## Verification Techniques for xMAS

## Citation for published version (APA):

Fedotov, A. (2022). Verification Techniques for xMAS. [Phd Thesis 1 (Research TU/e / Graduation TU/e), Mathematics and Computer Science]. Eindhoven University of Technology.

## Document status and date:

Published: 11/01/2022

## Document Version:

Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

## Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.
Link to publication


## General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25 fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

## Take down policy

If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.

# Verification Techniques for xMAS 

## PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus Prof.dr.ir. F.P.T. (Frank) Baaijens, voor een commissie aangewezen door het College voor Promoties, in het openbaar te verdedigen op dinsdag 11 Januari 2022 om 16:00 uur

door

Alexander Fedotov
geboren te Perm, Rusland

Dit proefschrift is goedgekeurd door de promotiecommissie:
voorzitter:
$1^{\mathrm{e}}$ promotor: copromotor(en): leden:
prof.dr. J.J. Lukkien
prof.dr.ir. J.F. Groote
dr.ir. J.J.A. Keiren
prof.dr.ir. J.P.M. Voeten
dr.ir. M.A. Reniers
dr. J.A. Pérez
prof.dr. M.G.J. van den Brand dr. J. Schmaltz

Het onderzoek of ontwerp dat in dit proefschrift wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.

# Verification Techniques for xMAS 

Alexander Fedotov

Copyright © 2021 by Alexander Fedotov
All rights reserved. Reproduction in whole or in part is prohibited without the written consent of the copyright owner.

IPA Dissertation Series 2022-01
A catalogue record is available from the Eindhoven University of Technology Library ISBN: 978-90-386-5432-4

Typeset with $\mathrm{EAT}_{\mathrm{E}} \mathrm{X}$ ( $\mathrm{T}_{\mathrm{E}} \mathrm{XLive} 2021$ )
Cover design by Vika Baida
Printed by Ipskamp Printing


The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics) and funded by NWO (Dutch Research Council) under grant no. 612.001.302. The author was employed at the Eindhoven University of Technology.

## Contents

Preface ..... iii
1 Introduction ..... 1
1.1 xMAS ..... 2
1.2 Research questions ..... 4
1.3 Contributions ..... 5
1.4 Related Work ..... 6
2 Semantics of xMAS ..... 9
2.1 Introduction ..... 9
2.2 Preliminaries ..... 10
2.3 Semantics of xMAS Networks ..... 13
2.4 Kripke Structure Semantics ..... 25
2.5 Correctness of the KS Semantics ..... 30
2.6 Conclusion ..... 42
3 Effective System Level Liveness Verification ..... 43
3.1 Introduction ..... 43
3.2 Preliminaries ..... 44
3.3 Life and Death of State Machines in xMAS ..... 53
3.4 Finite State Machines ..... 55
3.5 Idle and Block Equations for FSMs ..... 60
3.6 Invariants ..... 72
3.7 Experiments ..... 74
3.8 Conclusions ..... 79
4 Eliminating False Deadlocks in xMAS ..... 81
4.1 Introduction ..... 81
4.2 Preliminaries ..... 83
$4.3 \quad k$-step Backward Reachability ..... 89
4.4 Interpolation-Based Backward Reachability ..... 99
4.5 Discussion ..... 116
4.6 Conclusion ..... 118
5 Automatic Generation of Hardware Checkers from Formal Micro-architecturalSpecifications119
5.1 Introduction ..... 119
5.2 Background and Overview ..... 120
5.3 Interface Behaviour ..... 123
5.4 SystemVerilog Checkers ..... 128
5.5 Experimental Results and Discussion ..... 130
5.6 Conclusion ..... 132
6 Conclusion ..... 133
6.1 Summary ..... 133
6.2 Future Work ..... 134
Bibliography ..... 137
Index ..... 143
Summary ..... 143
Samenvatting ..... 145
Curriculum Vitae ..... 147

## Preface

When I, as a kid, was being asked who I wanted to become when I grow up, I was always answering without any doubt - a researcher (and I must say that it was an unusual answer among my peers). There is no coincidence, my parents are mathematicians, and I always looked at them with admiration. Thanks to them I like computers, mathematics and have the aspiration to discover new things.

My interest in formal methods and especially in solving problems using SMT appeared thanks to Hans Zantema, whose master's course Automated Reasoning inspired me. After writing my master's thesis under his supervision, I finally decided that I would like to do a Ph.D. in that field. Thanks to Julien Schmaltz, I got interested in applying formal methods to the domain of hardware. I want to thank him for setting up this interesting project and guiding me through the first two years of my Ph.D. Thanks to my co-promotor Jeroen Keiren and my promotor Jan Friso Groote, I am at the point of finishing my thesis. It was a great time working under their supervision. Thanks to my involvement in teaching activities (during my employment as a Ph.D. candidate, I was involved in three courses), I overcame my lack of confidence in giving public speeches. And finally, thanks to my former and current roommates and fellow Ph.D. students, I have not lost the positive attitude and motivation, despite all the challenges.

My special thanks go to NWO (Dutch Research Council) for supporting the project under grant no. 612.001.302.

## Chapter 1

## Introduction

Computers and, in a broad sense, electronic hardware play a crucial role in modern society. Correct functioning of electronic hardware is important as flaws can be costly and even unacceptable in safety-critical areas such as aircraft and nuclear plants. Designing new hardware usually starts from a high-level specification, which undergoes refinement and ultimately crystallizes in a final design. A new hardware design needs to undergo validation to make sure that it meets expectations in terms of desired functionality. Conventionally, validation of a new hardware design is done using simulation and testing [KG02].

Traditional means of hardware design validation have shortcomings. Testing and simulation of sophisticated designs often cannot offer exhaustive coverage, which only gives probabilistic assurance that the design is correct. Technologies evolve fast, and hardware is becoming more complex, which is making exhaustive coverage less and less feasible.

Formal verification becomes more widely adopted in the industry [KG02], aiming to address the shortcomings of the traditional approaches to hardware validation. Formal verification is, in essence, the use of the formal methods of mathematics to prove that an abstract representation of the object of interest conforms to the desired properties expressed formally. In contrast to simulation and testing, proving that the desired properties hold for the hardware design guarantees that all possible behaviors of the hardware are correct.

There are two main approaches to formally verifying hardware: property verification and implementation verification [KG02]. In property verification, the desired functionality is formulated in the form of temporal logic properties, which are then proved for an abstract representation of the design under verification. The use of temporal logic as a framework for specifying properties of hardware designs allows describing the desired behavior. In implementation verification, an implementation is made according to the specification. Ultimately, implementation verification establishes a formal relation between the specification and the implementation to verify that the implementation
is correct.
In practice, both property and implementation verification co-exist - a new hardware design is first verified using property verification; implementation verification takes place during further refinement iterations of the design so that each iteration uses a new implementation [KG02]. When both verification approaches are applied in combination, the properties formulated within the property verification phase have to hold for all the subsequent implementations of the design [KG02].

Unfortunately, formal verification is not a silver bullet. It is a challenge to scale the approach to the system level (this is usually related to the so-called "state space explosion problem"), which is one of the main reasons why the conventional means for hardware design validation are not yet superseded by formal verification.

## 1.1 xMAS

xMAS is a graphical language aimed at modeling and verification of hardware, introduced originally by researchers at Intel [CKO10]. The language has a compact set of easy-to-understand primitives. By composing these primitives, one can build hardware designs that cover many possible design scenarios. The composition of primitives in xMAS is made simple. It is enough to connect the output of one primitive to the input of another without the need of thinking of glue logic. Henceforth we will be using the term $x M A S$ networks to refer to hardware designs expressed in the xMAS language.


Figure 1.1: Core xMAS primitives [CKO10].
Figure 1.1 depicts the core primitives of xMAS. For now, we consider a high-level explanation of every primitive; more details will be given further when it is required. The queue primitive acts as a buffer for data packets. Queues in xMAS are parameterized by the size and act according to the "first in, first out" principle. The function primitive is used to transform data according to a pre-defined data transformation function $f$. The source primitive generates data packets; the decision to stay idle or to transfer a certain packet through output $o$ is taken non-deterministically during every clock cycle. The assumption is that data packets are generated infinitely often. The sink primitive uses its input $i$ to accept data packets. Analogously to the source, the assumption is that it accepts data infinitely often. The fork primitive has two pre-defined data transformation functions: $f$ and $g$; the primitive takes a data packet $d$ from its input $i$, and immediately transfers $f(d)$ through $a$ and $g(d)$ through $b$. The join primitive has a predefined binary data-transforming function $h$; the primitive simultaneously takes data packets $d$ and $e$ from its inputs $a$ and $b$, and immediately
transfers $h(d, e)$ through output $o$. The switch primitive is used for routing data whenever a data packet comes to the input $i$ of the switch, a (user-defined) arbitration function $r$ is used to decide which of the two outputs is used for the propagation of the packet. Finally, the merge primitive is used for data arbitration. Whenever there are data packets coming to both inputs $a$ and $b$ simultaneously, the merge uses a (user-defined) arbitration policy to give priority to one of the packets. For verification purposes, the merge can be assumed to be non-deterministic; that covers all possible arbitration policies.


Figure 1.2: Connecting primitives in xMAS .
To explain the connection between primitives and the concept behind data transfers, let us refer to Figure 1.2. In the figure, we have two primitives $A$ and $B$ connected using channel $x$. We call primitive $A$ the initiator and primitive $B$ the target. We use three arrows to depict channel $x$ on purpose since every channel in xMAS carries three signals - irdy, trdy, and data; irdy is used by the initiator to indicate that its ready to transfer, trdy is used by the target to indicate the readiness to accept a transfer, and data carries data from the initiator to the target. The three signals implement the hand-shake principle - data is only transferred if both the initiator and the target are ready for the transfer. The channels in xMAS are assumed to be persistent in terms of data transfers. That is, whenever irdy of a particular channel becomes set, it remains set, and the value of data of the channel stays the same until trdy of the channel becomes set and data is transferred successfully.

Intuitively, the semantics of xMAS can be explained as follows. We split an xMAS network into two parts - let the contents of all queues be the sequential part of the network and let irdy, trdy, and data signals of all channels be the combinatorial part of the network. Initially, all queues are empty. In every clock cycle, first, the combinatorial part gets updated, then the sequential part. More details on the xMAS semantics will be provided further as needed.


Figure 1.3: xMAS example
Example 1.1. In Figure 1.3, we show an xMAS network consisting of a source, a queue of size one, and a sink. For simplicity, we assume that the source can only generate simple tokens (that is, the data type of the source consists of a single element). Hence we can safely omit the values of data. For better understanding of how xMAS works,
we walk the reader through a possible execution of the network. We consider three states starting from an initial state.

1. Execution starts in a state where the queue is empty, the source is ready to generate a token, and the sink is ready to accept data. In that case, we have $x$.irdy since the source is ready to produce data, $x$.trdy since the queue is empty, $\neg$ y.irdy since the queue is empty and therefore has no data to transfer, and $y$.trdy since the sink is ready to accept data.
2. The queue contains a token since a data transfer happened from the source to the queue, the source decides to stay idle, and the sink is ready to accept data. We have $\neg x$.irdy as the source does not attempt to transfer data, $\neg x$.trdy as the queue is full and cannot accept more packets, y.irdy as the queue has data to transfer further, and $y$.trdy as the sink is ready to accept data.
3. The queue became empty since a data transfer happened from the queue to the sink, the source decides to stay idle again, and the sink decides not to accept data. In that case, we have $\neg x$.irdy, $x$.trdy, $\neg y$.irdy, and $\neg y$.trdy.

### 1.2 Research questions

The state-based perspective on xMAS is important due to several factors. Firstly, the original authors of the xMAS language define various properties of xMAS networks [GCK11] using Linear Temporal Logic (LTL) [BK08]; by its nature, LTL assumes a state-based setting. In addition, Wouda et al. introduced a verification technique, which relies on symbolic model checking [BK08] to analyze the reachability of deadlock states in a given xMAS network [WJS15]. The technique by Wouda et al. implies a state-based representation of xMAS. Despite its importance, there is no state-based formalization of $x \mathrm{MAS}$ in the literature; this raises the following question.

RQ1. What is the state-based semantics of $x M A S$ ?
Since its introduction, xMAS was extended with a Finite State Machine primitive by Verbeek et al. [Ver+17]. The FSM extension made xMAS more expressive. The authors of the extension demonstrated simultaneous verification of cache coherence protocols and the communication fabric. In their experiments, the authors assumed random access buffers instead of conventional for xMAS first-in-first-out queues. In that regard, it is interesting to find out whether xMAS with the FSM extension can tackle verification at the system level ${ }^{1}$ in a more generic setting.

RQ2. Is $x M A S$ with the FSM extension suitable for the verification at the system level?

The authors of $x$ MAS introduced an efficient technique based on SAT-solving [Cla+08; BHM21]; given an xMAS network, the technique translates the liveness problem of the network into a boolean satisfiability problem [GCK11; CKO12]. By the nature

[^0]of the technique, the absence of a satisfying assignment to the satisfiability problem guarantees the absence of deadlocks. However, a satisfying assignment might yield a false deadlock that is not reachable. The SAT-based verification technique for xMAS networks is not prone to false deadlocks only if the network under verification is acyclic and made of the core xMAS components [CK10]. Hence, the following research question arises.

RQ3. How can we improve the SAT-based verification technique so that it is not prone to false deadlocks in the case of $x M A S$ networks with FSMs?

As a high-level modeling language, xMAS can be used for specifying new hardware designs. However, there is a gap between high-level specifications expressed in xMAS and Register Transfer Level (RTL) ${ }^{2}$ implementations. The literature lacks methodologies for verifying the correctness of RTL implementations built according to the corresponding xMAS specifications. Naturally, there is the following research question.

RQ4. How can we verify the correctness of RTL implementations with respect to the $x M A S$ specifications?

### 1.3 Contributions

Further we focus on the contributions of the thesis which are grouped into chapters. Each of the chapters is mostly self-contained.

In Chapter 2, we answer RQ1 by precisely formulating the semantics of the xMAS language in terms of Kripke Structures [BK08] and proving its correctness with respect to the semantics in terms of irdy, trdy, and data signals. The formal semantics for xMAS lays the theoretical foundation for the subsequent work related to xMAS liveness verification techniques.

Looking for an answer for RQ2, we found out that the liveness verification technique for the xMAS FSM extension proposed by Verbeek et al. [Ver+17] is unsound. In Chapter 3, we provide a counter-example for the technique by Verbeek et al. We propose an alternative liveness verification approach for xMAS with FSMs, and prove that it is sound. We also show that our approach allows to verify liveness at the system level. The chapter is based on the work presented in a technical report Sound idle and block equations for finite state machines in $x M A S$ by Fedotov et al. [FKS19] and in a conference paper Effective System Level Liveness Verification by Fedotov et al. [FKS20].

Our liveness verification technique for xMAS with FSMs, introduced in Chapter 3 is not complete, i.e., it can report deadlock situations, which are unreachable. In Chapter 4, we propose two approaches to make the technique complete and evaluate their scalability; this answers RQ3.

[^1]In Chapter 5, we introduce a method which, given an xMAS network and a corresponding RTL implementation, automatically turns the xMAS network into an RTL specification, and further, the method checks the correctness of the implementation by verifying that the set of traces produced by the implementation is included in the set of traces produced by the specification. The chapter is based on the work presented in a conference paper Automatic generation of hardware checkers from formal micro-architectural specifications by Fedotov and Schmaltz [FS18]. The work presented in the chapter gives an answer to RQ4.

### 1.4 Related Work

### 1.4.1 Hardware modeling and verification

Describing hardware at the level of logic gates is cumbersome, especially with the modern hardware complexity. Hardware Description Languages (HDLs) were introduced to make the process of describing hardware more convenient. HDLs represent hardware using a Register Transfer Level abstraction. The two most widely-used HDLs are SystemVerilog [Tar20] and VHDL ${ }^{3}$ [HH12]. An HDL description of hardware can be simulated and synthesized. Simulation is conducted by supplying data to the HDL description and monitoring the outputs of the simulation; the aim of the simulation of HDL is to reveal potential design errors. Synthesis of HDL yields a description of the hardware in terms of logic gates; an HDL synthesizer might also optimize the HDL description [HH12]. Methodologies for formal verification of both SystemVerilog and VHDL are presented in the literature. In particular, SystemVerilog models can be verified using nu $X_{\text {mv }}$ model-checker [Irf+16] and CADP toolbox [Bou+18]; verification of VHDL models can be tackled using Cadence SMV model-checker[KRŠ06] and CV toolset [DSC98]. In comparison to xMAS, HDLs describe hardware at a lower level. While xMAS is tailored for specifying hardware designs, HDLs are more suitable for describing hardware implementations.

Besides low-level textual languages such as SystemVerilog and VHDL there exists a high-level graphical language SCADE, which is now widely used in safety-critical areas such as avionics and railways. Although the primary focus of the language is to model, simulate, and verify control software for embedded systems, the language can also be adapted for high-level modeling and verification of hardware. SCADE combines synchronous data-flow and Finite State Machines; data-flow can be controlled by means of merges and user-defined nodes [Ber07]. There is a precise formalization of the semantics of SCADE in the literature [Ber07]. SCADE is similar to xMAS in the sense that SCADE models consist of block diagrams that communicate synchronously using channels. In the context of hardware, xMAS stands out by the fact that it contains a compact set of primitives that are made specifically for hardware modeling. In addition, xMAS has an efficient verification method that translates the source model into a SAT-problem; SCADE lacks an analogous methodology.

[^2]
### 1.4.2 xMAS

Besides the techniques by the authors of xMAS and Wouda et al. that we already mentioned in the research questions subsection, there exist more verification techniques associated with xMAS. In addition, Verbeek and Schmaltz developed an algorithm based on wait-for graphs [SGG08] for xMAS liveness verification [VS11]. Chatterjee et al. developed a method that involves using a model checker that takes Register Transfer Level (RTL) specifications as input to verify several classes of properties on xMAS networks [CK10]. Van Gastel and Schmaltz formalized a subset of xMAS in ACL2 and developed a method to check certain properties on xMAS networks, such as non-emptiness of routing and correctness of progress conditions [GS13].

We already mentioned an FSM extension by Verbeek et al. [Ver+17] in the research questions subsection. Besides that, there are more extensions to xMAS. Burns et al. extended xMAS with support of synthesis and verification of Globally Asynchronous Locally Synchronous (GALS) architectures [BSY15]. The extension by Burns et al. involves a new synchronization primitive and a translation of xMAS into Circuit Petri nets [YGL00].

Joosten and Schmaltz bridged the gap between Register Transfer Level specifications expressed in such languages as SystemVerilog [Tar20] and abstract models [JS15]. They introduced a method that extracts an xMAS network automatically from a given RTL specification. The method is beneficial since high-level models are often more accessible to verification than low-level RTL specifications. In addition, highlevel models can provide invariants that are useful for the verification at the RTL level.

Das et al. developed a methodology to translate a subset of xMAS into Finite State Machines [DKB17]. It is worth noting that the work of Das et al. does not support queues. In their work, Das et al. propose to turn the FSM obtained from a given xMAS network into an SMV model manually. A symbolic model checker can be used on the SMV model to verify data progress properties of the xMAS model.

Zhao and Lu introduced a method to analyze the per-flow delay bound property on xMAS networks [ZL13]. The analysis is conducted by mapping a given xMAS network to its network calculus [Cru91a; Cru91b] model.
Verbeek and Schmaltz generalized xMAS to a family of Micro-architecture Description Languages [VS12]; the family of languages gives the ability to extend the set of basic xMAS primitives by describing new custom ones. Verbeek and Schmaltz also developed a methodology to derive liveness verification algorithms automatically for models built using custom primitives [VS12].

## Chapter 2

## Semantics of xMAS

### 2.1 Introduction

Hardware design starts with a high-level specification, which is usually made using block diagrams and accompanied with informal text describing the desired functionality. Subsequently, the high-level design undergoes refinement until a final design is obtained. Validation of the design consists of checking the conformance of the design to the desired specification. Conventionally, this is tackled by simulation and testing. However, growing complexity of hardware designs makes coverage of simulation and testing less and less complete. In this regard, formal verification becomes more and more adopted as a measure to address the above mentioned challenges [KG02]. However, scaling formal verification to the system level remains a challenge.


Figure 2.1: xMAS primitives [CKO10].

To address some of the issues with modeling and verification of communication fabrics, Intel introduced the xMAS (eXecutable Microarchitectural Specification) language [CKO10]. xMAS is a graphical language for modeling and verification of microarchitectures. The language has a small set of well-understood primitives, see Figure 2.1, which is expressive enough to capture the functionality of most communication fabrics. The language is designed in such a way that it is not required to write complex "glue" logic in order to connect the primitives. One builds microarchitectural models by structurally composing the primitives of the language.

Since its introduction, xMAS has been efficiently used for modeling, verification
and quality of service analysis of microarchitectures. In [GCK11], Gotmanov et al. introduce a method which reduces liveness verification of xMAS to a satisfiability problem. In [HBS11; Hol+12], Holcomb et al. use xMAS for performance analysis of performance analysis of large scale networks-on-chips. In [ZL13], Zhao et al. analyze per-flow delay bounds of xMAS networks. In [GVS14], Gastel et al. infer automatically channel types in xMAS networks. In [BSY15], Burns et al. conduct Globally Synchronous Locally Asynchronous (GALS) synthesis and verification by extending and translating xMAS into Petri nets. Verbeek et al. add Finite State Machines to xMAS and use xMAS with FSMs to verify cache coherence protocols [Ver+17]. Joosten et al. infer xMAS networks from RTL specifications and conduct liveness verification [JS13; JS14].

Chatterjee et al. present a semantics of xMAS by describing dependencies between input and output signals of primitives [CKO10]. Wouda et al. formalise a process algebra semantics of xMAS [WJS15]. Note that process algebra semantics induces Labelled Transition System (LTS) semantics. Although there exists work on transforming Labelled Transition Systems to Kripke Structures [RSW12], this approach is indirect, and the resulting KSs would differ from the Kripke Structure representation of xMAS that is already assumed in symbolic model checking techniques associated with the xMAS language. A Kripke Structure semantics of the language was never formalized in the literature.

Contributions. First of all, we give a presentation of xMAS semantics in terms of signal structures. To make the presentation of the semantics by Chatterjee et al. more structured, we define the notion of signal state, which also allows us to define initial signal states, the transition relation between signal states, and the composition of signal structures. We also formalize the semantics of xMAS in terms of Kripke Structures. In addition, we carefully prove that the signal semantics and the Kripke Structure semantics are bisimilar.

Structure of the chapter. In Section 2.2, we introduce the necessary notation that we use throughout the chapter. In the same section, we also recall the xMAS language as it was introduced by Chatterjee et al. In Section 2.3, we define xMAS networks as mathematical objects. In the same section, we formalise a semantics of xMAS networks in terms of irdy, trdy, and data signals, define the notion of signal state, initial signal state and define the transition relation between signal states. In Section 2.4, we formalise a semantics of xMAS networks in terms of Kripke Structures. In Section 2.5, we prove that our Kripke Structure semantics for xMAS networks is bisimilar to the signal structure semantics defined in Section 2.3. Finally, Section 2.6 concludes the chapter.

### 2.2 Preliminaries

We first introduce some notation that is used in the rest of the chapter. We write $\mathbb{N}$ to denote the set of natural numbers, $\mathbb{B}=\{$ false, true $\}$ to denote the set of booleans.

Given a partial function $f: A_{1} \times \ldots \times A_{n} \rightarrow B$ and arguments $a_{1} \in A_{1}, \ldots, a_{n} \in A_{n}$, we write $f\left(a_{1}, \ldots, a_{n}\right)=\perp$ in case $f$ is undefined for $a_{1}, \ldots, a_{n}$. Given a set $D$, we write List(D) to denote the set of all lists of elements of $D$. Empty lists are denoted using []. Given a list xs, the length of xs is denoted by $|\mathrm{xs}|$. We write rhead(xs) to denote the last element of list xs . That is, given a list $\mathrm{xs}=\left[x_{0}, \ldots, x_{n}\right]$, we have $\operatorname{rhead}(\mathrm{xs})=x_{n}$. Note that $\operatorname{rhead}([])=\perp$. We write $\operatorname{rtail}(\mathrm{xs})$ to denote list xs without its last element. That is, given a list $\mathrm{xs}=\left[x_{0}, \ldots, x_{n}\right]$, we have $\operatorname{rtail}(\mathrm{xs})=\left[x_{0}, \ldots, x_{n-1}\right]$. Similarly to rhead, rtail is undefined for empty lists.

### 2.2.1 xMAS Networks

An xMAS network consists of a set of primitives, connected using channels. Each channel transfers three signals: boolean signals irdy and trdy, and a signal data. The signals are used to implement a handshake principle. That is, signals irdy and data are used to signal that the initiator of the data transfer is ready, while trdy is used to signalize that the target of the data transfer is ready. The data is transferred when both irdy and trdy are true. For primitives $A$ and $B$ connected using channel $x$, this is illustrated in Figure 2.2.


Figure 2.2: Connection between primitives in xMAS.
For an execution of an xMAS network, we assume a synchronous timing model, which means that at each clock cycle:

- every initiator sets values for the corresponding irdy and data,
- every target sets a value for trdy,
- data transfers happen simultaneously in all channels which are ready to transfer (i.e., channels whose irdy and trdy are set).


### 2.2.2 xMAS Primitives

The behavior of xMAS primitives is described in terms of irdy, trdy and data signals [CKO10]. Further we recall the description of the core xMAS primitives as they were introduced by Chatterjee et al.


At each clock cycle, a source non-deterministically tries to insert packet $d$ in output channel $o$. Formally, for a single datum $d$ it is defined as follows. Let oracle be an unconstrained binary input (i.e., the value to oracle is assigned non-deterministically every clock cycle), and pre be the standard synchronous operator that returns the value of its argument in the
previous clock cycle, and false in the very first cycle. Then, the values of irdy and data signals of channel $o$ are defined as follows:

$$
o . \text { irdy }:=\text { oracle } \vee \operatorname{pre}(o . \text { irdy } \wedge \neg o . \operatorname{trdy}), \quad o . \text { data }:=d .
$$



A sink non-deterministically tries to consume data from input channel $i$ at every clock cycle. Formally, the values of trdy signal of channel $i$ are defined as follows:

$$
i . \operatorname{trdy}:=\text { oracle } \vee \text { pre( } i . \operatorname{trdy} \wedge \neg i . \text { irdy }) .
$$



A queue is a FIFO-buffer parameterized with a size $k$. A queue is ready to write data to output channel $o$ when it is not empty. The queue is ready to write as data the contents of its head. A queue is ready to accept data from input channel $i$ when it is not full. Let xs be a list which represents the current contents of the queue. Then, the values of trdy signals of channel $i$ and the values of irdy and data signals of channel $o$ are defined as follows:

$$
o . \text { irdy }:=|\mathrm{xs}|>0, \quad \text { o.data }:=\left\{\begin{array}{ll}
\operatorname{rhead}(\mathrm{xs}) & \text { if }|\mathrm{xs}|>0 \\
\perp & \text { otherwise }
\end{array} \quad \text { i.trdy }:=|\mathrm{xs}|<k .\right.
$$



A function primitive is parameterized with a function $f$, that is used to transform all data that flows through the primitive. Let $i$ and $o$ be its input and output channels respectively, and let $f$ be its data transforming function. Then, the values of trdy signals of channel $i$ and the values of irdy and data signals of channel $o$ are defined as follows:

$$
o . \mathbf{i r d y}:=i . \mathbf{i r d y}, \quad o . \text { data }:=f(i . \text { data }), \quad i . \operatorname{trdy}:=o . \operatorname{trdy}
$$



A fork is parameterized with two data transforming functions: $f$ and $g$; it has one input $i$ and two outputs: $o_{0}$ and $o_{1}$. A fork reads a data packet $d$ at its input and writes $f(d)$ and $g(d)$ to $o_{0}$ and $o_{1}$ respectively. A data transfer only happens when the input and both outputs are ready. The values of trdy signals of channel $i$, and the values of irdy and data signals of channels $o_{0}$ and $o_{1}$ are defined as follows:

$$
\begin{array}{ll}
o_{0} . \mathbf{i r d y}:=i . \mathbf{i r d y} \wedge o_{1} . \text { trdy }, & o_{0} . \text { data }:=f(i . \text { data }), \\
o_{1} . \mathbf{i r d y}:=i . \mathbf{i r d y} \wedge o_{0} . \text { trdy }, & o_{1} . \text { data }:=g(i . \text { data }),
\end{array}
$$

$$
i . \operatorname{trdy}:=o_{0} \cdot \operatorname{trdy} \wedge o_{1} \cdot \mathbf{t r d y} .
$$



A join is the dual of a fork. It has two inputs: $i_{0}$ and $i_{1}$, and one output $o$; it is also parameterized by a binary data transforming function $h$. A join reads data packets $d$ and $e$ from inputs $i_{1}$ and $i_{2}$ simultaneously, and immediately writes $h(d, e)$ to output $o$. A data
transfer only happens, when the output and both inputs are ready. The values of trdy signals of channels $i$ and $i^{\prime}$, and the values of irdy and data signals of channel $o$ are defined as follows:

$$
\begin{array}{rlr}
o . \text { irdy }:=i_{0} . \mathbf{i r d y} \wedge i_{1} . \mathbf{i r d y}, & o . \text { data }:=h\left(i_{0} . \text { data }, i_{1} . \text { data }\right), \\
i_{0} . \operatorname{trdy}:=i_{1} . \mathbf{i r d y} \wedge o . \operatorname{trdy}, & i_{1} . \text { trdy }:=i_{0} . \text { irdy } \wedge o . \operatorname{trdy} .
\end{array}
$$



A switch routes packets in the network. It has one input $i$, and two outputs $o_{0}$ and $o_{1}$. A switch is parameterized by a routing boolean function $r$. Informally, the switch applies $r$ to a packet $d$ at its input, and it routes the packet to the first output if $r(d)$, and to the second output otherwise. Formally the values of trdy signals of channel $i$, and the values of irdy and data signals of channels $o_{0}$ and $o_{1}$ are defined as follows:

$$
\begin{array}{ll}
o_{0} . \mathbf{i r d y}:=i . \mathbf{i r d y} \wedge r(i . \text { data }), & o_{0} . \text { data }:=i . \text { data, } \\
o_{1} . \mathbf{i r d y}:=i . \mathbf{i r d y} \wedge \neg r(i . \text { data }), & o_{1} . \text { data }:=i . \text { data, }
\end{array}
$$

$$
i . \text { trdy }:=\left(o_{0} . \text {.irdy } \wedge o_{0} . \mathbf{t r d y}\right) \vee\left(o_{1} . \text {.irdy } \wedge o_{1} . \text { trdy }\right)
$$

A merge has two inputs $i_{0}$ and $i_{1}$, and one output $o$. It selects
 one packet among two competing packets. Let $u$ be an unconstrained input, that is used for choosing between the input channels. Without constraints imposed on $u$, we leave the arbitration non-deterministic. ${ }^{1}$ The values of trdy signals of channels $i_{0}$ and $i_{1}$, and the values of irdy and data signals of channel $o$ are defined as follows:

$$
\begin{array}{ll}
o . \text { irdy }:=\left(u \wedge i_{0} . \mathbf{i r d y}\right) \vee\left(\neg u \wedge i_{1} . \text { irdy }\right), & o . \text { data }:= \begin{cases}i_{0} . \text { data } & \text { if } u \wedge i_{0} . \text { irdy }, \\
i_{1} . \text { data } & \text { if } \neg u \wedge i_{1} . \text { irdy }, \\
\perp & \text { otherwise },\end{cases} \\
i_{0} . \operatorname{trdy}:=u \wedge o . \operatorname{trdy} \wedge i_{0} . \mathbf{i r d y}, & i_{1} . \operatorname{trdy}:=\neg u \wedge o . \operatorname{trdy} \wedge i_{1} . \text {.irdy } .
\end{array}
$$

Before moving on, let us consider an example of an xMAS network depicted in Figure 2.3. The network consists of a source, a queue, and a sink. The queue in the example can store up to one datum. The source is connected to the queue using channel $x$, channel $y$ connects the queue and the sink.

### 2.3 Semantics of xMAS Networks

In the previous section we described xMAS primitives and their behavior in terms of signals. We now formalize the structure of xMAS networks, and define semantics in terms of signals.

[^3]

Figure 2.3: xMAS network example.

### 2.3.1 xMAS Networks

We first formalise xMAS networks. The set of component types is

$$
\Gamma=\{\text { source, sink, function, fork, join, switch, merge }\} \cup\left\{\text { queue }_{k} \mid k \in \mathbb{N}\right\} .
$$

Note that the queue type is parameterized in order to reflect the sizes of queues.
Definition 2.1. An $x$ MAS network is a structure ( $P, G, C, c$, chan, type) where:

- $P$ is a non-empty set of components;
- $G$ is a non-empty set of channels;
- $C$ is a non-empty set of data, which consists of all possible values of data signals of all channels $x \in G$;
- $c: G \rightarrow\left(2^{C} \backslash\{\emptyset\}\right)$ is the function that assigns sets of data to channels from $G$;
- chan : $P \times\{$ in, out $\} \times \mathbb{N} \rightarrow G$ is a partial function which, given a component $p \in P$, an input/output identifier and a channel number $n \in \mathbb{N}$, returns the channel connected to input (output) number $n$ of component $p$;
- type : $P \rightarrow \Gamma$ assigns a type to a component.

Sets of data that are transferred through channels $g \in G$ can be computed using the data propagation algorithm defined by Wouda et al. [WJS15]. In case data is undefined for a certain channel $g \in G$, data $g$ can be overapproximated such that $c(g)=C$.

The semantics of an xMAS network is only well-defined if the network is valid.
Definition 2.2. Given an xMAS network $N=(P, G, C, c$, chan, type $)$, we say that $N$ is valid if and only if each of the following holds:

1. $|P|>0$, that is, it has at least one primitive;
2. $c$ is such that for all $x \in G, c(x)$ returns the set which consists of all possible values of $x$.data ${ }^{2}$;
3. $G=\{\operatorname{chan}(p$, io,$n) \mid p \in P$, io $\in\{$ in, out $\}, n \in \mathbb{N}, \operatorname{chan}(p$, io,$n) \neq \perp\}$, that is, every channel is connected to at least one primitive;
4. for all primitives $p, p^{\prime} \in P$, and io $\in\{$ in, out $\}$, and $n, n^{\prime} \in \mathbb{N}$ it holds that

$$
\operatorname{chan}(p, \text { io }, n)=\operatorname{chan}\left(p^{\prime}, \text { io }, n^{\prime}\right) \Rightarrow p=p^{\prime} \wedge n=n^{\prime}
$$

[^4]that is, a channel cannot have more than one initiator and more than one target;
5. for all primitives $p, p^{\prime} \in P$, and $n, n^{\prime} \in \mathbb{N}$ it holds that
$$
\operatorname{chan}(p, \text { out }, n)=\operatorname{chan}\left(p^{\prime}, \text { in }, n^{\prime}\right) \Rightarrow p \neq p^{\prime}
$$
that is, a channel cannot have the same primitive as both the initiator and the target;
6. for all primitives $p \in P$ :
(a) if type $(p)=$ source then

- for all $n \in \mathbb{N}, \operatorname{chan}(p$, in,$n)=\perp$;
- for all $n \in \mathbb{N}$ such that $n>0, \operatorname{chan}(p$, out,$n)=\perp$;
- there is $o \in G$, such that $\operatorname{chan}(p$, out, 0$)=o$,
that is, every source primitive has no input channels and one output channel;
(b) if type $(p)=$ sink then
- for all $n \in \mathbb{N}, \operatorname{chan}(p$, out,$n)=\perp$;
- for all $n \in \mathbb{N}$ such that $n>0$, $\operatorname{chan}(p$, in,$n)=\perp$;
- there is $i \in G$, such that $\operatorname{chan}(p, \mathrm{in}, 0)=i$,
that is, every sink primitive has no output channels and one input channel;
(c) if type $(p)=$ queue $_{k}$ or type $(p)=$ function then
- for all $n \in \mathbb{N}$ such that $n>0$ and for all io $\in\{$ in, out $\}$,

$$
\operatorname{chan}(p, \mathrm{io}, n)=\perp ;
$$

- there is $i \in G$ such that $\operatorname{chan}(p, \mathrm{in}, 0)=i$;
- there is $o \in G$ such that $\operatorname{chan}(p$, out, 0$)=0$,
that is, every queue primitive and every function primitive has one input and one output channel;
(d) if type $(p)=$ fork or type $(p)=$ switch then
- for all $n \in \mathbb{N}$ such that $n>0, \operatorname{chan}(p$, in,$n)=\perp$;
- for all $n \in \mathbb{N}$ such that $n>1, \operatorname{chan}(p$, out,$n)=\perp$;
- there is $i \in G$ such that $\operatorname{chan}(p, \mathrm{in}, 0)=i$;
- there is $o_{0} \in G$ such that $\operatorname{chan}(p$, out, 0$)=o_{0}$;
- there is $o_{1} \in G$ such that $\operatorname{chan}(p$, out, 1$)=o_{1}$,
that is, every fork primitive and every switch primitive has one input channel and two output channels;
(e) if type $(p)=$ join or type $(p)=$ merge then
- for all $n \in \mathbb{N}$ such that $n>1$, $\operatorname{chan}(p$, in,$n)=\perp$;
- for all $n \in \mathbb{N}$ such that $n>0, \operatorname{chan}(p$, out,$n)=\perp$;
- there is $i_{0} \in G$ such that $\operatorname{chan}(p, \mathrm{in}, 0)=i_{0}$;
- there is $i_{1} \in G$ such that $\operatorname{chan}(p, \mathrm{in}, 1)=i_{1}$;
- there is $o \in G$ such that $\operatorname{chan}(p$, out, 0$)=0$,
that is, every join primitive and every merge primitive has two input channels and one output channel.


### 2.3.2 Signal Semantics of xMAS Networks

In Section 2.2, we described the behavior of xMAS primitives in terms of interdependencies between irdy, trdy and data signals of the input and output channels of the primitives. We now extend this to xMAS networks. Our approach is to introduce the notion of signal states, that is, values of irdy, trdy, and data signals of all channels and contents of all queues in the given network at a given moment in time. We also define initial signal state, that is, initial values of irdy, trdy, and data signals of all channels and initial contents of all queues. We also define transition relations between signal states.
First, we define $Q_{D^{\prime}}^{k}$, which represents all possible contents of a queue of type $D$ with capacity $k$.
Definition 2.3. Given $k \in \mathbb{N}^{+}$, and data $D$, then we define $Q_{D}^{k}=\{\mathrm{xs} \mid \mathrm{xs} \in$ $\operatorname{List}(D),|x s| \leq k\}$.

Now we introduce a set of signal labels $\operatorname{Sig}(N)$, which we use to represent values of irdy, trdy and data signals, as well as contents of queues.

Definition 2.4. Given an $x$ MAS network $N=(P, G, C, c$, chan, type $)$, the set of signal labels for $N$ is defined as follows:

$$
\begin{aligned}
\operatorname{Sig}(N) & =\{x . \text { irdy } \mid x \in G\} \cup\{x . \operatorname{trdy} \mid x \in G\} \cup\{x . \text { data }=d \mid x \in G, d \in c(x)\} \\
& \cup\left\{p . \text { queue }=\mathrm{xs} \mid p \in P, \text { type }(p)=\text { queue }_{k^{\prime}} \text { xs } \in Q_{c(\operatorname{chan}(p, \text { in }, 0))}^{k}\right\} .
\end{aligned}
$$

Given a valid xMAS network $N$, we define a validity property for all subsets $S$ of $\operatorname{Sig}(N)$, such that $S$ is valid if and only if:

- every channel in $N$ has exactly one corresponding data label in $S$, and
- every queue in $N$ has exactly one label in $S$ that reflects its contents, and
- all signal labels in $S$ are consistent with signal definitions for primitives from Subsection 2.2.2.

More formally, the validity property for subsets of a set of signal labels is defined as follows.

Definition 2.5. Given a valid $x$ MAS network $N=(P, G, C, c$, chan, type $)$, let $\operatorname{Sig}(N)$ be its set of signal labels. For all $S \subseteq \operatorname{Sig}(N)$, we say that $S$ is a valid set of signal labels if the following properties hold:

1. for all channels $x \in G$ :
(a) there is datum $d \in c(x)$ such that $x$.data $=d \in S$, and
(b) for all data $d, e \in c(x)$ if $x$.data $=d \in S$ and $x$.data $=e \in S$ then $d=e$;
2. for all primitives $p \in P$ :
(a) if $\operatorname{type}(p)=$ queue $_{k}$ with $\operatorname{chan}(p$, in, 0$)=i$ and $\operatorname{chan}(p$, out, 0$)=0$, then

- there is $\mathrm{xs} \in Q_{c(i)}^{k}$ such that $p$.queue $=\mathrm{xs} \in S$, and
- for all $\mathrm{xs}, \mathrm{xs}^{\prime} \in Q_{c(i)}^{k}$ if $p$.queue $=\mathrm{xs} \in S$ and $p$.queue $=\mathrm{xs}^{\prime} \in S$ then $\mathrm{xs}=\mathrm{xs}{ }^{\prime}$, and
- for all $\mathrm{xs} \in Q_{c(i)}^{k}$, if $p$.queue $=\mathrm{xs} \in S$ then $|\mathrm{xs}|<k \Leftrightarrow i$. trdy $\in S$, and
- for all $\mathrm{xs} \in Q_{c(i)}^{k}$, if $p$.queue $=\mathrm{xs} \in S$ then $|\mathrm{xs}|>0 \Leftrightarrow o$. irdy $\in S$, and
- for all $\mathrm{xs} \in Q_{c(i)}^{k}$, if $p$.queue $=\mathrm{xs} \in S$ then $|\mathrm{xs}|>0 \Rightarrow o$.data $=\operatorname{last}(\mathrm{xs}) \in$ S;
(b) if type $(p)=$ function with $\operatorname{chan}(p$, in, 0$)=i, \operatorname{chan}(p$, out, 0$)=o$, and with a data transforming function $f$, then for all $d \in c(i)$
- i.irdy $\in S \Leftrightarrow o$.irdy $\in S$, and
- o.trdy $\in S \Leftrightarrow i . t r d y \in S$, and
- $i$.data $=d \in S \Rightarrow$ o.data $=f(d) \in S$;
(c) if $\operatorname{type}(p)=$ fork with $\operatorname{chan}(p$, in, 0$)=i$, and $\operatorname{chan}(p$, out, 0$)=o_{0}$, and chan $(p$, out, 1$)=o_{1}$, and with data transforming functions $f$ and $g$, then for all $d \in c(i)$
- i.irdy $\in S \wedge o_{1}$.trdy $\in S \Leftrightarrow o_{0}$.irdy $\in S$, and
- $i$. irdy $\in S \wedge o_{0}$.trdy $\in S \Leftrightarrow o_{1}$.irdy $\in S$, and
- $o_{0}$.trdy $\in S \wedge o_{1}$.trdy $\in S \Leftrightarrow i . \operatorname{trdy} \in S$, and
- i.data $=d \in S \Leftrightarrow o_{0}$.data $=f(d) \in S$, and
- i.data $=d \in S \Leftrightarrow o_{1}$. data $=g(d) \in S$;
(d) if $\operatorname{type}(p)=$ join with $\operatorname{chan}(p$, in, 0$)=i_{0}$, and $\operatorname{chan}(p$, in, 1$)=i_{1}$, and chan $(p$, out 0$)=0$, and with a data transforming binary function $h$, then for all $d \in c\left(i_{0}\right), e \in c\left(i_{1}\right)$
- $i_{1}$. $\mathbf{i r d y} \in S \wedge o . \operatorname{trdy} \in S \Leftrightarrow i_{0}$. trdy $\in S$, and
- $i_{0}$.irdy $\in S \wedge o . t r d y \in S \Leftrightarrow i_{1}$.trdy $\in S$, and
- $i_{0} . \mathbf{i r d y} \in S \wedge i_{1}$.irdy $\in S \Leftrightarrow o$. irdy $\in S$, and
- $i_{0} \cdot$ data $=d \in S \wedge i_{1}$. data $=e \in S \Rightarrow$ o.data $=h(d, e) \in S$;
(e) if $\operatorname{type}(p)=\operatorname{switch}$ with $\operatorname{chan}(p$, in, 0$)=i$, and $\operatorname{chan}(p$, out, 0$)=o_{0}$, and chan $(p$, out, 1$)=o_{1}$, and a routing function $r$, then for all $d \in c(i)$
- $i . \mathbf{i r d y} \in S \wedge i$. data $=d \in S \wedge r(d) \Leftrightarrow o_{0}$.irdy $\in S$, and
- $i . \mathbf{i r d y} \in S \wedge i$.data $=d \in S \wedge \neg r(d) \Leftrightarrow o_{1}$.irdy $\in S$, and
- $\left(o_{0} \cdot \mathbf{i r d y} \in S \wedge o_{0} \cdot \mathbf{t r d y} \in S\right) \vee\left(o_{1} \cdot \mathbf{i r d y} \in S \wedge o_{1} \cdot \mathbf{t r d y} \in S\right) \Leftrightarrow i . t r d y \in S$, and
- $i$.data $=d \in S \wedge r(d) \Rightarrow o_{0}$.data $=d \in S$, and
- i.data $=d \in S \wedge \neg r(d) \Rightarrow o_{1}$. data $=d \in S$;
(f) if $\operatorname{type}(p)=$ merge with $\operatorname{chan}(p$, in, 0$)=i_{0}$, and $\operatorname{chan}(p, \mathrm{in}, 1)=i_{1}$, and chan $(p$, out, 0$)=o$, then for all $u \in \mathbb{B}, d \in c\left(i_{0}\right), e \in c\left(i_{1}\right)$ :
- $o$. .irdy $\in S \Leftrightarrow\left(u \wedge i_{0} . \mathbf{i r d y} \in S\right) \vee\left(\neg u \wedge i_{1} . \mathbf{i r d y} \in S\right)$, and
- $i_{0}$.trdy $\in S \Leftrightarrow u \wedge o . \operatorname{trdy} \in S \wedge i_{0}$.irdy $\in S$, and
- $i_{1}$, trdy $\in S \Leftrightarrow \neg u \wedge o . t r d y \in S \wedge i_{1}$.irdy $\in S$, and
- $u \wedge i_{0} . \mathbf{i r d y} \in S \wedge i_{0}$.data $=d \in S \Rightarrow o$.data $=d \in S$, and
- $\neg u \wedge i_{1} . \mathbf{i r d y} \in S \wedge i_{1}$.data $=e \in S \Rightarrow o$. data $=e \in S$.

The set of signal states is now defined as follows.
Definition 2.6. Given a valid $x$ MAS network $N=(P, G, C, c$, chan, type), we define the set of signal states as State $(N)=\{S \subseteq \operatorname{Sig}(N) \mid S$ is valid $\}$.

Further, we define the set of initial signal states. The initial signal states are those signal states in which all queues are empty. Formally, it is defined as follows.

Definition 2.7. Given a valid xMAS network $N=(P, G, C, c$, chan, type $)$, the set of initial signal states $\operatorname{Init}(N)$ is defined as $\{s \in \operatorname{State}(N) \mid \forall p \in P . \operatorname{type}(p)=$ queue $\Rightarrow$ $p$. queue $=[] \in s\}$.

The successor relation for signal states needs to reflect the effect of the standard synchronous operators pre of sources and sinks and the updates of the contents of the queues.

Definition 2.8. Given a valid xMAS network $N=(P, G, C, c, H$, type), successor relation $\operatorname{Next}(N) \subseteq \operatorname{State}(N) \times \operatorname{State}(N)$ is the minimal relation satisfying, for all $\left(s, s^{\prime}\right) \in \operatorname{Next}(N)$, and all primitives $p \in P$ :

- if type $(p)=$ source with chan $(p$, out, 0$)=o$, then $o . \operatorname{irdy} \in s \wedge o . \operatorname{trdy} \notin s \wedge o$. data $=$ $d \in s$ implies $o$. irdy $\in s^{\prime} \wedge o$. data $=d \in s^{\prime}$;
- if $\operatorname{type}(p)=\operatorname{sink}$ with $\operatorname{chan}(p$, in, 0$)=i$, then $i . \operatorname{trdy} \in s \wedge i$ irdy $\notin s$ implies $i . \operatorname{trdy} \in s^{\prime}$;
- if type $(p)=$ queue with $\operatorname{chan}(p$, in, 0$)=i$, and $\operatorname{chan}(p$, out, 0$)=0$, then:
$-\neg(i$. irdy $\in s \wedge i$. trdy $\in s) \wedge \neg(o$. irdy $\in s \wedge o . \operatorname{trdy} \in s) \wedge p$.queue $=\mathrm{xs} \in s$ implies $p$.queue $=\mathrm{xs} \in s^{\prime}$, and
- $i$. irdy $\in s \wedge i . \operatorname{trdy} \in s \wedge o$. data $=d \in s \wedge \neg(o$. irdy $\in s \wedge o . \operatorname{trdy} \in s) \wedge$ $p$. queue $=\mathrm{xs} \in s$ implies $p$.queue $=(d: \mathrm{xs}) \in s^{\prime}$, and
$-\neg(i . \mathbf{i r d y} \in s \wedge i . \operatorname{trdy} \in s) \wedge o . \mathbf{i r d y} \in s \wedge o . \operatorname{trdy} \in s \wedge p$.queue $=\mathrm{xs} \in s$ implies $p$.queue $=\operatorname{rtail}(\mathrm{xs}) \in s^{\prime}$, and
- $i$. irdy $\in s \wedge i . \operatorname{trdy} \in s \wedge o$. data $=d \in s \wedge o$. irdy $\in s \wedge o . \operatorname{trd} \mathbf{y} \in s \wedge p$. queue $=$ $\mathrm{xs} \in s$ implies $p$.queue $=(d: \operatorname{rtail}(\mathrm{xs})) \in s^{\prime}$.

Definition 2.8 only restricts sources, sinks, and queues since sources (sinks) need to take into account their previous states so that production (consumption) of data needs to be persistent; queues need to keep track of their contents; primitives with the other types can behave freely regardless of their previous states.

We combine the signal states, initial signal states and successor relations for signal states into the signal structure.

Definition 2.9. Given a set of signal states State, a set of signal initial states Init, and a signal successor relation Next, we define a signal structure $M=$ (State, Init, Next).

For a given xMAS network, the signal structure is defined as follows.
Definition 2.10. Given a valid xMAS network $N=(P, G, C, c$, chan, type), the signal structure of $N$ is $M(N)=(\operatorname{State}(N), \operatorname{Init}(N), \operatorname{Next}(N))$.
xMAS networks can be composed to obtain larger networks, provided that they are compatible. Networks are compatible if the sets of primitives are disjoint, they agree on the data sent along their shared channels, and each channel is connected to an input/output in at most one network.

Definition 2.11. Given two xMAS networks $N=(P, G, C, c$, chan, type $)$, and $N^{\prime}=$ ( $P^{\prime}, G^{\prime}, C^{\prime}, c^{\prime}$, chan ${ }^{\prime}$, type'), we say that $N$ and $N^{\prime}$ are compatible, if and only if the following holds:

- $P \cap P^{\prime}=\emptyset$, and
- for all channels $x \in G \cap G^{\prime}: c(x)=c^{\prime}(x)$, and
- for all primitives $p \in P, p^{\prime} \in P^{\prime}$, and for all io $\in\{$ in, out $\}$, and for all $n, n^{\prime} \in \mathbb{N}$ it holds that $\operatorname{chan}(p$, io, $n) \neq \operatorname{chan}^{\prime}\left(p^{\prime}\right.$, io, $\left.n^{\prime}\right)$.

We now formally define the composition of two compatible xMAS networks into a larger network.
Definition 2.12. Given two valid, compatible xMAS networks $N^{\prime}=\left(P^{\prime}, G^{\prime}, C^{\prime}, c^{\prime}\right.$, chan', type' $)$, and $N^{\prime \prime}=\left(P^{\prime \prime}, G^{\prime \prime}, C^{\prime \prime}, c^{\prime \prime}\right.$, chan $^{\prime \prime}$, type $\left.{ }^{\prime \prime}\right)$, the composition of the networks $N^{\prime}$ ו। $N^{\prime \prime}=(P, G, C, c$, chan, type $)$ is defined as follows:

1. $P=P^{\prime} \cup P^{\prime \prime}$,
2. $G=G^{\prime} \cup G^{\prime \prime}$,
3. $C=C^{\prime} \cup C^{\prime \prime}$,
4. $c$ is such that for all channels $x \in G$ :

$$
c(x)= \begin{cases}c^{\prime}(x) & \text { if } x \in G^{\prime} \\ c^{\prime \prime}(x) & \text { otherwise }\end{cases}
$$

5. for all primitives $p \in P$, and for all io $\in\{$ in, out $\}$, and for all $n \in \mathbb{N}$ it holds that

$$
\operatorname{chan}(p, \text { io }, n)= \begin{cases}\operatorname{chan}^{\prime}(p, \text { io }, n) & \text { if } p \in P^{\prime} \\ \operatorname{chan}^{\prime \prime}(p, \text { io }, n) & \text { otherwise }\end{cases}
$$

6. for all primitives $p \in P$ it holds that

$$
\operatorname{type}(p)= \begin{cases}\operatorname{type}^{\prime}(p) & \text { if } p \in P^{\prime} \\ \operatorname{type}^{\prime \prime}(p) & \text { otherwise }\end{cases}
$$

All valid xMAS networks with more than one primitive can be decomposed into two compatible networks. This is shown by the following lemma.

Lemma 2.13. For each valid $x M A S$ network $N=(P, G, C, c$, chan, type) with $|P|>1$, there are valid, compatible $x M A S$ networks $N^{\prime}$ and $N^{\prime \prime}$, such that $N=N^{\prime} \| N^{\prime \prime}$.

Proof. Fix valid xMAS network $N=(P, G, C, c$, chan, type $)$, with $|P|>1$. We define $N^{\prime}=\left(P^{\prime}, G^{\prime}, C^{\prime}, c^{\prime}\right.$, chan', type $)$, and $N^{\prime \prime}=\left(P^{\prime \prime}, G^{\prime \prime}, C^{\prime \prime}, c^{\prime \prime}\right.$, chan ${ }^{\prime \prime}$, type $\left.{ }^{\prime \prime}\right)$ such that:

- $P^{\prime}=\{a\}$ for some $a \in P$, and $P^{\prime \prime}=P \backslash P^{\prime} ;$
- $G^{\prime}=\left\{g \in G \mid p \in P^{\prime}\right.$, io $\in\{$ in, out $\}, n \in \mathbb{N}, \operatorname{chan}(p$, io, $\left.n)=g\right\}, G^{\prime \prime}=\{g \in G \mid p \in$ $P^{\prime \prime}$, io $\in\{$ in, out $\}, n \in \mathbb{N}, \operatorname{chan}(p$, io,$\left.n)=g\right\} ;$
- $C^{\prime}=\bigcup_{x \in G^{\prime}} c(x)$, and $C^{\prime \prime}=\bigcup_{x \in G^{\prime \prime}} c(x)$;
- $c^{\prime}: G^{\prime} \rightarrow 2^{C^{\prime}}$ is such that for all channels $x \in G^{\prime}$ it holds that $c^{\prime}(x)=c(x)$, and $c^{\prime \prime}: G^{\prime \prime} \rightarrow 2^{C^{\prime \prime}}$ is such that for all channels $y \in G^{\prime \prime}$ it holds that $c^{\prime \prime}(y)=c(y)$;
- chan' : $P^{\prime} \times\{$ in, out $\} \times \mathbb{N} \rightarrow G^{\prime}$ is such that for all primitives $p \in P^{\prime}$, and for all io $\in\{$ in, out $\}$, and for all $n \in \mathbb{N}$ it holds that $\operatorname{chan}^{\prime}(p$, io,$n)=\operatorname{chan}(p$, io, $n)$, and chan" ${ }^{\prime \prime}: P^{\prime \prime} \times\{$ in, out $\} \times \mathbb{N} \rightarrow G^{\prime \prime}$ is such that for all primitives $p \in P^{\prime \prime}$, and for all io $\in\{$ in, out $\}$, and for all $n \in \mathbb{N}$ it holds that $\operatorname{chan}^{\prime \prime}(p$, io,$n)=\operatorname{chan}(p$, io, $n)$;
- type ${ }^{\prime}: P^{\prime} \rightarrow \Gamma$ is such that for all primitives $p \in P^{\prime}$ it holds that type $(p)=$ type $(p)$, and type ${ }^{\prime \prime}: P^{\prime \prime} \rightarrow \Gamma$ is such that for all primitives $p \in P^{\prime \prime}$ it holds that $\operatorname{type}^{\prime \prime}(p)=\operatorname{type}(p)$.

From the way we define $N^{\prime}$ and $N^{\prime \prime}$, it is clear that $N=N^{\prime} \| N^{\prime \prime}$. It remains to be shown that $N^{\prime}$ and $N^{\prime \prime}$ are valid and compatible. We check each of the validity conditions for $N^{\prime}$ and $N^{\prime \prime}$.

1. $\left|P^{\prime}\right|>0$ and $\left|P^{\prime \prime}\right|>0$. Since $|P|>1, P^{\prime}=\{a\}$, and $P^{\prime \prime}=P \backslash P^{\prime}$, this follows immediately.
2. We prove that for all $x \in G^{\prime}, c^{\prime}(x)$ returns the set which consists of all possible values of $x$.data, and for all $y \in G^{\prime \prime}, c^{\prime \prime}(y)$ returns the set which consists of all possible values of $y$.data. Since $N$ is valid, for all $x \in G, c(x)$ returns the set which consists of all possible values of $x$.data. Since for all $x \in G^{\prime}, c^{\prime}(x)=c(x)$, and for all $y \in G^{\prime \prime}, c^{\prime \prime}(y)=c(y)$, the property holds.
3. We prove

$$
\begin{aligned}
G^{\prime} & =\left\{\operatorname{chan}^{\prime}(p, \text { io }, n) \mid p \in P^{\prime}, \text { io } \in\{\text { in, out }\}, n \in \mathbb{N}, \operatorname{chan}^{\prime}(p, \text { io }, n) \neq \perp\right\}, \text { and } \\
G^{\prime \prime} & =\left\{\operatorname{chan}^{\prime \prime}(p, \text { io }, n) \mid p \in P^{\prime \prime}, \text { io } \in\{\text { in, out }\}, n \in \mathbb{N}, \operatorname{chan}^{\prime \prime}(p, \text { io }, n) \neq \perp\right\} .
\end{aligned}
$$

Since $N$ is valid, for every channel $g \in G$, there are $p \in P$, io $\in\{$ in, out $\}, n \in \mathbb{N}$, such that $\operatorname{chan}(p$, io,$n)=g$. From definition of chan' and chan", for all $p^{\prime} \in$ $P^{\prime}, p^{\prime \prime} \in P^{\prime \prime}$, io $\in\{$ in, out $\}, n \in \mathbb{N}$, we have $\operatorname{chan}^{\prime}\left(p^{\prime}\right.$, io,$\left.n\right)=\operatorname{chan}\left(p^{\prime}\right.$, io,$\left.n\right)$ and chan $^{\prime \prime}\left(p^{\prime \prime}\right.$, io,$\left.n\right)=\operatorname{chan}\left(p^{\prime \prime}\right.$ io, $\left.n\right)$. Hence, using the definition of $G^{\prime}$ and $G^{\prime \prime}$, we conclude that the property holds.
4. We prove for all primitives $p, p^{\prime} \in P^{\prime}$, io $\in\{$ in, out $\}$, and $n, n^{\prime} \in \mathbb{N}$,

$$
\operatorname{chan}^{\prime}(p, \text { io }, n)=\operatorname{chan}^{\prime}\left(p^{\prime}, \text { io }, n^{\prime}\right) \Rightarrow p=p^{\prime} \wedge n=n^{\prime}
$$

Fix $p, p^{\prime} \in P^{\prime}$, io $\in\{$ in, out $\}$, and $n, n^{\prime} \in \mathbb{N}$. Since $N$ is valid, for all $p, p^{\prime} \in P$, io $\in\{$ in, out $\}$, and $n, n^{\prime} \in \mathbb{N}$, it holds that

$$
\operatorname{chan}(p, \mathrm{io}, n)=\operatorname{chan}\left(p^{\prime}, \mathrm{io}, n^{\prime}\right) \Rightarrow p=p^{\prime} \wedge n=n^{\prime}
$$

From the definition of $N^{\prime}$, we have that $\operatorname{chan}^{\prime}(p$, io, $n)=\operatorname{chan}(p$, io,$n)$. Hence,

$$
\operatorname{chan}^{\prime}(p, \text { io }, n)=\operatorname{chan}^{\prime}\left(p^{\prime}, \text { io }, n^{\prime}\right) \Rightarrow p=p^{\prime} \wedge n=n^{\prime}
$$

Similarly, it follows that for all primitives $p, p^{\prime} \in P^{\prime \prime}$, io $\in\{$ in, out $\}$, and $n, n^{\prime} \in \mathbb{N}$,

$$
\operatorname{chan}^{\prime \prime}(p, \text { io }, n)=\operatorname{chan}^{\prime \prime}\left(p^{\prime}, \text { io }, n^{\prime}\right) \Rightarrow p=p^{\prime} \wedge n=n^{\prime}
$$

5. We prove for all $p, p^{\prime} \in P^{\prime}$, io $\in\{$ in, out $\}$, and $n, n^{\prime} \in \mathbb{N}$, it holds that

$$
\operatorname{chan}^{\prime}(p, \text { out, } n)=\operatorname{chan}^{\prime}\left(p^{\prime}, \text { in }, n^{\prime}\right) \Rightarrow p \neq p^{\prime}
$$

Fix $p, p^{\prime} \in P$, and $n, n^{\prime} \in \mathbb{N}$. Since $N$ valid,

$$
\operatorname{chan}(p, \text { out }, n)=\operatorname{chan}\left(p^{\prime}, \text { in }, n^{\prime}\right) \Rightarrow p \neq p^{\prime}
$$

From the definition of $N^{\prime}$, we have that for all primitives $p \in P^{\prime}$, and for all io $\in\{$ in, out $\}$, and for all $n \in \mathbb{N}$ it holds that $\operatorname{chan}^{\prime}(p$, io,$n)=\operatorname{chan}(p$, io,$n)$. Hence,

$$
\operatorname{chan}^{\prime}(p, \text { out }, n)=\operatorname{chan}^{\prime}\left(p^{\prime}, \text { in }, n^{\prime}\right) \Rightarrow p \neq p^{\prime}
$$

Similarly, it follows that for all primitives $p, p^{\prime} \in P^{\prime \prime}$, and $n, n^{\prime} \in \mathbb{N}$, it holds that

$$
\operatorname{chan}^{\prime \prime}(p, \text { out }, n)=\operatorname{chan}^{\prime \prime}\left(p^{\prime}, \text { in }, n^{\prime}\right) \Leftrightarrow p \neq p^{\prime}
$$

6. We prove that conditions 6 (a) - 6 (e) of Definition 2.2 hold for $N^{\prime}$ and $N^{\prime \prime}$. Since $N$ is valid, for all $p \in P$, io $\in\{$ in, out $\}, n \in \mathbb{N}$, chan $(p$, io,$n)$ is consistent with conditions 6 (a) - 6 (e) of Definition 2.2. From the fact that $P=P^{\prime} \cup P^{\prime \prime}$, and since for all $p^{\prime} \in P^{\prime}$, io' $\in\{$ in, out $\}, n^{\prime} \in \mathbb{N}$, we have $\operatorname{chan}^{\prime}\left(p^{\prime}, \mathrm{io}^{\prime}, n^{\prime}\right)=\operatorname{chan}\left(p^{\prime}, \mathrm{io}^{\prime}, n^{\prime}\right)$ and for all $p^{\prime \prime} \in P^{\prime \prime}$, io ${ }^{\prime \prime} \in\{$ in, out $\}, n^{\prime \prime} \in \mathbb{N}$, we have chan" $\left(p^{\prime \prime}, \mathrm{io}^{\prime \prime}, n^{\prime \prime}\right)=$ chan $\left(p^{\prime \prime}, \mathrm{io}^{\prime \prime}, n^{\prime \prime}\right)$, we conclude that conditions 6 (a) - 6 (e) of Definition 2.2 hold for $N^{\prime}$ and $N^{\prime \prime}$.

Finally, we show that $N^{\prime}$ and $N^{\prime \prime}$ are compatible.

- $P \cap P^{\prime}=\emptyset$. This follows immediately from $P^{\prime}=\{a\}$ and $P^{\prime \prime}=P \backslash\{a\}$.
- For all channels $x \in G^{\prime} \cap G^{\prime \prime}$, it holds that $c^{\prime}(x)=c^{\prime \prime}(x)$. Fix $x \in G^{\prime} \cap G^{\prime \prime}$. Then, by definition of $c^{\prime}$ and $c^{\prime \prime}, c^{\prime}(x)=c(x)$ and $c^{\prime \prime}(x)=c(x)$. Hence, $c^{\prime}(x)=c^{\prime \prime}(x)$.
- We prove that for all $p^{\prime} \in P^{\prime}, p^{\prime \prime} \in P^{\prime \prime}$, io $\in\{$ in, out $\}, n^{\prime}, n^{\prime \prime} \in \mathbb{N}$ it holds that $\operatorname{chan}^{\prime}(p$, io,$n) \neq \operatorname{chan}^{\prime \prime}\left(p^{\prime}\right.$, io, $\left.n^{\prime}\right)$. Fix $p^{\prime} \in P^{\prime}, p^{\prime \prime} \in P^{\prime \prime}$, io $\in\{$ in, out $\}, n^{\prime}, n^{\prime \prime} \in \mathbb{N}$. Since $N$ is valid, according to condition 4 of Definition 2.2, chan $\left(p^{\prime}\right.$, io, $\left.n^{\prime}\right) \neq$ $\operatorname{chan}\left(p^{\prime \prime}\right.$, io, $\left.n^{\prime \prime}\right)$. By definition of $\operatorname{chan}^{\prime}, \operatorname{chan}^{\prime}\left(p^{\prime}\right.$, io, $\left.n^{\prime}\right)=\operatorname{chan}\left(p^{\prime}\right.$, io, $\left.n^{\prime}\right)$. By definition of chan", $\operatorname{chan}^{\prime \prime}\left(p^{\prime \prime}\right.$, io, $\left.n^{\prime \prime}\right)=\operatorname{chan}\left(p^{\prime \prime}\right.$, io, $\left.n^{\prime \prime}\right)$. Hence, it holds that $\operatorname{chan}^{\prime}\left(p^{\prime}\right.$, io, $\left.n^{\prime}\right) \neq \operatorname{chan}\left(p^{\prime \prime}\right.$, io, $\left.n^{\prime \prime}\right)$.

Another property of the composition of xMAS networks is that for all networks $N$, composed from networks $N^{\prime}$ and $N^{\prime \prime}$ it holds that any signal state of $N$ can be composed from the union of a signal state of $N^{\prime}$ and a signal state of $N^{\prime \prime}$. We formulate it in the following lemma.
Lemma 2.14. Given three valid $x M A S$ networks $N, N^{\prime}, N^{\prime \prime}$ such that $N=N^{\prime}{ }^{\prime} N^{\prime \prime}$, for all $s \in \operatorname{State}(N)$, there are $s^{\prime} \in \operatorname{State}\left(N^{\prime}\right)$ and $s^{\prime \prime} \in \operatorname{State}\left(N^{\prime \prime}\right)$ such that $s=s^{\prime} \cup s^{\prime \prime}$.

Proof. Let $N, N^{\prime}, N^{\prime \prime}$ be arbitrary xMAS networks such that $N=N^{\prime} \| N^{\prime \prime}$. Assume $N=$ $(P, G, C, c$, chan, type $), N^{\prime}=\left(P^{\prime}, G^{\prime}, C^{\prime}, c^{\prime}\right.$, chan', type $), N^{\prime \prime}=\left(P^{\prime \prime}, G^{\prime \prime}, C^{\prime \prime}, c^{\prime \prime}\right.$, chan", type").

Fix arbitrary $s \in \operatorname{State}(N)$. We construct $s^{\prime}, s^{\prime \prime}$ such that $s=s^{\prime} \cup s^{\prime \prime}, s^{\prime} \in \operatorname{State}\left(N^{\prime}\right)$, and $s^{\prime \prime} \in \operatorname{State}\left(N^{\prime \prime}\right)$ as follows:

$$
\begin{aligned}
s^{\prime}= & \left\{x . \text { irdy } \mid x . \text { irdy } \in s, x \in G^{\prime}\right\} \cup \\
& \left\{x . \text { trdy } \mid x . \text { trdy } \in s, x \in G^{\prime}\right\} \cup \\
& \left\{x . \text { data }=d \mid x . \text { data }=d \in s, x \in G^{\prime}\right\} \cup \\
& \left\{p . \text { queue }=\mathrm{xs} \mid p . \text { queue }=\mathrm{xs} \in s, p \in P^{\prime}\right\} . \\
s^{\prime \prime}= & \left\{x . \text { irdy } \mid x . \text { irdy } \in s, x \in G^{\prime \prime}\right\} \cup \\
& \left\{x . \text { trdy } \mid x . \text { trdy } \in s, x \in G^{\prime \prime}\right\} \cup \\
& \left\{x . \text { data }=d \mid x . \text { data }=d \in s, x \in G^{\prime \prime}\right\} \cup \\
& \left\{p . \text { queue }=\mathrm{xs} \mid p . \text { queue }=\mathrm{xs} \in s, p \in P^{\prime \prime}\right\} .
\end{aligned}
$$

We show that $s=s^{\prime} \cup s^{\prime \prime}$. Since $N=N^{\prime}{ }^{\prime \prime} N^{\prime \prime}$, according to Definition 2.12, we have:

- $P=P^{\prime} \cup P^{\prime \prime}$,
- $G=G^{\prime} \cup G^{\prime \prime}$,
- $\forall x \in G: c(x)= \begin{cases}c^{\prime}(x) & \text { if } x \in G^{\prime}, \\ c^{\prime \prime}(x) & \text { otherwise. }\end{cases}$

From this and definitions of $s^{\prime}$ and $s^{\prime \prime}$ it immediately follows that $s=s^{\prime} \cup s^{\prime \prime}$.
Further we prove $s^{\prime} \in \operatorname{State}\left(N^{\prime}\right)$ by checking that $s^{\prime}$ satisfies all conditions of Definition 2.5.

1. We check condition 1 of Definition 2.5.
(a) For all $x \in G^{\prime}$, there is datum $d \in c^{\prime}(x)$ such that $x$.data $=d \in s^{\prime}$. Fix an arbitrary channel $x \in G^{\prime}$. Since $N=N^{\prime} ॥ N^{\prime \prime}$, from Definition 2.12, $G^{\prime} \subseteq G$. By the definition of $s$, there is $d \in c(x)$ such that $x$.data $=d \in s$, pick such $d$. By the definition of $c^{\prime}, c^{\prime}(x)=c(x)$, hence $d \in c^{\prime}(x)$. By the definition of $s^{\prime}$, $x$. data $=d \in s^{\prime}$.
(b) For all data $d, e \in c^{\prime}(x)$, if $x$.data $=d \in s^{\prime}$ and $x$.data $=e \in s^{\prime}$ then $d=e$. Fix arbitrary $x \in G, d, e \in c^{\prime}(x)$, such that $x$.data $\in s^{\prime}$ and $x$.data $=e \in s^{\prime}$. Since $N=N^{\prime}$ ॥ $N^{\prime \prime}$, from Definition 2.12, $G^{\prime} \subseteq G$. From the latter and from the definition of $s^{\prime}$, we conclude that $x$.data $\in s$ and $x$.data $=e \in s$. Thus, according to Definition 2.5 (1), $d=e$.
2. We check condition 2 of Definition 2.5. Fix arbitrary $p \in P$.
(a) We demonstrate that if $\operatorname{type}(p)=$ queue $_{k}$, and $\operatorname{chan}(p, \mathrm{in}, 0)=i$, and chan $(p$, out, 0$)=o$, then

- there is $\mathrm{xs} \in Q_{c^{\prime}(i)}^{k}$ such that $p$.queue $=\mathrm{xs} \in s^{\prime}$, and
- for all $\mathrm{xs}, \mathrm{xs}^{\prime} \in Q_{c^{\prime}(i)}^{k}$ if $p$.queue $=\mathrm{xs} \in s^{\prime}$ and $p$.queue $=\mathrm{xs}^{\prime} \in s^{\prime}$ then $\mathrm{xs}=\mathrm{xs}^{\prime}$, and
- for all $\mathrm{xs} \in Q_{c^{\prime}(i)}^{k}$, if $p$.queue $=\mathrm{xs} \in s^{\prime}$ then $|\mathrm{xs}|<k \Leftrightarrow i . \operatorname{trdy} \in s^{\prime}$, and
- for all $\mathrm{xs} \in Q_{c^{\prime}(i)}^{k}$, if $p$.queue $=\mathrm{xs} \in s^{\prime}$ then $|\mathrm{xs}|>0 \Leftrightarrow o$. irdy $\in s^{\prime}$, and
- for all $\mathrm{xs} \in Q_{c^{\prime}(i)^{\prime}}^{k}$ if $p$.queue $=\mathrm{xs} \in s^{\prime}$ then $|\mathrm{xs}|>0 \Rightarrow$ o.data $=$ $\operatorname{last}(x s) \in s^{\prime}$;
$\operatorname{Assumetype}(p)=$ queue $_{k}$ with chan $(p$, in, 0$)=i$ and chan $(p$, out, 0$)=0$. We start with proving the first two items.
- From the definition of $s$, there is $\mathrm{xs} \in Q_{c(i)^{\prime}}^{k}$, such that $p$.queue $=$ xs $\in s$; pick such xs. Since $N=N^{\prime} \| N^{\prime \prime}$, from Definition 2.12, $P^{\prime} \subseteq P$. Therefore, from the definition of $s^{\prime}$, we conclude that $p$.queue $=\mathrm{xs} \in s^{\prime}$. From the definition of $c^{\prime}$, we have $c(i)=c^{\prime}(i)$, hence $x s \in Q_{c^{\prime}(i)}^{k}$.
- Fix arbitrary $\mathrm{xs}, \mathrm{xs}^{\prime} \in Q_{c^{\prime}(i)}^{k}$ such that $p$.queue $=\mathrm{xs} \in s^{\prime}$ and $p$.queue $=\mathrm{xs}^{\prime} \in s^{\prime}$. Since $P^{\prime} \subseteq P$, and according to definition of $c^{\prime}, c^{\prime}(i)=c(i)$, therefore $p$.queue $=\mathrm{xs} \in s$ and $p$.queue $=\mathrm{xs}^{\prime} \in s$. From Definition 2.6, we conclude that $\mathrm{xs}=\mathrm{xs}^{\prime}$.

We now prove the rest of the items. Fix an arbitrary $\mathrm{xs} \in Q_{c^{\prime}(i)^{\prime}}^{k}$ such that $p$.queue $=\mathrm{xs} \in s^{\prime}$. Recall that $P^{\prime} \subseteq P$, and $c^{\prime}(i)=c(i)$. From the definition of $s^{\prime}$ we have that $p$.queue $=\mathrm{xs} \in s^{\prime}$ only if $p$.queue $=x s \in s$, hence $p$.queue $=\mathrm{xs} \in s$.

- $\Rightarrow$ Assume $|\mathrm{xs}|<k$. Then, by Definition 2.6, i.trdy $\in s$. Since $G^{\prime} \subseteq G$, it follows from definition of $s^{\prime}$ that $i . t r d y \in s^{\prime}$.
$\Leftarrow$ Assume $i . \operatorname{trdy} \in s^{\prime}$. Since $G^{\prime} \subseteq G, i . \operatorname{trdy} \in s$. Then, by Definition 2.6, $|\mathrm{xs}|<k$.
- $\Rightarrow$ Assume $|\mathrm{xs}|>0$. Then, by Definition 2.6, o.irdy $\in s$. Since $G^{\prime} \subseteq G$, it follows from definition of $s^{\prime}$, that $o$.irdy $\in s^{\prime}$.
$\Leftarrow$ Assume o.irdy $\in s^{\prime}$. Since $G^{\prime} \subseteq G$, o.irdy $\in s$. Then, by Definition 2.6, |xs $\mid>0$.
- $\Rightarrow$ Assume $|\mathrm{xs}|>0$. Then, by Definition 2.6, o.data $=\operatorname{last}(\mathrm{xs}) \in s$. Since $G^{\prime} \subseteq G$, it follows from definition of $s^{\prime}$, that o.data $=$ $\operatorname{last}(x s) \in s^{\prime}$.
(b)-(f) We follow the same lines as in 2 (a) to check that conditions 2 (b) - 2(f) hold.

The proof that $s^{\prime \prime} \in \operatorname{State}\left(N^{\prime \prime}\right)$ is completely analogous to that of $s^{\prime} \in \operatorname{State}\left(N^{\prime}\right)$.

### 2.4 Kripke Structure Semantics

A Kripke structure (KS) is a natural way to capture the state-based semantics of xMAS networks. In this section we define the KS that captures the behavior of a given xMAS network.
Definition 2.15. A Kripke structure is a tuple $(S, I, \rightarrow, A P, L)$, where:

- $S$ is a set of states,
- $I \subseteq S$ is the set of initial states,
- $\rightarrow \subseteq S \times S$ is a transition relation,
- AP is a set of atomic propositions,
- $L: S \rightarrow 2^{\mathrm{AP}}$ is a labelling function.

We assume the transition relation $\rightarrow$ to be total, that is, for all $s \in S$, there is $s^{\prime}$, such that $\left(s, s^{\prime}\right) \in \rightarrow$. By convention, we write $s \rightarrow s^{\prime}$, whenever $\left(s, s^{\prime}\right) \in \rightarrow$.

Definition 2.16. Given a Kripke structure $K$, a path of $K$ is a (possibly infinite) sequence of states $s_{0} s_{1} s_{2} \ldots$, such that for all $i \geq 0$ it holds that $s_{i} \rightarrow s_{i+1}$.

### 2.4.1 Kripke Structure Semantics of Individual Components

We first describe the semantics of a valid xMAS network consisting of a single primitive as a Kripke Structure. For the rest of the subsection, fix an arbitrary valid xMAS network $N=(P, G, C, c$, chan, type) with $|P|=1$.

For convenience, let us first introduce some supplementary notation.
Notation 1. For $x \in G$, we define the atomic propositions related to channel $x$ as

$$
\operatorname{ap}(x)=\{x . \mathbf{i r d y}, x . \operatorname{trdy}\} \cup\{x . \text { data }=d \mid d \in c(x)\} .
$$

For primitives $p \in P$ with type $(p)=$ queue $_{k}$ and chan $(p$, in, 0$)=i$, we define the atomic propositions for queue $p$ as

$$
\operatorname{ap}_{q}(p)=\left\{p . \text { queue }=\mathrm{xs} \mid \mathrm{xs} \in Q_{c(i)}^{k}\right\} .
$$

For primitives $p \in P$ with type $(p)=\operatorname{merge}, \operatorname{chan}(p, \operatorname{in}, 0)=i$, and $\operatorname{chan}(p, \mathrm{in}, 1)=j$ we define the arbitration atomic propositions as follows:

$$
\operatorname{ap}_{m}(p)=\{p \cdot \mathrm{msel}=x \mid x \in\{i, j\}\}
$$

For variable $v$ and $x \in \mathbb{B} \cup C$, we define the following:

$$
\operatorname{lab}_{v}(x)=\left\{\begin{array}{l}
\{v \mid x=\text { true }\} \text { if } x \in \mathbb{B} \\
\{v=d \mid x=d\} \text { otherwise } .
\end{array}\right.
$$

Given a variable name $v$, and a value $e, \operatorname{lab}_{v}(e)$ generates the singleton set of labels containing $v=e$; if $e$ is a Boolean, and $e=$ false, $\operatorname{lab}_{v}(e)$ generates $\emptyset$.

We generalize this notation to a family of functions $\operatorname{lab}_{\left(v_{1}, \ldots, v_{n}\right)}\left(x_{1}, \ldots, x_{n}\right)$, defined as

$$
\operatorname{lab}_{\left(v_{1}, \ldots, v_{n}\right)}\left(x_{1}, \ldots, x_{n}\right)=\bigcup_{i=1}^{n} \operatorname{lab}_{v_{i}}\left(x_{i}\right)
$$

Further, we define the KS depending on the type of $p \in P$. Note, that the KS controls the trdy signals of its input channels, and irdy and data of its output channels. Therefore, irdy and data signals of input channels, and trdy of output channels are left unconstrained.

Source If type $(p)=\operatorname{source}$ with $\operatorname{chan}(p$, out, 0$)=o$, the KS for $N$ is defined as $\mathrm{KS}(N)=(S, I, \rightarrow, \mathrm{AP}, L)$, where:

- $S=\mathbb{B} \times \mathbb{B} \times c(o)$,
- $I=S$,
- $\rightarrow$ is the smallest relation satisfying the following:

$$
\begin{aligned}
& \operatorname{Src} 1 \frac{o_{\text {irdy }} \Rightarrow o_{\text {trdy }}}{\left(o_{\text {irdy }}, o_{\text {trdy }}, d\right) \rightarrow\left(o_{\text {irdy }}^{\prime}, o_{\text {trdy }}^{\prime} d^{\prime}\right)} \\
& \operatorname{Src} 2 \frac{(\text { true }, \text { false }, d) \rightarrow\left(\text { true }, o_{\text {trdy }}^{\prime}, d\right)}{}
\end{aligned}
$$

- $\mathrm{AP}=\mathrm{ap}(o)$,
- $L=\operatorname{lab}_{(o . i r d y, o . t r d y, 0 . d a t a)}$.

Sink If type $(p)=\operatorname{sink}$ with $\operatorname{chan}(p$, in, 0$)=i$, the KS for $N$ is defined as $\operatorname{KS}(N)=$ $(S, I, \rightarrow, A P, L)$, where:

- $S=\mathbb{B} \times \mathbb{B} \times c(i)$,
- $I=S$.
- $\rightarrow$ is the smallest relation satisfying the following:

$$
\begin{aligned}
& \text { Snk1 } \frac{i_{\text {trdy }} \Rightarrow i_{\text {irdy }}}{\left(i_{\text {irdy }}, i_{\text {trdy }}, d\right) \rightarrow\left(i_{\text {irdy }}^{\prime}, i_{\text {trdy }}^{\prime} d^{\prime}\right)} \\
& \text { Snk2 } \frac{(\text { false, true }, d) \rightarrow\left(i_{\text {irdy }}^{\prime}, \text { true }, d^{\prime}\right)}{}
\end{aligned}
$$

- $\mathrm{AP}=\mathrm{ap}(i)$,
- $L=\operatorname{lab}_{(i . i r d y, i . t r d y, i . d a t a)}$.

Queue If type $(p)=$ queue $_{k}$ with $\operatorname{chan}(p$, in, 0$)=i$ and $\operatorname{chan}(p$, out, 0$)=0$, the KS for $N$ is defined as $\operatorname{KS}(N)=(S, I, \rightarrow, \mathrm{AP}, L)$, where:

- $S=\left\{\left(\mathrm{xs}, i_{\text {irdy }}, i_{\text {trdy }}, d, o_{\text {irdy }}, o_{\text {trdy }}, e\right) \in Q_{c(i)}^{k} \times \mathbb{B} \times \mathbb{B} \times c(i) \times \mathbb{B} \times \mathbb{B} \times c(o)| | \mathrm{xs} \mid<k \Leftrightarrow\right.$ $\left.i_{\text {trdy }},|\mathrm{xs}|>0 \Leftrightarrow o_{\text {irdy }},|\mathrm{xs}|>0 \Rightarrow e=\operatorname{last}(\mathrm{xs})\right\}$.
- $I=\left\{\left(\mathrm{xs}, i_{\text {irdy }}, i_{\text {trdy }}, d, o_{\text {irdy }}, o_{\text {trdy }}, e\right) \in S \mid \mathrm{xs}=[]\right\}$.
- Assume $s=\left(\mathrm{xs}, i_{\text {irdy }}, i_{\text {trdy }}, d, o_{\text {irdy }}, o_{\text {trdy }}, e\right), s^{\prime}=\left(\mathrm{xs}^{\prime}, i_{\text {irdy }}^{\prime} i_{\text {trdy }}^{\prime}, d^{\prime}, o_{\text {irdy }}^{\prime}, o_{\text {trdy }}^{\prime}, e^{\prime}\right)$. Then, $\rightarrow$ is the smallest relation satisfying the following:

$$
\begin{aligned}
& \text { Q1 } \frac{\neg\left(i_{\text {irdy }} \wedge i_{\text {trdy }}\right) \quad \neg\left(o_{\text {irdy }} \wedge o_{\text {trdy }}\right) \quad \mathrm{xs}^{\prime}=\mathrm{xs}}{s \rightarrow s^{\prime}} \\
& \text { Q2 } \frac{\left(i_{\text {irdy }} \wedge i_{\text {trdy }}\right) \neg\left(o_{\text {irdy }} \wedge o_{\text {trdy }}\right) \quad \mathrm{xs}^{\prime}=(d: \text { xs })}{s \rightarrow s^{\prime}} \\
& \text { Q3 } \frac{\neg\left(i_{\text {irdy }} \wedge i_{\text {trdy }}\right)\left(o_{\text {irdy }} \wedge o_{\text {trdy }}\right) \quad \mathrm{xs}^{\prime}=(\text { rtail }(\mathrm{xs}))}{s \rightarrow s^{\prime}} \\
& \mathrm{Q} 4 \frac{\left(i_{\text {irdy }} \wedge i_{\text {trdy }}\right)\left(o_{\text {irdy }} \wedge o_{\text {trdy }}\right) \quad \mathrm{xs}^{\prime}=(d:(\operatorname{rtail}(\mathrm{xs})))}{s \rightarrow s^{\prime}},
\end{aligned}
$$

- $\mathrm{AP}=\mathrm{ap}(i) \cup \mathrm{ap}(o) \cup \mathrm{ap}_{q}(p)$,
- $L=\operatorname{lab}_{(p . q u e u e, i . i r d y, i . t r d y, i . d a t a, o . i r d y}, o$. trdy,$o$. data $)$.

Function If type $(p)=$ function with $\operatorname{chan}(p$, in, 0$)=i, \operatorname{chan}(p$, out, 0$)=0$, and with a data transforming function $f$, the KS for $N$ is defined as $\operatorname{KS}(N)=(S, I, \rightarrow, \mathrm{AP}, L)$, where:

- $S=\left\{\left(i_{\text {irdy }}, i_{\text {trdy }}, d, o_{\text {irdy }}, o_{\text {trdy }}, e\right) \in \mathbb{B} \times \mathbb{B} \times c(i) \times \mathbb{B} \times \mathbb{B} \times c(o) \mid\right.$

$$
\left.i_{\text {trdy }}=o_{\text {trdy }}, o_{\text {irdy }}=i_{\text {irdy }}, e=f(d)\right\}
$$

- $I=S$.
- $\rightarrow$ is the smallest relation satisfying the following, for all $s, s^{\prime} \in S$ :

$$
\text { Fun1 } \overline{s \rightarrow s^{\prime}},
$$

- $\mathrm{AP}=\mathrm{ap}(i) \cup \mathrm{ap}(o)$,
- $L=\operatorname{lab}_{(i . i r d y, i . t r d y}, i . d a t a, o$. irdy, 0. trdy,$o$. data $)$.

Fork If $\operatorname{type}(p)=$ fork with channels $\operatorname{chan}(p$, in, 0$)=i, \operatorname{chan}(p$, out, 0$)=0$, and chan $(p$, out, 1$)=u$, and with data transforming functions $f$ and $f^{\prime}$, the KS for $N$ is defined as $\operatorname{KS}(N)=(S, I, \rightarrow, A P, L)$, where:

- $S=\left\{\left(i_{\text {irdy }}, i_{\text {trdy }}, d, o_{\text {irdy }}, o_{\text {trdy }}, e, u_{\text {irdy }}, u_{\text {trdy }}, j\right) \in \mathbb{B} \times \mathbb{B} \times c(i) \times \mathbb{B} \times \mathbb{B} \times c(o) \times \mathbb{B} \times\right.$ $\mathbb{B} \times c(u) \mid e=f(d), u=f^{\prime}(d), o_{\text {irdy }}=i_{\text {irdy }} \wedge u_{\text {trdy }}, u_{\text {irdy }}=i_{\text {irdy }} \wedge o_{\text {trdy }}, i_{\text {trdy }}=$ $\left.o_{\text {trdy }} \wedge u_{\text {trdy }}\right\}$,
- $I=S$,
- $\rightarrow$ is the smallest relation satisfying the following, for all $s, s^{\prime} \in S$ :

$$
\text { Frk1 } \xlongequal[s \rightarrow s^{\prime}]{ }
$$

- $\mathrm{AP}=\operatorname{ap}(i) \cup \operatorname{ap}(o) \cup \mathrm{ap}(u)$,
- $L=\operatorname{lab}_{(i . i r d y}, i . t r d y, i . d a t a, 0 . i r d y, o . t r d y, o . d a t a, u . i r d y, u . t r d y, u$.data) .

Join If $\operatorname{type}(p)=$ join with $\operatorname{chan}(p$, in, 0$)=i, \operatorname{chan}(p$, in, 1$)=j, \operatorname{chan}(p$, out, 0$)=0$, and with a binary data transforming function $h$, the $\operatorname{KS}$ for $N$ is defined as $\operatorname{KS}(N)=$ $(S, I, \rightarrow, A P, L)$, where:

- $S=\left\{\left(i_{\text {irdy }}, i_{\text {trdy }}, d, j_{\text {irdy }}, j_{\text {trdy }}, e, o_{\text {irdy }}, o_{\text {trdy }}, l\right) \in \mathbb{B} \times \mathbb{B} \times c(i) \times \mathbb{B} \times \mathbb{B} \times c(o) \times \mathbb{B} \times\right.$ $\left.\mathbb{B} \times c(u) \mid l=h(d, e), i_{\text {trdy }}=j_{\text {irdy }} \wedge o_{\text {trdy }}, j_{\text {trdy }}=i_{\text {irdy }} \wedge o_{\text {trdy }}, o_{\text {irdy }}=i_{\text {irdy }} \wedge j_{\text {irdy }}\right\}$,
- $I=S$,
- $\rightarrow$ is the smallest relation satisfying the following, for all $s, s^{\prime} \in S$ :

$$
\operatorname{Jn1} \overline{s \rightarrow s^{\prime}},
$$

- $\mathrm{AP}=\operatorname{ap}(i) \cup \operatorname{ap}(j) \cup \operatorname{ap}(o)$,
- $L=\operatorname{lab}_{(i . i r d y, i . t r d y, i . d a t a, j, i \text { irdy }, j . \operatorname{trdy}, j \text {.data }, . \text {.irdy }, 0 . \text {.trdy }, 0 . \text { data })}$.

Switch If type $(p)=$ switch with channels $\operatorname{chan}(p$, in, 0$)=i$, $\operatorname{chan}(p$, out, 0$)=o$, $\operatorname{chan}(p$, out 1$)=u$, and a routing function $r$, the $\operatorname{KS}$ for $N$ is defined as $\operatorname{KS}(N)=$ $(S, I, \rightarrow, A P, L)$, where:

- $S=\left\{\left(i_{\text {irdy }}, i_{\text {trdy }}, d, o_{\text {irdy }}, o_{\text {trdy }}, e, u_{\text {irdy }}, u_{\text {trdy }}, l\right) \in \mathbb{B} \times \mathbb{B} \times c(i) \times \mathbb{B} \times \mathbb{B} \times c(o) \times \mathbb{B} \times\right.$ $\mathbb{B} \times c(u) \mid(r(d) \Rightarrow l=d),(\neg r(d) \Rightarrow l=e),\left(o_{\text {irdy }}=i_{\text {irdy }} \wedge r(d)\right), u_{\text {irdy }}=i_{\text {irdy }} \wedge$ $\left.\neg r(d), i_{\text {trdy }}=\left(o_{\text {irdy }} \wedge o_{\text {trdy }}\right) \vee\left(u_{\text {irdy }} \wedge u_{\text {trdy }}\right)\right\}$,
- $I=S$,
- $\rightarrow$ is the smallest relation satisfying the following, for all $s, s^{\prime} \in S$ :

$$
\mathrm{Sw} 1 \xrightarrow[s \rightarrow s^{\prime}]{ }
$$

- $\mathrm{AP}=\operatorname{ap}(i) \cup \operatorname{ap}(o) \cup \mathrm{ap}(u)$,

Merge If type $(p)=$ merge with channels chan $(p$, in, 0$)=i$, $\operatorname{chan}(p, \mathrm{in}, 1)=j$, and chan $(p$, out, 0$)=o$, the $\operatorname{KS}$ for $N$ is defined as $\operatorname{KS}(N)=(S, I, \rightarrow, A P, L)$, where:
- $S=\left\{\left(u, i_{\text {irdy }}, i_{\text {trdy }}, d, j_{\text {irdy }}, j_{\text {trdy }}, e, o_{\text {irdy }}, o_{\text {trdy }}, l\right) \in \mathbb{B} \times \mathbb{B} \times \mathbb{B} \times c(i) \times \mathbb{B} \times \mathbb{B} \times c(j) \times\right.$ $\mathbb{B} \times \mathbb{B} \times c(o) \mid o_{\text {irdy }}=\left(i \wedge i_{\text {irdy }}\right) \vee\left(\neg u \wedge j_{\text {irdy }}\right), i_{\text {trdy }}=u \wedge o_{\text {trdy }} \wedge i_{\text {irdy }}, j_{\text {trdy }}=$ $\left.\neg u \wedge o_{\text {trdy }} \wedge j_{\text {irdy }}, u \wedge i_{\text {irdy }} \Rightarrow l=d, \neg u \wedge j_{\text {irdy }} \Rightarrow l=e\right\}$,
- $I=S$,
- $\rightarrow$ is the smallest relation satisfying the following, for all $s, s^{\prime} \in S$ :

$$
\operatorname { M r g } 1 \longdiv { s \rightarrow s ^ { \prime } }
$$

- $\mathrm{AP}=\operatorname{ap}(i) \cup \mathrm{ap}(j) \cup \mathrm{ap}(o) \cup \mathrm{ap}_{m}(p)$,

Note that for every type of $p, \rightarrow$ is total.


### 2.4.2 Kripke Structure Semantics of xMAS Networks

To build Kripke Structures for more complex xMAS networks, we compose Kripke Structures of smaller networks. For this, we use the parallel composition of two KSs as introduced by Clarke et al.

Definition 2.17. ([CGL96]) Let $K^{\prime}=\left(S^{\prime}, I^{\prime}, \rightarrow^{\prime}, \mathrm{AP}^{\prime}, L^{\prime}\right), K^{\prime \prime}=\left(S^{\prime \prime}, I^{\prime \prime}, \rightarrow{ }^{\prime \prime}, \mathrm{AP}^{\prime \prime}, L^{\prime \prime}\right)$ be two Kripke Structures, then the parallel composition of $K^{\prime}$ and $K^{\prime \prime}$, denoted as $K^{\prime} \| K^{\prime \prime}$ is the Kripke Structure $K=(S, I, \rightarrow, \mathrm{AP}, L)$ defined as follows:

- $S=\left\{(p, q) \mid p \in S^{\prime}, q \in S^{\prime \prime},\left(L^{\prime}(p) \cap \mathrm{AP}^{\prime \prime}\right)=\left(L^{\prime \prime}(q) \cap \mathrm{AP}^{\prime}\right)\right\}$,
- $I=\left(I^{\prime} \times I^{\prime \prime}\right) \cap S$,
- $\rightarrow$ is the smallest relation satisfying, for all $p, p^{\prime} \in S^{\prime}, q, q^{\prime} \in S^{\prime \prime}$

$$
\frac{p \rightarrow^{\prime} p^{\prime} \quad q \rightarrow^{\prime \prime} q^{\prime}}{(p, q) \rightarrow\left(p^{\prime}, q^{\prime}\right)}
$$

- $L((p, q))=L^{\prime}(p) \cup L^{\prime \prime}(q)$, for $(p, q) \in S$,
- $\mathrm{AP}=\mathrm{AP}^{\prime} \cup \mathrm{AP}^{\prime \prime}$.

Note that states $p \in S^{\prime}$ and $q \in S^{\prime \prime}$ can only be combined into $(p, q)$ in $S$ if they agree on the values of the shared atomic propositions.
Using parallel composition, we can define the KS for a given valid xMAS network $N=(P, G, C, c$, chan, type $)$ with $|P|>1$.

Definition 2.18. Given a valid xMAS network $N=(P, G, C, c, H$, type $)$, the KS which represents $N$, denoted $\operatorname{KS}(N)$, is defined inductively in the number of primitives as follows.

- $|P|=1$. Then $\operatorname{KS}(N)$ is obtained as described in Section 2.4.1.
- $|P|>1$. According to Lemma 2.13, $N$ can be split into $N^{\prime}$ and $N^{\prime \prime}$, such that $N=N^{\prime} \| N^{\prime \prime}$. Let $N^{\prime}$ and $N^{\prime \prime}$ be such. Then $\operatorname{KS}(N)=\operatorname{KS}\left(N^{\prime}\right) \| \operatorname{KS}\left(N^{\prime \prime}\right)$.

For any $x$ MAS network $N$, the labelling function of $\operatorname{KS}(N)$ is bijective. We prove this in the lemma below.

Lemma 2.19. Let $K S(N)=(S, I, \rightarrow, A P, L)$ be the $K S$ for a valid $N=(P, G, C, c$, chan, type $)$. Then, $L$ is bijective.

Proof. Proof by induction on the number of primitives in $N$.

- $|P|=1$. Let $P=\{p\}$. For all types of $p$, it follows immediately from the definition of $L$ that $L$ is bijective.
- $|P|>1$. It follows from Definition 2.18 that there are xMAS networks $N^{\prime}$ and $N^{\prime \prime}$ such that $\operatorname{KS}(N)=\operatorname{KS}\left(N^{\prime}\right) \| K S\left(N^{\prime \prime}\right)$; let $N^{\prime}$ and $N^{\prime \prime}$ be such. Assume $\mathrm{KS}\left(N^{\prime}\right)=\left(S^{\prime}, I^{\prime}, \rightarrow^{\prime}, \mathrm{AP}^{\prime}, L^{\prime}\right)$ and $\mathrm{KS}\left(N^{\prime \prime}\right)=\left(S^{\prime \prime}, I^{\prime \prime}, \rightarrow^{\prime \prime}, \mathrm{AP}^{\prime \prime}, L^{\prime \prime}\right)$. According to the induction hypothesis, $L^{\prime}$ and $L^{\prime \prime}$ are bijective. We prove that $L$ is bijective, that is, for all $(p, q),\left(p^{\prime}, q^{\prime}\right) \in S$, that $L((p, q))=L\left(\left(p^{\prime}, q^{\prime}\right)\right)$ if and only if $(p, q)=$ $\left(p^{\prime}, q^{\prime}\right)$. Fix arbitrary $(p, q),\left(p^{\prime}, q^{\prime}\right) \in S$. We prove both directions separately.
$\Rightarrow$ Assume $L((p, q))=L\left(\left(p^{\prime}, q^{\prime}\right)\right)$. By Definition 2.18, $\operatorname{KS}(N)=\operatorname{KS}\left(N^{\prime}\right) \| \operatorname{KS}\left(N^{\prime \prime}\right)$. According to Definition 2.17, thus $L((p, q))=L^{\prime}(p) \cup L^{\prime \prime}(q)$ and $L\left(p^{\prime}, q^{\prime}\right)=$
$L^{\prime}\left(p^{\prime}\right) \cup L^{\prime \prime}\left(q^{\prime}\right)$. Since $L((p, q))=L\left(\left(p^{\prime}, q^{\prime}\right)\right)$, we have

$$
\begin{equation*}
L^{\prime}(p) \cup L^{\prime \prime}(q)=L^{\prime}\left(p^{\prime}\right) \cup L^{\prime \prime}\left(q^{\prime}\right) \tag{2.1}
\end{equation*}
$$

From Definition 2.17, we also get

$$
\begin{gather*}
L^{\prime}(p) \cap \mathrm{AP}^{\prime \prime}=L^{\prime \prime}(q) \cap \mathrm{AP}^{\prime}  \tag{2.2}\\
L^{\prime}\left(p^{\prime}\right) \cap \mathrm{AP}^{\prime \prime}=L^{\prime \prime}\left(q^{\prime}\right) \cap \mathrm{AP}^{\prime} \tag{2.3}
\end{gather*}
$$

We use these observations to prove that $L^{\prime}(p)=L^{\prime}\left(p^{\prime}\right)$; the argument for $L^{\prime \prime}(q)=L^{\prime \prime}\left(q^{\prime}\right)$ is analogous. We prove both directions separately.

* Fix $l \in L^{\prime}(p)$. Note that $l \in \mathrm{AP}^{\prime}$ by definition of $L^{\prime}$. We distinguish two cases:
$\cdot l \notin L^{\prime \prime}(q)$. From (2.2), we find $l \notin \mathrm{AP}^{\prime \prime}$, hence using (2.1) we immediately find $l \in L^{\prime}\left(p^{\prime}\right)$.
$\cdot l \in L^{\prime \prime}(q)$. Then $l \in \mathrm{AP}^{\prime \prime}$. From (2.1), $l \in \mathrm{AP}^{\prime} \cap \mathrm{AP}^{\prime \prime}$ and (2.2), and (2.3) it immediately follows that $l \in L^{\prime}\left(p^{\prime}\right)$ (and $\left.l \in L^{\prime \prime}\left(q^{\prime}\right)\right)$.
* Assume $l \in L^{\prime}\left(p^{\prime}\right)$; using the symmetric argument it follows that $l \in$ $L^{\prime}(p)$.

We have now established that $L^{\prime}(p)=L^{\prime}\left(p^{\prime}\right)$ and $L^{\prime \prime}(q)=L^{\prime \prime}\left(q^{\prime}\right)$. According to the induction hypothesis, therefore $p=p^{\prime}$ and $q=q^{\prime}$. Hence $(p, q)=$ ( $p^{\prime}, q^{\prime}$ )
$\Leftarrow$ Assume $(p, q)=\left(p^{\prime}, q^{\prime}\right)$. Then, $p=p^{\prime}$ and $q=q^{\prime}$. Hence, by the induction hypothesis, we have $L^{\prime}(p)=L^{\prime}\left(p^{\prime}\right)$ and $L^{\prime \prime}(q)=L^{\prime \prime}\left(q^{\prime}\right)$. From this, it follows that $L^{\prime}(p) \cup L^{\prime \prime}(q)=L^{\prime}\left(p^{\prime}\right) \cup L^{\prime \prime}\left(q^{\prime}\right)$. Using Definition 2.17, we conclude $L((p, q))=L\left(\left(p^{\prime}, q^{\prime}\right)\right)$.

### 2.5 Correctness of the KS Semantics

We show correctness of the KS semantics of xMAS networks by establishing a correspondence between the signal semantics and the KS semantics.
To relate a signal structure and a Kripke Structure, we formulate a bisimulation relation between both formalisms.

Definition 2.20. Given signal structure $M(N)=(\operatorname{State}(N), \operatorname{Init}(N), \operatorname{Next}(N))$ for some xMAS network $N$ and Kripke structure $K=(S, I, \rightarrow, A P, L)$, relation $R \subseteq$ State $(N) \times S$ is a bisimulation relation if and only if for every $s \in \operatorname{State}(N)$ and $p \in S$ such that $s R p$, the following hold:

- $s=L(p)$, and
- $s \in \operatorname{Init}(N)$ if and only if $p \in I$, and
- for all $s^{\prime} \in \operatorname{State}(N)$ such that $\left(s, s^{\prime}\right) \in \operatorname{Next}(N)$, there exists $p^{\prime} \in S$ such that $p \rightarrow p^{\prime}$ and $s^{\prime} R p^{\prime}$, and
- for all $p^{\prime} \in S$ such that $p \rightarrow p^{\prime}$, there exists $s^{\prime} \in \operatorname{State}(N)$ such that $\left(s, s^{\prime}\right) \in \operatorname{Next}(N)$ and $s^{\prime} R p^{\prime}$.

We say that $s \in \operatorname{State}(N)$ and $p \in P$ are bisimilar, denoted $s \leftrightarrow p$, iff there exists a bisimulation relation $R$ such that $s R p$. Signal structure $M(N)$ and Kripke structure $K$ are bisimilar iff for all $s \in \operatorname{Init}(N)$ there exists $p \in I$ such that $s \leftrightarrow p$ and for all $p \in I$ there exists $s \in \operatorname{Init}(N)$ such that $s \leftrightarrow p$.

As a stepping stone towards proving bisimulation between a signal structure and the KS of a given xMAS network, we show that the set of signal states of the signal structure is equal to the set of atomic proposition of the KS.

Lemma 2.21. Given a valid $x M A S$ network $N=(P, G, C, c$, chan, type), let $M(N)=$ $(\operatorname{State}(N), \operatorname{Init}(N), \operatorname{Next}(N))$ be the signal structure of $N$, and $K S(N)=(S, I, \rightarrow, A P, L)$ be the Kripke Structure representing $N$. Then, State $(N)=\{L(p) \mid p \in S\}$.

Proof. Proof by induction on the number of primitives of $N$.

- Base case. Assume $|P|=1$. Let $P=\{z\}$. We distinguish cases based on the type of $z$.
- $\operatorname{type}(z)=$ source. Since $N$ is valid, from Definition 2.2, assume without loss of generality that $G=\{0\}$ and $\operatorname{chan}(z$, out, 0$)=0$. According to Definition 2.6, $\operatorname{State}(N)=\{\{o$. data $=d\} \mid d \in c(o)\} \cup\{\{o$. irdy,$o$. data $=d\} \mid d \in c(o)\} \cup$ $\{\{o$. trdy,$o$. data $=d\} \mid d \in c(o)\} \cup\{\{o$. irdy, $o$. trdy,$o$. data $=d\} \mid d \in c(o)\}$. By the definition of $\mathrm{KS}(N), S=\mathbb{B} \times \mathbb{B} \times c(o)$, and hence $\{L(p) \mid p \in S\}=$ $\left\{\operatorname{lab}_{(o . \operatorname{irdy}, o . \operatorname{trdy}, 0 . \operatorname{data})}(p) \mid p \in S\right\}=\operatorname{State}(N)$ follows from the definition of lab $_{(o . i r d y, o . t r d y, o . d a t a)}$ immediately.
- $\operatorname{type}(z)=$ sink. Since $N$ is valid, from Definition 2.2, assume without loss of generality that $G=\{i\}$ and $\operatorname{chan}(z$, in, 0$)=i$. According to Definition 2.6, State $(N)=\{\{i$. data $=e\} \mid e \in c(i)\} \cup\{\{i$. irdy, $i$. data $=e\} \mid e \in c(i)\} \cup$ $\{\{i . \operatorname{trdy}, i$. data $=i\} \mid i \in c(i)\} \cup\{\{i . \mathbf{i r d y}, i . \operatorname{trdy}, i$. data $=e\} \mid e \in c(i)\}$. By the definition of $\operatorname{KS}(N), S=\mathbb{B} \times \mathbb{B} \times c(i)$, and hence $\{L(p) \mid p \in S\}=$ $\left\{\operatorname{lab}_{(i . i r d y, i . t r d y, i . d a t a)}(p) \mid p \in S\right\}=\operatorname{State}(N)$ follows from the definition of lab $_{(\text {i.irdy }, i . \text {.trdy }, i . \text { data })}$ immediately.
- $\operatorname{type}(z)=$ queue $_{k}$. Since $N$ is valid, from Definition 2.2, assume without loss of generality that $G=\{i, o\}, \operatorname{chan}(z, \mathrm{in}, 0)=i$, and $\operatorname{chan}(z$, out, 0$)=0$. According to Definition 2.6,

$$
\begin{aligned}
\text { State }(N)= & \{\{i . \text { data }=d, o . \text { data }=e, z . \text { queue }=\mathrm{xs}\} \cup \mathrm{IO} \mid \\
& d \in c(i), e \in c(o), \mathrm{xs} \in Q_{c(i)^{\prime}}^{k} \\
& \mathrm{IO} \subseteq\{i . \mathbf{i r d y}, i . \operatorname{trdy}, o . i \mathbf{i r d y}, o . \operatorname{trd} \mathbf{y}\}, \\
& |\mathrm{xs}|<k \Leftrightarrow i . \operatorname{trdy} \in \mathrm{IO}, \\
& |\mathrm{xs}|>0 \Leftrightarrow o . \mathbf{i r d y} \in \mathrm{IO}, \\
& |\mathrm{xs}|>0 \Rightarrow e=\operatorname{last}(\mathrm{xs})\} .
\end{aligned}
$$

By the definition of $\operatorname{KS}(N)$,

$$
\begin{aligned}
S= & \left\{\left(\mathrm{xs}, i_{\text {irdy }}, i_{\text {trdy }}, d_{,}, o_{\text {irdy }}, o_{\text {trdy }}, e\right) \mid\right. \\
& \mathrm{xs} \in Q_{c(i)}^{k}, i_{\text {irdy }}, i_{\text {trdy }}, o_{\text {irdy }}, o_{\text {trdy }} \in \mathbb{B}, \\
& d \in c(i), e \in c(o),|\mathrm{xs}|<k \Leftrightarrow i_{\text {trdy }} \\
& \left.|\mathrm{xs}|>0 \Leftrightarrow o_{\text {irdy }},|\mathrm{xs}|>0 \Rightarrow e=\operatorname{last}(\mathrm{xs})\right\},
\end{aligned}
$$

and hence $\{L(p) \mid p \in S\}=\left\{\operatorname{lab}_{(p . q u e u e, i . i r d y, i \text {.trdy }, i . \text { data }, 0 . \text { irdy }, o . \text {.trdy }, \text {.data })}(p) \mid p \in\right.$ $S\}=\operatorname{State}(N)$ follows immediately from the definition of

$$
\left.\operatorname{lab}_{(p . \text { queue }, i . i \text { irdy }, i . t r d y, i . d a t a, ~}^{0} \text {.irdy }, o . \operatorname{trdy}, o . \text { data }\right) .
$$

- $\operatorname{type}(z)=$ function. Let $f$ be the data transforming function of $z$. Since $N$ is valid, from Definition 2.2, assume without loss of generality that $G=\{i, o\}$, $\operatorname{chan}(z$, in, 0$)=i$, and $\operatorname{chan}(z$, out, 0$)=0$. According to Definition 2.6,

$$
\begin{aligned}
\text { State }(N)= & \{\{i . \mathbf{d a t a}=d, o . \text { data }=f(d)\} \cup \mathrm{IO} \mid \\
& d \in c(i), \\
& \mathrm{IO} \subseteq\{i . \mathbf{i r d y}, i . \operatorname{trdy}, o . \mathbf{i r d} \mathbf{y}, o . \operatorname{trdy}\}, \\
& (i . \mathbf{i r d} \mathbf{y} \in \mathrm{IO}) \Leftrightarrow(o . \mathbf{i r d y} \in \mathrm{IO}), \\
& (o . \operatorname{trdy} \in \mathrm{IO}) \Leftrightarrow(i . \operatorname{trd} \mathbf{y} \in \mathrm{IO})\} .
\end{aligned}
$$

By the Definition of $\operatorname{KS}(N)$, we have that the set of states of $\operatorname{KS}(N)$ is

$$
\begin{aligned}
S= & \left\{\left(\mathrm{xs}, i_{\text {irdy }}, i_{\text {trdy }}, d, o_{\text {irdy }}, o_{\text {trdy }}, e\right) \in \mathbb{B} \times \mathbb{B} \times c(i) \times \mathbb{B} \times \mathbb{B} \times f(d) \mid\right. \\
& \left.i_{\text {trdy }}=o_{\text {trdy }} \wedge o_{\text {irdy }}=i_{\text {irdy }}\right\},
\end{aligned}
$$

and hence $\{L(p) \mid p \in S\}=\left\{\operatorname{lab}_{(i . i \text { irdy }, i . \operatorname{trdy}, i . \text { data }, . \text { irdy }, 0 . \operatorname{trdy}, 0 . \text { data })}(p) \mid p \in S\right\}=$ State $(N)$ follows from the definition of $\operatorname{lab}_{(i . i r d y, i . t r d y}, i . d a t a, o . i r d y, o . \operatorname{trdy}, o$. data) immediately.

- $\operatorname{type}(z)=$ fork. Since $N$ is valid, from Definition 2.2, assume without loss of generality that $G=\{i, o, u\}, \operatorname{chan}(z$, in, 0$)=i, \operatorname{chan}(z$, out, 0$)=0$, and $\operatorname{chan}(z$, out, 1$)=u$. Let $f$ and $f^{\prime}$ be data transforming functions of $z$. According to Definition 2.6,

$$
\begin{aligned}
\text { State }(N)= & \left\{\left\{i . \text { data }=d, o . \text { data }=f(d), u . \text { data }=f^{\prime}(d)\right\} \cup \mathrm{IO} \mid\right. \\
& d \in c(i), \\
& I \mathrm{O} \subseteq\{i . \mathbf{i r d y}, i . \operatorname{trdy}, o . \mathbf{i r d y}, o . \operatorname{trdy}, u . \mathbf{i r d y}, u . \operatorname{trdy}\}, \\
& (i . \mathbf{i r d y} \in \mathrm{IO}) \wedge(u . \operatorname{trdy} \in \mathrm{IO}) \Leftrightarrow(o . \mathbf{i r d y} \in \mathrm{IO}), \\
& (i . \mathbf{i r d y} \in \mathrm{IO}) \wedge(o . \operatorname{trdy} \in \mathrm{IO}) \Leftrightarrow(u . \mathbf{i r d y} \in \mathrm{IO}), \\
& (o . \operatorname{trdy} \in \mathrm{IO}) \wedge(u . \operatorname{trdy} \in \mathrm{IO}) \Leftrightarrow(i . \operatorname{trdy} \in \mathrm{IO})\} .
\end{aligned}
$$

By the Definition of $\operatorname{KS}(N)$, we have that the set of states of $\operatorname{KS}(N)$ is

$$
\begin{aligned}
S= & \left\{\left(i_{\text {irdy }}, i_{\text {trdy }}, d, o_{\text {irdy }}, o_{\text {trdy }}, e, u_{\text {irdy }}, u_{\text {trdy }}, j\right) \mid\right. \\
& i_{\text {irdy }}, i_{\text {trdy }}, o_{\text {irdy }}, o_{\text {trdy }}, u_{\text {irdy }}, u_{\text {trdy }} \in \mathbb{B}, \\
& d \in c(i), e \in c(o), j \in c(u) \\
& e=f(d) \wedge u=f^{\prime}(d) \wedge o_{\text {irdy }}=i_{\text {irdy }} \wedge u_{\text {trdy }} \\
& \left.u_{\text {irdy }}=i_{\text {irdy }} \wedge o_{\text {trdy }}, i_{\text {trdy }}=o_{\text {trdy }} \wedge u_{\text {trdy }}\right\},
\end{aligned}
$$

and hence

$$
\begin{aligned}
& \{L(p) \mid p \in S\}= \\
& \quad\left\{\text { lab }_{(i . i r d y}, i . t r d y, i . \text { data }, o . \text { irdy }, o . \operatorname{trdy}, o . \text { data }, u . \text {.irdy }, u . \operatorname{trdy}, u .\right. \text { data) } \\
& (p) \mid p \in S\}
\end{aligned}
$$

follows from the definition of

$$
\operatorname{lab}_{(i . \mathbf{i r d y}, i . \operatorname{trdy}, i . \text { data }, o . \operatorname{irdy}, o . t r d y}, o . \text { data }, u . \operatorname{irdy}, u . \operatorname{trdy}, u . \text { data) }
$$

immediately.

- $\operatorname{type}(z)=$ join. Let $h$ be the routing function of $z$. Since $N$ is valid, from Definition 2.2, assume without loss of generality that $G=\{i, j, o\}$, and $\operatorname{chan}(z$, in, 0$)=i$, and $\operatorname{chan}(z$, in, 1$)=j$, and chan $(z$, out, 0$)=o$. According to Definition 2.6,

$$
\begin{aligned}
\text { State }(N)= & \{\{i . \text { data }=d, j . \text { data }=e, o . \text { data }=h(d, e)\} \cup \mathrm{IO} \mid \\
& d \in c(i), e \in c(j), \\
& \mathrm{IO} \subseteq\{i . \mathbf{i r d y}, i . \operatorname{trdy}, j . \mathbf{i r d y}, j . \operatorname{trdy}, o . \mathbf{i r d y}, o . \operatorname{trdy}\}, \\
& (j . \mathbf{i r d y} \in \mathrm{IO}) \wedge(o . \operatorname{trd} \mathbf{y} \in \mathrm{IO}) \Leftrightarrow(i . \operatorname{trdy} \in \mathrm{IO}), \\
& (i . \mathbf{i r d y} \in \mathrm{IO}) \wedge(o . \operatorname{trd} \mathbf{y} \in \mathrm{IO}) \Leftrightarrow(j . \operatorname{trdy} \in \mathrm{IO}), \\
& (i . \mathbf{i r d y} \in \mathrm{IO}) \wedge(j . \mathbf{i r d} \mathbf{y} \in \mathrm{IO}) \Leftrightarrow(o . \mathbf{i r d} \mathbf{y} \in \mathrm{IO})\} .
\end{aligned}
$$

By the Definition of $\operatorname{KS}(N)$, we have that the set of states of $\operatorname{KS}(N)$ is

$$
\begin{aligned}
S= & \left\{\left(i_{\text {irdy }}, i_{\text {trdy }}, d, j_{\text {irdy }}, j_{\text {trdy }}, e, o_{\text {irdy }}, o_{\text {trdy }}, l\right) \mid\right. \\
& i_{\text {trdy }}, j_{\text {irdy }}, j_{\text {trdy }}, o_{\text {irdy }}, o_{\text {trdy }} \in \mathbb{B}, \\
& d \in c(i), e \in c(o), l \in c(u) \\
& l=h(d, e) \wedge i_{\text {trdy }}=j_{\text {irdy }} \wedge o_{\text {trdy }}, \\
& \left.j_{\text {trdy }}=i_{\text {irdy }} \wedge o_{\text {trdy }}, o_{\text {irdy }}=i_{\text {irdy }} \wedge j_{\text {irdy }}\right\},
\end{aligned}
$$

 $p \in S\}=\operatorname{State}(N)$ follows from the definition of
immediately.
$-\operatorname{type}(z)=$ switch. Let $r$ be the routing function of $z$. Since $N$ is valid, from Definition 2.2, assume without loss of generality that $G=\{i, o, u\}$, $\operatorname{chan}(z$, in, 0$)=i, \operatorname{chan}(z$, out, 0$)=0$, and $\operatorname{chan}(z$, out, 1$)=u$. According to Definition 2.6,

$$
\begin{aligned}
\text { State }(N)= & \{\{i . \mathbf{d a t a}=d, o . \text { data }=d, u . \text { data }=d\} \cup \mathrm{IO} \mid \\
& d \in c(i), \\
& \mathrm{IO} \subseteq\{i . \mathbf{i r d y}, i . \operatorname{trd} \mathbf{y}, o . \mathbf{i r d y}, o . \operatorname{trd} \mathbf{y}, u . \mathbf{i r d} \mathbf{y}, u . \operatorname{trdy}\}, \\
& i . \mathbf{i r d y} \in \mathrm{IO} \wedge r(d) \Leftrightarrow o . \mathbf{i r d y} \in \mathrm{IO}, \\
& i . \mathbf{i r d y} \in \mathrm{IO}) \wedge \neg r(d) \Leftrightarrow u . \mathbf{i r d y} \in \mathrm{IO}, \\
& (o . \mathbf{i r d} \mathbf{y} \in \mathrm{IO} \wedge o . \operatorname{trd} \mathbf{y} \in \mathrm{IO}) \vee \\
& (u . \mathbf{i r d} \mathbf{y} \in \mathrm{IO} \wedge u . \operatorname{trd} \mathbf{y} \in \mathrm{IO}) \Leftrightarrow i . \operatorname{trd} \mathbf{y} \in \mathrm{IO}\} .
\end{aligned}
$$

By the Definition of $\operatorname{KS}(N)$, we have that the set of states of $\operatorname{KS}(N)$ is

$$
\begin{aligned}
S= & \left\{\left(i_{\text {irdy }}, i_{\text {trdy }}, d, o_{\text {irdy }}, o_{\text {trdy }}, e, u_{\text {irdy }}, u_{\text {trdy }}, j\right) \mid\right. \\
& i_{\text {irdy }}, i_{\text {trdy }}, o_{\text {irdy }}, o_{\text {trdy }}, u_{\text {irdy }}, u_{\text {trdy }} \in \mathbb{B} \\
& d \in c(i), e \in c(o), j \in c(u), \\
& r(d) \Rightarrow j=d \wedge \neg r(d) \Rightarrow j=e \wedge u_{\text {irdy }}=i_{\text {irdy }} \wedge \neg r(d), \\
& \left.i_{\text {trdy }}=\left(o_{\text {irdy }} \wedge o_{\text {trdy }}\right) \vee\left(u_{\text {irdy }} \wedge u_{\text {trdy }}\right)\right\},
\end{aligned}
$$

and hence

$$
\begin{aligned}
& \{L(p) \mid p \in S\}=
\end{aligned}
$$

$$
\begin{aligned}
& \text { State( } N \text { ) }
\end{aligned}
$$

follows from the definition of

$$
\operatorname{lab}_{(. i \mathbf{i r d y}, i . \operatorname{trdy}, i . d a t a, o . \operatorname{irdy}, o . t r d y}, o . \text { data }, u . \operatorname{irdy}, u . \operatorname{trdy}, u . \text { data) }
$$

immediately.

- $\operatorname{type}(z)=$ merge. Since $N$ is valid, from Definition 2.2, assume without loss of generality that $G=\{i, j, o\}, \operatorname{chan}(z$, in, 0$)=i, \operatorname{chan}(z$, in, 1$)=j$, and chan $(z$, out, 0$)=o$. According to Definition 2.6,

$$
\begin{aligned}
\text { State }(N)= & \{\{i . \text { data }=d, j . \text { data }=e, o . \text { data }=l\} \cup \mathrm{IO} \mid \\
& d \in c(i), e \in c(j), l \in c(o), u \in \mathbb{B}, \\
& \mathrm{IO} \subseteq\{i . \mathbf{i r d y}, i . \operatorname{trdy}, j . \mathbf{i r d} \mathbf{y}, j . \operatorname{trd} \mathbf{y}, o . \mathbf{i r d y}, o . \operatorname{trd} \mathbf{y}\}, \\
& o . \mathbf{i r d y} \in s \Leftrightarrow u \wedge i . \mathbf{i r d y} \in s \vee \neg u \wedge j . \mathbf{i r d y} \in s, \\
& i . \operatorname{trdy} \in s \Leftrightarrow u \wedge o . \operatorname{trdy} \in s \wedge i . \mathbf{i r d y} \in s, \\
& j . \operatorname{trdy} \in s \Leftrightarrow \neg u \wedge o . \operatorname{trdy} \in s \wedge j . \mathbf{i r d y} \in s, \\
& u \wedge i . \mathbf{i r d y} \in s \Rightarrow l=d, \\
& \neg u \wedge j . \mathbf{i r d y} \in s \Rightarrow l=e\} .
\end{aligned}
$$

By the Definition of $\operatorname{KS}(N)$, we have that the set of states of $\operatorname{KS}(N)$ is

$$
\begin{aligned}
S= & \left\{\left(u, i_{\text {irdy }}, i_{\text {trdy }}, d, j_{\text {irdy }}, j_{\text {trdy }}, e, o_{\text {irdy }}, o_{\text {trdy }}, l\right) \mid\right. \\
& u, i_{\text {irdy }}, i_{\text {trdy }}, j_{\text {irdy }}, j_{\text {trdy }}, o_{\text {irdy }}, o_{\text {trdy }} \in \mathbb{B}, \\
& d \in c(i), e \in c(j), l \in c(o), \\
& o_{\text {irdy }}=i_{\text {irdy }} \vee j_{\text {irdy }}, i_{\text {trdy }}=u \wedge o_{\text {trdy }} \wedge i_{\text {irdy }}, \\
& \left.j_{\text {trdy }}=\neg u \wedge o_{\text {trdy }} \wedge j_{\text {irdy }}, u \wedge i_{\text {irdy }} \Rightarrow l=d, \neg u \wedge j_{\text {irdy }} \Rightarrow l=e\right\},
\end{aligned}
$$

and hence

$$
\begin{aligned}
& \{L(p) \mid p \in S\}= \\
& \left\{\operatorname{lab}_{(u, i . i \mathbf{i r d y}, i . t r d y, i . d a t a, j . \operatorname{irdy}, j . \operatorname{trdy}, j \text {.data }, 0 . \operatorname{irdy}, o . \operatorname{trdy}, o . \text { data })}(p) \mid p \in S\right\}= \\
& \operatorname{State}(N)
\end{aligned}
$$

follows from the definition of

$$
\operatorname{lab}_{(u, i . i \mathbf{i r d y}, i . \operatorname{trdy}, i . \mathrm{data}, j . i \mathbf{r d y}, j . \operatorname{trdy}, j \text {.data }, 0 . \mathrm{irdy}, 0 . \mathrm{trdy}, o . \text { data })}
$$

immediately.

- Inductive step. Assume $N$ is such that $|P|>1$. From Lemma 2.13 it holds that $N$ can be split into two valid compatible networks $N^{\prime}$ and $N^{\prime \prime}$, such that $N=N^{\prime}$ ॥ $N^{\prime \prime}$. Let $N^{\prime}$ and $N^{\prime \prime}$ be such. Assume that $N^{\prime}=\left(P^{\prime}, G^{\prime}, C^{\prime}, c^{\prime}\right.$, chan', type) and $N^{\prime \prime}=\left(P^{\prime \prime}, G^{\prime \prime}, C^{\prime \prime}, c^{\prime \prime}\right.$, chan", type $)$. Let $M\left(N^{\prime}\right)=\left(\operatorname{State}\left(N^{\prime}\right), \operatorname{Init}\left(N^{\prime}\right), \operatorname{Next}\left(N^{\prime}\right)\right)$ and $M\left(N^{\prime \prime}\right)=\left(\operatorname{State}\left(N^{\prime \prime}\right), \operatorname{Init}\left(N^{\prime \prime}\right), \operatorname{Next}\left(N^{\prime \prime}\right)\right)$ be the signal structures for $N^{\prime}$ and $N^{\prime \prime}$ respectively. Let $\mathrm{KS}\left(N^{\prime}\right)=\left(S^{\prime}, I^{\prime}, \rightarrow^{\prime}, \mathrm{AP}^{\prime}, L^{\prime}\right)$ and $\mathrm{KS}\left(N^{\prime \prime}\right)=\left(S^{\prime \prime}, I^{\prime \prime}, \rightarrow^{\prime \prime}\right.$ , $\mathrm{AP}^{\prime \prime}, L^{\prime \prime}$ ) be two KSs representing $N^{\prime}$ and $N^{\prime \prime}$ respectively. The induction hypothesis is that $\operatorname{State}\left(N^{\prime}\right)=\left\{L^{\prime}\left(p^{\prime}\right) \mid p^{\prime} \in S^{\prime}\right\}$ and $\operatorname{State}\left(N^{\prime \prime}\right)=\left\{L^{\prime \prime}\left(p^{\prime \prime}\right) \mid p^{\prime \prime} \in\right.$ $\left.S^{\prime \prime}\right\}$. We prove that State $(N)=\{L(p) \mid p \in S\}$.
$\subseteq$ First, we show that for all $s \in \operatorname{State}(N)$, there is $p \in \mathrm{~S}$, such that $s=L(p)$. Fix an arbitrary $s \in \operatorname{State}(N)$. From Lemma 2.14, it follows that there are $s^{\prime} \in \operatorname{State}\left(N^{\prime}\right)$ and $s^{\prime \prime} \in \operatorname{State}\left(N^{\prime \prime}\right)$, such that $s=s^{\prime} \cup s^{\prime \prime}$; fix such $s^{\prime}$ and $s^{\prime \prime}$. By the induction hypothesis, there are $p^{\prime} \in S^{\prime}$ and $p^{\prime \prime} \in S^{\prime \prime}$, such that $s^{\prime}=L^{\prime}\left(p^{\prime}\right)$ and $s^{\prime \prime}=L^{\prime \prime}\left(p^{\prime \prime}\right)$; fix such $p^{\prime}$ and $p^{\prime \prime}$. Since $s=s^{\prime} \cup s^{\prime \prime}$, we conclude $s=L^{\prime}\left(p^{\prime}\right) \cup L^{\prime \prime}\left(p^{\prime \prime}\right)$. Since $N=N^{\prime} ॥ N^{\prime \prime}$, by Definition 2.18, we have $\operatorname{KS}(N)=\operatorname{KS}\left(N^{\prime}\right)| | \operatorname{KS}\left(N^{\prime \prime}\right)$. Therefore, according to Definition 2.17, $\left(p^{\prime}, p^{\prime \prime}\right) \in S$ with $L\left(p^{\prime}, p^{\prime \prime}\right)=L\left(p^{\prime}\right) \cup L\left(p^{\prime \prime}\right)$.
$\supseteq$ Now we show that for all $p \in S$, there is $s \in \operatorname{State}(N)$, such that $L(p)=s$. Fix an arbitrary $p \in S$. Since $N=N^{\prime}{ }^{\|} N^{\prime \prime}$, by Definition 2.18, we have $\operatorname{KS}(N)=\operatorname{KS}\left(N^{\prime}\right) \| \operatorname{KS}\left(N^{\prime \prime}\right)$. We therefore conclude that there are $p^{\prime} \in S^{\prime}$ and $p^{\prime \prime} \in S^{\prime \prime}$, such that $p=\left(p^{\prime}, p^{\prime \prime}\right)$ and hence $L(p)=L^{\prime}\left(p^{\prime}\right) \cup L^{\prime \prime}\left(p^{\prime \prime}\right)$. From the induction hypothesis, there are $s^{\prime} \in \operatorname{State}\left(N^{\prime}\right)$ and $s^{\prime \prime} \in \operatorname{State}\left(N^{\prime \prime}\right)$, such that $L^{\prime}\left(p^{\prime}\right)=s^{\prime}$ and $L^{\prime \prime}\left(p^{\prime \prime}\right)=s^{\prime \prime}$; fix such $s^{\prime}$ and $s^{\prime \prime}$. Since $L(p)=L^{\prime}\left(p^{\prime}\right) \cup L^{\prime \prime}\left(p^{\prime \prime}\right)$, we conclude $L(p)=s^{\prime} \cup s^{\prime \prime}$. Let $s=s^{\prime} \cup s^{\prime \prime}$. From the induction hypothesis, we know that $s^{\prime}$ and $s^{\prime \prime}$ satisfy all the validity conditions of Definition 2.5
to be signal states. Since $p$ can be obtained by composing $p^{\prime}$ and $p^{\prime}$, we know that $s^{\prime}$ and $s^{\prime \prime}$ agree on the shared signal labels. Hence, we conclude that $s$ satisfies all the validity conditions of Definition 2.5. Therefore, we conclude that $s \in \operatorname{State}(N)$ and $L(p)=s$.

Consequently, we have the following corollary that we use in the proof of the theorem that follows further.

Corollary 2.22. Given valid $x$ MAS networks $N, N^{\prime}$, and $N^{\prime \prime}$, such that $N=N^{\prime}{ }^{\prime} N^{\prime \prime}$, let State $(N)$, State $\left(N^{\prime}\right)$, and State $\left(N^{\prime \prime}\right)$ be the sets of signal states of the corresponding signal structures, and let $K S(N)=(S, I, \rightarrow, A P, L), K S\left(N^{\prime}\right)=\left(S^{\prime}, I^{\prime}, \rightarrow{ }^{\prime}, A P^{\prime}, L^{\prime}\right), K S\left(N^{\prime \prime}\right)=$ $\left(S^{\prime \prime}, I^{\prime \prime}, \rightarrow^{\prime \prime}, A P^{\prime \prime}, L^{\prime \prime}\right)$. For all $s \in \operatorname{State}(N), s^{\prime} \in \operatorname{State}\left(N^{\prime}\right), s^{\prime \prime} \in \operatorname{State}\left(N^{\prime \prime}\right)$, if $s=s^{\prime} \cup s^{\prime \prime}$, then there are $p \in S, p^{\prime} \in S^{\prime}, p^{\prime \prime} \in P^{\prime \prime}$, such that:

- $L(p)=L^{\prime}\left(p^{\prime}\right) \cup L^{\prime \prime}\left(p^{\prime \prime}\right)$,
- $s^{\prime}=L^{\prime}\left(p^{\prime}\right)$,
- $s^{\prime \prime}=L^{\prime \prime}\left(p^{\prime \prime}\right)$,
- $p=\left(p^{\prime}, p^{\prime \prime}\right)$.

Proof. Fix arbitrary $s \in \operatorname{State}(N), s^{\prime} \in \operatorname{State}\left(N^{\prime}\right), s^{\prime \prime} \in \operatorname{State}\left(N^{\prime \prime}\right)$, such that $s=s^{\prime} \cup s^{\prime \prime}$. By Lemma 2.21, there is $p \in S$, such that $s=L(p)$. Since $N=N^{\prime}$ ॥ $N^{\prime \prime}$, by Definition 2.18, we have $\operatorname{KS}(N)=\operatorname{KS}\left(N^{\prime}\right)| | \operatorname{KS}\left(N^{\prime \prime}\right)$. Hence, according to Definition 2.17, there are $p^{\prime} \in P^{\prime}, p^{\prime \prime} \in P^{\prime \prime}$, such that $p=\left(p^{\prime}, p^{\prime \prime}\right)$ and $L(p)=L^{\prime}\left(p^{\prime}\right) \cup L^{\prime \prime}\left(p^{\prime \prime}\right)$. By Lemma 2.19, $L$ is bijective. Using Lemma 2.21, we conclude $p=\left(p^{\prime}, p^{\prime \prime}\right), s^{\prime}=L^{\prime}\left(p^{\prime}\right)$, and $s^{\prime \prime}=L^{\prime \prime}\left(p^{\prime \prime}\right)$.

Finally, for a given valid xMAS network, we prove that its signal structure and Kripke Structure are bisimilar.

Theorem 2.23. For every valid $x M A S$ network $N, M(N) \leftrightarrows K S(N)$.

Proof. We prove the stronger statement that for every valid xMAS network $N$, with $M(N)=(\operatorname{State}(N), \operatorname{Init}(N), \operatorname{Next}(N))$ and $\operatorname{KS}(N)=(S, I, \rightarrow, A P, L)$, the relation

$$
R_{N}=\{(s, p) \mid s \in \operatorname{State}(N), p \in S, s=L(p)\}
$$

is a bisimulation relation.
We prove that $R_{N}$ is a bisimulation relation by induction on the number of primitives in $N$.

- Base case. In this case $N=(P, G, C, c$, chan, type $)$ is such that $|P|=1$. Assume $P=\{z\}$. Let $M(N)=(\operatorname{State}(N), \operatorname{Init}(N), \operatorname{Next}(N))$ and $\operatorname{KS}(N)=(S, I, \rightarrow, \operatorname{AP}, L)$. Fix arbitrary $s \in \operatorname{State}(N), p \in S$ such that $s R_{N} p$. Note that $s=L(p)$ by the definition of $R_{N}$. We check the condition on initial states and the transfer conditions for each type of $z$.
$-\operatorname{type}(z)=$ source. Since $\operatorname{State}(N)=\operatorname{Init}(N)$ and $S=I$ in this case, it immediately follows that $s \in \operatorname{Init}(N)$ if and only if $p \in I$.

Further we check the transfer conditions. We first show that for all $s^{\prime} \in$ State $(N)$, such that $\left(s, s^{\prime}\right) \in \operatorname{Next}(N)$, there exists $p^{\prime} \in S$ such that $p \rightarrow p^{\prime}$ and $s^{\prime} R_{N} p^{\prime}$. We fix $s^{\prime} \in \operatorname{State}(N)$ such that $\left(s, s^{\prime}\right) \in \operatorname{Next}(N)$ and distinguish two cases:

* $s=\{o$. irdy, o.data $=d\}$. By the definition of $R_{N}, L(p)=s$, hence according to Lemma 2.19, $p=$ (true, false, $d$ ). By the definition of $\operatorname{Next}(N)$, (o.irdy $\left.\in s^{\prime}\right) \wedge$ (o.data $\left.=d \in s^{\prime}\right)$, hence either $s^{\prime}=s$ or $s^{\prime}=\{o$. irdy $, o . \operatorname{trdy}, o$. data $=d\}$.

If $s^{\prime}=s$, this can be mimicked by (true, false, $d$ ) $\rightarrow$ (true, false, $d$ ) according to rule Src2. By the definition of $L, L($ (true, false, $d))=s^{\prime}$, hence $s^{\prime} R_{N}$ (true, false, $d$ ).

If $s^{\prime}=\{o$. irdy $, o . t r d y, o . d a t a=d\}$, this can be mimicked by

$$
\text { (true, false, } d \text { ) } \rightarrow \text { (true, true, } d \text { ) }
$$

according to rule $\operatorname{Src} 2$. By the definition of $L$ we have $L($ (true, true, $d))=$ $s^{\prime}$, hence $s^{\prime} R_{N}$ (true, true,$d$ ).
$* s$ is such that $o . \operatorname{irdy} \in s \Rightarrow o . \operatorname{trdy} \in s$. Let $p=\left(o_{\text {irdy }}, o_{\text {trdy }}, d\right)$; note that since $s R_{N} p, L(p)=s$, and according to Lemma 2.19, $o_{\text {irdy }} \Longrightarrow o_{\text {trdy }}$. Let $s^{\prime}$ be arbitrary such that $\left(s, s^{\prime}\right) \in \operatorname{Next}(N)$. Note that according to the definition of $\operatorname{Next}(N), s^{\prime}$ can be any state in State $(N)$. According to rule Src1, $p \rightarrow\left(o_{\text {irdy }}^{\prime} o_{\text {trdy }}^{\prime}, d^{\prime}\right)$ for any $o_{\text {irdy }}^{\prime}, o_{\text {trdy }}^{\prime}, d^{\prime}$, therefore we can select $o_{\text {irdy }}^{\prime}, o_{\text {trdy }}^{\prime} d^{\prime}$ such that $L\left(\left(o_{\text {irdy }}^{\prime}, o_{\text {trdy }}^{\prime}, d^{\prime}\right)\right)=s^{\prime}$, and $s^{\prime} R_{N}\left(o_{\text {irdy }}^{\prime}, o_{\text {trdy }}^{\prime} d^{\prime}\right)$.

Now we show that for all $p^{\prime} \in S$, such that $p \rightarrow p^{\prime}$, there exists $s^{\prime} \in \operatorname{State}(N)$ such that $\left(s, s^{\prime}\right) \in \operatorname{Next}(N)$ and $s^{\prime} R_{N} p^{\prime}$. We fix $p^{\prime} \in S$ such that $p \rightarrow p^{\prime}$ and distinguish cases based on $p$.

* $p=($ true, false,$d)$. By definition of $L, L(p)=\{o$. irdy, o.data $=d\}$, and according to the definition of $R_{N}, s=\{o . i$ irdy, $o$. data $=d\}$. According to Src2, $p^{\prime}=($ true, false,$d)$ or $p^{\prime}=($ true, true, $d)$. We distinguish both cases.

If $p^{\prime}=($ true, false,$d), L\left(p^{\prime}\right)=\{0$. irdy, o.data $=d\}$. According to the definition of $\operatorname{Next}(N), s^{\prime}=\{0$. irdy, o.data $=d\} \in \operatorname{Next}(N)$ is a valid successor of $s$ since it satisfies $o$.irdy $\in s^{\prime} \wedge o$.trdy $\notin s^{\prime}$, and by definition of $R_{N}, p^{\prime} R s^{\prime}$.

If $p^{\prime}=($ true, true,$d), L\left(p^{\prime}\right)=\{o$. irdy $, o . \operatorname{trdy}, o$. data $=d\}$. According to the definition of $\operatorname{Next}(N), s^{\prime}=\{0$. irdy, o.trdy, o.data $=d\} \in \operatorname{Next}(N)$ is a valid successor of $s$ since it satisfies $o . i r d y \in s^{\prime} \wedge o . t r d y \in s^{\prime}$, and by definition of $R, p^{\prime} R s^{\prime}$.

* $p=\left(o_{\text {irdy }}, o_{\text {trdy }}, d\right)$ such that $o_{\text {irdy }} \Rightarrow o_{\text {trdy }}$. According to Src1, $p^{\prime}$ can be any state in $S$. Since $s R_{N} p, s$ is such that $o . i r d y \in s$ implies $o$.trdy $\in s$, according to $\operatorname{Next}(N),\left(s, s^{\prime}\right) \in \operatorname{Next}(N)$ for all $s^{\prime} \in \operatorname{State}(N)$. We can therefore select $s^{\prime}$ such that $L\left(p^{\prime}\right)=s^{\prime}$, and by definition of $R_{N}, s^{\prime} R_{N} p^{\prime}$.
$-\operatorname{type}(z)=$ sink. The proof is analogous to the previous case.
- $\operatorname{type}(z)=$ queue. According to Definition 2.7, $\operatorname{Init}(N)$ comprises of all signal states $s \in \operatorname{Init}(N)$, for which it holds that $z$. queue $=[] \in s, i . \operatorname{trdy} \in s$, and $o$.irdy $\notin s$. From the definition of $I$, we have

$$
I=\left\{\left(\mathrm{xs}, i_{\text {irdy }}, i_{\text {trdy }}, d, o_{\text {irdy }}, o_{\text {trdy }}, e\right) \in S \mid \mathrm{xs}=[], i_{\text {trdy }}=\text { true }, o_{\text {irdy }}=\text { false }\right\} .
$$

Therefore, using the definition of $L$ we conclude that $s \in \operatorname{Init}(N)$ if and only if $p \in I$.

Further we check the transfer conditions. We fix $s^{\prime} \in \operatorname{State}(N)$ such that $\left(s, s^{\prime}\right) \in \operatorname{Next}(N)$. To show that there is $p^{\prime} \in S$ such that $p \rightarrow p^{\prime}$ and $s^{\prime} R_{N} p^{\prime}$, we distinguish cases based on the definition of $\operatorname{Next}(N)$.

* $s$ is such that $\neg(i . i r d y \in s \wedge i . \operatorname{trdy} \in s)$ and $\neg(o . i r d y \in s \wedge o . \operatorname{trdy} \in$ s) $\wedge z$.queue $=\mathrm{xs} \in s, s^{\prime}$ is such that $z$.queue $=\mathrm{xs} \in s^{\prime}$. By the definition of $R_{N}, L(p)=s$. Hence, according to Lemma 2.19,

$$
p=\left(\mathrm{xs}, i_{\mathrm{irdy}}, i_{\text {trdy }}, d, o_{\mathbf{i r d y}}, o_{\text {trdy }}, e\right),
$$

for some $i_{\text {irdy }}, i_{\text {trdy }}, o_{\text {irdy }}, o_{\text {trdy }} \in \mathbb{B}, d \in c(i), e \in c(0)$, such that $\neg\left(i_{\text {irdy }} \wedge\right.$ $\left.i_{\text {trdy }}\right) \wedge \neg\left(o_{\text {irdy }} \wedge o_{\text {trdy }}\right)$.

According to Lemma 2.21, there is $p^{\prime} \in S$, such that $s^{\prime}=L\left(p^{\prime}\right)$; let $p^{\prime}$ be such. We have $s^{\prime} R_{N} p^{\prime}$ by the definition of $R_{N}$. According to the rule Q1, a successor of $p$ is such that

$$
\left(\mathrm{xs}, i_{\text {irdy }}^{\prime}, i_{\text {trdy }}^{\prime}, d^{\prime}, o_{\text {irdy }}^{\prime}, o_{\text {trdy }}^{\prime}, e^{\prime}\right)
$$

for some $i_{\text {irdy }}^{\prime}, i_{\text {trdy }}^{\prime}, o_{\text {irdy }}^{\prime}, o_{\text {trdy }}^{\prime} \in \mathbb{B}, d^{\prime} \in c(i), e^{\prime} \in c(o)$. Thus, using the definition of $L$, we conclude $p \rightarrow p^{\prime}$.

* $s$ is such that $(i$. irdy $\in s \wedge i$. trdy $\in s \wedge i$. data $=d \in s)$, and $\neg(o . i \mathbf{i r d y} \in s \wedge$ $o . \operatorname{trdy} \in s)$, and $z$.queue $=\mathrm{xs} \in s, s^{\prime}$ is such that $z$.queue $=(d: \mathrm{xs}) \in s^{\prime}$. By the definition of $R_{N}, L(p)=s$. Hence, according to Lemma 2.19,

$$
p=\left(\mathrm{xs}, i_{\mathbf{i r d y}}, i_{\text {trdy }}, d, o_{\mathbf{i r d y}}, o_{\mathbf{t r d y}}, e\right)
$$

for some $i_{\text {irdy }}, i_{\text {trdy }}, o_{\text {irdy }}, o_{\text {trdy }} \in \mathbb{B}, d \in c(i), e \in c(0)$, and $\left(i_{\text {irdy }} \wedge i_{\text {trdy }}\right) \wedge$ $\neg\left(o_{\text {irdy }} \wedge o_{\text {trdy }}\right)$. According to Lemma 2.21, there is $p^{\prime} \in S$, such that $s^{\prime}=L\left(p^{\prime}\right)$; let $p^{\prime}$ be such. We have $s^{\prime} R_{N} p^{\prime}$ by the definition of $R_{N}$. According to the rule Q2, a successor of $p$ is such that

$$
\left((d: \mathrm{xs}), i_{\text {irdy }}^{\prime}, i_{\text {trdy }}^{\prime}, d^{\prime}, o_{\text {irdy }}^{\prime}, o_{\text {trdy }}^{\prime}, e^{\prime}\right)
$$

for some $i_{\text {irdy }}^{\prime}, i_{\text {trdy }}^{\prime}, o_{\text {irdy }}^{\prime}, o_{\text {trdy }}^{\prime} \in \mathbb{B}, d^{\prime} \in c(i), e^{\prime} \in c(o)$. Thus, using the definition of $L$, we conclude $p \rightarrow p^{\prime}$.

* $s$ is such that $\neg(i . i r d y \in s \wedge i . \operatorname{trdy} \in s)$, and ( $o . \mathbf{i r d y} \in s \wedge o . \operatorname{trdy} \in s$ ), and $z$.queue $=\mathrm{xs} \in s, s^{\prime}$ is such that $z$.queue $=(\operatorname{rtail}(\mathrm{xs})) \in s^{\prime}$. By the definition of $R_{N}, L(p)=s$. Hence, according to Lemma 2.19,

$$
p=\left(\mathrm{xs}, i_{\text {irdy }}, i_{\text {trdy }}, d, o_{\text {irdy }}, o_{\text {trdy }}, e\right),
$$

for some $i_{\text {irdy }}, i_{\text {trdy }}, o_{\text {irdy }}, o_{\text {trdy }} \in \mathbb{B}, d \in c(i), e \in c(0)$, such that $\neg\left(i_{\text {irdy }} \wedge\right.$ $\left.i_{\text {trdy }}\right) \wedge\left(o_{\text {irdy }} \wedge o_{\text {trdy }}\right)$. According to Lemma 2.21, there is $p^{\prime} \in S$, such that $s^{\prime}=L\left(p^{\prime}\right)$; let $p^{\prime}$ be such. We have $s^{\prime} R_{N} p^{\prime}$ by the definition of $R_{N}$. According to the rule Q3, a successor of $p$ is such that

$$
\left((\text { raial }(\mathrm{xs})), i_{\mathrm{irdy}}^{\prime}, i_{\text {trdy }}^{\prime}, d^{\prime}, o_{\mathrm{irdy}}^{\prime}, o_{\mathrm{trdy}}^{\prime}, e^{\prime}\right),
$$

for some $i_{\text {irdy }}^{\prime}, i_{\text {trdy }}^{\prime}, o_{\text {irdy }}^{\prime}, o_{\text {trdy }}^{\prime} \in \mathbb{B}, d^{\prime} \in c(i), e^{\prime} \in c(o)$. Thus, using the definition of $L$, we conclude $p \rightarrow p^{\prime}$.

* $s$ is such that (i.irdy $\in s \wedge i$. .trdy $\in s \wedge i$. data $=d \in s$ ), and (o.irdy $\in$ $s \wedge o . t r d y \in s)$, and $z$.queue $=x s \in s, s^{\prime}$ is such that $z$.queue $=(d:$ $($ rtail $(\mathrm{xs}))) \in s^{\prime}$. By the definition of $R, L(p)=s$. Hence, according to Lemma 2.19,

$$
p=\left(\mathrm{xs}, i_{\text {irdy }}, i_{\text {trdy }}, d, o_{\text {irdy }}, o_{\text {trdy }}, e\right),
$$

for some $i_{\text {irdy }}, i_{\text {trdy }}, o_{\text {irdy }}, o_{\text {trdy }} \in \mathbb{B}, d \in c(i), e \in c(0)$, and $\left(i_{\text {irdy }} \wedge i_{\text {trdy }}\right) \wedge$ $\left(o_{\text {irdy }} \wedge o_{\text {trdy }}\right)$. According to Lemma 2.21, there is $p^{\prime} \in S$, such that $s^{\prime}=L\left(p^{\prime}\right)$; let $p^{\prime}$ be such. We have $s^{\prime} R_{N} p^{\prime}$. According to the rule Q4, a successor of $p$ is such that

$$
\left((d: \operatorname{rtail}(\mathrm{xs})), i_{\text {irdy }}^{\prime}, i_{\text {trdy }}^{\prime}, d^{\prime}, o_{\text {irdy }}^{\prime}, o_{\text {trdy }}^{\prime}, e^{\prime}\right)
$$

for some $i_{\text {irdy }}^{\prime}, i_{\text {trdy }}^{\prime}, o_{\text {irdy }}^{\prime}, o_{\text {trdy }}^{\prime} \in \mathbb{B}, d^{\prime} \in c(i), e^{\prime} \in c(o)$. Thus, using the definition of $L$, we conclude $p \rightarrow p^{\prime}$.

Now we show that for all $p^{\prime} \in S$, such that $p \rightarrow p^{\prime}$, there exists $s^{\prime} \in$ State $(N)$ such that $\left(s, s^{\prime}\right) \in \operatorname{Next}(N)$ and $s^{\prime} R_{N} p^{\prime}$. Fix $p^{\prime}$ such that $p \rightarrow p^{\prime}$. Without loss of generality, assume $p=\left(\mathrm{xs}, i_{\text {irdy }}, i_{\text {trdy }}, d, o_{\text {irdy }}, o_{\text {trdy }}, e\right)$ and $p^{\prime}=\left(\mathrm{xs}, i_{\text {irdy }}^{\prime}, i_{\text {trdy }}^{\prime}, d^{\prime}, o_{\text {irdy }}^{\prime}, o_{\text {trdy }}^{\prime}, e^{\prime}\right)$. We distinguish the following cases based on the rules Q1, Q2, Q3, and Q4.

* Assume $\neg\left(i_{\text {irdy }} \wedge i_{\text {trdy }}\right)$, and $\neg\left(o_{\text {irdy }} \wedge o_{\text {trdy }}\right)$, and xs $^{\prime}=$ xs. By the definition of $R_{N}, s=L(p)$. Hence, according to Lemma 2.19, $s$ is such that $\neg(i$. irdy $\in s \wedge i$.trdy $\in s) \wedge \neg(o$. irdy $\in s \wedge o$. trdy $\in s) \wedge(z$.queue $=$ xs $\in s$ ). By Lemma 2.21, there is $s^{\prime} \in \operatorname{State}(N)$, such that $s^{\prime}=L\left(p^{\prime}\right)$. Let $s^{\prime}$ be such. Note, that $s^{\prime} R_{N} p^{\prime}$. According to Definition 2.8, a successor of $s$ is such that $(z$.queue $=x s \in s) \wedge\left(z\right.$.queue $\left.=x s \in s^{\prime}\right)$. Using the definition of $L$, we conclude $\left(s, s^{\prime}\right) \in \operatorname{Next}(N)$.
* Assume $\left(i_{\text {irdy }} \wedge i_{\text {trdy }}\right)$, and $\neg\left(o_{\text {irdy }} \wedge o_{\text {trdy }}\right)$, and xs $^{\prime}=(d:$ xs $)$. By the definition of $R_{N}, s=L(p)$. Hence, according to Lemma 2.19, $s$ is such that $(i . i \mathbf{i r d y} \in s \wedge i . \operatorname{trdy} \in s \wedge(o$. data $=d) \in s) \wedge \neg(o$. irdy $\in s \wedge o . \operatorname{trdy} \in$
s) $\wedge(z$. queue $=\mathrm{xs} \in s)$. By Lemma 2.21, there is $s^{\prime} \in \operatorname{State}(N)$, such that $s^{\prime}=L\left(p^{\prime}\right)$. Let $s^{\prime}$ be such. Note, that $s^{\prime} R_{N} p^{\prime}$. According to Definition 2.8, a successor of $s$ is such that $z$.queue $=x s \in s \wedge$ $z$. queue $=(d: x s) \in s^{\prime}$. Thus, using the definition of $L$, we conclude $\left(s, s^{\prime}\right) \in \operatorname{Next}(N)$.
* Assume $\neg\left(i_{\text {irdy }} \wedge i_{\text {trdy }}\right)$, and ( $\left.o_{\text {irdy }} \wedge o_{\text {trdy }}\right)$, and xs $^{\prime}=\operatorname{rtail}(\mathrm{xs})$. By the definition of $R_{N}, s=L(p)$. Hence, according to Lemma 2.19, $s$ is such that $\neg(i$. irdy $\in s \wedge i . \operatorname{trdy} \in s) \wedge(o$. irdy $\in s \wedge o . \operatorname{trd} \mathbf{y} \in s) \wedge(z$. queue $=$ xs $\in s$ ). Fix an arbitrary $p$, such that $p \rightarrow p^{\prime}$. By Lemma 2.21, there is $s^{\prime} \in \operatorname{State}(N)$, such that $s^{\prime}=L\left(p^{\prime}\right)$. Let $s^{\prime}$ be such. Note, that $s^{\prime} R_{N} p^{\prime}$. According to Definition 2.8, a successor of $s$ is such that $z$. queue $=x s \in s \wedge z$.queue $=\operatorname{rtail}(\mathrm{xs}) \in s^{\prime}$. Thus, using the definition of $L$, we conclude $\left(s, s^{\prime}\right) \in \operatorname{Next}(N)$.
* Assume $\left(i_{\text {irdy }} \wedge i_{\text {trdy }}\right)$, and $\left(o_{\text {irdy }} \wedge o_{\text {trdy }}\right)$, and $\mathrm{xs}^{\prime}=(d:$ rtail $(\mathrm{xs}))$. By the definition of $R_{N}, s=L(p)$. Hence, according to Lemma 2.19, $s$ is such that $(i$. irdy $\in s \wedge i . \operatorname{trdy} \in s \wedge i$. data $=d \in s) \wedge(o . i r d y \in s \wedge o . \operatorname{trdy} \in$ s) $\wedge(z$. queue $=\mathrm{xs} \in s)$. By Lemma 2.21, there is $s^{\prime} \in \operatorname{State}(N)$, such that $s^{\prime}=L\left(p^{\prime}\right)$. Let $s^{\prime}$ be such. Note, that $s^{\prime} R_{N} p^{\prime}$. By Definition 2.8, a successor of $s$ is such that $z$.queue $=\mathrm{xs} \in s \wedge z$.queue $=(d:$ rtail $(\mathrm{xs})) \in$ $s^{\prime}$. Thus, using the definition of $L$, we conclude $\left(s, s^{\prime}\right) \in \operatorname{Next}(N)$.
- $\operatorname{type}(z)=$ merge. Since $\operatorname{State}(N)=\operatorname{Init}(N)$ and $S=I$ in this case, it immediately follows that $s \in \operatorname{Init}(N)$ if and only if $p \in I$.

Further we check the transfer conditions. We first show that for all $s^{\prime} \in$ State $(N)$, such that $\left(s, s^{\prime}\right) \in \operatorname{Next}(N)$, there exists $p^{\prime} \in S$ such that $p \rightarrow p^{\prime}$ and $s^{\prime} R_{N} p^{\prime}$. Fix $s^{\prime} \in \operatorname{State}(N)$ such that $\left(s, s^{\prime}\right) \in \operatorname{State}(N)$. According to Lemma 2.21, there is $p^{\prime} \in S$, such that $s^{\prime}=L\left(p^{\prime}\right)$; let $p^{\prime}$ be such. Hence $s^{\prime} R_{N} p^{\prime}$ by the definition of $R_{N}$. From the rule Mrg1, for all $q \in S, p \rightarrow q$. Thus, $p \rightarrow p^{\prime}$.

Now we show that for all $p^{\prime} \in S$, such that $p \rightarrow p^{\prime}$, there exists $s^{\prime} \in \operatorname{State}(N)$ such that $\left(s, s^{\prime}\right) \in \operatorname{Next}(N)$ and $s^{\prime} R_{N} p^{\prime}$. Fix $p^{\prime} \in S$ such that $p \rightarrow p^{\prime}$. According to Lemma 2.21, there is $s \in \operatorname{State}(N)$, such that $s^{\prime}=L\left(p^{\prime}\right)$. Let $s^{\prime}$ be such. Note that by the definition of $R_{N}, s^{\prime} R_{N} p^{\prime}$. By Definition 2.8, for all $q \in \operatorname{State}(N),(s, q) \in \operatorname{Next}(N)$. Thus, $\left(s, s^{\prime}\right) \in \operatorname{Next}(N)$.
$-\operatorname{type}(z) \in\{$ function, fork, join, switch\}. Proofs are analogous to the case $\operatorname{type}(z)=$ merge.

- Inductive step. In this case $N=(P, G, C, c$, chan, type $)$ is such that $|P|>1$. We can split $N$ into two networks $N^{\prime}$ and $N^{\prime \prime}$ such that $N=N^{\prime} \| N^{\prime \prime}$ according to Lemma 2.13. Let $M\left(N^{\prime}\right)=\left(\operatorname{State}\left(N^{\prime}\right), \operatorname{Init}\left(N^{\prime}\right), \operatorname{Next}\left(N^{\prime}\right)\right)$ and $M\left(N^{\prime \prime}\right)=$ $\left(\right.$ State $\left.\left(N^{\prime \prime}\right), \operatorname{Init}\left(N^{\prime \prime}\right), \operatorname{Next}\left(N^{\prime \prime}\right)\right)$. Let $K S\left(N^{\prime}\right)=\left(S^{\prime}, I^{\prime}, \rightarrow{ }^{\prime}, \mathrm{AP}^{\prime}, L^{\prime}\right)$ and $\mathrm{KS}\left(N^{\prime \prime}\right)=$ $\left(S^{\prime \prime}, I^{\prime \prime}, \rightarrow \prime^{\prime \prime}, \mathrm{AP}^{\prime \prime}, L^{\prime \prime}\right)$. Let $R_{N^{\prime}}=\left\{(s, p) \mid s \in \operatorname{State}\left(N^{\prime}\right), p \in S^{\prime}, s=L^{\prime}(p)\right\}$ and $R_{N^{\prime \prime}}=\left\{(s, p) \mid s \in \operatorname{State}\left(N^{\prime \prime}\right), p \in S^{\prime \prime}, s=L^{\prime \prime}(p)\right\}$. According to the induction hypothesis, $R_{N^{\prime}}$ and $R_{N^{\prime \prime}}$ are bisimulation relations. We show that $R_{N}$ is a
bisimulation.
Fix $s \in \operatorname{State}(N), p \in S$ such that $s R_{N} p$. Note that $s=L(p)$ by the definition of $R_{N}$. We first check $s \in \operatorname{Init}(N)$ if and only if $p \in I$.
$\Rightarrow$ Assume $s \in \operatorname{Init}(N)$. According to Lemma 2.14 there are signal states $s^{\prime} \in \operatorname{State}\left(N^{\prime}\right)$ and $s^{\prime \prime} \in \operatorname{State}\left(N^{\prime \prime}\right)$, such that $s=s^{\prime} \cup s^{\prime \prime}$; let $s^{\prime}$ and $s^{\prime \prime}$ be such. From Corollary 2.22, there are $p^{\prime} \in S^{\prime}, p^{\prime \prime} \in S^{\prime \prime}$, such that:

$$
\begin{aligned}
L(p) & =L^{\prime}\left(p^{\prime}\right) \cup L^{\prime \prime}\left(p^{\prime \prime}\right), \\
s^{\prime} & =L^{\prime}\left(p^{\prime}\right), \\
s^{\prime \prime} & =L^{\prime \prime}\left(p^{\prime \prime}\right), \\
p & =\left(p^{\prime}, p^{\prime \prime}\right),
\end{aligned}
$$

and hence $s^{\prime} R_{N^{\prime}} p^{\prime}, s^{\prime \prime} R_{N^{\prime \prime}} p^{\prime \prime}$; we fix such $p^{\prime}$ and $p^{\prime \prime}$. Since $s \in \operatorname{Init}(N)$ and by Definitions 2.7 and 2.12, we have $s^{\prime} \in \operatorname{Init}\left(N^{\prime}\right)$ and $s^{\prime \prime} \in \operatorname{Init}\left(N^{\prime \prime}\right)$. Therefore, using the induction hypothesis and the fact that $s^{\prime} R_{N^{\prime}} p^{\prime}$, $s^{\prime \prime} R_{N^{\prime \prime}} p^{\prime \prime}$, we conclude $p^{\prime} \in I^{\prime}$ and $p^{\prime \prime} \in I^{\prime \prime}$. By Definition 2.17, we conclude $\left(p^{\prime}, p^{\prime \prime}\right) \in I$, and hence $p \in I$.
$\Leftarrow$ Assume $p \in I$. By Definition 2.18, $K S(N)=\operatorname{KS}\left(N^{\prime}\right) \| \operatorname{KS}\left(N^{\prime \prime}\right)$. Hence, there are $p^{\prime} \in S^{\prime}, p^{\prime \prime} \in S^{\prime \prime}$, such that $L(p)=L^{\prime}\left(p^{\prime}\right) \cup L^{\prime \prime}\left(p^{\prime \prime}\right)$; let $p^{\prime}$ and $p^{\prime \prime}$ be such. According to Lemma 2.21, there are states $s^{\prime} \in \operatorname{State}\left(N^{\prime}\right)$ and $s^{\prime \prime} \in \operatorname{State}\left(N^{\prime \prime}\right)$ such that $s^{\prime}=L^{\prime}\left(p^{\prime}\right)$ and $s^{\prime \prime}=L^{\prime \prime}\left(p^{\prime \prime}\right)$. Let $s^{\prime}$ and $s^{\prime \prime}$ be such. Note that $s=s^{\prime} \cup s^{\prime \prime}$. By the definition of $R_{N^{\prime}}, s^{\prime} R_{N^{\prime}} p^{\prime}$. By the definition of $R_{N^{\prime \prime}}$, $s^{\prime \prime} R_{N^{\prime \prime}} p^{\prime \prime}$. By Definition 2.17, $p^{\prime} \in I^{\prime}$ and $p^{\prime \prime} \in I^{\prime \prime}$. Then, from the induction hypothesis, we conclude that $s^{\prime} \in \operatorname{Init}\left(N^{\prime}\right)$ and $s^{\prime \prime} \in \operatorname{Init}\left(N^{\prime \prime}\right)$. By Definitions 2.7 and 2.12 we conclude $s \in \operatorname{Init}(N)$.

We next check the transfer conditions.

- We first show that for all $u \in \operatorname{State}(N)$, such that $(s, u) \in \operatorname{Next}(N)$, there exists $v \in S$ such that $p \rightarrow v$ and $u R_{N} v$. Fix $u \in \operatorname{State}(N)$ such that $(s, u) \in \operatorname{Next}(N)$. We have to show there exists $v \in S$ such that $p \rightarrow v$ and $u R_{N} v$.

According to Lemma 2.14, there are $s^{\prime} \in \operatorname{State}\left(N^{\prime}\right)$ and $s^{\prime \prime} \in \operatorname{State}\left(N^{\prime \prime}\right)$ such that $s=s^{\prime} \cup s^{\prime \prime} ;$ let $s^{\prime}$ and $s^{\prime \prime}$ be such. According to Corollary 2.22, $p=\left(p^{\prime}, p^{\prime \prime}\right)$ such that $L(p)=L^{\prime}\left(p^{\prime}\right) \cup L^{\prime \prime}\left(p^{\prime \prime}\right), s^{\prime}=L^{\prime}\left(p^{\prime}\right)$ and $s^{\prime \prime}=L^{\prime \prime}\left(p^{\prime \prime}\right)$. According to Lemma 2.14, there are $u^{\prime} \in \operatorname{State}\left(N^{\prime}\right)$ and $u^{\prime \prime} \in \operatorname{State}\left(N^{\prime \prime}\right)$ such that $u=u^{\prime} \cup u^{\prime \prime}$; fix such $u^{\prime}$ and $u^{\prime \prime}$. Since $s=s^{\prime} \cup s^{\prime \prime}, u=u^{\prime} \cup u^{\prime \prime}$ and $(s, u) \in \operatorname{Next}(N)$, according to Definition 2.8, $\left(s^{\prime}, u^{\prime}\right) \in \operatorname{Next}\left(N^{\prime}\right)$ and $\left(s^{\prime \prime}, u^{\prime \prime}\right) \in \operatorname{Next}\left(N^{\prime \prime}\right)$. Now, according to the induction hypothesis, since $s^{\prime}=L\left(p^{\prime}\right), s^{\prime} R_{N^{\prime}} p^{\prime}$, and since $\left(s^{\prime}, u^{\prime}\right) \in \operatorname{Next}\left(N^{\prime}\right)$, there exists $v^{\prime} \in S^{\prime}$ such that $p^{\prime} \rightarrow v^{\prime}$ and $u^{\prime} R_{N^{\prime}} v^{\prime}$. Likewise, there exists $v^{\prime \prime} \in S^{\prime \prime}$ such that $p^{\prime \prime} \rightarrow{ }^{\prime \prime} v^{\prime \prime}$ and $u^{\prime \prime} R_{N^{\prime \prime}} v^{\prime \prime}$. Let $v^{\prime}$ and $v^{\prime \prime}$ be such and let $v=\left(v^{\prime}, v^{\prime \prime}\right)$. Since $u^{\prime} R_{N^{\prime}} v^{\prime}, u^{\prime}=L\left(v^{\prime}\right)$; likewise since $u^{\prime \prime} R_{N^{\prime \prime}} v^{\prime \prime}, u^{\prime \prime}=L\left(v^{\prime \prime}\right)$. Due to $v=\left(v^{\prime}, v^{\prime \prime}\right)$, we know $L^{\prime}\left(v^{\prime}\right) \cap \mathrm{AP}^{\prime \prime}=L^{\prime \prime}\left(v^{\prime \prime}\right) \cap \mathrm{AP}^{\prime}$, hence $v \in S$. By

Definition 2.17, $p \rightarrow v$ and $L(v)=L^{\prime}\left(v^{\prime}\right) \cup L^{\prime \prime}\left(v^{\prime \prime}\right)$. From the latter it also follows that $L(v)=u$, hence $u R_{N} v$.

- Now we show that for all $v \in S$, such that $p \rightarrow v$, there exists $u \in \operatorname{State}(N)$ such that $(s, u) \in \operatorname{Next}(N)$ and $u R_{N} v$.

Since $N=N^{\prime}$ ॥ $N^{\prime \prime}$, by Definition 2.18, $\operatorname{KS}(N)=\operatorname{KS}\left(N^{\prime}\right)| | \operatorname{KS}\left(N^{\prime \prime}\right)$. Hence, there are $p^{\prime} \in S^{\prime}, p^{\prime \prime} \in S^{\prime \prime}$, such that $p=\left(p^{\prime}, p^{\prime \prime}\right)$. Let $p^{\prime}$ and $p^{\prime \prime}$ be such and observe $L(p)=L^{\prime}\left(p^{\prime}\right) \cup L^{\prime \prime}\left(p^{\prime \prime}\right)$.

According to Lemma 2.21, there are states $s^{\prime} \in \operatorname{State}\left(N^{\prime}\right)$ and $s^{\prime \prime} \in \operatorname{State}\left(N^{\prime \prime}\right)$ such that $s^{\prime}=L^{\prime}\left(p^{\prime}\right)$ and $s^{\prime \prime}=L^{\prime \prime}\left(p^{\prime \prime}\right)$. Let $s^{\prime}$ and $s^{\prime \prime}$ be such. Note that $s=s^{\prime} \cup s^{\prime \prime}$. By the definition of $R_{N^{\prime}}, s^{\prime} R_{N^{\prime}} p^{\prime}$. Similarly, by the definition of $R_{N^{\prime \prime}}, s^{\prime \prime} R_{N^{\prime \prime}} p^{\prime \prime}$.
Fix an arbitrary $v \in S$, such that $p \rightarrow v$. Since $\operatorname{KS}(N)=\operatorname{KS}\left(N^{\prime}\right) \| \operatorname{KS}\left(N^{\prime \prime}\right)$, there are $v^{\prime} \in S^{\prime}, v^{\prime \prime} \in S^{\prime \prime}$, such that $v=\left(v^{\prime}, v^{\prime \prime}\right)$; let $v^{\prime}$ and $v^{\prime \prime}$ be such. Hence $L(v)=L^{\prime}\left(v^{\prime}\right) \cup L^{\prime \prime}\left(v^{\prime \prime}\right)$. By Definition 2.17, we have $p^{\prime} \rightarrow^{\prime} v^{\prime}$ and $p^{\prime \prime} \rightarrow^{\prime \prime} v^{\prime \prime}$. By the induction hypothesis, since $s^{\prime} R_{N^{\prime}} p^{\prime}$, the existence of $v^{\prime}$ implies that there is $u^{\prime}$, such that $\left(s^{\prime}, u^{\prime}\right) \in \operatorname{Next}\left(N^{\prime}\right)$ and $u^{\prime} R_{N^{\prime}} v^{\prime}$; we fix such $u^{\prime}$. Analogously, since $s^{\prime \prime} R_{N^{\prime \prime}} p^{\prime \prime}$, there is $u^{\prime \prime}$, such that $\left(s^{\prime \prime}, u^{\prime \prime}\right) \in \operatorname{Next}\left(N^{\prime \prime}\right)$ and $u^{\prime \prime} R_{N^{\prime \prime}} v^{\prime \prime}$; fix such $u^{\prime \prime}$. Since $u^{\prime} R_{N^{\prime}} v^{\prime}$ and $u^{\prime \prime} R_{N^{\prime \prime}} v^{\prime \prime}$, we have $u^{\prime}=L^{\prime}\left(v^{\prime}\right)$ and $u^{\prime \prime}=L^{\prime \prime}\left(v^{\prime \prime}\right)$. Since $N=N^{\prime} ॥ N^{\prime \prime}$ and by Definitions 2.6 and 2.12 , there is $u \in \operatorname{State}(N)$, such that $u=u^{\prime} \cup u^{\prime \prime}$; let $u$ be such. Since $\left(s^{\prime}, u^{\prime}\right) \in \operatorname{Next}\left(N^{\prime}\right)$ and $\left(s^{\prime \prime}, u^{\prime \prime}\right) \in \operatorname{Next}\left(N^{\prime \prime}\right)$, by Definition 2.17, $(s, u) \in \operatorname{Next}(N)$. Since $u=u^{\prime} \cup u^{\prime \prime}, u=L(v)$ and hence we conclude $u R_{N} v$ 。

### 2.6 Conclusion

In the chapter, we gave a structured view at the behavior of xMAS networks in terms of irdy, trdy, and data signals by formulating signal semantics. We introduced a statebased semantics for xMAS in terms of Kripke Structures. We proved the correctness of the state-based semantics by showing bisimulation between signal structures and the corresponding KSs. The original authors of xMAS formulate properties over xMAS networks in terms of LTL, which assumes a KS semantics [GCK11]; Wouda et al. assume a KS semantics in their reachability analysis of deadlock states in xMAS [WJS15]. However, a KS semantics was not presented in the literature. The work in the chapter closes the gap and serves as a theoretical foundation for the subsequent work on the xMAS liveness verification.

## Chapter 3

## Effective System Level Liveness Verification

### 3.1 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 [CKO12] and associated techniques for the generation of invariants [CK10], property checking [CK10], and deadlock hunting [GCK11; VS11] have been proposed to address this challenge. These techniques are very efficient and have been extended to performance validation [LZ18], asynchronous circuits [BSY15], progress verification [DKB17], generalized to language families [VS12], and directly related to the Register Transfer Level [JS13; JS15].

Initially focused on the analysis of communication fabrics, Verbeek et al. [Ver+16; Ver+17] 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. Unfortunately, as we will show in this chapter, their method fails to detect some deadlocks, and is therefore unsound ${ }^{1}$.

Contributions. We present an example of an xMAS network with an FSM that has a dead input channel, and demonstrate that the approach from [Ver+17] fails to detect

[^5]this dead channel. This demonstrates 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 [Ver+17], we extend the idle and block equations from [GCK11]. 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 [Ver+16], 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 chapter. In the following sections, we recall the relevant part of the xMAS language. We introduce xMAS networks, the definition of liveness of channels and idle and block equations in Section 3.2. We recall Verbeek et al.'s extension of xMAS with automata and present a counterexample in Section 3.3. In Section 3.4 we present our xMAS finite state machines (FSMs). The idle and block equations for FSMs and their soundness are described in Section 3.5. Section 3.6 adapts the invariants from [Ver+17] to our FSMs. Our implementation is evaluated in Section 3.7. We conclude in Section 3.8.

### 3.2 Preliminaries

In this section, we introduce some notation that is used throughout the chapter. We also introduce the syntax and semantics of xMAS, specify the liveness of channels, and reiterate how liveness can be transformed into a safety problem using idle and block equations.

We write $\mathbb{N}$ to denote the set of natural numbers and $\mathbb{B}=\{$ false, true $\}$ to denote the set of booleans. We denote empty lists using $\emptyset$. Given a list xs, we denote its length by $|\mathrm{xs}|$. We write rhead $(\mathrm{xs})$ to denote the last element of list xs. That is, given a list $\mathrm{xs}=\left[x_{0}, \ldots, x_{n}\right]$, we have $\operatorname{rhead}(\mathrm{xs})=x_{n}$. Note that $\operatorname{rhead}(\emptyset)=\perp$, i.e., it is undefined. We write rtail(xs) to denote list xs without its last element. That is, given a list $\mathrm{xs}=\left[x_{0}, \ldots, x_{n}\right]$, we have $\operatorname{rtail}(\mathrm{xs})=\left[x_{0}, \ldots, x_{n-1}\right]$. Similarly to rhead, rtail is undefined for empty lists.

### 3.2.1 xMAS Syntax

xMAS [CKO12] 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 3.1.

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


Figure 3.1: Core xMAS primitives [CKO10].
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 occurs if and only $x$.irdy $\wedge x$.trdy.

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

$$
o . \mathbf{i r d y}:=i . \mathbf{i r d y} \quad i . \operatorname{trdy}:=o . \operatorname{trdy} \quad o . \text { data }:=f(i . \text { data })
$$

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 3.2: xMAS example.
Example 3.1. Consider the xMAS network in Figure 3.2. 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:

$$
x . \operatorname{irdy}:=\operatorname{oracle} \vee \operatorname{pre}(x . \operatorname{irdy} \wedge \neg x . \operatorname{trdy}) \quad x . \text { data }:=t
$$

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 [CKO12]. Let xs be the list representing the contents of the queue. Channel $x$ is the input channel of the queue, for which we have:

$$
x . \operatorname{trdy}:=|\mathrm{xs}|<1 .
$$

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:

$$
y . \text { irdy }:=|\mathrm{xs}|>0 \quad y . \text { data }:=\operatorname{rhead}(\mathrm{xs})
$$

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:

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

This definition is analogous to that of $x$.irdy.

### 3.2.2 Semantics of xMAS Networks

Recall from Chapter 2 that an xMAS network is defined as follows.
Definition 3.2. An $x$ MAS network is a structure ( $P, G, C, c$, chan, type) where:

- $P$ is the set of components;
- $G$ is the set of channels;
- $C$ is a non-empty set of data, which consists of all possible values of data signals of all channels $x \in G$;
- $c: G \rightarrow\left(2^{C} \backslash\{\emptyset\}\right)$ is the function that assigns sets of data to channels from $G$;
- chan : $P \times\{$ in, out $\} \times \mathbb{N} \rightarrow G$ is a partial function which, given a component $p \in P$, an input/output identifier and a channel number $n \in \mathbb{N}$, returns the channel connected to input (output) number $n$ of component $p$;
- type : $P \rightarrow \Gamma$ assigns a type to a component.

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 [GVS14].

We now provide a high-level explanation of the semantics of xMAS. An xMAS network has two parts: a sequential and a combinatorial one. The sequential part consists of the contents of all queues in the network, whilst the combinatorial part consists of the irdy, trdy, and data signals of all channels. In the beginning, the queues are empty. Every clock cycle comprises two updates - first, the combinatorial part gets updated, and then the sequential. For more details on the xMAS semantics, we refer the reader to Chapter 2 where we formalize the semantics of xMAS using Kripke Structures. In the Kripke Structure setting, the states of the Kripke Structure reflect the values of irdy, trdy, and data of all channels, as well as the contents of all queues of a given xMAS network. The combinatorial and sequential parts are implicitly updated while transitioning from one state of the Kripke Structure to another.

### 3.2.3 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 [BK08] for the details. The semantics of LTL quantifies over all paths.
Definition 3.3 (Path). A path is a possibly infinite sequence of states $\pi=s_{0}, s_{1}, s_{2}, \ldots$, such that for all $j>0, s_{j-1} \rightarrow s_{j}$. We use $\pi[j]$ to denote the state at position $j$ in the path, i.e., $s_{j}$, and $\pi[i .$.$] to denote the suffix of \pi$ starting at $s_{i}$. The set of paths starting in a state $s$ is denoted using Paths $(s)$, and for a $\operatorname{KS}(N)=(S, I, \rightarrow, A P, L)$ we write Paths $(\operatorname{KS}(N))$ to denote $\bigcup_{s \in I}$ Paths(s). For finite paths $\pi=s_{0}, \ldots, s_{n}$ we define $\operatorname{last}(\pi)=s_{n}$.

Further we introduce the notion of maximal path.
Definition 3.4 (Maximal path). Given a path $\pi$, we say that $\pi$ is maximal if and only if it is infinite, or it is finite, and last $(\pi)$ has no outgoing transitions.

### 3.2.4 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 3.5 (Live channel [GCK11]). Consider an xMAS network $N=(P, G, C$, $c$, chan, type). Channel $x \in G$ is live if and only if

$$
\mathrm{KS}(N) \vDash \mathrm{G}(x . \text {.irdy } \Longrightarrow \mathrm{F}(x . \text { irdy } \wedge x . \text { trdy }))
$$

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

$$
\mathrm{KS}(N) \vDash \mathrm{G}((x . \mathbf{i r d y} \wedge x . \text { data }=d) \Longrightarrow \mathrm{F}(x . \mathbf{i r d y} \wedge x . \operatorname{trdy} \wedge x . \text { data }=d))
$$

Note that Gotmanov et al. [GCK11] use property $\mathrm{G}(x$. irdy $\Longrightarrow \mathrm{F} x$.trdy) to describe liveness, 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 3.6 (Forward persistency [GCK11]). Consider an xMAS network (P, G, C, c, chan, type). Channel $x \in G$ is forward persistent if and only if for all $d \in c(x)$

$$
\mathrm{KS}(N) \vDash \mathrm{G}((x . \text { irdy } \wedge x . \text { data }=d \wedge \neg x . \text { trdy }) \Longrightarrow \mathrm{X}(x . \text {.irdy } \wedge x . \text { data }=d))
$$

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 3.7. For all $x M A S$ networks ( $P, G, C, c$, chan, type), and all channels $x \in G$, if $x$ is forward persistent, then $x$ is live if and only if for all $d \in c(x) x$ is live for value $d$.

Proof. Fix xMAS network $N$ and channel $x$, such that $x$ is forward persistent. We prove both directions separately.
$\Rightarrow$ Assume $x$ is live, which, by Definition 3.5, is equivalent to

$$
\mathrm{KS}(N) \vDash \mathrm{G}(x . \text { irdy } \Longrightarrow \mathrm{F}(x . \text { irdy } \wedge x . \text { trdy }))
$$

To prove: $x$ is live for $d$, which, by Definition 3.5 is equivalent to

$$
\mathrm{KS}(N) \vDash \mathrm{G}((x . \mathbf{i r d y} \wedge x . \text { data }=d) \Longrightarrow \mathrm{F}(x . \mathbf{i r d y} \wedge x . \operatorname{trdy} \wedge x . \text { data }=d))
$$

Now, consider an arbitrary path $\pi$ in $\operatorname{KS}(N)$. Fix arbitrary $i \geq 0$, and assume $\pi[i ..] \vDash x$.irdy $\wedge x$.data $=d$. According to the assumption, $\exists j \geq i$ such that $\pi[j ..] \vDash x$.irdy $\wedge x$.trdy. Let $j$ be the smallest such index. Since $\pi$ is persistent, for all $k$ such that $i \leq j \leq k, \pi[k] \vDash x$.irdy $\wedge x$.data $=d$, so $\pi[j ..] \vDash x$.irdy $\wedge$ $x$. data $=d \wedge x . \operatorname{trdy}$, hence $\pi \vDash \mathrm{G}((x$. irdy $\wedge x$. data $=d) \Longrightarrow \mathrm{F}(x$. irdy $\wedge$ $x \cdot \operatorname{trdy} \wedge x$.data $=d)$ ), that is, $x$ is live for $d$.
$\Leftarrow$ Assume for all $d \in c(x)$, that $x$ is live for $d$, which, by Definition 3.5 is equivalent to

$$
\mathrm{KS}(N) \vDash \mathrm{G}((x . \mathbf{i r d y} \wedge x . \text { data }=d) \Longrightarrow \mathrm{F}(x . \mathbf{i r d y} \wedge x . \operatorname{trdy} \wedge x . \text { data }=d))
$$

To prove: $x$ is live, which, by Definition 3.5 , is equivalent to

$$
\mathrm{KS}(N) \vDash \mathrm{G}(x . \text { irdy } \Longrightarrow \mathrm{F}(x . \text { irdy } \wedge x . \operatorname{trdy}))
$$

Consider an arbitrary path $\pi$ in $\operatorname{KS}(N)$. Fix arbitrary $i \geq 0$ and assume $\pi[i ..] \vDash$ $x$.irdy. According to the semantics of xMAS, there exists $d \in c(x)$ such that $\pi[i ..] \vDash x$.data $=d$. Hence, $\pi[i ..] \vDash x$.irdy $\wedge x$.data $=d$. According to the assumption, then $\pi[i ..] \vDash \mathrm{F}(x$. .irdy $\wedge x$.trdy $\wedge x$.data $=d)$, hence $\pi[i ..] \vDash$ $F(x$.irdy $\wedge x$.trdy $)$, so $\pi \vDash \mathrm{G}(x$.irdy $\Longrightarrow \mathrm{F}(x$.irdy $\wedge x$.trdy $))$, that is, $x$ is live.

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 [GCK11].

Theorem 3.8. For all $x M A S$ networks ( $P, G, C, c$, chan, type), and all channels $x \in G$, if $x$ is persistent, then

$$
K S(N) \vDash \mathrm{G}((x . \text { irdy } \wedge x . \text { data }=d) \Longrightarrow \mathrm{F}(x . \mathbf{i r d y} \wedge x . \text { data }=d \wedge x . \text { trdy }))
$$

if and only if

$$
K S(N) \vDash \operatorname{FG}(\neg x \text {.irdy } \vee x \text {.data } \neq d) \vee G F x . \text {.trdy. }
$$

Proof. Let ( $P, G, C, c$, chan, type) be an arbitrary xMAS network, and $x \in G$ an arbitrary channel, such that $x$ is persistent, i.e., for all $d \in c(x), \mathrm{KS}(N) \vDash \mathrm{G}((x$.irdy $\wedge x$.data $=$ $d \wedge \neg x$.trdy $) \Longrightarrow \mathrm{X}(x$. irdy $\wedge x$.data $=d)$ ).
We prove both directions separately.
$\Rightarrow$ Suppose $\mathrm{KS}(N) \vDash \mathrm{G}((x$. irdy $\wedge x$. data $=d) \Longrightarrow \mathrm{F}(x$.irdy $\wedge x$.data $=d \wedge$ $x$.trdy)), and let $\pi$ be an arbitrary path in $\operatorname{KS}(N)$. We need to show that $\pi \vDash$ $\mathrm{FG}(\neg x$.irdy $\vee x$.data $\neq d) \vee$ GFx.trdy, from which the result immediately follows.

Assume that $\pi \not \vDash \mathrm{FG}(\neg x$.irdy $\vee x$.data $\neq d)$, i.e., $\pi \vDash \operatorname{GF}(x$.irdy $\wedge x$.data $=d)$. We show that $\pi \vDash$ GFx.trdy, i.e., $\forall i \geq 0 . \exists j \geq i . \pi[j \ldots] \vDash x$.trdy. Let $i \geq 0$ be arbitrary. Since $\pi \vDash \operatorname{GF}(x$.irdy $\wedge x$.data $=d)$, there exists $i^{\prime} \geq i$ such that $\pi\left[i^{\prime} \ldots\right] \vDash x$.irdy $\wedge x$.data $=d$. Let $i^{\prime}$ be such. Since $\pi\left[i^{\prime} \ldots\right] \vDash(x$.irdy $\wedge x$.data $=$ $d) \Longrightarrow \mathrm{F}(x$.irdy $\wedge x$.data $=d \wedge x$.trdy $)$ according to our assumption, we have $\pi\left[i^{\prime} \ldots\right] \vDash \mathrm{F}(x$.irdy $\wedge x$.data $=d \wedge x$.trdy $)$, hence there exists $j \geq i^{\prime}$ such that $\pi[j \ldots] \vDash x$.irdy $\wedge x$.data $=d \wedge x$.trdy. Since we have chosen $i$ arbitrarily, we find $\pi \vDash$ GFx.trdy.
$\Leftarrow$ Suppose $\operatorname{KS}(N) \vDash \operatorname{FG}(\neg x$.irdy $\vee x$.data $\neq d) \vee$ GFx.trdy. Let $\pi$ be an arbitrary path in $\operatorname{KS}(N)$. We show that $\pi \vDash \mathrm{G}((x$.irdy $\wedge x$.data $=d) \Longrightarrow \mathrm{F}(x$.irdy $\wedge$ $x$. data $=d \wedge x$. trdy) $)$.

We distinguish cases based on the property that holds in $\pi$.
$-\pi \vDash \operatorname{FG}(\neg x$.irdy $\vee x$.data $=d)$. We know there exists $k \geq 0$ such that for all $j \geq k, \pi[j \ldots] \not \vDash(x$.irdy $\wedge x$.data $=d)$. Let $k$ be such.

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

This inspires the following simplification [GCK11]. 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 3.9 (Idle and blocked channels [GCK11]). Let $x$ be an arbitrary channel in
an xMAS network, and let $d \in c(x)$. We define

$$
\begin{aligned}
\operatorname{idle}(x(d)) & :=\mathrm{FG}(\neg x . \operatorname{irdy} \vee x . \text { data } \neq d) \\
\operatorname{block}(x) & :=\mathrm{FG} \neg x . \operatorname{trdy}
\end{aligned}
$$

Using these definitions, and Theorem 3.8, we have the following for forward persistent channels in an xMAS network.

Corollary 3.10. Let $N$ be an $x M A S$ 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 $K S(N) \vDash \mathbf{i d l e}(x(d)) \vee \neg \operatorname{block}(x)$,
2. channel $x$ is live iff $K S(N) \vDash\left(\bigwedge_{d \in c(x)} \mathbf{i d l e}(x(d))\right) \vee \neg \operatorname{block}(x)$.

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

1. directly from Theorem 3.8.
2. directly from part 1 of this corollary and Lemma 3.7.

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 $\neg \operatorname{idle}(x(d)) \wedge \operatorname{block}(x)$. 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 chapter, we use the following definitions.

Definition 3.11 (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

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

This definition allows us to formally define a dead channel.
Definition 3.12 (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 $\pi \in \operatorname{Paths}(N), \pi \vDash \operatorname{dead}(x(d))$.

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 3.11, block was defined only for $x$. We can refine this definition by introducing block $(x(d))$ as follows.

$$
\operatorname{block}(x(d)):=\mathrm{FG}(\neg x . \operatorname{trdy} \vee x . \text { data } \neq d)
$$

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

Lemma 3.13. Let $N$ be an $x M A S$ network, with $x$ a forward persistent channel in $N$, and $d \in c(x)$. Then $K S(N) \vDash \operatorname{live}(x(d))$ if and only if $K S(N) \vDash \operatorname{idle}(x(d)) \vee \neg \operatorname{block}(x(d))$.

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

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 3.14. Let $N$ be an $x M A S$ network with $x$ a forward persistent channel in $N$, and $d \in c(x)$. Then for all paths $\pi \in \operatorname{Paths}(N), \pi \vDash \operatorname{dead}(x(d))$ implies $\pi \vDash \bigwedge_{e \in c(x)} \operatorname{block}(x(e))$.

Proof. Fix an xMAS network $N$ and channel $x$, such that $x$ is forward persistent, and let $d \in c(x)$. Let $\pi \in \operatorname{Paths}(N)$ such that $\pi \vDash \operatorname{dead}(x(d))$. Let $e \in c(x)$ be arbitrary. Since $\pi \vDash \operatorname{dead}(x(d))$, $\pi \vDash \neg \operatorname{idle}(x(d)) \wedge \operatorname{block}(x)$, so $\pi \vDash \operatorname{block}(x)$, which immediately implies $\pi \vDash \operatorname{block}(x(e))$.

### 3.2.5 Idle and Block Equations

The main contribution of Gotmanov et al. [GCK11] 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 idle ${ }_{x}^{d}$, or never try to read from channel $x$, denoted using variable block $_{x}$. The encoding is such that, whenever there exists a path $\pi$ in the xMAS network such that $\pi \vDash \operatorname{dead}(x(d))$, then there is a satisfying assignment to the variables in the idle and block equations in which idle ${ }_{x}^{d}$ is false, and 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 ${ }^{2}$ :

$$
\begin{aligned}
\text { block }_{x} & =\text { block }_{y} \\
\text { idle }_{y}^{e} & =\bigwedge_{d \in c(x)}\left((f(d)=e) \Longrightarrow \text { idle }_{x}^{d}\right)
\end{aligned}
$$

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 3.15. We demonstrate idle and block equations in an xMAS network using the running example from Figure 3.2. Channel $x$ is the output channel of the source, which can produce tokens $k$ infinitely often, hence its output is not idle:

$$
\operatorname{idle}_{x}:=\perp .
$$

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

$$
\text { block }_{x}:=\text { full } \wedge \text { block }_{x}
$$

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:

$$
\operatorname{idle}_{x}:=\mathbf{e m p t y}_{\wedge} \wedge \text { idle }_{x} .
$$

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

$$
\text { block }_{x}:=\perp
$$

[^6]
### 3.3 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 [Ver+16; Ver+17].

### 3.3.1 xMAS Automata

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

Definition 3.16 ([Ver+17, 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 $C_{I}\left(C_{O}\right)$ the set of in (out) channels. A transition $t \in T$ is a tuple $\left(s, s^{\prime}, \varepsilon, \varphi\right)$ with $s$ and $s^{\prime}$ the begin and end states, function $\varepsilon:: C_{I} \times D \mapsto \mathbb{B}$ an event and function $\varphi:: C_{I} \times D \rightharpoonup\left(C_{O} \times D\right)$ a transformation.

In this definition, $\boldsymbol{\sim}$ indicates a partial function. Given transition $\left(s, s^{\prime}, \varepsilon, \varphi\right), \varepsilon(x, d)=$ 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 3.17 ([Ver+17, Definition 2]). Let $A$ be an xMAS automaton. Let $t=$ $\left(s, s^{\prime}, \varepsilon, \phi\right)$ be a transition of $A$.

$$
\operatorname{enabled}(t, x):=A . s \wedge x . \text { irdy } \wedge \varepsilon(x, x . \text { data }) \wedge r d y(\varphi(x, x . \text { data }))
$$

where A.s indicates $A$ is currently in state $s, r d y(\perp)=\operatorname{True}$, and $r d y(y, e)=y . \operatorname{trdy}$.
Finally, for xMAS automata, the semantics in terms of irdy and trdy signals are defined as follows.

Definition 3.18 ([Ver+17]). Let $A=\left(S, T, s_{0}, C_{I}, C_{O}\right)$ be an xMAS automaton. Assume $\mathrm{sel}_{A}$ is a fair selection function ${ }^{3}$ that chooses an enabled transition and the corresponding input channel that enables the transition. We now define:

$$
\begin{aligned}
& x . \operatorname{trdy}:=\operatorname{sel}_{A}=\left(x,,_{-}\right) \\
& y . \text { irdy }:=\operatorname{sel}_{A}=(y, t) \wedge \varphi(x, x . \text { data })=\left(y,{ }_{\prime}\right) \\
& y . \text { data }:= \begin{cases}e & \text { if sel } l_{A}=(y, t) \wedge \varphi(x, x . \text { data })=(y, e) \\
\perp & \text { otherwise }\end{cases}
\end{aligned}
$$

where $t=\left(s, s^{\prime}, \varepsilon, \varphi\right)$.

[^7]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 $A=\left(S, T, s_{0}, C_{I}, C_{O}\right)$ [Ver+17]:

$$
\begin{aligned}
\operatorname{deadA}_{A} & :=\bigvee_{s \in S} A . s \wedge \bigwedge_{t=\left(s, s^{\prime}, \varepsilon, \varphi\right) \in T} \bigwedge_{x \in C_{I}} \bigwedge_{d \in c(x)} \varepsilon(i, d) \Longrightarrow\left(\text { block }_{\varphi(x, d)} \vee \text { idle }_{x(d)}\right) \\
\text { block }_{x(d)} & :=\operatorname{dead} \mathbf{A}_{A} \vee \bigwedge_{t=\left(s, s^{\prime}, \varepsilon, \varphi\right) \in T} \neg \varepsilon(x, d) \\
\operatorname{idle}_{y(e)} & :=\operatorname{dead} \mathbf{A}_{A} \vee \bigwedge_{t=\left(s, s^{\prime}, \varepsilon, \varphi\right) \in T} \bigwedge_{x \in C_{I}} \bigwedge_{d \in c(x)} \varepsilon(x, d) \Longrightarrow \varphi(x, d) \neq(y, e)
\end{aligned}
$$

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 $\pi$ in $\operatorname{KS}(N)$ such that $\pi \vDash \operatorname{dead}\left(x(d)\right.$ ), for input channel $x$ of the automaton, then block $_{x(d)}$ is true. Likewise, if for some path $\pi$ in $\operatorname{KS}(N)$ such that $\pi \vDash \operatorname{dead}(y(e))$ for output channel $y$, then $\operatorname{idle}_{y(e)}$ 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.

### 3.3.2 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 3.19. Consider the state machine in Figure 3.3 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 $\varepsilon$ and $\varphi$ are defined (per transition) as follows:

$$
\begin{array}{ll}
\varepsilon_{1}(i, d):=i=x & \varphi_{1}(i, d):=(u, d) \\
\varepsilon_{2}(i, d):=i=y & \varphi_{2}(i, d):=(v, d) \\
\varepsilon_{3}(i, d):=i=x & \varphi_{3}(i, d):=(v, d)
\end{array}
$$

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 $s_{1}$.
Still, according to the definition by Verbeek et al., the machine is not dead as it will read $x$ infinitely often. In particular, block ${ }_{y(d)}$ is false, since $\varepsilon_{2}(i, d)=i=y$, hence


Figure 3.3: Finite state machine with deadlock not detected by Verbeek et al.'s approach
$\varepsilon_{2}(y, d)=$ 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 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.

### 3.4 Finite State Machines

Verbeek et al.'s definition of xMAS automata [Ver+16; Ver+17] 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 chapter, 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 the 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 3.20 (FSM). A finite state machine ( $F S M$ ) 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
- $T \subseteq S \times\left((I \times C) \cup\left\{\left\{_{-}\right\}\right) \times\left((O \times C) \cup\left\{\left\{_{-}\right\}\right) \times S\right.\right.$ 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 $s, s^{\prime}, s_{0}, s_{1}, \ldots$ 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 $s \xrightarrow{? x(d)!!y(e)} s^{\prime}$ to denote $\left(s,(x, d),(y, e), s^{\prime}\right) \in T$. Remark 3.21. 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 $t=s \xrightarrow{\nearrow^{/!y(e)}} s^{\prime}$ 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 $s \xrightarrow{? x_{t}!!y(e)} s^{\prime}$. Likewise, transitions $t=s \xrightarrow{? x(d) /} s^{\prime}$ 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 $s \xrightarrow{? x(d))!y_{t}} s^{\prime}$. Transitions $s \xrightarrow{\neq} s^{\prime}$ 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 2. Given an FSM $\left(S^{z}, s_{0}^{z}, I^{z}, O^{z}, T^{z}\right)$, we introduce the following notation. For state $s \in S^{z}$, we have the incoming and outgoing transitions of $s$ :

$$
\begin{aligned}
\operatorname{in}_{s}(s) & =\left\{s^{\prime} \xrightarrow{x(d) / y(e)} s^{\prime \prime} \in T^{z} \mid s=s^{\prime \prime}\right\} \\
\text { out }_{s}(s) & =\left\{s^{\prime} \xrightarrow{x(d) / y(e)} s^{\prime \prime} \in T^{z} \mid s=s^{\prime}\right\}
\end{aligned}
$$

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

$$
\begin{aligned}
\operatorname{read}(x, d) & =\left\{s \xrightarrow{i / o} s^{\prime} \in T^{z} \mid i=x(d)\right\} \\
\operatorname{write}(x, d) & =\left\{s \xrightarrow{i / o} s^{\prime} \in T^{z} \mid o=x(d)\right\}
\end{aligned}
$$

In an FSM, exactly one state is current at a time, this state is denoted cur(s). A transition $s \xrightarrow{x(d) / y(e)} s^{\prime}$ 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 3.22. Given an FSM $\left(S^{z}, s_{0^{\prime}}^{z} I^{z}, O^{z}, T^{z}\right)$, transition $s \xrightarrow{x(d) / y(e)} s^{\prime} \in T^{z}$ is enabled, denoted enabled $\left(s \xrightarrow{x(d) / y(e)} s^{\prime}\right)$ iff each of the following hold:

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

In any given state, there can be multiple enabled transitions. To resolve this nondeterminism, a scheduler sel is introduced that, at every clock cycle, selects an enabled transition. If a transition $t$ is selected, we denote this using selected $(t)$. Note $\operatorname{selected}(t) \Longrightarrow$ enabled $(t)$. Generally, sel is assumed to be fair, i.e., if state $s$ is visited infinitely often with $s \xrightarrow{x(d) / y(e)} s^{\prime}$ enabled, then $s \xrightarrow{x(d) / y(e)} s^{\prime}$ 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 of irdy, trdy and data is defined as follows.

Definition 3.23. Given an $\operatorname{FSM}\left(S^{z}, s_{0}^{z}, I^{z}, O^{z}, T^{z}\right)$, for $x \in I^{z}$ we define

- $x . \operatorname{trdy}:=\exists s \xrightarrow{x(d) / y(e)} s^{\prime} \in T^{z} . \operatorname{selected}\left(s \xrightarrow{x(d) / y(e)} s^{\prime}\right)$

For $y \in O^{z}$ we define

- y.irdy $:=\exists s \xrightarrow{x(d) / y(e)} s^{\prime} \in T^{z} . \operatorname{selected}\left(s \xrightarrow{x(d) / y(e)} s^{\prime}\right)$
- y.data $:= \begin{cases}e & \text { if } \exists s \xrightarrow{x(d) / y(e)} s^{\prime} \in T^{z} . \operatorname{selected}\left(s \xrightarrow{x(d) / y(e)} s^{\prime}\right) \\ \perp & \text { otherwise }\end{cases}$

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

We now introduce the KS semantics for the FSM primitive. For the semantics of other xMAS primitives and the xMAS networks, see Chapter 2.

Notation 3. For $x \in G$, we define the atomic propositions related to channel $x$ as

$$
\operatorname{ap}(x)=\{x . \mathbf{i r d y}, x . \operatorname{trdy}\} \cup\{x . \text { data }=d \mid d \in c(x)\} .
$$

For FSMs $z \in P$ with $z=\left(S^{z}, s_{0}^{z}, I^{z}, O^{z}, T^{z}\right)$, we define the following atomic propositions

$$
\operatorname{ap}_{\mathrm{fsm}}(z)=\left\{z . \mathbf{s e l}=t \mid t \in T^{z}\right\} \cup\left\{z . \mathrm{cur}=s \mid s \in S^{z}\right\} .
$$

For variable $v$ and $x \in \mathbb{B} \cup C$, we define the following:

$$
\operatorname{lab}_{v}(x)=\left\{\begin{array}{l}
\{v \mid x=\text { true }\} \text { if } x \in \mathbb{B}, \\
\{v=d \mid x=d\} \text { otherwise }
\end{array}\right.
$$

Given a variable name $v$, and a value $e, \operatorname{lab}_{v}(e)$ generates the singleton set of labels containing $v=e$; if $e$ is a Boolean, and $e=$ false, $\operatorname{lab}_{v}(e)$ generates $\emptyset$.

Let $N=(P, G, C, c$, chan, type $)$ be a valid xMAS network and $z \in P$ be a Finite State Machine with $z=\left(S^{z}, s_{0}^{z}, I^{z}, O^{z}, T^{z}\right)$, and $I^{z}=\left\{i_{1}, \ldots, i_{n}\right\}$, and $O^{z}=\left\{o_{1}, \ldots, o_{m}\right\}$. Then the Kripke Structure $(S, I, \rightarrow, A P, L)$ representing $z$ is defined as follows.

- The state of the Kripke Structure has to reflect irdy, trdy, and data values of inputs and outputs of $z$, its current state and the value of its arbiter. It is also necessary to take into account that sel selects an enabled transition and if a certain transition is selected it has the respective effect on the inputs and outputs of $z$. Thus,

$$
\begin{aligned}
S=\{ & \left\{\left(\mathbf{s e l}, i_{1 \text { irdy }}, i_{1 \text { trdy }}, d_{1}, \ldots, i_{n \mathbf{i r d y}}, i_{n \text { trdy }}, d_{n},\right.\right. \\
& \left.o_{1 \text { irdy }}, o_{1 \text { trdy }}, e_{1}, \ldots, o_{m \text { irdy }}, o_{m \text { trdy }}, e_{m}, \mathbf{c u r}\right) \in \\
& \left(\left(T^{z} \cup\{\perp\}\right) \times \prod_{i \in I^{z}}(\mathbb{B} \times \mathbb{B} \times c(i)) \times \prod_{o \in O^{z}}(\mathbb{B} \times \mathbb{B} \times c(o)) \times S^{z}\right) \mid \\
& \forall s \xrightarrow{? i_{k}(d)!o_{l}(e)} s^{\prime} \in T^{z} \cdot \mathbf{s e l}=s \xrightarrow{? i^{k}(d) /!o^{l}(e)} s^{\prime} \Rightarrow \\
& \left(\mathbf{c u r}=s \wedge i_{k \text { irdy }} \wedge d_{k}=d \wedge o_{l \text { trdy }} \wedge e_{l}=e\right), \\
& \bigwedge_{1 \leq k \leq n}\left(i_{k \text { trdy }} \Leftrightarrow\left(\exists s \xrightarrow{\left.? i_{k}(d)\right)!o_{l}(e)} s^{\prime} \in T^{z} \cdot \mathbf{s e l}=s \xrightarrow{\left.? i_{k}(d)\right)!o_{l}(e)} s^{\prime}\right)\right), \\
& \left.\bigwedge_{1 \leq l \leq m}\left(o_{l_{\text {lirdy }}} \Leftrightarrow\left(\exists s \xrightarrow{\left.? i_{k}(d)\right)!o_{l}(e)} s^{\prime} \in T^{z} \cdot \mathbf{s e l}=s \xrightarrow{? i_{k}(d)!o_{l}(e)} s^{\prime}\right)\right)\right\} .
\end{aligned}
$$

- For initial states we require that $\mathbf{c u r}=s_{0}^{z}$. That is,

$$
\begin{aligned}
& I=\left\{\left(\mathbf{s e l}, i_{1 \text { irdy }}, i_{1 \text { trdy }}, d_{1}, \ldots, i_{n \text { irdy }}, i_{n \text { trdy }}, d_{n}\right.\right. \\
&\left.\left.o_{1 \text { irdy }}, o_{1 \text { trdy }}, e_{1}, \ldots, o_{m \text { irdy }}, o_{m \text { trdy }}, e_{m}, s_{p_{0}}\right) \in S\right\}
\end{aligned}
$$

- Assume

$$
\begin{aligned}
& s=\left(\mathbf{s e l}, v_{1}, \ldots, v_{k}, \mathbf{c u r}\right) \\
& s^{\prime}=\left(\mathbf{s e l}^{\prime}, v^{\prime}{ }_{1}, \ldots, v^{\prime}{ }_{k}, \text { cur }^{\prime}\right) .
\end{aligned}
$$

Then, $\rightarrow$ is the smallest relation satisfying the following:

$$
\begin{aligned}
& \text { FSM1 } \xrightarrow[s \rightarrow s^{\prime}]{\mathbf{s e l}=p \xrightarrow{?(d) /!o(e)} q \quad \mathbf{c u r}=p \quad \mathbf{c u r}^{\prime}=q} \\
& \text { FSM2 } \frac{\mathbf{s e l}=\perp \quad \mathbf{c u r}=\mathbf{c u r}^{\prime}}{s \rightarrow s^{\prime}}
\end{aligned}
$$

Note that the dependency between input signals, output signals, and the value of the arbiter is expressed in the definition of $S$. The inference rules establish the dependency between the value of the arbiter, the current state, and the successor state.

- The set of atomic propositions is:

$$
\mathrm{AP}=\left(\bigcup_{i \in I^{z}} \mathrm{ap}(i)\right) \cup\left(\bigcup_{o \in \mathrm{O}^{z}} \mathrm{ap}(o)\right) \cup \mathrm{ap}_{\mathrm{fsm}}(p)
$$

- The labeling function is defined as follows:

$$
\begin{aligned}
& L\left(\left(\mathbf{s e l}, i_{1 \text { irdy }}, i_{1 \text { trdy }}, d_{1}, \ldots, i_{n \text { irdy }}, i_{n \text { trdy }}, d_{n},\right.\right. \\
& \left.\left.o_{1 \text { irdy }}, o_{1 \text { trdy }}, e_{1}, \ldots, o_{m \text { irdy }}, o_{m \text { trdy }}, e_{m}, \mathbf{c u r}\right)\right)= \\
& \left\{\operatorname{lab}_{p . \text { sel }}(\mathbf{s e l})\right\} \cup \bigcup_{1 \leq k \leq n}\left\{\operatorname{lab}_{i_{k} \cdot \mathbf{i r d y}}\left(i_{k_{\text {irdy }}}\right), \operatorname{lab}_{i_{k} \cdot \operatorname{trdy}}\left(i_{k \text { trdy }}\right), \operatorname{lab}_{i_{k} \cdot \text { data }}\left(d_{k}\right)\right\} \cup \\
& \left\{\operatorname{lab}_{p \cdot \mathbf{c u r}}(\mathbf{c u r})\right\} \cup \bigcup_{1 \leq l \leq m}\left\{\operatorname{lab}_{o_{l} \cdot \text { irdy }}\left(o_{l_{\text {irdy }}}\right), \operatorname{lab}_{o_{l} \cdot \text { trdy }}\left(o_{l \text { trdy }}\right), \operatorname{lab}_{o_{l} \cdot \text { data }}\left(e_{l}\right)\right\} \text {. }
\end{aligned}
$$

Let $\operatorname{KS}(N)=(S, I, \rightarrow, \mathrm{AP}, L)$ be the KS for a given xMAS network $N=(P, G, C, c$, chan, type) with $P=\{z\}$ and type $(z)=$ FSM and $z=\left(S^{z}, s_{0}^{z}, I^{z}, O^{z}, T^{z}\right)$. Then, for all $s \xrightarrow{x(d) / y(e)} s^{\prime} \in T^{z}, p \in S$, we have:

- enabled $\left(s \xrightarrow{x(d) / y(e)} s^{\prime}\right)$ implies $\{z . \mathbf{c u r}=s, x . \mathbf{i r d y}, x$. data $=d, y \cdot \operatorname{trdy}\} \subseteq L(p)$, and
- $\operatorname{selected}\left(s \xrightarrow{x(d) / y(e)} s^{\prime}\right)$ implies $\left\{z . \mathbf{s e l}=s \xrightarrow{x(d) / y(e)} s^{\prime}\right\} \subseteq L(p)$.

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 3.24. Given an $x M A S$ network $N$ with an $\operatorname{FSM}\left(S^{z}, s_{0}^{z}, I^{z}, O^{z}, T^{z}\right)$, for $y \in O^{z}$, we have y.irdy $\Longrightarrow y$.trdy in all states of $K S(N)$.

Proof. Fix an arbitrary state in $\mathrm{KS}(N)$. Let $y$ be a channel in the FSM, with $y$.irdy set to true. According to Definition 3.23, there exists a transition $s \xrightarrow{x(d) / y(e)} s^{\prime}$ such that cur(s) and selected $\left(s \xrightarrow{x(d) / y(e)} s^{\prime}\right)$. Since selected $\left(s \xrightarrow{x(d) / y(e)} s^{\prime}\right) \Longrightarrow \operatorname{enabled}\left(s \xrightarrow{x(d) / y(e)} s^{\prime}\right)$, we have enabled $\left(s \xrightarrow{x(d) / y(e)} s^{\prime}\right)$. By Definition 3.22, we immediately get y.trdy.

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 3.25. Given a path $\pi$, we say that $\pi$ is fair if and only if for all FSM primitives $M=\left(S^{M}, S_{0}^{M}, I^{M}, O^{M}, T^{M}\right)$ and local transitions $t \in T^{M}$, we have $\pi \vDash$ $($ GFenabled $(t)) \Longrightarrow($ GFselected $(t))$

### 3.5 Idle and Block Equations for FSMs

To define idle and block equations for finite state machines in the spirit of [GCK11], 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 3.26 (Idle states and dead transitions). Consider FSM ( $S^{z}, s_{0}^{z}, I^{z}, O^{z}, T^{z}$ ). For $s \in S^{z}$ and $t \in T^{z}$ we define the following.

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

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 3.27. Let $M=\left(S^{z}, s_{0}^{z}, I^{z}, O^{z}, T^{z}\right)$ be an FSM that appears in an $x M A S$ network $N$, and let $s$ be a state in $K S(N)$. Let $s^{\prime} \in S^{z}$ be an arbitrary state in $M$. Then

$$
s \vDash \mathrm{G}\left(\left(\operatorname{cur}\left(s^{\prime}\right) \wedge \bigwedge_{t \in \text { out }_{s}\left(s^{\prime}\right)} \neg \text { enabled }(t)\right) \Longrightarrow \operatorname{Xcur}\left(s^{\prime}\right)\right)
$$

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

$$
\pi \not \vDash \mathrm{G}\left(\left(\operatorname{cur}\left(s^{\prime}\right) \wedge \bigwedge_{t \in \text { out }_{s}\left(s^{\prime}\right)} \neg \operatorname{enabled}(t)\right) \Longrightarrow \operatorname{Xcur}\left(s^{\prime}\right)\right)
$$

This is equivalent to

$$
\pi \vDash \mathrm{F}\left(\left(\operatorname{cur}\left(s^{\prime}\right) \wedge \bigwedge_{t \in \text { out } t_{s}\left(s^{\prime}\right)} \neg e n a b l e d(t)\right) \wedge X \neg \operatorname{cur}\left(s^{\prime}\right)\right)
$$

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

$$
\pi[j . .] \vDash\left(\operatorname{cur}\left(s^{\prime}\right) \wedge \bigwedge_{t \in o u t_{s}\left(s^{\prime}\right)} \neg \operatorname{enabled}(t)\right) \wedge X \neg \operatorname{cur}\left(s^{\prime}\right) .
$$

So, in $\pi[j]$, the FSM is in state $s^{\prime}$, none of its outgoing transitions are enabled, and in $\pi[j+1]$, the FSM is in some state $s^{\prime \prime} \in S$, with $s^{\prime} \neq s^{\prime \prime}$. However, according to the semantics of xMAS networks, the FSM either takes an enabled transition, or $s^{\prime}=s^{\prime \prime}$, 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 3.28. Let $M=\left(S^{z}, s_{0}^{z}, I^{z}, O^{z}, T^{z}\right)$ be an FSM that appears in an $x M A S$ network $N$. Let $(S, I, \rightarrow, A P, L)$ be the Kripke Structure for $N$. For all states $s^{\prime} \in S^{z}$, all states $s \in S$, and paths $\pi \in \operatorname{Paths}(s)$ if

$$
\pi \vDash \mathrm{G}\left(\bigwedge_{t \in \text { out }\left(s_{s}^{\prime}\right)} \neg \operatorname{enabled}(t)\right)
$$

then for all $s^{\prime} \xrightarrow{x(d) / y(e)} s^{\prime \prime} \in$ out $_{s}\left(s^{\prime}\right)$, it holds that

$$
\pi \vDash \mathrm{G}\left(\left(\operatorname{cur}\left(s^{\prime}\right) \wedge x . \mathbf{i r d y} \wedge x . \text { data }=d\right) \Longrightarrow \mathrm{X}(x . \text {.irdy } \wedge x . \text { data }=d)\right)
$$

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

$$
\pi \vDash \mathrm{G}\left(\bigwedge_{t \in \text { out }_{s}\left(s^{\prime}\right)} \neg \operatorname{enabled}(t)\right) .
$$

Towards a contradiction, suppose that

$$
\pi \not \vDash \mathrm{G}\left(\left(\operatorname{cur}\left(s^{\prime}\right) \wedge x . \text { irdy } \wedge x . \text { data }=d\right) \Longrightarrow \mathrm{X}(x . \mathbf{i r d y} \wedge x . \text { data }=d)\right)
$$

Hence,

$$
\pi \vDash \mathrm{F}\left(\left(\operatorname{cur}\left(s^{\prime}\right) \wedge x . \text { irdy } \wedge x . \text { data }=d\right) \wedge \mathrm{X}(\neg x . \text { irdy } \vee x . \text { data } \neq d)\right)
$$

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

$$
\pi[j . .] \vDash\left(\operatorname{cur}\left(s^{\prime}\right) \wedge x . \mathbf{i r d y} \wedge x . \text { data }=d\right) \wedge X(\neg x . \mathbf{i r d y} \vee x . \text { data } \neq d)
$$

Since channels are persistent, it must be the case that $\pi[j ..] \vDash x$.trdy. But then we have $\pi[j ..] \vDash \operatorname{cur}\left(s^{\prime}\right) \wedge x$.irdy $\wedge x$.data $=d \wedge x$.trdy. However $x$.trdy is only true if there is an outgoing transition $s^{\prime} \xrightarrow{x(d) / y(e)} s^{\prime \prime}$ in $s^{\prime}$ such that $\pi[j ..] \vDash \operatorname{enabled}\left(s^{\prime} \xrightarrow{x(d) / y(e)} s^{\prime \prime}\right)$ according to Definition 3.23, 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 3.29. Let $M=\left(S^{z}, s_{0}^{z}, I^{z}, O^{z}, T^{z}\right)$ be an FSM that appears in an $x M A S$ network $N$. Let $(S, I, \rightarrow, A P, L)$ be the Kripke Structure for $N$. For all transitions $t=s^{\prime} \xrightarrow{x(d) / y(e)} s^{\prime \prime} \in T^{z}$, all states $s \in S$, and all paths $\pi \in$ Paths(s),

$$
\pi \vDash \mathrm{FG} \neg e n a b l e d(t) \text { if and only iff } \pi \vDash \operatorname{idle}\left(s^{\prime}\right) \vee \operatorname{idle}(x(d)) \vee \operatorname{block}(y) .
$$

Proof. Fix an arbitrary transition $t=s^{\prime} \xrightarrow{x(d) / y(e)} s^{\prime \prime}$, state $s$, and path $\pi \in \operatorname{Paths}(s)$.
We prove both directions separately.
$\Rightarrow$ Suppose that $\pi \vDash \mathrm{FG}_{\neg \text { enabled }(t) \text {. So, there is an index } i \geq 0 \text { such that }}$

$$
\begin{equation*}
\pi[i . .] \vDash \mathcal{G}_{\neg \text { enabled }}(t) . \tag{3.1}
\end{equation*}
$$

Towards a contradiction, suppose $\pi \not \vDash$ idle $\left(s^{\prime}\right) \vee$ idle $(x(d)) \vee \operatorname{block}(y(e))$. Hence, $\pi \vDash \neg \mathbf{i d l e}\left(s^{\prime}\right) \wedge \neg \mathbf{i d l e}(x(d)) \wedge \neg \mathbf{b l o c k}(y(e))$. Therefore, also:

$$
\begin{align*}
& \pi \vDash \neg \operatorname{idle}\left(s^{\prime}\right)  \tag{3.2}\\
& \pi \vDash \neg \operatorname{idle}(x(d))  \tag{3.3}\\
& \pi \vDash \neg \operatorname{block}(y) \tag{3.4}
\end{align*}
$$

Since $\neg \operatorname{idle}\left(s^{\prime}\right) \equiv \operatorname{GFcur}(s)$, from Equation (3.2) and the semantics of LTL, there exists $k_{1} \geq i$ such that $\pi\left[k_{1} ..\right] \vDash \operatorname{cur}\left(s^{\prime}\right)$.
Since $\neg \operatorname{idle}(x(d)) \equiv \operatorname{GF}(x$.irdy $\wedge x$.data $=d)$, from Equation (3.3) and the semantics of LTL, there exists $k_{2} \geq k_{1}$ such that $\pi\left[k_{2} ..\right] \vDash x$.irdy $\wedge x$.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 $\pi[l ..] \vDash \operatorname{cur}\left(s^{\prime}\right)$ according to our assumption and Lemma 3.27.
Since $\neg$ block $(y) \equiv$ GFy.trdy, from Equation (3.4) and the semantics of LTL, there exists $k_{3} \geq k_{2}$ such that $\pi\left[k_{3} ..\right] \vDash y$.trdy. Fix the smallest such $k_{3}$, and observe that for all $l$ such that $k_{2} \leq l \leq k_{3}$, we have $\pi[l ..] \vDash x$.irdy $\wedge x$.data $=d$ according to our assumption and Lemma 3.28. Furthermore, we have $\pi[$ l.. $] \vDash \operatorname{cur}\left(s^{\prime}\right)$ according to our assumption and Lemma 3.27.
But then, in particular, we have that

$$
\pi\left[k_{3} . .\right] \vDash \operatorname{cur}\left(s^{\prime}\right) \wedge x . \text { irdy } \wedge x . \text { data }=d \wedge y . \operatorname{trdy}
$$

hence $\pi\left[k_{3} ..\right] \vDash \operatorname{enabled}\left(s^{\prime} \xrightarrow{x(d) / y(e)} s^{\prime \prime}\right)$. But since $k_{3} \geq i$, this contradicts Equation 3.1.
$\Leftarrow$ Suppose $\pi \vDash \operatorname{idle}\left(s^{\prime}\right) \vee \operatorname{idle}(x(d)) \vee \operatorname{block}(y)$. We split the three cases.
$-\pi \vDash$ idle $\left(s^{\prime}\right)$. By definition of idle, $\pi \vDash \operatorname{FG} \neg \operatorname{cur}\left(s^{\prime}\right)$. So, there exists $i \geq 0$ such that for all $j \geq i, \pi[j ..] \vDash \neg \operatorname{cur}\left(s^{\prime}\right)$. Since $\neg \operatorname{cur}\left(s^{\prime}\right)$ implies $\neg$ enabled $\left(s^{\prime} \xrightarrow{x(d) / y(e)} s^{\prime \prime}\right)$ according to Definition 3.22, we have shown for all $j \geq i, \pi[j ..] \vDash \neg$ enabled $\left(s^{\prime} \xrightarrow{x(d) / y(e)} s^{\prime \prime}\right)$, hence $\pi \vDash$ FG $\operatorname{Fenabled}\left(s^{\prime} \xrightarrow{x(d) / y(e)}\right.$ $\left.s^{\prime \prime}\right)$.
$-\pi \vDash \operatorname{idle}(x(d))$. By definition of idle, $\pi \vDash \mathrm{FG} \neg(x$. irdy $\wedge x$.data $=d)$. This
 Definition 3.22.
$-\pi \vDash \operatorname{block}(y)$. 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 3.30. Given an $x M A S$ network $N$ with an $\operatorname{FSM}\left(S^{z}, s_{0}^{z}, I^{z}, O^{z}, T^{z}\right)$, for all states $s$ and for channels $y \in O^{z}$ and $e \in c(y)$, we have for all paths $\pi \in \operatorname{Paths}(s), \pi \not \vDash \operatorname{dead}(y(e))$.

Proof. Let $s, y \in O^{z}, e \in c(y)$ and $\pi \in$ Paths(s) be arbitrary.
Towards a contradiction, suppose $\pi \vDash \operatorname{dead}(y(e))$. Hence, $\pi \vDash \neg \operatorname{idle}(y(e)) \wedge \operatorname{block}(y(e))$. Since $\pi \vDash \neg \operatorname{idle}(y(e))$, by definition of idle $\pi \vDash \neg \mathrm{FG}(\neg y$.irdy $\vee y$.data $\neq e)$ hence $\pi \vDash \operatorname{GF}(y$. irdy $\wedge y$. data $=e)$.
Since $\pi \vDash \operatorname{block}(y(e)), \pi \vDash \operatorname{FG}(\neg y . \operatorname{trdy} \vee y$.data $\neq e)$. Hence, there exists $i \geq 0$ such that for all $j \geq i, \pi[j ..] \vDash \neg y$.trdy $\vee y$.data $\neq e$. Let $i$ be such. Since $\pi \vDash \operatorname{GF}(y$.irdy $\wedge$ $y$.data $=e$ ), for some $k \geq i$, we have $\pi[k ..] \vDash y . \operatorname{irdy} \wedge y$. data $=e$. According to Lemma 3.24, y.irdy $\Longrightarrow y . \operatorname{trdy}$, hence $\pi[k ..] \vDash y . \operatorname{trdy} \wedge y$.data $=e$, which is a contradiction. So, $\pi \not \models \operatorname{dead}(y(e))$.

### 3.5.1 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 3.31 (Idle and block equations for FSMs). Consider a Finite State Machine $M=\left(S^{z}, s_{0}^{z}, I^{z}, O^{z}, T^{z}\right)$. For $s^{\prime} \in S^{z}, x \in I^{z}, y \in O^{z}, d \in c(x), e \in c(y)$, and $s^{\prime} \xrightarrow{x(d) / y(e)} s^{\prime \prime} \in$ $T^{z}$ we define the following boolean equations.

$$
\begin{array}{rlrl}
\mathbf{b l o c k}_{x}^{d} & =\bigwedge_{t \in \operatorname{read}(x, d)} \operatorname{dead}_{t} & \text { block }_{x} & =\bigwedge_{d \in c(x)} \mathbf{b l o c k}_{x}^{d} \\
\mathbf{i d l e}_{y}^{e} & =\bigwedge_{t \in w r i t e(y, e)} \operatorname{dead}_{t} & \mathbf{i d l e}_{y}=\bigwedge_{e \in c(y)} \text { idle }_{y}^{e} \\
\operatorname{dead}_{s^{\prime}} \xrightarrow{x(d) / y(e)} s_{s^{\prime \prime}} & =\text { idle }_{s} \vee \text { idle }_{x}^{d} \vee \mathbf{b l o c k}_{y} & & \\
\mathbf{i d l e}_{s} & =\neg \operatorname{cur}_{s^{\prime}} \wedge \bigwedge_{t \in i n_{s}\left(s^{\prime}\right)} \operatorname{dead}_{t} & &
\end{array}
$$

The formula 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 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 3.14 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.

### 3.5.2 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.

## 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. We focus our reasoning on the finite state machines and 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 [GCK11].

Definition 3.32 (Consistency). We say that an assignment $\sigma$ 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 $\sigma$ 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 $\sigma$ to every variable in $\Phi$, 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 from $R$ 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 3.33. Consider the encoding $\operatorname{SAT}(N)$ of an xMAS network $N$, let $\pi \in$ Paths $(\operatorname{KS}(N))$ be a path, and $\sigma$ an assignment to the variables in $\operatorname{SAT}(N)$. Let $V$ and $R$ be subsets of the variables in $\operatorname{SAT}(N)$. We say that $\sigma$ is $\pi$-consistent with respect to $V$ and $R$ if $\sigma$ is consistent for all variables $v \in V$, and

- if $\operatorname{block}_{x}^{d} \in R$ then $\pi \vDash \operatorname{block}(x(d))$ iff $\sigma\left(\right.$ block $\left._{x}^{d}\right)=$ true, and
- if idle ${ }_{x}^{d} \in R$ then $\pi \vDash$ idle $(x(d))$ iff $\sigma\left(\right.$ idle $\left._{x}^{d}\right)=$ true.

Given an FSM $M=\left(S^{z}, s_{0}^{z}, I^{z}, O^{z}, T^{z}\right)$, we define

$$
\begin{aligned}
R(M) & =\left\{\text { idle }_{x}^{d} \mid x \in I^{z}, d \in c(x)\right\} \cup\left\{\text { block }_{x} \mid x \in O^{z}\right\} \\
V(M) & =\left\{\text { block }_{x}^{d} \mid x \in I^{z}, d \in c(x)\right\} \cup\left\{\text { block }_{x} \mid x \in I^{z}\right\} \\
& \cup\left\{\operatorname{idle}_{x}^{d} \mid x \in O^{z}, d \in c(x)\right\} \cup\left\{\text { idle }_{x} \mid x \in O^{z}\right\} \\
& \cup\left\{\operatorname{dead}_{t} \mid t \in T^{z}\right\}
\end{aligned}
$$

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

## Building a satisfying assignment

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

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 3.30.

Theorem 3.34. Let $M=\left(S^{z}, s_{0}^{z}, I^{z}, O^{z}, T^{z}\right)$ be an $F S M$ that appears in an $x M A S$ network $N$. Let $x \in I^{z}$ be channel with $d \in c(x)$. If there exists a fair maximal path $\pi \in \operatorname{Paths}(s)$ such that $\pi \vDash \operatorname{dead}(x(d))$, and an assignment $\sigma$ that is $\pi$-consistent with respect to $V(\operatorname{SAT}(N)) \backslash V(M)$ and $R(M)$, then there exists a satisfying assignment to the formula $\operatorname{SAT}(N) \wedge \neg \operatorname{idle}_{x}^{d} \wedge$ 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 3.35. Let $M=\left(S^{z}, s_{0}^{z}, I^{z}, O^{z}, T^{z}\right)$ be an FSM in an xMAS network $N, \pi$ a path in $\operatorname{KS}(N)$, and let $\sigma$ be a variable assignment that is $\pi$-consistent with respect to $V(\operatorname{SAT}(N)) \backslash V(M)$ and $R(M)$. Given a state $s^{\prime} \in S^{z}$, we define the variable assignment $\sigma_{s}$ to $V(\operatorname{SAT}(N))$ 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 $s^{\prime \prime} \in S^{z}$, transitions $t \in T^{z}$, channels $x \in I^{z}$,
$y \in O^{z}$, and $d \in c(x), e \in c(y):$

$$
\begin{aligned}
\sigma_{s}\left(\operatorname{cur}_{s^{\prime \prime}}\right) & :=s^{\prime}=s^{\prime \prime} & \sigma_{s}\left(\mathbf{i d l e}_{s^{\prime \prime}}\right) & :=s^{\prime} \neq s^{\prime \prime} \quad \sigma_{s}\left(\mathbf{d e a d}_{t}\right):=\text { true } \\
\sigma_{s}\left(\mathbf{b l o c k}_{x}^{d}\right) & :=\text { true } & \sigma_{s}\left(\mathbf{b l o c k}_{x}\right) & :=\text { true } \\
\sigma_{s}\left(\mathbf{i d l e}_{y}^{d}\right) & :=\text { true } & \sigma_{s}\left(\mathbf{i d l e}_{y}\right) & :=\text { true }
\end{aligned}
$$

When it is clear from the context that we evaluate a SAT formula in the context of $\sigma_{s}$ we omit it, and write, e.g., cur $_{s^{\prime}}$ instead of $\sigma_{s}\left(\mathbf{c u r}_{s^{\prime}}\right)$.

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 3.36. Let $M=\left(S^{z}, s_{0}^{z}, I^{z}, O^{z}, T^{z}\right)$ be an FSM that appears in an $x M A S$ network $N$. For all finite fair maximal paths $\pi \in \operatorname{Paths}(K S(N))$, such that last $(\pi) \vDash \operatorname{cur}\left(s^{\prime}\right)$ for some $s^{\prime} \in S^{z}$, and all assignments $\sigma$ that are $\pi$-consistent with respect to $V(\operatorname{SAT}(N)) \backslash V(M)$ and $R(M)$, the assignment $\sigma_{s}$ is a satisfying assignment.

Proof. Assume there exists a fair maximal path $\pi \in \operatorname{Paths}(N)$ such that $\pi$ is finite and $\operatorname{last}(\pi) \vDash \operatorname{cur}\left(s^{\prime}\right)$ for some $s^{\prime} \in S^{z}$.

We check consistency of the assignment $\sigma_{s}$. Note that $\sigma_{s}$ 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 true. We therefore focus on the other two cases:
idle $_{q}$. We first show for arbitrary $q \in S, \operatorname{idle}_{q}=\neg \operatorname{cur}_{q} \wedge \wedge_{t \in i i_{s}(q)} \operatorname{dead}_{t}$. If $q=s^{\prime}$, then idle $_{q}=$ false, and $\operatorname{cur}_{q}=$ true, therefore, false $=$ idle $_{q}=\neg \mathbf{c u r}_{q} \wedge$ $\bigwedge_{t \in i i_{s}(q)} \operatorname{dead}_{t}=$ false $\wedge \bigwedge_{t \in i n_{s}(q)} \operatorname{dead}_{t}=$ false is consistent. If $q \neq s^{\prime}$, then true $=\operatorname{idle}_{q}=\neg \operatorname{cur}_{q} \wedge \wedge_{t \in i n_{s}(q)}$ dead $_{t}=\neg$ false $\wedge \wedge_{t \in i n_{s}(q)}$ true $=$ true is consistent.
$\operatorname{dead}_{t}$. For arbitrary $q \xrightarrow{x(d) / y(e)} q^{\prime} \in T^{z}$ we show $\operatorname{dead}_{q} \xrightarrow{x(d) / y(e)} q^{\prime}=$ idle $_{q} \vee$ idle $_{x}^{d} \vee$ block $_{y}$ is consistent. If $q \neq s^{\prime}$, then true $=\operatorname{dead}_{q} \xrightarrow{x(d) / y(e)} q^{\prime}=\operatorname{idle}_{q} \vee$ idle $_{x}^{d} \vee$ block $_{y}=$ true $\vee$ idle $_{x}^{d} \vee$ block $_{y}=\operatorname{true}$. If $q=s^{\prime}$, then since $\pi$ is maximal, we have last $\left.(\pi) \vDash \mathrm{G}_{\neg \operatorname{enabled}(~}^{\text {}} \xrightarrow{x(d) / y(e)} q^{\prime}\right)$. Furthermore, since last $(\pi) \vDash \operatorname{cur}(s)$, we have last $(\pi) \vDash \neg \operatorname{idle}(s)$. Thus, from Lemma 3.29 it follows that $\operatorname{idle}(x(d))$ or block $(y)$; since both are in $R(M)$, and $\sigma$ is $\pi$-consistent, idle $_{x}^{d} \vee$ block $_{y}=$ true, and $\sigma_{s}$ is consistent.

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

Lemma 3.37. Let $M=\left(S^{z}, s_{0}^{z}, I^{z}, O^{z}, T^{z}\right)$ be an FSM that appears in an $x M A S$ network $N$. For all fair maximal paths $\pi \in \operatorname{Paths}(K S(N))$, and all $s^{\prime} \in S^{z}$ such that

$$
\pi \vDash \mathrm{FG}\left(\operatorname{cur}\left(s^{\prime}\right) \wedge \bigwedge_{t \in \text { out }_{s}\left(s^{\prime}\right)} \neg \operatorname{enabled}(t)\right),
$$

and all assignments $\sigma$ that are $\pi$-consistent with respect to $V(\mathrm{SAT}(N)) \backslash V(M)$ and $R(M)$, the assignment $\sigma_{s}$ is a satisfying assignment.

Proof. Assume there exists a fair maximal path $\pi \in \operatorname{Paths}(N)$ such that $\pi$ is infinite, and let $s^{\prime} \in S^{z}$ be such that $\pi \vDash \mathrm{FG}\left(\operatorname{cur}\left(s^{\prime}\right) \wedge \bigwedge_{t \in \text { out }_{s}\left(s^{\prime}\right)} \neg \operatorname{enabled}(t)\right)$.
We check consistency of the assignment $\sigma_{s}$. Note that $\sigma_{s}$ 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 true. We therefore focus on the other two cases.
idle $_{q}$. We first show for arbitrary $q \in S, \operatorname{idle}_{q}=\neg \operatorname{cur}_{q} \wedge \bigwedge_{t \in i i_{s}(q)} \operatorname{dead}_{t}$. If $q=s^{\prime}$, then idle $_{q}=$ false, and $\operatorname{cur}_{q}=$ true, therefore, false $=$ idle $_{q}=\neg \operatorname{cur}_{q} \wedge$ $\wedge_{t \in i n_{s}(q)} \operatorname{dead}_{t}=$ false $\wedge \wedge_{t \in i n_{s}(q)} \operatorname{dead}_{t}=$ false is consistent. If $q \neq s^{\prime}$, then true $=$ idle $_{q}=\neg \operatorname{cur}_{q} \wedge \bigwedge_{t \in i n_{s}(q)}$ dead $_{t}=\neg$ false $\wedge \bigwedge_{t \in i n_{s}(q)}$ true $=$ true is consistent.
dead $_{t}$. For arbitrary $q \xrightarrow{x(d) / y(e)} q^{\prime} \in T^{z}$ we show $\operatorname{dead}_{q} \xrightarrow{x(d) / y(e)} q^{\prime}=$ idle $_{q} \vee$ idle $_{x}^{d} \vee$ block $_{y}$ is consistent. If $q \neq s^{\prime}$, then true $=\operatorname{dead}_{q} \xrightarrow{x(d) / y(e)} q^{\prime}=\mathbf{i d l e}_{q} \vee$ idle $_{x}^{d} \vee$ block $_{y}=\operatorname{true} \vee$ idle $_{x}^{d} \vee$ block $_{y}=\operatorname{true}$. If $q=s^{\prime}$, then we have dead $q \xrightarrow{x(d) / y(e)} q^{\prime}=$ true, and idle $_{q}=\perp$, and we need to show that idle ${ }_{x}^{d} \vee$ block $_{y}=$ true. Since $q=s^{\prime}, \pi \vDash \operatorname{FG}\left(\operatorname{cur}(q) \wedge \bigwedge_{t \in o u t_{s}(q)} \neg\right.$ enabled $\left.(t)\right)$. Therefore, $\pi \vDash \operatorname{FGcur}(q)$ as well as $\pi \vDash$ FG $\wedge_{t \in o u t_{s}(q)} \neg$ enabled $\left.(t)\right)$. From this, it follows that $\pi \vDash \neg \operatorname{idle}(q)$, and $\pi \vDash$ FG $\operatorname{lenabled}\left(~ q \xrightarrow{x(d) / y(e)} q^{\prime}\right)$. Thus, according to Lemma 3.29, $\pi \vDash$ idle $(q) \vee$ idle $(x(d)) \vee \operatorname{block}(y)$. Since $\pi \vDash \neg \operatorname{idle}(q)$, we have $\pi \vDash \operatorname{idle}(x(d)) \vee \operatorname{block}(y(e))$ according to Lemma 3.29. Since idle $(x(d))$ and block $(y(e))$ are in $R(M)$, and $\sigma$ is $\pi$-consistent, we get that $\mathbf{i d l e}_{x}^{d} \vee$ block $_{y}^{e}=$ true. Thus it follows that $\sigma_{s}$ is consistent.

The assignment $\sigma_{s}$ 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 $\pi$.
Definition 3.38. Let $M=\left(S^{z}, s_{0}^{z}, I^{z}, O^{z}, T^{z}\right)$ be an FSM in an xMAS network $N$, let $\pi \in \operatorname{Paths}(\operatorname{KS}(N))$ be an infinite path, and let $\sigma$ be a variable assignment that is $\pi$ consistent with respect to $V(\operatorname{SAT}(N)) \backslash V(M)$ and $R(M)$. We construct assignment $\sigma_{\pi}$ such that for all $v \in V(\operatorname{SAT}(N)) \backslash V(M), \sigma_{\pi}(v)=\sigma(v)$, and for all $v \in V(M), \sigma_{\pi}(v)$ is as follows.

Since $\pi$ is infinite and the network and all its data is finite, $\pi$ is a lasso that consists of a prefix $\pi[0] \ldots \pi[i]$ and a cycle $\pi[i] \pi[i+1] \ldots \pi[i+n]$ 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 $\pi[i] \vDash \operatorname{cur}\left(s^{\prime}\right)$, i.e., $s^{\prime}$ is the local state of the FSM at the beginning of the loop.
For states $s^{\prime \prime} \in S^{z}$, transitions $t \in T^{z}$, channels $x \in I^{z}, y \in O^{z}$, and $d \in c(x), e \in c(y)$ :

$$
\begin{aligned}
\sigma_{\pi}\left(\mathbf{c u r}_{s^{\prime \prime}}\right) & :=s^{\prime}=s^{\prime \prime} \\
\sigma_{\pi}\left(\mathbf{i d l e}_{s^{\prime \prime}}\right) & :=\forall 0 \leq k \leq n \cdot \pi[i+k] \vDash \neg \operatorname{cur}\left(s^{\prime \prime}\right) \\
\sigma_{\pi}\left(\operatorname{dead}_{t}\right) & :=\forall 0 \leq k \leq n \cdot \pi[i+k] \vDash \neg \operatorname{enabled}(t) \\
\sigma_{\pi}\left(\mathbf{b l o c k}_{x}^{d}\right) & :=\forall t \in \operatorname{read}(x, d) \cdot \forall 0 \leq k \leq n \cdot \pi[i+k] \vDash \neg \operatorname{enabled}(t) \\
\sigma_{\pi}\left(\mathbf{b l o c k}_{x}\right) & :=\forall d \in c(x) \cdot \forall t \in \operatorname{read}(x, d) . \forall 0 \leq k \leq n \cdot \pi[i+k] \vDash \neg \operatorname{enabled}(t) \\
\sigma_{\pi}\left(\mathbf{i d l e}_{y}^{d}\right) & :=\forall t \in \operatorname{write}(y, e) \cdot \forall 0 \leq k \leq n \cdot \pi[i+k] \vDash \neg \operatorname{enabled}(t) \\
\sigma_{\pi}\left(\mathbf{i d l e}_{y}\right) & :=\forall e \in c(y) \cdot \forall t \in \operatorname{write}(y, e) \cdot \forall 0 \leq k \leq n \cdot \pi[i+k] \vDash \neg \operatorname{enabled}(t)
\end{aligned}
$$

When it is clear from the context that we evaluate a SAT formula in the context of $\sigma_{\pi}$ we omit it, and write, e.g., cur $_{s^{\prime \prime}}$ instead of $\sigma_{\pi}\left(\right.$ cur $\left._{s^{\prime \prime}}\right)$.
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 3.39. Let $M=\left(S^{z}, s_{0}^{z}, I^{z}, O^{z}, T^{z}\right)$ be an $F S M$ that appears in an $x M A S$ network $N$. For all infinite fair maximal paths $\pi \in \operatorname{Paths}(N)$, and all assignments $\sigma$, if for all $s^{\prime} \in S^{z}, \pi \vDash \operatorname{GF}\left(\operatorname{cur}(s) \Longrightarrow \bigvee_{t \in o u t_{s}\left(s^{\prime}\right)}\right.$ enabled $\left.(t)\right)$, and $\sigma$ is $\pi$-consistent with respect to $V(\operatorname{SAT}(N)) \backslash V(M)$ and $R(M)$, then the assignment $\sigma_{\pi}$ is consistent.

Proof. Fix an arbitrary fair maximal infinite path $\pi \in \operatorname{Paths}(\operatorname{KS}(N))$ and $\sigma$ such that for all $s^{\prime} \in S^{z}, \pi \vDash \operatorname{GF}\left(\operatorname{cur}\left(s^{\prime}\right) \Longrightarrow \bigvee_{t \in o u t_{s}\left(s^{\prime}\right)}\right.$ enabled $\left.(t)\right)$, and $\sigma$ is $\pi$-consistent with respect to $V(\operatorname{SAT}(N)) \backslash V(M)$ and $R(M)$.
We check that the assignment $\sigma_{\pi}$ is consistent. For this, let $i$ be such that it denotes the start of the cycle on $\pi$, and suppose $s^{\prime} \in S^{z}$ is such that $\pi[i] \vDash \operatorname{cur}\left(s^{\prime}\right)$.
idle $_{q}$. We first show for arbitrary $q \in S^{z}$ that idle $_{q}=\neg$ cur $_{s^{\prime}} \wedge \bigwedge_{t \in i n_{s}\left(s^{\prime}\right)}$ dead $_{t}$. We distinguish two cases. If $\mathbf{i d l e}_{q}=\operatorname{true}$, then $\pi[i ..] \vDash \operatorname{G} \neg \operatorname{cur}(q)$. Since $\pi$ is fair, $\pi \vDash \mathrm{FG} \wedge_{t \in i_{s}(q)} \neg$ enabled $(t)$, otherwise, one of the transitions would eventually be selected, and $q$ would be reached. Then, for all transitions $t \in \operatorname{in}_{s}(q), \pi \vDash$ $\mathrm{FG} \neg$ enabled $(t)$, hence dead $_{t}=$ true for all such transitions. Since $\pi[i ..] \vDash \operatorname{cur}\left(s^{\prime}\right)$, and $\pi[i ..] \vDash \operatorname{G}{ }_{\neg \operatorname{cur}}(q), s^{\prime} \neq q$, thus $\operatorname{cur}_{q}=$ false. Therefore, true $=\mathbf{i d l e}{ }_{q}=$ $\neg \operatorname{cur}_{q} \wedge \wedge_{t \in \eta_{s}(q)}$ dead $_{t}=$ true.
If idle $\mathbf{e}_{q}=$ false, then if $\pi[i] \vDash \operatorname{cur}(q), \operatorname{cur}_{q}=$ true, and the result follows immediately from false $=\mathbf{i d l e}_{q}=\neg \operatorname{cur}_{q} \wedge \wedge_{t \in i i_{s}(q)}$ dead $_{t}=$ false $\wedge \wedge_{t \in i n_{s}(q)}$ dead $_{t}=$ false. Suppose $\pi[i] \not \vDash \operatorname{cur}(q)$. Then, for some $k$ such that $0 \leq k \leq n, \pi[i+k] \vDash$ $\operatorname{cur}(q)$. Let $k$ be the smallest such that $\pi[i+k] \vDash \operatorname{cur}(q)$. Observe that $k>0$.

Then, $\pi[i+k-1] \vDash \operatorname{enabled}(t)$ for some $t \in \operatorname{in}_{s}(q)$. Hence, $\operatorname{dead}_{t}=$ false for this $t$. Then, $\bigwedge_{t \in i n_{s}(q)}$ dead $_{t}=$ false, therefore idle ${ }_{q}=\neg \operatorname{cur}_{q} \wedge \bigwedge_{t \in i n_{s}(q)}$ dead $_{t}=$ $\neg \operatorname{cur}_{q} \wedge$ false $=$ false is consistent.
$\operatorname{dead}_{t}$. For arbitrary $q \xrightarrow{x(d) / y(e)} q^{\prime} \in T^{z}$, we show $\operatorname{dead}_{q} \xrightarrow{x(d) / y(e)} q^{\prime}=\operatorname{idle}_{q} \vee$ idle $_{x}^{d} \vee$ block $_{y}$ is consistent.

If $\operatorname{dead}_{q} \xrightarrow{x(d) / y(e)} q^{\prime}=\operatorname{true}$, then $\pi[i+k] \models \neg \operatorname{enabled}\left(q \xrightarrow{x(d) / y(e)} q^{\prime}\right)$ for all $0 \leq k \leq n$, hence $\pi \vDash \mathrm{FG}_{\neg \text { enabled }(~} q \xrightarrow{x(d) / y(e)} q^{\prime}$ ). If there exists $0 \leq k \leq n$ such that $\pi[i+k] \vDash$ $\operatorname{cur}(q)$, then it must be the case that $\pi \vDash \operatorname{idle}(x(d)) \vee \operatorname{block}(y)$, otherwise $\pi \vDash$ GFenabled $\left(q \xrightarrow{x(d) / y(e)} q^{\prime}\right)$, which is a contradiction. Since idle ${ }_{x}^{d}$, block $_{y} \notin V(M)$, idle $_{x}^{d}=$ true or block $_{y}=$ true since $\sigma$ is $\pi$-consistent w.r.t $V(\mathrm{SAT}(N)) \backslash V(M)$ and $R(M)$, and true $=\operatorname{dead}_{q} \xrightarrow{x(d) / y(e)} q^{\prime}=\mathbf{i d l e}_{q} \vee$ idle $_{x}^{d} \vee$ block $_{y}=$ true is consistent.

If dead ${ }_{q} \xrightarrow{x(d) / y(e)} q^{\prime}=$ false, then $\pi[i+k] \vDash \operatorname{enabled}\left(q \xrightarrow{x(d) / y(e)} q^{\prime}\right)$ for some $k$. Then $\pi[i+k] \vDash \operatorname{cur}(q)$, thus idle $q=$ false, and also, from the definition of enabled, it follows immediately that $\pi \vDash \neg \mathbf{i d l e}(x(d)) \wedge \neg$ block $(y)$. Since idle $\boldsymbol{e}_{x}^{d}$, block $_{y} \notin$ $V(M)$, idle $_{x}^{d}=$ false and block $_{y}=$ false since $\sigma$ is $\pi$-consistent w.r.t $V(\operatorname{SAT}(N)) \backslash$ $V(M)$ and $R(M)$. Therefore, false $=\operatorname{dead}_{q} \xrightarrow{x(d) / y(e)} q^{\prime}=$ idle $_{q} \vee$ idle $_{x}^{d} \vee$ block $_{y}=$ false is consistent.
block $_{x}^{d}$. We show block $_{x}^{d}=\bigwedge_{t \in \operatorname{read}(x, d)}$ dead $_{t}$ for arbitrary channel $x$ and $d \in c(x)$. Observe that $\operatorname{block}_{x}^{d}=$ true if and only if $\forall t \in \operatorname{read}(x, d), 0 \leq k \leq n, \pi[i+k] \vDash$ $\neg$ enabled $(t)$. By definition of dead, dead ${ }_{t}=$ true for all such transitions $t$ as well, and the assignment is consistent by definition.
block $_{x}$. Consistency of block $_{x}=\bigwedge_{d \in c(x)}$ block $_{x}^{d}$ follows immediately from the definitions.
$\mathbf{i d l e}_{y}^{e}$. Showing for arbitrary channel $y$ and $e \in c(y)$ that idle $_{y}^{e}=\Lambda_{t \in w r i t e(~}^{\text {}, e)}$ $\operatorname{dead}_{t}$ and $\mathbf{i d l e}_{y}=\bigwedge_{e \in c(y)} \mathbf{i d l e}_{y}^{e}$ are consistent is analogous to the case of block $_{x}^{d}$.
idle $_{y}$. Consistency of $\mathbf{i d l e}_{y}=\bigwedge_{e \in c(y)} \mathbf{i d l e}_{y}^{e}$ follows immediately from the definitions. Hence $\sigma_{\pi}$ is a consistent satisfying assignment to $\operatorname{SAT}(N)$.

The following lemma shows that if an xMAS network $N$ satisfies idle(s) for some FSM state $s$, then there exists a satisfying assignment to $\operatorname{SAT}(N)$ in which idle $(s)=$ true. In the proof of the lemma, we demonstrate the construction of a satisfying assignment for formulas that involve our idle and block equations for FSMs.

Lemma 3.40. Let $M=\left(S^{z}, s_{0}^{z}, I^{z}, O^{z}, T^{z}\right)$ be an FSM that appears in an $x M A S$ network $N$, and let $s^{\prime} \in S^{z}$ be a local state in $M$. If there exists a fair maximal path $\pi \in \operatorname{Paths}(K S(N))$ such that $\pi \vDash \mathbf{i d l e}(s)$, and an assignment $\sigma$ that is $\pi$-consistent with respect to $V(\operatorname{SAT}(N)) \backslash V(M)$ and $R(M)$, then there exists a satisfying assignment to the formula $\operatorname{SAT}(N) \wedge$ idle $_{s}$.

Proof. Fix an arbitrary local state $s^{\prime} \in S^{z}$, and assume there exists a fair maximal path $\pi \in \operatorname{Paths}(\mathrm{KS}(N))$ such that $\pi \vDash \operatorname{idle}\left(s^{\prime}\right)$. Let $\sigma$ be an assignment as specified. According to the definition of idle, $\pi \vDash \mathrm{FG} \neg \mathrm{cur}\left(s^{\prime}\right)$. We distinguish two cases: either $\pi$ is finite, or $\pi$ is infinite.
$\pi$ is finite. Let $p \in S^{z}$ be such that last $(\pi) \vDash \operatorname{cur}(p)$. According to Lemma 3.36, $\sigma_{p}$ is a satisfying assignment for $\operatorname{SAT}(N)$. Since $\pi \vDash \operatorname{idle}\left(s^{\prime}\right)$, last $(\pi) \not \vDash \operatorname{cur}\left(s^{\prime}\right)$, hence $\sigma_{p}\left(\right.$ idle $\left._{s^{\prime}}\right)=$ true, and the assignment also satisfies $\operatorname{SAT}(N) \wedge$ idle $_{s^{\prime}}$.
$\pi$ is infinite. We distinguish two cases.
First, assume $\pi \vDash \operatorname{FG}\left(\operatorname{cur}\left(s^{\prime \prime}\right) \wedge \wedge_{t \in o u t_{s}\left(s^{\prime \prime}\right)} \neg\right.$ enabled $\left.(t)\right)$ for some $s^{\prime \prime} \in S^{z}$. Let $p$ be such a state. According to Lemma 3.37, $\sigma_{p}$ is a consistent assignment for SAT( $N$ ).

Since $\pi \vDash \operatorname{idle}\left(s^{\prime}\right)$, for some $i \geq 0$, and all $j \geq i, \pi[j] \vDash \neg \operatorname{cur}\left(s^{\prime}\right)$, which means it must be the case that $p \neq s^{\prime}$, hence $\sigma_{p}\left(\mathbf{i d l e}_{s^{\prime}}\right)=$ true hence $\sigma_{p}$ is a satisfying assignment for $\operatorname{SAT}(N) \wedge$ idle $_{s^{\prime}}$.
Otherwise, $\pi \not \vDash \operatorname{FG}\left(\operatorname{cur}\left(s^{\prime \prime}\right) \wedge \wedge_{t \in o u t_{s}\left(s^{\prime \prime}\right)} \neg \operatorname{enabled}(t)\right)$ for all $s^{\prime \prime} \in S^{z}$. Hence, for all $s^{\prime \prime} \in S^{z}, \pi \vDash \operatorname{GF}\left(\operatorname{cur}\left(s^{\prime \prime}\right) \xlongequal{\Longrightarrow} V_{t \in o u t_{s}\left(s^{\prime \prime}\right)}\right.$ enabled $\left.(t)\right)$. According to Lemma 3.39, $\sigma_{\pi}$ is consistent with $\operatorname{SAT}(N)$.
Let $i$ be the index that signals the start of the loop of the lasso. Observe that, since $\pi \vDash \operatorname{idle}\left(s^{\prime}\right), \pi \vDash \operatorname{FG} \neg \operatorname{cur}\left(s^{\prime}\right)$, so for all $0 \leq k \leq n, \pi[i+k] \vDash \neg \operatorname{cur}(s)$, hence $\sigma_{\pi}\left(\mathbf{i d l e}_{s}\right)=$ true, and $\sigma_{\pi}$ is a satisfying assignment for $\operatorname{SAT}(N) \wedge$ idle $_{s^{\prime}}$.

So, in all cases, a satisfying assignment for $\operatorname{SAT}(N) \wedge$ idle $_{S^{\prime}}$ exists.
We finally return to the proof of Theorem 3.34 to establish that our idle and block equations are sound. The structure of the proof is similar to that of the previous lemma.

Theorem 3.34. Let $M=\left(S^{z}, s_{0}^{z}, I^{z}, O^{z}, T^{z}\right)$ be an $F S M$ that appears in an $x M A S$ network $N$. Let $x \in I^{z}$ be channel with $d \in c(x)$. If there exists a fair maximal path $\pi \in$ Paths(s) such that $\pi \vDash \operatorname{dead}(x(d))$, and an assignment $\sigma$ that is $\pi$-consistent with respect to $V(\operatorname{SAT}(N)) \backslash V(M)$ and $R(M)$, then there exists a satisfying assignment to the formula $\operatorname{SAT}(N) \wedge \neg \mathbf{i d l e}{ }_{x}^{d} \wedge$ block $_{x}$.

Proof. Assume there exists a fair maximal path $\pi$ such that $\pi \vDash \operatorname{dead}(x(d))$. Let $\sigma$ be an assignment as specified.
By definition of dead, $\pi \vDash \neg \operatorname{idle}(x(d)) \wedge \operatorname{block}(x)$, hence $\pi \vDash \neg \operatorname{idle}(x(d))$ and $\pi \vDash$ $\operatorname{block}(x)$. Since $\mathbf{i d l e}_{x}^{d} \notin V(M), \sigma\left(\right.$ idle $\left._{x}^{d}\right)=$ false according to the assumptions.
We distinguish two cases
$\pi$ is finite. Let $p \in S^{z}$ be such that last $(\pi)=p$. Then according to Lemma 3.36, the assignment $\sigma_{p}$ is consistent with $\operatorname{SAT}(N)$. Note that $\sigma_{p}\left(\right.$ block $\left._{x}\right)=$ true and since $i d l e_{x}^{d} \notin V(M), \sigma_{p}\left(\mathbf{i d l e}_{x}^{d}\right)=\sigma\left(\mathbf{i d l e}_{x}^{d}\right)=$ false $)$. Hence we have a satisfying assignment for $\operatorname{SAT}(N) \wedge \neg$ idle $_{x}^{d} \wedge$ block $_{x}$.
$\pi$ is infinite. We distinguish two cases.
First, suppose $\pi \vDash \operatorname{FG}\left(\operatorname{cur}\left(s^{\prime}\right) \wedge \bigwedge_{t \in o u t_{s}\left(s^{\prime}\right)} \neg\right.$ enabled $\left.(t)\right)$ for some $s^{\prime} \in S$. Let $p$ be such. According to Lemma 3.37, $\sigma_{p}$ is consistent with $\operatorname{SAT}(N)$. Using similar reasoning as in the previous case, we can conclude that $\sigma_{p}$ is a satisfying assignment for $\operatorname{SAT}(N) \wedge \neg$ idle $_{x}^{d} \wedge$ block $_{x}$.

Otherwise, for all $s^{\prime \prime} \in S^{z}$, we have $\pi \not \vDash \operatorname{FG}\left(\operatorname{cur}\left(s^{\prime \prime}\right) \wedge \bigwedge_{t \in o u t_{s}\left(s^{\prime \prime}\right)} \neg\right.$ enabled $\left.(t)\right)$, i.e., $\pi \vDash \operatorname{GF}\left(\operatorname{cur}\left(s^{\prime \prime}\right) \Longrightarrow \bigvee_{t \in \text { out }_{s}\left(s^{\prime \prime}\right)}\right.$ enabled $\left.^{\prime}(t)\right)$.

According to Lemma 3.39, $\sigma_{\pi}$ is consistent with $\operatorname{SAT}(N)$. Note that since $\pi \vDash$ $\operatorname{dead}(x(d))$, $\pi \vDash \operatorname{block}(x(e))$ for all $e \in c(x)$, according to Lemma 3.14. Consider arbitrary $e \in c(x)$, we show that the assignment satisfies block ${ }_{x}^{e}$. From this and the definition it immediately follows that is satisfies block $_{x}$. Let $i$ be the index that signals the start of the loop of the lasso $\pi$. Since $\pi \vDash \operatorname{block}(x(e))$, $\pi \vDash \operatorname{FG}(\neg x$.trdy $\vee x$.data $\neq e)$. By definition of enabled, this implies $\pi \vDash$ $\mathrm{FG}(\neg$ enabled $(t))$ for all $t \in \operatorname{read}(x, e)$. Hence, for all $0 \leq k \leq n, \pi[i+k] \vDash \neg$ enabled $(t)$ for all $t \in \operatorname{read}(x, e)$. By definition of $\sigma_{\pi}$, we then have $\sigma_{\pi}\left(\mathbf{b l o c k}_{x}^{e}\right)=$ true. Since this holds for all $e$, by definition also $\sigma_{\pi}\left(\right.$ block $\left._{x}\right)=$ true, and $\sigma_{\pi}$ is a satisfying assignment for $\operatorname{SAT}(N) \wedge \neg$ idle $_{x}^{d} \wedge$ 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.

### 3.5.3 Examples

We reconsider the example that illustrated the approach from [Ver+16; Ver+17] was unsound, and show that our approach correctly detects deadlocks.

Example 3.41. Recall the finite state machine from Figure 3.3. We repeat it here as Figure 3.4, and update the notation to be consistent with our definitions. Note that the FSM only uses a single token as data, and data is therefore omitted from the figure.


Figure 3.4: Finite state machine from Figure 3.3

The full SAT encoding of this example is the following:

$$
\begin{aligned}
& \text { idle }_{s_{0}}=\neg \operatorname{cur}_{s_{0}} \wedge \operatorname{dead}_{s_{0}} \xrightarrow{x / 0} s_{s_{1}} \\
& \text { idle }_{s_{1}}=\neg \operatorname{cur}_{s_{1}} \wedge \operatorname{dead}_{s_{0}} \xrightarrow{y / z} s_{1} \wedge \operatorname{dead}_{s_{1}} \xrightarrow{x / z} s_{1} \\
& \text { dead }_{s_{0} \xrightarrow{x / 0} s_{1}}=\text { idle }_{s_{0}} \vee \text { idle }_{x} \vee \text { block }_{o} \\
& \text { dead } \underset{s_{0} \xrightarrow{y / z}}{ }=\text { idle }_{s_{0}} \vee \text { idle }_{y} \vee \text { block }_{z} \\
& \text { dead }_{s_{1} \xrightarrow{x / z} s_{1}}=\text { idle }_{s_{1}} \vee \text { idle }_{x} \vee \text { block }_{z} \\
& \text { block }_{x}=\operatorname{dead}_{s_{0}} \xrightarrow{x / 0} s_{s_{1}} \wedge \operatorname{dead}_{s_{1} \xrightarrow{x / z} s_{1}} \\
& \text { block }_{y}=\operatorname{dead}_{s_{0}} \xrightarrow{y / z} s_{1} \\
& \text { idle }_{0}=\operatorname{dead}_{s_{0} \xrightarrow{x / o} s_{1}} \\
& \text { idle }_{z}=\operatorname{dead}_{s_{1} \xrightarrow{x / z}}^{s_{1}}
\end{aligned}
$$

The environment is such that it guarantees that $\mathbf{i d l e}_{x}=\mathbf{i d l e}_{y}=$ block $_{o}=$ block $_{z}=$ false. Now, the following assignment is a satisfying assignment for this system:

$$
\begin{aligned}
\boldsymbol{\operatorname { c u r }}_{s_{0}} & :=\text { false } & \operatorname{cur}_{s_{1}} & :=\text { true } \\
\boldsymbol{\operatorname { i d l e }}_{s_{0}} & :=\text { true } & \operatorname{idle}_{s_{1}} & :=\text { false } \\
\text { dead }_{s_{0}} \xrightarrow{x / 0} s_{s_{1}} & :=\text { true } & \operatorname{dead}_{s_{0}} \xrightarrow{y / z} s_{1} & :=\text { true } \quad \text { dead }_{s_{1} \xrightarrow{x / z}}^{\text {block }_{1}}:=\text { false } \\
\text { block }_{y} & :=\text { false } & \text { blocke }_{y} & \\
\text { idle }_{0} & :=\text { true } & \text { idle }_{z} & :=\text { false }
\end{aligned}
$$

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

### 3.6 Invariants

In the form introduced thus far, there are many satisfying assignments to Boolean equations that are not reachable (for instance, see Figure 3.3). To restrict the number of satisfying assignments, the reachable state space can be approximated using invariants as proposed by Chatterjee and Kishinevsky [CK12]. Essentially, for every channel $x$, and datum $d \in c(x)$, a variable $\lambda_{x}^{d}$ is introduced to represents the number of times $d$ was transferred along channel $x$, i.e., the number of clock ticks at which $x$.irdy $\wedge x$.data $=d \wedge x$.trdy held true. For queues $q$, \#q. $d$ denotes the number of $d$ packets in the queue. For every primitive, the $\lambda$ 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 $f, \lambda_{y}^{e}=\sum\left\{\lambda_{x}^{d} \mid f(d)=e\right\}$, 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 their approach to our setting.

The first invariant requires that the FSM is always in exactly one state. Note that we abuse notation and write $\operatorname{cur}_{s}=1$ whenever cur $=$ true. ${ }^{4}$ This is [Ver+17, Invariant (1)]:

$$
\sum_{s \in S} \operatorname{cur}_{s}=1
$$

The second invariant relates the number of times incoming and outgoing transitions of a state $s$ have been taken. This uses variables $\kappa_{t}$, denoting the number of times transition $t$ has been taken. The resulting invariant for a state $s$ is [Ver+17, Invariant (2)]:

$$
\sum_{t \in \text { in }_{s}(s)} \kappa_{t}=\left(\sum_{t \in o u t_{s}(s)} \kappa_{t}\right)+\operatorname{cur}_{s}-\left(s=s_{0}\right)
$$

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 [Ver+17] 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 [Ver+17, Invariant (3)], where for $x \in I^{z}$ and $d \in c(x)$, we get:

$$
\lambda_{x}^{d}=\sum_{t \in \operatorname{read}(x, d)} \kappa_{t}
$$

For output channels $y \in O^{z}$ and $d \in c(y)$, we get a similar equation inspired by [Ver+17, Invariant (4)]:

$$
\lambda_{y}^{d}=\sum_{t \in w r i t e(y, d)} \kappa_{t}
$$

These invariants are incorporated in our proofs in a similar way as in [CK12; Ver+17].

[^8]
### 3.7 Experiments

We have implemented the idle and block equations described in Section 3.5 in our MaDL design \& verification toolset [FS21]. 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 [Cav+14] to check reachability of a state in which a channel of the given xMAS model is not idle and blocked.

### 3.7.1 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 3.5. 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 $o k$, or nok to the first output. By combining two such FSMs we obtain a "go/no go" building block, depicted in Figure 3.6.


Figure 3.5: "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 blocks from Figure 3.6.

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


Figure 3.6: "go/no go" block.
a new state with a self-loop reading $o k$ 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 3.7 gives an overview of the structure of our power domain models. A power domain controller (Figure 3.8e) 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.
Within every power domain, there is a number of device-controller pairs. Devices are modeled using two FSMs: the activity generation FSM (Figure 3.8c), and the device FSM (Figure 3.8a). 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 3.8b. 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


Figure 3.7: Power domain.
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 3.8d. We scale the power domain models by adding more power domains and more device-controller pairs to 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 [MB08]. For reachability checks, we use nuXmv, version 2.0.0 64-bit [Cav+14]. Instructions to reproduce the experiments and the script used to obtain the results reported in this chapter can be found at [FS21].

### 3.7.2 Results

The times required for the experiments are reported in Table 3.1. The Model column indicates the model that is evaluated. For go/no go models, the number in the name signifies the number of blocks. For power domain models, the first and second number in the name denote the number of power domains and device-controller pairs in every domain, respectively. \#FSMs reports the number of FSMs in the model. In the Live column, $\checkmark$ indicates that the model is deadlock free, $\boldsymbol{X}$ indicates it is not. For each instance, we list the result reported by the tool (Res.), where $\checkmark$ and $\boldsymbol{X}$ represent absence and presence of deadlocks, respectively. Running time for each instance is reported in seconds.

For both sets of models, SAT and reachability correctly report absence and existence of deadlocks in all models. The largest go/no go models contain 126 FSMs. Liveness of the largest deadlock free go/no go model is proven using SAT in 7 seconds.


Figure 3.8: FSMs of power domain.

| Model | \#FSMs | Live | SAT |  | Reachability |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  | Res. | Time (s) | Res. | Time (s) |
| gonogo_1 | 2 | $\checkmark$ | $\checkmark$ | 0.1 | $\checkmark$ | 0.2 |
| gonogo_1_dl | 2 | $x$ | $x$ | 0.1 | $x$ | 0.3 |
| gonogo_2 | 6 | $\checkmark$ | $\checkmark$ | 0.1 | $\checkmark$ | 0.5 |
| gonogo_2_dl | 6 | $x$ | $x$ | 0.1 | $x$ | 2.4 |
| gonogo_3 | 14 | $\checkmark$ | $\checkmark$ | 0.3 | $\checkmark$ | 1.5 |
| gonogo_3_dl | 14 | $X$ | $x$ | 0.3 | $x$ | 5.9 |
| gonogo_4 | 30 | $\checkmark$ | $\checkmark$ | 0.6 | $\checkmark$ | 5.5 |
| gonogo_4_dl | 30 | $X$ | $X$ | 0.6 | $X$ | 18.0 |
| gonogo_5 | 62 | $\checkmark$ | $\checkmark$ | 2.0 | $\checkmark$ | 21.2 |
| gonogo_5_dl | 62 | $x$ | $X$ | 1.9 | $x$ | 54.9 |
| gonogo_6 | 126 | $\checkmark$ | $\checkmark$ | 7.9 | $\checkmark$ | 92.7 |
| gonogo_6_dl | 126 | $x$ | $x$ | 6.7 | $x$ | 221.7 |
| power1_5 | 25 | $\checkmark$ | $\checkmark$ | 0.4 | $\checkmark$ | 1.6 |
| power1_5_dl | 25 | $X$ | $X$ | 0.2 | $X$ | 2.1 |
| power10_5 | 259 | $\checkmark$ | $\checkmark$ | 14.0 | $\checkmark$ | 120.0 |
| power10_5_dl | 259 | $X$ | $x$ | 10.1 | $x$ | 104.2 |
| power20_5 | 519 | $\checkmark$ | $\checkmark$ | 57.5 | $\checkmark$ | 564.8 |
| power20_5_dl | 519 | $X$ | $X$ | 50.3 | $X$ | 451.1 |
| power30_5 | 779 | $\checkmark$ | $\checkmark$ | 352.5 | $\checkmark$ | 1597.4 |
| power30_5_dl | 779 | $x$ | $x$ | 262.9 | $X$ | 1107.3 |
| power40_5 | 1039 | $\checkmark$ | $\checkmark$ | 410.2 | $\checkmark$ | n/a |
| power40_5_dl | 1039 | $X$ | $X$ | 245.6 | $X$ | n/a |
| power50_5 | 1299 | $\checkmark$ | $\checkmark$ | 542.2 | $\checkmark$ | n/a |
| power50_5_dl | 1299 | $x$ | $x$ | 481.1 | $x$ | n/a |

Table 3.1: Experimental results

Reachability analysis takes 1 minute 32 seconds for the same go/no go model. For the largest go/no go model with a deadlock, a deadlock is reported using SAT in 6 seconds. Using reachability it can be proven that a deadlock state is reachable in 3 minutes 41 seconds. As for the power domain experimental set, the largest models (both with and without deadlock) contain 1299 FSMs. For the largest model without a deadlock, SAT proves liveness in 9 minutes and 2 seconds. Analysis of the largest power domain model with a deadlock takes 8 minutes and 1 second using SAT. Reachability analysis for the power domain models with numbers of power domains larger than 30 was not possible in our case. This was caused by nuXmv exceeding the maximum allowed stack on MacOS.

### 3.7.3 Discussion

The results show that using our technique we can prove liveness of large xMAS models with FSMs. We plot the performance results on both sets of models in Figure 3.9. Note that we use the log-scale for the $y$-axis. The results show that


Figure 3.9: Visualization of the results.
both methods scale exponentially in the number of FSMs. However, using SAT for liveness verification significantly outperforms reachability for xMAS extended with FSMs. This is in line with our expectations, and aligns with results for standard xMAS [GCK11].

Although we do not encounter false deadlocks in our experiments, the fact that our method is incomplete implies that finding false deadlocks using SAT is possible. If SAT reports a deadlock, it is not known if the deadlock is reachable or not. In that case, reachability analysis is necessary.

### 3.8 Conclusions

We present a counter-example that is composed of a network with a deadlock that is not found by the technique of Verbeek et al. [Ver+17]. Subsequently, we propose an alternative encoding of liveness into a satisfiability problem. We carefully prove that if an xMAS network has a path to a state with a deadlock, there exists a satisfying assignment to the satisfiability problem we generate, i.e., our encoding is sound.

Finally, we introduce two sets of benchmarks including a simplified power control architecture inspired by industrial case-studies.

The benchmarks and our implementation are publicly available [FS21]. A network with 1299 state machines can be proven live in less than 10 minutes.

## Chapter 4

## Eliminating False Deadlocks in xMAS

### 4.1 Introduction

The importance of the correct functioning of hardware is evident, especially in safetycritical areas. Conventionally, validation of new hardware designs is conducted using testing and simulation. When dealing with complex hardware, testing and simulation cannot guarantee the full coverage of all possible hardware behaviors, which results in a probabilistic conclusion regarding the correctness of the hardware under validation. An alternative approach to validating new hardware designs, which currently becomes more commonly used, is formal verification. In contrast to the traditional approaches to validating hardware designs, formal verification covers all possible hardware behaviors. Besides its advantages, formal verification has a bottleneck as it is usually challenging to scale it to the system level.
xMAS is a language for modeling and verification of hardware designs. The language was introduced originally by researchers of Intel [CKO10]. xMAS contains a number of primitives, by composing which one builds models of hardware. There is a number of verification techniques associated with xMAS. In particular, there is an efficient SAT-based approach by Gotmanov et al. [GCK11; CKO12], which translates the liveness problem of a given xMAS network into a satisfiability problem, which can be then tackled using a SAT-solver. There is also an approach by Wouda et al. [WJS15], which translates a given xMAS network into an SMV model and then uses a symbolic model checker to analyze the reachability of deadlock states.

The SAT-based liveness verification technique discussed in Chapter 3 scales well; we were able to prove the liveness of an xMAS network consisting of 1299 FSMs in less than 10 minutes. Chatterjee and Kishinevsky proved the completeness of the technique for acyclic xMAS networks that contain only basic xMAS primitives [CKO10]; during the experiments with xMAS networks that involve FSMs, we observed false
deadlocks. This chapter focuses on improving the technique so that it becomes complete for any valid xMAS network with FSMs.

Contributions. We demonstrate an example of a false deadlock in the context of xMAS with FSMs. Further, we address this issue while trying to keep scalability reasonable. To check if a deadlock is spurious or not, we focus on backward reachability analysis of an initial state from a deadlock state. We introduce and evaluate two approaches to analyze backward reachability. We base both approaches on the backward reachability analysis due to the conjecture that in practice, the set of states that are backward reachable from a deadlock state is smaller than the set of states that are reachable from an initial state.

First, we start with a $k$-step backward reachability using SMT. The method generates a satisfiability problem that, given a deadlock state and a $k \geq 0$, answers whether an initial state is backward-reachable in up to $k$ steps or it is possible to do $k$ steps backward. The method is complete for large enough $k$. However, for large xMAS networks, full checking of backward reachability of an initial state from the given deadlock state might be hard. So, our $k$-step backward reachability method allows making a trade-off by using small enough $k$ to discard some false deadlocks while still being reasonably scalable. The experiments show that the method fails to prove liveness within a 30 -minute timeout for any model from the experimental set.

As a more effective alternative to the $k$-step backward reachability method, we introduce and evaluate an interpolation-based method for analyzing backward reachability. The method is inspired by the algorithm of McMillan [McM03]. We modified the algorithm of McMillan such that instead of analyzing the reachability of a deadlock state from an initial state, we check the backward reachability of an initial state from a deadlock state. To evaluate our interpolation-based backward reachability method, we use a set of xMAS networks. We also use the same experimental set to compare our interpolation-based backward reachability method with the original method of McMillan. The experiments show that our method performs reasonably well. Moreover, experiments show that in the context of xMAS liveness verification, checking the backward reachability of an initial state from a deadlock state might be advantageous than checking the reachability of a deadlock state from an initial state.

Structure of the chapter. In Section 4.2 we briefly introduce the xMAS language, an effective liveness verification technique for xMAS , and discuss false deadlocks in the context of xMAS liveness verification. In Section 4.3 we introduce and evaluate the $k$-step backward reachability approach. In Section 4.4, interpolation-based backward analysis is introduced and evaluated. In Section 4.5 we discuss the experimental results. Finally, in Section 4.6 we conclude.

### 4.2 Preliminaries

### 4.2.1 xMAS Language

xMAS is a graphical language that offers convenient modeling. The process of xMAS modeling involves composing primitives. The primitives are connected using typed channels.


Figure 4.1: Connection between primitives in xMAS.

Data is transferred through channels following a handshake protocol. For that, every channel carries three signals - irdy, trdy, and data. To give intuition regarding data transfers in xMAS , let us refer to Figure 4.1, where we depict channel $x$ which connects primitives $A$ and $B$. We call $A$ the source and $B$ the target of $x$. Signal irdy of $x$ is controlled by the source and is used to signalize that the source is ready to transfer data. Signal trdy of $x$ is used by $B$ to signalize that it is ready to accept data. Both irdy and trdy are binary signals and can be seen as booleans. Data goes from $A$ to $B$ through signal data of $x$; data transfer only happens if both irdy and trdy are true. In xMAS we assume that all channel are persistent. Whenever the initiator of a channel decides to transfer data, it keeps its irdy and data unchanged until the target of the channel accepts the data. In the current chapter, we only consider synchronous execution of xMAS networks. That is, we assume that all channels which are ready to transfer data do so simultaneously.

The core primitives of $x \mathrm{MAS}$ are depicted in Figure 4.2. In the current chapter we consider the xMAS language with the FSM extension, thus we provide an example of an FSM primitive in Figure 4.3. Further, we will be using source, sink, queue, and FSM primitives in our examples; thus, we provide a detailed description of these primitives. For a detailed description of the other primitives, we refer the reader to Chapter 2.


Figure 4.2: Core xMAS primitives [CKO10].
A source non-deterministically injects data into the network infinitely often. This is modeled using the unconstrained primary input oracle. Once a source decides to transfer datum $d$, it will keep trying until the transfer succeeds. This is modeled


Figure 4.3: A Finite State Machine in xMAS.
using the standard synchronous operator pre that returns the value of its argument in the previous clock cycle, and false in the very first cycle. Formally, the source is described as follows:

$$
\begin{aligned}
& o . \text { irdy }:=\text { oracle } \vee \text { pre }(o . i r d y \wedge \neg o . t r d y) \\
& o . \text { data }:=d .
\end{aligned}
$$

A sink consumes data from the network infinitely often:

$$
i . t r d y:=\text { oracle } \vee \text { pre }(i . \operatorname{trdy} \wedge \neg i . \text { irdy }) .
$$

A queue is a FIFO buffer with $k$ places. A queue is ready to write data to the output when it is not empty. The data the queue is ready to write is the head of the queue. A queue is ready to accept data when it is not full. Formally,

$$
\begin{aligned}
o . i r d y & :=|\mathrm{xs}|>0,
\end{aligned} \quad \text { o.data }:=\operatorname{rhead}(\mathrm{xs}),
$$

where $i$ and $o$ are the input and output channels of the queue respectively.
A finite state machine ( $F S M$ ) is a tuple $\left(S, s_{0}, I, O, T\right)$, 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; and $T \subseteq S \times(I \times C) \times(O \times C) \times S$ is the total transition relation. We use names $s, s^{\prime}, s_{1}, \ldots$ for states. We write $s \xrightarrow{? x(d) /!y(e)} s^{\prime}$ for $\left(s,(x, d),(y, e), s^{\prime}\right) \in T$. For state $s \in S, \operatorname{in}_{s}(s)$ and out $_{s}(s)$ denote the sets of incoming and outgoing transitions of $s$, respectively. Similarly, for channels $x \in(I \cup O)$, and data $d \in C(x), \operatorname{read}(x, d)$ and $\operatorname{write}(x, d)$ represent the sets of transitions reading $d$ from $x$ and writing $d$ to $x$, respectively. In an FSM, exactly one state is current at a time, this state is denoted $\operatorname{cur}(s)$.

A transition $s \xrightarrow{? x(d) /!y(e)} s^{\prime}$ 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. In any state, there can be multiple enabled transitions. To resolve this non-determinism, an arbiter sel is introduced that selects an enabled transition during every clock cycle. If transition $t$ is selected, this is denoted sel $=t$. Formally, given FSM $\left(S, s_{0}, I, O, T\right)$,
for all $i \in I, o \in O$ :

$$
\begin{aligned}
& i . \text {.trdy }:=\exists s \in S, d \in c(i), t \in\left(i n_{s}(s) \cap \operatorname{read}(i, d)\right) . \text { sel }=t, \\
& o . \text { irdy }:=\exists s \in S, e \in c(o), t \in\left(i n_{s}(s) \cap \operatorname{write}(o, e)\right) . \mathbf{s e l}=t, \\
& \text { o.data }:= \begin{cases}e & \text { if } \exists s \in S, e \in c(o), t \in\left(\operatorname{in}_{s}(s) \cap \operatorname{write}(o, e)\right) . \mathbf{s e l}=t \\
\perp & \text { otherwise. }\end{cases}
\end{aligned}
$$

To give a more formal perspective on xMAS, let us introduce the following definitions. The set of component types is
$\Gamma=\{$ source, sink, function, fork, join, switch, merge, FSM$\} \cup\left\{\right.$ queue $\left._{k} \mid k \in \mathbb{N}\right\}$.
Note that the queue type is parameterized in order to reflect the sizes of queues.
Definition 4.1. An xMAS network is a structure ( $P, G, C, c$, chan, type) where:

- $P$ is the set of components;
- $G$ is the set of channels;
- $C$ is a non-empty set of data, which consists of all possible values of data signals of all channels $x \in G$;
- $c: G \rightarrow\left(2^{C} \backslash\{\emptyset\}\right)$ is the function that assigns sets of data to channels from $G$;
- chan : $P \times\{$ in, out $\} \times \mathbb{N} \rightarrow G$ is a partial function which, given a component $p \in P$, an input/output identifier and a channel number $n \in \mathbb{N}$, returns the input (output) channel number $n$ of the component $p$;
- type : $P \rightarrow \Gamma$ is the function that assigns a type to a component.


### 4.2.2 Liveness in xMAS

Before we discuss liveness in the context of xMAS, it is important to recall the notion of idle and block from [GCK11]; see also Chapter 3. A channel is idle for some datum $d$ if eventually the initiator of the channel will never transfer $d$ through the channel. A channel is blocked if eventually the target of the channel will never accept data transfers through the channel. In terms of LTL, idle and block are defined as follows. Given a channel $x$ and a datum $d \in c(x)$,

$$
\begin{aligned}
\operatorname{idle}(x(d)) & :=\mathrm{FG}(\neg x . \operatorname{irdy} \vee x . \text { data } \neq d), \\
\operatorname{block}(x) & :=\mathrm{FG} \neg x . \operatorname{trdy} .
\end{aligned}
$$

Channel $x$ is live for some $d \in c(x)$ when it is idle for $d$ or not blocked:

$$
\operatorname{live}(x(d)):=\operatorname{idle}(x(d)) \vee \neg \operatorname{block}(x)
$$

Channel $x$ is dead for some $d \in c(x)$ when there is a path to a state that satisfies not live:

$$
\operatorname{dead}(x(d)):=\neg \operatorname{live}(x(d)) .
$$

Channel $x$ is live when it is live for all $d \in c(x)$ :

$$
\operatorname{live}(x):=\bigwedge_{d \in C(x)} \operatorname{live}(x(d))
$$

Channel $x$ is dead when there is $d \in c(x)$ for which $x$ is dead:

$$
\operatorname{dead}(x):=\bigvee_{d \in C(x)} \operatorname{dead}(x(d))
$$

The idea behind the SAT-based liveness verification technique for xMAS is to translate the liveness problem for a given xMAS network into a satisfiability problem by approximating dead property for every channel $x \in G$. The method works such that if the given xMAS network can reach a situation where one of its channels is dead, then the liveness satisfiability problem is satisfiable. However, if the liveness satisfiability problem has a satisfying assignment, it does not mean that the xMAS network can reach a situation with a dead channel. That is, the method is incomplete. When used with flow invariants introduced by Chatterjee and Kishinevsky [CK12], the method is complete for acyclic xMAS networks that contain only the core primitives. The invariants work as follows. For every channel $x$, and datum $d \in c(x)$, we introduce a variable $\lambda_{x}^{d}$ which reflects the number of times $d$ was transferred throughout channel $x$. For every queue $q$, we introduce a variable \#q. $d$ which expresses the number of $d$-packets stored in the queue. Invariants relate the $\lambda$ values for input and output channels, also taking into account the $\# q$ variables.

- For a queue primitive with input channel $i$ and output channel 0 , for all $d \in c(i)$, the number of $d$ packets transferred through output $o$ equals the number of $d$ packets transferred through input $i$ minus the number of $d$ packets that are still in the queue:

$$
\forall d \in c(i) \cdot \lambda_{o}^{d}=\lambda_{i}^{d}-\# q \cdot d .
$$

- For a function primitive with input channel $i$, output channel $o$, and a bijective data transforming function $f$, for all $d \in c(i)$, the number of $f(d)$ packets transferred through output $o$ is equal to the number of $d$ packets transferred through $i$. That is:

$$
\forall d \in c(i) \cdot \lambda_{o}^{f(d)}=\lambda_{i}^{d}
$$

- For a fork primitive with input channel $i$, and output channels $o$ and $u$, for all $d \in c(i)$, the number of $d$ packets transferred through each of the outgoing packets is equal to the number of $d$ packets transferred through $i$ :

$$
\forall y \in\{o, u\}, d \in c(y) \cdot \lambda_{y}^{d}=\lambda_{i}^{d} .
$$

- For a join primitive with input channels $\{i, j\}$, and output channel $o$, for all $x \in\{i, j\}$, and for all $d$ that go through $x$ and $o$, the number of $d$ packets transferred through $o$ equals the number of $d$ packets transferred through $x$. That is:

$$
\forall x \in\{i, j\}, d \in(c(o) \cap c(x)) \cdot \lambda_{o}^{d}=\lambda_{x}^{d} .
$$

- For a switch primitive with input channel $i$, and output channels $\{0, u\}$, for all $y \in\{0, u\}, d \in c(y)$, the number of $d$ packets transferred through output $y$ equals the number of $d$ packets transferred through $i$. That is:

$$
\forall y \in\{o, u\}, d \in c(i) . \lambda_{y}^{d}=\lambda_{i}^{d} .
$$

- For a merge primitive with input channels $\{i, j\}$, and output channel $o$, for all $d \in c(o)$, the number of $d$ packets transferred through output $o$ equals the sum of the numbers of $d$ packets transferred through $i$ and $j$. That is:

$$
\forall d \in c(o) \cdot \lambda_{o}^{d}=\lambda_{i}^{d}+\lambda_{j}^{d} .
$$

Verbeek et al. introduced invariants for FSMs in xMAS [Ver+17]. For the details, we refer the reader to Chapter 3.

### 4.2.3 Incompleteness of Flow Invariants

Even though the invariants presented above provide completeness for acyclic xMAS networks that contain core primitives only, incompleteness is still a problem for xMAS with FSMs. Further, we provide a detailed example of a false deadlock that cannot be ruled out using the flow invariants. Consider an xMAS network depicted in Figure 4.4. For simplicity, assume that data in the network is limited to tokens. Assume that the source provides tokens infinitely often and the sink is ready to consume tokens infinitely often. The invariants, in that case, would be as follows.

$$
\begin{aligned}
\mathbf{c u r}_{s_{0}} & +\mathbf{c u r}_{s_{1}}=1 \\
\mathbf{c u r}_{p_{0}} & +\operatorname{cur}_{p_{1}}=1 \\
\kappa_{i} & =\kappa_{s_{0} \rightarrow s_{1}}+\kappa_{s_{1} \rightarrow s_{0}} \\
\kappa_{a} & =\kappa_{s_{0} \rightarrow s_{1}} \\
\kappa_{b} & =\kappa_{s_{1} \rightarrow s_{0}} \\
\kappa_{a^{\prime}} & =\kappa_{p_{0} \rightarrow p_{1}}+\kappa_{p_{1} \rightarrow p_{1}} \\
\kappa_{b^{\prime}} & =\kappa_{p_{1} \rightarrow p_{0}} \\
\kappa_{0} & =\kappa_{p_{0} \rightarrow p_{1}}+\kappa_{p_{1} \rightarrow p_{1}}+\kappa_{p_{1} \rightarrow p_{0}} \\
\kappa_{a^{\prime}} & =\kappa_{a}-\# q_{0} \\
\kappa_{b^{\prime}} & =\kappa_{b}-\# q_{1} \\
\kappa_{0^{\prime}} & =\kappa_{0}-\# q_{2} \\
\kappa_{s_{1} \rightarrow s_{0}} & =\kappa_{s_{0} \rightarrow s_{1}}+\operatorname{cur}_{s_{0}}-\left(s_{0}=s_{0}\right) \\
\kappa_{s_{0} \rightarrow s_{1}} & =\kappa_{s_{1} \rightarrow s_{0}}+\operatorname{cur}_{s_{1}}-\left(s_{1}=s_{0}\right) \\
\kappa_{p_{1} \rightarrow p_{0}} & =\kappa_{p_{0} \rightarrow p_{1}}+\operatorname{cur}_{p_{0}}-\left(p_{0}=p_{0}\right) \\
\kappa_{p_{1} \rightarrow p_{1}}+\kappa_{p_{0} \rightarrow p_{1}} & =\kappa_{p_{1} \rightarrow p_{1}}+\kappa_{p_{1} \rightarrow p_{0}}+\operatorname{cur}_{p_{1}}-\left(p_{1}=p_{0}\right)
\end{aligned}
$$

These equations have a satisfying assignment provided below.

$$
\operatorname{cur}_{s_{0}}:=0 \quad \operatorname{cur}_{s_{1}}:=1
$$

$$
\begin{aligned}
\operatorname{cur}_{p_{0}} & :=1 \\
\kappa_{i} & :=3 \\
\kappa_{b} & :=1 \\
\kappa_{b^{\prime}} & :=0 \\
\kappa_{0^{\prime}} & :=2 \\
\# q_{1} & :=1 \\
\kappa_{s_{0} \rightarrow s_{1}} & :=2 \\
\kappa_{p_{0} \rightarrow p_{1}} & :=0 \\
\kappa_{p_{1} \rightarrow p_{1}} & :=2
\end{aligned}
$$

Thus, states $s_{1}$ and $p_{0}$ are the current states of the leftmost and rightmost FSMs, respectively. The leftmost FSM needs $q_{1}$ to be empty in order to leave its current state, but $q_{1}$ is full. The only way to make $q_{1}$ empty is to take $p_{1} \xrightarrow{? b^{\prime} /!o} p_{0}$ transition in the rightmost FSM. However, the rightmost FSM is stuck in $p_{0}$ since it needs $q_{0}$ to be non-empty in order to leave its current state. Note that channel $i$ is not $i d l e$ since the source tries to transfer tokens infinitely often, but the leftmost FSM lost the ability to accept data transfers. Hence, channel $i$ is also blocked, which makes it dead. The argument that the deadlock described above is unreachable is as follows. The


Figure 4.4: False deadlock example in xMAS
situation does not correspond to an initial state. So, it must have been reached by performing a transition. Call this transition $t$. Transition $t$ can only be $s_{0} \xrightarrow{? i /!a} s_{1}$ or $p_{1} \xrightarrow{? b^{\prime}!!o} p_{0}$. Distinguish these two cases:

- $t=s_{0} \xrightarrow{\text { ?i/!a }} s_{1}$. After taking the transition, $\# q_{0}=1$. This contradicts that $\# q_{0}=0$ in the assignment. So, this is impossible.
- $t=p_{1} \xrightarrow{? b^{\prime}!!o} p_{0}$. After taking the transition, one element is removed from $\# q_{1}$. So, $\# q_{1}<1$, which contradicts that $\# q_{1}=1$ in the assignment.

Hence, there cannot be a last transition allowing us to enter the situation mentioned above, and therefore it is unreachable.

## 4.3 -step Backward Reachability

To limit the number of false deadlocks reported by the SAT-based xMAS liveness verification method, we add $k$-step backward reachability encoding to the original liveness satisfiability problem. The added backward reachability encoding ensures that either an initial state is backward-reachable in up to $k$ steps, or it is possible to do $k$ steps backward without visiting the same state more than once. We further prove that the method guarantees completeness with a large enough $k$.

We start with considering important notions and definitions.
Definition 4.2. A Kripke Structure is a tuple $(S, I, \rightarrow, \mathrm{AP}, L)$, where:

- $S$ is a set of states,
- $I \subseteq S$ is the set of initial states,
- $\rightarrow \subseteq S \times S$ is a transition relation,
- AP is a set of atomic propositions,
- $L: S \rightarrow 2^{\mathrm{AP}}$ is a labelling function.

Further we introduce several types of reachability, starting with a $k$-step reachability of one state from another.

Definition 4.3. Given a $\mathrm{KS}(S, I, \rightarrow, \mathrm{AP}, L), k \geq 0$, and states $s, p \in S$, reachability of $p$ from $s$ in $k$ steps, denoted $s \rightarrow^{k} p$, is defined as follows:

$$
\begin{aligned}
& s \rightarrow^{0} p \text { if } s=p \\
& s \rightarrow^{k+1} p \text { if there is } s^{\prime} \in S \text { such that } s \rightarrow^{k} s^{\prime} \text { and } s^{\prime} \rightarrow p .
\end{aligned}
$$

Next, we introduce reachability of one state in a KS from another in any number of steps.

Definition 4.4. Given a $\mathrm{KS}(S, I, \rightarrow, \mathrm{AP}, L)$, and states $s, p \in S$, we say that $p$ is reachable from $s$, denoted $s \rightarrow^{*} p$ if and only if there is $k \geq 0$ such that $s \rightarrow^{k} p$.
We also introduce reachability of one state from another with the requirement to visit only the states from the given set $S^{\prime}$.

Definition 4.5. Given a $K S(S, I, \rightarrow, \mathrm{AP}, L)$, states $s, p \in S$, and $S^{\prime} \subseteq S$, we say that state $s$ is reachable from state $p$ visiting $S^{\prime}$, denoted $p \rightarrow{ }_{S^{\prime}}^{*} s$ if and only if:

$$
\begin{aligned}
& s=p \text {, or } \\
& \exists s^{\prime} \in\left(S^{\prime} \backslash\{s\}\right) \cdot\left(p \rightarrow_{S^{\prime} \backslash\{s\}}^{*} s^{\prime} \wedge s^{\prime} \rightarrow s\right) .
\end{aligned}
$$

Reachability with the requirement to visit particular states that we introduced above is helpful for requiring to avoid visiting the same state twice before the destination state is reached. In other words, that definition allow us to require avoiding cycles on the way to the target state.

Given a state $s \in S$, and a non-negative number $k$, we want to encode steps backward, such that with every step, we keep track of the states that were seen so far, and we also keep track of the number of steps that we have done so far; we stop when exactly $k$ steps backward were taken. We express such backward reachability recursively as follows.

Definition 4.6. Given a $K S(S, I, \rightarrow, \mathrm{AP}, L)$, for all $s \in S, S^{\prime} \subseteq S$,

$$
\begin{aligned}
\operatorname{Back}_{0}\left(s, S^{\prime}\right) & :=\mathrm{\top}, \\
\operatorname{Back}_{n+1}\left(s, S^{\prime}\right) & :=\exists s^{\prime} \in S \backslash\left(S^{\prime} \cup\{s\}\right) .\left(\operatorname{Back}_{n}\left(s^{\prime}, S^{\prime} \cup\{s\}\right) \wedge s^{\prime} \rightarrow s\right) \vee(s \in I) .
\end{aligned}
$$

For all $s \in S, S^{\prime} \subseteq S$ and $0 \leq i \leq k, \operatorname{Back}_{i}\left(s, S^{\prime}\right)$ are boolean-valued formulas, where $S^{\prime}$ is used to keep track of the states that were discovered so far; $i$ is used to keep track of the steps that were taken backward. Thus, $\operatorname{Back}_{0}\left(s, S^{\prime}\right)$ is always true since the number of backward steps is exceeded.
As part of establishing the correctness of $\operatorname{Back}_{i}\left(s, S^{\prime}\right)$, we need to show that for all states $s \in S$ if there is an initial state $s_{0} \in I$ such that $s$ is reachable from $s_{0}$, then $\operatorname{Back}_{n}(s, \emptyset)$ holds for any $n$. But first, we introduce and prove a lemma and a corollary as stepping stones.
Lemma 4.7. Let $S, S^{\prime}$ be sets such that $S^{\prime} \subseteq S$, and let $s \in S$. Then $S \backslash\left(S^{\prime} \backslash\{s\}\right) \equiv\left(S \backslash S^{\prime}\right) \cup\{s\}$.
Proof. Fix sets $S, S^{\prime}$ such that $S^{\prime} \subseteq S$ and fix $s \in S$. We prove both direction separately.
$\Rightarrow$ We fix $p \in\left(S \backslash\left(S^{\prime} \backslash\{s\}\right)\right)$ and prove that $p \in\left(\left(S \backslash S^{\prime}\right) \cup\{s\}\right)$. By the definition of $\backslash$, $p \in S$ and $p \notin\left(S^{\prime} \backslash\{s\}\right)$. Also, by the definition of $\backslash$, either $p \notin S^{\prime}$ or $p=s$. We distinguish both cases.

- if $p \notin S^{\prime}$, then $p \in\left(S \backslash S^{\prime}\right)$. Hence, $p \in\left(\left(S \backslash S^{\prime}\right) \cup\{s\}\right)$.
- If $p=s$, then it immediately follows that $p \in\left(\left(S \backslash S^{\prime}\right) \cup\{s\}\right)$.
$\Leftarrow$ We fix $p \in\left(\left(S \backslash S^{\prime}\right) \cup\{s\}\right)$ and prove that $p \in\left(S \backslash\left(S^{\prime} \backslash\{s\}\right)\right)$. By the definition of $\cup$, either $p \in\left(S \backslash S^{\prime}\right)$ or $p=s$. We distinguish both cases.
- If $p \in\left(S \backslash S^{\prime}\right)$, then by the definition of $\backslash, p \in S$ and $p \notin S^{\prime}$. Since $p \notin S^{\prime}$ then $p \notin\left(S^{\prime} \backslash\{s\}\right)$. Since $p \in S$ and $p \notin\left(S^{\prime} \backslash\{s\}\right)$, we use the definition of $\backslash$ to conclude $p \in\left(S \backslash\left(S^{\prime} \backslash\{s\}\right)\right)$.
- If $p=s$, then $p \in S$. Also, $p \notin\left(S^{\prime} \backslash\{s\}\right)$. Since $p=s$ and $p \notin\left(S^{\prime} \backslash\{s\}\right)$, $p \in\left(S \backslash\left(S^{\prime} \backslash\{s\}\right)\right)$.

Corollary 4.8. Let $S, S^{\prime}$ be sets such that $S^{\prime} \subseteq S$, and let $s \in S^{\prime}$. Then $S \backslash\left(S^{\prime} \backslash\{s\}\right) \equiv$ $\left(S \backslash S^{\prime}\right) \cup\{s\}$.

Proof. Fix sets $S, S^{\prime}$ such that $S^{\prime} \subseteq S$, and let $s \in S^{\prime}$. Since $S^{\prime} \subseteq S, s \in S$. Therefore, using Lemma 4.7, we conclude $S \backslash\left(S^{\prime} \backslash\{s\}\right) \equiv\left(S \backslash S^{\prime}\right) \cup\{s\}$.

Now we can use the lemma and corollary to prove that if a state $s$ is reachable from an initial state, then $\operatorname{Back}_{n}(s, \emptyset)$ is true holds for all $n \leq 0$.

Lemma 4.9. Given a $K S(S, I, \rightarrow, A P, L)$, for all $n \in \mathbb{N}, s \in S$, if there is $s_{0} \in I$ such that $s_{0} \rightarrow{ }^{*}$ s then $\operatorname{Back}_{n}(s, \emptyset)$.

Proof. We prove that for all $n \in \mathbb{N}, s \in S, S^{\prime} \subseteq S$, if there is $s_{0} \in I$ such that $s_{0} \rightarrow_{S^{\prime}}^{*} s$ then $\operatorname{Back}_{n}\left(s, S \backslash S^{\prime}\right)$, which is a stronger statement. Further we proceed by induction on $n$.

- Base case, $n=0$. Let $s_{0} \in I$. Fix $s \in S$, and $S^{\prime} \subseteq S$ such that $s_{0} \rightarrow_{S^{\prime}}^{*}$ s. By Definition 4.6, Back $_{0}\left(s, S \backslash S^{\prime}\right)$ is true.
- Inductive step, $n=l+1$. The induction hypothesis is that for all $s^{\prime} \in S, S^{\prime \prime} \subseteq S$, if there is $s_{0} \in I$ such that $s_{0} \rightarrow_{S^{\prime \prime}}^{*} s^{\prime}$ then $\operatorname{Back}_{l}\left(s^{\prime}, S \backslash S^{\prime \prime}\right)$ is true. Let $s_{0} \in I$. Fix $s \in S$ and $S^{\prime} \subseteq S$ such that $s_{0} \rightarrow_{S^{\prime}}^{*} s$. We prove that $\operatorname{Back}_{l+1}\left(s, S \backslash S^{\prime}\right)$ is true. Distinguish the following cases based on $s$.
$-s \in I$. Back $_{l+1}^{\prime}\left(s, S \backslash S^{\prime}\right)$ follows immediately from Definition 4.6.
$-s \notin I$. By Definition 4.5, there is $s^{\prime} \in\left(S^{\prime} \backslash\{s\}\right)$ such that $s_{0} \rightarrow_{S^{\prime} \backslash\{s\}}^{*} s^{\prime}$ and $s^{\prime} \rightarrow$ $s$; let $s^{\prime}$ be such. According to the induction hypothesis, $\operatorname{Back}_{l}\left(s^{\prime}, S \backslash\left(S^{\prime} \backslash\{s\}\right)\right)$ is true. Therefore, since $S^{\prime} \subseteq S$ and $s \in S$, by Corrolary 4.8, $\operatorname{Back}_{l}\left(s^{\prime},(S \backslash\right.$ $\left.\left.S^{\prime}\right) \cup\{s\}\right)$. Since $s^{\prime} \rightarrow s$ and $\operatorname{Back}_{l}\left(s^{\prime},\left(S \backslash S^{\prime}\right) \cup\{s\}\right)$, we use Definition 4.6 to conclude that $\operatorname{Back}_{l+1}\left(s, S \backslash S^{\prime}\right)$ is true.

We relate Definition 4.4 and Definition 4.5, which we further use to prove that given a state $s$, if $s$ is not reachable from an initial state, then $\operatorname{Back}_{n}(s, \emptyset)$ does not hold.

Lemma 4.10. Given a $K S(S, I, \rightarrow, A P, L), S^{\prime} \subseteq S$, and states $s, p \in S$, if $s \rightarrow_{S^{\prime}}^{*} p$ then $s \rightarrow^{*} p$.

Proof. Proof by induction on the size of $\left|S^{\prime}\right|$.

- Base case, $\left|S^{\prime}\right|=0 . \quad S^{\prime}=\emptyset$. Fix $s, p \in S$ such that $s \rightarrow_{\emptyset}^{*} p$. Since $S^{\prime}=\emptyset$, from Definition 4.5,s $=p$. Hence, by Definition 4.3, $s \rightarrow^{0} p$ and therefore, by Definition 4.4, $s \rightarrow^{*} p$.
- Inductive step, $\left|S^{\prime}\right|=l+1$. Fix $S^{\prime} \subseteq S, s, p \in S$ such that $\left|S^{\prime}\right|=l+1$ and $s \rightarrow{ }_{S^{\prime}} p$. The IH is that for all $S^{\prime \prime} \subseteq S$ such that $\left|S^{\prime \prime}\right|=l$, and for all $p^{\prime} \in S$, if $s \rightarrow_{S^{\prime \prime}}^{*} p^{\prime}$ then $s \rightarrow{ }^{*} p^{\prime}$. We prove that $s \rightarrow{ }^{*} p$. Since $s \rightarrow{ }_{s^{\prime}}^{*} p$ and $\left|S^{\prime}\right|=l+1$, from Definition 4.5, we know that there is $p^{\prime} \in\left(S^{\prime} \backslash\{p\}\right)$ such that $s \rightarrow_{\left.S^{\prime} \backslash p\right\}}^{*} p^{\prime}$ and $p^{\prime} \rightarrow p$; let $p^{\prime}$ be such. Since $s \rightarrow_{S^{\prime} \backslash\{p\}}^{*} p^{\prime}$, by the $\mathrm{IH}, s \rightarrow^{*} p^{\prime}$. By Definition 4.4, there is $k \geq 0$, such that $s \rightarrow^{k} p^{\prime}$. Therefore, using Definition 4.3 and $p^{\prime} \rightarrow p$, we conclude that $s \rightarrow^{k+1} p$. Finally, by Definition 4.4, we conclude $s \rightarrow^{*} p$.

We next prove a supplementary lemma that will come in handy in the subsequent lemma.

Lemma 4.11. Let $S, S^{\prime}$ be sets such that $S^{\prime} \subseteq S$ and $s \in S^{\prime}$. Then $S \backslash\left(\left(S \backslash S^{\prime}\right) \cup\{s\}\right) \equiv S^{\prime} \backslash\{s\}$.

Proof. By Corollary 4.8, $S \backslash\left(S^{\prime} \backslash\{s\}\right) \equiv\left(S \backslash S^{\prime}\right) \cup\{s\}$. We therefore prove $S \backslash\left(S \backslash\left(S^{\prime} \backslash\right.\right.$ $\{s\})) \equiv S^{\prime} \backslash\{s\}$. Consider both directions separately.
$\Rightarrow$ We fix $p \in\left(S \backslash\left(S \backslash\left(S^{\prime} \backslash\{s\}\right)\right)\right)$ and prove $p \in\left(\left(S \backslash S^{\prime}\right) \cup\{s\}\right)$. By the definition of $\backslash$, $p \in S$ and $p \notin\left(S \backslash\left(S^{\prime} \backslash\{s\}\right)\right)$. Again, by the definition of $\backslash, p \notin S$ or $p \in\left(S^{\prime} \backslash\{s\}\right)$. Since $p \in S \wedge\left(p \notin S \vee\left(S^{\prime} \backslash\{s\}\right)\right)$, we conclude $p \in\left(S^{\prime} \backslash\{s\}\right)$
$\Leftarrow$ We fix $p \in\left(\left(S \backslash S^{\prime}\right) \cup\{s\}\right)$ and prove $p \in\left(S \backslash\left(S \backslash\left(S^{\prime} \backslash\{s\}\right)\right)\right)$. Since $p \in\left(S^{\prime} \backslash\{s\}\right)$ and $\left(S^{\prime} \backslash\{s\}\right) \subseteq S, p \in S$. Assume $p \in\left(S \backslash\left(S^{\prime} \backslash\{s\}\right)\right)$. Then $p \in S$ and $p \notin\left(S^{\prime} \backslash\{s\}\right)$, which is a contradiction, since $p \in\left(S^{\prime} \backslash\{s\}\right)$. Hence, $p \notin\left(S \backslash\left(S^{\prime} \backslash\{s\}\right)\right)$. Since $p \in S$ and $p \notin\left(S \backslash\left(S^{\prime} \backslash\{s\}\right)\right)$, we conclude $p \in\left(S \backslash\left(S \backslash\left(S^{\prime} \backslash\{s\}\right)\right)\right.$ ).

Let us split the set of states $S$ into $S^{\prime} \subseteq S$ and $S \backslash S^{\prime}$, such that there are no initial states in $S^{\prime}$. We show that if $\operatorname{Back}_{\left|S^{\prime}\right|}\left(s, S \backslash S^{\prime}\right)$ then $s$ is reachable from an initial state by visiting states in $S^{\prime}$.

Lemma 4.12. Given a $K S(S, I, \rightarrow, A P, L)$, for all $S^{\prime} \subseteq S$, such that $I \cap S^{\prime} \neq \emptyset$, and for all $s \in S^{\prime}$, if $\operatorname{Back}_{\left|S^{\prime}\right|}\left(s, S \backslash S^{\prime}\right)$ is true then there is $s_{0} \in I$ such that $s_{0} \rightarrow{ }_{S^{\prime}}^{*}$ s.

Proof. Proof by induction on the size of $S^{\prime}$.

- Base case, $\left|S^{\prime}\right|=1$. Fix $s \in S^{\prime}$ such that Back $_{1}\left(s, S \backslash S^{\prime}\right)$ is true. Since $I \cap S^{\prime} \neq \emptyset$ and $I=S^{\prime}=\left\{s_{0}\right\}$, we have $s=s_{0}$. Hence, by Definition 4.5, $s_{0} \rightarrow{ }_{S^{\prime}}^{*} s$.
- Inductive step, $\left|S^{\prime}\right|=l+2$. Fix $S^{\prime} \subseteq S$ such that $I \cap S^{\prime} \neq \emptyset$ and $\left|S^{\prime}\right|=l+2$, fix $s \in S^{\prime}$ such that $\operatorname{Back}_{l+2}\left(s, S \backslash S^{\prime}\right)$. The IH is that for all $S^{\prime \prime} \subseteq S$ such that $I \cap S^{\prime \prime} \neq \emptyset$ and $\left|S^{\prime \prime}\right|=l+1$, and for all $s^{\prime} \in S^{\prime \prime}$, if $\operatorname{Back}_{l+1}\left(s^{\prime}, S \backslash S^{\prime \prime}\right)$ then there is $s_{0} \in I$ such that $s_{0} \rightarrow{ }_{S^{\prime \prime}}^{*} s^{\prime}$. We prove that there is $s_{0} \in I$ such that $s_{0} \rightarrow_{S^{\prime}}^{*} s$. From Definition 4.6, since $\operatorname{Back}_{l+2}\left(s, S \backslash S^{\prime}\right)$, either $s \in I$ or there is $s^{\prime} \in\left(S \backslash\left(\left(S \backslash S^{\prime}\right) \cup\{s\}\right)\right)$ such that Back $_{l+1}\left(s^{\prime},\left(S \backslash S^{\prime}\right) \cup\{s\}\right)$ and $s^{\prime} \rightarrow s$. Consider both cases separately.
- Assume $s \in I$, then $s_{0} \rightarrow{ }_{S^{\prime}}{ }^{\prime}$ immediately follows from Definition 4.5.
- Assume there is $s^{\prime} \in\left(S \backslash\left(\left(S \backslash S^{\prime}\right) \cup\{s\}\right)\right)$ such that $\operatorname{Back}_{l+1}\left(s^{\prime},\left(S \backslash S^{\prime}\right) \cup\{s\}\right)$ and $s^{\prime} \rightarrow s$; let $s^{\prime}$ be such. Since $S^{\prime} \subseteq S$ and $s \in S^{\prime}$, using Corollary 4.8, we have $s^{\prime} \in\left(S^{\prime} \backslash\{s\}\right)$ and using Lemma 4.11, we have Back $_{l+1}\left(s^{\prime}, S \backslash\left(S^{\prime} \backslash\{s\}\right)\right)$. Since $\left|S^{\prime}\right|=l+2$ and $s \in S^{\prime}$, we conclude $\left|S^{\prime} \backslash\{s\}\right|=l+1$. Since $S^{\prime} \subseteq S$ such that $I \cap S^{\prime} \neq \emptyset$ and $s \notin I,\left(S^{\prime} \backslash\{s\}\right) \subseteq S$ such that $I \cap\left(S^{\prime} \backslash\{s\}\right) \neq \emptyset$. Therefore, we can use the IH and conclude that there is $s_{0} \in I$ such that $s_{0} \rightarrow_{S^{\prime} \backslash\{s\}}^{*} s^{\prime}$; let $s_{0}$ be such. Since $s_{0} \rightarrow_{S^{\prime} \backslash\{s\}}^{*} s^{\prime}$ and $s^{\prime} \rightarrow s$, we use Definition 4.6 to conclude $s_{0} \rightarrow{ }_{S^{\prime}}^{*} s$.

Finally, we prove that given a state $s$, if $s$ cannot be reached from an initial state, then $\operatorname{Back}_{n}(s, \emptyset)$ is false.

Lemma 4.13. Given a $K S(S, I, \rightarrow, A P, L)$, there is $n \in \mathbb{N}$ such that for all $s \in S$, if there is $s_{0} \in I$ such that $s_{0} \nrightarrow{ }^{*}$ s then $\operatorname{Back}_{n}(s, \emptyset)$ is false.

Proof. Observe that by Lemma 4.12, for all $s \in S$, if $\operatorname{Back}_{|S|}(s, \emptyset)$ then there is $s_{0} \in I$ such that $s_{0} \rightarrow s^{\prime} s$ and moreover by Lemma 4.10, $s_{0} \rightarrow{ }^{*} s$. Hence, $|S|$ witnesses the claim.

### 4.3.1 Encoding $k$-step Backward Reachablity

In this subsection, we describe how to obtain a boolean-valued formula based on a given xMAS network $N$ so that the formula expresses deadlock states in $N$ and the $k$-step backward reachability from a deadlock state; the formula can be easily adapted for using with an SMT-solver.
We base the encoding on the Kripke Structure semantics of xMAS described in Chapter 2. However, in the said chapter we introduced a KS semantics only for the core primitive of the xMAS language. Since in the current chapter we also consider Finite State Machines in xMAS, we need to introduce the KS semantics for the FSM primitive.
Given an xMAS network $N$, let $\operatorname{KS}(N)=(S, I, \rightarrow, \mathrm{AP}, L)$ be the Kripke Structure reflecting $N$ and let $k \geq 0$ be a backward reachability bound. Let $\pi=s_{0} s_{1} \ldots s_{k}$ be an arbitrary sequence of states in $\mathrm{KS}(N)$, such that for all $0 \leq i<k$ it holds that $s_{i} \rightarrow s_{i+1}$ and $s_{0} \in I$. Then, to express symbolically $s_{i}$, we use a vector of variables $V_{i}$ defined as follows. Let $G=\left\{g_{1}, \ldots, g_{n}\right\}$. Let $\left\{p_{1}, \ldots, p_{m}\right\} \subseteq P$ be the set of merges of $N$. Let $\left\{z_{1}, \ldots, z_{l}\right\} \subseteq P$ be the set of FSMs of $N$. Then,

$$
\begin{aligned}
V_{i}= & \left(g_{1} \cdot \mathbf{i r d y}^{i}, g_{0} \cdot \operatorname{trdy}^{i}, g_{0} \cdot \text { data }^{i}, \ldots, g_{n} \cdot \mathbf{i r d y}^{i}, g_{n} \cdot \operatorname{trdy}^{i}, g_{n} \cdot \text { data }^{i},\right. \\
& \left.p_{1} \cdot \mathbf{m s e l}{ }^{i}, \ldots, p_{m} \cdot \mathbf{m s e l}^{i}, z_{1} \cdot \mathbf{s e l}^{i}, z_{1} \cdot \text { cur }^{i}, \ldots, z_{l} \cdot \text { sel }^{l}, z_{l} \cdot \text { cur }^{l}\right),
\end{aligned}
$$

such that:

- for all $1 \leq j \leq n$, the values of $g_{j} \cdot$ irdy ${ }^{i}$ and $g_{j} \cdot \operatorname{trdy}^{i}$ are booleans and the value of $g_{j}$.data ${ }^{i}$ is an element of $c\left(g_{j}\right)$; the variables express the signals of channel $g_{j}$ after $i$ steps of execution;
- for all $1 \leq j \leq m$, the value of $p_{j} \cdot$ msel $^{i}$ is an element of the following set

$$
\left\{\operatorname{chan}\left(p_{j}, \text { in }, 0\right), \operatorname{chan}\left(p_{j}, \text { in }, 0\right)\right\} ;
$$

$p_{j}$.msel ${ }^{i}$ expresses the value of the arbiter of $p_{j}$ after $i$ steps of execution;

- for all $1 \leq j \leq l$, the value of $z_{j}$.sel ${ }^{i}$ is an element of $T^{z_{j}}$ and the value of $z_{j} \cdot$ cur $^{i}$ is an element of $S^{z_{j}}$, where $T^{z_{j}}$ is the transition relation of $z_{j}$ and $S^{z_{j}}$ is the set of states of $z_{j} ; z_{j}$.cur ${ }^{i}$ expresses the current state of $z_{j}$ after $i$ steps of execution.
Given a vector of variables $V=\left(v_{1}, \ldots, v_{n}\right), S=\left(s_{1}, \ldots, s_{n}\right)$ is a vector of values for $V$. A state predicate $P$ is a boolean-valued formula over variables $V$. Given a vector of variables $W=\left(w_{1}, \ldots, w_{n}\right)$ and a formula $P$ over some vector of variables $V=\left(v_{1}, \ldots, v_{n}\right)$, we write $P(W)$ to denote formula $P$ with $W$ substituting $V$. Similarly, given a vector of state values $S$ and a formula $P$ over some vector of variables $V$, we write $P(S)$ to denote formula $P$ with values $S$ substituting $V$. A state relation $T$
is a boolean-valued formula over two vectors of state variables; we write $T\left(V, V^{\prime}\right)$ to denote that there is a transition from $V$ to $V^{\prime}$.

Let Init be a state predicate that expresses the condition for a state to be in I, Final be a state predicate that expresses the deadlock condition, State be a state predicate that expresses state consistency with respect to the KS semantics of $N$, NotSeen be a state predicate that expresses the condition for a state to be different from the states $S_{j}$, where $i<j \leq k$. Let $T$ be a boolean-valued formula that expresses the transition relation of $K S(N)$ symbolically. For all $0 \leq i \leq k$, let Back $_{i}$ be a boolean variable that is used to characterize that it is possible to do $i$ steps backward without visiting the same state more than once. Then, the $k$-step backward reachability is expressed using the following formula.

$$
\begin{aligned}
\operatorname{KReach}_{k}:= & \left.\bigwedge_{1 \leq i \leq k}\left(\operatorname{Back}_{i} \Leftrightarrow\left(\left(\operatorname{Back}_{i-1} \wedge \operatorname{NotSeen}\left(V_{i-1}\right) \wedge T\left(V_{i-1}, V_{i}\right)\right)\right) \vee \operatorname{lnit}\left(V_{i}\right)\right)\right) \\
& \wedge \bigwedge_{0 \leq i \leq k} \operatorname{State}\left(V_{i}\right) \wedge \operatorname{Back}_{0} \wedge \operatorname{Back}_{k} \wedge \operatorname{Final}\left(V_{k}\right) .
\end{aligned}
$$

We characterize states using the following predicate. The characterization is based on the definition of the set of states of $\operatorname{KS}(N)$, see Chapter 2.

$$
\begin{aligned}
& \text { State }\left(V_{i}\right):=\bigwedge_{p \in P . \text { type }(p)=\text { queue }_{k}} \text { QueueState }_{p}\left(V_{i}\right) \wedge \\
& \bigwedge_{p \in P \text {.type }(p) \text { =function }} \text { FunctionState }_{p}\left(V_{i}\right) \wedge \\
& \bigwedge_{p \in P \text {.type }(p)=\text { fork }} \text { ForkState }_{p}\left(V_{i}\right) \wedge \\
& \bigwedge_{p \in \text { P.type }(p)=\text { join }} \operatorname{JoinState}_{p}\left(V_{i}\right) \wedge \\
& \bigwedge_{p \in \operatorname{P.type}(p)=\text { switch }} \text { SwitchState }_{p}\left(V_{i}\right) \wedge \\
& \bigwedge_{p \in P . \text {.type }(p)=\text { merge }} \operatorname{MergeState}_{p}\left(V_{i}\right) \wedge \\
& \bigwedge_{p \in P . t y p e ~}^{(p)=\mathrm{fsm}} \mathrm{FSMState}_{p}\left(V_{i}\right) .
\end{aligned}
$$

Let $p \in P$ be a queue primitive with $\operatorname{chan}(p$, in, 0$)=x$ and $\operatorname{chan}(p$, out, 0$)=y$. Then,

$$
\begin{aligned}
\text { QueueState }_{p}\left(V_{i}\right):= & \left(x \cdot \operatorname{trdy}^{i} \Leftrightarrow \neg(\mid p \cdot \text { queue } \mid=k)\right) \wedge \\
& \left(y \cdot \operatorname{irdy}^{i} \Leftrightarrow \neg(\mid p \cdot \text { queue }\right. \\
& \left.\left(y \cdot \operatorname{data}^{i}=0\right)\right) \wedge \\
& =\operatorname{last}(p \cdot \text { queue })) .
\end{aligned}
$$

Let $p \in P$ be a function primitive with $\operatorname{chan}(p$, in, 0$)=x, \operatorname{chan}(p$, out, 0$)=y$, and a
data transforming function $f$. Then,

$$
\left.\left.\begin{array}{rl}
\text { FunctionState }_{p}\left(V_{i}\right):= & (y \cdot \mathbf{i r d y} \\
& \Leftrightarrow \\
& \left(x \cdot \operatorname{trdy}^{i} \Leftrightarrow y \cdot \mathbf{i r d} \mathbf{y}^{i}\right) \wedge \\
& \left(y \cdot \operatorname{trdata}^{i}\right) \wedge \\
i
\end{array}\right)\left(x \cdot \mathbf{d a t a}^{i}\right)\right) .
$$

Let $p \in P$ be a fork primitive with $\operatorname{chan}(p$, in, 0$)=x$, $\operatorname{chan}(p$, out, 0$)=y$, and chan $(p$, out, 1$)=z$, and data transforming functions $f$ and $f^{\prime}$. Then,

$$
\begin{aligned}
& \text { ForkState }_{p}\left(V_{i}\right):=\left(y . \mathbf{i r d y}^{i} \Leftrightarrow x . \mathbf{i r d y}^{i} \wedge z . \operatorname{trdy}^{i}\right) \wedge \\
& \left(z . \mathbf{i r d} \mathbf{y}^{i} \Leftrightarrow x . \mathbf{i r d y}^{i} \wedge y . \operatorname{trdy}^{i}\right) \wedge \\
& \left(x \cdot \boldsymbol{t r d y}^{i} \Leftrightarrow y \cdot \operatorname{trdy}^{i} \wedge z \cdot \boldsymbol{t r d y}^{i}\right) \wedge \\
& \left(y \cdot \text { data }^{i}=f\left(x . \text { data }^{i}\right)\right) \wedge \\
& \left(z . \text { data }^{i}=f^{\prime}\left(x . \text { data }^{i}\right)\right) .
\end{aligned}
$$

Let $p \in P$ be a join primitive with $\operatorname{chan}(p$, in, 0$)=x, \operatorname{chan}(p$, in, 1$)=y, \operatorname{chan}(p$, out, 0$)=$ $z$, and a routing function $h$. Then,

$$
\begin{aligned}
\operatorname{JoinState}_{p}\left(V_{i}\right):= & \left(z \cdot \operatorname{irdy}^{i} \Leftrightarrow x \cdot \operatorname{irdy}^{i} \wedge y \cdot \operatorname{irdy}^{i}\right) \wedge \\
& \left(x \cdot \operatorname{trdy}^{i} \Leftrightarrow y \cdot \operatorname{irdy}^{i} \wedge z \cdot \operatorname{trdy}^{i}\right) \wedge \\
& \left(y \cdot \operatorname{trdy}^{i} \Leftrightarrow x \cdot \operatorname{irdy}^{i} \wedge z \cdot \operatorname{trdy}^{i}\right) \wedge \\
& \left(z \cdot \operatorname{data}^{i}=h\left(x \cdot \operatorname{data}^{i}, y \cdot \operatorname{data}^{i}\right)\right) .
\end{aligned}
$$

Let $p \in P$ be a switch primitive with $\operatorname{chan}(p$, in, 0$)=x$, and $\operatorname{chan}(p$, out, 0$)=y$, and chan $(p$, out, 1$)=z$, and a routing function $r$. Then,

$$
\begin{aligned}
& \operatorname{SwitchState}_{p}\left(V_{i}\right):=\left(y . \mathbf{i r d y}^{i} \Leftrightarrow x . \mathbf{i r d y}^{i} \wedge r\left(x . \text { data }^{i}\right)\right) \wedge \\
& \left(z . \mathbf{i r d y}^{i} \Leftrightarrow x . \mathbf{i r d y}^{i} \wedge \neg r\left(x . \boldsymbol{d a t a}^{i}\right)\right) \wedge \\
& \left(x . \operatorname{trdy}^{i} \Leftrightarrow\left(y . \mathbf{i r d y}^{i} \wedge y . \operatorname{trd}^{i}\right) \vee\left(z \cdot \mathbf{i r d} \mathbf{y}^{i} \wedge z . \operatorname{trd}^{i}\right)\right) \wedge \\
& \left(y \cdot \boldsymbol{d a t a}^{i}=x \cdot \text { data }^{i}\right) \wedge \\
& \left(z . \text { data }^{i}=x . \text { data }^{i}\right) .
\end{aligned}
$$

Let $p \in P$ be a merge primitive with $\operatorname{chan}(p, \mathrm{in}, 0)=x$, and $\operatorname{chan}(p, \mathrm{in}, 1)=y$, and $\operatorname{chan}(p$, out, 0$)=z$. Then,

$$
\begin{aligned}
& \operatorname{MergeState}_{p}\left(V_{i}\right):=\left(z . \text {.irdy }^{i} \Leftrightarrow\left(p . \text { msel }^{i} \wedge x . \text {.irdy }^{i}\right) \vee\left(\neg p . \text { msel }_{p}^{i} \wedge y . \text { ird }^{i}\right)\right) \wedge \\
& \left(x . \text {.trdy }^{i} \Leftrightarrow p . \mathbf{m s e l}^{i} \wedge z . \operatorname{trdy}^{i} \wedge x . \mathbf{i r d}^{i}{ }^{i}\right) \wedge \\
& \left(y . \operatorname{trd} \mathbf{y}^{i} \Leftrightarrow \neg p \cdot \mathbf{m s e l}{ }^{i} \wedge z \cdot \operatorname{trdy}^{i} \wedge y . \text { irdy }^{i}\right) \wedge \\
& \left(\left(p . \mathbf{m s e l}^{i} \wedge x . \mathbf{i r d y}^{i}\right) \Rightarrow z \cdot \boldsymbol{d a t a}^{i}=x \cdot \mathbf{d a t a}^{i}\right) \wedge \\
& \left(\left(\neg p \cdot \mathbf{m s e l}^{i} \wedge y \cdot \mathbf{i r d y}^{i}\right) \Rightarrow z \cdot \text { data }^{i}=y \cdot \text { data }^{i}\right) .
\end{aligned}
$$

Let $p \in P$ be an FSM primitive with $p=\left(S^{p}, s_{0}^{p}, I^{p}, O^{p}, T^{p}\right)$, and $I^{p}=\left\{i_{1}, \ldots, i_{v}\right\}$, and $O^{p}=\left\{o_{1}, \ldots, o_{w}\right\}$. Then,

$$
\begin{aligned}
\operatorname{FSMState}_{p}\left(V_{i}\right) & :=\left(\forall s \xrightarrow{\left.? i_{k}(d)\right)!o_{l}(e)} s^{\prime} \in T^{p} \cdot\left(p \cdot \mathbf{s e l} \mathbf{l}^{i}=s \xrightarrow{? i_{k}(d) /!o_{l}(e)} s^{\prime}\right)\right. \\
& \left.\Rightarrow\left(p \cdot \mathbf{c u r}^{i}=s \wedge i_{k} \cdot \mathbf{i r d y ^ { i }} \wedge i_{k} \cdot \mathbf{d a t a}^{i}=d \wedge o_{l} \cdot \mathbf{t r d y}^{i} \wedge o_{l} \cdot \mathbf{d a t a}^{i}=e\right)\right) \wedge \\
& \bigwedge_{1 \leq k \leq v}\left(i_{k} \cdot \operatorname{trdy}^{i} \Leftrightarrow\left(\exists s \xrightarrow{\left.? i_{k}(d)\right)!o_{l}(e)} s^{\prime} \in T^{p} \cdot\left(p \cdot \mathbf{s e l}^{i}=s \xrightarrow{? i_{k}(d)!!o_{l}(e)} s^{\prime}\right)\right)\right) \wedge \\
& \bigwedge_{1 \leq l \leq w}\left(o_{l} \cdot \mathbf{i r d y}^{i} \Leftrightarrow\left(\exists s \xrightarrow{\left.? i_{k}(d)\right)!o_{l}(e)} s^{\prime} \in T^{p} \cdot\left(p \cdot \mathbf{s e l}{ }^{i}=s \xrightarrow{? i_{k}(d)!!o_{l}(e)} s^{\prime}\right)\right)\right) .
\end{aligned}
$$

State predicate NotSeen requires that for all $i<j \leq k$, the given state $V_{i}$ differs from $V_{j}$. Hence,

$$
\operatorname{NotSeen}\left(V_{i}\right):=\bigwedge_{i<j \leq k} V_{i} \neq V_{j}
$$

According to the KS semantics, the transition relations of functions, forks, joins, switches, and merges allow to transition from any state to any state. Hence, in $T$ we only need to take care of sources, sinks, queues and FSMs.

$$
T\left(V_{i}, V_{i+1}\right):=\bigwedge_{p \in P . \text {.type }(p)=\text { source }}^{\bigwedge_{p \in P . \operatorname{type}(p)=\operatorname{sink}} \operatorname{SourceStep}_{p}\left(V_{i}, V_{i+1}\right) \wedge} \bigwedge_{p \in P . \text {.type }(p)=\text { queue }} \bigwedge_{p \in P . \operatorname{Pinpe}(p)=\mathrm{fsm}}\left(V_{i}, V_{i+1}\right) \wedge
$$

For sources, we encode that in case the source tried to transfer data of certain type after step $i$ but failed to do so, the source tries to transfer data of the same type after step $i+1$. Let $p \in P$ be a source primitive with chan $(p, o u t, 0)=o$. Using Src1 and Src2 from the definition of the transition relation for the source primitive (see Chapter 2), we get

$$
\text { SourceStep }_{p}\left(V_{i}, V_{i+1}\right):=\left(o . \operatorname{irdy}^{i} \wedge \neg 0 . \operatorname{trdy}^{i}\right) \Rightarrow\left(0 . \mathbf{i r d y}^{i+1} \wedge 0 . \text { data }^{i}=0 . \text { data }^{i+1}\right)
$$

For sinks, the encoding reflects that if the sink tried to accept data after step $i$, but there was no data to accept, then the sink tries to accept data after step $i+1$. Let $p \in P$ be a sink primitive with $\operatorname{chan}(p, \mathrm{in}, 0)=i$. Using Snk1 and Snk2 from the definition of the transition relation for the sink primitive (see Chapter 2), we get

$$
\operatorname{SinkStep}_{p}\left(V_{i}, V_{i+1}\right):=\left(i . \operatorname{trdy}^{i} \wedge \neg i . \mathbf{i r d y}{ }^{i}\right) \Rightarrow i . \operatorname{trdy}^{i+1}
$$

For queues, we encode the relation between contents of the queue after step $i$ and contents of the queue after step $i+1$ with respect to the input and output signals after step $i$. Let $p \in P$ be a queue primitive with $\operatorname{chan}(p$, in, 0$)=x$ and $\operatorname{chan}(p$, out, 0$)=y$. Using Q1 - Q4 from the definition of the transition relation for the queue primitive (see Chapter 2), we get

$$
\begin{aligned}
& \text { QueueStep }_{p}\left(V_{i}, V_{i+1}\right):=\left(\neg\left(i . \operatorname{irdy}{ }^{i} \wedge i . \operatorname{trd} \mathbf{y}^{i}\right) \wedge \neg\left(o . \mathbf{i r d} \mathbf{y}^{i} \wedge o . \operatorname{trdy}^{i}\right)\right. \\
& \left.\Rightarrow p . \text { queue }^{i+1}=p . \text { queue }^{i}\right) \wedge \\
& \left(\left(i . \mathbf{i r d} \mathbf{y}^{i} \wedge i . \operatorname{trd} \mathbf{y}^{i}\right) \wedge \neg\left(o . \mathbf{i r d} \mathbf{y}^{i} \wedge o . \boldsymbol{t r d y}^{i}\right)\right. \\
& \left.\Rightarrow p \cdot \text { queue }^{i+1}=\left(i . \text { data }^{i}: p \cdot \text { queue }^{i}\right)\right) \wedge \\
& \left(\neg\left(i . \mathbf{i r d} \mathbf{y}^{i} \wedge i . \operatorname{trd} \mathbf{y}^{i}\right) \wedge\left(o . \mathbf{i r d}^{i} \wedge o . \mathbf{t r d y}^{i}\right)\right. \\
& \left.\left.\Rightarrow p . \text { queue }^{i+1}=\left(\text { rtail }^{(p . q u e u e}{ }^{i}\right)\right)\right) \wedge \\
& \left(\left(i . \mathbf{i r d} \mathbf{y}^{i} \wedge i . \operatorname{trd} \mathbf{y}^{i}\right) \wedge\left(o . \mathbf{i r d y}^{i} \wedge o . \boldsymbol{t r d y}^{i}\right)\right. \\
& \left.\Rightarrow p . \text { queue }^{i+1}\left(i . \text { data }^{i}:\left(\operatorname{rtail}\left(p . \text { queue }^{i}\right)\right)\right)\right) .
\end{aligned}
$$

For FSMs, if a transition $s \xrightarrow{? i(d)!!o(e)} s^{\prime}$ of an FSM was selected after step $i$, the FSM after step $i+1$ is at the target state $s^{\prime}$. Let $p \in P$ be an FSM primitive with $p=\left(S^{p}, s_{0}^{p}, I^{p}, O^{p}, T^{p}\right)$. Using FSM1 and FSM2 from the definition of the transition relation for the FSM primitive, we get

$$
\operatorname{FSMStep}_{p}\left(V_{i}, V_{i+1}\right):=\bigwedge_{s \xrightarrow{x(d) / y(e)} s^{\prime} \in T^{p}}\left(p \cdot \mathbf{s e l}^{i}=s \xrightarrow{x(d) / y(e)} s^{\prime} \Rightarrow p \cdot \mathbf{c u r}^{i+1}=s^{\prime}\right)
$$

Initial state predicate Init is expressed as follows. We require that all queues are empty, and the states of all FSMs are initial. That is,

$$
\operatorname{Init}\left(V_{i}\right)=\bigwedge_{p \in P \cdot \operatorname{type}(p)=\text { queue }_{k}} \mid p \cdot \text { queue } \mid=0 \wedge \bigwedge_{p \in P \cdot \operatorname{type}(p)=\mathrm{fsm}} p \cdot \mathbf{c u r}=s_{0}^{p} .
$$

### 4.3.2 Evaluation of $k$-step Backward Reachability

We have implemented a tool that, given an xMAS network and a non-negative bound $k$, generates an SMT-problem that approximates the liveness of the given xMAS network. In addition, the SMT-problem contains the $k$-step backward reachability from every state with a dead channel. If there is no satisfying assignment to the SMTproblem, we conclude that the xMAS network is deadlock-free. Otherwise, there is a deadlock state, from which it is possible to do $k$ steps backward without visiting the same state more than once. Note, that with a large enough $k$, completeness of the method is guaranteed for any valid xMAS network.

To evaluate the effectiveness of the $k$-step backward reachability method, we used a modification of power domain models described in Chapter 2. The SAT-based


Figure 4.5: Modification of the top controller FSM.
liveness verification approach did not report false deadlocks for the original power domain models from Chapter 2. Hence, we modified the top controller FSM as depicted in Figure 4.5. Consequently, the SAT-based liveness verification approach reports a false deadlock for the modified power domain models. The fact that there are no real deadlocks was checked by the reachability analysis of deadlock states using nuXmv. For the experiments with the $k$-step backward reachability, we used the modified power domain models containing one power domain and from two to seven device-controller pairs. The goal of the experiments was to verify if the $k$-step backward reachability is effective enough to prove the liveness of every modified power domain model from our experimental set.

We executed the experiments on a MacBook Pro 2015, 2,7GHz Intel Core i5, 16Gb RAM, running MacOS Big Sur 11.3. For SAT solving, we use the MathSAT5 solver, version 5.6.5 64-bit [Cim+13]. Instructions to reproduce the experiments and the script used to obtain our results are available at [FS21].

We scripted our experimental runs such that when the tool reports a deadlock, the bound is increased, and the experiment is re-started; the script re-starts the experiment until either the tool reports the absence of deadlocks or a 30-minute timeout is reached. The initial values of bounds for every experiment was set to 3 .

We observe that for every model from our experimental set, the tool reports the presence of deadlocks; deadlock freedom was not reported for any of the models within 30 minutes. Using $k$-step backward reachability makes the problem of proving liveness hard even for a model with 9 FSMs. If we refer to the results of the experiments with power domain models from Chapter 3, the SAT-based liveness verification method proved the liveness of a power domain model that contained 1299 FSMs in less than 10 minutes. This comparison leads us to the conclusion that adding $k$-step backward reachability to the SAT-based liveness verification method reduces the performance of the method dramatically, which makes it hardly applicable in practice.

The results show that none of the models from the experimental set can be proven deadlock-free within 30 minutes. This leads to a conclusion that the $k$-step backward reachability is not effective from the practical point of view.

### 4.4 Interpolation-Based Backward Reachability

As a more advanced alternative to encoding backward reachability explicitly, we adapt the idea to use interpolation in reachability analysis presented by McMillan [McM03] to the xMAS setting. In contrast to McMillan et al., we unroll the transition relation backwards and use interpolation to over-approximate the set of states that are backward reachable. We start by introducing notation and notions that are important in introducing interpolation-based backward reachability analysis.

Definition 4.14 ([McM03]). Given two boolean-valued formulas $A$ and $B$, such that there is no satisfying assignment to $A \wedge B$, an interpolant $\operatorname{IPL}(A, B)$ is a boolean-valued formula such that:

- $A \Rightarrow \operatorname{IPL}(A, B)$, and
- there is no satisfying assignment to $\operatorname{IPL}(A, B) \wedge B$.

For the purpose of analyzing backward reachability using interpolation, we represent a Kripke Structure symbolically using boolean-valued formulas. Reuse the notation for vectors of state variables, vectors of state values, state predicates and state relations from Subsection 4.3. Given a boolean-valued formula $\varphi$, we write $\operatorname{SAT}(\varphi)$ if $\varphi$ has a satisfying assignment and UNSAT $(\varphi)$ otherwise.

Given an xMAS network $N=(P, G, C, c$, chan, type $)$, let $K S(N)=(S, I, \rightarrow, A P, L)$ be the corresponding Kripke Structure. We express $\operatorname{KS}(N)$ symbolically as $M=$
(Init, State, T, Final), where Init, State, and Final are state predicates introduced in Subsection 4.3.1. Further, we also use the vectors of state variables and state values that we used in Subsection 4.3.1.

Given $M$, we call a sequence of states $S_{0} \ldots S_{k}$ a path if and only if it holds that

$$
\bigwedge_{0 \leq i<k} T\left(S_{i}, S_{i+1}\right) \wedge \bigwedge_{0 \leq i \leq k} \operatorname{State}\left(S_{i}\right)
$$

where $k$ is the length of the path (i.e. the number of steps needed to reach $S_{k}$ from $\left.S_{0}\right)$. We say that a sequence of states $S_{0} \ldots S_{k}$ is a backward run of $M$ if and only if the following holds:

$$
\operatorname{Final}\left(S_{0}\right) \wedge\left(\bigwedge_{0 \leq i<k} T\left(S_{i+1}, S_{i}\right)\right) \wedge \operatorname{Init}\left(S_{k}\right) \wedge \bigwedge_{0 \leq i \leq k} \operatorname{State}\left(S_{i}\right)
$$

To introduce a characterization of all backward runs of $M$ with length up to $k$ it is convenient to prove that from any initial state of $\operatorname{KS}(N)$ it is possible to take a step backwards to an initial state.

Lemma 4.15. Given a valid $x M A S$ network $N=(P, G, C, c$, chan, type), let $K S(N)=$ $(S, I, \rightarrow, A P, L)$ be the Kripke Structure representing $N$. Then there is $W \subseteq I$ such that the following properties hold.
(1) $\forall s \in W, g \in G .(g . i r d y \notin L(s))$.
(2) Let $H=\{g \in G \mid \forall p \in P, n \in \mathbb{N} . \operatorname{chan}(p, i n, n) \neq g\}$ and $K=\{g \in G \mid \forall p \in P, n \in$ $\mathbb{N}$.chan $(p$, out,$n) \neq g\}$. Then the following hold.

- If $H \neq \emptyset \wedge K=\emptyset$, assume $H=\left\{g_{1}, \ldots, g_{l}\right\}$. Then,

$$
\forall\left(b_{1}, \ldots, b_{l}\right) \in \mathbb{B}^{l} .\left(\exists s \in W . \bigwedge_{1 \leq j \leq l}\left(b_{j} \Leftrightarrow g_{j} \cdot \mathbf{t r d y} \in L(s)\right)\right) .
$$

- If $H=\emptyset \wedge K \neq \emptyset$, assume $K=\left\{x_{1}, \ldots, x_{h}\right\}$. Then,

$$
\forall\left(d_{1}, \ldots, d_{h}\right) \in\left(\prod_{1 \leq u \leq h} c\left(x_{u}\right)\right) .\left(\exists s \in W . \bigwedge_{1 \leq u \leq h}\left(x_{u} \cdot \text { data }=d_{u} \in L(s)\right)\right) .
$$

- If $H \neq \emptyset \wedge K \neq \emptyset$, assume $H=\left\{g_{1}, \ldots, g_{l}\right\}$ and $K=\left\{x_{1}, \ldots, x_{h}\right\}$. Then,

$$
\begin{aligned}
\forall\left(b_{1}, \ldots, b_{l}\right) \in \mathbb{B}^{l},\left(d_{1}, \ldots, d_{h}\right) & \in\left(\prod_{1 \leq u \leq h} c\left(x_{h}\right)\right) . \\
(\exists s \in W \cdot & \bigwedge_{1 \leq j \leq l}\left(b_{j} \Leftrightarrow g_{j} \cdot \mathbf{t r d y} \in L(s)\right) \wedge \\
& \left.\bigwedge_{1 \leq u \leq h}\left(x_{u} \cdot \text { data }=d_{u} \in L(s)\right)\right) .
\end{aligned}
$$

(3) $W \neq \emptyset$.
(4) $\forall s \in W, s^{\prime} \in I .\left(s, s^{\prime}\right) \in \rightarrow$.

Proof. Proof is by induction on the number of primitives in $N$.

- Base case. Assume $|P|=1$. Let $P=\{z\}$. We distinguish cases based on the type of $z$.
$-\operatorname{type}(z)=$ source with chan $(z$, out, 0$)=o$. By definition, $I=\mathbb{B} \times \mathbb{B} \times c(o)$. Let $W=\left\{\left(\right.\right.$ false $\left.\left., o_{\text {trdy }}, d\right) \in I\right\}$. We check that $W$ satisfies properties (1)-(4) of the lemma.
(1) By the definition of $L$, for all $s \in W$ it holds that $o$.irdy $\notin L(s)$.
(2) Observe that $H=\{o\}$ and $K=\emptyset$. We show that for all $b \in \mathbb{B}$ there is $s \in W$ such that $b \Leftrightarrow o . t r d y \in L(s)$.

Fix arbitrary $b \in \mathbb{B}$. Observe that (false, $b, d) \in W$. Using the definition of $L$ we have $b \Leftrightarrow o . t r d y \in L(s)$.
(3) Proving (2), we showed a witness that $W \neq \emptyset$.
(4) Fix arbitrary $s \in W, s^{\prime} \in I$. Without loss of generality, let

$$
\begin{aligned}
s & =\left(o_{\mathbf{i r d y}}, o_{\mathbf{t r d y}}, d\right), \\
s^{\prime} & =\left(o_{\mathbf{i r d y}}^{\prime}, o_{\mathbf{t r d y}}^{\prime}, d^{\prime}\right) .
\end{aligned}
$$

By the definition of $W, o_{\text {irdy }}=$ false, hence according to rule Src1 from the definition of $\rightarrow$ of the source KS we conclude $\left(s, s^{\prime}\right) \in \rightarrow$.
$-\operatorname{type}(z)=\operatorname{sink}$ with chan $(z, \mathrm{in}, 0)=i$. By definition, $I=\mathbb{B} \times \mathbb{B} \times c(i)$. Let $W=\{($ false, false, $d) \in I\}$. We check that $W$ satisfies properties (1)-(4) of the lemma.
(1) By the definition of $L$, for all $s \in W$ it holds that $i . i r d y ~ \notin L(s)$.
(2) Observe that $H=\emptyset$ and $K=\{i\}$. We show that for all $d \in c(i)$ there is $s \in W$ such that $i$.data $=d \in L(s)$. Fix arbitrary $d \in c(i)$. Observe that (false, false, $d$ ) $\in W$; using the definition of $L$ we have $i$.data $=d \in$ $L($ false, false, $d)$ ).
(3) Proving (2), we showed a witness that $W \neq \emptyset$.
(4) Fix arbitrary $s \in I, s^{\prime} \in W$. Without loss of generality, let

$$
\begin{aligned}
s & =\left(i_{\text {irdy }}, i_{\text {trdy }}, d\right), \\
s^{\prime} & =\left(i_{\text {irdy }}^{\prime}, i_{\text {trdy }}^{\prime}, d^{\prime}\right) .
\end{aligned}
$$

By the definition of $W, i_{\text {irdy }}=$ false, hence according to rule Snk1 from the definition of $\rightarrow$ of the sink KS we conclude $\left(s, s^{\prime}\right) \in \rightarrow$.
$-\operatorname{type}(z)=$ queue with $\operatorname{chan}(z$, in, 0$)=i$ and $\operatorname{chan}(z$, out, 0$)=o$. By definition, $I=\left\{\left(\emptyset, i_{\text {irdy }}\right.\right.$, true, $e$, false, $\left.\left.o_{\text {trdy }}, d\right) \mid i_{\text {irdy }}, o_{\text {trdy }} \in \mathbb{B}, e \in c(i), d \in c(o)\right\}$. Let $W=\left\{\left(\emptyset\right.\right.$, false, true,$e$, false $\left.\left., o_{\text {trdy }}, d\right) \in I\right\}$. We check that $W$ satisfies properties (1)-(4) of the lemma.
(1) By the definition of $L$, for all $s \in W, g \in\{i, o\}$ it holds that $g$.irdy $\notin L(s)$.
(2) Observe that $H=\{o\}$ and $K=\{i\}$. We show that for all $b \in \mathbb{B}, d \in c(i)$ there is $s \in W$ such that $b \Leftrightarrow o . \operatorname{trdy} \in L(s)$ and $i$. data $=d \in L(s)$.

Fix arbitrary $b \in \mathbb{B}, d \in c(i)$. Observe that for all $e \in c(o)$, there is

$$
(\emptyset, \text { false }, \text { true }, d, \text { false }, b, e) \in W
$$

. Using the definition of $L$ we have

$$
\begin{aligned}
& b \Leftrightarrow o . \operatorname{trdy} \in L((\emptyset, \text { false, true, } d \text {, false, } b, e)), \\
& i . \text { data }=d \in L((\emptyset, \text { false, true, } d, \text { false }, b, e)) .
\end{aligned}
$$

(3) Proving (2), we showed a witness that $W \neq \emptyset$.
(4) Fix arbitrary $s \in W, s^{\prime} \in I$. Without loss of generality, let

$$
\begin{aligned}
& s=\left(\mathrm{xs}, i_{\text {irdy }}, i_{\text {trdy }}, e, o_{\text {irdy }}, o_{\text {trdy }}, d\right), \\
& s^{\prime}=\left(\mathrm{xs}^{\prime}, i_{\text {irdy }}^{\prime}, i_{\text {trdy }}^{\prime}, e^{\prime}, o_{\text {irdy }}^{\prime}, o_{\text {trdy }}^{\prime}, d^{\prime}\right)
\end{aligned}
$$

By the definitions of $W$ it holds that $\neg\left(i_{\text {irdy }} \wedge i_{\text {trdy }}\right), \neg\left(o_{\text {irdy }} \wedge o_{\text {trdy }}\right)$, by the definition of $I$ it holds that $\mathrm{xs}=\mathrm{xs}^{\prime}$, hence according to rule Q1 from the definition of $\rightarrow$ of the queue KS we conclude $\left(s, s^{\prime}\right) \in \rightarrow$.

- type $(z)=$ FSM with $z=\left(S^{z}, s_{0}^{z}, I^{z}, O^{z}, T^{z}\right)$, and $I^{z}=\left\{i_{1}, \ldots, i_{n}\right\}$, and $O^{z}=$ $\left\{o_{1}, \ldots, o_{m}\right\}$. By definition,

$$
\begin{aligned}
I= & \left\{\left(\mathbf{s e l}, i_{1 \text { irdy }}, i_{1 \text { trdy }}, d_{1}, \ldots, i_{n \text { irdy }}, i_{n \text { trdy }}, d_{n}\right.\right. \\
& \left.o_{1 \text { irdy }}, o_{1 \text { trdy }}, e_{1}, \ldots, o_{m \text { irdy }}, o_{m \text { trdy }}, e_{m}, s_{0}^{z}\right) \in \\
& \left((T \cup\{\perp\}) \times \prod_{i \in I^{Z}}(\mathbb{B} \times \mathbb{B} \times c(i)) \times \prod_{o \in O^{z}}(\mathbb{B} \times \mathbb{B} \times c(o)) \times S^{z}\right) \mid \\
& \forall s \xrightarrow{\left.? i_{k}(d)\right)!o_{l}(e)} s^{\prime} \in T . \mathbf{s e l}=s \xrightarrow{? i_{k}(d) /!o_{l}(e)} s^{\prime} \Rightarrow \\
& \left(\mathbf{c u r}=s \wedge i_{k \text { irdy }} \wedge d_{k}=d \wedge o_{l \text { trdy }} \wedge e_{l}=e\right), \\
& \bigwedge_{1 \leq k \leq n}\left(i_{k \text { trdy }} \Leftrightarrow\left(\exists s \xrightarrow{\left.? i_{k}(d)\right)!o_{l}(e)} s^{\prime} \in T . \mathbf{s e l}=s \xrightarrow{\left.? i_{k}(d)\right)!o_{l}(e)} s^{\prime}\right)\right), \\
& \left.\bigwedge_{1 \leq l \leq m}\left(o_{l \text { irdy }} \Leftrightarrow\left(\exists s \xrightarrow{\left.? i_{k}(d)\right)!o_{l}(e)} s^{\prime} \in T . \mathbf{s e l}=s \xrightarrow{\left.? i_{k}(d)\right)!o_{l}(e)} s^{\prime}\right)\right)\right\} .
\end{aligned}
$$

Let

$$
\begin{aligned}
W= & \left\{\left(\text { sel }, \text { false }, \text { false }, d_{1}, \ldots, \text { false }, \text { false }, d_{n}\right.\right. \\
& \text { false } \left.\left., o_{1 \text { trdy }}, e_{1}, \ldots, \text { false }, o_{m \text { trdy }}, e_{m}, s_{0 z}\right) \in I\right\} .
\end{aligned}
$$

We check that $W$ satisfies properties (1)-(4) of the lemma.
(1) By the definition of $L$, for all $s \in W, g \in G$ it holds that $g$.irdy $\notin L(s)$.
(2) Observe that $H=O^{z}$ and $K=I^{z}$. Assume $H=\left\{o_{1}, \ldots, o_{m}\right\}$ and $K=\left\{i_{1}, \ldots, i_{n}\right\}$. We show that for all $\left(b_{1}, \ldots, b_{m}\right) \in \mathbb{B}^{m},\left(d_{1}, \ldots, d_{n}\right) \in$ ( $\prod_{1 \leq u \leq n} c\left(i_{u}\right)$ ) there is $s \in W$ such that

$$
\bigwedge_{1 \leq j \leq m}\left(b_{j} \Leftrightarrow o_{j} \cdot \mathbf{t r d y} \in L(s)\right) \wedge \bigwedge_{1 \leq u \leq n}\left(i_{u} \cdot \mathbf{d a t a}=d_{u} \in L(s)\right) .
$$

Fix arbitrary $\left(b_{1}, \ldots, b_{m}\right) \in \mathbb{B}^{m},\left(d_{1}, \ldots, d_{n}\right) \in\left(\prod_{1 \leq u \leq n} c\left(i_{u}\right)\right)$. Observe that there is

$$
\begin{aligned}
& \left\{\left(\text { sel, false, false, } d_{1}, \ldots, \text { false, false }, d_{n}\right.\right. \\
& \text { false, } \left.\left.b_{1}, e_{1}, \ldots, \text { false }, b_{m}, e_{m}, s_{0 z}\right)\right\} \in W
\end{aligned}
$$

let $s$ be such state. According to the definition of $L$, it holds that

$$
\bigwedge_{1 \leq j \leq m}\left(b_{j} \Leftrightarrow o_{j} \cdot \mathbf{t r d y} \in L(s)\right) \wedge \bigwedge_{1 \leq u \leq n}\left(i_{u} \cdot \mathbf{d a t a}=d_{u} \in L(s)\right) .
$$

(3) Proving (2), we showed a witness that $W \neq \emptyset$.
(4) Fix arbitrary $s \in W, s^{\prime} \in I$. Without loss of generality, let

$$
\begin{aligned}
s= & \left(\text { sel }, i_{1 \text { irdy }}, i_{1 \text { trdy }}, d_{1}, \ldots, i_{n \text { irdy }}, i_{n \text { trdy }}, d_{n}\right. \\
& \left.o_{1 \text { irdy }}, o_{1 \text { trdy }}, e_{1}, \ldots, o_{m \text { irdy }}, o_{m \text { trdy }}, e_{m}, s^{z}\right), \\
s^{\prime}= & \left(\mathbf{s e l}^{\prime}, i_{1 \text { irdy }}^{\prime}, i_{1 \text { trdy }}^{\prime}, d_{1}^{\prime}, \ldots, i_{n \text { irdy }}^{\prime}, i_{n \text { trdy }}^{\prime}, d_{n}^{\prime}\right. \\
& \left.o_{1 \text { irdy }}^{\prime}, o_{1 \text { trdy }}^{\prime}, e_{1}^{\prime}, \ldots, o_{m \text { irdy }}^{\prime}, o_{m \text { trdy }}^{\prime}, e_{m}^{\prime}, s^{\prime z}\right) .
\end{aligned}
$$

Observe that in $s$ there is no enabled transition since all trdy are false. Hence, we have sel $=\perp$. By the definition of $W$ we have $s^{z}=s^{\prime z}=s_{0}^{z}$. Hence, according to rule FSM1 from the definition of $\rightarrow$ of the FSM KS we conclude $\left(s, s^{\prime}\right) \in \rightarrow$.

- $\operatorname{type}(z)=$ function with $\operatorname{chan}(z$, in, 0$)=i, \operatorname{chan}(z$, out, 0$)=0$, and with a data transforming function $f$. By definition,

$$
\begin{aligned}
I= & \left\{\left(i_{\text {irdy }}, i_{\text {trdy }}, d, o_{\text {irdy }}, o_{\text {trdy }}, e\right) \in \mathbb{B} \times \mathbb{B} \times c(i) \times \mathbb{B} \times \mathbb{B} \times c(o) \mid\right. \\
& \left.i_{\text {trdy }}=o_{\text {trdy }}, o_{\text {irdy }}=i_{\text {irdy }}, e=f(d)\right\} .
\end{aligned}
$$

Let $W=\left\{\left(\right.\right.$ false $, i_{\text {trdy }}, d$, false $\left.\left., o_{\text {trdy }}, e\right) \in I\right\}$. We check that $W$ satisfies properties (1)-(4) of the lemma.
(1) By the definition of $L$, for all $g \in\{i, o\}, s \in W$ it holds that $g$.irdy $\notin L(s)$.
(2) Observe that $H=\{o\} K=\{i\}$. We show that for all $b \in \mathbb{B}, d \in c(i)$ there is $s \in W$ such that $b \Leftrightarrow o . \operatorname{trdy} \in L(s) \wedge i$. data $=d \in L(s)$. Fix arbitrary $b \in \mathbb{B}, d \in c(i)$. Observe that there is

$$
\text { (false, false, } d, \text { false }, b, e) \in W
$$

Using the definition of $L$ we have

$$
\begin{gathered}
b \Leftrightarrow o . \text { trdy } \in L((\text { false, false }, d, \text { false }, b, e)) \wedge \\
i . \text { data }=d \in L((\text { false, false }, d, \text { false }, b, e)) .
\end{gathered}
$$

(3) Proving (2), we showed a witness that $W \neq \emptyset$.
(4) Fix $s \in W, s^{\prime} \in I$. Without loss of generality, let

$$
\begin{aligned}
s & =\left(i_{\text {irdy }}, i_{\text {trdy }}, d^{\prime}, o_{\text {irdy }}, o_{\text {trdy }}, e\right) \\
s^{\prime} & =\left(i_{\text {irdy }}^{\prime}, i_{\text {trdy }}^{\prime}, d^{\prime}, o_{\text {irdy }}^{\prime}, o_{\text {trdy }}^{\prime}, e^{\prime}\right)
\end{aligned}
$$

According to rule Fun1 from the definition of $\rightarrow$ of the function KS we conclude $\left(s, s^{\prime}\right) \in \rightarrow$.

- $\operatorname{type}(z)=$ fork with $\operatorname{chan}(z$, in, 0$)=i, \operatorname{chan}(z$, out, 0$)=o, \operatorname{chan}(z$, out, 1$)=u$, and with data transforming functions $f$ and $f^{\prime}$. According to the definition of the fork KS,

$$
\begin{aligned}
I=\{ & \left\{\left(i_{\text {irdy }}, i_{\text {trdy }}, d, o_{\text {irdy }}, o_{\text {trdy }}, e, u_{\text {irdy }}, u_{\text {trdy }}, j\right)\right. \\
& \in \mathbb{B} \times \mathbb{B} \times c(i) \times \mathbb{B} \times \mathbb{B} \times c(o) \times \mathbb{B} \times \mathbb{B} \times c(u) \mid \\
& e=f(d), j=f^{\prime}(d), o_{\text {irdy }}=i_{\text {irdy }} \wedge u_{\text {trdy }}, u_{\text {irdy }}=i_{\text {irdy }} \wedge o_{\text {trdy }}, \\
& \left.i_{\text {trdy }}=o_{\text {trdy }} \wedge u_{\text {trdy }}\right\} .
\end{aligned}
$$

Let $W=\left\{\left(\right.\right.$ false $, i_{\text {trdy }}, d$, false, $o_{\text {trdy }}, e$, false $\left.\left., u_{\text {trdy }}, j\right) \in I\right\}$.
We check that $W$ satisfies properties (1)-(4) of the lemma.
(1) By the definition of $L$, for all $s \in W, g \in\{i, o, u\}$ it holds that $g$.irdy $\notin L(s)$.
(2) Observe that $H=\{0, u\}$ and $K=\{i\}$. We show that for all $\left(b_{1}, b_{2}\right) \in$ $\mathbb{B}^{2}, d \in c(i)$ there is $s \in W$ such that $b_{1} \Leftrightarrow o . \operatorname{trdy} \in L(s)$ and $b_{2} \Leftrightarrow$ $u . \operatorname{trdy} \in L(s)$ and $i$. data $=d \in L(s)$.

Fix arbitrary $\left(b_{1}, b_{2}\right) \in \mathbb{B}^{2}, d \in c(i)$. Observe that there is

$$
\text { (false, } \left.i_{\text {trdy }}, d \text {, false, } b_{1}, e, \text { false }, b_{2}, j\right) \in W
$$

let $s$ be such state. Using the definition of $L$ we have $b_{1} \Leftrightarrow o . \operatorname{trdy} \in L(s)$, and $b_{2} \Leftrightarrow u . \operatorname{trdy} \in L(s)$, and $i$. data $=d \in L(s)$.
(3) Proving (2), we showed a witness that $W \neq \emptyset$.
(4) Fix $s \in W, s^{\prime} \in I$. Without loss of generality, let

$$
\begin{aligned}
s & =\left(i_{\text {irdy }}, i_{\text {trdy }}, d, o_{\text {irdy }}, o_{\text {trdy }}, e, u_{\text {irdy }}, u_{\text {trdy }}, j\right) \\
s^{\prime} & =\left(i^{\text {irdy }}, i_{\text {trdy }}^{\prime}, d^{\prime}, o_{\text {irdy }}^{\prime}, o_{\text {trdy }}^{\prime}, e^{\prime}, u_{\text {irdy }}^{\prime}, u_{\text {trdy }}^{\prime}, j^{\prime}\right) .
\end{aligned}
$$

According to rule Frk1 from the definition of $\rightarrow$ of the fork KS we conclude $\left(s^{\prime}, s\right) \in \rightarrow$.

- $\operatorname{type}(z)=$ join with $\operatorname{chan}(z$, in, 0$)=i, \operatorname{chan}(z$, in, 1$)=j, \operatorname{chan}(z$, out, 0$)=o$, and a routing function $h$. According to the definition of the join KS, $I=\left\{\left(i_{\text {irdy }}, i_{\text {trdy }}, d, j_{\text {irdy }}, j_{\text {trdy }}, e, o_{\text {irdy }}, o_{\text {trdy }}, l\right) \in \mathbb{B} \times \mathbb{B} \times c(i) \times \mathbb{B} \times \mathbb{B} \times c(o) \times\right.$ $\mathbb{B} \times \mathbb{B} \times c(u) \mid l=h(d, e), i_{\text {trdy }}=j_{\text {irdy }} \wedge o_{\text {trdy }}, j_{\text {trdy }}=i_{\text {irdy }} \wedge o_{\text {trdy }}, o_{\text {irdy }}=$ $\left.i_{\text {irdy }} \wedge j_{\text {irdy }}\right\}$. Let $W=\left\{\left(\right.\right.$ false, $i_{\text {trdy }}, d$, false, $j_{\text {trdy }}, e$, false, $\left.\left.o_{\text {trdy }}, l\right) \in I\right\}$.
We check that $W$ satisfies properties (1)-(4) of the lemma.
(1) By the definition of $L$, for all $s \in W, g \in\{i, j, o\}$ it holds that $g$.irdy $\notin L(s)$.
(2) Observe that $H=\{o\}$ and $K=\{i, j\}$. We show that for all $b \in \mathbb{B},\left(d_{1}, d_{2}\right) \in$ $(c(i) \times c(j))$ there is $s \in W$ such that $b \Leftrightarrow o . \operatorname{trdy} \in L(s)$, and $i$.data $=d_{1} \in$ $L(s)$, and $j$.data $=d_{2} \in L(s)$.
Fix arbitrary $b \in \mathbb{B}$. Observe that there is

$$
\text { (false, } \left.i_{\text {trdy }}, d_{1}, \text { false, } j_{\text {trdy }}, d_{2}, \text { false }, b, l\right) \in W,
$$

let $s$ be such state. Using the definition of $L$ we have $b \Leftrightarrow o . \operatorname{trdy} \in L(s)$, and $i$.data $=d_{1} \in L(s)$, and $j$. data $=d_{2} \in L(s)$.
(3) Proving (2), we showed a witness that $W \neq \emptyset$.
(4) Fix $s \in W, s^{\prime} \in I$. Without loss of generality, let

$$
\begin{aligned}
& s=\left(i_{\text {irdy }}, i_{\text {trdy }}, d, j_{\text {irdy }}, j_{\text {trdy }}, e, o_{\text {irdy }}, o_{\text {trdy }}, l\right), \\
& s^{\prime}=\left(i_{\text {irdy }}, i^{\prime} \text { trdy }, d^{\prime}, j^{\prime} \text { irdy }{ }^{\prime} j^{\prime} \text { trdy }, e^{\prime}, o_{\text {irdy }}, o_{\text {trdy }}, l^{\prime}\right) \text {. }
\end{aligned}
$$

According to rule Jn1 from the definition of $\rightarrow$ of the join KS we conclude $\left(s, s^{\prime}\right) \in \rightarrow$.

- $\operatorname{type}(z)=\operatorname{switch}$ with $\operatorname{chan}(z$, in, 0$)=i, \operatorname{chan}(z$, out, 0$)=o, \operatorname{chan}(z$, out, 1$)=$ $u$, and a routing function $r$. According to the definition of the switch $\mathrm{KS}, I=\left\{\left(i_{\text {irdy }}, i_{\text {trdy }}, d, o_{\text {irdy }}, o_{\text {trdy }}, e, u_{\text {irdy }}, u_{\text {trdy }}, l\right) \in \mathbb{B} \times \mathbb{B} \times c(i) \times \mathbb{B} \times \mathbb{B} \times\right.$ $c(o) \times \mathbb{B} \times \mathbb{B} \times c(u) \mid(r(d) \Longrightarrow l=d),(\neg r(d) \Longrightarrow l=e),\left(o_{\text {irdy }}=i_{\text {irdy }} \wedge\right.$ $\left.r(d)), u_{\text {irdy }}=i_{\text {irdy }} \wedge \neg r(d), i_{\text {trdy }}=\left(o_{\text {irdy }} \wedge o_{\text {trdy }}\right) \vee\left(u_{\text {irdy }} \wedge u_{\text {trdy }}\right)\right\}$. Let $W=$ $\left\{\left(\right.\right.$ false $, i_{\text {trdy }}, d$, false, $o_{\text {trdy }}, e$, false, $\left.\left.u_{\text {trdy }}, l\right) \in I\right\}$.
We check that $W$ satisfies properties (1)-(4) of the lemma.
(1) By the definition of $L$, for all $s \in W, g \in\{i, o, u\}$ it holds that $g$.irdy $\notin L(s)$.
(2) Observe that $H=\{0, u\}$ and $K=\{i\}$. We show that for all $\left(b_{1}, b_{2}\right) \in$ $\mathbb{B}^{2}, d \in c(i)$ there is $s \in W$ such that $b_{1} \Leftrightarrow o . t r d y \in L(s)$, and $b_{2} \Leftrightarrow$ $u . \operatorname{trdy} \in L(s)$, and $i$. data $=d \in L(s)$.

Fix arbitrary $\left(b_{1}, b_{2}\right) \in \mathbb{B}^{2}, d \in c(i)$. Observe that there is

$$
\text { (false, } \left.i_{\text {trdy }}, d, \text { false, } b_{1}, e, \text { false }, b_{2}, l\right) \in W
$$

let $s$ be such state. Using the definition of $L$ we have $b_{1} \Leftrightarrow o . \operatorname{trdy} \in L(s)$ and $b_{2} \Leftrightarrow u . \operatorname{trdy} \in L(s)$, and $i$. data $=d \in L(s)$.
(3) Proving (2), we showed a witness that $W \neq \emptyset$.
(4) Fix arbitrary $s \in W, s^{\prime} \in I$. Without loss of generality, let

$$
\begin{aligned}
s & =\left(i_{\text {irdy }}, i_{\text {trdy }}, d^{\prime}, o_{\text {irdy }}, o_{\text {trdy }}, e, u_{\text {irdy }}, u_{\text {trdy }}, l\right) \\
s^{\prime} & =\left(i_{\text {irdy }}^{\prime}, i_{\text {trdy }}^{\prime}, d^{\prime}, o_{\text {irdy }}^{\prime}, o_{\text {trdy }}^{\prime}, e^{\prime}, u_{\text {irdy }}^{\prime}, u_{\text {trdy }}^{\prime}, l^{\prime}\right)
\end{aligned}
$$

According to rule Sw1 from the definition of $\rightarrow$ of the switch KS we conclude $\left(s, s^{\prime}\right) \in \rightarrow$.

- $\operatorname{type}(z)=$ merge with $\operatorname{chan}(z$, in, 0$)=i, \operatorname{chan}(z$, in, 1$)=j, \operatorname{chan}(z$, out, 0$)=0$. According to the definition of the merge KS,

$$
\begin{aligned}
& I=\left\{\left(u, i_{\text {irdy }}, i_{\text {trdy }}, d, j_{\text {irdy }}, j_{\text {trdy }}, e, o_{\text {irdy }}, o_{\text {trdy }}, l\right)\right. \\
& \in \mathbb{B} \times \mathbb{B} \times \mathbb{B} \times c(i) \times \mathbb{B} \times \mathbb{B} \times c(j) \times \mathbb{B} \times \mathbb{B} \times c(o) \mid \\
& o_{\text {irdy }}=\left(u \wedge i_{\text {irdy }}\right) \vee\left(\neg u \wedge j_{\text {irdy }}\right), i_{\text {trdy }}=u \wedge o_{\text {trdy }} \wedge i_{\text {irdy }}, \\
&\left.j_{\text {trdy }}=\neg u \wedge o_{\text {trdy }} \wedge j_{\text {irdy }}, u \wedge i_{\text {irdy }} \Rightarrow l=d, \neg u \wedge j_{\text {irdy }} \Rightarrow l=e\right\} .
\end{aligned}
$$

Let $W=\left\{\left(u\right.\right.$, false $, i_{\text {trdy }}, d$, false, $j_{\text {trdy }}, e$, false, $\left.\left.o_{\text {trdy }}, l\right) \in I\right\}$.
We check that $W$ satisfies properties (1)-(4) of the lemma.
(1) By the definition of $L$, for all $g \in\{i, j, o\}, s \in W$ it holds that $g$.irdy $\notin L(s)$.
(2) Observe that $H=\{0\}$ and $K=\{i, j\}$. We show that for all $b \in \mathbb{B},\left(d_{1}, d_{2}\right) \in$ $(c(i) \times c(j))$ there is $s \in W$ such that $b \Leftrightarrow o . \operatorname{trdy} \in L(s)$, and $i$. data $=d_{1} \in$ $L(s)$, and $j$.data $=d_{2} \in L(s)$. Fix arbitrary $b \in \mathbb{B},\left(d_{1}, d_{2}\right) \in(c(i) \times c(j))$. Observe that there is

$$
\left(u, \text { false }, i_{\text {trdy }}, d_{1}, \text { false }, j_{\text {trdy }}, d_{2}, \text { false }, b, l\right) \in W,
$$

let $s$ be such state. Using the definition of $L$ we have $b \Leftrightarrow o . \operatorname{trdy} \in L(s)$, and $i$.data $=d_{1} \in L(s)$, and $j$.data $=d_{2} \in L(s)$.
(3) Proving (2), we showed a witness that $W \neq \emptyset$.
(4) Fix $s \in W, s^{\prime} \in I$. Without loss of generality, let

$$
\begin{aligned}
& s=\left(u, i_{\text {irdy }}, i_{\text {trdy }}, d, j_{\text {irdy }}, j_{\text {trdy }}, e, o_{\text {irdy }}, o_{\text {trdy }}, l\right) \text {, } \\
& s^{\prime}=\left(u^{\prime}, i^{\prime}{ }_{\text {irdy }}, i^{\prime}{ }_{\text {trdy }}, d^{\prime}, j^{\prime}{ }_{\text {irdy }}, j^{\prime} \text { trdy }, e^{\prime}, o^{\prime} \text { irdy }, o_{\text {trdy }}^{\prime}, l^{\prime}\right) .
\end{aligned}
$$

According to rule Mrg1 from the definition of $\rightarrow$ of the merge KS we conclude $\left(s, s^{\prime}\right) \in \rightarrow$.

- Inductive step. Assume $N$ is such that $|P|>1$. From Lemma 2.13 it holds that $N$ can be split into two valid compatible networks $N^{\prime}$ and $N^{\prime \prime}$, such that $N=N^{\prime}$ ॥ $N^{\prime \prime}$. We fix such $N^{\prime}$ and $N^{\prime \prime}$. Assume that

$$
\begin{aligned}
N^{\prime} & =\left(P^{\prime}, G^{\prime}, C^{\prime}, c^{\prime}, \text { chan', type }\right), \text { and } \\
N^{\prime \prime} & =\left(P^{\prime \prime}, G^{\prime \prime}, C^{\prime \prime}, c^{\prime \prime}, \text { chan }{ }^{\prime \prime} \text {, type }\right) .
\end{aligned}
$$

Let

$$
\begin{aligned}
\mathrm{KS}\left(N^{\prime}\right) & =\left(S^{\prime}, I^{\prime}, \rightarrow^{\prime}, \mathrm{AP}^{\prime}, L^{\prime}\right), \text { and } \\
\operatorname{KS}\left(N^{\prime \prime}\right) & =\left(S^{\prime \prime}, I^{\prime \prime}, \rightarrow^{\prime \prime}, \mathrm{AP}^{\prime \prime}, L^{\prime \prime}\right)
\end{aligned}
$$

be two KSs representing $N^{\prime}$ and $N^{\prime \prime}$ respectively. Then, by the definition of $\operatorname{KS}(N)$, we have $\operatorname{KS}(N)=\operatorname{KS}\left(N^{\prime}\right) \| \operatorname{KS}\left(N^{\prime \prime}\right)$. The induction hypothesis is that there are $W^{\prime} \subseteq I^{\prime}$ and $W^{\prime \prime} \subseteq I^{\prime \prime}$ such that the following properties hold.

- For all $s^{\prime} \in W^{\prime}, g^{\prime} \in G^{\prime}$ it holds that $g^{\prime} . \operatorname{irdy} \notin L^{\prime}\left(s^{\prime}\right)$ and for all $s^{\prime \prime} \in W^{\prime \prime}, g^{\prime \prime} \in$ $G^{\prime \prime}$ it holds that $g^{\prime \prime}$.irdy $\notin L^{\prime \prime}\left(s^{\prime \prime}\right)$.
- Let $H^{\prime}=\left\{g \in G^{\prime} \mid \forall p \in P^{\prime}, n \in \mathbb{N} . \operatorname{chan}(p\right.$, in,$\left.n) \neq g\right\}, K^{\prime}=\left\{g \in G^{\prime} \mid \forall p \in\right.$ $P^{\prime}, n \in \mathbb{N} . c h a n(p$, out, $\left.n) \neq g\right\}$. Then the following holds.
* If $H^{\prime} \neq \emptyset \wedge K^{\prime}=\emptyset$, assume $H^{\prime}=\left\{g_{1}^{\prime}, \ldots, g_{l^{\prime}}^{\prime}\right\}$. Then,

$$
\forall\left(b_{1}, \ldots, b_{l^{\prime}}\right) \in \mathbb{B}^{l^{\prime}} .\left(\exists s \in W^{\prime} . \bigwedge_{1 \leq j \leq l^{\prime}}\left(b_{j} \Leftrightarrow g_{j}^{\prime} \cdot \operatorname{trdy} \in L^{\prime}(s)\right)\right) .
$$

* If $H^{\prime}=\emptyset \wedge K^{\prime} \neq \emptyset$, assume $K^{\prime}=\left\{x_{1}^{\prime}, \ldots, x_{h^{\prime}}^{\prime}\right\}$. Then,

$$
\begin{aligned}
\forall\left(d_{1}, \ldots, d_{h^{\prime}}\right) \in & \left(\prod_{1 \leq u \leq h^{\prime}} c^{\prime}\left(x_{u}\right)\right) . \\
& \left(\exists s \in W^{\prime} \cdot \bigwedge_{1 \leq u \leq h}\left(x_{u}^{\prime} \cdot \text { data }=d_{u} \in L^{\prime}(s)\right)\right) .
\end{aligned}
$$

* If $H^{\prime} \neq \emptyset \wedge K^{\prime} \neq \emptyset$, assume $H^{\prime}=\left\{g_{1}^{\prime}, \ldots, g_{l^{\prime}}^{\prime}\right\}$ and $K^{\prime}=\left\{x_{1}^{\prime}, \ldots, x_{h^{\prime}}^{\prime}\right\}$. Then,

$$
\begin{aligned}
\forall\left(b_{1}, \ldots, b_{l^{\prime}}\right) \in \mathbb{B}^{l},\left(d_{1}, \ldots, d_{h^{\prime}}\right) \in & \left(\prod_{1 \leq u \leq h^{\prime}} c^{\prime}\left(x_{u}\right)\right) . \\
\left(\exists s \in W^{\prime} \cdot\right. & \bigwedge_{1 \leq j \leq l^{\prime}}\left(b_{j} \Leftrightarrow g_{j}^{\prime} \cdot \operatorname{trdy} \in L^{\prime}(s)\right) \wedge \\
& \left.\bigwedge_{1 \leq u \leq h^{\prime}}\left(x_{u}^{\prime} \cdot \text { data }=d_{u} \in L^{\prime}(s)\right)\right) .
\end{aligned}
$$

Similarly, let $H^{\prime \prime}=\left\{g \in G^{\prime \prime} \mid \forall p \in P^{\prime \prime}, n \in \mathbb{N} . c h a n(p\right.$, in,$\left.n) \neq g\right\}, K^{\prime \prime}=\{g \in$ $G^{\prime \prime} \mid \forall p \in P^{\prime}, n \in \mathbb{N}$.chan( $p$, out, $\left.\left.n\right) \neq g\right\}$. Then the following holds.

* If $H^{\prime \prime} \neq \emptyset \wedge K^{\prime \prime}=\emptyset$, assume $H^{\prime \prime}=\left\{g_{1}^{\prime \prime}, \ldots, g_{l^{\prime \prime}}^{\prime \prime}\right\}$. Then,

$$
\forall\left(b_{1}, \ldots, b_{l^{\prime \prime}}\right) \in \mathbb{B}^{l^{\prime \prime}} .\left(\exists s \in W^{\prime \prime} . \bigwedge_{1 \leq j \leq l^{\prime}}\left(b_{j} \Leftrightarrow g_{j}^{\prime \prime} \cdot \operatorname{trd} \mathbf{y} \in L^{\prime \prime}(s)\right)\right)
$$

* If $H^{\prime \prime}=\emptyset \wedge K^{\prime \prime} \neq \emptyset$, assume $K^{\prime \prime}=\left\{x_{1}^{\prime \prime}, \ldots, x_{h^{\prime \prime}}^{\prime \prime}\right\}$. Then,

$$
\begin{aligned}
\forall\left(d_{1}, \ldots, d_{h^{\prime \prime}}\right) \in & \left(\prod_{1 \leq u \leq h^{\prime \prime}} c^{\prime \prime}\left(x_{u}^{\prime \prime}\right)\right) \\
& \left(\exists s \in W^{\prime \prime} \cdot \bigwedge_{1 \leq u \leq h^{\prime \prime}}\left(x_{u}^{\prime \prime} \cdot \text { data }=d_{u} \in L^{\prime \prime}(s)\right)\right) .
\end{aligned}
$$

* If $H^{\prime \prime} \neq \emptyset \wedge K^{\prime \prime} \neq \emptyset$, assume $H^{\prime \prime}=\left\{g_{1}^{\prime \prime}, \ldots, g_{l^{\prime \prime}}^{\prime \prime}\right\}$ and $K^{\prime \prime}=\left\{x_{1}^{\prime \prime}, \ldots, x_{h^{\prime \prime}}^{\prime \prime}\right\}$. Then,

$$
\begin{aligned}
\forall\left(b_{1}, \ldots, b_{l^{\prime \prime}}\right) \in \mathbb{B}^{l^{\prime \prime}},\left(d_{1}, \ldots, d_{h^{\prime \prime}}\right) \in\left(\prod_{1 \leq u \leq h^{\prime \prime}} c^{\prime \prime}\left(x_{u}^{\prime \prime}\right)\right) . \\
\left(\exists s \in W^{\prime \prime} \cdot \bigwedge_{1 \leq j \leq l^{\prime \prime}}\left(b_{j} \Leftrightarrow g_{j}^{\prime \prime} \cdot \mathbf{t r d y} \in L^{\prime \prime}(s)\right) \wedge\right. \\
\left.\bigwedge_{1 \leq u \leq h^{\prime \prime}}\left(x_{u}^{\prime \prime} \cdot \text { data }=d_{u} \in L^{\prime \prime}(s)\right)\right) .
\end{aligned}
$$

- $W^{\prime} \neq \emptyset$ and $W^{\prime \prime} \neq \emptyset$.
- For all $s^{\prime} \in W^{\prime}, s \in I^{\prime}$ we have $\left(s^{\prime}, s\right) \in \rightarrow^{\prime}$. Similarly, for all $s^{\prime} \in W^{\prime \prime}, s \in I^{\prime \prime}$ we have $\left(s^{\prime}, s\right) \in \rightarrow{ }^{\prime \prime}$.

We construct set $W$ and show that $W \subseteq I$ and properties (1)-(4) of the lemma hold for $W$.

Let $W$ be as follows.

$$
\begin{aligned}
& W=\left\{\left(s^{\prime}, s^{\prime \prime}\right) \in W^{\prime}\right. \times W^{\prime \prime} \mid \\
& \forall g \in H^{\prime} \cap K^{\prime \prime} \\
& .\left(g . t r d y \in L^{\prime \prime}\left(s^{\prime \prime}\right) \wedge g . \text { data }=d \in L^{\prime \prime}\left(s^{\prime \prime}\right)\right. \\
&\left.\Rightarrow g . \operatorname{trdy} \in L^{\prime}\left(s^{\prime}\right) \wedge g \cdot \text { data }=d \in L^{\prime}\left(s^{\prime}\right)\right), \\
& \forall g \in H^{\prime \prime} \cap K^{\prime} .\left(g . \text { trdy } \in L^{\prime}\left(s^{\prime}\right) \wedge g \cdot d a t a=d \in L^{\prime}\left(s^{\prime}\right)\right. \\
&\left.\left.\Rightarrow g . t r d y \in L^{\prime \prime}\left(s^{\prime \prime}\right) \wedge g . \text { data }=d \in L^{\prime \prime}\left(s^{\prime \prime}\right)\right)\right\} .
\end{aligned}
$$

Fix arbitrary $\left(s^{\prime}, s^{\prime \prime}\right) \in W$ and show that $\left(s^{\prime}, s^{\prime \prime}\right) \in I$. By the definition of the parallel composition of two KSs, we have $I=\left(I^{\prime} \times I^{\prime \prime}\right) \cap S$. Hence, showing that $\left(s^{\prime}, s^{\prime \prime}\right) \in I$ is equivalent to showing that $\left(s^{\prime}, s^{\prime \prime}\right) \in\left(I^{\prime} \times I^{\prime \prime}\right)$ and $\left(s^{\prime}, s^{\prime \prime}\right) \in S$. From the induction hypothesis, we know that $W^{\prime}$ is a subset of $I^{\prime}$ and $W^{\prime \prime}$ is a subset of $I^{\prime \prime}$, hence we conclude $\left(s^{\prime}, s^{\prime \prime}\right) \in\left(I^{\prime} \times I^{\prime \prime}\right)$.
Now we show that $\left(s^{\prime}, s^{\prime \prime}\right) \in S$. Since $\operatorname{KS}(N)=\operatorname{KS}\left(N^{\prime}\right) \| \operatorname{KS}\left(N^{\prime \prime}\right)$, by the definition of the parallel composition of two KSs we have $S=\left\{\left(q^{\prime}, q^{\prime \prime}\right) \mid q^{\prime} \in\right.$
$\left.S^{\prime}, q^{\prime \prime} \in S^{\prime \prime},\left(L^{\prime}\left(q^{\prime}\right) \cap A P^{\prime \prime}\right)=\left(L^{\prime \prime}\left(q^{\prime \prime}\right) \cap A P^{\prime}\right)\right\}$. We show that $\left(L^{\prime}\left(q^{\prime}\right) \cap A P^{\prime \prime}\right)=$ $\left(L^{\prime \prime}\left(q^{\prime \prime}\right) \cap A P^{\prime}\right)$.

Fix arbitrary atomic proposition $a$ such that $a \in\left(L^{\prime}\left(s^{\prime}\right) \cap A P^{\prime \prime}\right)$. Since $N^{\prime}$ and $N^{\prime \prime}$ are valid and compatible, $a$ can be either of the form $g$. irdy, $g . \operatorname{trdy}$, or $g$. data $=d$ for some $g \in G^{\prime} \cap G^{\prime \prime}$ and $d \in c(g)$. From the induction hypothesis we know that atomic propositions of the form $g$.irdy are neither in $L\left(s^{\prime}\right)$ nor in $L^{\prime \prime}\left(s^{\prime \prime}\right)$. Thus $a$ can be of the form $g$.irdy or $g . d a t a=d$. By the definition of $W$ we conclude that if $a \in\left(L^{\prime}\left(s^{\prime}\right) \cap A P^{\prime \prime}\right)$ then $a \in\left(L^{\prime \prime}\left(s^{\prime \prime}\right) \cap A P^{\prime}\right)$. Hence, $\left(s^{\prime}, s^{\prime \prime}\right) \in S$. We showed that $\left(s^{\prime}, s^{\prime \prime}\right) \in\left(I^{\prime} \times I^{\prime \prime}\right)$ and $\left(s^{\prime}, s^{\prime \prime}\right) \in S$ and hence we conclude that $\left(s^{\prime}, s^{\prime \prime}\right) \in I$.

Now we show that conditions (1)-(4) of the lemma hold for $W$.
(1) We show that for all $g \in G,\left(s^{\prime}, s^{\prime \prime}\right) \in W$ it holds that $g$.irdy $\notin L\left(s^{\prime}, s^{\prime \prime}\right)$. From the definition of the parallel composition of two KSs, we have $L\left(\left(s^{\prime}, s^{\prime \prime}\right)\right)=$ $L^{\prime}\left(s^{\prime}\right) \cup L^{\prime \prime}\left(s^{\prime \prime}\right)$. By the induction hypothesis we have that for all $g \in G^{\prime}$, $g$.irdy $\notin L^{\prime}\left(s^{\prime}\right)$ and for all $g \in G^{\prime \prime}$, $g$.irdy $\notin L^{\prime \prime}\left(s^{\prime \prime}\right)$. By the definition of the parallel composition of two xMAS networks we have $G=G^{\prime} \cup G^{\prime \prime}$ hence for all $g \in G$ it holds that $g$.irdy $\notin L\left(s^{\prime}, s^{\prime \prime}\right)$.
(2) Let $H=\{g \in G \mid \forall p \in P, n \in \mathbb{N} . \operatorname{chan}(p$, in, $n) \neq g\}$ and $K=\{g \in G \mid \forall p \in$ $P, n \in \mathbb{N}$.chan $(p$, out, $n) \neq g\}$. We distinguish three cases.

* Assume $H \neq \emptyset \wedge K=\emptyset$. Assume $H=\left\{g_{1}, \ldots, g_{l}\right\}$ and fix $\left(b_{1}, \ldots, b_{l}\right) \in$ $\mathbb{B}^{l}$. We construct ( $s^{\prime}, s^{\prime \prime}$ ) and show ( $\left.s^{\prime}, s^{\prime \prime}\right) \in W$ and

$$
\bigwedge_{1 \leq j \leq l}\left(b_{j} \cdot \operatorname{trdy} \Leftrightarrow g_{j} \cdot \mathbf{t r d y} \in L(s)\right) .
$$

From the induction hypothesis, we know that there are $s^{\prime} \in W^{\prime}, s^{\prime \prime} \in$ $W^{\prime \prime}$ such that the following holds.
$\left(*_{1}\right) \forall g \in H^{\prime} \cap K^{\prime \prime} .\left(g . \operatorname{trdy} \in L^{\prime \prime}\left(s^{\prime \prime}\right) \wedge g\right.$. data $=d \in L^{\prime \prime}\left(s^{\prime \prime}\right) \Rightarrow g$. trdy $\in$ $L^{\prime}\left(s^{\prime}\right) \wedge g$.data $\left.=d \in L^{\prime}\left(s^{\prime}\right)\right)$. The intuition here is that, by the induction hypothesis, for all channels $g$ that are in both $H^{\prime}$ and $K^{\prime \prime}$, we can find a pair of states $s^{\prime} \in W^{\prime}$ and $s^{\prime \prime} \in W^{\prime \prime}$, such that all the labels in $L^{\prime}\left(s^{\prime}\right)$ and $L^{\prime \prime}\left(s^{\prime \prime}\right)$ that correspond to $g$ match.
$\left(*_{2}\right) \forall g \in H^{\prime \prime} \cap K^{\prime} .\left(g . t r d y \in L^{\prime}\left(s^{\prime}\right) \wedge g\right.$. data $=d \in L^{\prime}\left(s^{\prime}\right) \Rightarrow g . t r d y \in$ $L^{\prime \prime}\left(s^{\prime \prime}\right) \wedge g$.data $\left.=d \in L^{\prime \prime}\left(s^{\prime \prime}\right)\right)$. Here, the intuition is the same as for ( $*_{1}$ )
$\left(*_{3}\right) \bigwedge_{1 \leq j \leq l^{\prime}}\left(g_{j}^{\prime} \in\left(H^{\prime} \backslash K^{\prime \prime}\right) \Rightarrow\left(b_{j}^{\prime} \Leftrightarrow g_{j}^{\prime} \cdot\right.\right.$ trdy $\left.\in L^{\prime}\left(s^{\prime}\right)\right)$, and
$\left(*_{4}\right) \bigwedge_{1 \leq u \leq l^{\prime \prime}}\left(g_{j}^{\prime \prime} \in\left(H^{\prime \prime} \backslash K^{\prime}\right) \Rightarrow\left(b_{j}^{\prime \prime} \Leftrightarrow g_{j}^{\prime \prime} \cdot \operatorname{trdy} \in L^{\prime \prime}\left(s^{\prime \prime}\right)\right)\right.$.
Let $s^{\prime} \in W^{\prime}$ and $s^{\prime \prime} \in W^{\prime \prime}$ be such states. From ( $*_{1}$ ) and ( $*_{2}$ ) it immediately follows that $\left(s^{\prime}, s^{\prime \prime}\right) \in W$.

Observe that for all $g \in H$ it holds that $g \in\left(H^{\prime} \backslash K^{\prime \prime}\right)$ or $g \in\left(H^{\prime \prime} \backslash K^{\prime}\right)$.

Hence, from ( $*_{3}$ ) and ( $*_{4}$ ) we have

$$
\bigwedge_{1 \leq j \leq l}\left(\left(b_{j} \Leftrightarrow g_{j} \cdot \mathbf{t r d y} \in L^{\prime}\left(s^{\prime}\right)\right) \vee\left(b_{j} \Leftrightarrow g_{j} \cdot \mathbf{t r d y} \in L^{\prime \prime}\left(s^{\prime \prime}\right)\right)\right)
$$

From the definition of $L$ we have $L\left(s^{\prime}, s^{\prime \prime}\right)=L^{\prime}\left(s^{\prime}\right) \cup L^{\prime \prime}\left(s^{\prime \prime}\right)$. Hence, we conclude

$$
\bigwedge_{1 \leq j \leq l}\left(\left(b_{j} \Leftrightarrow g_{j} \cdot \mathbf{t r d y} \in L\left(\left(s^{\prime}, s^{\prime \prime}\right)\right)\right)\right)
$$

* Assume $H=\emptyset \wedge K \neq \emptyset$. Assume $K=\left\{x_{1}, \ldots, x_{h}\right\}$ and fix $\left(d_{1}, \ldots, d_{h}\right) \in$ $\left(\prod_{1 \leq u \leq h}\left(c\left(x_{u}\right)\right)\right)$. From the induction hypothesis, we know that there are $s^{\prime} \in W^{\prime}, s^{\prime \prime} \in W^{\prime \prime}$, such that the following holds.
$\left(*_{1}\right) \forall g \in H^{\prime} \cap K^{\prime \prime} .\left(g . \operatorname{trdy} \in L^{\prime \prime}\left(s^{\prime \prime}\right) \wedge g\right.$.data $=d \in L^{\prime \prime}\left(s^{\prime \prime}\right) \Rightarrow g . \operatorname{trdy} \in$ $L^{\prime}\left(s^{\prime}\right) \wedge g$. data $\left.=d \in L^{\prime}\left(s^{\prime}\right)\right)$, and
$\left(*_{2}\right) \forall g \in H^{\prime \prime} \cap K^{\prime} .\left(g . t r d y \in L^{\prime}\left(s^{\prime}\right) \wedge g\right.$. data $=d \in L^{\prime}\left(s^{\prime}\right) \Rightarrow g . t r d y \in$ $L^{\prime \prime}\left(s^{\prime \prime}\right) \wedge g \cdot$ data $\left.=d \in L^{\prime \prime}\left(s^{\prime \prime}\right)\right)$.
$\left(*_{3}\right) \bigwedge_{1 \leq j \leq h^{\prime}}\left(x_{j}^{\prime} \in\left(H^{\prime} \backslash K^{\prime \prime}\right) \Rightarrow\left(x_{j}^{\prime} \cdot\right.\right.$ data $\left.=d_{j}^{\prime} \in L^{\prime}\left(s^{\prime}\right)\right)$, and
$\left(*_{4}\right) \bigwedge_{1 \leq u \leq h^{\prime \prime}}\left(x_{j}^{\prime \prime} \in\left(H^{\prime \prime} \backslash K^{\prime}\right) \Rightarrow\left(x_{j}^{\prime \prime} \cdot\right.\right.$ data $\left.=d_{j}^{\prime \prime} \in L^{\prime \prime}\left(s^{\prime \prime}\right)\right)$.
Let $s^{\prime} \in W^{\prime}$ and $s^{\prime \prime} \in W^{\prime \prime}$ be such states. From ( $*_{1}$ ) and ( $*_{2}$ ) it immediately follows that $\left(s^{\prime}, s^{\prime \prime}\right) \in W$.

Observe that for all $x \in K$ it holds that $x \in\left(H^{\prime} \backslash K^{\prime \prime}\right)$ or $x \in\left(H^{\prime \prime} \backslash K^{\prime}\right)$. Hence, from ( $*_{3}$ ) and ( $*_{4}$ ) we have

$$
\bigwedge_{1 \leq u \leq h}\left(\left(x_{u} \cdot \text { data }=d_{u} \in L^{\prime}\left(s^{\prime}\right)\right) \vee\left(x_{u} \cdot \text { data }=d_{u} \in L^{\prime \prime}\left(s^{\prime \prime}\right)\right)\right)
$$

From the definition of $L$ we have $L\left(s^{\prime}, s^{\prime \prime}\right)=L^{\prime}\left(s^{\prime}\right) \cup L^{\prime \prime}\left(s^{\prime \prime}\right)$. Hence, we conclude

$$
\bigwedge_{1 \leq u \leq h}\left(\left(x_{j} \cdot \text { data }=d_{j} \in L\left(\left(s^{\prime}, s^{\prime \prime}\right)\right)\right)\right)
$$

* Assume $H \neq \emptyset \wedge K \neq \emptyset$. Assume $H=\left\{g_{1}, \ldots, g_{l}\right\}, K=\left\{x_{1}, \ldots, x_{h}\right\}$, and fix $\left(b_{1}, \ldots, b_{l}\right) \in \mathbb{B}^{l},\left(d_{1}, \ldots, d_{h}\right) \in\left(\prod_{1 \leq u \leq h}\left(c\left(x_{u}\right)\right)\right)$. From the induction hypothesis, we know that there are $s^{\prime} \in W^{\prime}, s^{\prime \prime} \in W^{\prime \prime}$ such that the following holds.
$\left(*_{1}\right) \forall g \in H^{\prime} \cap K^{\prime \prime} .\left(g . \operatorname{trdy} \in L^{\prime \prime}\left(s^{\prime \prime}\right) \wedge g\right.$. data $=d \in L^{\prime \prime}\left(s^{\prime \prime}\right) \Rightarrow g . \operatorname{trdy} \in$ $L^{\prime}\left(s^{\prime}\right) \wedge g$. data $\left.=d \in L^{\prime}\left(s^{\prime}\right)\right)$, and
$\left(*_{2}\right) \forall g \in H^{\prime \prime} \cap K^{\prime} .\left(g\right.$. trdy $\in L^{\prime}\left(s^{\prime}\right) \wedge g$. data $=d \in L^{\prime}\left(s^{\prime}\right) \Rightarrow g . t r d y \in$ $L^{\prime \prime}\left(s^{\prime \prime}\right) \wedge g$. data $\left.=d \in L^{\prime \prime}\left(s^{\prime \prime}\right)\right)$.
$\left(*_{3}\right) \bigwedge_{1 \leq j \leq l^{\prime}}\left(g_{j}^{\prime} \in\left(H^{\prime} \backslash K^{\prime \prime}\right) \Rightarrow\left(b_{j}^{\prime} \Leftrightarrow g_{j}^{\prime} \cdot \operatorname{trdy} \in L^{\prime}\left(s^{\prime}\right)\right)\right.$, and
$\left(*_{4}\right) \bigwedge_{1 \leq j \leq l^{\prime \prime}}\left(g_{j}^{\prime \prime} \in\left(H^{\prime \prime} \backslash K^{\prime}\right) \Rightarrow\left(b_{j}^{\prime \prime} \Leftrightarrow g_{j}^{\prime \prime} \cdot \operatorname{trdy} \in L^{\prime \prime}\left(s^{\prime \prime}\right)\right)\right.$, and
$\left(*_{5}\right) \bigwedge_{1 \leq u \leq h^{\prime}}\left(x_{u}^{\prime} \in\left(H^{\prime} \backslash K^{\prime \prime}\right) \Rightarrow\left(x_{u}^{\prime} \cdot\right.\right.$ data $\left.=d_{u}^{\prime} \in L^{\prime}\left(s^{\prime}\right)\right)$, and
(*6) $\wedge_{1 \leq u \leq h^{\prime \prime}}\left(x_{u}^{\prime \prime} \in\left(H^{\prime \prime} \backslash K^{\prime}\right) \Rightarrow\left(x_{u}^{\prime \prime} \cdot\right.\right.$ data $\left.=d_{u}^{\prime \prime} \in L^{\prime \prime}\left(s^{\prime \prime}\right)\right)$.
Let $s^{\prime} \in W^{\prime}$ and $s^{\prime \prime} \in W^{\prime \prime}$ be such states. From (*1 $)$ and (*2) it immediately follows that $\left(s^{\prime}, s^{\prime \prime}\right) \in W$.
Observe that for all $g \in(H \cup K)$ it holds that $g \in\left(H^{\prime} \backslash K^{\prime \prime}\right)$ or $g \in$ $\left(H^{\prime \prime} \backslash K^{\prime}\right)$. Hence, from $\left(*_{3}\right)-\left(*_{6}\right)$ we have

$$
\begin{aligned}
& \bigwedge_{1 \leq j \leq l}\left(\left(b_{j} \Leftrightarrow g_{j} \cdot \mathbf{t r d y} \in L^{\prime}\left(s^{\prime}\right)\right) \vee\left(b_{j} \Leftrightarrow g_{j} \cdot \mathbf{t r d y} \in L^{\prime \prime}\left(s^{\prime \prime}\right)\right)\right) \wedge \\
& \bigwedge_{1 \leq u \leq h}\left(\left(x_{u} \cdot \mathbf{d a t a}=d_{u} \in L^{\prime}\left(s^{\prime}\right)\right) \vee\left(x_{u} \cdot \mathbf{d a t a}=d_{u} \in L^{\prime \prime}\left(s^{\prime \prime}\right)\right)\right) .
\end{aligned}
$$

From the definition of $L$ we have $L\left(\left(s^{\prime}, s^{\prime \prime}\right)\right)=L^{\prime}\left(s^{\prime}\right) \cup L^{\prime \prime}\left(s^{\prime \prime}\right)$. Hence, we conclude

$$
\bigwedge_{1 \leq j \leq l}\left(b_{j} \Leftrightarrow g_{j} \cdot \operatorname{trdy} \in L(s)\right) \wedge \bigwedge_{1 \leq u \leq h}\left(x_{u} \cdot \text { data }=d_{u} \in L(s)\right) .
$$

(3) For cases when $H \neq \emptyset \wedge K=\emptyset, H=\emptyset \wedge K \neq \emptyset$, and $H \neq \emptyset \wedge K \neq \emptyset$, we already showed witnesses that $W \neq \emptyset$. Now we assume $H=\emptyset \wedge K=\emptyset$. From the induction hypothesis we know that $W^{\prime} \neq \emptyset, W^{\prime \prime} \neq \emptyset$, and there are $s^{\prime} \in W^{\prime}, s^{\prime \prime} \in W^{\prime \prime}$, such that:
$\left(*_{1}\right) \forall g \in H^{\prime} \cap K^{\prime \prime} .\left(g . \operatorname{trdy} \in L^{\prime \prime}\left(s^{\prime \prime}\right) \wedge g\right.$.data $=d \in L^{\prime \prime}\left(s^{\prime \prime}\right) \Rightarrow g . \operatorname{trdy} \in$ $L^{\prime}\left(s^{\prime}\right) \wedge g \cdot$ data $\left.=d \in L^{\prime}\left(s^{\prime}\right)\right)$, and
(*2) $\forall g \in H^{\prime \prime} \cap K^{\prime} .\left(g . \operatorname{trdy} \in L^{\prime}\left(s^{\prime}\right) \wedge g\right.$. data $=d \in L^{\prime}\left(s^{\prime}\right) \Rightarrow g . \operatorname{trdy} \in L^{\prime \prime}\left(s^{\prime \prime}\right) \wedge$ g.data $\left.=d \in L^{\prime \prime}\left(s^{\prime \prime}\right)\right)$.

From the induction hypothesis it immediately follows that $\left(s^{\prime}, s^{\prime \prime}\right) \in W$. Hence $\left(s^{\prime}, s^{\prime \prime}\right)$ witnesses that $W \neq \emptyset$ when $H=\emptyset \wedge K=\emptyset$.
(4) We need to show that for all $s \in W, s^{\prime} \in I$ it holds that $\left(\left(s^{\prime}, s^{\prime \prime}\right)\right) \in \rightarrow$. Fix arbitrary $\left(p^{\prime}, p^{\prime \prime}\right) \in W,\left(q^{\prime}, q^{\prime \prime}\right) \in I$. By the induction hypothesis we have that for all $p^{\prime} \in W^{\prime}, q^{\prime} \in I^{\prime}\left(p^{\prime}, q^{\prime}\right) \in \rightarrow^{\prime}$ and for all $p^{\prime \prime} \in W^{\prime \prime}, q^{\prime \prime} \in I^{\prime \prime}$ it holds that $\left(p^{\prime \prime}, q^{\prime \prime}\right) \in \rightarrow^{\prime \prime}$. Hence, using the definition of $\rightarrow$ of the parallel composition of two KSs we conclude that $\left(p^{\prime}, p^{\prime \prime}\right) \rightarrow\left(q^{\prime}, q^{\prime \prime}\right)$.

Theorem 4.16. Given a valid $x M A S$ network $N=(P, G, C, c$, chan, type $)$, let $K S(N)=$ $(S, I, \rightarrow, A P, L)$ be the Kripke Structure representing $N$. Then, for all $s \in I$ there is $s^{\prime} \in I$ such that $\left(s^{\prime}, s\right) \in \rightarrow$.

Proof. Let $W$ be a subset of $I$ such that the following holds.
(1) $\forall s \in W, g \in G .(g . i r d y \notin L(s))$.
(2) Let $H=\{g \in G \mid \forall p \in P, n \in \mathbb{N} \cdot \operatorname{chan}(p$, in, $n) \neq g\}$ and $K=\{g \in G \mid \forall p \in P, n \in$ $\mathbb{N}$.chan $(p$, out,$n) \neq g\}$. Then the following hold.

- If $H \neq \emptyset \wedge K=\emptyset$, assume $H=\left\{g_{1}, \ldots, g_{l}\right\}$. Then,

$$
\forall\left(b_{1}, \ldots, b_{l}\right) \in \mathbb{B}^{l} .\left(\exists s \in W . \bigwedge_{1 \leq j \leq l}\left(b_{j} \Leftrightarrow g_{j} \cdot \mathbf{t r d y} \in L(s)\right)\right)
$$

- If $H=\emptyset \wedge K \neq \emptyset$, assume $K=\left\{x_{1}, \ldots, x_{h}\right\}$. Then,

$$
\forall\left(d_{1}, \ldots, d_{h}\right) \in\left(\prod_{1 \leq u \leq h} c\left(x_{u}\right)\right) .\left(\exists s \in W . \bigwedge_{1 \leq u \leq h}\left(x_{u} . \text { data }=d_{u} \in L(s)\right)\right) .
$$

- If $H \neq \emptyset \wedge K \neq \emptyset$, assume $H=\left\{g_{1}, \ldots, g_{l}\right\}$ and $K=\left\{x_{1}, \ldots, x_{h}\right\}$. Then,

$$
\begin{aligned}
\forall\left(b_{1}, \ldots, b_{l}\right) \in \mathbb{B}^{l},\left(d_{1}, \ldots, d_{h}\right) \in( & \left.\prod_{1 \leq u \leq h} c\left(x_{h}\right)\right) . \\
(\exists s \in W . & \bigwedge_{1 \leq j \leq l}\left(b_{j} \Leftrightarrow g_{j} \cdot \mathbf{t r d y} \in L(s)\right) \wedge \\
& \left.\bigwedge_{1 \leq u \leq h}\left(x_{u} \cdot \text { data }=d_{u} \in L(s)\right)\right) .
\end{aligned}
$$

(3) $W \neq \emptyset$.
(4) $\forall s \in W, s^{\prime} \in I .\left(s, s^{\prime}\right) \in \rightarrow$.

Fix arbitrary $s \in I$. From Lemma 4.15 it follows that there is $s^{\prime} \in W$ such that $\left(s^{\prime}, s\right) \in \rightarrow$. Since $W \subseteq I$ we conclude that for all $s \in I$ there is $s^{\prime} \in I$ such that $\left(s^{\prime}, s\right) \in \rightarrow$.

### 4.4.1 Characterizing All Backward Runs in $M$

Further, we characterize all backward runs of $M$ with length up to $k$. Let $V_{0}, \ldots, V_{k}$ be vectors of state variables. Theorem 4.16 allows us to use the following characterization:

$$
\operatorname{BackRuns}(M)=\operatorname{Final}\left(V_{0}\right) \wedge\left(\bigwedge_{0 \leq i<k} T\left(V_{i+1}, V_{i}\right)\right) \wedge\left(\bigvee_{0 \leq i \leq k} \operatorname{Init}\left(V_{i}\right)\right) \wedge \bigwedge_{0 \leq i \leq k} \operatorname{State}\left(V_{i}\right)
$$

Note that without the fact that every initial state has a possibility to do a step backward to an initial state, the formula provided above would be incorrect.
All backward runs of $M$ with length up to $k+1$ can be split up into prefixes and suffixes as follows. Prefixes of backward runs of $M$ are characterized by:

$$
\operatorname{Pref}(M)=T\left(V_{1}, V_{0}\right) \wedge \operatorname{Final}\left(V_{0}\right) \wedge \bigwedge_{0 \leq i \leq 1} \operatorname{State}\left(V_{i}\right)
$$

Suffixes with length up to $k$ of backward runs of $M$ are then characterized by:

$$
\operatorname{Suf}_{k}(M)=\left(\bigvee_{1 \leq i \leq k+1} \operatorname{Init}\left(V_{i}\right)\right) \wedge\left(\bigwedge_{0<i \leq k} T\left(V_{i+1}, V_{i}\right)\right) \wedge \bigwedge_{1 \leq i \leq k+1} \operatorname{State}\left(V_{i}\right)
$$

```
Algorithm 1 Interpolation-based backward reachability analysis
    function BackReach(M,k)
        if \(\operatorname{SAT}\left(\operatorname{Init}\left(V_{0}\right) \wedge \operatorname{Final}\left(V_{0}\right)\right)\) then
            return true
        end if
        \(R \leftarrow \operatorname{Final}\left(V_{0}\right)\)
        while true do
        \(M^{\prime} \leftarrow(\) Init, State, \(T, R)\)
        \(A \leftarrow \operatorname{Pref}\left(M^{\prime}\right)\)
        \(B \leftarrow \operatorname{Suf}_{k}\left(M^{\prime}\right)\)
        if \(\operatorname{SAT}(A \wedge B)\) then
            if \(\operatorname{Final}\left(V_{0}\right)=R\) then
                return true
            else
                Abort
                end if
        else
            \(P \leftarrow \operatorname{IPL}(A, B)\)
            \(R^{\prime} \leftarrow P\left(V_{0}\right)\)
            if \(R^{\prime} \Rightarrow R\) then
                    return false
                end if
                \(R \leftarrow R \vee R^{\prime}\)
        end if
        end while
    end function
```


### 4.4.2 The Algorithm

Algorithm 1 uses interpolation to check if it is possible to discover an initial state by going backwards from a final state in $M=$ (Init, State, $T$, Final). The intuition behind the algorithm is as follows. The algorithm iteratively over-approximates the states that are backward reachable from a deadlock state; when a fix-point is reached, the over-approximation is done. If there are no initial states in the overapproximation, the conclusion is that there is no backward run in $M$ of any length. Note that the algorithm is unbounded; parameter $k$ is used to set the precision of the over-approximation.

Given $M$ and a $k>0$, the algorithm first checks if there exists an initial state which is also final. This is done by checking satisfiability of $\operatorname{Init}\left(V_{0}\right) \wedge \operatorname{Final}\left(V_{0}\right)$. If there is no such state, we let $R=\operatorname{Final}\left(V_{0}\right)$. That is, we start approximating backward reachable states with the set of final states. We then treat $R$ as the condition for a state to be final and check satisfiability of $\operatorname{Pref}(($ Init, $\operatorname{State}, T, R)) \wedge \operatorname{Suf}_{k}(($ Init, State, $T, R))$. If $\operatorname{Pref}(($ Init, State, $T, R)) \wedge \operatorname{Suf}_{k}(($ Init, State, $T, R))$ is satisfiable, we know that it is possible to discover an initial state by taking at least one and not more than $k+1$ steps backwards from $R$. In the first iteration of the algorithm, since $R=\operatorname{Final}\left(V_{0}\right)$, we
know that we found a path that connects an initial and a final state. In the subsequent iterations of the algorithm, we need to check if $R=\operatorname{Final}\left(V_{0}\right)$. If it is not the case, the algorithm aborts without deciding backward reachability of an initial state, since due to over-approximation, $R$ might contain a state which is not final. In that case, we need to re-run the algorithm with an increased $k$.

If there is no satisfying assignment to

$$
\operatorname{Pref}((\text { Init, State }, T, R)) \wedge \operatorname{Suf}_{k}((\text { Init, State }, T, R))
$$

we compute an interpolant $P$ of $\operatorname{Pref}((\operatorname{Init}$, State, $T, R))$ and $\operatorname{Suf}_{k}((\operatorname{lnit}$, State, $T, R))$. By Definition 4.14, we have $\operatorname{Pref}(($ Init, State, $T, R)) \Rightarrow P$, and since

$$
\operatorname{Pref}((\text { Init, State }, T, R))=T\left(V_{1}, V_{0}\right) \wedge R\left(V_{0}\right) \wedge \bigwedge_{0 \leq i \leq 1} \operatorname{State}\left(V_{i}\right)
$$

we know that $P$ holds in $V_{1}$. Since $P \wedge \operatorname{Suf}_{k}(($ Init, State, $T, R))$ is not satisfiable, we conclude that there is not path form an initial state to $V_{1}$ with length $k$. Thus $R \vee P\left(V_{0}\right)$ gives us a new over-approximation of states from which it is not possible to discover an initial state by going backwards. Since $M$ is finite, we inevitably reach a fix-point when the new over-approximation is the same as the previous one, which will give us an inductive invariant that holds in all states that can be discovered by going backwards from a deadlock state. Since no state in $R$ satisfies $I$, we conclude that it is not possible to discover an initial state by going backwards from an initial state at all. To establish correctness of the algorithm, we first prove that if it terminates, it returns true if and only if $M$ has a backward run.
Theorem 4.17. For $k>0$, if $\operatorname{BackReach}(M, k)$ terminates without aborting, it returns true if and only if $M$ has a backward run.

Proof. Fix arbitrary $M$ and $k>0$ and assume that Algorithm 1 terminates without aborting. Distiniguish the following cases.

- BackReach $(M, k)$ returns true. We know that either $\operatorname{Init}\left(V_{0}\right) \wedge \operatorname{Final}\left(V_{0}\right)$ has a satisfying assignment, or $\operatorname{Pref}\left(M^{\prime}\right) \wedge \operatorname{Suff}_{k}\left(M^{\prime}\right)$ has a satisfying assignment and $R=\operatorname{Final}\left(V_{0}\right)$. That is, either there is a backward run of zero length, or there is a backward run with length up to $k$ steps.
- BackReach $(M, k)$ returns false. Further we prove that there is no backward run of any length. Fix arbitrary $n$ and a sequence of states $S_{0} \ldots S_{n}$ such that Final $\left(S_{0}\right)$ and for all $0 \leq i<n$ it holds that $T\left(S_{i+1}, T_{i}\right)$. We show that $S_{0} \ldots S_{n}$ is not a backward run.

As a stepping stone towards proving that $S_{0} \ldots S_{n}$ is not a backward run, we first show that for all $0 \leq i \leq n$, we have $R\left(S_{i}\right)$. Since during the first iteration of the algorithm $R \leftarrow$ Final $\left(V_{0}\right)$ and during the subsequent iterations $R \leftarrow R \vee R^{\prime}$, we conclude that $\operatorname{Final}\left(V_{0}\right) \Rightarrow R$. Since Final $\left(S_{0}\right)$ is true, we conclude that $R\left(S_{0}\right)$. By the definition of $P$ we have $A \rightarrow P$, hence for all states $S, S^{\prime}$ we have $R(S) \wedge T\left(S^{\prime}, S\right) \Rightarrow R^{\prime}\left(S^{\prime}\right)$. Since Algorithm 1 returned false, we know that $R^{\prime} \Rightarrow R$ and hence we conclude that for all states $S, S^{\prime}$ we have $R(S) \wedge T\left(S^{\prime}, S\right) \Rightarrow R\left(S^{\prime}\right)$.

From the latter, from the fact that $R\left(S_{0}\right)$, and since for all $0 \leq i<n$ it holds that $T\left(S_{i+1}, S_{i}\right)$, we conclude that for all $0 \leq i \leq n$ we have $R\left(S_{i}\right)$.

As another stepping stone towards showing that $S_{0} \ldots S_{n}$ is not a backward run, we prove that $R \wedge$ Init has no satisfying assignment. Initially, $R=$ Final and since Algorithm 1 returned false, we know that Init $\wedge$ Final is not satisfiable. Within subsequent iterations of Algorithm $1 P \wedge B$ was not satisfiable and hence $R^{\prime} \wedge$ Final had no satisfying assignment within each iteration. Hence, after Algorithm 1 terminated, we conclude that $R \wedge$ Init has no satisfying assignment.
We already established that $R$ holds in all states of the sequence $S_{0} \ldots S_{n}$. Since $R\left(S_{n}\right)$ and $R \wedge$ Init is not satisfiable, we conclude that $S_{n}$ is not an initial state and hence $S_{0} \ldots S_{n}$ is not a backward run. We showed that all sequences of states are not backward runs.

Now we prove that with a sufficiently large $k$, the algorithm terminates by returning either true or false.

Theorem 4.18. For every $M$, there is $k$ such that $\operatorname{BackReach}(M, k)$ terminates without aborting.

Proof. Fix arbitrary $M$ and let $k$ be equal to the length of the longest path which starts in the initial state and involves unique states only. Distinguish the following cases.

- $M$ is such that there is a backward run. Since $k$ is such that there is a path starting in the initial state which involves all reachable states, there is a satisfying assignment to $A \wedge B$. Hence, the algorithm terminates in the first iteration returning a true.
- $M$ is such that there is no backward run. Since $k$ is big enough to explore all reachable states starting from the initial state and at the same time there is no backward run, we conclude that there is no backward run of any length. Hence, at every iteration of the while-loop, there will be no satisfying assignment to $A \wedge B$. At the same time, $R$ will continue to grow. Due to the fact that $M$ is finite, at some point $R$ reaches a fix-point and the algorithm terminates returning a false.

We showed correctness of the interpolation-based backward reachability analysis as well as the fact that there is always a sufficiently large $k$ which leads to termination of the lagorithm without the execution. It is also important to note that using interpolation in the backward reachability analysis is superior to the explicit backward reachability analysis presented earlier in the chapter, since the interpolation-based method is unbounded.

### 4.4.3 Evaluation of Interpolation-Based Backward Reachability

We implemented the interpolation-based backward reachability analysis discussed earlier in this chapter. The implementation works in such a way that based on a given
xMAS network and a given parameter $k$, it generates problems for an SMT-solver to approximate backward reachable states. The tool terminates when either it decides that the given xMAS network is deadlock-free, or it concludes that an initial state is reachable from a deadlock state, or it decides that the precision of the approximation is not enough to draw a conclusion. In the latter case, the verification should be re-run with an increased parameter $k$.

Additionally, we implemented the McMillan's method for conducting the reachability analysis of a deadlock state from an initial state. We did this with the purpose of comparing the forward and backward interpolation-based reachability analysis.
We also used the IC3 algorithm of NuXmv in order to compare it with backward (forward) interpolation-based reachability analysis.
For the experiments, we used two sets of models. One set is the same power domains set of models as we used for the $k$-step backward reachability analysis; see Subsection 4.3.2 of the current chapter. Every model in the set is deadlock-free, but SAT-based verification reports a deadlock. For the other set of models, we modified each power domain model from the first experimental set in such a way that the model contains a deadlock; these deadlock modifications were done in the same way as in Chapter 3.

Similar to the experiments with $k$-step backward reachability, the experiments with interpolation-based backward reachability analysis were conducted using a MacBook Pro 2015, 2,7GHz Intel Core i5, 16Gb RAM, running MacOS Big Sur 11.3. For SAT solving, we use the MathSAT5 solver, version 5.6.5 64-bit [Cim+13]. MathSAT5 was chosen due to the fact that it conveniently computes interpolants needed for our method. Instructions to reproduce the experiments and the script used to obtain our results are available at [FS21].

For experimental runs, we used a script, which started an experiment with a $k=3$. Such starting value of $k$ was due to the fact that mostly, runs with $k<3$ are redundant. If the result is inconclusive, the script re-started the experiment with $k=k+1$. Every run lasted until either the deadlock freedom or the presence of a deadlock was concluded.

Based on the results that we obtained we conclude that IC3 outperforms both backward and forward interpolation-based reachability analysis methods for both deadlock-free and deadlock models. Backward interpolation-based reachablity performs worse than forward interpolation-based reachablity approach on deadlock-free models. On the models with real deadlocks, backward interpolation-based reachability method shows advantage over the forward interpolation-based method.

### 4.5 Discussion

Let us refer to Table 4.1 and Figure 4.6. Our experiments showed that the $k$-step backward reachability approach is hardly applicable on practice. Although, from the theoretical standpoint, the method is sound and complete.

| Model | \#FSMs | DLF | Backward |  | Forward |  | IC3 |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  | DLF | Time (s) | DLF | Time (s) | DLF | Time (s) |
| pd_1_2 | 9 | $\checkmark$ | $\checkmark$ | 23.922 | $\checkmark$ | 20.261 | $\checkmark$ | 1.004 |
| pd_1_3 | 14 | $\checkmark$ | $\checkmark$ | 46.950 | $\checkmark$ | 13.704 | $\checkmark$ | 1.279 |
| pd_1_4 | 19 | $\checkmark$ | $\checkmark$ | 65.919 | $\checkmark$ | 20.929 | $\checkmark$ | 1.759 |
| pd_1_5 | 24 | $\checkmark$ | $\checkmark$ | 93.327 | $\checkmark$ | 35.278 | $\checkmark$ | 3.234 |
| pd_1_6 | 29 | $\checkmark$ | $\checkmark$ | 190.044 | $\checkmark$ | 55.889 | $\checkmark$ | 3.073 |
| pd_1_7 | 34 | $\checkmark$ | $\checkmark$ | 259.032 | $\checkmark$ | 116.807 | $\checkmark$ | 3.883 |
| pd_1_2_dl | 9 | $x$ | $x$ | 5.730 | $x$ | 12.421 | $x$ | 2.163 |
| pd_1_3_dl | 14 | $x$ | $x$ | 10.757 | $x$ | 21.108 | $x$ | 2.979 |
| pd_1_4_dl | 19 | $x$ | $x$ | 17.828 | $x$ | 32.780 | $x$ | 7.574 |
| pd_1_5_dl | 24 | $x$ | $x$ | 23.042 | $x$ | 52.356 | $x$ | 9.173 |
| pd_1_6_dl | 29 | $x$ | $x$ | 31.304 | $x$ | 67.720 | $x$ | 7.467 |
| pd_1_7_dl | 34 | $x$ | $x$ | 38.219 | $x$ | 88.684 | $x$ | 20.204 |

Table 4.1: Experimental results on limiting false deadlocks


Figure 4.6: Visualization of the results.

Based on the evaluation of the interpolation-based backward reachability analysis, we observed that the backward reachability performed worse than the forward reachability on the deadlock-free models. This refutes our initial conjecture that, in practice, the set of states that are backward reachable from a deadlock state is smaller than the set of states that are reachable from an initial state. In addition, our experiments showed that the backward reachability is superior to the forward reachability for finding real deadlocks. Our interpolation-based backward reachability analysis is applicable in practice. However, the comparison with IC3 showed that our method is not state-of-the-art.

### 4.6 Conclusion

The SAT-based approach to verify liveness of xMAS networks is prone to false deadlocks. In the current chapter we investigated ways of limiting the numbers of false deadlocks reported by the SAT-based liveness verification. For that, we presented two approaches to make the SAT-based approach to verify liveness complete. The first approach adds $k$-step reachability encoding to the original SAT-problem. The second approach is inspired by interpolation-based reachability analysis method introduced by McMillan et al. [McM03]. The chapter contains correctness proofs for both methods as well as the evaluation of effectiveness of the methods.

## Chapter 5

## Automatic Generation of Hardware Checkers from Formal Micro-architectural Specifications

### 5.1 Introduction

Interconnects play a crucial role in the correctness and performance of modern MultiProcessor Systems-on-Chips (MPSoC's). The significant number of queues induces a large state space and the distributed control hinders the application of localisation techniques [RB12]. To combat this challenge, researchers have explored techniques to formally model and analyse abstractions of interconnect implementations. In particular, techniques based on the xMAS language proposed by Intel have attracted a lot of interest: modeling [CKO12], verification of safety [CK12] and liveness properties [VS11; GCK11], performance evaluations [HS14; XZ15], or asynchronous designs [BSY17; VJS13].

These analyses are performed on abstractions. Establishing a relation between these abstract models and Register Transfer Level (RTL) implementations is the challenge tackled in this chapter. An approach was proposed by Joosten and Schmaltz [JS15]. The authors proposed to automatically extract xMAS models from RTL designs. A limitation is that this approach only discovers the basic xMAS primitives. In contrast with the work of Joosten and Schmaltz, we infer RTL code from abstract microarchitectural models. We consider the Micro-architectural Description Language (MaDL), a textual version of xMAS with new primitives easing the modeling but making their discovery even more challenging. We propose a method to turn a given xMAS network into a Non-deterministic Finite Automaton (NFA) with actions
corresponding to the behavior of the interface of the given network (i.e. the behavior of its sources and sinks). The NFA is then determinized. We also add a special error state to the DFA such that whenever the DFA gets an action that is impossible w.r.t. to the xMAS semantics, the DFA then gets to the error state and gets stuck there. We then translate the resulting DFA into SystemVerilog [Tar20]. We call the obtained SystemVerilog code a hardware checker. It is then possible to use the hardware checker with the corresponding RTL implementation as inputs to a SystemVerilog model-checker so that the model checker verifies if one of the traces of the RTL implementation can be used to reach the error state in the hardware checker. The said methodology is used to prove trace inclusion between an RTL design and a MaDL model.

Contributions. Our contributions are (1) a method to represent the visible behaviour of MaDL models, (2) a method to turn this representation into a SystemVerilog checker, and (3) several experimental examples, including a virtual channel and a re-order buffer.

Structure of the chapter. In Section 5.2, we provide background information about MaDL. In Section 5.3, we explain how we extract the visible behavior of a given MaDL model and encode it in an NFA. In Section 5.4, we describe how we obtain hardware checkers using the visible behavior of the given MaDL model; this includes determinization of the NFA obtained before and adding an error state, and translating it into SystemVerilog. Section 5.5 contains experimental results and discussion. We conclude in Section 5.6.

### 5.2 Background and Overview

MaDL is an open-source project ${ }^{1}$ that involves a description language and associated analysis techniques. The language originates from xMAS - for eXecutable MicroArchitectural Specifications - proposed by Intel [CKO12]. In contrast to xMAS, MaDL has a textual syntax, recursive data types, loops and parameters. It allows the definition of macro blocks for compositional modeling. Its verification engines implement invariant generation [CK12], deadlock analysis [VS11; GCK11], deadlock reachability [WJS15], and the generation of SystemVerilog prototypes. We introduce the concepts of MaDL relevant to this chapter. More details can be found in the MaDL GitHub pages.

## Channels, transfers, and persistency

Primitives are connected via typed channels. A channel connects exactly two primitives called the initiator and the target. A channel consists of three signals:

- irdy: high when the channel contains valid data, that is, the initiator is ready to transfer;

[^9]

Figure 5.1: Original primitives of the xMAS language [CKO10].

- data: the data contained in the channel;
- trdy: high when the target of the channel is ready to accept data.

When both irdy and trdy are true, the data is transferred from the initiator to the target.

Persistency means that every primitive commits to a transfer, namely, when irdy is true it must remain high until trdy is true.

## Data types and colors

MaDL supports the following data types: constant, enumeration, and struct. The following code snippet declares several constant types, making up enumeration types, used in a struct type.

```
const dstQ, dst1, req, rsp;
enum dst_t {dstQ;dst1;};
enum trans_t {req;rsp;};
struct pkt_t { dst : dst_t; type: trans_t; };
```

We often refer to a value of a packet type using the term color. In the above example, a packet with dst $=$ dst $\theta$ and type $=$ req is a color.

## Primitives

The basic primitives of MaDL are the ones originally proposed in the xMAS language ${ }^{2}$. Their graphical representation is pictured in Figure 5.1. We briefly sketch their semantics.

A source non-deterministically injects a packet in the network. A sink consumes a packet non-deterministically. Sources and sinks are assumed to be fair. They always eventually inject or consume packets. A fork consumes the input packet and produces two output packets. This happens if and only if both outputs can accept a new packet. A join consumes two packets and produces one packet. As generally done in related works [VS11; GCK11], a join has a control input and a data input. When both inputs have a packet, the packet on the data input is propagated to the output. A merge non-deterministically arbitrates when its two inputs have a packet. Arbitration is left abstract but is assumed to be fair. Each input always eventually is granted. A

[^10]switch routes the input packet to one output. The decision solely depends on the data contained in the input packet. A function modifies its input packet. A queue stores and forwards packets following a First-In-First-Out (FIFO) policy.

MaDL also adds new primitives. In particular, it adds a non-deterministic demultiplexer, called LoadBalancer and a complex sorting primitive called MultiMatch. A LoadBalancer has one input and $n$ outputs; if there is data at its input, the LoadBalancer decides non-deterministically, through which output the data is transferred further. A MultiMatch has $n$ outputs each controlled by a match input. It also has $m$ data inputs. At each cycle, a predicate between each data input and each control input is evaluated. If the predicate holds, the data input is forwarded to the output controlled by the match input. Otherwise, the data input stalls. The experimental section shows the construction of a re-order buffer using these two primitives.

### 5.2.1 Running Example

The basic syntax of MaDL consists of statements declaring and connecting channels. These statements have the following syntax:

```
chan <outs> := <Primitive> (<ins>);
```

Note that in this example, channels in list "ins" must be declared somewhere else.
Example 5.1. We consider a simple network composed of a source injecting packets with color red. Packets are sent to both queues $q_{0}$ and $q_{1}$. The merge nondeterministically takes one packet from either queue and forwards it to the sink. The corresponding MaDL code is the following:

```
const red;
```

chan to_qQ, to_q1 := Fork(Source(red));
chan qQ_out $:=$ Queue ( 1, to_qQ) [q@];
chan q1_out $:=$ Queue(1,to_q1)[q1];
Sink(Merge(q0_out, q1_out));


Figure 5.2: A simple network (running example).
In Figure 5.2, we visualise the example using xMAS. This example will further be used as a running example.

### 5.3 Interface Behaviour

In this section, we define the notion of the interface of a given MaDL model. We then define the interface actions for the model; using the set of actions we describe how we obtain the NFA that represents the visible behavior of the given MaDL model.

In contrast to the Kripke Structure semantics that we introduced in Chapter 2, the interface behavior gives an action-based perspective on xMAS/MaDL. Based on a given xMAS network, we generate a Non-deterministic Finite Automaton (NFA); the states of the generated NFA capture the contents of all queues of the network, the actions of the NFA reflect the irdy, trdy, and data values of output channels of all sources and input channels of all sinks of the network.

### 5.3.1 Interface Actions

The interface of a MaDL model consists of all sources and sinks. Let Src be the set of all sources of a given model. Let Snk be the set of all sinks of a given model. For any $s \in \operatorname{Src} \cup$ Snk, let $C_{s}$ be the set of colors produced by source $s$ or consumed by sink $s$. A source $s \in \operatorname{Src}$ either stays idle - represented by an Idle action - or requests the injection of a packet with color $c \in C_{S}$ - represented by an Inject ${ }_{c}$ action. Note that in our setting, sources are not restricted to produce values of a singleton type. An idle action corresponds to $\neg$ irdy of the source's output, whereas an Inject ${ }_{c}$ action corresponds to irdy $\wedge$ data $=c$. A sink $s \in$ Snk either rejects packets of any color - represented by a Reject action - or is ready to consume a packet with color $c$ represented by a Consume ${ }_{c}$ action. Reject corresponds to $\neg$ trdy of the sink's input, while Consume ${ }_{c}$ corresponds to trdy $\wedge$ data $=c$. The complete set of interface actions is the cross-product of all source and sink actions. The formal definition of actions is as follows:

Definition 5.2. (Actions). For any $s \in \operatorname{Src}$, let $R_{s}=\left\{x \mid x=\operatorname{Idle} \vee x=\operatorname{Inject}_{c^{\prime}} c \in C_{x}\right\}$ be the set of source actions of $s$. For any $s \in$ Snk, let $N_{s}=\{x \mid x=$ Reject $\vee x=$ Consume $\left._{c}, c \in C_{s}\right\}$, be the set of sink actions of $s$. Let $R^{\prime}=\prod_{s \in S r c} R_{s}$ be a cartesian product of sets of actions of all sources from Src. Let $N^{\prime}=\prod_{s \in S n k} N_{s}$ be a cartesian product of sets of actions of all sinks from Snk. Then, the set of global actions $A$ is defined as $R^{\prime} \times N^{\prime}$.

Thereby, global actions involve sources and sinks only.
Example 5.3. Consider the running example. The set of sources is $\mathrm{Src}=\{\operatorname{src} 0\}$. The set of colors possibly injected at that source is $C_{\text {src } 0}=\{$ red $\}$. The source therefore has two possible actions: either it is idle or it tries to inject a red packet. The possible actions at the source are the following:

$$
R_{\text {src0 }}=\left\{\text { Idle, } \text { Inject }_{\text {red }}\right\}
$$

Similarly, the sink either consumes a red packet or rejects any packet. The possible actions at the sink are the following:

$$
N_{\text {snk } 0}=\left\{\text { Reject }, \text { Consume }{ }_{\text {red }}\right\}
$$

The set of global actions is then the cross product of the source and sink actions. This defines the following set:

$$
\begin{aligned}
A= & \{(\text { Idle, Reject }), \\
& (\text { Idle, Consume } \\
& \left(\text { Inject }_{\text {red }}, \text { Reject }\right), \\
& \left.\left(\text { Inject }_{\text {red }}, \text { Consume }_{\text {red }}\right)\right\}
\end{aligned}
$$

### 5.3.2 Action Behaviour

The behaviour of the network in terms of its interface actions is represented by an NFA. We now (1) define the notion of states, (2) define possible actions in a state, and (3) define the state update function.

## NFA states

The state is defined by the states of the queues and the states of the sources. Each queue is represented by an ordered list of the colors stored in the queue. The head of the list corresponds to the head of the queue.
Sources need to be persistent. This implicit assumption in MaDL needs to be explicit in the NFA to properly characterise possible legal actions. Persistency of sources means that if a source tries to inject a packet with a given color - that is, executes an action Inject ${ }_{c}$ for some color $c$ - the source is obliged to keep trying to inject this color until it succeeds. To reflect this, a source is either in a state "free" - where it is free to inject any color or to remain idle - or in a state "next $c$ " expressing the fact that the source is committed to inject color $c$.
The global state of the NFA is defined by the product of the queue states and the source states.

Definition 5.4. Given a set of sources Src , for all $s \in \mathrm{Src}$, the set of source states of $s$ is defined as $S_{s}=\{$ Free $\} \cup\left\{\operatorname{Next}_{c_{1}}, \operatorname{Next}_{c_{2}}, \ldots, \operatorname{Next}_{c_{n}}\right\}$, where $c_{1}, c_{2}, \ldots, c_{n}$ are the colors that can be injected by $s$. Given a set of queues $Q$ and a set of sources Src, for any $q \in Q$, let $S_{q}$ be the set of all possible contents of $q$, and for any $s \in \operatorname{Src}$, let $S_{s}$ be the set of states of $s$. Then, the set of all global states is defined as follows:

$$
S=\prod_{q \in Q} S_{q} \times \prod_{s \in \operatorname{Src}} S_{s}
$$

In the initial state of the NFA, all sources are free, and all queues are empty. Note that by not defining states of sinks, we do not take into account the persistency of sinks. This results in a more compact global state space.

Example 5.5. Consider the running example. The states of the source are either idle or committed to color red:

$$
S_{\text {src } 0}=\left\{\text { Free, } \text { Next }_{\text {red }}\right\}
$$

Given the set of queues $Q=\left\{q_{0}, q_{1}\right\}$, we define the set of states of $q_{0}$ and $q_{1}$ as $S_{q_{0}}=S_{q_{1}}=\{[],[r e d]\}$, where [] denotes an empty queue. Finally, the set of global states is the cross product of source and queue states:

$$
\begin{aligned}
S= & \left\{(\text { Free, }[],[]),(\text { Free, }[\text { red }],[\text { red }]),\left(\text { Next }_{\text {red }},[\text { red }],[\text { red }]\right),\right. \\
& (\text { Free, }[],[\text { red }]),(\text { Free, }[\text { red }],[]),\left(\text { Next }_{\text {red },[],[\text { red }]),}\right. \\
& \left(\text { Next }_{\text {red }},[\text { red },[]),\left(\text { Next }_{\text {red }},[],[]\right)\right\} .
\end{aligned}
$$

## NFA transition relation

To define the transition relation, we need to specify (1) the possible transitions and their labels, that is, the possible actions in the current state and (2) the state update, that is, the new occupancy of the queues and the new state of the sources.

Persistency creates the constraint that if a source is in state "next $c$ ", the local action at that source must be a re-try of sending color $c$, namely, action Inject ${ }_{c}$. If a source is free, then a source can try to inject any of the colors defined by its type or choose to remain idle.

Definition 5.6. Given a global action $a \in A$, a global state $s \in S$. Let $\operatorname{Src}=\left\{p \_0, \ldots, p \_n\right\}$ be the set of all sources, $a=\left(a_{p_{0}}, \ldots, a_{p_{n}}, a_{0}, \ldots, a_{m}\right), s=\left(s_{p_{0}}, \ldots, s_{p_{n}}, s_{0}, \ldots, s_{l}\right)$. Action $a$ is a valid action in state $s$ if the following condition holds:

$$
\bigwedge_{0 \leq i \leq n}\left(s_{p_{i}}=\operatorname{Next}_{c} \Rightarrow a_{p_{i}}=\text { Inject }_{c}\right) .
$$

Except for persistency, actions are unconstrained. In any state, any valid action is possible. The NFA has a transition labelled with this global action in that state.

Example 5.7. Figure 5.3 shows the states and transitions of the NFA obtained for the running example.

Note, that there are four non-deterministic transitions from state 1. This happens because both queues contain packets, and the merge can decide non-deterministically, from which queue to consume a packet. Also, the NFA is not input enabled. The transitions that are not allowed according to Definition 5.6 are absent.

## State update

In a MaDL model, forks and joins ensure that several channels transfer together, that is, they all have their irdy and trdy signals asserted at the same time. Wouda and Schmaltz formalised this notion of transfer islands [WJS15]. A transfer island is a set of channels such that a channel fires - irdy and trdy are asserted - if and only if all other channels in the island also fire.

Definition 5.8. Let $M$ denote the set of channels of a MaDL model. A transfer island is a non-empty set of channels $I \subseteq M$, such that for any $x \in I, x$ transfers $x$.irdy $\wedge x$.trdy - if and only if all the channels from $I \backslash\{x\}$ transfer as well.


Action labels:
0 (Idle, Reject),
1 (Inject ${ }_{\text {red }}$, Reject),
2 (Idle, Consume ${ }_{\text {red }}$ ),
3 (Inject ${ }_{\text {red }}$, Consume ${ }_{\text {red }}$ ).
State labels:
0 (Free, [], []),
1 (Free, [red], [red]),
2 (Next ${ }_{\text {red }}$, [red], [red]),
3 (Free, [], [red]),
4 (Free, [red], []),
5 ( $\mathrm{Next}_{\text {red }}$, [], [red]),
6 ( $\mathrm{Next}_{\text {red }}$, [red], []),
7 (Next ${ }_{\text {red }}$ [], []).

Figure 5.3: NFA (Running example).

We denote the set of all transfer islands of a given MaDL model by $I^{\prime}$.
Example 5.9. Consider the running example. Assume previously unnamed output channels are named by appending '_out' to the name of the input primitive. An additional ' $a$ ' is used to denote the top and bottom outputs. The set of islands is the following:

$$
\begin{aligned}
I^{\prime}= & \{\{\text { scr0:red_out,to_q0, to_q1\}, } \\
& \{\text { q0_out, mrg0_out }\}, \\
& \{\text { q1_out, mrg0_out }\}\}
\end{aligned}
$$

The first island groups together all channels between the source and the two queues. The other two islands identify the selection by the merge of one of its inputs.

To manipulate islands, we define the input primitive of a transfer island. An input primitive of a transfer island is a primitive, some output channels of which are in the island. Similarly, an output primitive of a transfer island is a primitive some input channels of which are in the island.

Definition 5.10. Given a transfer island $x$ and a primitive $p$, let Out ${ }_{p}$ denote the set of output channels of $p$, and let $\operatorname{In}_{p}$ denote the set of input channels of $p$. We call $p$ an input primitive of transfer island $x$, if $\mathrm{Out}_{p} \cap I \neq \varnothing \wedge \operatorname{In}_{p} \cap I=\varnothing$. We call $x$ an output primitive of $I$, if $\mathrm{Out}_{x} \cap I=\varnothing \wedge \mathrm{In}_{x} \cap I \neq \varnothing$.

We are ready to introduce the firing conditions for an island. Basically, an island fires when its input primitives are ready to transfer data and its output primitives are ready to consume data. Given a global action, a global state, and an island $I$, the island is able to transfer if the following holds:

- for all $x \in \operatorname{Src}$, if $x$ is an input primitive of $I$, then the action of $x$ is Inject $_{c}$, where $c \in C_{x}$,
- for all $q \in Q$, if $q$ is an input primitive of $I$, then $q$ is non empty,
- for all $y \in \operatorname{Snk}$, if $y$ is an output primitive of $I$, then the action of $y$ is Consume ${ }_{c}$, where $c$ is the packet that is transferred to $y$,
- for all $z \in Q$, if $z$ is an output primitive of $I$, then $z$ is not full.

Given a global action and a global state, all queues of all firing islands are updated. This models the synchronous update of all queues. Each queue of a firing island $I$ is updated in the following way:

- for all queues $q \in Q$, if $q$ is an input primitive of $I$, then dequeue from $q$,
- for all queues $q \in Q$, if $q$ is an output primitive of $I$, then enqueue the packet that is being transferred through the island into $q$.

Given a source $x \in \operatorname{Src}$, its state $s_{x}$ and its action $a$, for all packets $c \in C_{s}$ that $s$ can inject and for all transfer islands $I \in I^{\prime}$, the way source state is updated is as follows.

For all transfer islands $I \in I^{\prime}$, if:

- $s_{x}=$ Next $_{c}$,
- $a=$ Inject $_{c^{\prime}}$
- $x$ is an input primitive of $I$,
- I can transfer,
then the successor of $s_{x}$ is Free.
If there exists an $I \in I^{\prime}$, such that $I$ cannot transfer and the following holds:
- $a=$ Inject $_{c^{\prime}}$
- $x$ is an input primitive of $I$,
then the successor of $s_{x}$ is $\mathrm{Next}_{c}$.
Let $a \in S$ and $s \in S$ be given global action and global state. Let $M$ be the set of all merges, LB be the set of all loadbalancers, MM be the set of all multimatches, and $I_{\text {trans }}$ be the set of islands that can transfer w.r.t. $a$ and $s$. For all $m \in M$, let $\operatorname{In}_{m}$ be the set of input channels of $m$. For all $l \in \mathrm{LB}$, let $\mathrm{Out}_{l}$ be the set of output channels of $l$. For all $n \in \mathrm{MM}$, let $\operatorname{InM}_{n}$ be the set of match input channels of $n$ and $\operatorname{InD}_{n}$ be the set of data input channels of $n$. If there are $m \in M, l \in \mathrm{LB}, n \in \mathrm{MM}$, such that $\operatorname{In}_{m} \cap I_{\text {trans }}>1 \vee \mathrm{Out}_{l} \cap I_{\text {trans }}>1 \vee \operatorname{InD}_{n} \cap I_{\text {trans }}>1 \vee \operatorname{InM}_{n} \cap I_{\text {trans }}>1$, then we need to consider maximal subsets of $I_{\text {trans }}$, such that for all $m \in M, l \in \mathrm{LB}, n \in \mathrm{MM}$ it holds that $\operatorname{In}_{m} \cap I_{\text {trans }} \leq 1 \wedge$ Out $_{l} \cap I_{\text {trans }} \leq 1 \wedge \operatorname{InD}_{n} \cap I_{\text {trans }} \leq 1 \wedge \operatorname{InM}_{n} \cap I_{\text {trans }} \leq 1$. That is, there are no conflicting transfers. Computing several successor states for a given state and action leads to non-determinism.


Figure 5.4: Overall approach

### 5.4 SystemVerilog Checkers

Given a MaDL specification, we generate a SystemVerilog checker, which is, in essence, a Finite Automaton expressed in SystemVerilog that captures the interface behavior of the given MaDL network. In addition to the clock and reset inputs, this checker has the following interface:

- for each source $s \in$ Src, a s.irdy input of type bit, and a s.data input of the corresponding type;
- for each $k \in$ Snk, a $k$.trdy input of type bit, and a $k$.data input of the corresponding type,
- two flags named Error and Overflow.

Figure 5.4 illustrates the connection of the checker - denoted by 'Spec' - to an RTL implementation - denoted by 'Impl'. Both 'Spec' and 'Impl' are driven by the same source and sink actions. The data produced by 'Impl' are fed to 'Spec'. The checker will raise the Error flag when it detects an illegal action. Proving that the situation when the Error flag is high is unreachable proves that the traces of 'Impl' are included into the traces of 'Spec'. As explained below and because of non-determinism, the checker conducts determinization "on the fly"; for that, the checker maintains a queue of all possible current states. Computing the size of this queue is a very difficult problem. The overflow flag indicates an overflow on that queue. The size of this queue is a parameter in the SystemVerilog code and can easily be adjusted. An overflow of this queue means that we underestimated the maximum number of possible current states.

It is important to note that in order to make our checkers suitable for formal verification tools, the code must be restricted to the synthesizable subset of SystemVerilog.

The main computations performed by the checker are pictured in Figure 5.5. To represent non-determinism, the checker maintains a queue containing all possible current states. Let us call this queue st_chk_q. Each state consists of the following elements:


Figure 5.5: Core checker computations

- for each source $k$ : a commitment flag $k \_f r e e$ that is high when the source is in a Next state and a $k$ _state variable representing the injected color.
- for each queue $q$ : a queue state identifier $q_{-}$state.

Instead of representing each queue by an actual queue in the checker, we precompute for each queue all possible state transitions in a case statement. This effectively results in a large increase in the number of gates but a major decrease in the number of flops. This trade-off results in more efficient formal verification.

Let us now walk the reader through the computation steps conducted by a hardware checker by referring to Figure 5.5 as an example.

## Step (1)

The first step is to pick the first state in st_chk_q (i.e. one of the possible current states). Let us call this state st_chk_curr. It is checked whether the current actions are legal w.r.t. this state. If not, the Error flag is raised and computation stops. Otherwise, the computation of the possible next states is started.
Example 5.11. Consider the running example. It has only one source and therefore only one invalid action. Given a state $x$ and a source $s$, an action is illegal iff:

$$
\neg x . s \text { free } \wedge\left(\neg \text { s.irdy } \vee \text { s.data } \neq x . s \_ \text {state }\right)
$$

## Step (2)

The second step is to compute the islands that are possibly active in the current states. This results in a bit vector where each position indicates whether a specific island is
active or not. Let us call this bit vector isl_act.
Example 5.12. Let us recall that the set of islands is

$$
\begin{aligned}
I^{\prime}= & \{\{\text { scr0:red_out, to_q0, to_q1\}, } \\
& \{\text { q0_out, mrg0_out }\}, \\
& \{\text { q1_out, mrg0_out }\}\} .
\end{aligned}
$$

Consider state (Free, red, red), and action (Inject ${ }_{\text {red }}$, Consume ${ }_{\text {red }}$ ). The first island is inactive, since both queues are full. The second and the third islands are active, since the queues are non-empty, and the source action is Consume ${ }_{\text {red }}$. Hence, isl_act $=011$.

## Step (3)

In general, not all the active islands in isl_act can fire simultaneously. For instance, two islands with a common arbiter can be active in the current state but the arbiter must make a non-deterministic choice. The third step is to extract from the current active islands the set of possible legal island configurations. This is a queue of bit vectors. Let us call this queue isl_legal_confs.

Example 5.13. In case isl_act $=011$, the second and the third islands are active, but cannot fire simultaneously due to the common arbiter in the merge primitive. Hence, we split 011 in the following way: isl_legal_confs $=\{010,001\}$.

## Step (4)

Finally, for each legal configuration a new state is computed and enqueued in a new global state st_chk_q. When this queue is full, but a new state should be enqueued, the Overflow flag is raised.
Example 5.14. Again, consider the following state and action respectively:

$$
\begin{aligned}
& \text { (Free, red, red), } \\
& \text { (Inject }_{\text {red }}, \text { Consume }_{\text {red }} \text { ). }
\end{aligned}
$$

For this state and action, isl_legal_confs $=\{010,001\}$. Thus, we compute two state updates. As the queues are full and the source tries to inject a packet, the next state of the source in each case is "Next". Finally, we obtain two distinct successor states $s t \_c h k_{-} q=\left\{\left(\right.\right.$ Next $_{\text {red }},[]$, red $),\left(\right.$ Next $_{\text {red }}$, red, []$\left.)\right\}$.

### 5.5 Experimental Results and Discussion

Setup. Experiments are conducted using five MaDL examples. SM is the running example (see Figure 5.2). SMC is a modification of the running example: (1) the fork is replaced by a switch and (2) the source can inject two distinct colors. SLB is composed of a source, a load balancer with three outputs. Each output is connected

| name | flops <br> spec. | flops <br> impl. | time(s) <br> proof | time(s) <br> !ovf | time(s) <br> cex |
| :--- | :--- | :--- | :--- | :--- | :--- |
| SM | 31 | 11 | 0,5 | 0,5 | 0,1 |
| SMC | 23 | 11 | 0,1 | 2,0 | 0,1 |
| SLB | 101 | 17 | 8 | 3,7 | 0,6 |
| VC | 28 | 27 | 0,2 | 0,1 | 0,1 |
| ROB | 61 | 18 | 17,5 | 17,6 | 1,9 |

Table 5.1: Experimental results
to a queue. Outputs of all queues are connected to sinks. VC is a virtual channel. The implementation uses credit-flow control and corresponds to Figure 3 by Ray and Brayton [RB12]. The specification simply consists of two independent queues, each one with its own source and sink. The structure of the two circuits is completely different. ROB is a re-order buffer with two inputs and one output and defined as follows:

```
struct pkt { tp: [0:0];};
pred f (p: pkt, q: pkt) { p.tp == q.tp};
bus<2> j;
bus<2> i := LoadBalancer(Source(pkt));
let j[0] := Queue(1,i[0]);
let j[1] := Queue(1,i[1]);
chan q_m := Queue(1,Source(pkt));
chan o_s := MultiMatch(f,q_m,j);
Sink(o_s);
```

Packets enter at the source feeding the LoadBalancer. Packet leaves at the output of the MultiMatch in the order given by the second source.

Specifications are obtained by translating the MaDL models into SystemVerilog checkers. Implementations are obtained by translating the same MaDL models into Verilog, except for VC where two different MaDL models are used. When generating implementations, non-determinism is removed. All merges and load balancers implement a round-robin policy. All queues are circular buffers. The combination of a specification and an implementation are given to a commercial formal verification tool to check if there is an unbounded proof of the safety properties that the Error flag and the Overflow flag are always low. Proof times in the table are only given for the Error flag. Experiments are run on a CentOS 6.8 server with four 16-core AMD Opteron 6276 2,3 GHz processors and 128GB 1600MHz memory.

Results. The results are shown in the Table 5.1. The first two columns give the number of flops of the specifications and the implementations. The next columns give the time in seconds to prove trace inclusion, to prove absence of overflows, and to find a counter-example. Errors are injected in implementations by either modifying queue sizes or leaving the data input a free variable. In all cases, an illegal
action is correctly and quickly detected.
Discussion. As expected, the number of flops is larger for specifications than for implementations. This is due to the non-determinism in the specifications. SLB shows that a LoadBalancer introduces more non-determinism than a merge. This is where the increase in the number of flops is the highest (6x). ROB has a LoadBalancer and a MultiMatch, two primitives with a high degree of non-determinism and complex logic. Predictably, ROB is the hardest example for the verification tool, with a proof time of 17,5 seconds. Note that even though the specification of VC is structurally completely different from its implementation, the verification takes a negligible amount of time.

### 5.6 Conclusion

The work presented in this chapter describes an approach to use xMAS in the context of implementation verification. We presented a method to bridge the gap between high-level hardware specifications and RTL implementations. We turned a statebased non-deterministic specification into a hardware checker for checking the inclusion of traces of the given implementation into the traces of the given specification. We exemplified our approach on several examples including a credit-flow virtual channel and a re-order buffer.

As for possible future work, there is still a lot to be done for improving the scalability of the method and making implementation verification on a system level using xMAS possible. Also, another important extension to the method is to add the support of the Finite State Machine primitive.

## Chapter 6

## Conclusion

### 6.1 Summary

The focus of the thesis is verification techniques for the xMAS language and related topics.

The liveness property for xMAS networks is defined using Linear Temporal Logic (LTL) [GCK11], which assumes an xMAS semantics expressed within a framework naturally suitable for temporal logic properties (Kripke Structures, for example). Moreover, there exists a reachability analysis technique for xMAS, which represents xMAS networks as Kripke Structures [GCK11; WJS15]. Despite all that, the literature contains neither a Kripke Structure semantics of xMAS nor a semantics defined in terms of a similar framework. In Chapter 2, we filled the said gap by formally defining the semantics of the xMAS language in terms of Kripke Structures and carefully proving its correctness. This answered the research question RQ1.

To make simultaneous verification of a cache coherence protocol and communication fabrics possible, Verbeek et al. introduced a Finite State Machine extension to the xMAS language [Ver+17]. In Chapter 3, we provided a counter-example for the xMAS with FSMs liveness verification method of Verbeek et al. In the same chapter, we introduced an alternative solution for the liveness verification of xMAS networks with FSMs and proved the correctness of our solution. In addition, we showed that using xMAS with FSMs in combination with our liveness verification method makes system-level liveness verification possible. This answered the research question RQ2.

The state-of-the-art method of liveness verification of xMAS networks is unfortunately not complete. In some instances, the method might report the presence of a deadlock even though the deadlock situation is not reachable Chapter 4 of this thesis is dedicated to investigating ways of solving the issue with spurious deadlocks. In the chapter, we introduced two approaches to making the state-of-the-art liveness verification of xMAS networks complete. One approach is based on an SMT-encoding of $k$-step backward reachability of initial states from the deadlock
state. Another approach uses interpolation to analyze the backward reachability of initial states from the deadlock state. We proved the correctness of both approaches and evaluated their performance. We showed possible ways to make the SAT-based verification technique for xMAS with FSMs complete. However, the technique does not scale well. Hence, we consider RQ3 answered partially.

Besides property verification on abstract hardware models, an essential part of hardware validation is to verify the correctness of hardware implementations. In Chapter 5 , we bridged the gap between abstract specifications expressed in the xMAS language and RTL implementations made according to xMAS specifications. The method that we introduced in the chapter automatically verifies the correctness of a given RTL implementation by turning the respective xMAS specification into an RTL Finite State Machine and checking the inclusion of the implementation's traces into the specification's traces. This answered the research question RQ4.

### 6.2 Future Work

Wouda et al. introduced a transfer islands optimization for xMAS [WJS15], which can be used within the xMAS reachability analysis technique. However, Wouda et al. do not evaluate the effect of their optimization, which leaves the question of whether transfer islands improve the performance of reachability analysis open. The Kripke Structure semantics, which we introduced in Chapter 2 can be used to study the impact of the transfer islands optimization on the xMAS reachability analysis. The KS semantics is helpful in that case because Wouda's verification tool based on transfer islands generates SMV models, and the KS semantics can be directly translated into SMV as well.

Finite State Machines in xMAS discussed in Chapter 3 have a limitation - within a single transition, it is only possible to read from (write to) a single channel. Allowing simultaneous multiple reads and writes within a single transition would make xMAS Finite State Machines more compact, impacting the overall sizes of xMAS networks positively.

In Chapter 4, consider the problem of false deadlocks in the state-of-the-art SATbased xMAS liveness verification technique. To address the issue, Chatterjee and Kishinevsky [CK12] proposed flow invariants that approximate the reachable state space. The flow invariants provide completeness only to xMAS networks that neither contain combinatorial cycles nor FSMs. We proposed two approaches to fix the incompleteness problem. Even though with our approaches, completeness is achieved even for xMAS networks with FSMs or cycles, scalability is an issue. It would be interesting to work towards more efficient SMT-encoding used in our approaches. A completely new, more scalable approach would also be beneficial.

The method for relating xMAS specifications and RTL implementations which we introduced in Chapter 5 relies on an action-based representation of xMAS networks. An obvious next step is to present the action-based semantics used for the said method formally and show the correspondence between this action-based semantics
and our Kripke Structure semantics. For proving the correspondence between the action-based and the state-based semantics, work of Willemse et al. [RSW12] can be used. Additionally, the action-based semantics can help to establish the action-based xMAS deadlock property, equivalent to the LTL one.

## Bibliography

[BK08] C. Baier and J.-P. Katoen. Principles of Model Checking. The MIT Press, 2008. isbn: 978-0-262-02649-9.
[Ber07] G. Berry. "SCADE: Synchronous Design and Validation of Embedded Control Software". In: Aug. 2007, pp. 19-33. noi: 10.1007/978-1-4020-6254-4_2.
[BHM21] A. Biere, M. Heule, and H. van Maaren. Handbook of satisfiability. English. Second edition. Amsterdam: IOS Press, 2021. url: http://ebooks.iospress. nl/isbn/978-1-64368-161-0.
[Bou+18] A. Bouzafour et al. "Model-Checking Synthesizable SystemVerilog Descriptions of Asynchronous Circuits". In: 2018 24th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC). 2018, pp. 3442. doi: 10.1109/ASYNC.2018.00021.
[BSY15] F. Burns, D. Sokolov, and A. Yakovlev. "GALS Synthesis and Verification for xMAS Models". In: Proceedings of the 2015 Design, Automation E Test in Europe Conference \& Exhibition. DATE '15. San Jose, CA, USA: EDA Consortium, 2015, pp. 1419-1424. Dor: 10.7873/DATE.2015.0063.
[BSY17] F. Burns, D. Sokolov, and A. Yakovlev. "A Structured Visual Approach to GALS Modeling and Verification of Communication Circuits". In: IEEE Trans. on CAD of Integrated Circuits and Systems 36.6 (2017), pp. 938-951. doi: 10.1109/TCAD.2016.2611508.
[Cav+14] R. Cavada et al. "The nuXmv symbolic model checker". In: Computer Aided Verification. Springer. 2014, pp. 334-342.
[CK10] S. Chatterjee and M. Kishinevsky. "Automatic Generation of Inductive Invariants from High-Level Microarchitectural Models of Communication Fabrics". en. In: Computer Aided Verification. Springer, Berlin, Heidelberg, 2010, pp. 321-338. doi: 10.1007/978-3-642-14295-6_29.
[CK12] S. Chatterjee and M. Kishinevsky. "Automatic Generation of Inductive Invariants from High-Level Microarchitectural Models of Communication Fabrics". en. In: Formal Methods in System Design 40.2 (2012), pp. 147169. Doi: 10.1007/s10703-011-0134-0.
[CKO10] S. Chatterjee, M. Kishinevsky, and U. Y. Ogras. "Quick Formal Modeling of Communication Fabrics to Enable Verification". In: 2010 IEEE International High Level Design Validation and Test Workshop (HLDVT). 2010, pp. 42-49. дoi: 10.1109/HLDVT.2010.5496662.
[CKO12] S. Chatterjee, M. Kishinevsky, and U. Y. Ogras. "xMAS: Quick Formal Modeling of Communication Fabrics to Enable Verification". In: IEEE Design Test of Computers 29.3 (2012), pp. 80-88. doi: 10.1109/MDT.2012. 2205998.
[Cim+13] A. Cimatti et al. "The MathSAT5 SMT Solver". In: Tools and Algorithms for the Construction and Analysis of Systems. Ed. by Nir Piterman and Scott A. Smolka. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 93-107. isbn: 978-3-642-36742-7.
[Cla+08] K. Claessen et al. "SAT-solving in practice". In: June 2008, pp. 61-67. doi: 10.1109/WODES.2008.4605923.
[CGL96] E. Clarke, O. Grumberg, and D. Long. "Model Checking". In: International Summer School on Deductive Program Design, 1994. Springer-Verlag Nato Asi, Series F, Vol. 152. 1996.
[Cru91a] R.L. Cruz. "A calculus for network delay. I. Network elements in isolation". In: IEEE Transactions on Information Theory 37.1 (1991), pp. 114-131. Dor: 10.1109/18.61109.
[Cru91b] R.L. Cruz. "A calculus for network delay. II. Network analysis". In: IEEE Transactions on Information Theory 37.1 (1991), pp. 132-141. Dor: 10.1109/ 18.61110.
[DKB17] S. Das, C. Karfa, and S. Biswas. "xMAS Based Accurate Modeling and Progress Verification of NoCs". en. In: VLSI Design and Test. Springer, Singapore, 2017, pp. 792-804. dor: 10.1007/978-981-10-7470-7_74.
[DSC98] D. Déharbe, S. Shankar, and E. Clarke. "Model Checking VHDL with CV". In: Nov. 1998, pp. 508-514. dor: 10.1007/3-540-49519-3_33.
[FKS19] A. Fedotov, J.J.A. Keiren, and J. Schmaltz. Sound idle and block equations for finite state machines in $x M A S$. English. Computer science reports. Technische Universiteit Eindhoven, Nov. 2019.
[FKS20] A. Fedotov, J.J.A. Keiren, and J. Schmaltz. "Effective System Level Liveness Verification". In: Proceedings of the 20th Conference on Formal Methods in Computer-Aided Design - FMCAD 2020. TU Wien Academic Press, 2020, pp. 7-15. doi: 10.34727/2020/isbn.978-3-85448-042-6_7.
[FS18] 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). 2018, pp. 1568-1573. dor: 10.23919/DATE.2018.8342265.
[FS21] A. Fedotov and J. Schmaltz. MaDL Design and Verification Tools. 2021. url: https://github.com/MaDL-DVT/madl-dvt.
[GS13] B. van Gastel and J. Schmaltz. "A Formalisation of xMAS". In: Electronic Proceedings in Theoretical Computer Science 114 (2013), pp. 111-126. dor: 10.4204/EPTCS.114.9. arXiv: 1304.7862.
[GVS14] 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 (VLSISoC). 2014, pp. 1-6. doi: 10.1109/VLSI-SoC.2014.7004168.
[GCK11] A. Gotmanov, S. Chatterjee, and M. Kishinevsky. "Verifying DeadlockFreedom of Communication Fabrics". en. In: Verification, Model Checking,
and Abstract Interpretation. Springer, Berlin, Heidelberg, 2011, pp. 214231. Dor: 10.1007/978-3-642-18275-4_16.
[HH12] D. Harris and S. Harris. "Digital Design and Computer Architecture: Second Edition". In: Digital Design and Computer Architecture: Second Edition (July 2012), pp. 1-690.
[HBS11] D. Holcomb, B. Brady, and S. Seshia. "Abstraction-based performance verification of NoCs". In: 2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC). 2011, pp. 492-497.
[HS14] D. Holcomb and S. Seshia. "Compositional Performance Verification of Network-on-Chip Designs". In: IEEE Trans. on CAD of Integrated Circuits and Systems 33.9 (2014), pp. 1370-1383. Doi: 10.1109/TCAD.2014.2331342.
[Hol+12] D. Holcomb et al. "Compositional performance verification of NoC designs". In: Tenth ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMCODE2012). 2012, pp. 1-10. Dor: 10.1109/ MEMCOD.2012.6292294.
[Irf+16] A. Irfan et al. "Verilog2SMV: A tool for word-level verification". In: 2016 Design, Automation Test in Europe Conference Exhibition (DATE). 2016, pp. 1156-1159.
[JS13] 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). 2013, pp. 57-64.
[JS14] S.J.C. Joosten and J. Schmaltz. "Scalable Liveness Verification for Communication Fabrics". In: Proceedings of the Conference on Design, Automation $\mathcal{E}$ Test in Europe. DATE '14. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2014, 113:1-113:6. Isbn: 978-3-9815370-2-4. URL: http://dl.acm.org/citation.cfm?id=2616606.2616745 (visited on 05/23/2019).
[JS15] 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. 2015, pp. 1413-1418. noi: 10.7873/DATE.2015.0616.
[KG02] C. Kern and M. Greenstreet. "Formal Verification In Hardware Design: A Survey". In: ACM Transactions on Design Automation of Electronic Systems 4 (Dec. 2002). DoI: 10.1145/307988.307989.
[KRŠ06] T. Kratochvíla, V. Rehak, and D. Šafránek. "Formal Verification of a FIFO Component in Design of Network Monitoring Hardware". In: CESNET Conference 2006-1st CESNET Conference on Advanced Communications and Grids, Proceedings (Jan. 2006), pp. 151-160.
[LZ18] Z. Lu and X. Zhao. "xMAS-Based QoS Analysis Methodology". In: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 37.2 (2018), pp. 364-377. dor: 10.1109/TCAD.2017.2706561.
[McM03] K.L. McMillan. "Interpolation and SAT-Based Model Checking". In: Computer Aided Verification. Ed. by Warren A. Hunt and Fabio Somenzi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 1-13. Isbn: 978-3-540-45069-6.
[MB08] L. de Moura and N. Bjørner. "Z3: An Efficient SMT Solver". en. In: Tools and Algorithms for the Construction and Analysis of Systems. Ed. by C. R. Ramakrishnan and J. Rehof. LNCS. Berlin, Heidelberg: Springer, 2008, pp. 337-340. doi: 10.1007/978-3-540-78800-3_24.
[RB12] S. Ray and R.K. Brayton. "Scalable progress verification in credit-based flow-control systems". In: DATE. Ed. by W. Rosenstiel and L. Thiele. IEEE, 2012, pp. 905-910. Isbn: 978-1-4577-2145-8.
[RSW12] M.A. Reniers, R. Schoren, and T.A.C. Willemse. "Results on Embeddings Between State-Based and Event-Based Systems". In: The Computer Journal 57.1 (Dec. 2012), pp. 73-92. doi: 10.1093/comjnl/bxs156. eprint: https: //academic.oup.com/comjnl/article-pdf/57/1/73/1141835/bxs156.pdf.
[SGG08] A. Silberschatz, P.B. Galvin, and G. Gagne. Operating system concepts. English. 8th ed. Hoboken, N.J.: J. Wiley \& Sons, 2008. url: https://www. safaribooksonline.com/library/view/-/9780470128725/?ar.
[Tar20] V. Taraate. SystemVerilog for Hardware Description: RTL Design and Verification. Jan. 2020. Doi: 10.1007/978-981-15-4405-7.
[VJS13] F. Verbeek, S.J.C. Joosten, and J. Schmaltz. "Formal Deadlock Verification for Click Circuits". In: Asynchronous Circuits and Systems (ASYNC), 2013 IEEE 19th International Symposium on. IEEE. 2013, pp. 183-190.
[VS11] F. Verbeek and J. Schmaltz. "Hunting Deadlocks Efficiently in Microarchitectural Models of Communication Fabrics". In: Proceedings of the International Conference on Formal Methods in Computer-Aided Design. FMCAD '11. Austin, TX: FMCAD Inc, 2011, pp. 223-231. isbn: 978-0-9835678-1-3. URL: http://dl.acm.org/citation.cfm? $\mathrm{id}=2157654.2157688$ (visited on 05/14/2019).
[VS12] 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). 2012, pp. 25-32. noi: 10.1109/ HLDVT.2012.6418239.
[Ver+16] F. Verbeek et al. "ADVOCAT: Automated Deadlock Verification for onChip Cache Coherence and Interconnects". In: 2016 Design, Automation Test in Europe Conference Exhibition (DATE). 2016, pp. 1640-1645.
[Ver+17] F. Verbeek et al. "Deadlock Verification of Cache Coherence Protocols and Communication Fabrics". In: IEEE Transactions on Computers 66.2 (2017), pp. 272-284. doi: 10.1109/TC.2016.2584060.
[WJS15] 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. Washington, DC, USA: IEEE Computer Society, 2015, pp. 198-207. Dor: 10.1109/ MEMCOD.2015.7340487.
[XZ15] Z. Xueqian and L. Zhonghai. "Heuristics-Aided Tightness Evaluation of Analytical Bounds in Networks-on-Chip". In: IEEE Trans. on CAD of Integrated Circuits and Systems 34.6 (2015), pp. 986-999. doi: 10.1109/ TCAD.2015.2402176.
[YGL00] A. Yakovlev, L. Gomes, and L. Lavagno. Hardware Design and Petri Nets. Jan. 2000. Doi: 10.1007/978-1-4757-3143-9.
[ZL13] X. Zhao and Z. Lu. "Per-flow delay bound analysis based on a formalized microarchitectural model". In: 2013 Seventh IEEE/ACM International Symposium on Networks-on-Chip (NoCS). 2013, pp. 1-8. dor: 10.1109/NoCS. 2013.6558411.

## Summary

## Verification Techniques for xMAS

Computers and related hardware are essential in the modern world. It is crucial that new hardware functions correctly. Conventionally, validation of new hardware designs is done using simulation and testing. The latter has a drawback as full coverage of all possible behaviors is impossible for complex hardware designs. On the other hand, Formal Verification covers all possible behaviors as it proves the correctness of new hardware designs using the formal methods of mathematics. However, Formal Verification is not a panacea as it is a challenge to scale it to the system level. In this thesis, among other things, we scaled Formal Verification to the system level.
xMAS is a language designed for convenient modeling and scalable formal verification of hardware. This thesis contains work on formal verification techniques in xMAS and related topics.

Although there exist xMAS verification techniques that rely on a state-based representation of xMAS [WJS15], a state-based semantics is not described in the literature. In Chapter 2, we formulated the semantics of the xMAS language in terms of Kripke Structures and proved its correctness. It serves as a theoretical basis for the subsequent work and fills the gap in the existing literature.

In Chapter 3, we demonstrated that the liveness verification technique for the xMAS Finite State Machine extension introduced by Verbeek et al. [Ver+17] is unsound by providing a counter-example. In the same chapter, we provided our own liveness verification approach for xMAS with FSMs and proved that it is sound. In addition, we showed that using xMAS with FSMs in combination with our verification technique can allow liveness verification at the system level.

The state-of-the-art liveness verification technique for xMAS is prone to reporting spurious deadlocks. In Chapter 4, we introduced two approaches to solve the spurious deadlock issue. One approach is based on an SMT-encoding of $k$-step backward reachability of an initial state from the deadlock state. Another approach uses interpolation to tackle the reachability problem mentioned above. In the same chapter, we prove the correctness of both approaches and evaluate their performance.

Hardware is implemented based on high-level designs. An important question is whether the implementation conforms to the high-level design, which is usually called the specification. In Chapter 5, we introduced a novel method that, given an abstract xMAS specification, automatically checks the correctness of RTL implementations made according to the xMAS specification. The method takes an xMAS specification and the respective RTL implementation as input. It then turns the specification into an RTL Finite State Machine and checks that all the traces produced by the implementation are included in the traces of the specification.

## Samenvatting

## Verificatietechnieken voor xMAS

Computers en computerhardware zijn van groot belang in de moderne wereld. Het is cruciaal dat nieuwe hardware correct werkt. Normaal gesproken wordt validatie van nieuwe hardware gedaan met behulp van simulatie en testen. Simuleren en testen hebben een nadeel: het volledig afdekken van alle mogelijke gedragingen is niet haalbaar voor complexe hardware-ontwerpen. Formele verificatie lost dit probleem op. Het bewijst de juistheid van nieuwe hardware-ontwerpen met behulp van formele, wiskundige methoden. Formele verificatie is echter niet in alle gevallen een oplossing, omdat het lastig is om formele verificatie naar systeemniveau te schalen.
xMAS is een taal voor het modelleren en formeel verifiëren van hardware. Dit proefschrift doet onderzoeken naar de formele verificatietechnieken van de liveness eigenschap in xMAS en onderwerpen die hieraan gerelateerd zijn.

In Hoofdstuk 2 hebben we de semantiek van de xMAS-taal geformuleerd in termen van Kripke-structuren en de juistheid van deze semantiek bewezen. Dit hoofdstuk dient als theoretische basis voor het daaropvolgende onderzoek en vult het gat in de bestaande literatuur met betrekking tot dit onderwerp op.

In Hoofdstuk 3 hebben we aangetoond dat de verificatietechniek voor de liveness eigenschap voor de xMAS Finite State Machine-extensie, geïntroduceerd door Verbeek et al. [WJS15], incorrect is. Dit wordt bewezen door een tegenvoorbeeld te geven. In hetzelfde hoofdstuk introduceren we onze eigen benadering van verificatie van de liveness eigenshap voor xMAS met FSM's en hebben we bewezen dat deze correct is. Daarnaast hebben we laten zien dat het gebruik van xMAS met FSM's in combinatie met onze verificatietechniek de liveness eigenshappen verificatie op systeemniveau mogelijk maakt.

De verificatietechniek uit Hoofdstuk 3 is gevoelig voor het melden van valse deadlocks. In Hoofdstuk 4 hebben we twee benaderingen geïntroduceerd om het valse deadlock-probleem op te lossen en hebben we de correctheid van beide benaderingen bewezen. We evalueerden ook de prestaties van de twee benaderingen.

In Hoofdstuk 5 hebben we een nieuwe methode geïntroduceerd die, op basis van

Samenvatting
de abstracte xMAS-specificatie, automatisch de correctheid controleert van RTLimplementaties die zijn gemaakt volgens de xMAS-specificatie. De methode neemt een xMAS-specificatie en de RTL-implementatie als input. Het zet de specificatie vervolgens om in een RTL Finite State Machine en controleert of alle traces die door de implementatie worden geproduceerd, zijn opgenomen in de traces van de specificatie.

## Curriculum Vitae


#### Abstract

Alexander Fedotov was born in Perm, Russian Federation. After finishing school, he started the Mathematics and Computer Science bachelor's program at Perm National Research Polytechnic University. After obtaining the bachelor's degree, Alexander decided to get industrial experience, so he moved to Moscow and worked in Test Automation and Quality Assurance. Following that, he moved to Nijmegen, The Netherlands, to pursue a master's degree. He finished the Computing Science master's program at Radboud University Nijmegen. As part of his master's, Alexander did a research internship in the group of Frits Vaandrager. Subsequently, Alexander started as a Ph.D. candidate in the Formal Systems Analysis group at Eindhoven Technical University. The project concerned working on verification techniques associated with the xMAS language. The title of his dissertation is "Verification Techniques for xMAS ".


## Titles in the IPA Dissertation Series since 2019

S.M.J. de Putter. Verification of Concurrent Systems in a Model-Driven Engineering Workflow. Faculty of Mathematics and Computer Science, TU/e. 2019-01
S.M. Thaler. Automation for Information Security using Machine Learning. Faculty of Mathematics and Computer Science, TU/e. 2019-02

Ö. Babur. Model Analytics and Management. Faculty of Mathematics and Computer Science, TU/e. 2019-03
A. Afroozeh and A. Izmaylova. Practical General Top-down Parsers. Faculty of Science, UvA. 2019-04
S. Kisfaludi-Bak. ETH-Tight Algorithms for Geometric Network Problems. Faculty of Mathematics and Computer Science, TU/e. 2019-05
J. Moerman. Nominal Techniques and Black Box Testing for Automata Learning. Faculty of Science, Mathematics and Computer Science, RU. 2019-06
V. Bloemen. Strong Connectivity and Shortest Paths for Checking Models. Faculty of Electrical Engineering, Mathematics \& Computer Science, UT. 2019-07
T.H.A. Castermans. Algorithms for Visualization in Digital Humanities. Faculty of Mathematics and Computer Science, TU/e. 2019-08
W.M. Sonke. Algorithms for River Network Analysis. Faculty of Mathematics and Computer Science, TU/e. 2019-09
J.J.G. Meijer. Efficient Learning and Analysis of System Behavior. Faculty of Elec-
trical Engineering, Mathematics \& Computer Science, UT. 2019-10
P.R. Griffioen. A Unit-Aware Matrix Language and its Application in Control and Auditing. Faculty of Science, UvA. 2019-11
A.A. Sawant. The impact of API evolution on API consumers and how this can be affected by API producers and language designers. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2019-12
W.H.M. Oortwijn. Deductive Techniques for Model-Based Concurrency Verification. Faculty of Electrical Engineering, Mathematics \& Computer Science, UT. 2019-13
M.A. Cano Grijalba. Session-Based Concurrency: Between Operational and Declarative Views. Faculty of Science and Engineering, RUG. 2020-01
T.C. Nägele. CoHLA: Rapid Cosimulation Construction. Faculty of Science, Mathematics and Computer Science, RU. 2020-02
R.A. van Rozen. Languages of Games and Play: Automating Game Design E Enabling Live Programming. Faculty of Science, UvA. 2020-03
B. Changizi. Constraint-Based Analysis of Business Process Models. Faculty of Mathematics and Natural Sciences, UL. 2020-04
N. Naus. Assisting End Users in Workflow Systems. Faculty of Science, UU. 2020-05
J.J.H.M. Wulms. Stability of Geometric Algorithms. Faculty of Mathematics and Computer Science, TU/e. 2020-06
T.S. Neele. Reductions for Parity Games and Model Checking. Faculty of Mathematics and Computer Science, TU/e. 2020-07
P. van den Bos. Coverage and Games in Model-Based Testing. Faculty of Science, RU. 2020-08
M.F.M. Sondag. Algorithms for Coherent Rectangular Visualizations. Faculty of Mathematics and Computer Science, TU/e. 2020-09
D.Frumin. Concurrent Separation Logics for Safety, Refinement, and Security. Faculty of Science, Mathematics and Computer Science, RU. 2021-01
A. Bentkamp. Superposition for HigherOrder Logic. Faculty of Sciences, Department of Computer Science, VUA. 2021-02
P. Derakhshanfar. Carving Information Sources to Drive Search-based Crash Reproduction and Test Case Generation. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2021-03
K. Aslam. Deriving Behavioral Specifications of Industrial Software Components. Faculty of Mathematics and Computer Science, TU/e. 2021-04
W. Silva Torres. Supporting MultiDomain Model Management. Faculty of Mathematics and Computer Science, TU/e. 2021-05
A. Fedotov. Verification Techniques for $x M A S$. Faculty of Mathematics and Computer Science, TU/e. 2022-01


[^0]:    ${ }^{1}$ By the verification at the system level, we mean verification of whole systems and not just isolated parts of systems.

[^1]:    ${ }^{2}$ That is, hardware is modeled using the flow of signals between registers

[^2]:    ${ }^{3}$ VHDL stands for VHSIC Hardware Description Language, where VHSIC is for Very High-Speed Integrated Circuits Program.

[^3]:    ${ }^{1}$ Note, that in [WJS15] o.irdy is defined as $o . \operatorname{irdy}:=i_{0} . \operatorname{irdy} \vee i_{1} . \mathbf{i r d y}$, which is not correct, in case $u$ is unconstrained. The problem arises when, for example, $u=$ false, $i_{0}$.irdy $=$ true, and $i_{1}$.irdy $=$ false. Then, we have $o$. irdy $=$ true, $i_{0} \cdot \operatorname{trdy}=$ false, $i_{1} \cdot \operatorname{trdy}=$ false, and $o$. data $=$ false, which can lead to a data transfer through $o$, while neither of the inputs is ready, and the data at the output is undefined.

[^4]:    ${ }^{2}$ Consistency of $c$ can be checked using the data propagation defined by Wouda et al. [WJS15]

[^5]:    ${ }^{1}$ That is, the approach might yield unreachable deadlock states.

[^6]:    ${ }^{2}$ According to Gotmanov et al. [GCK11].

[^7]:    ${ }^{3}$ That is, the function selects a transition with the corresponding input channel among enabled transitions such that every transition and input channel pair is selected infinitely often.

[^8]:    ${ }^{4}$ The expression can easily be expanded to a boolean condition.

[^9]:    ${ }^{1}$ https://github.com/MaDL-DVT/madl-dvt

[^10]:    ${ }^{2}$ That is, sources, sinks, queues, functions, forks, switches, joins, and merges

