An Approach to Guided Incremental Specification

Thomas Gabler and Sabine März-Rössel
Corporate Research and Development
Siemens AG
Munich, Germany

Abstract

In this paper we present an approach to guided incremental specification based on specifications derived from properties in terms of input and output waveforms. Since this leads to an incompletely specified finite automaton several different finite automata may reproduce the specified behaviour. This freedom is exploited to guide the designer. Intelligent questions are created from a graph-based representation of the automata.

1 Introduction

Finite state machines provide a powerful concept in hardware design especially for synthesis and validation both through formal verification or through simulation. The mathematical theory of finite automata is in an advanced state, for an introduction the reader is referred to [Gur89, Hol82, UH92, PE72].

While a finite automaton is a static object, the specification process is dynamic. This implies certain inaccuracies when modeling the intended functionality. It may for example happen that one does not want to specify the inputs and outputs in full generality at a certain stage of the specification process, or that one does not want to specify from the very beginning exceptional states the completed specification will contain. The process of specification can be understood as a finite sequence of property specifications resulting in a finite sequence of more and more precise specifications of the desired functionality. We assume the properties to be input and output waveforms with arbitrary characters in the input and output alphabets.

Incremental specification is then understood to be the transition from one stage of the specification process to the next. In this paper we present an automated approach to guided incremental specification. At any stage, a finite state machine reproducing the set of specified input and output waveforms can be generated. Since several different finite automata may reproduce the behavior specified through the waveforms, there is freedom for specification refinement. This freedom is exploited to guide the designer through questions which lead to an extension of the actual set of properties or input and output waveforms, respectively. As a by-product a synthesizable VHDL model can be generated by the underlying tool.

A lot of research is done in the area of automatic generation of HDL models from the specification. [Kho93] presents a modeling methodology for the development of executable HDL models. This method is based on the separate capture of interface specifications, functional specifications, and their interrelation. [Bor88] focuses on the synthesis of bus-functional models through building an event graph. A front-end tool for wave diagrams is used for constrained specification. An overview of some specification languages based on protocol engineering is given in [Boc93]. This paper provides a short introduction to several languages that have been developed for the specification of communication protocols and services. The approach of [Hof] emphasizes the incremental view of the specification process, which means to decide while specifying whether or not the current automaton is consistent. This approach is based on an interpretation of the input, output, and state sets as partially ordered sets with certain restrictions.

2 Problem Formulation

2.1 Basic Definitions

It is assumed that the designer has to specify the function of a model \( M \) (see Figure 1) whose inputs and outputs are known and/or determined by the environment.

![Figure 1: Model \( M \)](image)

The vectors \( x \) and \( y \)

\[
x = (x_1, x_2, \ldots, x_n)^T, x \in I \tag{1}
\]

\[
y = (y_1, y_2, \ldots, y_m)^T, y \in O \tag{2}
\]

are elements of any input set \( I \) and any output set \( O \) at a certain time. The input waveform \( X \)

\[
X = [x_{t1}, x_{t2}, \ldots, x_{tk}] \tag{3}
\]
is defined as array of several input vectors at different times \( t_1, t_2, \ldots, t_k \). It is assumed that these times are equidistant:

\[
t_{i+1} - t_i = \Delta t, \quad i \in 1 \ldots k - 1.
\]

The equidistance implies the model \( \mathcal{M} \) to represent purely synchronous hardware with an additional clock signal. The clock cycle time is \( \Delta t \). The output waveform \( \mathbf{Y} \) is defined, accordingly. A pair of input and output waveforms is called property \( \varphi_i \),

\[
\varphi_i = (\mathbf{X}_i, \mathbf{Y}_i)
\]

which corresponds to a point in the property space \( \Phi \). Figure 2 shows the function \( \mathcal{F} \) in the property space \( \Phi \) reflecting the functionality of the model \( \mathcal{M} \) subject to specification\(^1\).

\begin{figure}[h]
\centering
\includegraphics[width=0.4\textwidth]{property_space.png}
\caption{Property Space \( \Phi \)}
\end{figure}

A specification is the collection of several properties \( \varphi_i \). A specification \( \text{spec}_l \) of level \( l \) is defined as:

\[
\text{spec}_l = [\varphi_1, \varphi_2, \ldots, \varphi_l]
\]

It describes \( l \) points in property space \( \Phi \). Incremental specification is the transition

\[
\text{spec}_l \Rightarrow \text{spec}_{l+1}.
\]

If some \( l \) different properties are given, the specification can be refined through adding a property \( \varphi_{l+1} \) to the model. At any stage, there is a partial specification which becomes the final specification as soon as the designer decides that the set of properties is sufficiently large.

A finite automaton is defined to be the quintuple

\[
\mathcal{A} = (I, O, S, \sigma, \omega).
\]

where \( I \) is the input set, \( O \) is the output set, and \( S \) is the finite state set. The state transition function \( \sigma : I \times S \rightarrow S \) produces for the current state \( s \) and the given input \( i \) the next state, and the output function \( \omega : I \times S \rightarrow O \) computes the output for the same situation.

2.2 Partial Specifications

Given a partial specification \( \text{spec}_l \) consisting of \( l \) points \( \varphi_i, i \in \{1 \ldots l\} \) (i.e., input \( \mathbf{X}_i, i \in \{1 \ldots l\} \) and output waveforms \( \mathbf{Y}_i, i \in \{1 \ldots l\} \)). We are now looking for all automata \( \mathcal{A}_j \) producing the output waveforms \( \mathbf{Y}_i \) from the input waveform \( \mathbf{X}_i, i \in \{1 \ldots l\} \) under the assumption that each property has the same start state.

\[
\mathcal{A}_j = (X_i, Y_i, S_j, \sigma_j, \omega_j) \quad j \in \{1 \ldots k\}, i \in \{1 \ldots l\}
\]

Let us assume, that we know all \( k \) state-minimal deterministic automata that satisfy the specified properties:

\[
\| S_j \| \rightarrow \text{minimal} \quad j = 1 \ldots k
\]

Then, the following situations may occur:

1. \( k = 0 \). There is no deterministic automaton which satisfies the properties. This is caused by inconsistent properties \( \varphi_i \), and the corresponding (partial) specification is called inconsistent.

   As an example, assume that \( \mathbf{X}_1 = [0', 0'] \), \( \mathbf{Y}_1 = [0', 0'] \) and \( \mathbf{X}_2 = [0', 0'] \), \( \mathbf{Y}_2 = [0', 1'] \). There is no deterministic automaton, which for the same start state can create different output sequences on two identical input sequences. Therefore this is an inconsistent specification.

2. \( k = 1 \). There exists one and only one state-minimal automaton satisfying the requirements. The corresponding (partial) specification is called unambiguous.

3. \( k > 1 \). There exist several state-minimal automata satisfying the properties \( \varphi_i \). This specification is said to be ambiguous\(^2\).

The goal is to help the designer in refining her/his specification by suggesting a further property \( \varphi_{l+1} \). The effect of an additional property \( \varphi_{l+1} \) may be the following: An additional property \( \varphi_{l+1} \) may be reproduced by some of the \( k \) finite automata. Then one can use this property to eliminate those automata which do not have the property, i.e., the property is used to reduce the degree of ambiguity. Or, the additional property \( \varphi_{l+1} \) may not be a property of any of the \( k \) finite automata since the output waveform cannot be produced. This requires to reconstruct \( k_{l+1} \) finite automata for \( \text{spec}_{l+1} \) with a larger number of states since the finite automata for \( \text{spec}_l \) were state-minimal. And finally, the additional property \( \varphi_{l+1} \) may extend the input alphabet, which of course requires to reconstruct the finite automata for \( \text{spec}_{l+1} \).

Let us now assume that the designer has created a partial and ambiguous specification \( \text{spec}_l \), i.e., \( k > 1 \). Then, at least one further property \( \varphi_{l+1} \) is needed to derive a more precise and possibly unambiguous specification \( \text{spec}_{l+1} \) from \( \text{spec}_l \). How can the designer in

\(^{1}\text{Function } \mathcal{F} : X \rightarrow Y \text{ produces the output waveforms from the input waveforms. } \mathcal{F} \text{ is a discrete function and not continuous or differentiable due to the fact that } X \text{ and } Y \text{ are discrete.}\)

\(^{2}\text{Note, that the corresponding finite automata may nevertheless be completely specified w.r.t. the current input alphabet. A finite automaton is completely specified if each state has a next-state transition on each character of the input alphabet.}\)
this situation be supported in the specification process?

Figure 3: \( \text{spec}_i \)

Figure 3 shows the partial and ambiguous specification \( \text{spec}_2 \) in the property space \( \Phi \). There are two automata \( A_1 \) and \( A_2 \) both satisfying \( \varphi_1 = (X_i, Y_i) \) and \( \varphi_2 = (X_2, Y_2) \). The goal is to successively derive an unambiguous specification through one or more additional properties. This is illustrated in Figure 4. In order to reduce the degree of ambiguity in the specification, one has to find a differentiating input waveform, i.e., an input waveform \( X_3 \) where the corresponding output waveforms \( Y_3 \) are different in the two automata \( A_1 \) and \( A_2 \). In searching for such a waveform the designer can be asked:

*If you apply the input waveform \( X_3 \) to the model, what is the desired output waveform?*

Three answers are possible:

1. The desired output waveform is \( Y_{3a} \). The resulting specification is unambiguous because \( \varphi_i = (X_i, Y_i), i = 1 \ldots 3 \) is valid for automaton \( A_1 \) only.

2. The desired output waveform is \( Y_{3b} \). The specification is unambiguous because \( \varphi_i = (X_i, Y_i), i = 1 \ldots 3 \) is valid for automaton \( A_2 \) only.

3. The desired output waveform is neither \( Y_{3a} \) nor \( Y_{3b} \). Therefore neither of the two automata is suitable to describe the desired behavior and one or more new finite automata satisfying the three properties have to be constructed.

The new specification \( \text{spec}_3 \) may be ambiguous, and the game is played again until the designer decides to stop with adding properties or until the specification is unambiguous.

3 Solution

3.1 Building the Finite Automata

Given a partial specification \( \text{spec}_i \) consisting of \( i \) properties \( \varphi_i = (X_i, Y_i) \), \( i = 1 \ldots l \):

\[
\varphi_i = (X_i, Y_i) = ([x_{1i} \ldots x_{iL(i)}], [y_{1i} \ldots y_{iL(i)}])
\]

The waveform length of property \( \varphi_i \) which corresponds to the number of time steps is denoted by \( L(i) \). Note, that the waveform lengths of different properties may differ. The vector of all input values at a certain time is denoted by \( x \); the vector of all output values by \( y \). If there are

\[
N = \sum_{i=1}^{l} L(i)
\]

input and output vectors specified, we are looking for a deterministic and state-minimal automaton producing the output waveform \( Y_i \) from the input waveform \( X_i \) such that the \( N \) different input and output waveforms correspond to transitions in the finite state machine. We assume, that the first transitions of all properties \( \varphi_i, i = 1 \ldots l \) start from the same state (initial state).

The task can be done by applying algorithms for state minimization. For an introduction the reader is referred to [UH92, PE72]. First a non-minimal finite automaton containing \( N \) states is built from the waveforms. Each input and output vector at a certain time corresponds to a state. Now, we are able to introduce the definition of an *inconsistent specification*. A specification is said to be inconsistent if there exists an output or a next-state indeterminism. Now, we have to minimize the finite automaton. This is done by applying the algorithm for marking pairs of inequivalent states ([UH92]). This results in a compatibility table. For incompletely specified machines it is known that selecting state compatibility pairs or sets for finding a minimum-state machine is an NP-complete problem. In most cases the use of maximal compatible and incompatible sets in a relative simple heuristic selection process suffices to obtain minimal realizations. We use the method given by Paull and Unger [PU89] to generate maximal compatible sets. Compatible states are merged. For incompatible sets we transform the states to a graph. Each state in the initial finite state machine corresponds to a node. There is an edge between two nodes if the nodes are incompatible. We are now able to formulate the generation of a state-minimal finite automaton from the
given properties as a graph coloring problem where each color corresponds to a state. To solve this problem one can employ well-known node coloring algorithms. Furthermore, we are now able to define an ambiguous specification. Is there a choice for coloring the graph with the same chromatic number (up to an isomorphism) then the specification is said to be ambiguous, otherwise the specification is said to be unambiguous. This coloring freedom can be exploited to guide incremental specification.

3.2 Guided Incremental Specification

3.2.1 An Example

Given the property \( \varphi_1 \):

\[
\begin{array}{|c|c|c|}
\hline
\text{input} & \text{output} \\
\hline
1 & A \quad C \\
2 & B \quad C \\
3 & A \quad D \\
4 & B \quad D \\
\hline
\end{array}
\]

Figure 5 shows the corresponding graph \( G \). It is easy to see that \( G \) can be colored with two colors (e.g., red and blue). There is, however, a freedom for coloring. Node 1 and Node 3 (Node 2 and Node 4, respectively) can exchange their colors. Hence, there are two ways for coloring \( G \) (up to renaming states) and the specification is ambiguous.

\[
\begin{array}{|c|c|c|c|c|}
\hline
\text{input} & 1 & 2 & 3 & 4 \\
\hline
\text{output} & \text{red} & \text{red} & \text{blue} & \text{blue} \\
\hline
\text{output} & \text{red} & \text{blue} & \text{blue} & \text{red} \\
\hline
\end{array}
\]

The minimized automaton has two (chromatic number) states (red, blue). Figures 6 and 7 show the two automata \( A_1, A_2 \):

The task is to differentiate these two automata through an additional property (waveform) proposed to the designer. Imagine graph \( G \): If there were an additional edge 1–2 then there are less possibilities for coloring (only one).

\[
\begin{array}{|c|c|c|c|c|}
\hline
\text{input} & 1 & 2 & 3 & 4 \\
\hline
\text{output} & \text{red} & \text{red} & \text{blue} & \text{blue} \\
\hline
\text{output} & \text{red} & \text{blue} & \text{blue} & \text{red} \\
\hline
\end{array}
\]

Initially, there were two automata, now there is only one automaton \( A_1 \). Hence, an edge 1–2 can result from the description of an additional property. Later, this edge will be called a refining edge. There must be an edge between node 1 and 2 if the input value of both nodes are equal, but the output values are different. The input value of Node 1 is \( A \). The designer is asked for the new property \( \varphi_2 \):

\[
\begin{array}{|c|c|}
\hline
\text{input} & \text{output} \\
\hline
1 & A \quad C \\
2 & A \quad ? \\
\hline
\end{array}
\]

Now there are two possibilities:

1. \( ? = D \). As a consequence an edge has to be inserted between nodes 1 and 2 in Graph \( G \). This means that there exists only one possible automaton \( A_2 \).

2. \( ? = C \). As a consequence an edge has to be inserted between nodes 2 and 3. This means that there exists only one possible automaton \( A_1 \).

The number of potential automata was reduced through an 'intelligent' question. It is important to point out that this concept is more powerful than only to build a completely specified automaton\(^3\). Although both automata \( A_1 \) and \( A_2 \) are completely specified, there is a way to support the designer in refining her/his specification.

3.2.2 Refining Edges

Now the term refining edge will be introduced. If an edge is inserted between two non-adjacent nodes of a graph, and the nodes are colored again, then this

\( ^3 \)A finite automaton is completely specified if each state has a next-state transition on each character of the input alphabet.
edge is called a **refining edge** if the chromatic number remains and the number of ways can be colored decreases thereby. For an introduction to graph theory the reader is referred to [Gou88]. Denote the chromatic number of graph $G$ by $\chi(G)$ and define a particular chromatic function $\pi_G(k)$ such that $\pi_G(k)$ is the number of ways $G$ can be $k$-colored. Also, define the **color degree** of node $P$ to be the number of colors used to color the nodes adjacent to node $P$.

If one builds a new graph $G'$ by adding an edge $e$ to graph $G$ then this edge $e$ is said to be a **refining edge** if and only if

$$\chi(G') = \chi(G) \quad \text{and} \quad \pi_G(\chi(G')) < \pi_G(\chi(G)). \quad (12)$$

The second requirement is needed to exclude the trivial case, where the additional edge does not increase the color degree of some nodes. Intuitively, one can imagine that $\pi_G$ decreases if the color degree of some nodes increases.

A specification is called an **ambiguous specification** if and only if there exists one or more refining edges. If there are no refining edges the specification is said to be **unambiguous**.

Due to the definition of refining edges the task of finding refining edges is NP-complete. A simple heuristic was developed to solve the problem in colored graphs. With the help of the following two items it may be possible to re-color a colored graph $G$ and to find refining edges thereby.

1. A node $P$ can be colored with $\chi(G) - \text{colordegree}(P)$ different colors. If

$$\chi(G) - \text{colordegree}(P) > 1$$

there is a freedom for coloring node $P$. A refining edge would restrict this freedom. There is a refining edge between nodes $P$ and $R$ if

- $P$ and $R$ are not adjacent and
- $R$ has a different color from the colors of nodes adjacent to $P$.

This means that the edge between nodes $P$ and $R$ would increase the color degree of node $P$.

2. First we need the condition for interchanging the colors of two adjacent nodes. Both colors of nodes $P$ and $Q$ are interchangeable if the color degree of both nodes $P$ and $Q$ would decrease after removing the edge between nodes $P$ and $Q$. This means that the color of each node adjacent to $P$ (except $Q$) is different from the color of node $Q$. Therefore, $P$ can be colored with color of $Q$. This holds for $Q$, respectively.

Imagine two adjacent nodes $P$ and $Q$ whose colors are interchangeable. A refining edge should restrict this freedom for coloring. There is a refining edge between node $P$ and some third node $R$ if

<table>
<thead>
<tr>
<th>ckt</th>
<th>initial</th>
<th>prop</th>
<th>questions</th>
<th>prop</th>
<th>final</th>
<th>nodes</th>
<th>edges</th>
<th>states</th>
</tr>
</thead>
<tbody>
<tr>
<td>tkt</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>29</td>
<td>25</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>cntr</td>
<td>1</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>496</td>
<td>5274</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>alu</td>
<td>32</td>
<td>2</td>
<td>994</td>
<td>1026</td>
<td>2050</td>
<td>1024</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>tlc</td>
<td>1</td>
<td>18</td>
<td>4</td>
<td>2</td>
<td>55</td>
<td>1248</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>busrc</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>11</td>
<td>22</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Results

(a) $P$ and $R$ are not adjacent,
(b) colors of node $P$ and node $Q$ are interchangeable and
(c) $color(R) = color(Q)$.

This holds for $Q$, accordingly. If you insert such a refining edge in graph $G$ then the colors of nodes $P$ and $Q$ are no longer interchangeable.

If an ambiguous partial specification $spec_i$ has been described so far, a part of an additional property $\varphi_{i+1}$ can be constructed and proposed to the designer. Assume there is a refining edge between nodes $n_{a,\tau_a}$ and $n_{b,\tau_b}$. The refining edge would become a ‘real’ edge if there is a different output value for the same input value. Therefore, one can ask the designer:

*If you are in state $n_{a,\tau_a}$ and the input value is the same as in state $n_{b,\tau_b}$, what would the output value be?*

If this output value in State $n_{a,\tau_a}$ is different from the output value in State $n_{b,\tau_b}$ then there has to be an edge between these two nodes (refining edge). As the State $n_{a,\tau_a}$ corresponds to a waveform from the initial state $s_0$ to the State $n_{a,\tau_a}$ (property $a$, $\tau_0$ time steps) the question consists of this waveform (to reach State $n_{a,\tau_a}$) and the input value of node $n_{b,\tau_b}$ (property $b$ at time $\tau_0 \Delta t$). The new property $\varphi_{i+1} = (X_{i+1}, Y_{i+1})$ is calculated in the following manner:

$$x_{i+1, i'} = x_{a, i'} \quad \text{for } i = 1 \ldots \tau_a - 1 \quad (13)$$

$$y_{i+1, i'} = y_{a, i'} \quad \text{for } i = 1 \ldots \tau_a - 1 \quad (14)$$

$$x_{i+1, \tau_a} = x_{b, \tau_b} \quad \text{for } i = 1 \ldots \tau_a - 1 \quad (15)$$

$$y_{i+1, \tau_b} = \text{???} \quad \text{for } i = 1 \ldots \tau_a - 1 \quad (16)$$

If there is more than one refining edge the edge with the smallest corresponding waveform of the new property $\varphi_{i+1}$ is chosen.

It is interesting that there are two possible new waveforms for each refining edge. The reason is that a (refining) edge is commutative w.r.t. exchanging the nodes $n_{a,\tau_a}$ and $n_{b,\tau_b}$.

4 Results

Our approach to guided incremental specification has been implemented in C++. Building the finite automata is based on the **Brelaz Color-Degree Algorithm** [Gou88]. The tool is capable of generating VHDL-code at any stage of the specification process.
The code is synthesizable since the input and output alphabets are represented in enumeration types. Table 1 summarizes the results for some examples. For each circuit the initial number of properties and states, the number of generated questions and the final number of properties, nodes, edges, and states resulting at the end of the specification process are listed. In these examples the final specifications are unambiguous.

The following conclusions can be derived from the experiment.

- The generated questions have been 'good' questions, since they efficiently led to the most 'interesting' waveforms to specify.

- When the RESET function was specified, a lot of new properties were calculated. This shows the need to support more abstract properties than those we have been looking at in this paper. The designer would like to have the possibility to say If that signal has this value at any time then do something.

- It is clear, that an unacceptable amount of properties was needed when specifying a model such as an ALU with only a few states and a lot of combinatorics. That is the reason why this approach of incremental specification is well suited for controller designs but not for combinatorics design.

Generally, the designer gets a deeper insight in the functionality during an early stage of specification, the key motivation for our approach to guided incremental specification.

Throughout the paper we introduced our concept by using certain assumptions: We assumed to be able to reach a state-minimal automaton. For an incompletely specified automaton, however, finding a state-minimal one is known to be NP-complete. Incompletely specified automata are the typical case in the incremental specification process. It may therefore happen, that specifications seem to be ambiguous that are not. Since the designer controls when to stop the specification process, this problem does not constitute a blocking point to the approach. Furthermore, we assume that we know all k state-minimal automata or in other words that we are able to find 'true' refining edges. Since heuristics are used, we do not know whether a refining edge really differentiates among the state-minimal automata. The experimental results, however, show that even within large examples 'good' questions are asked by the tool. From the experiments we can conclude that although no exact algorithms were used for coloring and searching for refining edges there was no loss in quality visible w.r.t. the generated questions.

5 Conclusion

We presented a novel approach to supporting incremental specification. We discussed how finite state machines are generated from the description of input and output behavior and how the freedom can be exploited for guiding an incremental specification process.

Future work is directed towards the extension of the algorithm to allow to describe RESET and don’t cares and to employ graphical interface specifications [Kho93]. In the long term the approach will be extended to combine incremental specification and formal verification methods.

Acknowledgment

The authors would like to thank Michael Mrva for his helpful comments on this paper.

References


