow, L)$)
2: $\quad \mathcal{A} \leftarrow \{(p, s, \alpha, \mathtt{tt}) \mid p \in \mathcal{P} \wedge \alpha \in L(s)\}$
3: $\quad$ **for all** $i < |\phi|$ **do**
4: $\quad\quad$ **for all** ψ **in** $cl(\phi)$ **where** $|\psi| = i$ **do**
5: $\quad\quad\quad \mathcal{A} \leftarrow$ CHECKFORMULA$(\psi, \mathcal{K}, \mathcal{A})$
6: $\quad\quad$ **end for**
7: $\quad$ **end for**
8: **end procedure**

The algorithm starts by initializing the assumption function using the labeling function defined in kripke fragment. After that, it traverses the structure of formula, starting from smallest formulas and uses computed results to process more complex formulas. Function CHECKFORMULA computes all states and colors where formula ψ holds and returns assumption function updated accordingly. This is done using the local information contained in given kripke fragment, assumptions previously computed for smaller formulas and also by communicating with other processes. Note that only assumptions relevant for particular process are computed and returned (each process has information only about it's own state space).

3.3 Common operations

In this section, we define functions used to simplify the algorithm description. Let us fix a formula ϕ and a parametrised kripke fragment $\mathcal{K} = (\mathcal{P}, S, S_0, \rightarrow, L)$ as an input of the algorithm.

Intuitively, function $validStates : cl(\phi) \times AS_{\mathcal{K}}^{\phi} \rightarrow S \times 2^{\mathcal{P}}$ computes a set of states and parameters where truth of given formula is assumed. It is also responsible for handling of boolean operators, since these can be computed without any inter-process communication.

$$validStates(\phi_1 \wedge \phi_2, \mathcal{A}) = \{(s, p) \mid \mathcal{A}(s, p, \phi_2) = \mathtt{tt} \wedge \mathcal{A}(s, p, \phi_2) = \mathtt{tt}\}$$

$$validStates(\phi_1 \vee \phi_2, \mathcal{A}) = \{(s,p) \mid \mathcal{A}(s,p,\phi_2) = \texttt{tt} \vee \mathcal{A}(s,p,\phi_2) = \texttt{tt}\}$$

$$validStates(\neg\phi, \mathcal{A}) = \{(s,p) \mid \mathcal{A}(s,p,\phi) = \texttt{ff}\}$$

$$validStates(\texttt{tt}, \mathcal{A}) = \{(s,p) \mid s \in S \wedge p \in \mathcal{P}\}$$

$$validStates(\texttt{ff}, \mathcal{A}) = \emptyset$$

$$validStates(\phi, \mathcal{A}) = \{(s,p) \mid \mathcal{A}(s,p,\phi) = \texttt{tt}\}$$

Another useful function is $validColours : cl(\phi) \times S \times AS^{\phi}_{\mathcal{K}} \to 2^{\mathcal{P}}$ which returns a set of parameters for which given formula is assumed to be true in given state.

$$validColours(\phi, state, \mathcal{A}) = \{p | (state, p) \in validStates\}$$

We also write $successors_{\mathcal{K}} : S \to S \times 2^{\mathcal{P}}$ which computes set of direct successors of given node including the color sets labeling the appropriate transitions.

$$predecessors(from) = \{(to, P) \mid P = \{p \mid from \xrightarrow{p} to\}\}$$

Symmetrically, function $predecessors : S \to S \times 2^{\mathcal{P}}$ computes set of direct predecessors of given node including the color sets labeling the appropriate transitions.

$$predecessors(to) = \{(from, P) \mid P = \{p \mid from \xrightarrow{p} to\}\}$$

We also define function $update : AS^{\phi}_{\mathcal{K}} \times S \times 2^{\mathcal{P}} \times tcl(\phi) \to AS^{\phi}_{\mathcal{K}}$ which takes current assumptions, a state, set of parameters and a formula and returns assumptions updated so that for all parameters of the given set, formula holds in given state.

$update(\mathcal{A}, state, colours, \phi)$:
for all $p \in colours$ **do**
  Set $\mathcal{A}(state, p, \phi) = \texttt{tt}$
end for

3.4 Temporal Operators

In this section, we describe how the CHECKFORMULA is implemented for each of the temporal operators. Note that all of the following algorithms has implicit termination and therefore needs a proper termination detection algorithm to correctly terminate.

3.4.1 Exist Next Operator

```
1: Process variables:
2: K = (id, f, P, S, I, -p->, L)                ▷ Kripke fragment
3: φ = EXφ1                                      ▷ CTL formula
4: A                                             ▷ Initial assumption function
5: procedure INIT
6:     for all (state, colSet) in validStates(φ1, A) do
7:         for all (pred, tranCol) in predecessors(state) do
8:             SEND(f(state), (pred, colSet ∩ tranCol))
9:         end for
10:    end for
11: end procedure
12: procedure RECEIVE(colSet, to)
13:    A ← update(A, to, colSet, φ)
14: end procedure
```

The simplest of temporal operators is the EX operator. During initialization, all states and colors where ϕ holds are computed. For each of such states, all predecessors are considered and appropriate message that will cause assumption update is sent.

3.4.2 Exist Until Operator

```
1: Process variables:
2: K = (id, f, P, S, I, -p->, L)                ▷ Kripke fragment
3: φ = Eφ1Uφ2                                    ▷ CTL formula
4: A                                             ▷ Initial assumption function
5: procedure INIT
6:     for all (state, colSet) in validStates(φ2, A) do
7:         A ← update(A, state, colSet, φ)
8:         for all (pred, tranCol) in predecessors(state) do
```

```
9:              SEND(f(state), (pred, colSet ∩ tranCol))
10:         end for
11:     end for
12: end procedure
13: procedure RECEIVE(colSet, state)
14:     colSet ← colSet ∩ valid(φ1, to, A)
15:     if colSet ≠ ∅ and colSet \ valid(φ, to, A) ≠ ∅ then
16:         A ← update(A, to, colSet, φ)
17:         for all (pred, tranCol) in predecessors(to) do
18:             SEND(f(pred), (pred, colSat ∩ tranCol))
19:         end for
20:     end if
21: end procedure
```

The EU operator is a little more complex, but again fairly simple. The algorithm starts by computing all states and colours where ϕ_2 is true. Starting from these states, a backpropagation of parameter sets along the reversed transitions is performed. During the computation, the propagated parameter set is updated to reflect the validity of ϕ_1 and the validity of transitions used along the path. Note that backpropagation is stoped as soon as there is no new information computed (*colSat* is either empty or equal to already computed assumptions).

3.4.3 All Until Operator

```
1: Process variables:
2: K = (id, f, P, S, I, -p->, L)          ▷ Kripke fragment
3: φ = Aφ1Uφ2                             ▷ CTL formula
4: T = -p->                               ▷ Uncovered edges
5: A                                      ▷ Initial assumption function
6: procedure INIT
7:     for all (state, colSet) in validStates(φ2) do
8:         for all (pred, tranCol) in predecessors(state) do
9:             SEND(f(state), (state, pred, P ∩ tranCol))
10:        end for
11:    end for
12: end procedure
```

13: **procedure** RECEIVE($colSet, from, to$)
14: $T \leftarrow T \setminus \{(to, p, from) | p \in colSet\}$
15: $colSet \leftarrow \{p | p \in colSet \wedge \forall s_2 \in S.(to, p, s_2) \notin T\}$
16: $colSet \leftarrow colSet \cap valid(\phi_1, to, \mathcal{A})$
17: **if** $colSet \neq \emptyset$ **and** $colSet \setminus valid(\phi, to, \mathcal{A}) \neq \emptyset$ **then**
18: $\mathcal{A} \leftarrow update(\mathcal{A}, to, colSet, \phi)$
19: **for all** $(pred, tranCol)$ **in** $predecessors(to)$ **do**
20: SEND$(f(pred), (to, pred, colSet \cap tranCol))$
21: **end for**
22: **end if**
23: **end procedure**

The AU operator is the most complex one to handle. As opposed to EX, which requires at least one valid successor to be true, AU requires that all successors of the specific node are valid. In order to compute such information, we create a copy of transition relation and call it T.

During the computation, T is modified in such way, so that we can guarantee that if edge is not present in T, this edge leads to a state where either ϕ_2 or $A\phi_1 U\phi_2$ holds. This way, we can guarantee that only appropriate states and colors are marked as valid by our algorithm.

A First appendix

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

This is the second paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

And after the second paragraph follows the third paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

After this fourth paragraph, we start a new paragraph sequence. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information

about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

B Another appendix

This is the second paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

And after the second paragraph follows the third paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

After this fourth paragraph, we start a new paragraph sequence. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information

about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

This is the second paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.