Specification mining for asynchronous controllers

Javier de San Pedro
Universitat Politecnica de Catalunya
Barcelona, Spain

Thomas Bourgeat
Massachusetts Institute of Technology
Cambridge, Massachusetts

Jordi Cortadella
Universitat Politecnica de Catalunya
Barcelona, Spain

***Abstract*—The paper presents a first effort at exploring a novel area in the domain of asynchronous controllers: specification mining. Rather than synthesizing circuits from specifications, we aim at doing reverse engineering, i.e., discovering safe specifications from the circuits that preserve a set of pre-defined behavioral properties (e.g., hazard freeness). The specifications are discovered without any previous knowledge of the behavior of the circuit environment. This area may open new opportunities for re-synthesis and verification of asynchronous controllers. The effectiveness of the proposed approach is demonstrated by mining concurrent specifications (Signal Transition Graphs) from multiple implementations of 4-phase handshake controllers and some controllers with choice.**

I. Introduction

The design automation efforts in the area of asynchronous circuits have been mostly focused on two problems: synthesis and formal verification. The synthesis problem consists of obtaining a circuit from a specification, e.g., a gate netlist from a Signal Transition Graph (STG). The verification problem consists of checking the conformance of a circuit with regard to a specification.

In this paper we study a new problem for asynchronous circuits: *Specification Mining*. The problem consists of discovering formal specifications from implementations [1]. Specification mining is becoming popular in the software engineering community as a machine-learning approach to infer properties from the observable behavior of the systems [15]. One example is the work presented in [12] in which safe and permissive interfaces (sequences of library calls) are synthesized for software systems in such a way that the interfaces do not violate the internal invariants of the system. Another example is [16] where specifications represented as Message Sequence Graphs are synthesized from the traces observed from the execution of a concurrent system. In [17], a method is presented which automatically infers high-level descriptions from circuits, representing them as a combination of instances of abstract functional blocks from a predetermined library.

This paper tackles the problem of discovering safe interfaces for asynchronous controllers without any previous knowledge of their original specifications. The problem can be illustrated using the example in Fig. 1(a). The circuit implements a handshake controller in which only the initial state is known (all handshake signals at 0, RST=1). The reset signal (RST) is assumed to be silent during the normal operation of the circuit and the cross-coupled NOR gates are assumed to have an atomic behavior (negligible internal delay). We pose the following challenge:

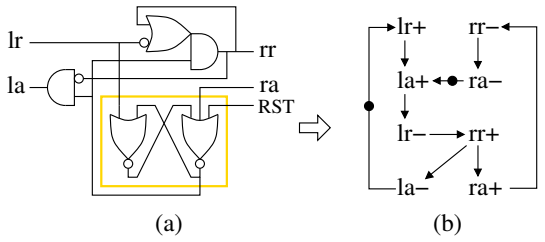


Fig. 1. Specification mining of a handshake controller.

> *Can we discover a specification of the interface that guarantees a speed-independent behavior of the circuit?*

The answer to this question is not unique. Several interfaces could exercise the circuit without producing any hazard. In particular, an empty interface (no events) would guarantee such behavior. Our interest is to find maximally concurrent interfaces that honor the desired properties of the circuit.

Fig. 1(b) shows one possible safe interface. This interface has been discovered automatically by the approach proposed in this paper and coincides with the L440R2044 4-phase controller (according to the nomenclature in [4]).

Specification mining can define not only properties that must be preserved in the circuit, but also properties of the interface. For example, it is possible to enforce that the interface is choice-free, i.e., no conflicts in the environment.

Specification mining opens a new research direction in the area of asynchronous controllers that could potentially have applications in different domains, e.g.,

- Reverse engineering, to discover the behavior of some intricate controllers for which no specification is known.
- Re-synthesis of asynchronous controllers, since the discovered interfaces can be used as specifications for synthesis tools that can produce higher-quality solutions and substitute the existing ones [13].
- Compositional verification, by substituting some components of a large circuit by the mined specifications. In this way, an assume/guarantee scheme could be applied to verify the circuit by using the mined interfaces while hiding the internal signals of the components [6], [19].

This work has been partially supported by funds from the Spanish Ministry for Economy and Competitiveness and the European Union (FEDER funds) under grant TIN2013-46181-C2-1-R and the Generalitat de Catalunya (2014 SGR 1034 and FI-DGR 2015).

*2016 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes,creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works. DOI 10.1109/ASYNC.*2016.10

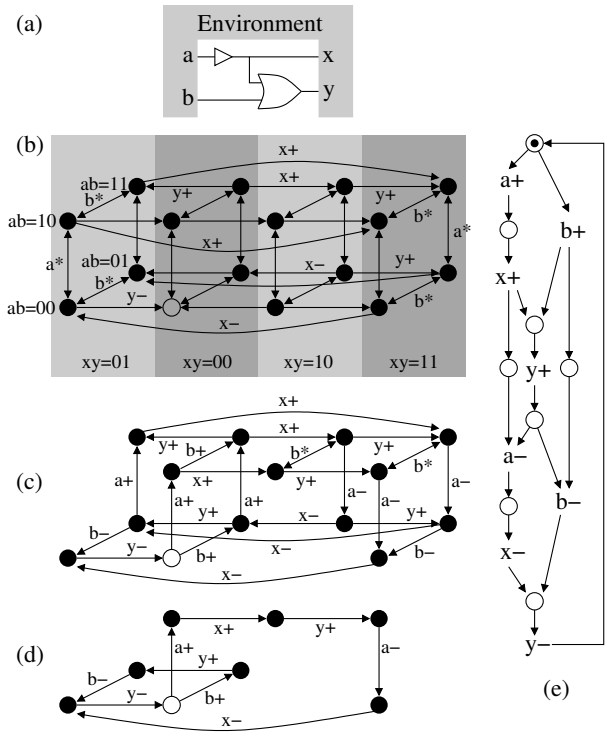


Fig. 2. Simple circuit for specification mining.

The main goal of the paper is to demonstrate that specification mining is feasible for a variety of controllers. The application of this paradigm to specific problems in asynchronous design and verification is out of the scope of this work.

II. Overview

This section gives an informal overview of the approach proposed for specification mining, using the example shown in Fig 2. The goal is to obtain a specification for the environment of the circuit shown in the Fig. 2(a) in such a way that certain behavioral properties are guaranteed.

In this particular case, we would like the circuit to be speed-independent (SI) and have a delay-insensitive (DI) interface. Speed-independence is guaranteed when the circuit is *output persistent* (only input signals can disable each other). A circuit has a DI interface if its behavior does not depend on the arrival order of the inputs.

The labeled transition system (LTS) shown in Fig. 2(b) shows all possible behaviors of the circuit under a free environment, i.e., all input signals can switch at any time instant. Every state corresponds to a binary vector that represents the value of the signals at that state. We will identify each state by its binary vector $\langle abxy \rangle$.

The labels x^+ and x^- represent rising and falling transitions of signal x. The double arcs $\overset{a^*}{\longleftrightarrow}$ represent alternating a^+ and a^- transitions between a pair of states and are used to model the switching of input signals in a free environment. The transitions of x and y are depicted in the horizontal direction, whereas the transitions of a and b are depicted in the vertical and diagonal directions, respectively. For the sake of clarity, not all the labels are shown in the picture, although they can be easily deduced from the depicted information.

We will assume that the initial (reset) state is also known. In the example, the initial state is $\langle 0000 \rangle$ (unfilled circle).

A free environment leads to many circuit malfunctions (hazards). For example, transition $\langle 0010 \rangle \xrightarrow{x^-} \langle 0000 \rangle$ produces a violation of output persistence that may be manifested as a glitch in signal y, since y^+ is enabled in $\langle 0010 \rangle$ and disabled in $\langle 0000 \rangle$.

The goal of specification mining is to *discover* one or several specifications for the environment that:

- have good properties, e.g., guarantee a hazard-free behavior of the circuit, and
- are general enough to cover a large set of behaviors.

As an example, the cyclic behavior $(b^+y^+b^-y^-)^*$ is hazard free. However, we might be unsatisfied by the fact that signals a and x are not exercised.

Fig. 2(c) depicts an LTS with output persistence, i.e., no output transition can be disabled by another transition. The largest LTS fulfilling this property can be uniquely obtained from the one in Fig. 2(b) by deleting the transitions that produce violations of output persistence.

Still, Fig. 2(c) does not model a DI interface[1]. For example, the transitions a^- and b^+ are enabled in state $\langle 1011 \rangle$. The arrival of a^- (leading to $\langle 0011 \rangle$) disables b^+, whereas the arrival of b^+ (leading to $\langle 1111 \rangle$) does not disable a^-.

Unfortunately, there is no unique solution when trying to find a subset of the LTS that fulfills the DI interfacing conditions. Given the fact that the circuit cannot be modified, the only chances are reduced to constraining the environment by deleting some input transitions. In the previous example, a DI interface can be obtained in different ways, e.g., by removing either transition a^- or b^+ from state $\langle 1011 \rangle$. Other deletions are also required in other parts of the LTS to guarantee a complete DI interface.

Fig. 2(d) depicts an LTS that models an SI circuit with a DI interface. An STG modeling the same behavior is shown in Fig. 2(e). Obtaining this specification is the most challenging problem tackled in this paper. We will show how the problem can be modeled with a Boolean formula and solved using SAT or Integer Linear Programming.

The main goal of this paper is to propose a methodology to obtain specifications from a circuit that fulfill a set of properties defined a priori.

III. Background

A. Labeled Transition Systems and Circuits

A Labeled Transition System is a structure (S, L, T, s_0) where S is a finite set of states, L is a finite alphabet of labels, T is a subset of $S \times L \times S$ and $s_0 \in S$ is the initial state. We will also denote by $s_i \xrightarrow{a} s_j$ the transition (s_i, a, s_j). A circuit C is a structure $C = (X, G, s_0)$ where :

[1]DI interfacing will be formally defined in Section III-E.

- $X = I \cup O \cup Z$ is the set of signals, with I, O and Z being pairwise disjoint sets that represent the input, output and internal signals of the circuit, respectively.
- $G : (O \cup Z) \rightarrow f(X)$ is a set of gates that assigns a Boolean function to each non-input signal of the circuit. We denote by $f_{x_i}(X)$ the Boolean function assigned to signal x_i.
- s_0 is a binary vector representing the value of the signals at the initial state.

Given a circuit $C = (X, G, s_0)$, we define LTS$(C) = (S, X, T, s_0)$ as the LTS associated to C and generated by a free environment. Formally:

- $S = \{0, 1\}^n$, where $n = |X|$.
- $T = T_{env} \cup T_g$, where T_{env} and T_g are the transitions produced by the environment and the circuit, respectively (defined later).

Note that the alphabet of labels of LTS(C) is the set of signals of the circuit and the initial state coincides with the initial state of the circuit. When convenient, we will distinguish by x^+ and x^- the rising and falling transitions of signal x.

S is the set binary vectors representing all possible states of the signals. Given a state $s = (x_1, \ldots, x_n)$, we denote by $s(x_i)$ the value of x_i in s. Given a state $s = (x_1, \ldots, x_i, \ldots, x_n)$, we denote by $s^{\neg x_i} = (x_1, \ldots, \neg x_i, \ldots, x_n)$ the state in which the values of the signals are identical to the ones of s except for x_i, that has the complementary value. Note that if $s_1 \xrightarrow{x} s_2 \in T$, then $s_2 = s_1^{\neg x}$ and $s_1 = s_2^{\neg x}$.

The set of transitions T_{env} in a free-environment circuit are defined as follows:

$$T_{env} = \{(s, x, s^{\neg x}) \mid s \in S \wedge x \in I\},$$

representing the fact that any input signal can switch at any state. The set of transitions T_g produced by the circuit is defined as follows:

$$T_g = \{(s, x, s^{\neg x}) \mid s \in S \wedge x \in (O \cup Z) \wedge s(x) \neq f_x(s)\},$$

representing all transitions of non-input signals produced by the logic gates when the value at the output of the gate is different from the function computed by the gate.

It is important to realize that the LTS associated to a circuit is always deterministic since every transition exiting a state can only lead to a unique state.

B. Circuits with constrained environment

The main purpose of this paper is to find a specification of the environment of the circuit that fulfills certain properties. In other words, finding a subset of transitions of T_{env} that prevent the circuit from reaching states in which the desired properties are violated.

Given a circuit C, its free-environment LTS$(C) = (S, X, T, s_0)$ and a subset of transitions $E \subseteq T_{env}$ representing a constrained environment, we denote by LTS(C, E) the LTS obtained from LTS(C) after deleting the transitions in $T_{env} \setminus E$. Formally, LTS$(C, E) = (S', X, T', s_0)$ is the maximal LTS such that:

- $S' \subseteq S$ is the subset of reachable states from s_0.
- T' is the maximal set of reachable transitions from s_0 such that $T' \subseteq (E \cup T_g)$.

LTS(C, E) can be computed from LTS(C) by deleting the transitions in $T_{env} \setminus E$ and iteratively deleting unreachable states and transitions until a greatest fixed point is reached.

C. Properties of an LTS

Let LTS$(C, E) = (S, X, T, s_0)$ be the LTS associated to a circuit C with environment E. We say that x is *enabled* in state s if $s \xrightarrow{x} s^{\neg x} \in T$. We say that x *disables* y in state s if $s \xrightarrow{x} s^{\neg x} \in T$, y is enabled in s and not enabled in $s^{\neg x}$. Finally, we say that x *triggers* y in state s if $s^{\neg x} \xrightarrow{x} s \in T$, y is not enabled in $s^{\neg x}$ and enabled in s.

Definition. In an LTS a signal x is *persistent* if no signal $y \neq x$ disables it. If a signal x disables another signal y in any state s, then there is a *conflict* between x and y.

Example. Let us consider the LTS in Fig. 2(b), where the initial state is $s_0 = \langle 0000 \rangle$ (unfilled circle). Let us consider the state $s_1 = \langle 1000 \rangle$ and the transition $s_0 \xrightarrow{a^+} s_1$. We can say that a^+ triggers x^+ in s_1, since x^+ is not enabled in s_0 but it is in s_1. Let us now consider the transition $s_1 \xrightarrow{a^-} s_0$. We can see that a^- disables x^+ in s_1 since x^+ is enabled in s_1 but not in s_0. Notice how Fig. 2(c) and (d) show LTSs where both x and y are persistent signals while a, b are in conflict.

D. Speed-independence

In this paper, we deal with circuits with unbounded gate delays, i.e., any gate of the circuit can switch at any time as long as it is enabled. A circuit whose behavior does not depend on the delay of its gates is called *speed-independent*.

Proposition [9]. Given a circuit C and an environment E, C under E is speed-independent iif in LTS(C, E) all pairs of signals x, y are in conflict only if both x, y are input signals.

This property implies every non-input signal is persistent.

E. Delay insensitive interfacing

Another desired property is that the behavior of the circuit is insensitive to the arrival order of the input transitions. This property is called *delay-insensitive (DI) interfacing* and is formally defined as follows.

Proposition [20]. The LTS associated to a circuit satisfies the DI interfacing conditions if no input transition triggers another input transition.

Rather than dealing with pure delay insensitivity, DI interfacing assumes that wire delays can be kept under control within the circuit and only tolerance to delay variability at the interface is required.

F. Multi-environment interfaces

In some scenarios for re-synthesis and compositional verification, a system may have been split into different components (circuits). From the point of view of the circuit of interest, the surrounding components can be considered as a set of

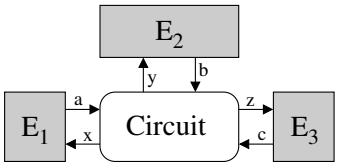


Fig. 3. Multi-environment interfacing.

TABLE I
ALLOWED RELATIONS IN A CIRCUIT WITH MULTIPLE ENVIRONMENTS

x	y	x triggers y	x disables y
Input	Input	Violates DI	*Only if* $x, y \in E_i$
Output	Output	Allowed	Violates SI
Input	Output	Allowed	Violates SI
Output	Input	*Only if* $x, y \in E_i$	Violates SI

independent environments, $E_1 \dots E_n$, that interact with the circuit, as shown in the example of Fig. 3.

When mining specifications for a circuit, we might want to consider multi-environmental scenarios where independence between different environments is to be preserved. Informally, this means that, given two environments E_i and E_j, signals from E_i cannot directly trigger or disable inputs from E_j. Such a causality relation would imply a connection between E_i, E_j outside of the circuit of interest, making E_i and E_j dependent from each other. However, an input from E_i may excite an output in a different environment E_j, via the circuit.

Definition. Let us consider the LTS (S, X, T, s_0) associated to a circuit. Let the set of signals of the circuit be

$$X = X_1 \cup \ldots \cup X_n \cup Z$$

where Z is the set of internal signals and $\{X_1, \ldots, X_n\}$ is a partition of the set of input/output signals $(I \cup O)$, with each X_i corresponding to a different environment. The LTS preserves the multi-environment interface for partition $\{X_1, \ldots, X_n\}$ if:

$$\forall a \in X_i, b \in X_j \ \cap \ I, \ i \neq j : a \text{ cannot trigger or disable } b.$$

In the example of Fig. 3, the preservation of the multi-environment interface would not allow $\{x, a\}$ to trigger or disable $\{b, c\}$, $\{y, b\}$ to trigger/disable $\{a, c\}$ and $\{z, c\}$ to trigger/disable $\{a, b\}$.

Note that, by this definition, a circuit where each environment has one input only, i.e. $\forall E_i$, $|E_i \cap I| \leq 1$, any LTS preserving the multi-environment interface would be input-persistent, as no signal would be allowed to disable an input.

Table I summarizes the previous properties, showing the allowed causality relations between two different signals x, y depending on the type (input or output) of each signal. For example, x cannot trigger y if both x and y are inputs, since that would violate the delay-insensitive interfacing property. However, x may disable y, but only if both x, y are in the same environment E_i.

IV. SPECIFICATION MINING

This section describes the main contribution of this work: the process used to mine a specification from a circuit C, while guaranteeing a set of properties for both the circuit and the interface. The process works by starting from a free environment E, and then constraining this environment until both the environment and the circuit under such environment satisfy all properties.

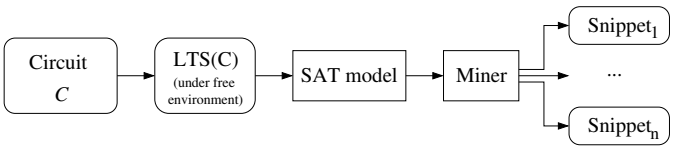


Fig. 4. Overview of the specification mining flow.

Note, however, that for certain properties there may be more than one specification satisfying all the properties. The environments in each specification may only exercise a small subset of the full circuit behavior. In these situations, our flow will discover each of these specifications, which we call *snippets*. The original specification of the circuit will be contained in one of these snippets. Our mining flow will give priority to the most general snippet, containing the environment exercising most behavior from the circuit. An example of this will be discussed in Section VI.

A summary of the proposed mining flow can be seen in Fig. 4. The first step constructs LTS(C) containing all the behaviors of the circuit under a free environment, starting from the circuit netlist. For the circuit in Fig. 2, this would correspond to the LTS in (b).

The desired properties for the mined specifications are then specified into a set of constraints on top of this LTS. In this section, we will specify these constraints as satisfiability (SAT) formulae. Every truth assignment of the formula represents a valid environment under which circuit C satisfies all the desired properties, and from which a snippet may be synthesized.

The most interesting snippets can be then synthesized into different types of specifications, such as Signal Transition Graphs (STGs). For most circuits, only the snippet containing the most general behavior will be of interest. However, the secondary snippets may provide an insight into alternative behaviors of the system.

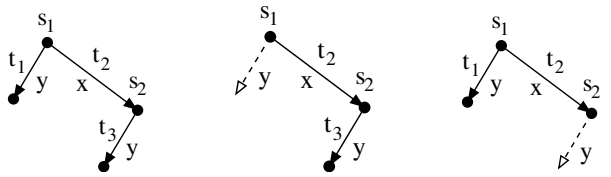
The following sections describe this flow in more detail, including examples that show how the most common circuit properties are modeled.

A. Satisfiability model for behavioral properties

Many of the most interesting circuit properties imply constraints on the causality/concurrency/choice relations between events of the LTS. Table I shows the causality constraints to guarantee speed independence, delay insensitive interface and multi-environment properties.

In this section, we show how different circuit and environment properties can be mapped into constraints between different signals, and how these constraints can be implemented on a SAT model.

Let LTS$(C) = (S, X, T, s_0)$ be the LTS associated to a circuit C constructed using the method defined in the previous section. The SAT model extracts a subset of LTS(C),



(a) Persistence of y. (b) x triggers y. (c) x disables y.

Fig. 5. Causality patterns in LTS.

LTS(C, E), satisfying the required properties. In our formulation, for every transition $t_i \in T$, we define a variable with the same name indicating whether t_i is selected, i.e. whether $t_i \in$ LTS(C, E). Definitions of the most typical constraints are as follows:

1) x cannot trigger y: For every pair of states $s_1, s_2 \in S$, with a transition $t_2 = s_1 \xrightarrow{x} s_2$, x triggers y if y is enabled in s_2 but not in s_1. i.e., when $t_3 = s_2 \xrightarrow{y}$ exists but $t_1 = s_1 \xrightarrow{y}$ does not. This pattern is illustrated in Fig. 5(b).

To guarantee this property, if both t_2 and t_3 are selected, then t_1 must exist and be selected:

$$t_2 \wedge t_3 \implies t_1.$$

In case t_1 does not exist in LTS(C), then the previous constraint must be rewritten accordingly, forbidding selection of both t_2 and t_3:

$$\neg(t_2 \wedge t_3).$$

These constraints may be used for example to enforce the delay-insensitive interfacing property when applied to pairs of inputs signals, as shown in Table I.

2) x cannot disable y (persistence of y): For every pair of states $s_1, s_2 \in S$, with a transition $t_2 = s_1 \xrightarrow{x} s_2$, x disables y if y is enabled in s_1 but not in s_2. This is similar to the trigger definition above, except that the roles of t_1 and t_3 are reversed: x disables y when $t_1 = s_1 \xrightarrow{y}$ exists but $t_3 = s_2 \xrightarrow{y}$ does not, as seen in Fig. 5(c).

Thus, to satisfy the property, selecting t_1 and t_2 implies t_3 must exist and be selected:

$$t_1 \wedge t_2 \implies t_3.$$

Similar to the previous constraint, this constraint can be simplified if any of t_1, t_2, t_3 is not present in LTS(C).

3) Preservation of non-input signals: The SAT model searches for a subset LTS(C, E) representing the behavior of the circuit under a constrained environment E. For this reason, the model should only remove transitions of input signals, and potentially all transitions from unreachable states under environment E.

However, it must always select all non-input transitions from a reachable state. Thus, for every state s, the following constraint must be added, which forces the selection of non-input transitions if any incoming transition is selected:

$$\forall s \in S: \quad \bigvee_{t_i = s_1 \xrightarrow{x} s \in T} t_i \implies \bigwedge_{\substack{t_j = s \xrightarrow{y} s_2 \in T \\ y \notin I}} t_j.$$

4) Strong connectedness: This property ensures that the initial state is reachable from every other reachable state. There are several known methods to require conectedness with SAT or ILP models [7]. To identify the initial state, we assume that the values of the output signals are known at reset time, and that it is stable, i.e. no output transitions are enabled.

5) Additional constraints: Many controllers impose additional properties on the environment that can be modeled as constraints between different inputs. For example, in Section VII we show a circuit with a mutual exclusion requirement between two different input signals, i.e. the two signals cannot be enabled simultaneously.

These properties can be enforced by removing states from LTS(C). For example, guaranteeing mutual exclusion between two input signals x, y is equivalent to removing every state s where $s(x) \vee s(y)$. In the proposed formulation, this can be achieved by prohibiting the selection of any t_i incident to s.

B. Algorithm for specification mining

Algorithm 1 Extracting LTS snippets

Input: Circuit C and a set of desirable properties P
Output: LTS$_1, \ldots,$ LTS$_n$ under which C satisfies P
$L \leftarrow$ LTS(C) ▷ construct full LTS from C
$R \leftarrow T($LTS$(C))$ ▷ transitions not yet in any snippet
$i \leftarrow 1$
while $|R| > 0$ **do**
  LTS$_i \leftarrow$ SOLVE(LTS(C), P, maximize $|t_i \in R|$)
    ▷ extract subset of LTS(C) satisfying P
  if LTS$_i = \emptyset$ **break**
  $R \leftarrow R \setminus T(LTS_i)$ ▷ subtract from remaining transitions
  $i \leftarrow i + 1$
return LTS$_1, \ldots,$ LTS$_n$

Algorithm 1 describes the procedure to mine specifications using the SAT model described in the previous section. At the start of the procedure, the complete LTS(C) is built. The algorithm iterates, generating a new snippet on each cycle, until all transitions from LTS(C) appear on at least one snippet or it is impossible to create new ones without violating P. To account for the former, R contains all transitions not yet included in any LTS$_1, \ldots,$ LTS$_i$.

Procedure SOLVE uses the SAT model to find the subset of LTS(C) satisfying P that contains the largest subset of transitions from R. Thus, every iteration discovers a snippet containing the largest behavior from C not yet covered in any previous snippet. Different strategies may be used to solve the SAT model with the cost function, such as MaxSAT or ILP.

V. PROPERTIES OF THE SPECIFICATION MODEL

In the previous section we have shown a method to mine snippets in which both the circuit and the environment satisfy a set of properties. These snippets are provided in the form of LTSs. However, it is often desirable to use more succinct representations, such as Signal Transition Graphs (STGs). An STG may be obtained from an LTS using Petri net synthesis tools such as petrify [8].

This section shows that, by adding some constraints during the mining process, properties of the specification model can be enforced. For example, structural Petri net properties, such as marked graphs or free-choiceness, can be modeled in this way. These extra properties may contribute to enhance the visualization and analysis of the specification models.

This section is focused on structural properties of Petri nets, generating two different types of STGs: marked graphs and free choice.

A. Marked Graphs

A marked graph is a Petri net in which all places have exactly one predecessor and one successor transition [18].

Forward and backwards persistence are necessary conditions for a strongly-connected LTS to model the space state of a marked graph [3]. Thus, to obtain a marked graph, it is necessary to extend the constraints of the mining flow to prevent conflicts between all pairs of signals. Backwards persistence can be guaranteed using a similar set of constraints.

B. Free-choiceness

A Petri net is free choice is for any two transitions x and y that share a predecessor place p, then x and y have only one predecessor [18]. While there is a choice in p between x and y, we say the choice is *free* because on any marking where x can be fired, y can be fired too, and vice versa.

In a LTS, this is equivalent to guaranteeing that if x, y are in conflict, then x must be enabled in all the states where y is enabled, and y must be enabled in all states where x is. We model this by introducing a new Boolean variable, $\text{choice}_{x,y}$, which indicates whether the snippet contains a conflict between x and y, and a new set of constraints which relate these variables to the relationship between x and y.

The first set of constraints removes all conflicts between x, y (identical to the constraint described in section IV-A2), unless $\text{choice}_{x,y}$ is asserted:

$$(t_1 \wedge t_2 \implies t_3) \vee \text{choice}_{x,y}.$$

In addition, for every state s where either x or y is enabled, a constraint is added forcing both to be enabled or disabled if $\text{choice}_{x,y}$ is asserted. With $t_1 = s \xrightarrow{x} s_1$ and $t_2 = s \xrightarrow{y} s_2$:

$$\text{choice}_{x,y} \implies t_1 = t_2.$$

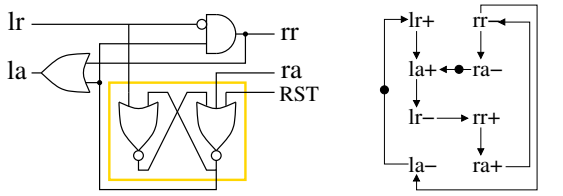
As in previous constraints, the formula is simplified appropriately if there is no t_1 or t_2 in s. For example, if there is a state s where t_1 exists but t_2 does not, the constraint becomes:

$$\text{choice}_{x,y} \implies \neg t_1.$$

VI. Case study: mining specifications for 4-phase latch controllers

The 4-phase latch controller is at the core of the data paths of many asynchronous designs. A 4-phase latch controller is composed of 4 handshake signals controlling 2 channels: left (*lr*, *la*) and right (*rr*, *ra*), as shown in the example of Fig. 1.

In [4], the design space of 4-phase controllers is studied. While these designs all have the same external interface, they vary on the level of concurrency allowed by the protocol. Each variation cuts away certain states of the controller.



(a) Circuit (b) Mined specification

Fig. 6. Example of circuit and mined specification for *L440oR2264*.

Every variation is given a name depending on the number of states that are removed. *L000oR0000* is the version with all states, and thus, the most concurrent of all variations. The circuit in Fig. 1 represents *L440oR2044*, which removes 18 states, and results in a more constrained protocol. *L440oR2264* removes an additional 4 states, resulting in a slightly simpler circuit also shown in Fig. 6

In this case study, we will focus on the 137 controllers presented in [4] which are speed independent and deadlock-free. In the experiment we will synthesize a circuit for each one of these controllers, and then rediscover the specifications from each one using the proposed mining flow.

A. Environment setup

The input to our mining flow is a netlist. To generate circuits for each of the 137 controllers, we used petrify [8] to synthesize gate netlists from the specifications. After generating the LTS with free environment from the circuit, we transformed the SAT models into ILP, and used Gurobi [11] to mine the most general specification for each controller.

We configured our mining flow to ensure the following circuit and environment properties:

- Speed independence and delay-insensitive interfacing.
- Strong connectedness.
- Multi-environment interface to ensure the independence between the left and right channels. This will be further discussed in Section VI-C.

No other information was given to the miner.

B. Results

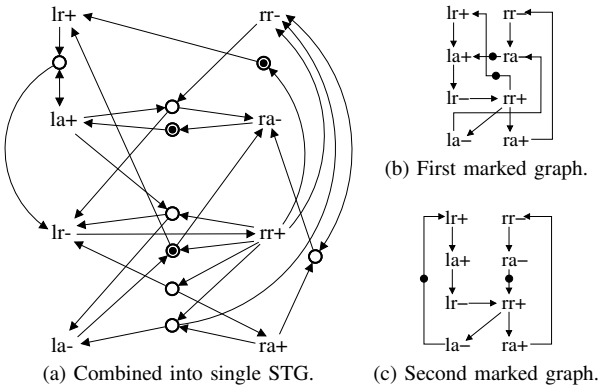
The 137 specifications were mined from the circuits in 177 seconds (Intel Core i5-2520M). Each run of the ILP model took less than 1 second on average, with the rest of time spent in generating the model and preparing the environment.

The size of the ILP model is $O(|T|^2)$ where T is the set of transitions in the LTS of the circuit. However, our implementation performs a preprocessing in which many redundant constraints are removed before generating the ILP model. The most concurrent circuit, with 256 states, also resulted in the largest model, with 267 variables and 996 constraints.

For each one of the 137 circuits, the first snippet obtained from the mining flow was always bisimilar to the original specification of the controller.

TABLE II
CIRCUIT IMPLEMENTATIONS ALLOWING ADDITIONAL BEHAVIOR AFTER REMOVING CONSTRAINTS

	Snippets	Circuits
Identical behavior	1	25
Additional behavior	1	59
	2	48
	3	5



(a) Combined into single STG. (b) First marked graph. (c) Second marked graph.

Fig. 7. Mined specification for L440oR2044.

C. Relaxing constraints to discover additional behavior

We also experimented our mining flow by discovering additional behaviors when relaxing some of the environmental constraints imposed in previous section. In particular, we allowed the left and right environments to be dependent from each other. In practice this means that the output of one channel can trigger the input of the other channel (i.e., *la* can trigger *ra* and *rr* can trigger *lr*). However, we still preserved the speed-independence and delay-insensitive interfacing properties.

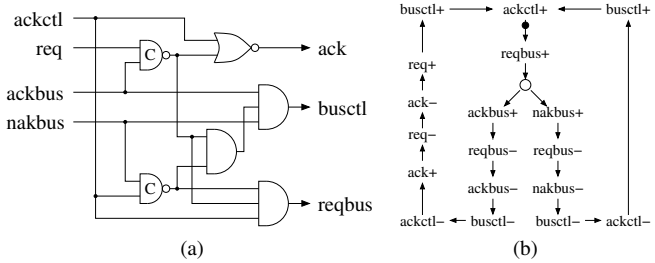
With this reduced set of constraints, our tool discovered more general specifications for 113 out the 137 controllers. All the specifications still include the original specifications. In addition, out of the 113 specifications with additional behavior, 53 required a minimum of two snippets. That is, no single snippet was able to model the entire behavior of these 53 snippets without violating DI or SI. Table II shows the total numbers of circuits that exhibited additional behavior and/or required more than one snippet.

An example of a protocol where additional behavior is discovered is *L440oR2044*, whose original specification and circuit are shown in Fig. 1. An STG showing the additional behavior is represented in Fig. 7(a). To aid legibility, our mining flow was configured to enforce the marked graph property described in section V-A, which divides this specification into the two snippets shown in Fig. 7(b) and (c).

Notice that the snippet in Fig. 7(b), however, assumes an environment where the left and right channels are not independent. For example, output $rr+$ triggers input $lr+$, which is on a different channel. Thus, the multi-environment constraints used in the last section would allow only the behavior described by snippet (c). This snippet is bisimilar to the original specification of *L440oR2044*.

TABLE III
SUMMARY OF FREE-CHOICE RESULTS

Benchmark	Num. of snippets	Original spec. included	ILP runtime [sec.]
SM-latch [14]	1	Snippet #1	0.10
RLM [5]	4	Snippet #4	0.14
1-bit variable [2]	1	Snippet #1	0.31
alloc-outbound [9]	3	Snippet #1	0.73
vmebus [9]	4	Snippet #1	0.12
A/D converter ctrl. [9]	1	Snippet #1	0.43
tsend-csm [10]	–	–	$>$ 1 h.



(a) (b)

Fig. 8. Circuit and mined specification for *alloc-outbound*.

VII. MINING CONTROLLERS WITH CHOICE

This section shows the results of applying the mining flow to a selection of asynchronous controllers from well-known benchmarks. In these examples, we introduce environments with input conflicts, i.e. inputs may disable other inputs.

As in the previous case study, the properties of SI and DI interfacing are enforced. When possible, we also enforce multi-environment interfacing as well as any required mutual exclusion between pairs of input signals.

Table III reports the total number of snippets discovered by our mining flow, as well as whether the original specification of the circuit was included in one of the discovered snippets. The rest of this section delves into the details of some of the test cases with interesting properties.

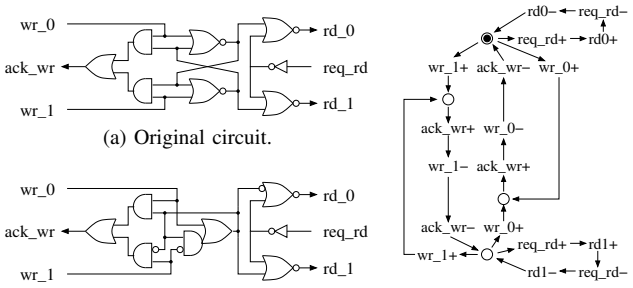
A. alloc-outbound

alloc-outbound is part of a set of well-known academic benchmarks [9], representing part of an HP bus controller. The circuit used as input for the mining flow is shown in Fig. 8. The interface is composed of three different environments: 1) $reqbus, ackbus, nakbus$, 2) $busctl, ackctl$, 3) ack, req. Notice only environment 1 has more than one input, with only signals $ackbus$ and $nakbus$ allowed to be in conflict.

Without this constraint, the number of possible SI and DI snippets grows to 12. The ILP runtime also rises up to 1 hour, showing the effectiveness of the multi-environment constraint in restricting the size of the state space. With this constraint, there are only 3 valid snippets, with the original specification being the first discovered snippet, shown in Fig. 8(b).

B. 1-bit variable

In this example we use a simple implementation of a 1-bit variable with a single read and write port [2]. The original circuit is shown in Fig. 9(a). The write port includes the input signals *wr_0*, *wr_1* as well as the write acknowledgment signal



(a) Original circuit.

(c) Resynthesized circuit.

(b) Mined specification.

Fig. 9. Circuit and mined specification for a 1-bit variable.

ack_wr. The read port includes *req_rd* as well as the response signals *rd_0*, *rd_1*.

Of significance is that the original circuit, in Fig. 9(a), may go into metastability if both *wr_0* and *wr_1* are asserted. Our mining flow, thus, discovers environments in which *wr_0* and *wr_1* are mutually exclusive.

Hazards may also be produced when simultaneous read and write requests are asserted. In this particular implementation, hazards only occur when the read value is different from the one being written (e.g., read a 1 while writing a 0). Hazards are not produced when both values are the same. The mined specification (not shown in the paper) accepts concurrent read/write requests of the same value.

Yet, because of higher-level environmental conditions, it may be desirable to enforce the mined behavior to have mutually exclusive inputs, i.e.,:

$$wr\_0 + wr\_1 + req\_rd \leq 1$$

When configured to honor this property, the mining flow generates the specification shown in Fig. 9(b).

Figure 9(c) shows an alternative implementation of the circuit obtained by synthesizing the mined specification. Interestingly, this implementation has no metastability problems when both *wr_0* and *wr_1* are asserted, although glitches may be observed when such situation occurs. This feature, however, is irrelevant for a well-behaved environment that would never allow both inputs to be asserted simultaneously.

C. Negative results

Not all the experimental results are as attractive as the ones presented in previous sections. One of the major challenges of specification mining is to deal with state explosion. The runtime of the ILP models grows exponentially with the number of states present in LTS(C).

As seen with *alloc-outbound*, constraining the environment is an effective approach to handle state spaces that are initially too large to explore. However, some designs are not amenable to this type of constraints. For example, in *tsend-csm*, separate environments cannot be assumed. Our approach explores the full state space, resulting in large ILP models.

To obtain a reasonable runtime in these situations, further environment constraints are necessary. This is an area for further research. For larger circuits, divide-and-conquer approaches, as in compositional verification scenarios [6], [19], may be necessary.

VIII. Conclusions

The intricate structure of asynchronous controllers makes their design error-prone. Discovering safe specifications contributes to understanding the implicit protocols behind them and their properties.

This paper has presented a novel approach for behavior discovery that can offer useful mechanisms for re-synthesis and verification. As new challenges for the future we envision two directions: applying specification mining to compositional verification and mining specifications with bounded delays.

References

[1] G. Ammons, R. Bodík, and J. R. Larus. Mining specifications. In *29th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL)*, pages 4–16, 2002.

[2] K. v. Berkel. *Handshake Circuits: an Asynchronous Architecture for VLSI Programming*, volume 5 of *International Series on Parallel Computation*. Cambridge University Press, 1993.

[3] E. Best and R. Devillers. Characterisation of the state spaces of live and bounded marked graph Petri nets. In *Language and Automata Theory and Appl.*, volume 8370 of *LNCS*, pages 161–172. Springer, 2014.

[4] G. Birtwistle and K. Stevens. Modelling mixed 4phase pipelines: Structures and patterns. In *20th IEEE Int. Symp. on Asynchronous Circuits and Systems (ASYNC)*, pages 27–36, 2014.

[5] T.-A. Chu. *Synthesis of Self-Timed VLSI Circuits from Graph-Theoretic Specifications*. PhD thesis, MIT, June 1987.

[6] J. M. Cobleigh, D. Giannakopoulou, and C. S. Păsăreanu. Learning assumptions for compositional verification. In *Proc. of the 9th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'03, pages 331–346. Springer-Verlag, 2003.

[7] N. Cohen. Several graph problems and their Linear Program formulations. Technical report, INRIA, July 2010.

[8] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Transactions on Information and Systems*, E80-D(3):315–325, Mar. 1997.

[9] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. *Logic Synthesis of Asynchronous Controllers and Interfaces*. Springer, 2002.

[10] R. M. Fuhrer and S. M. Nowick. *Sequential Optimization of Asynchronous and Synchronous Finite-State Machines: Algorithms and Tools*. Kluwer Academic Publishers, 2001.

[11] Gurobi Inc. *Gurobi Optimizer Reference Manual*. 2015.

[12] T. A. Henzinger, R. Jhala, and R. Majumdar. Permissive interfaces. *SIGSOFT Softw. Eng. Notes*, 30(5):31–40, Sept. 2005.

[13] T. Kolks, S. Vercauteren, and B. Lin. Control resynthesis for control-dominated asynchronous designs. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, Mar. 1996.

[14] A. Kondratyev, M. Kishinevsky, A. Taubin, and S. Ten. Analysis of Petri nets by ordering relations in reduced unfoldings. *Formal Methods in System Design*, 12(1):5–38, Jan. 1998.

[15] I. Krka, Y. Brun, and N. Medvidovic. Automatic mining of specifications from invocation traces and method invariants. In *22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 178–189, 2014.

[16] S. Kumar, S.-C. Khoo, A. Roychoudhury, and D. Lo. Mining message sequence graphs. In *33rd International Conference on Software Engineering (ICSE)*, pages 91–100, 2011.

[17] W. Li, Z. Wasson, and S. Seshia. Reverse engineering circuits using behavioral pattern mining. In *IEEE Int. Symp. on Hardware-Oriented Security and Trust (HOST)*, pages 83–88, June 2012.

[18] T. Murata. Petri Nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, Apr. 1989.

[19] O. Roig, J. Cortadella, and E. Pastor. Hierarchical gate-level verification of speed-independent circuits. In *Asynchronous Design Methodologies*, pages 129–137. IEEE Computer Society Press, May 1995.

[20] H. Saito, A. Kondratyev, J. Cortadella, L. Lavagno, and A. Yakovlev. What is the cost of delay insensitivity? In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 316–323, Nov. 1999.