Sound idle and block equations for finite state machines in xMAS

Technische Universiteit Eindhoven
Department of Mathematics and Computer Science

Sound idle and block equations for finite state machines in xMAS

Alexander Fedotov, Jeroen J.A. Keiren and Julien Schmaltz

19/04

ISSN 0926-4515

editor: prof.dr.ir. J.J. van Wijk

Computer Science Reports 19-04
Eindhoven, November 2019

Sound idle and block equations for finite state machines in xMAS

Alexander Fedotov Jeroen J.A. Keiren
Department of Mathematics and Computer Science, Eindhoven University of Technology, The Netherlands

Julien Schmaltz
ICT Group

{A.Fedotov, J.J.A.Keiren}@tue.nl, julien.schmaltz@ict.nl

Abstract

The xMAS language allows the high-level modeling of communication fabrics. For micro-architectural models expressed in xMAS, it was shown that liveness can be proven effectively using a reduction to SAT. Verbeek *et al.* extended xMAS with finite state machines (FSMs) to model and verify liveness of the combination of cache coherence protocols and interconnects. To support state machines, they extended the existing reductions to SAT to incorporate FSMs. We present counterexamples showing that this technique is unsound and fails to detect certain deadlocks. We propose an alternative reduction of liveness for xMAS networks with FSMs to SAT. We prove the correctness of our approach and evaluate its performance.

1 Introduction

Formal verification has been successfully introduced in many design flows of hardware and software systems. More and more often, the sign-off decision for hardware blocks is taken solely on the results of formal proofs, the so-called *formal sign-off*. Scaling formal verification to the system level remains a challenge.

The xMAS language [7] and associated techniques for invariant generation [5], property checking [5], and deadlock hunting [12, 16] have been proposed to address this challenge. These techniques are very efficient and were extended to performance validation [15], asynchronous circuits [3], progress verification [8], generalized to language families [17], and directly related to the Register Transfer Level [13, 14, 10].

Initially focused on the analysis of communication fabrics, Verbeek *et al.* [18, 19] introduced state machines into xMAS. The state machine extension allows the modeling and analysis of complex cooperating state machines under the constraints imposed by micro-architectural choices. In particular, they demonstrated the verification of large systems consisting of nodes running cache coherence protocols and communicating via a Network-on-Chip. The work by Verbeek *et al.* aims to scale verification to the system level by translating liveness verification of xMAS extended with (finite) state machines to satisfiability. Sadly, as we will show in this paper, their method fails to detect some deadlocks, and is therefore *unsound*.

Contributions. We present an example of an xMAS network with an FSM that has a dead input channel, and demonstrate that the approach from [19] fails to detect this dead channel. This demonstrats that the approach is unsound. To address the issue, we propose an alternative transformation of liveness verification of xMAS networks with FSMs into a satisfiability problem. Similar to the work from [19], we extend the idle and block equations from [12]. We prove that our extension to idle and block equations is sound, i.e., if the xMAS network has a path to a state with a local deadlock, then there exists a satisfying assignment to the satisfiability problem we generate. We recall the invariants from [18], that are used to restrict the number of false deadlocks detected by our approach. Finally, we use a set of benchmarks to demonstrate that our approach is efficient.

Structure of the paper. In the following sections, we introduce the relevant part of the xMAS language. We introduce xMAS networks, the definition of liveness of channels and idle and block equations in Section 2. We recall Verbeek *et al.*'s extension of xMAS with automata and present a counterexample in Section 3. In Section 4 we present our xMAS finite state machines (FSMs). The idle and block equations for FSMs and their soundness are described in Section 5. Section 6 adapts the invariants from [19] to our FSMs. Our implementation is evaluated in Section 7. We conclude in Section 8.

2 Preliminaries

In this section we introduce the syntax and semantics of xMAS, specify liveness of channels, and reiterate how liveness can be transformed into a safety problem using idle and block equations.

2.1 xMAS syntax

xMAS [7] is a graphical language aimed at modeling and verifying communication fabrics. An xMAS network comprises a number of primitives connected by typed channels. The core xMAS primitives are provided in Figure 1.

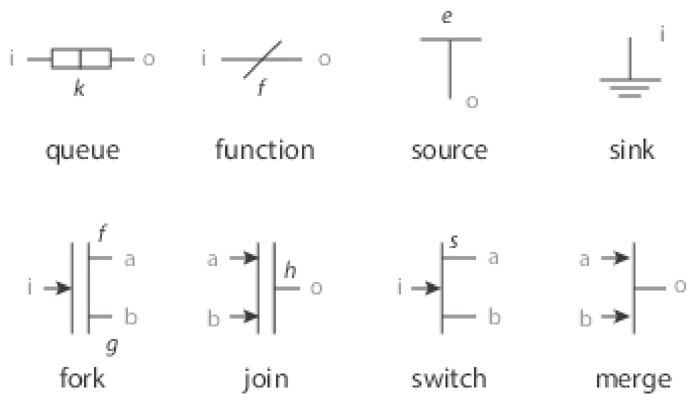


Figure 1: Core xMAS primitives

A queue is a FIFO buffer with k places. A function transforms messages using a specified function f. Sources and sinks inject and consume messages. Sources and sinks are assumed to be fair, namely, they always eventually inject or consume messages. A fork duplicates the message at its input to its two outputs. The duplication occurs if and only if the two outputs are ready to accept a copy of the input message. The join is the dual of the fork. The output of the join depends on function h applied to the data at the two inputs. Typically, one of the input channels is identified as a data input and the other input channel is called a token input, and the data input is simply forwarded to the output. A switch routes messages depending on their content and a switching function s. A merge is a fair arbiter passing input messages to its output.

The progress of messages between two primitives is controlled by a simple handshake protocol. Each channel consists of three signals, one for data and two boolean control signals called **irdy** and **trdy**. Consider the transfer of data between two primitives called A (the initiator) and B (the target) via channel x. We say that A is ready to transfer data through x.**data** if x.**irdy** is true. We say that B is ready to accept the data if x.**trdy** is true. The data transfer happens if and only $x.\mathbf{irdy} \wedge x.\mathbf{trdy}$.

Formally, for instance, the function primitive is defined in terms of its input port i, output port o and function f as follows:

$$o.\textbf{irdy} := i.\textbf{irdy} \qquad i.\textbf{trdy} := o.\textbf{trdy} \qquad o.\textbf{data} := f(i.\textbf{data})$$

We here see that the function primitive is a purely combinatorial component that applies a function to whatever data is available on its input, provided the initiator of the input and the target of the output are ready.

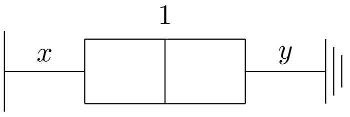


Figure 2: xMAS example.

Example 1. Consider the xMAS network in Figure 2. We use this network as a running example. The network consists a source, a queue, and a sink. The source produces tokens t. Channel x is the output channel of the source, hence:

$$x.\textbf{irdy} := \textbf{oracle} \vee \textbf{pre}(x.\textbf{irdy} \wedge \neg x.\textbf{trdy}) \qquad x.\textbf{data} := t,$$

where **pre** is the standard synchronous operator that returns the value of its argument in the previous clock cycle, and *false* in the very first cycle. Non-determinism of the data generation of the source is represented by the unconstrained primary input **oracle** [7]. Channel x is the input channel of the queue, for which we have:

$$x.\textbf{trdy} := \neg\textbf{is\_full}.$$

This signifies that a queue can always accept a data transfer when it is not full. Channel y is the output channel of the queue, for which we have:

$$y.\textbf{irdy} := \neg\textbf{is\_empty} \qquad y.\textbf{data} := \textbf{head},$$

where **head** refers to the data at the head of the queue. The queue is ready to transfer data whenever it is not empty. The same channel y is the input channel of the sink, and we have:

$$y.\textbf{trdy} := \textbf{oracle} \vee \textbf{pre}(y.\textbf{trdy} \wedge \neg y.\textbf{irdy}).$$

This definition is analogous to that of $x.\textbf{irdy}$.

2.2 Semantics of xMAS networks

All components in an xMAS network are stateless, except for the queues. The semantics of an xMAS network consists of two parts. First, a combinatorial part, that updates the values of the **irdy**, **trdy** and **data** signals of the channels, that ultimately determines the state changes that are enabled. Second, a sequential part that performs the synchronous update of the state of all components. We here introduce that semantics in more detail. For a more elaborate exposition the reader is referred to [20].

Definition 1 (xMAS network)**.** An xMAS network N is a tuple (P, G), where

- P is a set of xMAS primitives
- G is a set of channels

such that every channel $x \in G$ connects exactly one output port of a primitive to an input port of a primitive, and all ports of the primitives are connected to exactly one channel.

Given a channel $g \in G$, by $C(g)$ we denote the set of *colors* that can be transferred through g. By C we denote the set of all colors of the given network.

We assume that an xMAS network is syntactically correct, and that it does not contain combinatorial cycles of **irdy** and **trdy** signals. This can be statically checked [11]. Furthermore, when the input of a channel is ready, we can be sure the data that is on the channel is in the set $C(x)$. Formally, for all channels $x \in G$, $x.\mathbf{irdy} \implies x.\mathbf{data} \in C(x)$.

We describe the semantics of an xMAS network as a state transition diagram. The network state is the composition of the local states of all primitives. For queues, their state is determined by the content of the queue. All stateless primitives have a single state. Given a primitive $p \in P$, we denote the set of local states of p by S^p. The global state in the network is the product of the states of the individual primitives, i.e., $S = \prod_{p \in P} S^p$.

Every primitive has a transition relation from state to state that reads from a set of input channels and writes to a set of output channels. For primitive $p \in P$, states $s, s' \in S^p$, inputs $x_1(d_1), \ldots x_n(d_n)$, and outputs $y_1(e_1), \ldots y_m(e_m)$ we write $s \xrightarrow{\{x_1(d_1),\ldots,x_n(d_n)\}/\{y_1(e_1),\ldots,y_m(e_m)\}} s'$ for the transition from state s to s' while reading $d_1, \ldots, d_n$ from channels $x_1, \ldots x_n$, and writing $e_1, \ldots e_n$ to channels $y_1, \ldots, y_m$. We abbreviate this using $s \xrightarrow{\vec{x}(\vec{d})/\vec{y}(\vec{e})} s'$. Whether such a transition is enabled is controlled by the values **irdy**, **trdy** and **data** of each of the channels involved. We write *enabled*$(s \xrightarrow{\vec{x}(\vec{d})/\vec{y}(\vec{e})} s')$ if the transition is enabled. In case a primitive allows for non-determinism, a scheduler resolves this non-determinism, and selects one enabled transition. We write *selected*$(s \xrightarrow{\vec{x}(\vec{d})/\vec{y}(\vec{e})} s')$ if the transition is selected.

Note that the scheduler ensures that in every local state, exactly one transition is selected. Also, progress is forced, so, if from a given (local) state a transition is enabled, it is guaranteed that one of the enabled transitions will be taken.

Given a global state $(s^0, \ldots, s^n)$, the global transition relation from this state is now defined using the following operational rule:

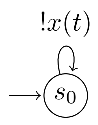
$$\frac{\begin{array}{c} s^0 = s'^0 \vee s^0 \xrightarrow{\{x_1^0(d_1^0),\ldots,x_{k_0}^0(d_{k_0}^0))\}/\{y_1^0(e_1^0),\ldots y_{m_1}^0(e_{m_1}^0)\}} s'^0 \\ \vdots \\ s^n = s'^n \vee s^n \xrightarrow{\{x_1^n(d_1^n),\ldots,x_{k_n}^n(d_{k_n}^n))\}/\{y_1^n(e_1^n),\ldots y_{m_1}^n(e_{m_n}^n)\}} s'^n \end{array}}{(s^0, \ldots, s^n) \xrightarrow{X} (s'^0, \ldots, s'^n)}$$

where $X = \bigcup_{0 \leq i \leq n} \bigcup_{1 \leq j \leq n_i} x_j^i(d_j^i)$, $X \neq \emptyset$.

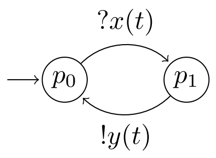
This transition relation is too large since it allows components not to take a transition, even when a transition is enabled. We therefore prune the transition relation by removing all transitions $(s^0, \ldots, s^n) \xrightarrow{X} (s'^0, \ldots, s'^n)$ for which there exists a transition $(s^0, \ldots, s^n) \xrightarrow{X'} (s''^0, \ldots, s''^n)$ such that $X \subset X'$.

Note that the semantics implies that, whenever there is no primitive with a selected transition, the state transition system is in a global deadlock.

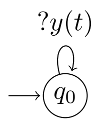
Example 2. Recall the example from Figure 1. First we consider the semantics of the individual components in the network. The source writes tokens t to channel x, which is visualized in Figure 3a. The queue needs to keep track of its state. The semantics of the queue is depicted in Figure 3b. Since we only have tokens t, and the size of the queue is 1, it is enough for the queue to remember whether it is full or empty. When the queue is empty (state p_0), it can read from x and go to the full state (p_1). When it is full, it can write its content to channel y and return to the initial state. The sink can repeatedly read from its input channel y, which is reflected in Figure 3c. The semantics of the whole network, synchronizing communication on channels x and y is depicted in Figure 3d.



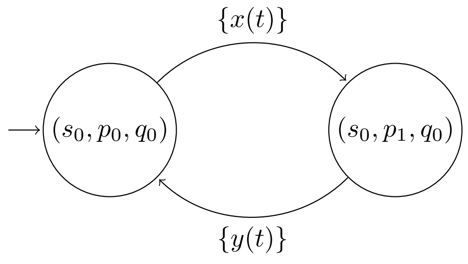
(a) Source.



(b) Queue.



(c) Sink.



(d) Semantics of the network.

Figure 3: State machines for the semantics of the network in Figure 1.

2.3 Paths

Liveness of channels is defined using linear temporal logic (LTL). LTL and its semantics are considered standard, and we refer to text books such as [2] for the details. The semantics of LTL quantifies over all paths. To interpret LTL on xMAS networks, we first define paths in such networks.

Definition 2 (Path)**.** A *path* is a possibly infinite sequence of global states $\pi = \vec{s}_0, \vec{s}_1, \vec{s}_2, \ldots$, such that for all $j > 0$, $\vec{s}_{j-1} \xrightarrow{X} \vec{s}_j$ for some X. We use $\pi[j]$ to denote the state at position j in the path, i.e., $\vec{s}_j$, and $\pi[i..]$ to denote the suffix of π starting at $\vec{s}_i$. The set of paths starting in a state $\vec{s}$ is denoted using $\mathsf{Paths}(\vec{s})$, and for an xMAS network N we write $\mathsf{Paths}(N)$ to denote $\mathsf{Paths}(\vec{s}_0)$, where $\vec{s}_0$ is the initial state of the network N. For finite paths $\pi = \vec{s}_0, \ldots, \vec{s}_n$ we define $\mathsf{last}(\pi) = \vec{s}_n$.

Further we introduce the notion of maximal path.

Definition 3 (Maximal path)**.** Given a path π, we say that π is maximal if and only if it is infinite, or it is finite, and $\mathsf{last}(\pi)$ has no outgoing transitions.

2.4 Liveness of channels

In xMAS, a channel is *live* whenever, always when its initiator is ready to transfer data, transfer will eventually be successful, meaning that both initiator and target are ready. This is formalized using the following LTL property.

Definition 4 (Live channel [12][1])**.** Consider an xMAS network $N = (P, G)$. Channel $x \in G$ is live if and only if

$$N \models \mathsf{G}(x.\mathbf{irdy} \implies \mathsf{F}(x.\mathbf{irdy} \wedge x.\mathbf{trdy})).$$

Furthermore, channel x is live for value d if and only if

$$N \models \mathsf{G}((x.\mathbf{irdy} \wedge x.\mathbf{data} = d) \implies \mathsf{F}(x.\mathbf{irdy} \wedge x.\mathbf{trdy} \wedge x.\mathbf{data} = d)).$$

[1]Gotmanov et al. [12] use property $\mathsf{G}(x.\mathbf{irdy} \implies \mathsf{F}\ x.\mathbf{trdy})$, which does not guarantee that the transfer eventually succeeds if persistency is not assumed.

Channels in xMAS networks are typically assumed to be (forward) persistent.

Definition 5 (Forward persistency [12])**.** Consider an xMAS network $N = (P, G)$. Channel $x \in G$ is forward persistent if and only if for all $d \in C(x)$

$$N \models \mathsf{G}((x.\mathbf{irdy} \wedge x.\mathbf{data} = d \wedge \neg x.\mathbf{trdy}) \implies \mathsf{X}(x.\mathbf{irdy} \wedge x.\mathbf{data} = d)).$$

In what follows, we assume channels to be forward persistent. Note that, when assuming forward persistency, both notions of live channels introduced previously are closely related.

Lemma 1. *For all xMAS networks $N = (P, G)$, and all channels $x \in G$, if x is forward persistent, then*

$$N \models \mathsf{G}(x.\mathbf{irdy} \implies \mathsf{F}(x.\mathbf{irdy} \wedge x.\mathbf{trdy})).$$

if and only if for all $d \in C(x)$

$$N \models \mathsf{G}((x.\mathbf{irdy} \wedge x.\mathbf{data} = d) \implies \mathsf{F}(x.\mathbf{irdy} \wedge x.\mathbf{trdy} \wedge x.\mathbf{data} = d)).$$

Proof. Fix xMAS network N and channel x, such that x is forward persistent. We prove both directions separately.

- $\Rightarrow$ Assume $N \models \mathsf{G}(x.\mathbf{irdy} \implies \mathsf{F}(x.\mathbf{irdy} \wedge x.\mathbf{trdy}))$. Now, consider an arbitrary path π in N. To prove: $\pi \models \mathsf{G}((x.\mathbf{irdy} \wedge x.\mathbf{data} = d) \implies \mathsf{F}(x.\mathbf{irdy} \wedge x.\mathbf{trdy} \wedge x.\mathbf{data} = d))$

  Fix arbitrary $i \geq 0$, and assume $\pi[i..] \models x.\mathbf{irdy} \wedge x.\mathbf{data} = d$. According to the assumption, $\exists j \geq i$ such that $\pi[j..] \models x.\mathbf{irdy} \wedge x.\mathbf{trdy}$. Let j be the smallest such index. Since π is persistent, for all k such that $i \leq j \leq k$, $\pi[k] \models x.\mathbf{irdy} \wedge x.\mathbf{data} = d$, so $\pi[j..] \models x.\mathbf{irdy} \wedge x.\mathbf{data} = d \wedge x.\mathbf{trdy}$, hence $\pi \models \mathsf{G}((x.\mathbf{irdy} \wedge x.\mathbf{data} = d) \implies \mathsf{F}(x.\mathbf{irdy} \wedge x.\mathbf{trdy} \wedge x.\mathbf{data} = d))$.

- $\Leftarrow$ Assume for all $d \in C(x)$, $N \models \mathsf{G}((x.\mathbf{irdy} \wedge x.\mathbf{data} = d) \implies \mathsf{F}(x.\mathbf{irdy} \wedge x.\mathbf{trdy} \wedge x.\mathbf{data} = d))$. Consider an arbitrary path π in N. To prove: $\pi \models \mathsf{G}(x.\mathbf{irdy} \implies \mathsf{F}(x.\mathbf{irdy} \wedge x.\mathbf{trdy}))$

  Fix arbitrary $i \geq 0$ and assume $\pi[i..] \models x.\mathbf{irdy}$. According to the semantics of xMAS, there exists $d \in C(x)$ such that $\pi[i..] \models x.\mathbf{data} = d$. Hence, $\pi[i..] \models x.\mathbf{irdy} \wedge x.\mathbf{data} = d$. According to the assumption, then $\pi[i..] \models \mathsf{F}(x.\mathbf{irdy} \wedge x.\mathbf{trdy} \wedge x.\mathbf{data} = d)$, hence $\pi[i..] \models F(x.\mathbf{irdy} \wedge x.\mathbf{trdy})$, so $\pi \models \mathsf{G}(x.\mathbf{irdy} \implies \mathsf{F}(x.\mathbf{irdy} \wedge x.\mathbf{trdy}))$. □

Using the assumption of forward persistency, we can now simplify the definition of a live channel. This is formalized in the following theorem, which is an adaptation of a similar, but weaker theorem in [12].

Theorem 1. *For all xMAS networks $N = (P, G)$, and all channels $x \in G$, if x is persistent, then*

$$N \models \mathsf{G}((x.\mathbf{irdy} \wedge x.\mathbf{data} = d) \implies \mathsf{F}(x.\mathbf{irdy} \wedge x.\mathbf{data} = d \wedge x.\mathbf{trdy}))$$

if and only if

$$N \models \mathsf{FG}(\neg x.\mathbf{irdy} \vee x.\mathbf{data} \neq d) \vee \mathsf{GF} x.\mathbf{trdy}.$$

Proof. Let $N = (P, G)$ be an arbitrary xMAS network, and $x \in G$ an arbitrary channel, such that x is persistent, i.e., for all $d \in C(x)$, $N \models \mathsf{G}((x.\mathbf{irdy} \wedge x.\mathbf{data} = d \wedge \neg x.\mathbf{trdy}) \implies \mathsf{X}(x.\mathbf{irdy} \wedge x.\mathbf{data} = d))$.

We prove both directions separately.

- $\Rightarrow$ Suppose $N \models \mathsf{G}((x.\mathbf{irdy} \wedge x.\mathbf{data} = d) \implies \mathsf{F}(x.\mathbf{irdy} \wedge x.\mathbf{data} = d \wedge x.\mathbf{trdy}))$, and let π be an arbitrary path in N. We need to show that $\pi \models \mathsf{FG}(\neg x.\mathbf{irdy} \vee x.\mathbf{data} \neq d) \vee \mathsf{GF} x.\mathbf{trdy}$, from which the result immediately follows.

Assume that $\pi \not\models \mathsf{FG}(\neg x.\mathbf{irdy} \lor x.\mathbf{data} \neq d)$, i.e., $\pi \models \mathsf{GF}(x.\mathbf{irdy} \land x.\mathbf{data} = d)$. We show that $\pi \models \mathsf{GF}x.\mathbf{trdy}$, i.e., $\forall i \geq 0.\exists j \geq i.\pi[j\ldots] \models x.\mathbf{trdy}$. Let $i \geq 0$ be arbitrary. Since $\pi \models \mathsf{GF}(x.\mathbf{irdy} \land x.\mathbf{data} = d)$, there exists $i' \geq i$ such that $\pi[i'\ldots] \models x.\mathbf{irdy} \land x.\mathbf{data} = d$. Let i' be such. Since $\pi[i'\ldots] \models (x.\mathbf{irdy} \land x.\mathbf{data} = d) \implies \mathsf{F}(x.\mathbf{irdy} \land x.\mathbf{data} = d \land x.\mathbf{trdy})$ according to our assumption, we have $\pi[i'\ldots] \models \mathsf{F}(x.\mathbf{irdy} \land x.\mathbf{data} = d \land x.\mathbf{trdy})$, hence there exists $j \geq i'$ such that $\pi[j\ldots] \models x.\mathbf{irdy} \land x.\mathbf{data} = d \land x.\mathbf{trdy}$. Since we have chosen i arbitrarily, we find $\pi \models \mathsf{GF}x.\mathbf{trdy}$.

$\Leftarrow$ Suppose $N \models \mathsf{FG}(\neg x.\mathbf{irdy} \lor x.\mathbf{data} \neq d) \lor \mathsf{GF}x.\mathbf{trdy}$. Let π be an arbitrary path in N. We show that $\pi \models \mathsf{G}((x.\mathbf{irdy} \land x.\mathbf{data} = d) \implies \mathsf{F}(x.\mathbf{irdy} \land x.\mathbf{data} = d \land x.\mathbf{trdy}))$.

We distinguish cases based on the property that holds in π.

- $\pi \models \mathsf{FG}(\neg x.\mathbf{irdy} \lor x.\mathbf{data} = d)$. We know there exists $k \geq 0$ such that for all $j \geq k$, $\pi[j\ldots] \not\models (x.\mathbf{irdy} \land x.\mathbf{data} = d)$. Let k be such.

  We show that for all $i \geq 0$, $\pi[i\ldots] \models (x.\mathbf{irdy} \land x.\mathbf{data} = d) \implies \mathsf{F}(x.\mathbf{irdy} \land x.\mathbf{data} = d \land x.\mathbf{trdy})$. For the case $i \geq k$ this follows immediately, since $x.\mathbf{irdy} \land x.\mathbf{data} = d$ does not hold. So, suppose $0 \leq i < k$, and assume $x.\mathbf{irdy} \land x.\mathbf{data} = d$. Towards a contradiction, suppose that $\pi[i\ldots] \models \mathsf{G}\neg(x.\mathbf{irdy} \land x.\mathbf{data} = d \land x.\mathbf{trdy})$. Since we have $\pi \models \mathsf{G}((x.\mathbf{irdy} \land x.\mathbf{data} = d \land \neg x.\mathbf{trdy}) \implies \mathsf{X}(x.\mathbf{irdy} \land x.\mathbf{data} = d)$ due to forward persistency, we have that for all $j \geq i$, $\pi[j] \models x.\mathbf{irdy} \land x.data = d$, in particular, this contradicts the fact that $\pi[k\ldots] \not\models x.\mathbf{irdy} \land x.\mathbf{data} = d$, hence $\pi \models \mathsf{G}((x.\mathbf{irdy} \land x.\mathbf{data} = d) \implies \mathsf{F}(x.\mathbf{irdy} \land x.\mathbf{data} = d \land x.\mathbf{trdy}))$.
- $\pi \models \mathsf{GF}x.\mathbf{trdy}$. Let $i \geq 0$ be arbitrary, and assume $\pi[i\ldots] \models x.\mathbf{irdy} \land x.\mathbf{data} = d$. From our assumption, for some $j \geq i$ we have $\pi[j\ldots] \models x.\mathbf{trdy}$. Consider the smallest such j. We prove that $\pi[j\ldots] \models x.\mathbf{irdy} \land x.\mathbf{data} = d \land x.\mathbf{trdy}$. From forward persistency, and the fact that j is the smallest index such that $\pi[j\ldots] \models x.\mathbf{trdy} \land x.\mathbf{data} = d$, we find that for all k, $i \leq k \leq j$, $\pi[k\ldots] \models x.\mathbf{irdy} \land x.\mathbf{data} = d$, hence $\pi[j\ldots] \models x.\mathbf{irdy} \land x.\mathbf{data} = d \land x.\mathbf{trdy}$. Therefore, $\pi \models \mathsf{G}((x.\mathbf{irdy} \land x.\mathbf{data} = d) \implies \mathsf{F}(x.\mathbf{irdy} \land x.\mathbf{data} = d \land x.\mathbf{trdy}))$. □

This inspires the following simplification [12]. We say that a channel is *idle* for d if eventually the initiator will never send message d along that channel, and it is *blocked* if eventually the target will never be able to receive message d along that channel.

Definition 6 (Idle and blocked channels [12])**.** Let x be an arbitrary channel in an xMAS network, and let $d \in C(x)$. We define

$$\mathbf{idle}(x(d)) := \mathsf{FG}(\neg x.\mathbf{irdy} \lor x.\mathbf{data} \neq d)$$
$$\mathbf{block}(x) := \mathsf{FG}\neg x.\mathbf{trdy}$$

Using these definitions, and Theorem 1, we have the following for forward persistent channels in an xMAS network.

Corollary 1. *Let N be an xMAS network, with x a forward persistent channel in N. We have the following correspondences:*

1. *for all $d \in C(x)$, channel x is live for d iff $N \models \mathbf{idle}(x(d)) \lor \neg\mathbf{block}(x)$,*
2. *channel x is live iff $N \models \left(\bigwedge_{d \in C(x)} \mathbf{idle}(x(d))\right) \lor \neg\mathbf{block}(x)$.*

Proof. The proofs for path parts follow from our previous results as follows:

1. directly from Theorem 1.
2. directly from part 1 of this corollary and Lemma 1. □

A local deadlock is defined as a *dead* channel, where a channel is dead for value d if and only if it is not live for d. This means there exists a path in the xMAS network to a state that satisfies $\neg\mathbf{idle}(x(d)) \wedge \mathbf{block}(x)$. In other words, a channel is dead whenever its initiator is ready to transfer datum d and its target will never be ready to accept the data. In the rest of this article, we use the following definitions.

Definition 7 (Formulas for live and dead channels)**.** Let N be an xMAS network, with x a forward persistent channel in N, and $d \in C(x)$. We define

$$\begin{aligned}
\mathbf{live}(x(d)) &:= \mathbf{idle}(x(d)) \vee \neg\mathbf{block}(x)\\
\mathbf{dead}(x(d)) &:= \neg\mathbf{live}(x(d))\\
\mathbf{live}(x) &:= \bigwedge_{d \in C(x)} \mathbf{live}(x(d))\\
\mathbf{dead}(x) &:= \bigvee_{d \in C(x)} \mathbf{dead}(x(d))
\end{aligned}$$

This definition allows us to formally define a dead channel.

Definition 8 (Dead channel)**.** Let N be an xMAS network, with x a forward persistent channel in N, and $d \in C(x)$. Channel x is dead for d if and only for some path $\pi \in \mathsf{Paths}(N)$, $\pi \models \mathbf{dead}(x(d))$.

Observe that in the definition of dead channel we evaluate the LTL formula over a *path*, whereas for determining whether a channel is live we evaluate the corresponding LTL formula over a *network*.

In Definition 7, we only defined $\mathbf{block}(x)$. We can refine this definition by introducing $\mathbf{block}(x(d))$ as follows.

$$\mathbf{block}(x(d)) := \mathsf{FG}(\neg x.\mathbf{trdy} \vee x.\mathbf{data} \neq d)$$

It is easy to see that $\mathbf{block}(x)$ implies $\mathbf{block}(x(d))$ for any $d \in C(x)$. In the definition of $\mathbf{live}(x(d))$ we can freely replace $\mathbf{block}(x)$ by $\mathbf{block}(x(d))$ as shown by the following lemma.

Lemma 2. *Let N be an xMAS network, with x a forward persistent channel in N, and $d \in C(x)$. Then $N \models \mathbf{live}(x(d))$ if and only if $N \models \mathbf{idle}(x(d)) \vee \neg\mathbf{block}(x(d))$.*

Proof. Let N, x and d be such. We prove both directions separately.

- $\Rightarrow$ Suppose $N \models \mathbf{live}(x(d))$, hence $N \models \mathbf{idle}(x(d)) \vee \neg\mathbf{block}(x)$. Fix an arbitrary path π in N. We have to show that $\pi \models \mathbf{idle}(x(d)) \vee \neg\mathbf{block}(x(d))$. Assume $\pi \models \neg\mathbf{idle}(x(d))$. We show that $\pi \models \neg\mathbf{block}(x(d))$, i.e., for all $i \geq 0$, there is $j \geq i$ such that $\pi[j..] \models x.\mathbf{trdy} \wedge x.\mathbf{data} = d$. Fix arbitrary $i \geq 0$. Since $\pi \models \neg\mathbf{idle}(x(d))$, for some $j \geq i$, $\pi[j..] \models x.\mathbf{irdy} \wedge x.\mathbf{data} = d$. Since $\pi \models \neg\mathbf{block}(x)$, for some $k \geq j$, $\pi[k..] \models x.\mathbf{trdy}$. Consider the smallest such k, than according to forward persistency, $\pi[k..] \models x.\mathbf{irdy} \wedge x.\mathbf{data} = d$, hence $\pi[k..] \models x.\mathbf{data} = d \wedge x.\mathbf{trdy}$, so $\pi \models \neg\mathbf{block}(x(d))$.
- $\Leftarrow$ Suppose $N \models \mathbf{idle}(x(d)) \vee \neg\mathbf{block}(x(d))$. Fix an arbitrary path π in N, and assume $\pi \models \neg\mathbf{idle}(x(d))$. Then $\pi \models \neg\mathbf{block}(x(d))$, i.e., $\pi \models \mathsf{GF}(x.\mathbf{trdy} \wedge x.\mathbf{data} = d)$, then it immediately follows that $\pi \models \mathsf{GF}(x.\mathbf{trdy})$, hence $\pi \models \mathbf{block}(x)$. $\square$

As a consequence, we can freely use definitions of block that depend on a single data value.

Additionally, it follows straightforwardly that, whenever a channel x is dead for d, that channel is blocked for all values e. This is formalized by the following lemma.

Lemma 3. *Let N be an xMAS network with x a forward persistent channel in N, and $d \in C(x)$. Then for all paths $\pi \in \mathsf{Paths}(N)$, $\pi \models \mathbf{dead}(x(d))$ implies $\pi \models \bigwedge_{e \in C(x)} \mathbf{block}(x(e))$.*

Proof. Fix an xMAS network N and channel x, and let $d \in C(x)$. Let $\pi \in \mathsf{Paths}(N)$ such that $\pi \models \mathbf{dead}(x(d))$. Let $e \in C(x)$ be arbitrary. Since $\pi \models \mathbf{dead}(x(d))$, $\pi \models \neg\mathbf{idle}(x(d)) \wedge \mathbf{block}(x)$, so $\pi \models \mathbf{block}(x)$, which immediately implies $\pi \models \mathbf{block}(x(e))$. $\square$

2.5 Idle and block equations

The main contribution of Gotmanov *et al.* [12] is to express deadlock conditions for each primitive using equations over boolean variables. If these *idle and block equations* are satisfiable, a (possible) deadlock has been detected; if they are unsatisfiable, the network is guaranteed to be deadlock free. The method is sound but incomplete; if the equations are satisfiable, the assignment to the boolean variables may constitute a deadlock state that is unreachable in the network. This is alleviated to some extent by incorporating invariants that approximate the reachable states.

The boolean variables express the conditions under which a primitive will never try to output value d, denoted using variable $\mathbf{idle}_x^d$, or never try to read from channel x, denoted using variable $\mathbf{block}_x$. The encoding is such that, whenever there exists a path π in the xMAS network such that $\pi \models \mathbf{dead}(x(d))$, then there is a satisfying assignment to the variables in the idle and block equations in which $\mathbf{idle}_x^d$ is *false*, and $\mathbf{block}_x$ is *true*.

As an example, we consider the idle and block equations for the function primitive with input channel x and output channel y, for input value d. Equations for **idle** and **block** are encoded as follows:

$$\begin{aligned} \mathbf{block}_x &= \mathbf{block}_y \\ \mathbf{idle}_y^e &= \bigwedge_{d \in C(x)} \Big((f(d) = e) \implies \mathbf{idle}_x^d \Big) \end{aligned}$$

Intuitively, this means that the input channel of the function primitive is blocked if its output channel is blocked, and the output channel is idle for value e if the input channel is idle for all values that are mapped onto e by the function f.

Note that in this way, the idle and block equations essentially approximate the LTL specifications of idle and block defined before.

Example 3. We demonstrate idle and block equations in an xMAS network using the running example from Figure 2. Channel x is the output channel of the source, which can produce tokens k infinitely often, hence its output is not idle:

$$\mathbf{idle}_x := \bot.$$

Channel x is the input channel of the queue. The queue is blocked if it is full and its output is blocked:

$$\mathbf{block}_x := \mathbf{full} \wedge \mathbf{block}_x,$$

Channel y is the output channel of the queue. The output of the queue is idle, when the queue is empty, and its input is idle:

$$\mathbf{idle}_x := \mathbf{empty} \wedge \mathbf{idle}_x.$$

Finally, channel y is the input channel of the sink. The sink can consume data infinitely often, so x is not blocked:

$$\mathbf{block}_x := \bot.$$

3 Life and death of state machines in xMAS

Verbeek *et al.* describe an extension of xMAS with finite state machines for the integrated verification of, for instance, cache coherence protocols together with their underlying communication fabric [18, 19].

3.1 xMAS automata

We first recall the definition of finite state machines and the corresponding semantics as described by Verbeek *et al.*

Definition 9 ([19, Definition 1])**.** Let D denote the type of packet. An XMAS automaton is a tuple (S, T, s_0, C_I, C_O) with S the set of states, T the set of transitions, s_0 the initial state and C_I (C_O) the set of in (out) channels. A transition $t \in T$ is a tuple $(s, s', \varepsilon, \varphi)$ with s and s' the begin and end states, function $\varepsilon :: C_I \times D \mapsto \mathbb{B}$ an event and function $\phi :: C_I \times D \rightharpoonup (C_O \times D)$ a transformation.

In this definition, $\rightharpoonup$ indicates a partial function. Given transition $(s, s', \varepsilon, \varphi)$, $\varepsilon(x, d) = \mathit{True}$ indicates that the transition can be enabled by packet d at input channel x. Likewise, $\varphi(i, d)$ indicates the output packet that is produced at a specified output channel. If nothing is produced, this is denoted using $\perp$.

Before giving the semantics of xMAS automata, we first define when a transition is enabled.

Definition 10 ([19, Definition 2])**.** Let A be an xMAS automaton. Let $t = (s, s', \varepsilon, \phi)$ be a transition of A.

$$\mathit{enabled}(t, x) := A.s \wedge x.\mathbf{irdy} \wedge \varepsilon(x, x.\mathbf{data}) \wedge \mathit{rdy}(\varphi(x, x.\mathbf{data}))$$

where $A.s$ indicates A is currently in state s, $\mathit{rdy}(\perp) = \mathit{True}$, and $\mathit{rdy}(y, e) = y.\mathbf{trdy}$.

Finally, for xMAS automata, the semantics in terms of **irdy** and **trdy** signals are defined as follows.

Definition 11 ([19])**.** Let $A = (S, T, s_0, C_I, C_O)$ be an xMAS automaton. Assume sel_A is a fair selection function that chooses an enabled transition and the corresponding input channel that enables the transition. We now define:

$$\begin{aligned}
x.\mathbf{trdy} &:= \mathsf{sel}_A = (x, \_) \\
y.\mathbf{irdy} &:= \mathsf{sel}_A = (y, t) \wedge \varphi(x, x.\mathbf{data}) = (y, \_) \\
y.\mathbf{data} &:= \begin{cases} e & \text{if } \mathsf{sel}_A = (y, t) \wedge \varphi(x, x.\mathbf{data}) = (y, e) \\ \perp & \text{otherwise} \end{cases}
\end{aligned}$$

where $t = (s, s', \varepsilon, \varphi)$.

Verbeek *et al.* next construct idle and block equations for xMAS automata as follows. First, they state that an xMAS automaton is dead if "there exists a state for which all outgoing transitions can be dead". This results in the following idle and block equations for xMAS automaton $A = (S, T, s_0, C_I, C_O)$ [19]:

$$\begin{aligned}
\mathbf{deadA}_A &:= \bigvee_{s \in S} .A.s \wedge \bigwedge_{t=(s,s',\varepsilon,\varphi)\in T} \bigwedge_{x \in C_I} \bigwedge_{d \in C(x)} \varepsilon(i, d) \implies (\mathbf{block}_{\varphi(x,d)} \vee \mathbf{idle}_{x(d)}) \\
\mathbf{block}_{x(d)} &:= \mathbf{deadA}_A \vee \bigwedge_{t=(s,s',\varepsilon,\varphi)\in T} \neg\varepsilon(x, d) \\
\mathbf{idle}_{y(e)} &:= \mathbf{deadA}_A \vee \bigwedge_{t=(s,s',\varepsilon,\varphi)\in T} \bigwedge_{x \in C_I} \bigwedge_{d \in C(x)} \varepsilon(x, d) \implies \varphi(x, d) \neq (y, e)
\end{aligned}$$

The intent is for the definition to be such that, when the automaton is considered in the context of an xMAS network N, and there exists a path π in N such that $\pi \models \mathbf{dead}(x(d))$, for input channel x of the automaton, then $\mathbf{block}_{x(d)}$ is true. Likewise, if for some path π in N such that $\pi \models \mathbf{dead}(y(e))$ for output channel y, then $\mathbf{idle}_{y(e)}$ is false.

Intuitively, in the definition above, **block** is true at an input channel if and only if either all transitions in the state machine can never be taken or *the state machine has no transition reading on that channel.* Similarly, **idle** holds at an output channel if and only if either all transitions can never be taken of *the state has no transition writing on that channel.*

3.2 Life and death of state machines: a counter-example

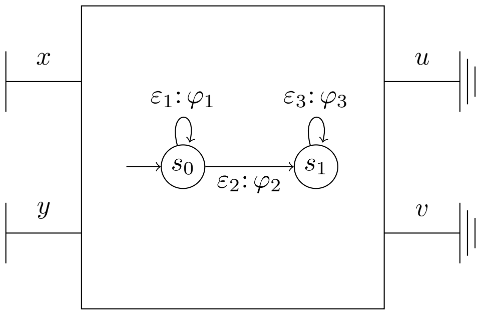
Unfortunately, there are xMAS networks with finite state machines that, according to the approach of Verbeek *et al.* are deadlock free, but that do in fact contain a deadlock. This is illustrated by the following example.

Example 4. Consider the state machine in Figure 4 with two input channels x and y, connected to sources, and two output channels u and v, connected to sinks. All channels only transfer datum d.

The functions ε and φ are defined (per transition) as follows:

$$\begin{aligned}
\varepsilon_1(i,d) &:= i = x & \qquad \varphi_1(i,d) &:= (u,d)\\
\varepsilon_2(i,d) &:= i = y & \qquad \varphi_2(i,d) &:= (v,d)\\
\varepsilon_3(i,d) &:= i = x & \qquad \varphi_3(i,d) &:= (v,d)
\end{aligned}$$

So, initially in s_0, the machine can either read d from channel x and produce d on channel u, and stay in s_0, or it can read d from channel y *once*, and produce d on channel v, and reach s_1. In that state, the machine never reads from y nor writes to channel u, and only reads from x, writes to v, and stays in s_1.



x
y
u
v
$\varepsilon_1 \colon \varphi_1$
$\varepsilon_3 \colon \varphi_3$
s_0
s_1
$\varepsilon_2 \colon \varphi_2$

Figure 4: Finite state machine with deadlock not detected by Verbeek *et al.*'s approach

Still, according to the definition by Verbeek *et al.*, the machine is not *dead* as it will read x infinitely often. In particular, $\mathbf{block}_{y(d)}$ is false, since $\varepsilon_2(i,d) = i = y$, hence $\varepsilon_2(y,d) = \mathit{True}$ according to their definition. Clearly, once state s_1 is reached, messages waiting on channel y will never be read.

The example clearly illustrates that, even though channel y is dead for d, this is not detected by the idle and block equations, since $\mathbf{block}_{y(d)}$ is false. The encoding by Verbeek *et al.* to idle and block equations is therefore unsound.

Intuitively, the problem with their definition is as follows. If a state machine has no available enabled transitions, *idle* and *block* are true on all channels. This is captured by **deadA**, and is, in fact, correct. The error lies in the second part of each definition. A state machine can block a channel while still having a transition reading that channel. For instance, if that channel is read *finitely many times.* A similar argument holds for **idle** and output channels.

4 Finite state machines

Verbeek *et al.*'s definition of xMAS automata [18, 19] allows for the symbolic description of channels and data read and written along transitions. However, it still only allows reading and writing (at most) one channel on every transition. To simplify presentation in this paper, we adapt the definition of xMAS automata into what we call finite state machines (FSMs). The key difference between FSMs and xMAS automata is that we require explicit definition of every datum read/written on a transition. Note that this does not fundamentally alter the expressive power:

the number of channels, as well as the data transferred along the channels are generally assumed to be finite, so they can simply be expanded in the finite state machine.

In the rest of this section we introduce finite state machines into xMAS.

Definition 12 (Finite state machine)**.** A *finite state machine (FSM)* is a tuple (S, s_0, I, O, T), where:

- S is a finite set of states;
- $s_0 \in S$ is an initial state;
- I is a finite set of input channels;
- O is a finite set of output channels
- $T \subseteq S \times ((I \times C) \cup \{\_\}) \times ((O \times C) \cup \{\_\}) \times S$ is the transition relation. We require T to be total, i.e., every state has at least one outgoing transition.

We also let $G = I \cup O$ be the set of all channels used by the FSM.

We typically use names $s, s', s_0, s_1, \ldots$ for states, and x and y for channels, and we write $?x(d)$ to denote a read of data d on input channel x, and $!y(e)$ to denote a write of data e on output channel y. We typically write $s \xrightarrow{?x(d)/!y(e)} s'$ to denote $(s, (x, d), (y, e), s') \in T$.

Remark 1. Henceforth, we require every transition to read from a channel and to write to a channel for the sake of simplicity. In other words, we assume the signature $T \subseteq S \times (I \times C) \times (O \times C) \times S$.

This is not a fundamental restriction. Transitions $t = s \xrightarrow{\_/!y(e)} s'$ that do not read from an input channel can be modeled by introducing a new channel x_t that is connected to a source and the FSM, and be replaced by $s \xrightarrow{?x_t/!y(e)} s'$. Likewise, transitions $t = s \xrightarrow{?x(d)/\_} s'$ that do not write to an output channel can be modeled using a channel y_t connected to a sink and the FSM, and be replaced by $s \xrightarrow{?x(d)/!y_t} s'$. Transitions $s \xrightarrow{\_/\_} s'$ that neither read an input channel nor write an output channel can be modeled using a combination of the above.

We also introduce some notation that will be helpful in defining idle and block equations for FSMs.

Notation 1. Given FSM (S, s_0, I, O, T), we introduce the following notation. For state $s \in S$, we have the incoming and outgoing transitions of s:

$$\begin{aligned} in_s(s) &= \{s' \xrightarrow{x(d)/y(e)} s'' \in T \mid s = s''\} \\ out_s(s) &= \{s' \xrightarrow{x(d)/y(e)} s'' \in T \mid s = s'\} \end{aligned}$$

For channels $x \in G$, and data $d \in C(x)$ we have the transitions reading d from x and writing d to x:

$$\begin{aligned} read(x, d) &= \{s \xrightarrow{i/o} s' \in T \mid i = x(d)\} \\ write(x, d) &= \{s \xrightarrow{i/o} s' \in T \mid o = x(d)\} \end{aligned}$$

In an FSM, exactly one state is current at a time, this state is denoted $cur(s)$. A transition $s \xrightarrow{x(d)/y(e)} s'$ is enabled if and only if s is the current state, the input channel x is ready to send d, and the output channel y is ready to receive. Note that whether the input and output channels are ready depends on the environment of the FSM.

Definition 13. Given FSM (S, s_0, I, O, T), transition $s \xrightarrow{x(d)/y(e)} s' \in T$ is enabled, denoted $enabled(s \xrightarrow{x(d)/y(e)} s')$ iff each of the following hold:

- $cur(s)$, and
- $x.\textbf{irdy} \wedge x.\textbf{data} = d$, and
- $y.\textbf{trdy}$.

In any given state, there can be multiple enabled transitions. To resolve this non-determinism, a scheduler *sel* is introduced that, at every clock cycle, selects an enabled transition. If a transition t is selected, we denote this using $\mathit{selected}(t)$. Note $\mathit{selected}(t) \implies \mathit{enabled}(t)$. Generally, *sel* is assumed to be fair, i.e., if state s is visited infinitely often with $s \xrightarrow{x(d)/y(e)} s'$ enabled, then $s \xrightarrow{x(d)/y(e)} s'$ will be selected infinitely often.

The environment of the finite state machine, in order to execute, relies on the FSM indicating whether it is ready to send along an outgoing channel, or to read along an incoming channel. The semantics in terms if **irdy**, **trdy** and **data** is defined as follows.

Definition 14. Given FSM (S, s_0, I, O, T), for $x \in I$ we define

- $x.\textbf{trdy} := \exists s \xrightarrow{x(d)/y(e)} s' \in T.\mathit{selected}(s \xrightarrow{x(d)/y(e)} s')$

For $y \in O$ we define

- $y.\textbf{irdy} := \exists s \xrightarrow{x(d)/y(e)} s' \in T.\mathit{selected}(s \xrightarrow{x(d)/y(e)} s')$
- $y.\textbf{data} := \begin{cases} e & \text{if } \exists s \xrightarrow{x(d)/y(e)} s' \in T.\mathit{selected}(s \xrightarrow{x(d)/y(e)} s') \\ \perp & \text{otherwise} \end{cases}$

Note that $y.\textbf{data}$ is well-defined since always exactly one of the enabled transitions is selected.

Example 5. Consider the xMAS network in Figure 5. The network consists of two FSM components. The first FSM has an input channel u, that is connected to a source, and an output channel x that is connected to the second FSM. The second FSM has input channels x (connected to the first FSM), and v, connected to a source, and it has output channel y connected to a sink. The data along all channels consists of a single token t.

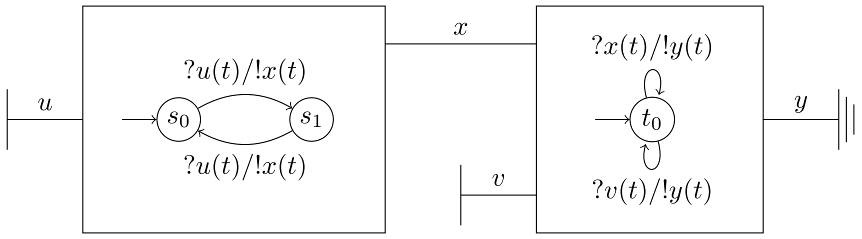


Figure 5: Synchronization example

The state transition diagram of the semantics of the network consisting of these two FSMs with the sources and sink, and channels u, v, x, y, according to the semantics from Section 2.2 is as shown in Figure 6.

Since the scheduler non-deterministically chooses between enabled transitions, and **irdy** is only set for the output channel of a selected transition, whenever **irdy** is set for an output channel of an FSM, than the target of that channel is ready to receive, i.e., **trdy** is set.

Lemma 4. *Given an xMAS network N with an FSM (S, s_0, I, O, T), for $y \in O$, we have $y.\textbf{irdy} \implies y.\textbf{trdy}$ in all global states of N.*

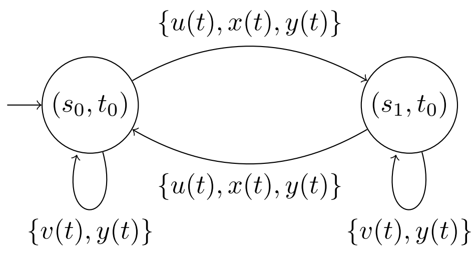


Figure 6: State machine semantics of the network with the FSMs from Figure 5

Proof. Fix an arbitrary global state in N. Let y be a channel in the FSM, with $y.\mathbf{irdy}$ set to true. According to Definition 14, there exists a transition $s \xrightarrow{x(d)/y(e)} s'$ such that $cur(s)$ and $selected(s \xrightarrow{x(d)/y(e)} s')$. Since $selected(s \xrightarrow{x(d)/y(e)} s') \implies enabled(s \xrightarrow{x(d)/y(e)} s')$, we have $enabled(s \xrightarrow{x(d)/y(e)} s')$. By Definition 13, we immediately get $y.\mathbf{trdy}$. $\square$

Finite state machines in xMAS are non-deterministic. In principle, this could lead to an enabled transition being ignored for an infinite amount of time. Since we assume the existence of fair schedulers to resolve the non-determinism in finite state machines, we only verify liveness of the xMAS network along fair paths. Such paths are defined as follows.

Definition 15. Given a path π, we say that π is fair if and only if for all FSM primitives $M = (S^M, s_0^m, I^M, O^M, T^M)$ and local transitions $t \in T^M$, we have $\pi \models (\mathsf{GF}\mathit{enabled}(t)) \implies (\mathsf{GF}\mathit{selected}(t))$

5 Idle and block equations for FSMs

To define idle and block equations for finite state machines in the spirit of [12], recall that there are two reasons for an input channel of an FSM to become dead. The first is structural: no state from which the channel can be read is ever reached again. The second depends on the environment: no transition along which the channel is read ever becomes enabled (in particular because the output channel that transition writes to is blocked).

The following two notions help capture this intuition. If a state is never reached, we say that it is idle. Likewise, if a transition is never enabled, we say that it is dead. Formally, this is defined as follows.

Definition 16 (Idle states and dead transitions)**.** Consider FSM (S, s_0, I, O, T). For $s \in S$ and $t \in T$ we define the following.

$$\begin{aligned} \mathbf{idle}(s) &:= \mathsf{FG}\neg cur(s) \\ \mathbf{dead}(t) &:= \mathsf{FG}\neg enabled(t) \end{aligned}$$

We now establish some properties about finite state machines in the context of an xMAS network. We show that whenever an FSM is in a particular state, it will stay in that state as long as none of its outgoing transitions are enabled.

Lemma 5. *Let $M = (S, s_0, I, O, T)$ be an FSM that appears in an xMAS network N, and let $\vec{s}$ be a global state in N. Let $s \in S$ be an arbitrary state in M. Then*

$$\vec{s} \models \mathsf{G}((cur(s) \wedge \bigwedge_{t \in out_s(s)} \neg enabled(t)) \implies \mathsf{X}cur(s))$$

Proof. We need to show that the property holds for all paths $\pi \in \mathsf{Paths}(\vec{s})$. So, consider an arbitrary such path. Towards a contradiction, suppose

$$\pi \not\models \mathsf{G}((\mathit{cur}(s) \wedge \bigwedge_{t \in \mathit{out}_s(s)} \neg \mathit{enabled}(t)) \implies \mathsf{X}\mathit{cur}(s)).$$

This is equivalent to

$$\pi \models \mathsf{F}((\mathit{cur}(s) \wedge \bigwedge_{t \in \mathit{out}_s(s)} \neg \mathit{enabled}(t)) \wedge \mathsf{X}\neg \mathit{cur}(s)).$$

Therefore, according to the LTL semantics, for some $j \geq 0$, we have

$$\pi[j..] \models (\mathit{cur}(s) \wedge \bigwedge_{t \in \mathit{out}_s(s)} \neg \mathit{enabled}(t)) \wedge \mathsf{X}\neg \mathit{cur}(s).$$

So, in $\pi[j]$, the FSM is in state s, none of its outgoing transitions are enabled, and in $\pi[j+1]$, the FSM is in some state $s' \in S$, with $s \neq s'$. However, according to the semantics of xMAS networks, the FSM either takes an enabled transition, or $s = s'$, hence we have a contradiction. □

We also derive the following property from persistency of channels. Whenever none of the outgoing transitions of a state in an FSM ever becomes enabled, then for all outgoing transitions of that state, if the input channel of the transition is ready to send a particular value, it will remain ready to send that value.

Lemma 6. *Let $M = (S, s_0, I, O, T)$ be an FSM that appears in an xMAS network N. For all local states $s \in S$, all global states $\vec{s}$, and paths $\pi \in \mathsf{Paths}(\vec{s})$ if*

$$\pi \models \mathsf{G}(\bigwedge_{t \in \mathit{out}_s(s)} \neg \mathit{enabled}(t))$$

then for all $s \xrightarrow{x(d)/y(e)} s' \in \mathit{out}_s(s)$, it holds that

$$\pi \models \mathsf{G}((\mathit{cur}(s) \wedge x.\mathbf{irdy} \wedge x.\mathbf{data} = d) \implies \mathsf{X}(x.\mathbf{irdy} \wedge x.\mathbf{data} = d))$$

Proof. Let s be an arbitrary state in the FSM, and let $\vec{s}$ be an arbitrary global state in N, and $\pi \in \mathsf{Paths}(\vec{s})$. Assume that

$$\pi \models \mathsf{G}(\bigwedge_{t \in \mathit{out}_s(s)} \neg \mathit{enabled}(t)).$$

Towards a contradiction, suppose that

$$\pi \not\models \mathsf{G}\left((\mathit{cur}(s) \wedge x.\mathbf{irdy} \wedge x.\mathbf{data} = d) \implies \mathsf{X}(x.\mathbf{irdy} \wedge x.\mathbf{data} = d)\right).$$

Hence,

$$\pi \models \mathsf{F}\left((\mathit{cur}(s) \wedge x.\mathbf{irdy} \wedge x.\mathbf{data} = d) \wedge \mathsf{X}(\neg x.\mathbf{irdy} \vee x.\mathbf{data} \neq d)\right).$$

According to the LTL semantics, there exists $j \geq 0$ such that

$$\pi[j..] \models (\mathit{cur}(s) \wedge x.\mathbf{irdy} \wedge x.\mathbf{data} = d) \wedge \mathsf{X}(\neg x.\mathbf{irdy} \vee x.\mathbf{data} \neq d).$$

Since channels are persistent, it must be the case that $\pi[j..] \models x.\mathbf{trdy}$. But then we have $\pi[j..] \models \mathit{cur}(s) \wedge x.\mathbf{irdy} \wedge x.\mathbf{data} = d \wedge x.\mathbf{trdy}$. However $x.\mathbf{trdy}$ is only true if there is an outgoing transition $s \xrightarrow{x(d)/y(e)} s'$ in s such that $\pi[j..] \models \mathit{enabled}(s \xrightarrow{x(d)/y(e)} s')$ according to Definition 14, which contradicts our assumption. □

These lemmata allow us to relate a transition eventually never becoming enabled along a path, to the observation that, along the same path, either the source state of the transition is idle, the input channel is idle, or the output channel is blocked.

Lemma 7. *Let $M = (S, s_0, I, O, T)$ be an FSM that appears in an xMAS network N. For all transitions $t = s \xrightarrow{x(d)/y(e)} s' \in T$, all global states $\vec{s}$, and all paths $\pi \in \mathsf{Paths}(\vec{s})$,*

$$\pi \models \mathsf{FG}\neg enabled(t) \text{ if and only iff } \pi \models \mathbf{idle}(s) \vee \mathbf{idle}(x(d)) \vee \mathbf{block}(y).$$

Proof. Fix an arbitrary transition $t = s \xrightarrow{x(d)/y(e)} s'$, global state $\vec{s}$, and path $\pi \in \mathsf{Paths}(\vec{s})$.
We prove both directions separately.

$\Rightarrow$ Suppose that $\pi \models \mathsf{FG}\neg enabled(t)$. So, there is an index $i \geq 0$ such that

$$\pi[i..] \models \mathsf{G}\neg enabled(t). \tag{1}$$

Towards a contradiction, suppose $\pi \not\models \mathbf{idle}(s) \vee \mathbf{idle}(x(d)) \vee \mathbf{block}(y(e))$. Hence, $\pi \models \neg\mathbf{idle}(s) \wedge \neg\mathbf{idle}(x(d)) \wedge \neg\mathbf{block}(y(e))$. Therefore, also:

$$\pi \models \neg\mathbf{idle}(s) \tag{2}$$
$$\pi \models \neg\mathbf{idle}(x(d)) \tag{3}$$
$$\pi \models \neg\mathbf{block}(y) \tag{4}$$

Since $\neg\mathbf{idle}(s) \equiv \mathsf{GF}cur(s)$, from Equation (2) and the semantics of LTL, there exists $k_1 \geq i$ such that $\pi[k_1..] \models cur(s)$.

Since $\neg\mathbf{idle}(x(d)) \equiv \mathsf{GF}(x.\mathbf{irdy} \wedge x.\mathbf{data} = d)$, from Equation (3) and the semantics of LTL, there exists $k_2 \geq k_1$ such that $\pi[k_2..] \models x.\mathbf{irdy} \wedge x.\mathbf{data} = d$. Fix the smallest such k_2, and observe that for all l such that $k_1 \leq l \leq k_2$, we have $\pi[l..] \models cur(s)$ according to our assumption and Lemma 5.

Since $\neg\mathbf{block}(y) \equiv \mathsf{GF}y.\mathbf{trdy}$, from Equation (4) and the semantics of LTL, there exists $k_3 \geq k_2$ such that $\pi[k_3..] \models y.\mathbf{trdy}$. Fix the smallest such k_3, and observe that for all l such that $k_2 \leq l \leq k_3$, we have $\pi[l..] \models x.\mathbf{irdy} \wedge x.\mathbf{data} = d$ according to our assumption and Lemma 6. Furthermore, we have $\pi[l..] \models cur(s)$ according to our assumption and Lemma 5.

But then, in particular, we have that

$$\pi[k_3..] \models cur(s) \wedge x.\mathbf{irdy} \wedge x.\mathbf{data} = d \wedge y.\mathbf{trdy}$$

hence $\pi[k_3..] \models enabled(s \xrightarrow{x(d)/y(e)} s')$. But since $k_3 \geq i$, this contradicts Equation 1.

$\Leftarrow$ Suppose $\pi \models \mathbf{idle}(s) \vee \mathbf{idle}(x(d)) \vee \mathbf{block}(y)$. We split the three cases.

- $\pi \models \mathbf{idle}(s)$. By definition of **idle**, $\pi \models \mathsf{FG}\neg cur(s)$. So, there exists $i \geq 0$ such that for all $j \geq i$, $\pi[j..] \models \neg cur(s)$. Since $\neg cur(s)$ implies $\neg enabled(s \xrightarrow{x(d)/y(e)} s')$ according to Definition 13, we have shown for all $j \geq i$, $\pi[j..] \models \neg enabled(s \xrightarrow{x(d)/y(e)} s')$, hence $\pi \models \mathsf{FG}\neg enabled(s \xrightarrow{x(d)/y(e)} s')$.
- $\pi \models \mathbf{idle}(x(d))$. By definition of **idle**, $\pi \models \mathsf{FG}\neg(x.\mathbf{irdy} \wedge x.\mathbf{data} = d)$. This again immediately implies $\pi \models \mathsf{FG}\neg enabled(s \xrightarrow{x(d)/y(e)} s')$ according to Definition 13.
- $\pi \models \mathbf{block}(y)$. The reasoning is again similar to the previous cases. □

The following lemma shows that output channels of a finite state machine are never dead. This is a consequence of how the semantics of FSMs resolves non-determinism.

Lemma 8. *Given an xMAS network N with an FSM (S, s_0, I, O, T), for all global states $\vec{s}$ and for channels $y \in O$ and $e \in C(y)$, we have for all paths $\pi \in \mathsf{Paths}(\vec{s})$, $\pi \not\models \mathbf{dead}(y(e))$.*

Proof. Let $\vec{s}$, $y \in O$, $e \in C(y)$ and $\pi \in \mathsf{Paths}(\vec{s})$ be arbitrary.

Towards a contradiction, suppose $\pi \models \mathbf{dead}(y(e))$. Hence, $\pi \models \neg\mathbf{idle}(y(e)) \wedge \mathbf{block}(y(e))$. Since $\pi \models \neg\mathbf{idle}(y(e))$, by definition of idle $\pi \models \neg\mathsf{FG}(\neg y.\mathbf{irdy} \vee y.\mathbf{data} \neq e)$ hence $\pi \models \mathsf{GF}(y.\mathbf{irdy} \wedge y.\mathbf{data} = e)$.

Since $\pi \models \mathbf{block}(y(e))$, $\pi \models \mathsf{FG}(\neg y.\mathbf{trdy} \vee y.\mathbf{data} \neq e)$. Hence, there exists $i \geq 0$ such that for all $j \geq i$, $\pi[j..] \models \neg y.\mathbf{trdy} \vee y.\mathbf{data} \neq e$. Let i be such. Since $\pi \models \mathsf{GF}(y.\mathbf{irdy} \wedge y.\mathbf{data} = e)$, for some $k \geq i$, we have $\pi[k..] \models y.\mathbf{irdy} \wedge y.\mathbf{data} = e$. According to Lemma 4, $y.\mathbf{irdy} \implies y.\mathbf{trdy}$, hence $\pi[k..] \models y.\mathbf{trdy} \wedge y.\mathbf{data} = e$, which is a contradiction. So, $\pi \not\models \mathbf{dead}(y(e))$. □

5.1 Idle and block equations for FSMs

We extend idle and block equations for xMAS networks by providing equations for finite state machines. The equations refer to some variables that are defined in idle and block equations of other components. Specifically, **idle** of incoming channels and **block** of outgoing channels is used in the encoding.

Definition 17 (Idle and block equations for FSMs)**.** Consider an FSM $M = (S, s_0, I, O, T)$. For $s \in S$, $x \in I$, $y \in O$, $d \in C(x)$, $e \in C(y)$, and $s \xrightarrow{x(d)/y(e)} s' \in T$ we define the following boolean equations.

$$\mathbf{block}_x^d = \bigwedge_{t \in read(x,d)} \mathbf{dead}_t \qquad \mathbf{block}_x = \bigwedge_{d \in C(x)} \mathbf{block}_x^d$$

$$\mathbf{idle}_y^e = \bigwedge_{t \in write(y,e)} \mathbf{dead}_t \qquad \mathbf{idle}_y = \bigwedge_{e \in C(y)} \mathbf{idle}_y^e$$

$$\mathbf{dead}_{s \xrightarrow{x(d)/y(e)} s'} = \mathbf{idle}_s \vee \mathbf{idle}_x^d \vee \mathbf{block}_y$$

$$\mathbf{idle}_s = \neg cur_s \wedge \bigwedge_{t \in in_s(s)} \mathbf{dead}_t$$

The formula $\mathsf{SAT}(M)$ consists of the conjunction of all of the above equations for all states, transitions and channels. We refer to the encoding of an entire network N as $\mathsf{SAT}(N)$.

The intuition behind the encoding is as follows. If a state is not current, and none of its incoming transitions can ever become enabled, the state is effectively unreachable, and thus the state is idle. In turn, a transition is dead if it can never become enabled. This is the case if either its source state or its incoming channel is idle, or its outgoing channel is blocked. An input channel is blocked for a given data value if no transition will read that value from the channel. Likewise an output channel is idle for a value if that value is never written to it. An output channel is idle if it is idle for all values, meaning that no value will ever be written to it. In input channel is blocked if it is blocked for all values. Intuitively, one might argue that an input channel should be blocked if it is blocked for some value, however, since we are interested in detecting dead channels, and in Lemma 3 we have proven that if a channel is dead, it is blocked for all values that could be sent along that channel, using conjunction here is sufficient to obtain soundness.

5.2 Soundness of idle and block equations

We finally prove that the idle and block equations that we have constructed are sound in the sense that, if there is a channel that is dead for a particular value, then there is a satisfying assignment to the boolean equations that shows this.

5.2.1 Consistency with the environment

The idle and block evaluations of FSMs are evaluated in the context of the idle and block equations of the entire network. To focus our reasoning to the part considering only the finite state machines, we assume consistency of assignments on the other components in the network. Correctness of the equations of the rest of the network follows from the original results in [12].

Definition 18 (Consistency)**.** We say that an assignment σ that assigns constants to variables is consistent for an equation $\Phi = \Psi$ in the encoding to idle and block equations if and only if $\sigma(\Phi) = \sigma(\Psi)$. The assignment σ is consistent for Boolean variable v if the defining equation $v = \Phi$ is consistent.

Note that in the above, $\sigma(\Phi)$ denotes the value that is obtained by assigning the constant from σ to every variable in Φ, and subsequently simplifying the resulting formula.

To formalise the assumption on the environment under which we can construct a satisfying assignment for the idle and block equations of an FSM, we assume we have a consistent assignment for all variables V that are not controlled by the FSM. For the variables R that are relevant to the truth value of the FSM, we further assume that they get a value that is consistent with a given path.

Definition 19. Consider the encoding $\mathsf{SAT}(N)$ of an xMAS network N, let $\pi \in \mathsf{Paths}(N)$ be a path, and σ an assignment to the variables in $\mathsf{SAT}(N)$. Let V and R be subsets of the variables in $\mathsf{SAT}(N)$. We say that σ is π-consistent with respect to V and R if σ is consistent for all variables $v \in V$, and

- if $\mathbf{block}_x^d \in R$ then $\pi \models \mathbf{block}(x(d))$ iff $\sigma(\mathbf{block}_x^d) = \top$, and
- if $\mathbf{idle}_x^d \in R$ then $\pi \models \mathbf{idle}(x(d))$ iff $\sigma(\mathbf{idle}_x^d) = \top$.

Given an FSM $M = (S, s_0, I, O, T)$, we define

$$\begin{aligned} R(M) &= \{\mathbf{idle}_x^d \mid x \in I, d \in C(x)\} \cup \{\mathbf{block}_x \mid x \in O\} \\ V(M) &= \{\mathbf{block}_x^d \mid x \in I, d \in C(x)\} \cup \{\mathbf{block}_x \mid x \in I\} \\ &\quad \cup \{\mathbf{idle}_x^d \mid x \in O, d \in C(x)\} \cup \{\mathbf{idle}_x \mid x \in O\} \\ &\quad \cup \{\mathbf{dead}_t \mid t \in T\} \end{aligned}$$

We write $V(\mathsf{SAT}(N))$ for the set of all variables in the encoding $\mathsf{SAT}(N)$. Note that $V(M)$ denotes the variables controlled by an FSM, an $V(\mathsf{SAT}(N)) \setminus V(M)$ consists of all variables controlled by the environment.

5.2.2 Building a satisfying assignment

For the soundness proof, the idea is now as follows. If we have a path π in network N on which channel x in an FSM is dead for d, and variable assignment σ that is π-consistent with respect to $V(\mathsf{SAT}(N)) \setminus V(M)$ and $R(M)$, then we can modify σ to some σ' such that σ' remains consistent, and such that it is a satisfying assignment for $\mathsf{SAT}(N) \wedge \neg\mathbf{idle}_x^d \wedge \mathbf{block}_x^d$.

Note that for soundness, we only have to consider the input channels of FSMs, since we have already shown that output channels of FSMs cannot be dead in Lemma 8.

Theorem 2. *Let $M = (S, s_0, I, O, T)$ be an FSM that appears in an xMAS network N. Let $x \in I$ be channel with $d \in C(x)$. If there exists a fair maximal path $\pi \in \mathsf{Paths}(\vec{s})$ such that $\pi \models \mathbf{dead}(x(d))$, and an assignment σ that is π-consistent with respect to $V(\mathsf{SAT}(N)) \setminus V(M)$ and $R(M)$, then there exists a satisfying assignment to the formula $\mathsf{SAT}(N) \wedge \neg\mathbf{idle}_x^d \wedge \mathbf{block}_x$.*

We postpone the proof of the theorem, and first prove some additional results from which the theorem follows. In particular, observe that a maximal path can either be finite or infinite, and in an infinite path in an xMAS network, the FSM can be stuck in a state locally. We construct satisfying assignments for each of the cases, and show that the assignments are satisfying assignments to the equations.

We first define the assignment for the case where an FSM is stuck locally (either on a finite or an infinite fair maximal path).

Definition 20. Let $M = (S, s_0, I, O, T)$ be an FSM in an xMAS network N, π a path in N, and let σ be a variable assignment that is π-consistent with respect to $V(\mathsf{SAT}(N)) \setminus V(M)$ and $R(M)$. Given a state $s \in S$, we define the variable assignment σ_s to $V(\mathsf{SAT}(N))$ as follows. For all variables $v \notin V(M)$, $\sigma_s(v) = \sigma(v)$. For $v \in V(M)$, the assignment is as follows. For states $s' \in S$, transitions $t \in T$, channels $x \in I$, $y \in O$, and $d \in C(x)$, $e \in C(y)$:

$$\sigma_s(cur_{s'}) := s = s' \qquad \sigma_s(\mathbf{idle}_{s'}) := s \neq s' \qquad \sigma_s(\mathbf{dead}_t) := \top$$
$$\sigma_s(\mathbf{block}^d_x) := \top \qquad \sigma_s(\mathbf{block}_x) := \top$$
$$\sigma_s(\mathbf{idle}^d_y) := \top \qquad \sigma_s(\mathbf{idle}_y) := \top$$

When it is clear from the context that we evaluate a SAT formula in the context of σ_s we omit it, and write, e.g., $cur_{s'}$ instead of $\sigma_s(cur_{s'})$.

We first show that if a (fair) maximal path in a network containing the FSM is finite (and thus ends in a global deadlock), the previous definition gives a satisfying assignment for the encoding to SAT.

Lemma 9. *Let $M = (S, s_0, I, O, T)$ be an FSM that appears in an xMAS network N. For all* finite fair maximal paths $\pi \in \mathsf{Paths}(N)$*, such that* $\mathsf{last}(\pi) \models cur(s)$ *for some $s \in S$, and all assignments σ that are π-consistent with respect to $V(\mathsf{SAT}(N)) \setminus V(M)$ and $R(M)$, the assignment σ_s is a satisfying assignment.*

Proof. Assume there exists a fair maximal path $\pi \in \mathsf{Paths}(N)$ such that π is finite and $\mathsf{last}(\pi) \models cur(s)$ for some $s \in S$.

We check consistency of the assignment σ_s. Note that σ_s is consistent for all equations that are generated for other components. We therefore consider only the equations generated for M. Note that the equations for **block** and **idle** of channels are trivially consistent since they only depend on **dead**, and all occurrences of **dead** are assigned $\top$. We therefore focus on the other two cases:

$\mathbf{idle}_q$**.** We first show for arbitrary $q \in S$, $\mathbf{idle}_q = \neg cur_q \wedge \bigwedge_{t \in in_s(q)} \mathbf{dead}_t$. If $q = s$, then $\mathbf{idle}_q = \bot$, and $cur_q = \top$, therefore, $\bot = \mathbf{idle}_q = \neg cur_q \wedge \bigwedge_{t \in in_s(q)} \mathbf{dead}_t = \bot \wedge \bigwedge_{t \in in_s(q)} \mathbf{dead}_t = \bot$ is consistent. If $q \neq s$, then $\top = \mathbf{idle}_q = \neg cur_q \wedge \bigwedge_{t \in in_s(q)} \mathbf{dead}_t = \neg\bot \wedge \bigwedge_{t \in in_s(q)} \top = \top$ is consistent.

$\mathbf{dead}_t$**.** For arbitrary $q \xrightarrow{x(d)/y(e)} q' \in T$ we show $\mathbf{dead}_{q \xrightarrow{x(d)/y(e)} q'} = \mathbf{idle}_q \vee \mathbf{idle}^d_x \vee \mathbf{block}_y$ is consistent. If $q \neq s$, then $\top = \mathbf{dead}_{q \xrightarrow{x(d)/y(e)} q'} = \mathbf{idle}_q \vee \mathbf{idle}^d_x \vee \mathbf{block}_y = \top \vee \mathbf{idle}^d_x \vee \mathbf{block}_y = \top$. If $q = s$, then since π is maximal, we have $\mathsf{last}(\pi) \models \mathsf{G}\neg enabled(q \xrightarrow{x(d)/y(e)} q')$. Furthermore, since $\mathsf{last}(\pi) \models cur(s)$, we have $\mathsf{last}(\pi) \models \neg\mathbf{idle}(s)$. Thus, from Lemma 7 it follows that $\mathbf{idle}(x(d))$ or $\mathbf{block}(y)$; since both are in $R(M)$, and σ is π-consistent, $\mathbf{idle}^d_x \vee \mathbf{block}_y = \top$, and σ_s is consistent. $\square$

In case a path is finite, but the FSM is stuck in a local deadlock, the same assignment is also a satisfying assignment.

Lemma 10. *Let $M = (S, s_0, I, O, T)$ be an FSM that appears in an xMAS network N. For all* fair maximal paths $\pi \in \mathsf{Paths}(N)$*, and all $s \in S$ such that $\pi \models \mathsf{FG}\left(cur(s) \wedge \bigwedge_{t \in out_s(s)} \neg enabled(t)\right)$, and all assignments σ that are π-consistent with respect to $V(\mathsf{SAT}(N)) \setminus V(M)$ and $R(M)$, the assignment σ_s is a satisfying assignment.*

Proof. Assume there exists a fair maximal path $\pi \in \mathsf{Paths}(N)$ such that π is infinite, and let $s \in S$ be such that $\pi \models \mathsf{FG}\left(cur(s) \wedge \bigwedge_{t \in out_s(s)} \neg enabled(t)\right)$.

We check consistency of the assignment σ_s. Note that σ_s is consistent for all equations that are generated for other components by construction. We therefore consider only the equations generated for M. Note that the equations for **block** and **idle** are trivially consistent since they only depend on **dead**, and all occurrences of **dead** are assigned $\top$. We therefore focus on the other two cases.

$\mathbf{idle}_q$**.** We first show for arbitrary $q \in S$, $\mathbf{idle}_q = \neg cur_q \wedge \bigwedge_{t \in in_s(q)} \mathbf{dead}_t$. If $q = s$, then $\mathbf{idle}_q = \bot$, and $cur_q = \top$, therefore, $\bot = \mathbf{idle}_q = \neg cur_q \wedge \bigwedge_{t \in in_s(q)} \mathbf{dead}_t = \bot \wedge \bigwedge_{t \in in_s(q)} \mathbf{dead}_t = \bot$ is consistent. If $q \neq s$, then $\top = \mathbf{idle}_q = \neg cur_q \wedge \bigwedge_{t \in in_s(q)} \mathbf{dead}_t = \neg\bot \wedge \bigwedge_{t \in in_s(q)} \top = \top$ is consistent.

$\mathbf{dead}_t$**.** For arbitrary $q \xrightarrow{x(d)/y(e)} q' \in T$ we show $\mathbf{dead}_{q \xrightarrow{x(d)/y(e)} q'} = \mathbf{idle}_q \vee \mathbf{idle}_x^d \vee \mathbf{block}_y$ is consistent. If $q \neq s$, then $\top = \mathbf{dead}_{q \xrightarrow{x(d)/y(e)} q'} = \mathbf{idle}_q \vee \mathbf{idle}_x^d \vee \mathbf{block}_y = \top \vee \mathbf{idle}_x^d \vee \mathbf{block}_y = \top$. If $q = s$, then we have $\mathbf{dead}_{q \xrightarrow{x(d)/y(e)} q'} = \top$, and $\mathbf{idle}_q = \bot$, and we need to show that $\mathbf{idle}_x^d \vee \mathbf{block}_y = \top$. Since $q = s$, $\pi \models \mathsf{FG}(cur(q) \wedge \bigwedge_{t \in out_s(q)} \neg enabled(t))$. Therefore, $\pi \models \mathsf{FG} cur(q)$ as well as $\pi \models \mathsf{FG} \bigwedge_{t \in out_s(q)} \neg enabled(t))$. From this, it follows that $\pi \models \neg\mathbf{idle}(q)$, and $\pi \models \mathsf{FG}\neg enabled(q \xrightarrow{x(d)/y(e)} q')$. Thus, according to Lemma 7, $\pi \models \mathbf{idle}(q) \vee \mathbf{idle}(x(d)) \vee \mathbf{block}(y)$. Since $\pi \models \neg\mathbf{idle}(q)$, we have $\pi \models \mathbf{idle}(x(d)) \vee \mathbf{block}(y(e))$ according to Lemma 7. Since $\mathbf{idle}(x(d))$ and $\mathbf{block}(y(e))$ are in $R(M)$, and σ is π-consistent, we get that $\mathbf{idle}_x^d \vee \mathbf{block}_y^e = \top$. Thus it follows that σ_s is consistent. $\square$

The assignment σ_s was used to construct a satisfying assignment in case the FSM is stuck in a local deadlock. When an FSM is not stuck, we construct a satisfying assignment based on an infinite path π.

Definition 21. Let $M = (S, s_0, I, O, T)$ be an FSM in an xMAS network N, let $\pi \in \mathsf{Paths}(N)$ be an infinite path, and let σ be a variable assignment that is π-consistent with respect to $V(\mathsf{SAT}(N)) \setminus V(M)$ and $R(M)$. We construct assignment σ_π such that for all $v \in V(\mathsf{SAT}(N)) \setminus V(M)$, $\sigma_\pi(v) = \sigma(v)$, and for all $v \in V(M)$, $\sigma_\pi(v)$ is as follows.

Since π is infinite and the network and all its data is finite, π is a lasso that consists of a prefix $\pi[0] \dots \pi[i]$ and a cycle $\pi[i]\pi[i+1] \dots \pi[i+n]$ such that $\pi[i+n+1] = \pi[i]$. So, $\pi[i]$ is the first state on the lasso that is on the cycle. Let $s \in S$ be such that $\pi[i] \models cur(s)$, i.e., s is the local state of the FSM at the beginning of the loop.

For states $s' \in S$, transitions $t \in T$, channels $x \in I$, $y \in O$, and $d \in C(x)$, $e \in C(y)$:

$$\begin{aligned}
\sigma_\pi(cur_{s'}) &:= s = s' \\
\sigma_\pi(\mathbf{idle}_{s'}) &:= \forall 0 \leq k \leq n. \pi[i+k] \models \neg cur(s') \\
\sigma_\pi(\mathbf{dead}_t) &:= \forall 0 \leq k \leq n. \pi[i+k] \models \neg enabled(t) \\
\sigma_\pi(\mathbf{block}_x^d) &:= \forall t \in read(x,d). \forall 0 \leq k \leq n. \pi[i+k] \models \neg enabled(t) \\
\sigma_\pi(\mathbf{block}_x) &:= \forall d \in C(x). \forall t \in read(x,d). \forall 0 \leq k \leq n. \pi[i+k] \models \neg enabled(t) \\
\sigma_\pi(\mathbf{idle}_y^d) &:= \forall t \in write(y,e). \forall 0 \leq k \leq n. \pi[i+k] \models \neg enabled(t) \\
\sigma_\pi(\mathbf{idle}_y) &:= \forall e \in C(y). \forall t \in write(y,e). \forall 0 \leq k \leq n. \pi[i+k] \models \neg enabled(t)
\end{aligned}$$

When it is clear from the context that we evaluate a SAT formula in the context of σ_π we omit it, and write, e.g., $cur_{s'}$ instead of $\sigma_\pi(cur_{s'})$.

We next show that for infinite, fair maximal paths in a network containing the FSM, on which the FSM is not stuck locally, the previous definition gives a satisfying assignment for the encoding to SAT.

Lemma 11. *Let $M = (S, s_0, I, O, T)$ be an FSM that appears in an xMAS network N. For all* infinite fair maximal paths $\pi \in \mathsf{Paths}(N)$, *and all assignments σ, if for all $s \in S$, $\pi \models \mathsf{GF}\left(cur(s) \implies \bigvee_{t \in out_s(s)} enabled(t)\right)$, and σ is π-consistent with respect to $V(\mathsf{SAT}(N)) \setminus V(M)$ and $R(M)$, then the assignment σ_π is consistent.*

Proof. Fix an arbitrary fair maximal infinite path $\pi \in \mathsf{Paths}(N)$ and σ such that for all $s \in S$, $\pi \models \mathsf{GF}\left(\mathit{cur}(s) \implies \bigvee_{t \in \mathit{outs}_s(s)} \mathit{enabled}(t)\right)$, and σ is π-consistent with respect to $V(\mathsf{SAT}(N)) \setminus V(M)$ and $R(M)$.

We check that the assignment σ_π is consistent. For this, let i be such that it denotes the start of the cycle on π, and suppose $s \in S$ is such that $\pi[i] \models \mathit{cur}(s)$.

idle$_q$**.** We first show for arbitrary $q \in S$ that $\mathbf{idle}_q = \neg \mathit{cur}_s \wedge \bigwedge_{t \in \mathit{ins}_s(s)} \mathbf{dead}_t$. We distinguish two cases. If $\mathbf{idle}_q = \top$, then $\pi[i..] \models \mathsf{G}\neg \mathit{cur}(q)$. Since π is fair, $\pi \models \mathsf{FG} \bigwedge_{t \in \mathit{ins}_s(q)} \neg \mathit{enabled}(t)$, otherwise, one of the transitions would eventually be selected, and q would be reached. Then, for all transitions $t \in \mathit{ins}_s(q)$, $\pi \models \mathsf{FG}\neg \mathit{enabled}(t)$, hence $\mathbf{dead}_t = \top$ for all such transitions. Since $\pi[i..] \models \mathit{cur}(s)$, and $\pi[i..] \models \mathsf{G}\neg \mathit{cur}(q)$, $s \neq q$, thus $\mathit{cur}_q = \bot$. Therefore, $\top = \mathbf{idle}_q = \neg \mathit{cur}_q \wedge \bigwedge_{t \in \mathit{ins}_s(q)} \mathbf{dead}_t = \top$.

If $\mathbf{idle}_q = \bot$, then if $\pi[i] \models \mathit{cur}(q)$, $\mathit{cur}_q = \top$, and the result follows immediately from $\bot = \mathbf{idle}_q = \neg \mathit{cur}_q \wedge \bigwedge_{t \in \mathit{ins}_s(q)} \mathbf{dead}_t = \bot \wedge \bigwedge_{t \in \mathit{ins}_s(q)} \mathbf{dead}_t = \bot$. Suppose $\pi[i] \not\models \mathit{cur}(q)$. Then, for some k such that $0 \leq k \leq n$, $\pi[i+k] \models \mathit{cur}(q)$. Let k be the smallest such that $\pi[i+k] \models \mathit{cur}(q)$. Observe that $k > 0$. Then, $\pi[i+k-1] \models \mathit{enabled}(t)$ for some $t \in \mathit{ins}_s(q)$. Hence, $\mathbf{dead}_t = \bot$ for this t. Then, $\bigwedge_{t \in \mathit{ins}_s(q)} \mathbf{dead}_t = \bot$, therefore $\mathbf{idle}_q = \neg \mathit{cur}_q \wedge \bigwedge_{t \in \mathit{ins}_s(q)} \mathbf{dead}_t = \neg \mathit{cur}_q \wedge \bot = \bot$ is consistent.

dead$_t$**.** For arbitrary $q \xrightarrow{x(d)/y(e)} q' \in T$, we show $\mathbf{dead}_{q \xrightarrow{x(d)/y(e)} q'} = \mathbf{idle}_q \vee \mathbf{idle}_x^d \vee \mathbf{block}_y$ is consistent.

If $\mathbf{dead}_{q \xrightarrow{x(d)/y(e)} q'} = \top$, then $\pi[i+k] \models \neg \mathit{enabled}(q \xrightarrow{x(d)/y(e)} q')$ for all $0 \leq k \leq n$, hence $\pi \models \mathsf{FG}\neg \mathit{enabled}(q \xrightarrow{x(d)/y(e)} q')$. If there exists $0 \leq k \leq n$ such that $\pi[i+k] \models \mathit{cur}(q)$, then it must be the case that $\pi \models \mathbf{idle}(x(d)) \vee \mathbf{block}(y)$, otherwise $\pi \models \mathsf{GF}\mathit{enabled}(q \xrightarrow{x(d)/y(e)} q')$, which is a contradiction. Since $\mathbf{idle}_x^d, \mathbf{block}_y \notin V(M)$, $\mathbf{idle}_x^d = \top$ or $\mathbf{block}_y = \top$ since σ is π-consistent w.r.t $V(\mathsf{SAT}(N)) \setminus V(M)$ and $R(M)$, and $\top = \mathbf{dead}_{q \xrightarrow{x(d)/y(e)} q'} = \mathbf{idle}_q \vee \mathbf{idle}_x^d \vee \mathbf{block}_y = \top$ is consistent.

If $\mathbf{dead}_{q \xrightarrow{x(d)/y(e)} q'} = \bot$, then $\pi[i+k] \models \mathit{enabled}(q \xrightarrow{x(d)/y(e)} q')$ for some k. Then $\pi[i+k] \models \mathit{cur}(q)$, thus $\mathbf{idle}_q = \bot$, and also, from the definition of enabled, it follows immediately that $\pi \models \neg \mathbf{idle}(x(d)) \wedge \neg \mathbf{block}(y)$. Since $\mathbf{idle}_x^d, \mathbf{block}_y \notin V(M)$, $\mathbf{idle}_x^d = \bot$ and $\mathbf{block}_y = \bot$ since σ is π-consistent w.r.t $V(\mathsf{SAT}(N)) \setminus V(M)$ and $R(M)$. Therefore, $\bot = \mathbf{dead}_{q \xrightarrow{x(d)/y(e)} q'} = \mathbf{idle}_q \vee \mathbf{idle}_x^d \vee \mathbf{block}_y = \bot$ is consistent.

block$_x^d$**.** We show $\mathbf{block}_x^d = \bigwedge_{t \in \mathit{read}(x,d)} \mathbf{dead}_t$ for arbitrary channel x and $d \in C(x)$. Observe that $\mathbf{block}_x^d = \top$ if and only if $\forall t \in \mathit{read}(x,d)$, $0 \leq k \leq n$, $\pi[i+k] \models \neg \mathit{enabled}(t)$. By definition of **dead**, $\mathbf{dead}_t = \top$ for all such transitions t as well, and the assignment is consistent by definition.

block$_x$**.** Consistency of $\mathbf{block}_x = \bigwedge_{d \in C(x)} \mathbf{block}_x^d$ follows immediately from the definitions.

idle$_y^e$**.** Showing for arbitrary channel y and $e \in C(y)$ that $\mathbf{idle}_y^e = \bigwedge_{t \in \mathit{write}(y,e)} \mathbf{dead}_t$ and $\mathbf{idle}_y = \bigwedge_{e \in C(y)} \mathbf{idle}_y^e$ are consistent is analogous to the case of $\mathbf{block}_x^d$.

idle$_y$**.** Consistency of $\mathbf{idle}_y = \bigwedge_{e \in C(y)} \mathbf{idle}_y^e$ follows immediately from the definitions.

Hence σ_π is a consistent satisfying assignment to $\mathsf{SAT}(N)$. □

The following lemma shows that if an xMAS network N satisfies $\mathbf{idle}(s)$ for some FSM state s, then there exists a satisfying assignment to $\mathsf{SAT}(N)$ in which $\mathbf{idle}(s) = \top$.

Lemma 12. *Let $M = (S, s_0, I, O, T)$ be an FSM that appears in an xMAS network N, and let $s \in S$ be a local state in M. If there exists a fair maximal path $\pi \in \mathsf{Paths}(N)$ such that $\pi \models \mathbf{idle}(s)$, and an assignment σ that is π-consistent with respect to $V(\mathsf{SAT}(N)) \setminus V(M)$ and $R(M)$, then there exists a satisfying assignment to the formula $\mathsf{SAT}(N) \wedge \mathbf{idle}_s$.*

Proof. Fix an arbitrary local state s, and assume there exists a fair maximal path $\pi \in \mathsf{Paths}(N)$ such that $\pi \models \mathbf{idle}(s)$. Let σ be an assignment as specified. According to the definition of **idle**, $\pi \models \mathsf{FG}\neg cur(s)$. We distinguish two cases: either π is finite, or π is infinite.

π is finite. Let $p \in S$ be such that $\mathsf{last}(\pi) \models cur(p)$. According to Lemma 9, σ_p is a satisfying assignment for $\mathsf{SAT}(N)$. Since $\pi \models \mathbf{idle}(s)$, $\mathsf{last}(\pi) \not\models cur(s)$, hence $\sigma_p(\mathbf{idle}_s) = \top$, and the assignment also satisfies $\mathsf{SAT}(N) \wedge \mathbf{idle}_s$.

π is infinite. We distinguish two cases.

First, assume $\pi \models \mathsf{FG}(cur(s') \wedge \bigwedge_{t \in out_s(s')} \neg enabled(t))$ for some $s' \in S$. Let p be such a state. According to Lemma 10, σ_p is a consistent assignment for $\mathsf{SAT}(N)$.

Since $\pi \models \mathbf{idle}(s)$, for some $i \geq 0$, and all $j \geq i$, $\pi[j] \models \neg cur(s)$, which means it must be the case that $p \neq s$, hence $\sigma_p(\mathbf{idle}_s) = \top$ hence σ_p is a satisfying assignment for $\mathsf{SAT}(N) \wedge \mathbf{idle}_s$.

Otherwise, $\pi \not\models \mathsf{FG}(cur(s') \wedge \bigwedge_{t \in out_s(s')} \neg enabled(t))$ for all $s' \in S$. Hence, for all $s' \in S$, $\pi \models \mathsf{GF}(cur(s') \implies \bigvee_{t \in out_s(s')} enabled(t))$. According to Lemma 11, σ_π is consistent with $\mathsf{SAT}(N)$.

Let i be the index that signals the start of the loop of the lasso. Observe that, since $\pi \models \mathbf{idle}(s)$, $\pi \models \mathsf{FG}\neg cur(s)$, so for all $0 \leq k \leq n$, $\pi[i+k] \models \neg cur(s)$, hence $\sigma_\pi(\mathbf{idle}_s) = \top$, and σ_π is a satisfying assignment for $\mathsf{SAT}(N) \wedge \mathbf{idle}_s$.

So, in all cases, a satisfying assignment for $\mathsf{SAT}(N) \wedge \mathbf{idle}_s$ exists. $\square$

We finally return to the proof of Theorem 2 to establish that our idle and block equations are sound. The structure of the proof is similar to that of the previous lemma.

Theorem 2. *Let $M = (S, s_0, I, O, T)$ be an FSM that appears in an xMAS network N. Let $x \in I$ be channel with $d \in C(x)$. If there exists a fair maximal path $\pi \in \mathsf{Paths}(\vec{s})$ such that $\pi \models \mathbf{dead}(x(d))$, and an assignment σ that is π-consistent with respect to $V(\mathsf{SAT}(N)) \setminus V(M)$ and $R(M)$, then there exists a satisfying assignment to the formula $\mathsf{SAT}(N) \wedge \neg\mathbf{idle}^d_x \wedge \mathbf{block}_x$.*

Proof. Assume there exists a fair maximal path π such that $\pi \models \mathbf{dead}(x(d))$. Let σ be an assignment as specified.

By definition of **dead**, $\pi \models \neg\mathbf{idle}(x(d)) \wedge \mathbf{block}(x)$, hence $\pi \models \neg\mathbf{idle}(x(d))$ and $\pi \models \mathbf{block}(x)$. Since $\mathbf{idle}^d_x \notin V(M)$, $\sigma(\mathbf{idle}^d_x) = \bot$ according to the assumptions.

We distinguish two cases

π is finite. Let $p \in S$ be such that $\mathsf{last}(\pi) = p$. Then according to Lemma 9, the assignment σ_p is consistent with $\mathsf{SAT}(N)$. Note that $\sigma_p(\mathbf{block}_x) = \top$ and since $idle^d_x \notin V(M)$, $\sigma_p(\mathbf{idle}^d_x) = \sigma(\mathbf{idle}^d_x) = \bot)$. Hence we have a satisfying assignment for $\mathsf{SAT}(N) \wedge \neg\mathbf{idle}^d_x \wedge \mathbf{block}_x$.

π is infinite. We distinguish two cases.

First, suppose $\pi \models \mathsf{FG}(cur(s') \wedge \bigwedge_{t \in out_s(s')} \neg enabled(t))$ for some $s' \in S$. Let p be such. According to Lemma 10, σ_p is consistent with $\mathsf{SAT}(N)$. Using similar reasoning as in the previous case, we can conclude that σ_p is a satisfying assignment for $\mathsf{SAT}(N) \wedge \neg\mathbf{idle}^d_x \wedge \mathbf{block}_x$.

Otherwise, for all $s' \in S$, we have $\pi \not\models \mathsf{FG}(cur(s') \wedge \bigwedge_{t \in out_s(s')} \neg enabled(t))$, i.e., $\pi \models \mathsf{GF}(cur(s') \implies \bigvee_{t \in out_s(s')} enabled(t))$.

According to Lemma 11, σ_π is consistent with $\mathsf{SAT}(N)$. Note that since $\pi \models \mathbf{dead}(x(d))$, $\pi \models \mathbf{block}(x(e))$ for all $e \in C(x)$, according to Lemma 3. Consider arbitrary $e \in C(x)$, we show that the assignment satisfies $\mathbf{block}_x^e$. From this and the definition it immediately follows that is satisfies $\mathbf{block}_x$. Let i be the index that signals the start of the loop of the lasso π. Since $\pi \models \mathbf{block}(x(e))$, $\pi \models \mathsf{FG}(\neg x.\mathbf{trdy} \vee x.\mathbf{data} \neq e)$. By definition of *enabled*, this implies $\pi \models \mathsf{FG}(\neg \mathit{enabled}(t))$ for all $t \in \mathit{read}(x, e)$. Hence, for all $0 \leq k \leq n$, $\pi[i+k] \models \neg \mathit{enabled}(t)$ for all $t \in \mathit{read}(x, e)$. By definition of σ_π, we then have $\sigma_\pi(\mathbf{block}_x^e) = \top$. Since this holds for all e, by definition also $\sigma_\pi(\mathbf{block}_x) = \top$, and σ_π is a satisfying assignment for $\mathsf{SAT}(N) \wedge \neg \mathbf{idle}_x^d \wedge \mathbf{block}_x$. □

So, assuming that idle and block equations for other components are sound, we have proven that also the idle and block equations for finite state machines are sound.

5.3 Examples

We reconsider the example that illustrated the approach from [18, 19] was unsound, and show that our approach correctly detects deadlocks.

Example 6. Recall the finite state machine from Figure 4. We repeat it here as Figure 7, and update the notation to be consistent with our definitions. Note that the FSM only uses a single color as data, and data is therefore omitted from the figure.

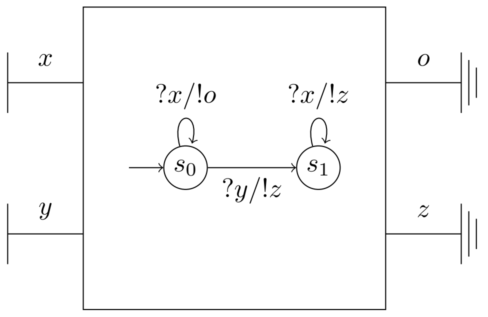


Figure 7: Finite state machine from Figure 4

The full SAT encoding of this example is the following:

$$
\begin{aligned}
\mathbf{idle}_{s_0} &= \neg \mathit{cur}_{s_0} \wedge \mathbf{dead}_{s_0 \xrightarrow{x/o} s_1} \\
\mathbf{idle}_{s_1} &= \neg \mathit{cur}_{s_1} \wedge \mathbf{dead}_{s_0 \xrightarrow{y/z} s_1} \wedge \mathbf{dead}_{s_1 \xrightarrow{x/z} s_1} \\
\mathbf{dead}_{s_0 \xrightarrow{x/o} s_1} &= \mathbf{idle}_{s_0} \vee \mathbf{idle}_x \vee \mathbf{block}_o \\
\mathbf{dead}_{s_0 \xrightarrow{y/z} s_1} &= \mathbf{idle}_{s_0} \vee \mathbf{idle}_y \vee \mathbf{block}_z \\
\mathbf{dead}_{s_1 \xrightarrow{x/z} s_1} &= \mathbf{idle}_{s_1} \vee \mathbf{idle}_x \vee \mathbf{block}_z \\
\mathbf{block}_x &= \mathbf{dead}_{s_0 \xrightarrow{x/o} s_1} \wedge \mathbf{dead}_{s_1 \xrightarrow{x/z} s_1} \\
\mathbf{block}_y &= \mathbf{dead}_{s_0 \xrightarrow{y/z} s_1} \\
\mathbf{idle}_o &= \mathbf{dead}_{s_0 \xrightarrow{x/o} s_1} \\
\mathbf{idle}_z &= \mathbf{dead}_{s_1 \xrightarrow{x/z} s_1}
\end{aligned}
$$

The environment is such that it guarantees that $\mathbf{idle}_x = \mathbf{idle}_y = \mathbf{block}_o = \mathbf{block}_z = \bot$. Now, the following assignment is a satisfying assignment for this system:

$$
\begin{array}{rrr}
\mathit{cur}_{s_0} := \bot & \mathit{cur}_{s_1} := \top & \\
\mathbf{idle}_{s_0} := \top & \mathbf{idle}_{s_1} := \bot & \\
\mathbf{dead}_{s_0 \xrightarrow{x/o} s_1} := \top & \mathbf{dead}_{s_0 \xrightarrow{y/z} s_1} := \top & \mathbf{dead}_{s_1 \xrightarrow{x/z} s_1} := \bot \\
\mathbf{block}_x := \bot & \mathbf{block}_y := \top & \\
\mathbf{idle}_o := \top & \mathbf{idle}_z := \bot &
\end{array}
$$

Note that this assignment satisfies $\mathbf{block}_y = \top$. We thus satisfy $\neg\mathbf{idle}_y \wedge \mathbf{block}_y$, hence y is dead, and we correctly detect the deadlock in this network.

6 Invariants

In the form introduced thus far, there are many satisfying assignments to Boolean equations that are not reachable. To restrict the number of satisfying assignments, the reachable state space can be approximated using invariants as proposed by Chatterjee and Kishinevsky [6]. Essentially, for every channel x, and datum $d \in C(x)$, a variable λ_x^d is introduced that represents the number of times d was transferred along channel x, i.e., the number of clock ticks at which $x.\mathbf{irdy} \wedge x.\mathbf{data} = d \wedge x.\mathbf{trdy}$ held true. For queues q, $\#q.d$ denotes the number of d-packets in the queue. For every primitive, the λ values for input and output channels are related, and for queues the content is taken into account.

For example, for queues with input channel x and output channel y, we have $\lambda_y^d = \lambda_x^d - \#q.d$, i.e., the number of times d has been transferred along the outgoing channel equals the number of times d has been received, minus the number of d-packets that are still in the queue.

For the function primitive that we saw before, with input channel x, output channel y and function f, $\lambda_y^e = \sum\{\lambda_x^d \mid f(d) = e\}$, i.e., the number of times e is sent along y equals the number of times a packet d that is mapped onto e has been received along x.

Verbeek *et al.* described invariants for xMAS automata. We here translate the approach to our setting.

The first invariant requires that the FSM is always in exactly one state. Note that we abuse notation and write $cur_s = 1$ whenever $cur_s = \top$.[2] This is [19, Invariant (1)]:

$$\sum_{s \in S} cur_s = 1.$$

The second invariant relates the number of times incoming and outgoing transitions of a state s have been taken. This uses variables κ_t, denoting the number of times transition t has been taken. The resulting invariant for a state s is [19, Invariant (2)]:

$$\sum_{t \in in_s(s)} \kappa_t = \left(\sum_{t \in out_s(s)} \kappa_t \right) + cur_s - (s = s_0)$$

Note that $s = s_0$ takes care of the fact that the automaton initially ends up in the initial state s_0 without taking a transition, and cur_s accounts for the situation where, if s is the current state, we still have to take an outgoing transition.

Due to our simplified presentation of FSMs, the other invariants presented in [19] can be simplified. They relate the number of times data has been transferred along an input channel of an automaton to the number of times a transition reading that data has been taken, and likewise for output channels.

For the input channels we thus get a simplification of [19, Invariant (3)], where for $x \in I$ and $d \in C(x)$, we get:

$$\lambda_x^d = \sum_{t \in read(x,d)} \kappa_t.$$

For output channels $y \in O$ and $d \in C(y)$, we get a similar equation inspired by [19, Invariant (4)]:

$$\lambda_y^d = \sum_{t \in write(y,d)} \kappa_t.$$

These invariants are incorporated in our proofs in a similar way as in [6, 19].

[2]The expression can easily be expanded to a boolean condition.

7 Experiments

We have implemented the idle and block equations described in Section 5 in our MaDL design & verification toolset [1]. This toolset uses an an xMAS model as input, and from this it automatically generates an SMT problem that directly incorporates the idle and block equations. The SMT problem is then solved by a state-of-the-art SMT-solver to verify liveness. Additionally, the toolset can generate a model that encodes the xMAS network and the way it behaves responding to external stimuli in the SMV specification format. In the SMV model, block and idle equations are used as invariants. This enables the nuXmv model-checker [4] to check reachability of a state in which a channel of the given xMAS model is not idle and blocked.

7.1 Experimental setup

We perform experiments with two kinds of models. The first set of models is inspired by "go/no go" testing. The second models a power domains architecture, and is inspired by industrial practice. Every model in the set has a corresponding modification in which some channel is dead.

Go/no go models The "go/no go" models are built as follows. The basic building block is the FSM, depicted in Figure 8. The FSM has two inputs and two outputs. It FSM reads from the first input, writing the signal which is read to the second output. Then it reads from the second input, and depending on the data, which was read from both inputs, it either writes *ok*, or *nok* to the first output. By combining two such FSMs we obtain a "go/no go" building block, depicted in Figure 9.

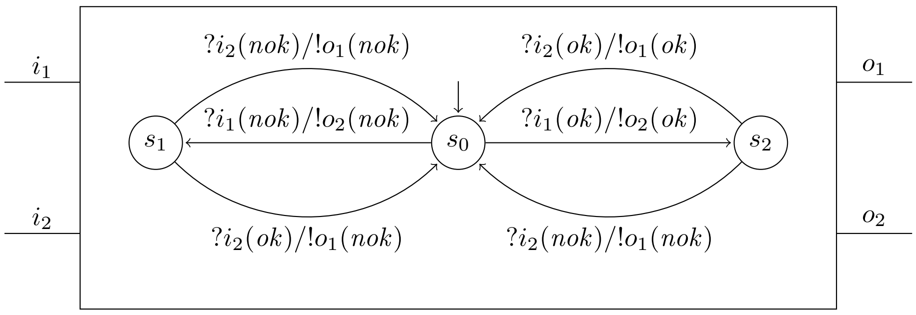


Figure 8: "go/no go" FSM.

We construct models of varying sizes by composing these blocks similarly to the way one builds a binary tree, i.e., we add new "go/no go" blocks by connecting the output of each new block we are adding to an input of a block which is a leaf of the tree. Every "go/no go" model has a number in its name, which denotes the number of FSMs from Figure 8.

To obtain "go/no go" models with deadlocks, we modify deadlock-free "go/no go" models by altering an FSM which is part of a building block whose inputs are not connected to another "go/no go" block. The modification is done as follows. We add a new state with a self-loop reading *ok* from channel i. We also add a transition from s_0 to this new state, on which *nok* is read from channel i. The modified FSM now has a reachable state in which channel i is blocked for *nok*, and in which all output channels of the FSM are idle.

Power domain models Systems on chip require power efficiency. This is achieved by a power control architecture that turns power domains on and off depending on the needs of an application. For our experiments, we model a dynamic power management policy, which is an abstraction of the ones used in industrial practice.

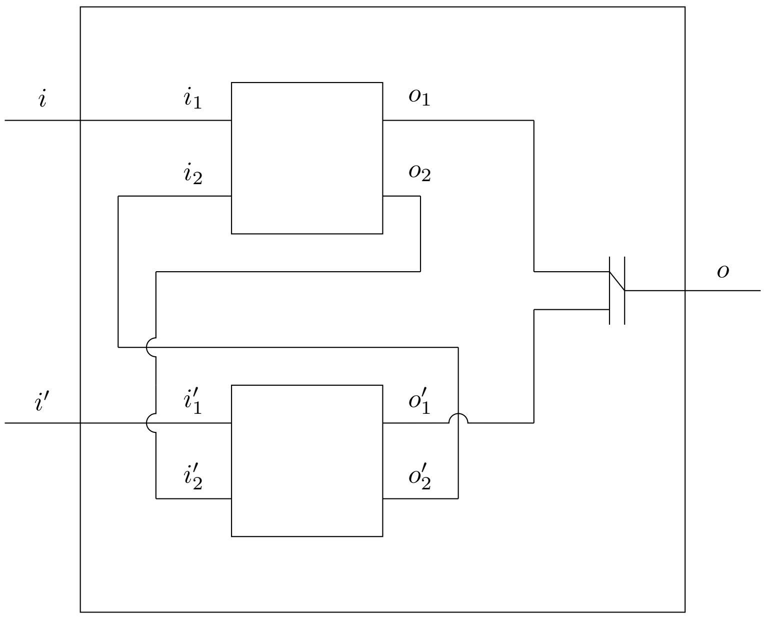


Figure 9: “go/no go” block.

A power domain controller powers on the domain when a device belonging to the domain shows activity. When there is no activity within the power domain, and all device controllers indicate that their respective devices are turned off, the power domain controller cuts power.

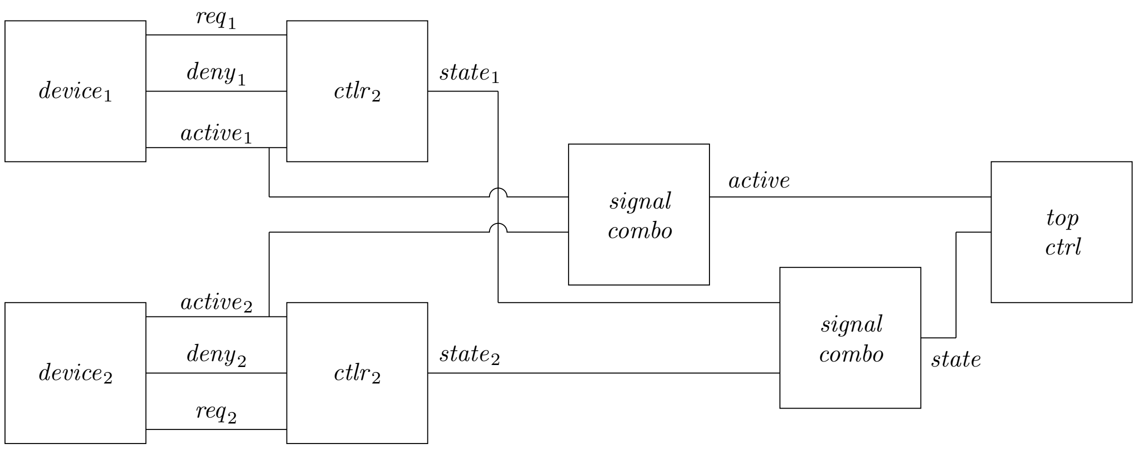


Figure 10: Power domain.

Within every power domain, there is a number of device-controller pairs. Devices are modeled using two FSMs: the activity generation FSM (Figure 11c), and the device FSM (Figure 11a). We also connect a source to the input of every activity generator. The device FSM reacts to turn on and turn off requests. Note that a turn off request can be denied. Device controllers are modeled by the FSM depicted in Figure 11b. A controller responds to activity, by requesting to turn on the corresponding device. Upon turning on the device, the controller sends a turned on signal to the power domain controller. In the absence of activity, the controller sends a turn off request to the device, and in case the request is not denied, the controller sends a turned off signal further to the domain power controller. Activity and powered on signals are combined using the FSM depicted in Figure 11d. We scale the power domain models by adding more power domains and more device-controller pairs to domains.

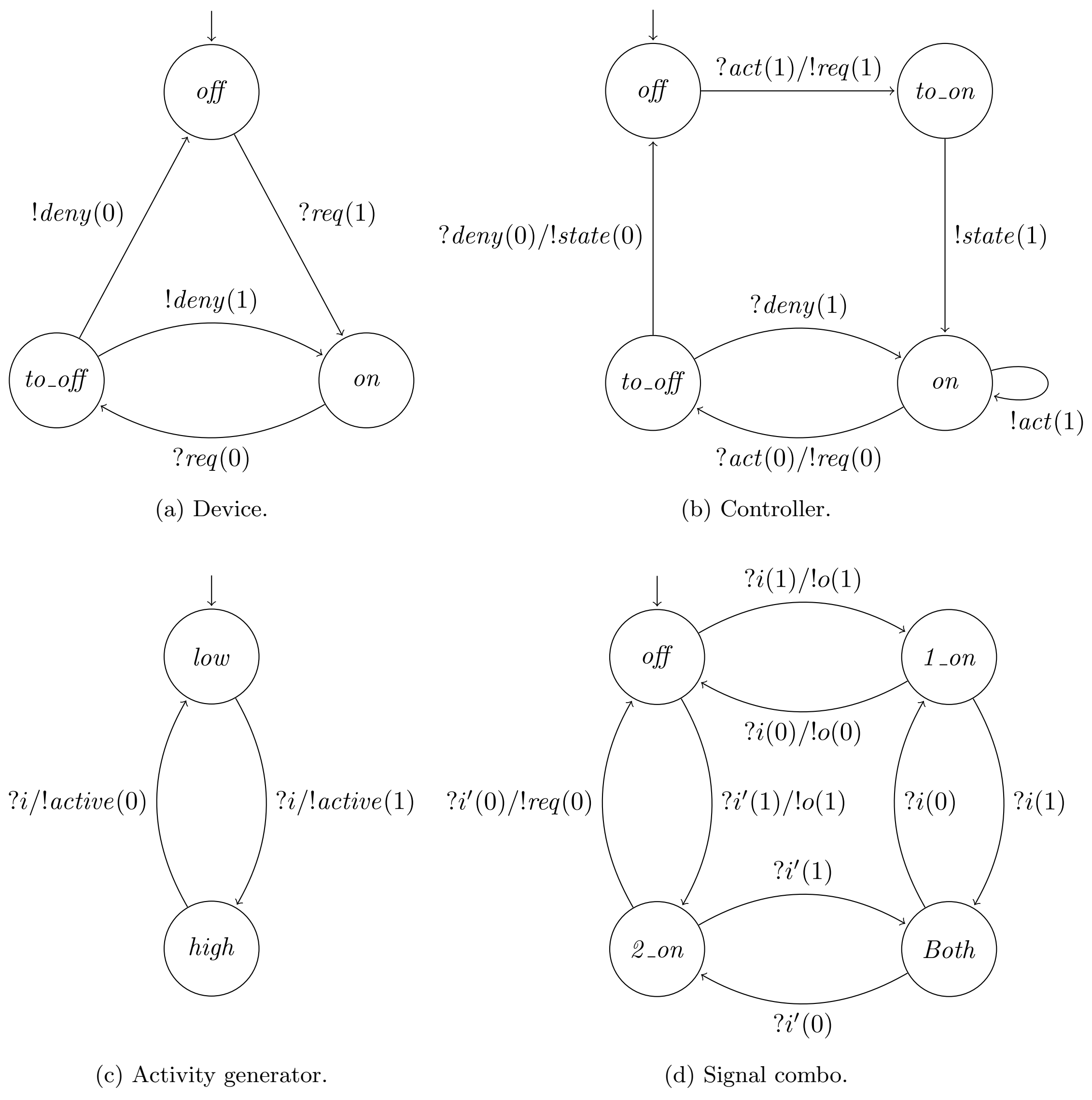


Figure 11: FSMs for power domains.

Power domains, and within these domains the devices and device controllers are indexed. To add a deadlock to a model, we change the FSM of the device controller with the highest index within the power domain that has the highest index as follows. We add a new state which can be reached from the *on* state by reading 0 from channel *act*. From the newly added state, it is only possible to read 1 from *act*. Therefore, from this new state, channel *act* is dead for 0, and all outgoing channels of the FSM are idle.

All experiments were conducted on a MacBook Pro 2015, 2,7GHz Intel Core i5, 16Gb RAM, running MacOS Sierra. For SMT problem solving, we use the Z3 SMT-solver, version 4.8.0 64-bit [9]. For reachability checks, we use NuXmv, version 2.0.0 64-bit [4]. Instructions to reproduce the experiments and the script used to obtain the results reported in this paper can be found at [1].

Model	#FSMs	Deadlock free	SMT		Reachability	
			Result	Time (s)	Result	Time (s)
go_no_go_7	14	✓	✓	1.488	✓	3.237
go_no_go_7_dl	14	✗	✗	1.549	✗	10.012
go_no_go_11	22	✓	✓	3.781	✓	9.764
go_no_go_11_dl	22	✗	✗	3.833	✗	21.875
go_no_go_15	30	✓	✓	4.162	✓	11.087
go_no_go_15_dl	30	✗	✗	5.134	✗	28.036
go_no_go_23	46	✓	✓	12.925	✓	33.465
go_no_go_23_dl	46	✗	✗	13.887	✗	69.259
go_no_go_31	62	✓	✓	18.265	✓	43.432
go_no_go_31_dl	62	✗	✗	15.894	✗	91.137
go_no_go_63	126	✓	✓	67.812	✓	178.331
go_no_go_63_dl	126	✗	✗	65.104	✗	293.121
power_1_5	25	✓	✗	2.453	✓	126.708
power_1_5_dl	25	✗	✗	2.323	✗	55.676
power_2_5	51	✓	✗	7.714	✓	375.586
power_2_5_dl	51	✗	✗	7.683	✗	283.736
power_3_5	77	✓	✗	16.691	✓	880.156
power_3_5_dl	77	✗	✗	16.713	✗	567.414
power_4_5	103	✓	✗	30.156	✓	1830.834
power_4_5_dl	103	✗	✗	35.731	✗	879.080
power_5_5	129	✓	✗	47.905	✓	4027.134
power_5_5_dl	129	✗	✗	54.214	✗	1937.699
power_6_5	155	✓	✗	73.473	✓	7204.605
power_6_5_dl	155	✗	✗	67.092	✗	2293.866
power_7_5	181	✓	✗	93.207	✓	10780.738
power_7_5_dl	181	✗	✗	149.617	✗	4245.406

Table 1: Experimental results

7.2 Results

The times required for the experiments are included in Table 1. The *Model* column indicates the model that is evaluated, *#FSMs* reports the number of FSMs in the model. In the *Deadlock free* column, ✓ indicates that the model is deadlock free, ✗ indicates the model has a deadlock. For SMT and reachability (using nuXmv), we report the *Result*, where ✓ indicates the tool reports the model is deadlock free, and ✗ indicates the tool reports the model contains a deadlock. For each of the instances we also report the running time (in seconds).

The numbers we used in the model names have the following meaning. In case of the "go/no go" models, the number signifies the number of "go/no go" FSM pairs. In case of the power domains models, the first number denotes the number of domains, and the second number denotes the number of device-controller pairs in every power domain.

For "go/no go" models, both SMT and reachability report that deadlock free models are indeed deadlock free, and in models with a deadlock indeed a deadlock is detected. The largest deadlock free "go/no go" model contains 126 FSMs, and is proven to be live using SMT in 1 minute and 7 seconds. Reachability analysis takes almost 3 minutes for the same model. The largest "go/no go" model with a deadlock also contains 126 FSMs. Using SMT, a deadlock is reported in 1 minute and 5 seconds. Using reachability it can be proven that a deadlock state is reachable in 4 minutes 53 seconds.

For power domain models, SMT always reports deadlocks, whereas reachability correctly produces an output that is consistent with the expected result. The largest deadlock-free power domain model contains 181 FSM. SMT reports a possible deadlock, and the reachability check shows that there are no reachable deadlock states in approximately 3 hours. The largest power domain model with a deadlock contains 181 FSM, for which SMT reports a deadlock in 2 minutes and 29 seconds. Using reachability, existence of a deadlock state was proven in 1 hour and 10 minutes.

7.3 Discussion

The performance results show that using SMT for liveness verification outperforms reachability for xMAS extended with FSMs. This is in line with our expectations, and also aligns with results for standard xMAS [12]. The experiments also show that for some deadlock-free models, SMT-based verification reports false deadlocks. This comes from the incompleteness of the SMT-based method.

From the results we conclude that both SMT-based verification and the reachability perform well. To compare the scalability of both, we split the experimental data into four groups: "go/no go" deadlock-free, "go/no go" deadlock, power domain deadlock-free, and power domain deadlock models. In Figure 12, we visualize the performance results for these groups of models using line graphs. Note, that for the y-axis, we use a logarithmic scale. From the visualization, we conclude, that both the SMT-based method and the reachability analysis scale exponentially, but SMT is significantly faster.

Although false deadlocks may be reported using the SMT encoding, it serves as a good preprocessing step for full reachability. In case no deadlocks are reported, one can be sure the model is deadlock free, and the verification terminates quickly. Full reachability only needs to be computed for models for which deadlocks are reported.

8 Conclusions

We demonstrated that the approach to verify liveness of xMAS networks with finite state machines proposed by Verbeek *et al.* [19] is unsound. We proposed an alternative method to prove liveness of xMAS networks containing FSMs. Furthermore, we proved our method is sound. We evaluated our technique on a set of benchmarks, and show that it performs well compared to reachability using nuXmv. The method is incomplete in the sense that the method can report deadlocks for models that are deadlock free. This was also observed for some of the examples in our experiments. For

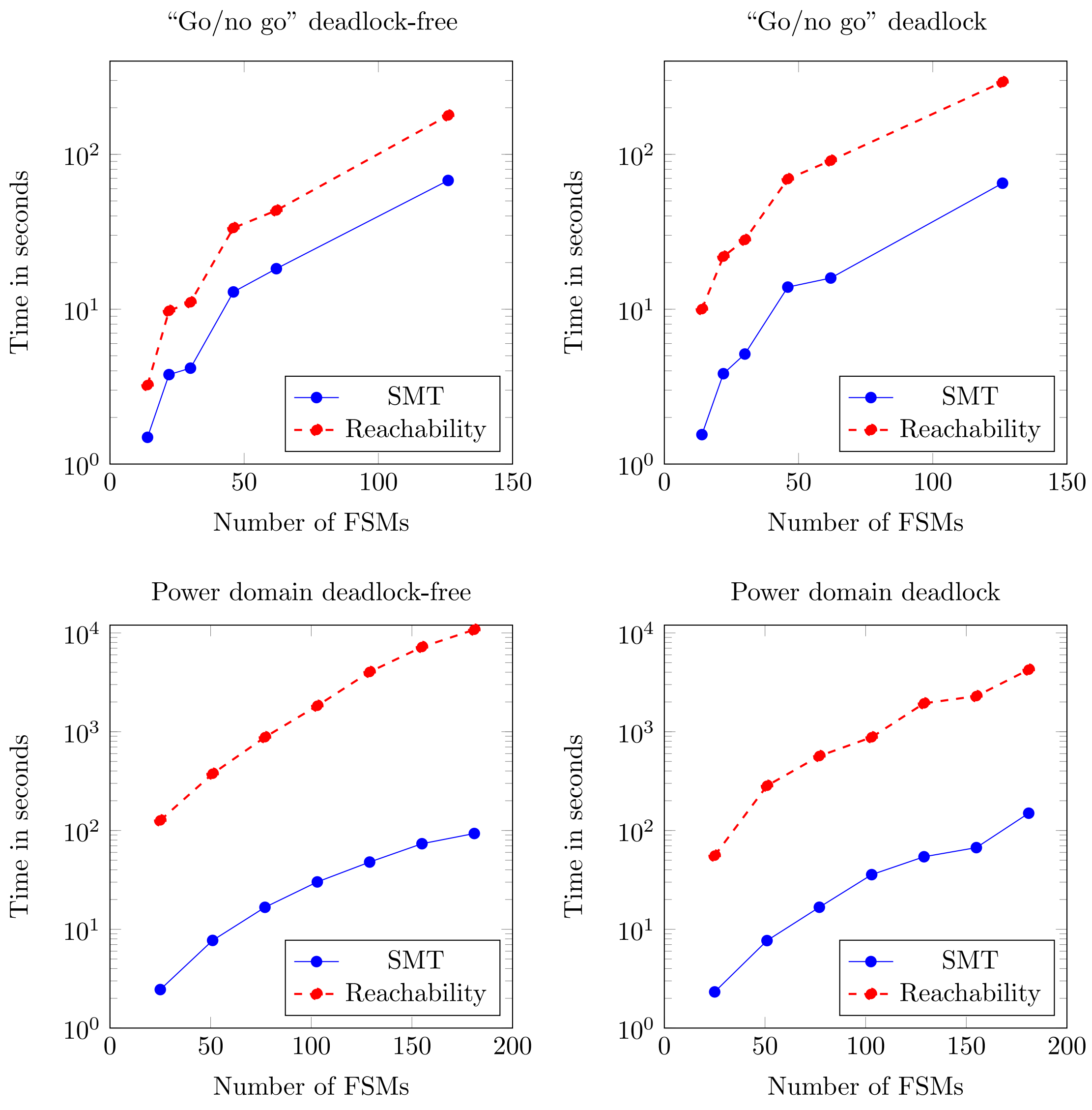


Figure 12: Visualization of the results.

cases where a possible deadlock is reported, but it is not known if the deadlock state is reachable or not, we showed that the reachability analysis can be done reasonably efficiently using a symbolic model checker.

Future work. As future work it should be investigated whether more false deadlocks can be eliminated by deriving stronger invariants. Also, investigating an alternative encoding to SMT by including reachability, inspired by bounded model checking, could make the method complete provided an appropriate bound can be derived. Additionally, the finite state machines presented in this paper always read from and write to exactly one channel. This restriction could be relaxed to read and write multiple channels on a single transition. This enables more compact modeling of some finite state machines; for instance, the go/no go FSM in Figure 8 could be updated to simultaneously read from i_1 and i_2.

References

[1] MaDL design and verification tools. URL `https://github.com/MaDL-DVT/madl-dvt`.

[2] C. Baier and J.-P. Katoen. *Principles of Model Checking.* The MIT Press, 2008. ISBN 978-0-262-02649-9.

[3] F. Burns, D. Sokolov, and A. Yakovlev. GALS synthesis and verification for xMAS models. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, DATE '15, pages 1419–1424, San Jose, CA, USA, 2015. EDA Consortium. ISBN 978-3-9815370-4-8. doi: 10.7873/DATE.2015.0063.

[4] R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, and S. Tonetta. The nuXmv symbolic model checker. In *Computer Aided Verification*, pages 334–342, 2014. doi: 10.1007/978-3-319-08867-9_22.

[5] S. Chatterjee and M. Kishinevsky. Automatic generation of inductive invariants from high-level microarchitectural models of communication fabrics. In *Computer Aided Verification*, pages 321–338. Springer, Berlin, Heidelberg, 2010. doi: 10.1007/978-3-642-14295-6_29.

[6] S. Chatterjee and M. Kishinevsky. Automatic generation of inductive invariants from high-level microarchitectural models of communication fabrics. *Formal Methods in System Design*, 40(2):147–169, 2012. doi: 10.1007/s10703-011-0134-0.

[7] S. Chatterjee, M. Kishinevsky, and U. Y. Ogras. xMAS: Quick formal modeling of communication fabrics to enable verification. *IEEE Design Test of Computers*, 29(3):80–88, 2012. doi: 10.1109/MDT.2012.2205998.

[8] S. Das, C. Karfa, and S. Biswas. xMAS based accurate modeling and progress verification of NoCs. In *VLSI Design and Test*, pages 792–804. Springer, Singapore, 2017. doi: 10.1007/978-981-10-7470-7_74.

[9] L. de Moura and N. Bjørner. Z3: An efficient SMT solver. In C. R. Ramakrishnan and J. Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, LNCS, pages 337–340, Berlin, Heidelberg, 2008. Springer. ISBN 978-3-540-78800-3. doi: 10.1007/978-3-540-78800-3_24.

[10] A. Fedotov and J. Schmaltz. Automatic generation of hardware checkers from formal micro-architectural specifications. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1568–1573, 2018. doi: 10.23919/DATE.2018.8342265.

[11] B. van Gastel, F. Verbeek, and J. Schmaltz. Inference of channel types in micro-architectural models of on-chip communication networks. In *2014 22nd International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 1–6, 2014. doi: 10.1109/VLSI-SoC.2014.7004168.

[12] A. Gotmanov, S. Chatterjee, and M. Kishinevsky. Verifying deadlock-freedom of communication fabrics. In *Verification, Model Checking, and Abstract Interpretation*, pages 214–231. Springer, Berlin, Heidelberg, 2011. doi: 10.1007/978-3-642-18275-4_16.

[13] S. J. C. Joosten and J. Schmaltz. Generation of inductive invariants from register transfer level designs of communication fabrics. In *2013 Eleventh ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE 2013)*, pages 57–64, 2013.

[14] S. J. C. Joosten and J. Schmaltz. Automatic extraction of micro-architectural models of communication fabrics from register transfer level designs. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pages 1413–1418, 2015. doi: 10.7873/DATE.2015.0616.

[15] Z. Lu and X. Zhao. xMAS-based QoS analysis methodology. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(2):364–377, 2018. doi: 10.1109/TCAD.2017.2706561.

[16] F. Verbeek and J. Schmaltz. Hunting deadlocks efficiently in microarchitectural models of communication fabrics. In *2011 Formal Methods in Computer-Aided Design (FMCAD)*, pages 223–231, 2011.

[17] F. Verbeek and J. Schmaltz. Automatic generation of deadlock detection algorithms for a family of microarchitecture description languages of communication fabrics. In *2012 IEEE International High Level Design Validation and Test Workshop (HLDVT)*, pages 25–32, 2012. doi: 10.1109/HLDVT.2012.6418239.

[18] F. Verbeek, P. M. Yaghini, A. Eghbal, and N. Bagherzadeh. ADVOCAT: Automated deadlock verification for on-chip cache coherence and interconnects. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1640–1645, 2016.

[19] F. Verbeek, P. M. Yaghini, A. Eghbal, and N. Bagherzadeh. Deadlock verification of cache coherence protocols and communication fabrics. *IEEE Transactions on Computers*, 66(2): 272–284, 2017. doi: 10.1109/TC.2016.2584060.

[20] S. Wouda, S. J. C. Joosten, and J. Schmaltz. Process algebra semantics & reachability analysis for micro-architectural models of communication fabrics. In *Proceedings of the 2015 ACM/IEEE International Conference on Formal Methods and Models for Codesign*, MEMOCODE '15, pages 198–207, Washington, DC, USA, 2015. IEEE Computer Society. doi: 10.1109/MEMCOD.2015.7340487.

14/01	Jan Friso Groote, Remco van der Hofstad and Matthias Raffelsieper	On the Random Structure of Behavioural Transition Systems
14/02	Maurice H. ter Beek and Erik P. de Vink	Using mCRL2 for the analysis of software product lines
14/03	Frank Peeters, Ion Barosan, Tao Yue and Alexander Serebrenik	A Modeling Environment Supporting the Co-evolution of User Requirements and Design
14/04	Jan Friso Groote and Hans Zantema	A probabilistic analysis of the Game of the Goose
14/05	Hrishikesh Salunkhe, Orlando Moreira and Kees van Berkel	Buffer Allocation for Real-Time Streaming on a Multi-Processor without Back-Pressure
14/06	D. Bera, K.M. van Hee and H. Nijmeijer	Relationship between Simulink and Petri nets
14/07	Reinder J. Bril and Jinkyu Lee	CRTS 2014 - Proceedings of the 7th International Workshop on Compositional Theory and Technology for Real-Time Embedded Systems
14/08	Fatih Turkmen, Jerry den Hartog, Silvio Ranise and Nicola Zannone	Analysis of XACML Policies with SMT
14/09	Ana-Maria Şutîi, Tom Verhoeff and M.G.J. van den Brand	Ontologies in domain specific languages – A systematic literature review
14/10	M. Stolikj, T.M.M. Meyfroyt, P.J.L. Cuijpers and J.J. Lukkien	Improving the Performance of Trickle-Based Data Dissemination in Low-Power Networks
15/01	Önder Babur, Tom Verhoeff and Mark van den Brand	Multiphysics and Multiscale Software Frameworks: An Annotated Bibliography
15/02	Various	Proceedings of the First International Workshop on Investigating Dataflow In Embedded computing Architectures (IDEA 2015)
15/03	Hrishikesh Salunkhe, Alok Lele, Orlando Moreira and Kees van Berkel	Buffer Allocation for Realtime Streaming Applications Running on a Multi-processor without Back-pressure
15/04	J.G.M. Mengerink, R.R.H. Schiffelers, A. Serebrenik, M.G.J. van den Brand	Evolution Specification Evaluation in Industrial MDSE Ecosystems
15/05	Sarmen Keshishzadeh and Jan Friso Groote	Exact Real Arithmetic with Pertubation Analysis and Proof of Correctness
15/06	Jan Friso Groote and Anton Wijs	An $O(m \log n)$ Algorithm for Stuttering Equivalence and Branching Bisimulation
17/01	Ammar Osaiweran, Jelena Marincic Jan Friso Groote	Assessing the quality of tabular state machines through metrics
17/02	J.F. Groote and e.P. de Vink	Problem solving using process algebra considered insightful
18/01	L. Sanchez, W. Wesselink and T.A.C. Willemse	BDD-Based Parity Game Solving: A comparison of Zielonka's Recursive Algorithm, Priority Promotion and Fixpoint Iteration
18/02	Mahmoud Talebi	First-order Closure Approximations for Middle-Sized Systems with Non-linear Rates
18/03	Thomas Neele, Tim A.C. Willemse and Jan Friso Groote	Solving Parameterised Boolean Equation Systems with Infinite Data Through Quotienting (Technical Report)
18/04	Daan Leermakers, Behrooz Razeghi, Shideh Rezaeifar, Boris Škorić, Olga Taran Slava Voloshynovskiy	Optical PUF statistics
19/01	Maurice Laveaux, Jan Friso Groote and Tim A.C. Willemse	Correct and Efficient Antichain Algorithms for Refinement Checking
19/02	Thomas Neel, Tim A.C. Willemse and Wieger Wesselink	Partial-Order Reduction for Parity Games with an Application on Parameterised Boolean Equation Systems (Technical Report)
19/03	David N. Jansen, Jan Friso Groote, Jeroen J.A. Keiren and Anton Wijs	A simpler $O (m \log n)$ algorithm for branching bisimilarity on labelled transition systems
19/04	Alexander Fedotov, Jeroen J.A. Keiren and Julien Schmaltz	Sound idle and block equations for finite state machines in xMAS