Symbolic Algebraic Discrete Systems - Applied to the JAS 39 Fighter Aircraft, Part II*

Johan Gunnarsson
Division of Automatic Control
Department of Electrical Engineering
Linköping University
S-581 83 Linköping, Sweden
johan@isy.liu.se
http://control.isy.liu.se
August 30, 1996

Submitted to the 35th Conference on Decision and Control

Abstract
Symbolic algebraic analysis techniques are applied to the landing gear subsystem in the new Swedish fighter aircraft, JAS 39 Gripen. Our methods are based on polynomials over finite fields (with Boolean algebra and propositional logic as special cases). Polynomials are used to represent the basic dynamic equations for the processes (controller and plant) as well as static properties of these. Temporal algebra (or temporal logic) is used to represent specifications of system behavior. These specifications are verified both on a model of the landing gear controller, and a model of the closed loop behavior of the landing gear controller connected to a plant. The model of the landing gear controller is made from the actual implementation in Pascal.

The tools used are developed by the authors in Mathematica and uses an efficient implementation of binary decision diagrams (BDDs).

This project demonstrates possible use and feasibility of these methods and tools on a complex industrial process (the landing control sub system on a fighter). It also highlights some of the user aspects involved in working with this methodology, both computational and principal.

Keywords: Discrete Event Systems, Temporal Algebra, Polynomials over Finite Fields, Dynamic Verification, Application.

1 Introduction

We have modeled and analyzed an existing discrete subsystem of a modern fighter aircraft, the landing gear system on the JAS 39 Gripen. This system was designed and implemented

*This work was supported by the Swedish National Board for Industrial and Technical Development (NUTEK), which is gratefully acknowledged.
without any formal methods or tools as is usually the current practice for discrete dynamic systems in industry today [4]. We have built a mathematical model of this system and analyzed its behavior w.r.t. to this specification. The main focus has not been on the specific system, but rather on the general methods that can be applied to discrete dynamic systems of industrial size, e.g. the process is fairly complex, with some 80 variables, of which 66 are Boolean.

This paper describes the second part of the project, where the focus is on analysis. The objective of the first part of this project was to build a mathematical model of the behavior of the landing gear controller (LGC). This was done by the development of a compiler that translates Pascal code to model of polynomial relations \( M(z, z^+) \). Further information of this work can be found in [7, 8, 5]. In this paper we will regard \( M(z, z^+) \) as a given polynomial model of the LGC.

We will describe the methods and tools used in this project. Then an overview of the landing gear system will be given before we present the dynamic verification performed on this application. Finally there will be some conclusions.

2 Methods and Tools

2.1 The Polynomial Framework

Quantities and relations in finite discrete event systems (DES) can be represented by finite relations. These relations are in turn represented mathematically by polynomials over finite fields \( \mathbb{F}_q[\mathbb{Z}] \), i.e. polynomials of variables in the set \( \mathbb{Z} \) with coefficients from a finite field \( \mathbb{F}_q \). By further restricting the class of polynomials we construct a quotient polynomial ring (see [3]) that gives a one to one correspondence between polynomials and relations as well as a compact representation of the relations.

The computational framework used for manipulating polynomials is based on binary decision diagrams (BDD), which give a powerful representation as well as fast computations which allow us to manipulate rather complex systems.

For more information of polynomials over finite fields and its tools see the tutorial paper [6].

2.2 Modeling

By modeling we essentially mean building a mathematical model. The preferred mathematical model type for our purposes is a polynomial model of the form:

\[
M(z, z^+) \tag{1}
\]

where \( z = [z_1, \ldots, z_n] \) are all the the system variables typically classified as \textit{input}, \textit{output} and \textit{state}, but other groupings could be possible. Furthermore \( z^+ \) denotes the value of \( z \) one time instant into the future.

The model \( M \) is a polynomial in \( z \) and \( z^+ \). In particular we will use the finite field \( \mathbb{F}_2 = \{0, 1\} \) in this application. This means that our polynomials are essentially Boolean polynomials. The main reason is the application itself and the (large scale) tools (see Section 2.4) we have developed so far.

A state space model as used in e.g. simulation typically is of the form:

\[
x^+ = f(x, u), \quad y = g(x, u) \tag{2}
\]
This is just a special case of the model in equation (1) since if we let \( z = [x; u; y] \) we get:

\[
M(z, z^+) := (x^+ = f(x, u)) \land (y = g(x, u)) \land (y^+ = g(x^+, u^+))
\]  
(3)

where \( \land \) denotes \textit{and}.

### 2.3 Analysis – Verification

#### 2.3.1 One Step Analysis

By doing one step analysis we look at a single step of the system dynamics, i.e. to resolve equation systems of the form:

\[
M(z, z^+) \land R_1(z) \land R_2(z^+)
\]  
(4)

where \( M(z, z^+) \) is the process description and \( R_1(z) \) and \( R_2(z^+) \) are restrictions on \( z \) and \( z^+ \) respectively. The actual analysis job is then to solve the system of equations or to prove that no such solution exists. The restrictions \( R_1 \) and \( R_2 \) may be constructed from several constraints combined by the usual Boolean operators \( \land, \lor, -, \leftrightarrow, \rightarrow \).

#### 2.3.2 Dynamic Analysis

By dynamic (or multiple step) analysis we mean analysis questions over arbitrarily many time steps. The results from these questions may be a simple \textit{true} or \textit{false} or a set of states, e.g., the set of states that are reachable in zero or arbitrarily many steps from some initial state.

Given a process model \( M(z, z^+) \) we can compute the set of states \( R_k(z) \) reachable in \( k \) steps or less from some initial set of states \( I(z) \) as:

\[
R_0(z) := I(z)
\]

\[
R_{k+1}(z) := R_k(z) \lor (\exists \hat{z} \ (R_k(\hat{z}) \land M(\hat{z}, z)))
\]  
(5)

In each iteration we compute a polynomial representing all states that are reachable from the states represented by \( R_k(z) \). This set is added by union to \( R_k(z) \).

Since we are dealing with finite state systems this iteration will reach a fixed point, i.e. \( R_d(z) = R_{d+1}(z) \) for some finite \( d \). The number of steps \( d \) is the \textit{depth} of the system which in most engineering applications is far below its maximal possible depth which is \( 2^n \) for a \( n \) variable system. The depth of LGC is 5 in our case.

Alternative methods such as testing or simulation are infeasible for complex systems, e.g. in the LGC we have in the order of 10,000 reachable states out of \( 2^{26} \) potentially reachable states.

---

**Example 1** Suppose we have the simple system below:

![Diagram of a system with states 0, 1, and 2 connected by transitions]

The transitions of this system could be described by the relation \( R = \{(0, 1), (1, 2)\} \). We can obtain a polynomial model by first encoding the states 0, 1, 2 as binary vectors and then to Boolean expressions:

\[
0 \mapsto [0, 0] \mapsto \neg x_1 \land \neg x_2, \ \ 1 \mapsto [0, 1] \mapsto \neg x_1 \land x_2, \ \ 2 \mapsto [1, 0] \mapsto x_1 \land \neg x_2
\]
From the relation \( R \) and the state encoding above we get the polynomial model:

\[
M(x, x^+) := \{ [\neg x_1 \land \neg x_2] \land [\neg x_1^+ \land x_2^+] \} \lor \{ [\neg x_1 \land x_2] \land [x_1^+ \land \neg x_2^+] \}
\]

We can now compute the set of reachable states from state \( 0 \), using (5):

\[
\begin{align*}
R_0(x) &:= I(x) - \neg x_1 \land \neg x_2 \\
R_1(x) &:= [\neg x_1] \lor [\neg x_2] \\
R_2(x) &:= [\neg x_1 \land \neg x_2] \lor [\neg x_1] \land x_2 \\
R_3(x) &:= [\neg x_1 \land \neg x_2] \lor [\neg x_1] \land x_2 \\
\end{align*}
\]

Hence we reach a fixed point for \( k = 2 \) steps, i.e. in two steps we can reach any reachable state.

In this example we could not have found out that 2 is a reachable state by just static analysis of \( M(x, x^+) \) and the initial state information. In some cases this is important since some undesirable action might be performed by the controller if it ever reaches state 2.

There are a multitude of other types of dynamic analysis that are possible and many of them are related to the idea of reachable states either backward or forward in time.

### 2.3.3 Temporal Algebra and Verification

Since many specifications are written in something close to natural language, we could greatly simplify our analysis task if we could more or less directly translate this to a formal specification. In this application we have used temporal algebra (or temporal logic since we use the binary Boolean algebra) to achieve this task. In Table 1 some of the most common temporal algebra constructs are given. Furthermore there is a set of dual constructs to the ones in Table 1 where \( E \) is replaced by \( A \) having the meaning that we exchange the words can hold with must hold. A simple instance of this is:

\[
AX[P(z)] \sim P(z) \text{ must hold for all possible next time instants}
\]

In this manner we have attempted to interpret parts of the landing gear specifications into temporal algebra expressions. Temporal Algebra is an extension of the specification language computation tree logic (CTL) \([2]\), made for polynomial systems over finite domains by \([3]\).

The analysis part, or verification, is a mixture of one step and multiple step analysis. For each temporal algebra expression \( f(z) \) and process model \( M(z, z^+) \) we compute the set of states from which the temporal algebra statement becomes true.

Assume that we have process model \( M(z, z^+) \) and a temporal algebraic expression \( f \). Then the verification \( V(M, f) \) is computed as follows:

<table>
<thead>
<tr>
<th>Temporal Algebra</th>
<th>Natural Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P(z) )</td>
<td>( P(z) ) holds in the initial state.</td>
</tr>
<tr>
<td>( EX[P(z)] )</td>
<td>( P(z) ) can hold in the next time step.</td>
</tr>
<tr>
<td>( EU[P_1(z), P_2(z)] )</td>
<td>( P_1(z) ) will hold for finitely many steps and then ( P_2(z) ) can hold in the next step.</td>
</tr>
<tr>
<td>( EF[P(z)] )</td>
<td>( P(z) ) can hold at some future time.</td>
</tr>
<tr>
<td>( EG[P(z)] )</td>
<td>( P(z) ) can hold at all future times, i.e. from this point onwards.</td>
</tr>
</tbody>
</table>

Table 1: Some of the most common temporal algebra constructs.
**Atomic expressions:** Let \( f \) be an atomic expression. (Boolean in this case.)

\[
V(r, f) := f
\]

**Combination expression:**

\[
V(M, \neg f) := \neg V(M, f) \\
V(M, f \land g) := V(M, f) \land V(M, g) \\
V(M, f \lor g) := V(M, f) \lor V(M, g)
\]

**Next state expression:**

\[
V(M, EX[f]) := \exists \bar{z}. M(z, \bar{z}) \land V(M, f)[\bar{z}]
\]

**Future state expression:**

\[
V(M, EU[f,g]) := \lim_{k \to \infty} V(M, EUL_k[f,g]) \\
V(M, EF[f]) := V(M, EU[true, f]) \\
V(M, EG[f]) := V(M, \neg EU[true, \neg f])
\]

where

\[
EUL_{k+1}[f,g] := g \lor f \land EX[EUL_k[f,g]], \quad EUL_0[f,g] := g.
\]

**Example 2** Consider the process from example 1. We wish to verify the specification:

We should always be able to reach the safe state 2 as the next state.

In terms of temporal algebra this becomes:

\[
EX[2] = EX[x_1 \land \neg x_2]
\]

where we have used the algebraic encoding to the right. The actual verification then computes:

\[
V(M(x, x^+), EX[x_1 \land \neg x_2]) := \exists x^+ M(x, x^+) \land (x_1^+ \land \neg x_2^+) \\
:= (\neg x_1) \land x_2
\]

As expected this returns the state 1 in its encoded form, since this is the only state from which we can reach 2. Suppose we now have the process and an initial state specified, then the above temporal algebra formula would be verified if the returned set of states was a superset of the reachable states, i.e. we could reach 2 from every reachable state. In the case above this is clearly not the case if our initial state is 0, since the set of reachable states is \([0, 1, 2]\) in that case. Generally this extra level of reasoning is of course built into our verifier.

The verification of temporal algebra expressions require the same type of fixed point computations as was seen for the reachability analysis above. For more details regarding temporal algebra (or temporal logic), see [2].
2.4 Software Tools

In this project we use a experimental software system consisting of Mathematica [9] code together with externally linked C code for critical operations through the MathLink structured communication protocol.

The C code used in this package is an efficient implementation of binary decision diagrams (BDDs) which we use as a computation engine for polynomials in $\mathbb{F}_2 [Z]$. BDD is a method to represent large Boolean expressions. Boolean expressions and polynomials in $\mathbb{F}_2 [Z]$ is essentially the same thing, and BDD can be used to effectively represent polynomials in $\mathbb{F}_2 [Z]$. Efficient tools for BDD exists [1] whereas there is yet no usable tool for the more general $Q$-ary Decision Diagrams (QDD) for polynomials in $\mathbb{F}_q [Z]$, see [3]. Therefore BDDs are used in this project.

The basic idea used in BDDs is to rewrite Boolean (or $\mathbb{F}_2 [Z]$) expressions in a recursive form and reuse common subexpressions, a technique that has been used in compiler optimization for several decades. In the case of Boolean expressions this leads to highly efficient computations in most cases. Using an efficient symbolic algebraic computation engine is crucial if we are to be able to analyze realistically sized examples.

2.4.1 Modeling

In the modeling part of the project the implemented Pascal code of the LGC was compiled to a polynomial model $M(z,z^+)$. The Pascal code is first parsed to an intermediate code called MPascal which essentially is a the same Pascal code written as a Mathematica expression. This code is then processed by a compiler, also written in Mathematica. The result from the compiler is a polynomial model $M(z,z^+)$ represented as a BDD, where all relations between input variables and output variables are stored whereas temporary variables in the code are removed. See [7, 5] for details.

2.4.2 Analysis

Tools for analysis of the polynomial models was developed in Mathematica as well. For these tools the efficiency of the underlying computation engine is even more important than for modeling. For multiple step analysis we have to do fix point computations, i.e., to iterate until the answer remains the same between two iterations, and it is essential to reduce data complexity in these iterations. In our case this is done by the BDD package that always represent expressions a simple as possible with respect to the variable order chosen.
### Commands

<table>
<thead>
<tr>
<th>Commands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ReachableStates[M[z,z^+],I[z]]</code></td>
<td>Returns the set of states reachable from <code>I[z]</code> for the model <code>M[z,z^+]</code>. See (5).</td>
</tr>
<tr>
<td><code>BDDTLEvaluate[M[z,z^+], F]</code></td>
<td>Returns the set of states from which the temporal algebra expression <code>F</code> is true. <code>BDDTLEvaluate</code> works as the operator <code>V</code> in Section 2.3.3.</td>
</tr>
<tr>
<td><code>BDDSolve[R[z]]</code></td>
<td>Returns all solutions of <code>R[z] = true</code> for the variables <code>z</code>.</td>
</tr>
<tr>
<td><code>BDDCountSolutions[R[z]]</code></td>
<td>Counts all solutions of <code>R[z] = true</code>. This is often more interesting than a huge set of solutions.</td>
</tr>
<tr>
<td><code>BDDRandomSolve[R[z]]</code></td>
<td>Returns one possible solution of <code>R[z] = true</code> chose by random.</td>
</tr>
</tbody>
</table>

Table 2: Some of the most common commands for analysis.

Figure 2: The fighter JAS 39 Gripen.

To give a taste of how to use the analysis tools Table 2 shows the most important commands used in the project.

The commands `BDDSolve` and `BDDCountSolutions` searches recursively through the BDD graph starting at the top down to the constant 1. `BDDRandomSolve` chose one possible path down to the constant 1.

### 3 The Landing Gear Process

#### 3.1 Overview

The case studies in Section 4 concerns the landing gear system on the Swedish fighter JAS 39 Gripen, depicted in Figure 2.

The landing gear system consists of the LGC and three landing gears with corresponding doors. A simplified block description of the complete system is shown in Figure 3, where the arrows should be interpreted as signal vectors.
3.2 The Physical System and Signal Interface

Besides the fact that the LGC itself is discrete, the domains of all actuator and measurement signals are discrete. However the underlying system is continuous.

All gears and doors are operated in parallel, which means that the actuators supports 4 different maneuvering commands, i.e. extension and retraction of gears and doors. The fifth actuator controls the hydraulic pressure in the landing gear.

The feedback from the landing gear consists of several switches placed on the gears and doors. Each switch has two contacts for security reasons. This makes a total of 30 binary signals from gears and doors.

Two binary signals detect pilot commands, and five mode signals give information from other system units in the aircraft. The mode signals take integer values $0, \ldots, 15$.

There are basically three maneuver types, retraction, extension and emergency extension. Ordinary extension or retraction is commanded by a lever in the cockpit. During emergency extension the control signals are generated by hardware logic and not by the LGC.

The LGC contains 3 timers used for detecting certain cases of timeout. The one most used in the analysis in Section 4 indicates that the extension maneuver is not completed within a certain time. When this timeout occur the pilot have to restart the extension by ordering retraction and then extension again.

4 Verification of Closed Loop Dynamics

This section presents the result obtained during the second part of this project where the objective was to verify specifications on the closed loop landing gear system.

At first, the task seemed to be too hard for the tools available. Previously we had managed to perform a modeling of a complex system from implemented code and also computed the set of reachable states in the LGC. But more fine tuning were needed both on the tools and on the modeling procedure to gain complexity advantages. Finally the efforts payed off and we managed to keep the computed data in the computer primary memory.

In this section we will go through the analysis step by step in the same order as was done in the project. This is a natural consequence of the sentence: *try simple things first*.

For the analysis below the LGC is assumed to be in normal mode, i.e. no emergency or failure modes allowed.

4.1 Open loop analysis

First we restate the result of the dynamic verification performed locally on the LGC (see [8, 5, 3]).

The number of state variables in the LGC is 26 which means that the maximal number of different states is $2^{26} \approx 10^8$. By computing the set of reachable states $R_c(z)$ of the LGC...
it turn out that only 10015 states were reachable and the maximal depth was 5, i.e., every state can be reach in at most 5 iterations.

The polynomial relation $R_c(z)$ is used to restrict the polynomial model of the LGC as

$$C(z, z^+):= R_c(z) \land M(z, z^+) \land R_c(z^+)$$

where the model both in present state and in the next state is restricted to keep the model relation symmetric. The restriction of the polynomial model reduces complexity and indicates that we can make use of the sparse nature of the system.

Open loop means in this case that during the computations of the reachable states, all inputs to the LGC were unconstrained. This is not true in reality since the inputs connected to a plant which in turn are controlled by the LGC. It is also more natural to state specifications on the behavior of the plant or sometimes both controller and plant. Therefore we have to build a plant model.

### 4.2 Closed Loop Analysis

The physical plant of the landing gear system is the landing gear itself with inputs controlling the hydraulic actuators, and outputs connected to several switches. As mentioned in Section 3 the interface between the plant and the LGC is discrete which means that we can make a discrete model having the same behavior as seen from the LGC.

The LGC can only determine the state of the gears to be in three different regions: retracted, middle and extended. The same is also true for the doors which has the regions: closed, middle and open. All the gears and the doors are operated in parallel.

The simplest model having this behavior is a double 3-state automata. The automata for the gears will look like

where the inputs symbol Out$_G$ and In$_G$ are place holders for the LGC actuator outputs. The output from the plant are signals from the switches indicating the positions of gears and doors. For the gear model the outputs are defined as in Table 3.

For the doors we get a similar model.

The plant model for both gears and doors are written in MPascal and then compiled into polynomial model $P_1(z, z^+)$. The resulting model has 8 Boolean state variables. Each automaton has a single integer state variable, and integers are represented by 4 Boolean variables in this case. This makes it easy to enlarge the model later. The output variables are 12 and input variables are 5.
The structure of this model is
\[ \mathbf{P}_1(z_p, z_p^+) := (x_p^+ = f_p(x_p, u_p)) \land (y_p = g_p(x_p)). \]

The outputs \( y_p \) of the plant do not directly depend on the inputs \( u_p \). This is important to ensure that we avoid an algebraic loop when connecting plant and LGC. The structure of the LGC model is
\[ \mathbf{C}(z_c, z_c^+) := (x_c^+ = f_c(x_c, u_c)) \land (y_c = g(x_c, u_c)) \]
where \( y_c \) depend directly on \( u_c \).

The variables in the plant model \( \mathbf{P}_1(z_p, z_p^+) \) are the same as those in the LGC model. Therefore we can easily compute the closed loop system \( \mathbf{G}_c(z, z^+) \) by
\[ \mathbf{G}_1(z, z^+) := \mathbf{C}(z_c, z_c^+) \land \mathbf{P}_1(z_p, z_p^+) \]
where \( z = z_c \cup z_p \).

Before analyzing the closed loop model we need a specification or a test case from which we can formulate a temporal algebraic expression. For the landing gear system the most critical maneuver is extension, i.e., the landing gears should always reach the extended state when the pilot pushes the gear extension button. More formally we say:

The gear should always reach the extended state \( \text{Gear}(\text{ext}) \) in finite time, when pilot command is extension \( \text{Pilot}(\text{ext}) \).

This specification can directly be translated to temporal algebra expression \( \mathbf{AG}[\text{Pilot}(\text{ext}) \rightarrow \text{Gear}(\text{ext})] \). By verifying this expression we will get all states from which the specification is true. It is often more convenient to search for the states not fulfilling the specification. Therefore we define the first temporal expression as
\[ F_1(z) := \mathbf{EG}[-(\text{Pilot}(\text{ext}) \rightarrow \text{Gear}(\text{ext}))]. \]

Verifying this statement by
\[ S_1(z) := \text{BDDTEvaluate}[\mathbf{G}_1(z, z^+), F_1(z)] \]
we get a polynomial \( S_1(z) \) including 82 variables and with 5 different solutions. By analyzing these solutions further, using knowledge from the SAAB company, we found that for all 5 solutions a time out condition was set. This means that the analysis had found the cases of when extension time has exceeded its limit and the maneuver is stopped. The pilot then has to restart the extension by choosing retraction first and then extension. To avoid this trap we reformulate the specification as follows

Having the pilot command retract \( \text{Pilot}(\text{ret}) \) and in the next state having \( \text{Pilot}(\text{ext}) \) and not \( \text{TimeOut} \) the gear should always reach the state \( \text{Gear}(\text{ext}) \) in finite time.

As before we formulate the temporal expression for finding the errors.
\[ F_2(z) := \text{Pilot}(\text{ret}) \land \mathbf{EX}[\mathbf{EG}[-(\text{Pilot}(\text{ext}) \land \neg \text{TimeOut}) \rightarrow \text{Gear}(\text{ext})]] \]

This is verified by
\[ S_2(z) := \text{BDDTEvaluate}[\mathbf{G}_1(z, z^+), F_2(z)] \]
which returns $S_2(z) \equiv \text{false}$, i.e., we have proved that the gears will always reach the extended state provided conditions above. Note that we do not specify a initial state for the analysis. This means that this verification hold for all behaviors of the system preceding the this extension maneuver. By this we have proved a liveliness property, i.e., the system cannot be trapped in a dead-lock situation.

The plant model $P_1(z_{p}, z_{p}^+)$ used above has no errors or disturbances on the signals. But there is also a need to take some possible failures on the switches into account. Therefore another plant model was created from the first one, where the outputs are filtered and disturbed by two signals $E_G$ and $E_D$. See Figure 4. The disturbance mapping $\tau$ models the possibility of shortcuts in the switches measuring the state of the plant. Therefore when $E_G = \text{true}$ all gear switches are $\text{true}$, i.e., the gears seams to be both retracted and extended simultaneously. The disturbance $E_D$ works in the same way for the doors. We will refer to this plant model by the polynomial model $P_2(z_{p}, z_{p}^+)$. The closed loop system is

$$G_2(z, z^+) := C(z_c, z_c^+) \wedge P_2(z_{p}, z_{p}^+)$$

In spite of the disturbance on the plant outputs we want the LGC to fulfill its task as stated in $F_2(z)$, i.e., the extension maneuver should be completed in finite time for $P_2(z_{p}, z_{p}^+)$ also. By computing

$$S_3(z) := \text{BDDLEvaluate}[G_2(z, z^+), F_2(z)]$$

we get the result $S_3(z) \equiv \text{false}$, which shows that in spite of the disturbances of plant $P_2(z_{p}, z_{p}^+)$ the extension maneuver will always be completed.

For the plants used above we have distinguished between three different regions for the gears: retracted, middle and extended. We used a three state automaton as a model for the plant behavior. By this we have made an important assumption about the system. Since the plant model works synchronous to the LGC it might be relevant to consider how many iterations the LGC needs during a normal plant maneuver. For the plants $P_1$ and $P_2$ the LGC only needs 2 iterations to reach the end state of the plant. The real implemented LGC iterates several times per second, which means that a reasonable assumption is that it takes more than 20 iterations to fulfill a maneuver. To get a more realistic plant model we can build a model with several middle states. If we let the number of middle states be larger than the maximal depth of the LGC and if the dynamics of the LGC always reach a fixed point (the dynamics stops) then we know that this fixed point will be reached during the middle states. In our case the maximal depth of the LGC is 5. By choosing a plant with 8 states by which 6 are middle states we have an appropriate plant model.

However there are some disadvantages by this method. First of all the complexity increases since we add more states to the system. It turns out that by adding more states to the plant we get harder complexity problems doing verification compared to increasing the
complexity of the temporal algebra statements. Secondly, we can only verify the system for a fixed depth of the plant. We cannot in one verification test plants with several different depth. The last drawback is that we at this stage do not know if the LGC reaches a fixed point. This feature can be examined by our analysis tools, but it is not necessary if we use the power of temporal algebra instead.

The most general plant in the sense of capturing all possible depth would be the non-deterministic automaton which remains in the middle state an arbitrarily number of iterations and then goes to the end state. But if the plant should be used for liveness verification we have to have a model that terminates to an end states after a finite number of steps. We do this by introducing condition signals Man_G (Maneuver) for the gears and Man_D for the doors. See Figure 5 for the gear model. The model for the doors is constructed in the same way. The gear and door models are combined into a polynomial model P_3(z_p,z^+_p) which is used for the closed loop model

\[ G_3(z, z^+) := C(z_c, z^+_c) \land P_3(z_p, z^+_p). \] (11)

We adjust the definition of the temporal expression F_2(z) for the plant model P_3(z_p,z^+_p) such that the model will reach the extended state after a finite time. The result is:

\[ F_3(z) := Pilot(ret) \land \text{EX}[EU[F_2(z),EG[\neg \text{Man}_G \land \neg \text{Man}_D \land \tilde{F}_2(z)]]] \] (12)

where \( \tilde{F}_2(z) = \neg((\text{Pilot}(\text{ext}) \land \neg \text{TimeOut}) \rightarrow \text{Gear}(\text{ext})) \). In this way we have captured all plants with a finite depth.

Verifying this statement by

\[ S_4(z) := \text{BDDTEvaluate}[G_3(z, z^+), F_3(z)]. \]

The polynomial relation \( S_4(z) \) is identical to false which shows that we will always reach the extended states for all plants with finite depth.

This analysis was possible to do without increasing the complexity of the model, instead by using temporal expressions for building more general models we can analyze more complex behavior.

## 5 Conclusion

We have given an example of how one may verify a discrete dynamic control system by building a model of the whole process:

\[ G(z, z^+) := C(z, z^+) \land P(z, z^+) \] (13)
where $C(z, z^+)$ is the controller and $P(z, z^+)$ is the plant. Furthermore $G(z, z^+)$ is a polynomial over a finite field, which in this work has been the Boolean field.

We can also build a model of the specification $F$ using temporal algebra which let us specify conditions over time or sequences. Temporal algebra is also used to give simpler plantmodels with lower complexity. Using the closed loop system model $G(z, z^+)$ and the specification $F$ we can then either verify or falsify the system behavior w.r.t. the specification. In case we falsify the system behavior we can also generate a sequence of inputs that exhibits the failing behavior. This can then be independently verified in a system simulator and the error should be characterized well enough for modification of the controller.

The developed methods and tools allow us to analyze industrial scale discrete systems, using symbolic algebraic discrete systems theory. In particular this allows us to prove (or disprove) that the system behaves according to its specification.

For dynamic systems dynamic analysis will ultimately be needed and hence an algebraic computation engine that can handle dynamic analysis is necessary.

References


