Reordering and Simulation in Concurrent Systems

Dipl.-Ing. Christoph Baumann

Institute for Computer Architecture und Parallel Computing
Department of Computer Science, Saarland University

September 19, 2012

TECHNICAL REPORT

Abstract

In this document we present intermediate results of our ongoing work in the field of the formal specification and verification of concurrent systems with shared memory. In particular here we focus on the specification of such systems and develop a generic framework to model their operational semantics. Every participant in the concurrent system is represented by an abstract state machine that can be instantiated individually, e.g., with instruction set architectures, device descriptions or higher-level language semantics. Machines of the system may access shared memory asynchronously and additionally communicate with each other using Moore and Mealy output signals. In order to justify the application of verification techniques on a higher level we provide a concurrent simulation theorem that allows transfering correctness properties to a lower-level specification of the overall system. The simulation theorem is based on a reordering theorem that reduces arbitrary system executions to orderly ones where we interleave machines executing blocks of steps containing at most one shared memory acccess. Memory safety is guaranteed by a simple ownership policy which all participants must be proven to obey.

1 Introduction

There is no such thing as sequentiality. In fact, as far as we can tell, the whole universe is inherently a parallel system. Galaxies, stars and planets are roaming space in parallel, steering each other by gravity and "communicating" by the emission and reflection of light. All lifeforms on Earth exist in parallel and influence each other, e.g. by producing sound waves, performing motions or applying force. On the gate level of a processor signals change their logical value in parallel and interact with each other through logic circuitry. Every electron of an atom is revolving around the atomic core in parallel with all other electrons. They are affecting each other by their electromagnetism. On any level of abstraction processes in the real world are running concurrently. Nevertheless in computer science (and possibly elsewhere) we are usually modelling this kind of concurrency in a different way.

Firstly, we look at systems in an isolated way. We encapsulate a certain number of components that make up the system we want to describe and separate it from its environment. The components can communicate with each other via shared variables or messages. Communication with the environment occurs only via defined input and output channels. Secondly, in case we do not speak about lockstep parallel systems, we interleave the steps of individual components in some arbitrary order. Thus we assume the rest of the system to pause during the step of one participant. What computer science has in common with the examples above is that systems are described at a level of detail that is appropriate to make meaningful statements about a system and to predict its behaviour.

Following these three principles seems to make it easier for the human mind to conceive the functionality of concurrent systems and to argue about them. One must however not forget that all these abstractions have prerequisites and rely on certain assumptions on lower-level properties of the system as well as the environment. For instance it is a common assumption that the environment does not interfere with the system via hidden channels. These include also physical side-channels and we assume, for instance, that bits in our system are not flipped due to electromagnetic radiation, or that the computer system running programs we are examining is not being destroyed during execution. A specification of a system is therefore only complete if one expresses also the underlying assumptions on the behaviour of its environment. Moreover the interleaving of component steps usually assumes some sequential consistent shared memory, which is not provided by default in modern computer systems. Configuring caches and address translation in a proper way and using a certain programming discipline are means to reduce caches, memory management units and store buffers. To ensure sequential consistency of the shared memory one additionally needs the absence of data races on shared variables. Memory safety conditions must guarantee that atomic updates of shared memory stay atomic until the lowest layer of specification.

In general all abstractions we make should be justified. The best justification we can have is a formal simulation proof between the abstract system and its low-level implementation. This document provides (1) a framework to model concurrent systems with shared memory and communication signals at any level of abstraction (2) a simulation theorem to relate different abstraction layers (3) a general order reduction theorem for concurrent systems that enables applying the simulation theorem (4) verification conditions for memory safety (5) a complete correctness proof of our approach including the aforementioned theorems in paper and pencil mathematics.

1.1 Overview

As motivated above we want to look at asynchronous concurrent systems on different layers of abstraction and connect these layers formally using simulation relations, or refinement mappings respectively. This requires defining formal semantics for every layer and we want to be able to do so in a uniform manner. To this end we introduce a framework that allows to define instances of what we call a *Concurrent System with Shared Memory and Ownership*, abbreviated by the acronym *Cosmos*. A *Cosmos* model consists essentially of a number of infinite state machines which can be instantiated individually, and a shared memory. What is special is additional specification components representing the dynamic *ownership* state.

We use a simple ownership model to keep track which memory addresses are owned by a particular machine in the system and which addresses are shared by all participating machines. We specify ownership invariants and a memory access policy based on ownership that provably guarantees memory safety if all machines adhere to it. Moreover we can use ownership later on to prove commutativity of certain machine steps. Thus it is a cornerstone principle of our approach. In fact every instantiation of a machine in the *Cosmos* model must come with a program logic containing individual safety conditions for operations of that machine. We require that safe operations preserve ownership invariants and perform only sound memory accesses wrt. the ownership model, i.e., they must obey the ownership policy. This policy demands, e.g., that one machine does not write addresses owned by other machines or that shared memory may only be accessed by steps that are recognized by the program logic as so-called $\mathcal{IO}$ *steps*, where $\mathcal{IO}$ stands for input/output. Every machine instantiation must specify which of its state transitions are considered $\mathcal{IO}$[1]. Intuitively an $\mathcal{IO}$ step represents a communication between machines in the *Cosmos* model and is usually implemented via shared memory.

In addition to shared memory we allow the machines also to have Moore and Mealy output signals for communication with each other as well as external inputs. These signals are useful to model interrupt lines in case of ISA models but also message protocols in case of higher-level models. Activations of internal outputs and reactions to internal inputs are considered $\mathcal{IO}$ steps. External inputs provide an interface with the environment but also means to model nondeterminism. Thus by definition we can require all machines in a *Cosmos* model to be deterministic automata, and their transition functions rely only on (a portion of) the system state as well as internal and external inputs. We execute a *Cosmos* model consuming a sequence of external inputs and an explicit schedule determining the interleaving of machines.

Now we can have different instantiations of *Cosmos* models describing the same system at different levels of detail. For instance we might look at a computer system were C programs are executed concurrently on multiple processor cores. In the real world these programs are compiled to a certain multicore

[1]The individual definition of $\mathcal{IO}$ must be however consistent with memory safety conditions

instruction set architecture (ISA) which we can also model as a *Cosmos* (possibly a reduced version under certain software conditions). Each program is compiled separately and there exists a local simulation relation and a theorem that links C and ISA execution on one machine. It would be desirable to compose the local simulation theorems to one global simulation theorem spanning all machines in the *Cosmos*. Naturally this would base on the local theorems stating that for every particular machine its local simulation relation is preserved by executions of the overall system. However we can not prove such a theorem for arbitrary interleavings of machine steps because a simulation relation need not generally hold for all states of an execution on the abstract level and usually only holds for a few steps on the concrete level where the steps are refined[2]. In fact we need to assume schedules on the low level where machines are executing blocks of steps from one consistency point (a configuration that is consistent to/simulates an abstract one) until the next one. Furthermore we cannot just combine arbitrary programs in a concurrent fashion. There is a number of requirements on the nature of the simulation theorems under consideration and the notion of the shared resources on both levels. We list all the necessary assumptions including memory safety on the abstract level and prove the global *Cosmos* model simulation theorem based on the correctness of the local simulations which have to be verified individually. The assumption of block scheduling is justified by an order reduction theorem.

In order reduction we reduce the number of possible interleavings of different processor steps. The core argument to enable reduction is that the effect of a safe concurrent system execution does only partly rely on the scheduling on machine steps. In fact it relies only on the external input sequence and the schedule of $\mathcal{IO}$ *steps*. All non-$\mathcal{IO}$ actions can be reordered arbitrarily as long as program order is maintained. The reordering preserves the effect of the original trace because non-$\mathcal{IO}$ steps only modify local information. This can be enforced by the ownership policy and is the main lemma of the proof. Instead of non-$\mathcal{IO}$ operations we also speak of local steps and we call the configuration from which an $\mathcal{IO}$ step origins $\mathcal{IO}$ point.

For the reduction theorem we reduce arbitrary interleavings to so-called $\mathcal{IO}$ *block schedules*, which are interleavings of blocks of steps of same machine. Each $\mathcal{IO}$ block usually contains exactly one $\mathcal{IO}$ step at the beginning of the block. We construct the blocks in a way that we are later on able to apply the local simulation theorems on them. Here we assume for simplicity that all $\mathcal{IO}$ points are consistency points[3]. The reduction requires that all $\mathcal{IO}$ block schedules are proven to obey the ownership policy and preserve ownership invariants. Then this also holds for all arbitrary schedules and for each of them we can find a consistent $\mathcal{IO}$ block schedule. Additionally we prove that the reduction preserves safety properties. We do not consider liveness in this report. Apart from memory safety conditions the theorem also relies on certain constraints on the *Cosmos* model components and parameters that must be fulfilled by all instantiations.

The subsequent sections are structured as follows. In Section 2 we introduce the *Cosmos* model. For clarity we first develop a simplified version which

[2]The only exception to this is one-step simulations where one abstact step maps to a single concrete one.

[3]This assumption can be relaxed for more general simulations relations. A necessary precondition is that between two $\mathcal{IO}$ points of the same machine it passes at least one consistency point.

is extended in Section 3 with additional communication channels. We procede in Section 4 to define the order reduction theorem and prove it. In Section 6 we give a semantics for a machine which only executes $\mathcal{IO}$ blocks. In the sixth Section we first set up a framework to define local simulation theorems in a unified way. We then list the requirements needed to combine all local simulations into a global one. We conclude this report with the proof of that simulation theorem.

1.2 Related Work

The first formal approaches for the specification and verification of sequential programs date back to the 60s [Flo67, Hoa69]. In the subsequent decades myriads of different tools and languages have been developed to tackle the problem in a more efficient way and to apply techniques to more and complex systems. The most challenging targets for formal methods may be operating systems, and recent history has seen several attempts on the verification of sequential OS microkernels [Kle09]. The approach of the Verisoft project [AHL+09] however comes probably closest to our vision of modelling computer systems on various abstraction levels that are linked by simulation theorems, thus enabling *pervasive* verification. There a semantics stack was developed spanning abtraction layers from the gate level hardware description up to the user application view. Nevertheless only a sequential, academic system was considered. The succeeding Verisoft XT project aimed at transferring this approach to the real world and developed tools and methods for the formal verification of inductrial concurrent operating systems and hypervisors [CAB+09, LS09, BBBB09]. The theory presented in this report was conceived in an effort to justify the specification and verification approach used in the Verisoft XT project, were the automated verifier VCC [CDH+09] was employed to prove code correctness on the concurrent C level.

The VCC tool is just milestone in the long history of specification and verification methods for concurrent systems [Lam93]. As early as 1975 Ashcroft [Ash75] and Owicki/Gries [OG76] extended Floyd's and Hoare's methods to the concurrent case. The approach was based on assertions that were added to programs in order to show the preservation of global invariants. Instead of using programs to specify other programs Lamport suggested to use state machines and mathematical formulas as a universal specification language. His Temporal Logic of Actions (TLA) [Lam94] allows to define safety and liveness properties of programs and algorithms in a uniform and well-defined way. Systems can be composed by disjunction of TLA specifications [AL95] and specified at different levels of abstraction. In case a refinement mapping exists between two layers it can be shown that one specification implements the other by showing an implication in TLA [AL91]. However, this simulation approach seems only to be suitable for the program refinement or simple simulation theorems where the concrete specification makes steps in parallel with the abstract one (which might stutter). For general simulation theorems where n abstract steps simulate m concrete ones we can not use TLA.

In 1978 Hoare introduced an algebraic method to model communicating sequential processes [Hoa78]. Subsequently the language he proposed evolved into the well-known CSP process algebra [Hoa85] and a whole community following his his approach was formed. Several other process algebras followed in the wake of CSP, most prominently Milner's π-calculus. However this methodology seems only appropriate to model systems at a very abstract level, e.g., to specify algorithms and protocols. Modelling complex systems by algebraic structures and their transformations is at best messy, if not infeasible.

Another approach to concurrent system specification are I/O Automata as introduced by Lynch and Tuttle [LT87]. I/O Automata are basically infinite state machines which are characterized by actions they can perform. Internal actions are distinguished from externally visible input and output actions that are employed for communication between automata. Lynch uses these automata to model sequential, parallel and asynchronously concurrent shared memory systems and algorithms [Lyn96]. It is possible to compose several automata into a bigger one given certain composability requirements on their actions. Moreover it was shown how to prove simulation theorems between automata. However it was not treated how the automatas working on shared variables should be composed (an overall automaton containing all subautomata was assumed) nor how simulation is maintained by composition of I/O automata. Finally the need to list all actions of an automaton and divide them into input, output and internal ones appears to be feasible for small-scale systems and algorithms. Modelling a realistic system like a modern instruction set architecture working massively on shared memory as an I/O automaton, in contrast, seems to be a rather tedious task. Nevertheless we feel that using composable state machines is the the right methodology in order to obtain a formally linked semantics stack. In this way we are inspired by the work of Lynch and Lamport. Gurevich also followed the idea of modelling systems as *Abstract State Machines* (ASMs) [Gur00, BG03] and a programming language exists to specify and execute such models [GRS05]. However the ASM approach does not support asynchronous concurrency [Gur04].

As to the best of our knowledge there is no prominent formalism in literature general enough to serve our purposes we define our own framework for modelling concurrent systems using well-known concepts like automata theory and simulation relations. Concerning (order) reduction a multitude of related work exists. Lipton was the first to describe how several steps of an operation could be aggregated into a monolithic atomic action, given that it contained only one access to shared memory [Lip75]. His results were improved by Doeppner [Doe77] and Lamport [LS89], who showed that safety properties are preserved by the reduction. Later Lamport [Lam90] specialized his results for systems with explicit message passing, ruling out interrupts along the way. Generally these approaches had in common that a particular program is examined and the specific operation with its sub-steps that shall be reduced is perfectly well-defined. In our case we would like to show a general reduction theorem for arbitrarily instantiated *Cosmos* models using the memory safety of all traces. In order to do so we have to reorder local steps of machines forming a schedule of $\mathcal{IO}$ blocks. The structure of these blocks is only constrained in that they have to start in an $\mathcal{IO}$ block and depending on the inputs the length of a block may vary even if it starts in the same configuration. However in the approaches above one must exactly identify the consecutive steps to be reduced into one operation and one must define predicates to tell, e.g., whether one is within this operation or not. Thus it is hard (though not impossible) to fit the aforementioned program-based reordering approach to our execution-based scenario where basically all local steps may be reordered.

Moreover the reduction theorems presented above require commutativity for steps of a certain program unconditionally and they preserve only global safety properties that may not be influenced by the local reordered steps. Local properties of the program are thus not transferred down. On the other hand in our case reordering can only be done if the steps are local and memory-safe. As memory safety depends on global (the ownership state) and local information (the program state), this property cannot be transferred by the existing reduction theorems. This also holds for the work of Cohen and Lamport [CL98] on reduction in TLA. The latter however introduced the preservation of liveness properties which we do not touch within the frame of this report. It should be also noted that commutativity arguments like the ones we use in the proof of our reduction theorem are closely related to the notion of non-interference in information flow theory [GM82, Rus92]. Simply put, it is argued that an action does not interfere with a certain domain if it can be pruned from a computation without changing results visible in that domain. If this is the case it can be placed elsewhere in the computation, yielding commutativity.

Besides the classic theorems there is more recent work on order reduction. Several specific approaches have been proposed to employ reduction in model checking in order to tackle the problem of state explosion [FQ04, FFQ05, FG05]. Also the formal verification tool VCC [CDH$^+$09] used in the Verisoft XT project relies on order reduction. Threads are verified as if they run sequentially and concurrent actions of other threads are only interleaved before shared memory accesses. A reduction theorem to justify this "coarse scheduling" was proposed and proven [CMST09], however it only covered the existence of a safe equivalent coarse schedule for a given arbitrary safe schedule. The soundness of the methodology, i.e., that also for every unsafe trace there exists an unsafe coarse schedule, remained open. Nevertheless this approach was the inspiration for the reduction theorem presented below. In fact, the VCC reduction theorem can be derived from an instantiation of our general reduction theorem. In contrast we use a different ownership model then the VCC tool. There ownership is based on typed objects while in our model ownership is based on memory addresses. Our ownership model is a simplified version of the one presented by Cohen and Schirmer in order to justify sequential consistent memory for a memory system implementing Total Store Order [CMST09]. Ownership of memory and ownership transfer is not a new concept. It is also used in Concurrent Separation Logic [Bro04].

1.3 Notation and Conventions

In the scope of this document we use the following notation and conventions.

1.3.1 Types and Records

Types are usually identified by blackboard bold or blackletter font in case their names consist of a single letter. The natural numbers $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$ contain zero. $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$ denotes the strictly positive natural numbers. $\mathbb{N}_i \subset \mathbb{N}$ with $i > 0$ and $\#\mathbb{N}_i = i$ defines the set of the i smallest natural numbers.

$$\mathbb{N}_i = \{0, \ldots, i - 1\}$$

The set of Boolean values $\{0, 1\}$ is represented by $\mathbb{B}$. We define new types using the standard mathematical notation where $\times$ creates tuples and $\rightarrow$ creates total functions. Partial functions are identified by a $\rightharpoonup$ arrow in their type signature. We treat record types as n-tuples where each component c_i can have a different type t_i. Let $r \in R$ be such a record. We can declare it in two equivalent ways:

$$r = (r.c_0 : t_0,\ \dots\ , r.c_{n-1} : t_{n-1}) \qquad R = \{c_0 : t_0\ ;\ \dots\ ;\ c_{n-1} : t_{n-1}\}$$

Record component can have any type — also records — but we forbid recursive types here. Sometimes we use $c_i \subseteq t_i$ instead of $c_i : t_i$ to denote that c_i is a powerset of t_i do We introduce a handy record update notation that allows modifying single record components. Let $U = \bigcup_{i=0}^{m-1} id_i$ be a set of m different component identifiers, representing the components of record r to be updated, i.e., $\forall i, j < m.\ i \neq j \implies id_i \neq id_j$. and $\forall i < m \exists j < n.\ id_i = c_j$. The new values for the respective components shall be v_0 to v_{m-1}. Then the updated record r' is defined by:

$$\begin{gathered} r' = r[\![id_0 := v_0;\ id_1 := v_1;\ \dots\ ;\ id_{m-1} := v_{m-1}]\!] \\ \Updownarrow \\ \forall i < m.\ r'.id_i = v_i \wedge \forall c \notin U.\ r'.c = r.c \end{gathered}$$

1.3.2 Propositional Logic

Logical propositions contain conjunction $\wedge$, disjunction $\vee$, negation, implication $\implies$, equivalence $\iff$ and brackets. For negation literature knows several symbols.

$$/x \quad \equiv \quad {\sim}x \quad \equiv \quad \neg x \quad \equiv \quad \overline{x}$$

Here we will use mostly $/x$ and sometimes $\overline{x}$ where it saves brackets. Definitions and identity is denoted by $\equiv$. The priority order $\prec$ of logical operators used in this document is defined below from weakest to strongest binding.

$$\iff \ \prec \ \implies \ \prec \ \vee \ \prec \ \wedge \ \prec \ /$$

To display voluminous conjunctions and disjunctions in a clear and compact style we introduce a notation to combine all propositions p_i from a finite set P. For $P = \emptyset$ we have $\bigwedge P = 1$ and $\bigvee P = 0$. Otherwise:

$$\begin{aligned} \bigwedge P = \bigwedge \{\, p_0, p_1, \dots\ \} &\equiv ((p) \wedge \bigwedge(P \setminus \{p\})) \quad \text{s.t. } p = \epsilon P \\ \bigvee P = \bigvee \{\, p_0, p_1, \dots\ \} &\equiv ((p) \vee \bigvee(P \setminus \{p\})) \quad \text{s.t. } p = \epsilon P \end{aligned}$$

Here ϵ is Hilbert's choice operator that chooses one element out of a given set.

1.3.3 Set Notation

In general we use standard set theory, however we extend it by a few shorthand definition. The disjoint union of sets is denoted by $\uplus$. Let A, B, C be sets of the same type, then:

$$C = A \uplus B \iff C = A \cup B\ \wedge\ A \cap B = \emptyset$$

To express that two sets are equal for some element α, i.e. that α is either contained or not contained in *both* sets, we choose the following notation

$$A =_\alpha A' \;\equiv\; \alpha \in A \Leftrightarrow \alpha \in A'$$

If both sets are subset of some superset B, i.e., $A, A' \subseteq B$, we can easily show the property, that if A and A' agree on all elements of B, they must be equal.

$$(\forall \alpha \in B.\; A =_\alpha A') \Longleftrightarrow A = A'$$

1.3.4 Sequences

Moreover in this docoument we will deal excessively with computation sequences. An arbitrary lengthy sequence of elements $a^i \in \mathbb{A}$ has type $\mathbb{A}^*$ and is represented by $\underset{\rightarrow}{a}$.

$$\underset{\rightarrow}{a} : \mathbb{A}^* \iff \underset{\rightarrow}{a} = \varepsilon \;\vee\; \exists a^0 \in \mathbb{A}, \underset{\rightarrow}{a'} : \mathbb{A}^*.\; \underset{\rightarrow}{a} = a^0, \underset{\rightarrow}{a'}$$

Let ε be the empty sequence, then we define the length of sequences as follows.

$$|\varepsilon| = 0 \qquad\qquad |a^0, \underset{\rightarrow}{a'}| = |\underset{\rightarrow}{a'}| + 1$$

For manipulating sequences we define the function *pop* which removes the first i members of a sequence.

$$pop(\underset{\rightarrow}{a}, i) = \begin{cases} \underset{\rightarrow}{a} & : \quad i = 0 \\ pop(\underset{\rightarrow}{a'}, i-1) & : \quad i > 0 \wedge \underset{\rightarrow}{a} = a^0, \underset{\rightarrow}{a'} \\ \varepsilon & : \quad \text{otherwise} \end{cases}$$

Function *tail* yields the remainder after removing the head of a sequence.

$$tl(\underset{\rightarrow}{a}) = pop(\underset{\rightarrow}{a}, 1)$$

Furthermore we can number the members of a sequence in ascending order starting from the beginning.

$$\underset{\rightarrow}{a} = a^0, a^1, a^2, \ldots$$

Consequently we can select finite subsequences of a sequence via interval notation. First of all for $a \leq b$ we introduce the following integer intervals.

$$\begin{array}{lcl@{\qquad\qquad}lcl} [a:b] & \equiv & \{a, a+1, \ldots, b\} & [a:b) & \equiv & [a:b-1] \\ (a:b] & \equiv & [a+1:b] & (a:b) & \equiv & [a+1:b-1] \end{array}$$

Now subsequences of $\underset{\rightarrow}{x}$ are easily defined recursively. Let $0 \leq a \leq b$, then:

$$\underset{\rightarrow}{x}[a:b] \equiv \begin{cases} x^a & : \quad a = b \\ x^a, \underset{\rightarrow}{x}[a+1:b] & : \quad \text{otherwise} \end{cases}$$

For open intervals, e.g., $\underset{\rightarrow}{x}[a:b)$ the equivalent closed interval from above shall be used.

1.3.5 Computations

Computations are sequences of configurations from a state space $\mathbb{S}$. They rely on transition function δ that transform the state using inputs from some domain $\mathbb{I}$:

$$\delta : \mathbb{S} \times \mathbb{I} \to \mathbb{S}$$

A computation $\underrightarrow{s} : \mathbb{S}^*$ for input sequence $\underrightarrow{in} = in^0, in^1, in^2, \ldots : \mathbb{I}^*$ is represented by the shorthand $\underrightarrow{s}_{\delta,in}$.

$$\underrightarrow{s}_{\delta,in} \equiv s^0, s^1, s^2, \ldots \quad \text{s.t. } \forall j \geq 0.\ s^{j+1} = \delta(s^j, in^j)$$

To denote state transitions we us the following arrow notation for $i, n \in \mathbb{N}$.

$$s^i \longrightarrow^n_{\delta,in} s^{i+n} \;\equiv\; \begin{cases} s^{i+1} = \delta(s^i, in^i) \wedge s^{i+1} \longrightarrow^{n-1}_{\delta,in} s^{i+n} & : \quad n > 0 \\ 1 & : \quad n = 0 \end{cases}$$

It can be generalized for states $s, s' \in \mathbb{S}$ omitting the index i.

$$s \longrightarrow^n_{\delta,in} s' \;\equiv\; \exists \underrightarrow{a} : \mathbb{S}^*.\ a^0 \longrightarrow^n_{\delta,in} a^n \wedge s = a^0 \wedge s' = a^n$$

There are two special versions of this:

$$s \longrightarrow_{\delta,in} s' \equiv s \longrightarrow^1_{\delta,in} s' \qquad\qquad s \longrightarrow^*_{\delta,in} s' \equiv \exists n.\ s \longrightarrow^n_{\delta,in} s'$$

Given some particular input $x \in \mathbb{I}$ we also use $s \longrightarrow_{\delta,x} s'$ instead of $s' = \delta(s, x)$. Implicite conversion is possible between $n+1$-tuples of sequences and sequences of $n+1$-tuples with the same length $m+1$.

$$\begin{array}{ccc} (\underrightarrow{in_0}, \ldots, \underrightarrow{in_n}) & = & ([in_0^0, \ldots, in_0^m], \ldots, [in_n^0, \ldots, in_n^m]) \\ \updownarrow & & \updownarrow \\ \underrightarrow{(in_0, \ldots, in_n)} & = & (in_0^0, \ldots, in_n^0), \ldots, (in_0^m, \ldots, in_n^m) \end{array}$$

This is useful for transition functions that take more than one input. If in is for instance a triple (a, b, c) by the implicite conversion ($\updownarrow$) we can still use the arrow notation to denote an n-step transition from s to s' under the input sequences $\underrightarrow{a}$, $\underrightarrow{b}$, $\underrightarrow{c}$ and so forth. We write:

$$s \longrightarrow^n_{\delta,a,b,c,\ldots} s'$$

2 Concurrent Machine Model with Shared Memory and Ownership

In order to model multiprocessor systems later we first introduce a general model for machines that are concurrently accessing a shared memory. We speak of a **Co**ncurrent system with **s**hared **m**emory and **o**wner**s**hip (*Cosmos*). Accesses are governed by an ownership policy guaranteeing sound memory accesses i.e., the absence of data races on the shared memory. The ownership model builds on previous work by Cohen and Schirmer. There it was used to ensure sequential consistency for a shared memory system with store buffers (TSO). Here we use it to show a reordering theorem were arbitrary interleavings of machine steps are reordered into a coarse schedule of blocks of machine steps, were each block starts with a shared memory access and may be followed by local computations. In the model presented below each machine can be instantiated arbitrarily. However for each instantiation a program logic has to be provided in terms of safety conditions on the machine transitions in order to ensure memory soundness and maintain ownership invariants. We first define the *Cosmos* model in general where machines are only communicating via shared memory. Later on we extend the model with inputs, outputs and visible components. Afterwards we will formulate and proof the reordering theorem and instantiate the *Cosmos* model with a generic ISA model to obtain a Multiprocessor ISA Model with Ownership.

2.1 Configuration and Parameters

We consider a *Cosmos* model with type $\mathbb{C}$, which is a concurrent system of $np \in \mathbb{N}$ generic machines and a shared memory with the address range $\mathfrak{A}$. Note that on the top level we only consider the memory shared between the machines in the *Cosmos* model . Other communication must be handled internally.

A machine p is modelled as an Moore Automata without inputs or outputs for now as explained above. Machines could be devices, processors or even *Cosmos* models themselves. To develop a model as general as possible we parametrize it by a number of system parameters and types. These polymorphic types shall be identified by gothic font. Functions and values declared based on these types are also polymorphic and must be instantiated for a particular system. All components related to ownership or program logics are printed using calligraphic font.

A configuration $C \in \mathbb{C}$ has three components $(C.s, C.m, C.\mathcal{S})$:

- $C.s : \mathbb{N}_{np} \to \mathbb{S}$ - the state of each machine in the system, state $C.s[p]$ of machine p is a record with two components:
  - $C.s[p].\mathcal{O} \subseteq \mathfrak{A}$ - the set of addresses owned by machine p. Owned addresses may only be modified by their owner.
  - $C.s[p].c \in \mathfrak{C}_p$ - the configuration of machine p, where $\mathfrak{C}_p$ is an instantiable machine configuration type. Some components in $\mathfrak{C}_p$ may be defined to be visible to (but not modifiable by) other machines.
- $C.m : \mathbb{M}$ - the memory shared by all machines in the system, $\mathbb{M} = \mathfrak{A} \to \mathfrak{V}$, where $\mathfrak{V}$ denotes the range of values for memory cell contents.
- $C.\mathcal{S} \subseteq \mathfrak{A}$ - the set of shared writable addresses. Shared addresses can be owned or unowned. In the first case they can only be read by machines other than the owner.

We use the following shortcuts for state components:

$$\begin{array}{rclcrcl} \mathcal{O}_p(C) & \equiv & C.s[p].\mathcal{O} & \qquad & c_p(C) & \equiv & C.s[p].c \\ \mathcal{O}_p & \equiv & \mathcal{O}_p(C) & & c_p & \equiv & c_p(C) \end{array}$$

Primed, accented or indexed versions of these abbreviations translate in the obvious way to the functions taking the corresponding version of argument C e.g., $\hat{c}_p^i = c_p(\hat{C}^i)$.

In addition to np, $\mathfrak{A}$, $\mathfrak{V}$ and the configuration types $\mathfrak{C}_p$ we have the following *Cosmos* model system parameters:

- $\delta_p : \mathbb{C} \times \mathbb{N}_{np} \rightarrow \mathbb{C}^p$ for $p \in \mathbb{N}_{np}$ - a transition function for machine p.
- $\underset{\rightarrow}{\sigma} : \mathbb{N}_{np}^*$ - a schedule determining the interleaving of machine steps.
- $\mathcal{R} \subseteq \mathfrak{A}$ - the set of read-only addresses. Such addresses are neither shared writable, nor owned by any machine
- $\surd_p : \mathbb{C} \rightarrow \mathbb{B}$ for $p \in \mathbb{N}_{np}$ - a program logic to determine safe steps of machine p wrt. ownership and further safety conditions
- $\mathcal{IO}_p : \mathbb{C} \times \mathbb{N}_{np} \rightarrow \mathbb{B}$ for $p \in \mathbb{N}_{np}$ - a predicate based on the program logic of machine p identifying $\mathcal{IO}$ steps in the next transition of p depending on its inputs. For example $\mathcal{IO}$ steps include but are not limited to all reads or writes on shared writable memory.
- $reads_p, writes_p : \mathbb{C} \times \mathbb{N}_{np} \rightarrow 2^{\mathfrak{A}}$ for $p \in \mathbb{N}_{np}$ - functions yielding the sets of memory addresses read and written by machine p in its next step under the specified inputs.

In the declarations above $\mathbb{C}^p$ stands for the records type of the configuration that transition function δ_p produces. It is a projection of a full configuration C containing all machine states to a configuration $C|_p$ which only contains the state for machine p. The projection function $\cdot|_p : \mathbb{C} \times \mathbb{N}_{np} \rightarrow \mathbb{C}^p$ is formally defined as:

$$C|_p = \{C|_p.s, C|_p.m, C|_p.\mathcal{S}\} \qquad \text{s.t.} \qquad \bigwedge \left\{ \begin{array}{l} C|_p.s = C.s[p], \\ C|_p.m = C.m, \\ C|_p.\mathcal{S} = C.\mathcal{S} \end{array} \right\}$$

Note that in the Cohen-Schirmer theory $\mathcal{IO}$ memory instructions are denoted as *volatile* accesses. However to avoid confusion with the notion of volatile accesses on the C level we rename the concept here. Actually there is a close connection between volatile and $\mathcal{IO}$ accesses, as there are certain compiler requirements for all accesses that are compiled to ISA $\mathcal{IO}$ operations. In fact all volatile accesses on the C level become $\mathcal{IO}$ accesses on the ISA level. Nevertheless they are not congruent i.e., there might be more $\mathcal{IO}$ accesses which do not stem from volatile memory operations. We will see this later on.

In contrast to the Cohen-Schirmer model we also confined ourselves to treat the read-only addresses as a fixed parameter of the system. In the scope of this thesis we assume that the concurrent system we are focussing on is already initialized and after that point the read-only addresses should be constant. This implies a restriction of ownership transfer after the initialization phase i.e., no read-only addresses may be acquired by machines and all released addresses must stay writable. The restriction is motivated by the reordering proof further below. If addresses may be put in or taken out of the $\mathcal{R}$ set, there needs to be a synchronization policy between the machines governing when it is safe to acquire or release read-only addresses. If the set of read-only addresses was not fixed we would actually have to deal with an ordinary shared memory and all the complexity it is accompanied by. As we can think of little application scenarios of temporary read-only addresses, besides self-modifying code, we just omit this possibility here. Note that the initialization of concurrent systems is usually achieved in a sequential manner where only a single processor is active. Hence a different model must be considered to treat this case anyway.

2.2 Semantics

Now we will define the transition function Δ of the overall system. It computes the next state for a system step in a computation $\underset{\rightarrow}{C}$ which is based on the schedule $\underset{\rightarrow}{\sigma}$. We need a notation $C\lceil d \rceil_p$ to embed projected configurations $d \in \mathbb{C}^p$ in *Cosmos* model configurations $C \in \mathbb{C}$.

$$C\lceil d \rceil_p = C[\![m := d.m;\ s[p] := d.s;\ \mathcal{S} := d.\mathcal{S}]\!]$$

Then we define Δ for a step of p as follows.

$$\Delta(C, p) = C\lceil \delta_p(C, p) \rceil_p$$

Machine p is chosen according to schedule $\underset{\rightarrow}{\sigma}$ and transition function δ_p is computing the resulting projected configuration. The components of the *Cosmos* model configuration are updated accordingly. By construction a machine cannot alter the states of other machines including their ownership sets. There are more restrictions on the instantiation of *Cosmos* model parameters and their interaction defined in the next section.

2.3 Parameter Constraints

Not all parameters of a *Cosmos* model can be instantiated arbitrarily, in fact there are several constraints that must be discharged by an instantiation. The constrained instantiable parameters for the system without inputs, outputs and visible components are δ_p, $\mathcal{IO}_p$, $writes_p$ and $reads_p$.

Firstly, the step functions should be deterministic, i.e., no δ_p should make any non-deterministic choices. Secondly steps of p must not depend on components in other machines. Formally we collect these conditions in $constr_\delta(p)$. In the following let $R = reads_p(C, p)$ and $W = writes_p(C, p)$.

$$\frac{C \sim_p^R C' \Longrightarrow C\lceil \delta_p(C, p) \rceil_p \sim_p^W C'\lceil \delta_p(C', p) \rceil_p}{C, C' \in \mathbb{C} \vdash constr_\delta(p)}$$

Here the equality of information accessed by p (excluding visible components of other machines) is denoted by the relation $C \sim_p^A C'$ for sets of memory addresses $A \subseteq \mathfrak{A}$. Below we define several equivalence relations for different

Cosmos model components.

$$\begin{array}{lcl}
C \overset{m}{\sim}(A)\ C' & \equiv & \forall adr \in A.\ C.m(adr) = C'.m(adr) \\
C \sim_p^A C' & \equiv & C.s[p] = C'.s[p] \wedge C \overset{m}{\sim}(A)\ C' \\
C \sim_p C' & \equiv & C \sim_p^{\mathcal{O}_p \cup \mathcal{R}} C' \\
C \overset{s}{\sim} C' & \equiv & C.\mathcal{S} = C'.\mathcal{S} \wedge C \overset{m}{\sim}(C.\mathcal{S})\ C' \\
C \overset{o}{\sim} C' & \equiv & \forall p \in \mathbb{N}_{np}.\ \mathcal{O}_p = \mathcal{O}'_p \wedge C.\mathcal{S} = C'.\mathcal{S} \\
C \overset{o}{\sim}_p C' & \equiv & \mathcal{O}_p = \mathcal{O}'_p \wedge \mathcal{O}_p \cap C.\mathcal{S} = \mathcal{O}'_p \cap C'.\mathcal{S}
\end{array}$$

The relation $C \sim_p C'$ denotes the equality of machine state and locally visible memory. The equality of shared memory is given by $C \overset{s}{\sim} C'$, while $C \overset{o}{\sim} C'$ talks about the ownership state. The local versions of this relation for machine p is denoted by $C \overset{o}{\sim}_p C'$. It states that the ownership configuration of p is equivalent in to systems C and C' iff p owns the same addresses and these are partitioned identically into shared and local addresses. We quickly observe:

Corollary 1 *Examining the definitions of $\overset{o}{\sim}$ and $\overset{o}{\sim}_p$ we see that $C \overset{o}{\sim} C'$ implies $C \overset{o}{\sim}_p C'$ for all p.*

Moreover we require that the functions $reads_p$ and $writes_p$ should only rely on the local configuration of machine p

$$\frac{C \sim_p^R C' \Longrightarrow \bigwedge \left\{ \begin{array}{l} reads_p(C,p) = reads_p(C',p), \\ writes_p(C,p) = writes_p(C',p) \end{array} \right\}}{C, C' \in \mathbb{C} \vdash constr_{pred}(p)}$$

In the same way we demand that $\mathcal{IO}_p$ depends only on the state and local memory of machine p. Thus the program logic for p does not rely on the state of other machines or the contents of shared writable memory.

$$\frac{C \sim_p^R C' \implies \mathcal{IO}_p(C,p) = \mathcal{IO}_p(C',p)}{C, C' \in \mathbb{C} \vdash constr_{\mathcal{IO}}(p)}$$

Finally we are revisiting the $writes_p$ predicate. Naturally all memory cells not written by a machine step should maintain their values.

$$\frac{\forall adr \in \mathfrak{A} \setminus writes_p(C,p).\ \delta_p(C,p).m(adr) = C.m(adr)}{C \in \mathbb{C} \vdash constr_{mem}(p)}$$

2.4 Sound Memory Accesses and Ownership Invariants

With the ownership model consisting of $\mathfrak{A}$, $\mathcal{O}_p$, $\mathcal{R}$ and $\mathcal{S}$ we are imposing restrictions on the possible execution traces $\underset{\rightarrow}{C}_{\Delta,\sigma}$ in order to be able to perform reordering later. The predicate $sound(C,p)$ collects all necessary conditions, i.e., the access policy enforced by a consistent ownership configuration for memory accesses. We treat the cases of $\mathcal{IO}$ and local steps separately and start with the local, i.e., non-$\mathcal{IO}$ steps of machine p.

$$\frac{\begin{array}{c} /\mathcal{IO}_p(C,p) \qquad reads_p(C,p) \subseteq \mathcal{O}_p \cup \mathcal{R} \qquad writes_p(C,p) \subseteq \mathcal{O}_p \setminus C.\mathcal{S} \\ \mathcal{O}_p(\Delta(C,p)) = \mathcal{O}_p(C) \qquad \Delta(C,p).\mathcal{S} = C.\mathcal{S} \end{array}}{sound(C,p)}$$

All addresses being accessed in a local machine step must be either owned or read-only. To shared writable addresses which are owned by p and read-only addresses just reads are allowed in case of non-$\mathcal{IO}$ steps. Also the ownership configuration must stay constant for local operations.

$\mathcal{IO}$ step memory accesses may additionally target all shared read-write addresses but must not write them if they are owned by another machine. Ownership transitions of p are enabled but must not modify the sets of shared addresses owned by other machines.

$$\frac{\begin{array}{cc} \mathcal{IO}_p(C) & \forall p' \neq p.\ \mathcal{O}_{p'} \cap \Delta(C,p).\mathcal{S} = \mathcal{O}_{p'} \cap C.\mathcal{S} \\ reads_p(C) \subseteq \mathcal{O}_p \cup C.\mathcal{S} \cup \mathcal{R} & writes_p(C) \subseteq \mathcal{O}_p \cup C.\mathcal{S} \setminus \bigcup_{p' \neq p} \mathcal{O}_{p'} \end{array}}{sound(C,p)}$$

Additionally there are aforementioned consistency conditions between the ownership sets and the sets of shared addresses, which must be maintained by all transitions of the ownership configuration. For *Cosmos* models we denote these invariants by the predicate $inv : \mathbb{C} \to \mathbb{B}$:

$$\frac{\begin{array}{cc} \forall p, p'.\ p \neq p' \implies \mathcal{O}_p \cap \mathcal{O}_{p'} = \emptyset & \forall p.\ \mathcal{O}_p \cap \mathcal{R} = \emptyset \\ \forall adr \in \mathfrak{A}.\ (\nexists p.\ adr \in \mathcal{O}_p) \implies adr \in C.\mathcal{S} \cup \mathcal{R} & C.\mathcal{S} \cap \mathcal{R} = \emptyset \end{array}}{inv(C)}$$

The consistency conditions here include the disjointness of ownership sets and that shared read-only addresses can not be owned or shared read-write. Also all unowned addresses must be either shared writable or read-only.

2.5 Safety Conditions

Similar to the safety conditions in the store buffer reduction theorem every instantiation has to define constraints on the instructions and their effects on the ownership model in order to guarantee valid configurations of the ghost components and to enforce among others sound memory accesses wrt. ownership. These safety conditions can be defined similarly to the ones in the Cohen-Schirmer model. For every machine p we require a safety judgement $\surd_p$ on configurations $C \in \mathbb{C}$ depending on a set $R \subseteq Ad$ of read-only addresses as well as the total number of machines in the system. The safety conditions are defined wrt. an address space $Ad \subseteq \mathfrak{A}$.

$$R, Ad, n \vdash C \surd_p$$

Whenever a configuration is considered safe according to the safety judgement all possible steps out of that configuration must preserve the ownership invariants of the system and perform only sound transitions. Hence the correctness of the individual safety conditions must be proven for each instantiation. Let $safe(C,p)$ denote the safety wrt. $\surd_p$ of a step of machine p in configuration C under external inputs ext.

$$safe(C,p) \equiv \mathcal{R}, \mathfrak{A}, np \vdash C \surd_p$$

For the safety conditions we have to prove aforementioned lemma that all safe steps are sound and are preserving ownership invariants.

$$\forall p < np.\ inv(C) \wedge safe(C,p) \implies sound(C,p) \wedge inv(\Delta(C,p))$$

On the top level it is not defined how the ownership can be transfered between machines, however the safety conditions of p must guarantee that $inv(C)$ is preserved when p manipulates the ownership configuration. Similar to the $\mathcal{IO}_p$ predicate there is a constraint on the safety judgement that it should rely on system components and visible memory as well as the ownership configuration and machine outputs during steps of p.

$$\begin{aligned}
&constr_{safe}(p) \equiv \\
&\quad \forall C, C' \in \mathbb{C}. \\
&\qquad C \sim_p C' \wedge \begin{cases} C \overset{o}{\sim}_p C' & : \quad /\mathcal{IO}_p(C) \\ C \overset{o}{\sim} C' & : \quad \mathcal{IO}_p(C) \end{cases} \implies safe(C,p) = safe(C',p)
\end{aligned}$$

Thus we have a model that governs a concurrent execution $\underset{\rightarrow}{C}_{\delta,\sigma}$ by ownership and safety conditions. Accesses to shared resources must be marked as special $\mathcal{IO}$ operations representing, e.g., atomic memory instructions in case of multi-core processor systems. Local operations can only target owned unshared data that is not observable by other machines. Therefore it is possible to reorder these local steps as will be seen later on. First we extend our system with further means of communication.

3 Inputs, Outputs and Visible Components

Until now the only way for machines in our system to communicate is via the shared memory. While this is a convenient model it does not reflect the reality of many existing concurrent systems. In fact there might also be communication via interrupts, bus connections, I/O port registers and the like. Last but not least there may be external inputs and outputs of the system. In the following we strive to model these means of communication in a way as general as possible, so that it can be instantiated arbitrarily afterwards.

3.1 Additional Parameters

We extend our automaton model for a machine p with inputs in $\mathfrak{I}_p \subseteq \mathbb{O} \cup \mathfrak{E}$ and outputs using the alphabet $\mathfrak{O}_p$. Here $\mathfrak{E}$ represents external inputs to the system while $\mathbb{O}$ is the union of the outputs of all machines inside the system.

$$\mathbb{O} \equiv \bigcup_{p \in \mathbb{N}_{np}} \mathfrak{O}_p$$

For the new setting we have to update the following *Cosmos* model system parameters, as they also depend on inputs now.

- $\delta_p : \mathbb{C} \times \mathbb{N}_{np} \times 2^{\mathfrak{I}_p} \rightarrow \mathbb{C}^p$ for $p \in \mathbb{N}_{np}$
- $\mathcal{IO}_p : \mathbb{C} \times \mathbb{N}_{np} \times 2^{\mathfrak{I}_p} \rightarrow \mathbb{B}$ for $p \in \mathbb{N}_{np}$
- $reads_p, writes_p : \mathbb{C} \times \mathbb{N}_{np} \times 2^{\mathfrak{I}_p} \rightarrow 2^{\mathfrak{A}}$ for $p \in \mathbb{N}_{np}$

We introduce new parameters below.

- $\omega_p : \mathbb{C} \rightarrow 2^{\mathfrak{O}_p}$ for $p \in \mathbb{N}_{np}$ - the output function for machine p which may depend not only on $C.m$ and c_p but also on visible components of other machines' states.

- $\overset{v}{\sim}_p \subseteq \mathfrak{C}_p \times \mathfrak{C}_p$ for all $p \in \mathbb{N}_{np}$ - an equivalence relation implicitely encoding which components of machine configuration c_p are visible to other machines. Iff all these components are equal in two configurations of machine p then the relation holds.

- $vis_p \ : \ \mathbb{C} \times \mathbb{N}_{np} \times 2^{\mathfrak{I}_p} \ \rightarrow \ \mathbb{B}$ - predicate signaling that the next step of machine p accesses visible state components of any other machine.

We have introduced visible components above, on which the execution of other machines may depend. Thus we are extending the notion of sharing from memory addresses to machine components. This is useful when we consider, e.g., virtual machines that are not virtualized completely, i.e., some components of the underlying hardware are shared explicitely. Also for interrupts we will see later on that certain processor registers can be shared between the interruptible program and its interrupt handler. Thus for every machine p a subset of components may be visible and each update of these components by p or steps by other machines depending on them must be considered $\mathcal{IO}$ steps.

Furthermore observe that the function ω_p is defined on the complete state of the system. We will later restrict the outputs generated by it to depend on the visible compents and shared memory. Nevertheless they are inherently Mealy output signals of machine p wrt. the inputs of other machines, i.e., they might change even if p is not making a step. In contrast the visible components of p can only change in a ($\mathcal{IO}$) step of p. Hence we can consider them as the Moore outputs of machine p.

3.2 Extended Semantics

We also have to extend the transition function Δ of the overall system. A computation $\underrightarrow{C}$ of a CSO now also depends on an external input sequence $\underrightarrow{ext}$. Moreover we have to map the corresponding inputs to the machine p that takes the next setp. We define $Delta$ for a step of p under external input ext as follows.

$$\Delta(C, p, ext) = C\lceil \delta_p(C, p, in(C, p, ext)) \rceil_p$$

The inputs to machine p are calculated from the outputs of all machines and the external inputs by the function $in(C, p, ext)$ as defined below.

$$in(C, p, ext) \equiv \left(\bigcup_{p' \in \mathbb{N}_{np}} \omega_{p'}(C) \cup ext \right) \cap \mathfrak{I}_p$$

Considering computations $\underrightarrow{C}, \underrightarrow{\sigma}, \underrightarrow{ext}$ we also write $in_p(C^i)$ for $in(C^i, p, ext^i)$. Note that $\underrightarrow{ext}$ may contain external inputs for machines that are not currently scheduled. These inputs do not influence the execution of the system and are thus redundant. However to keep the model simple we do not restrict the external input sequence but allow it to contain arbitrary irrelevant inputs.

3.3 Updated Parameter Constraints

The constraints on the parameters introduced before must be adapted to the new situation where we have input and output signals as well as visible components. Furthermore we need to constrain the new parameters $\mathfrak{I}_p$, $\mathfrak{O}_p$, ω_p, $\overset{v}{\sim}_p$ and vis_p.

The step function of p may now also depend on visible components of others. Such accesses are denoted by the vis_p predicate. We update $constr_\delta(p)$ using the abbreviations $R = reads_p(C, p, in)$ and $W = writes_p(C, p, in)$.

$$\frac{\bigwedge \left\{ \begin{array}{l} C \sim_p^R C', \\ vis_p(C, p, in) \Rightarrow C \overset{v}{\sim} C' \end{array} \right\} \Longrightarrow C\lceil \delta_p(C, p, in) \rceil_p \sim_p^W C' \lceil \delta_p(C', p, in) \rceil_p}{C, C' \in \mathbb{C}, in \subseteq \mathfrak{I}_p \vdash constr_\delta(p)}$$

Here the equality of all visible components is denoted by the relation $C \overset{v}{\sim} C'$. We also define the relation $C \overset{sv}{\sim} C'$ which states the consistency of the shared state in the system (excluding read-only addresses).

$$\begin{array}{lcl} C \overset{v}{\sim} C' & \equiv & \forall p \in \mathbb{N}_{np}.\, c_p \overset{v}{\sim}_p c'_p \\ C \overset{sv}{\sim} C' & \equiv & C \overset{s}{\sim} C' \wedge C \overset{v}{\sim} C' \end{array}$$

In same way the functions and predicates $reads_p$, $writes_p$ and vis_p should only rely on the local configuration of machine p

$$\frac{C \sim_p^R C' \Longrightarrow \bigwedge \left\{ \begin{array}{l} reads_p(C, p, in) = reads_p(C', p, in), \\ writes_p(C, p, in) = writes_p(C', p, in), \\ vis_p(C, p, in) = vis_p(C', p, in) \end{array} \right\}}{C, C' \in \mathbb{C}, in \subseteq \mathfrak{I}_p \vdash constr_{pred}(p)}$$

As another parameter constraint $\overset{v}{\sim}_p$ has to be an equivalence relation.

$$\frac{c_p \overset{v}{\sim}_p c_p \qquad c_p \overset{v}{\sim}_p c'_p \Longrightarrow c'_p \overset{v}{\sim}_p c_p \qquad c_p \overset{v}{\sim}_p c'_p \wedge c'_p \overset{v}{\sim}_p c''_p \Longrightarrow c_p \overset{v}{\sim}_p c''_p}{C, C', C'' \in \mathbb{C} \vdash constr_v(p)}$$

Moreover we assume that every output or external input is at most handled by one machine in the *Cosmos* model so that there are no races on the reaction to input signals i.e.:

$$\frac{\forall p' \in \mathbb{N}_{np}.\, p \neq p' \Longrightarrow \mathfrak{I}_p \cap \mathfrak{I}_{p'} = \emptyset}{constr_{\mathfrak{I}}(p)}$$

Similarly we demand that all outputs and external inputs are unique. We only consider internal Mealy output signals here.

$$\frac{\begin{array}{c} \mathfrak{O}_p \cap \mathfrak{E} = \emptyset \qquad \forall p' \in \mathbb{N}_{np}.\, p \neq p' \Longrightarrow \mathfrak{O}_p \cap \mathfrak{O}_{p'} = \emptyset \\ \forall \alpha \in \mathfrak{O}_p.\, \exists p' \in \mathbb{N}_{np}.\, \alpha \in \mathfrak{I}_{p'} \end{array}}{constr_{\mathfrak{O}}(p)}$$

We restrict the output functions as follows.

$$\frac{\begin{array}{c} c_p = c'_p \wedge c_{p'} \overset{v}{\sim}_{p'} c'_{p'} \Longrightarrow \omega_p(C) =_\alpha \omega_p(C') \\ \omega_p(C) =_\alpha \omega_p(C') \wedge c_p = c''_p \wedge c'_p = c'''_p \wedge c''_{p'} \overset{v}{\sim}_{p'} c'''_{p'} \Longrightarrow \omega_p(C'') =_\alpha \omega_p(C''') \end{array}}{C, C', C'', C''' \in \mathbb{C}, p', \alpha \in \mathfrak{O}_p \cap \mathfrak{I}_{p'} \vdash constr_\omega(p)}$$

In general the constraint implies that the generated outputs of p for any machine p' depend only on the machine state of p and the visible components of p'.[4] Additionally the behaviour of output signals in case p does not make a step must be predictable from the changes in the visible components of the receiver. Later on we will also demand that outputs must be kept active until a reaction of a receiver via a change in the visible components is detected.

In order to identify situations when machine p reacts to an input (e.g. by starting an interrupt handler) we introduce the following predicate.

$$reacts(C, p, in) \equiv \delta_p(C, p, in) \neq \delta_p(C, p, \emptyset)$$

We detect such a situation by comparing the next configuration wrt. the actual inputs to one where all inputs are deactivated. When a deterministic machine changes its configuration differently for a given input, then it must have been in reaction to that input. Such situations represent a communication with the environment, therefore we consider them as $\mathcal{IO}$ steps, too.

Another property we will need is that reactions for the same inputs only depend on the configuration of machines (e.g. interrupt mask registers, APIC state etc.). Also inputs do not cancel out each other i.e., whenever there is a reaction to an input, no matter how much more inputs we add the machine will still react to one of them. Note that by construction machines only react to their personal inputs.

$$\frac{\begin{array}{c} c_p = c'_p \Longrightarrow reacts(C, p, in) = reacts(C', p, in) \\ /reacts(C, p, in) \wedge in' \subseteq in \Longrightarrow /reacts(C, p, in') \end{array}}{C, C' \in \mathbb{C}, in, in' \subseteq \mathfrak{I}_p \vdash constr_{react}(p)}$$

In addition $\mathcal{IO}_p$ must be redefined such that all reactions to inputs are also considered $\mathcal{IO}$ steps. Reacting to an input is part of a machine's communication with its environment, thus we model it as an $\mathcal{IO}$ operation. Moreover local steps stay local if their configurations are locally equivalent and there are not more inputs than in the original step. With R defined as above:

$$\frac{\begin{array}{c} C \sim_p^R C' \implies \mathcal{IO}_p(C, p, in) = \mathcal{IO}_p(C', p, in) \\ reacts(C, p, in) \Longrightarrow \mathcal{IO}_p(C, p, in) \\ \forall in' \subseteq in.\ /\mathcal{IO}_p(C, p, in) \wedge C \sim_p^R C' \implies /\mathcal{IO}_p(C', p, in') \end{array}}{C, C' \in \mathbb{C}, in \subseteq \mathfrak{I}_p \vdash constr_{\mathcal{IO}}(p)}$$

At last we add inputs to the constraint on the $writes_p$ predicate.

$$\frac{\forall adr \in \mathfrak{A} \setminus writes_p(C, p, in). \quad \delta_p(C, p, in).m(adr) = C.m(adr)}{C \in \mathbb{C}, in, \subseteq \mathfrak{I}_p \vdash constr_{mem}(p)}$$

With the updated parameter constraints defined we can now turn to the soundness conditions for the extended *Cosmos* model with inputs, outputs and visible components.

[4]This means that we do not consider memory mapped I/O ports to be part of shared memory. Such ports shall be part of the visible components of machines so that other machines may react to changes of them by their outputs also when they are not stepped. Note that this setting restricts *Cosmos* models to systems where I/O ports are not shared between different machines and there are fixed pairs of communication partners.

3.4 Soundness for Inputs, Outputs and Visible Components

The predicate $sound(C, p, ext)$ collects all necessary soundness conditions for the new system with (external) inputs, outputs and visible components. It is an extension of $sound(C, p)$ which is implied by $sound(C, p, ext)$. Again we treat the cases of $\mathcal{IO}$ and local steps separately. First we define a shorthand pattern for commonly used predicates of the form $pred_p(C, p, in)$.

$$pred_p(C) \equiv pred_p(C, p, in(C, p, ext))$$

Now we consider the local i.e. non-$\mathcal{IO}$ steps of machine p.

$$\frac{\begin{array}{c} /\mathcal{IO}_p(C) \qquad reads_p(C) \subseteq \mathcal{O}_p \cup \mathcal{R} \qquad writes_p(C) \subseteq \mathcal{O}_p \setminus C.\mathcal{S} \\ c_p(\Delta(C, p, ext)) \overset{v}{\sim}_p c_p \qquad \omega_p(\Delta(C, p, ext)) = \omega_p(C) \qquad /vis_p(C) \\ \mathcal{O}_p(\Delta(C, p, ext)) = \mathcal{O}_p(C) \qquad \Delta(C, p, ext).\mathcal{S} = C.\mathcal{S} \end{array}}{sound(C, p, ext)}$$

In addition to the shared memory access restrictions for non-$\mathcal{IO}$ steps, machines in this case may not alter visible components. Also local steps may not depend on visible components of others. In general the outputs of all machines in the system are not allowed to change for local steps of p and by construction p does not react to the inputs from other machines.

$$\frac{\begin{array}{c} \mathcal{IO}_p(C) \qquad \alpha \in \omega_p(C) \setminus \mathfrak{I}_p \Longrightarrow \alpha \in \omega_p(\Delta(C, p, ext)) \\ \forall p' \neq p.\ \mathcal{O}_{p'} \cap \Delta(C, p, ext).\mathcal{S} = \mathcal{O}_{p'} \cap C.\mathcal{S} \\ reads_p(C) \subseteq \mathcal{O}_p \cup C.\mathcal{S} \cup \mathcal{R} \qquad writes_p(C) \subseteq \mathcal{O}_p \cup C.\mathcal{S} \setminus \bigcup_{p' \neq p} \mathcal{O}_{p'} \end{array}}{sound(C, p, ext)}$$

In $\mathcal{IO}$ steps machine outputs must not be deactivated unless they are targetting themselves only which is benign. Besides that we have the same restrictions for $\mathcal{IO}$ steps as before. The ownership invariants are not touched by the introduction of the additional ways of communication.

3.5 Extended Safety Conditions

We also have to consider inputs $in \subseteq \mathfrak{I}_p$ in the program logic for p now.

$$R, Ad, n, in \vdash C\sqrt{}_p$$

Furthermore $safe(C, p, ext)$ denotes the safety wrt. $\sqrt{}_p$ of a step of machine p in configuration C under external inputs ext.

$$safe(C, p, ext) \equiv \mathcal{R}, \mathfrak{A}, np, in_p(C) \vdash C\sqrt{}_p$$

We write $safe_p(C^i)$ instead of $safe(C^i, p, ext^i)$ for all i whenever it is unambiguous and we are considering computations $\underrightarrow{C}$, $\underrightarrow{\sigma}$, $\underrightarrow{ext}$ with $\sigma^i = p$. Similarly $safe_p(C)$ and $safe_p(C')$ is short for $safe(C, p, ext)$, and $safe(C', p, ext)$ respectively. The safety lemma to be proven for every instantiation is stated below.

Lemma 1 *(Correctness of Safety Conditions) We consider a* Cosmos *model configuration C^i and its next state C^{i+1} that was reached by a step of any machine p under* *external inputs ext^i obeying its safety conditions. If ownership invariants were fulfilled before the step, all memory accesses of the step are sound and we again have a configuration where invariants hold.*

$$\forall p < np.\; inv(C^i) \wedge safe_p(C^i) \Longrightarrow sound(C^i, p, ext^i) \wedge inv(C^{i+1})$$

We also update $constr_{safe}(p)$ by taking inputs into account. As local steps are by definition never reacting to inputs safety can in this case be transferred to systems where we have the same local configuration but potentially less inputs using $constr_{react}(p)$. Only if we have more inputs a local step can become an unsafe ($\mathcal{IO}$) step. When examining the safety of $\mathcal{IO}$ steps in related system configurations identical inputs must be taken into account.

$$\begin{aligned}
&constr_{safe}(p) \equiv \\
&\quad \forall C, C' \in \mathbb{C}, ext \subseteq \mathfrak{E}. \\
&\qquad C \sim_p C' \wedge \begin{cases} C \overset{o}{\sim}_p C' \wedge in(C', p, ext) \subseteq in_p(C) & : \quad /\mathcal{IO}_p(C) \\ C \overset{o}{\sim} C' \wedge in(C', p, ext) = in_p(C) & : \quad \mathcal{IO}_p(C) \end{cases} \\
&\qquad\qquad\qquad\qquad \Longrightarrow \quad safe_p(C) = safe(C', p, ext)
\end{aligned}$$

This finishes the *Cosmos* model definition and we focus on reordering in the following.

4 $\mathcal{IO}$-Block Schedule Reordering

The ownership model introduced for *Cosmos* models allows us not only to impose safety conditions for sound concurrent memory accesses by the machines of the system. It also allows for a reordering of machine steps in the concurrent model. In the following we will provide theorems exploiting this fact in order to reduce the interleaving of machines and justify the assumption of coarse scheduling. We want to consider schedules where there are interleaved blocks of execution steps of a single machine p. Each such block starts with p performing an $\mathcal{IO}$ operation which is followed by a sequence of local computations. Such blocks we call $\mathcal{IO}$-blocks. Having a schedule interleaving only such $\mathcal{IO}$-blocks is convenient for Multiprocessor ISA machines when we want to apply compiler consistency and go up to the C and Assembly level later on. However it also applies to the modelling of systems with devices as well as preemptive threads running concurrently on the same processor. Moreover it justifies the concurrent verification approach of tools like VCC.

4.1 $\mathcal{IO}$-Block Schedules

Given a *Cosmos* model computation $\underset{\rightarrow}{C}_{\Delta,\sigma,ext}$ and a machine p we already defined the predicate $\mathcal{IO}(C^i, \sigma^i, ext^i)$ which states that in configuration C^i machine σ^i is at an $\mathcal{IO}$-point. Such $\mathcal{IO}$ accesses are usually used to access memory shared with other machines and for other atomic actions that have system-wide effects. As there are specific compiler optimization restrictions for these operations we will later on be able to apply compiler consistency at $\mathcal{IO}$-points and lift the execution from the machine to the C-Level for an Multiprocessor ISA instantiation.

Now we will define the structure of our desired schedule. Taking computation $\underrightarrow{C}$, schedule $\underrightarrow{\sigma}$ and external input sequence $\underrightarrow{ext}$ we denote by the predicate $\mathcal{IO}sched(\underrightarrow{C}, \underrightarrow{\sigma}, \underrightarrow{ext}, n)$ that for the first $n \in \mathbb{N}$ steps the computation exhibits a valid *$\mathcal{IO}$-block schedule*. For $n \leq 1$ we have:

$$\mathcal{IO}sched(\underrightarrow{C}, \underrightarrow{\sigma}, \underrightarrow{ext}, 0) \equiv 1 \qquad \mathcal{IO}sched(\underrightarrow{C}, \underrightarrow{\sigma}, \underrightarrow{ext}, 1) \equiv 1$$

Each $\mathcal{IO}$-block schedule must start in an initial configuration with the first instruction of one machine. For $n > 1$ the predicate is defined recursively as follows using the shorthand $\mathcal{IO}_{\sigma^i}(C^i) \equiv \mathcal{IO}_{\sigma^i}(C^i, \sigma^i, in(C^i, \sigma^i, ext^i))$ for any computation $\underrightarrow{C}, \underrightarrow{\sigma}, \underrightarrow{ext}$ and index i:

$$\begin{aligned}
&\mathcal{IO}sched(\underrightarrow{C}, \underrightarrow{\sigma}, \underrightarrow{ext}, n) \equiv \\
&\quad \mathcal{IO}sched(\underrightarrow{C}, \underrightarrow{\sigma}, \underrightarrow{ext}, n-1) \wedge \\
&\quad (\sigma^{n-1} \neq \sigma^{n-2} \Longrightarrow (\mathcal{IO}_{\sigma^{n-1}}(C^{n-1}) \vee \forall i < n-1.\ \sigma^i \neq \sigma^{n-1}))
\end{aligned}$$

This means an $\mathcal{IO}$-block schedule can be extended by adding a step of

1. the same currently running machine, or
2. another machine which
   (a) is currently at an $\mathcal{IO}$-point i.e., will be executing an $\mathcal{IO}$ step, or
   (b) was never scheduled before.

Thus the schedule consists of blocks of steps by the same machine which start with an $\mathcal{IO}$ step (or its first step) and are followed by instructions on the same machine which are not $\mathcal{IO}$ steps.

4.2 Reordering Theorem

Every trace of steps in a multiprocessor system can be reordered into an equivalent $\mathcal{IO}$-block schedule independent of its safety with respect to the ownership model. We introduce the predicate $safety(\underrightarrow{C}, \underrightarrow{\sigma}, \underrightarrow{ext}, n)$ which takes as inputs a CSO computation sequence $\underrightarrow{C}$, an arbitrary schedule $\underrightarrow{\sigma}$ and an external input sequence $\underrightarrow{ext}$. It denotes the safety of all states that are reachable by n steps of the concurrent system.

$$\frac{C^0 \longrightarrow^n_{\Delta,\sigma,ext} C^n \qquad inv(C^0) \qquad \forall i < n.\ safe_{\sigma^i}(C^i)}{safety(\underrightarrow{C}, \underrightarrow{\sigma}, \underrightarrow{ext}, n)}$$

That means that the trace is safe iff its initial configuration C^0 fulfills the ownership invariants and all steps from there leading into a configuration C^n are safe. For such safe traces we can construct a simulating trace of the same length which is an $\mathcal{IO}$-block schedule and also safe. Moreover we need to show that also for unsafe traces we are able to construct an unsafe $\mathcal{IO}$-block schedule starting in the same configuration.

When constructing equivalent $\mathcal{IO}$-block schedules it does not suffice to end up in the same configuration. We need to make sure that the new schedule is actually a reordering of the other. We capture this property and the relation between the reordered steps in the following judgement where we introduce a permutation function $u : \mathbb{N} \leftrightarrow \mathbb{N}$ which encodes the inverse reordering of machine steps thus yielding the original order. The shorthand $in(C^i)$ replaces $in(C^i, \sigma^i, ext^i)$ for any computation $\underrightarrow{C}, \underrightarrow{\sigma}, \underrightarrow{ext}$ and index i.

$$\frac{\begin{array}{c} \hat{\sigma}^i = \sigma^{u(i)} \qquad \hat{C}^i \sim_{\hat{\sigma}^i} C^{u(i)} \qquad \mathcal{IO}_{\hat{\sigma}^i}(\hat{C}^i) = \mathcal{IO}_{\sigma^{u(i)}}(C^{u(i)}) \\ \hat{C}^i \overset{o}{\sim}_{\hat{\sigma}^i} C^{u(i)} \qquad in(\hat{C}^i) \subseteq in(C^{u(i)}) \qquad e\hat{x}t^i = ext^{u(i)} \\ \mathcal{IO}_{\hat{\sigma}^i}(\hat{C}^i) \Longrightarrow \hat{C}^i \overset{sv}{\sim} C^{u(i)} \wedge \hat{C}^i \overset{o}{\sim} C^{u(i)} \wedge in(\hat{C}^i) = in(C^{u(i)}) \end{array}}{\underrightarrow{C}, \underrightarrow{\sigma}, \underrightarrow{ext}, \underrightarrow{\hat{C}}, \underrightarrow{\hat{\sigma}}, \underrightarrow{e\hat{x}t}, i \vdash u\surd_{reord}}$$

Step i of the new schedule is part of a reordered schedule according to u^{-1} if the schedules are consistent and $\mathcal{IO}$ steps are preserved. Moreover associated steps generally must be equivalent wrt. local components and ownership. For $\mathcal{IO}$ operations we also require that the same shared and ownership state is visible and these steps must observe identical inputs in both traces. For local steps we only demand that there are no more inputs than in the original trace as non-$\mathcal{IO}$ steps do not react to their internal inputs anyway. External inputs however are observed identically in corresponding states.

In the following for clarity we abbreviate inputs to predicates of the form $\underrightarrow{C}, \underrightarrow{\sigma}, \underrightarrow{ext}$ by the first sequence (here: $\underrightarrow{C}$) wherever this is possible. Assuming that a *Cosmos* model instantiation fulfills all parameter constraints we can now formulate our reordering theorem for $\mathcal{IO}$-block schedules, which justifies considering only such interleavings of machine steps. It consists of two parts.

Theorem 1 *($\mathcal{IO}$-Block Schedule Reordering)*

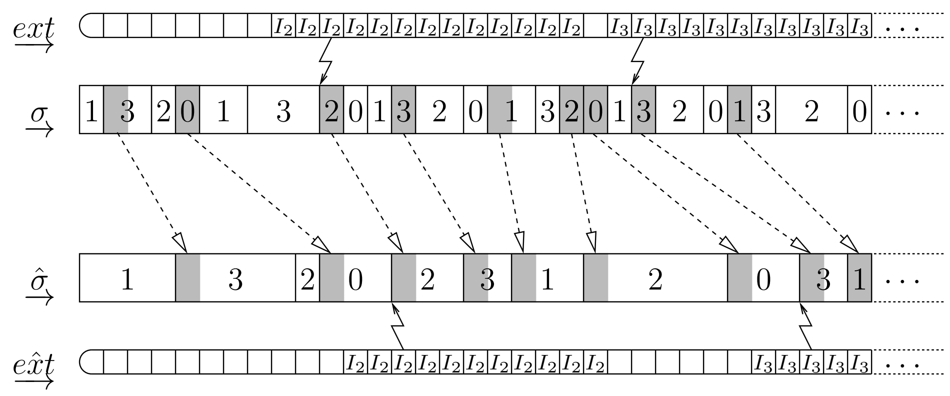
1. *(Existence) For any safe* Cosmos *model computation $\underrightarrow{C}, \underrightarrow{\sigma}, \underrightarrow{ext}$ which starts in a consistent state we can find a safe $\mathcal{IO}$-block schedule $\underrightarrow{\hat{C}}, \underrightarrow{\hat{\sigma}}, \underrightarrow{e\hat{x}t}$ simulating the original one. The new schedule is a valid reordering of $\underrightarrow{C}$.*

$$\begin{array}{lcl} \forall \underrightarrow{C}, \underrightarrow{\sigma}, \underrightarrow{ext}, n & . & safety(\underrightarrow{C}, n) \\ \quad \Downarrow & & \\ \exists \underrightarrow{\hat{C}}, \underrightarrow{\hat{\sigma}}, \underrightarrow{e\hat{x}t}, u & . & \bigwedge \left\{ \begin{array}{l} safety(\underrightarrow{\hat{C}}, n), \quad \mathcal{IO}sched(\underrightarrow{\hat{C}}, n), \\ \hat{C}^0 = C^0, \quad \hat{C}^n = C^n, \\ \forall i < n.\ \underrightarrow{C}, \underrightarrow{\hat{C}}, i \vdash u\surd_{reord} \end{array} \right\} \end{array}$$

2. *(Soundness) For any unsafe schedule we can find an unsafe $\mathcal{IO}$-block schedule which starts in the same initial configuration.*

$$\begin{array}{lcl} \forall \underrightarrow{C}, \underrightarrow{\sigma}, \underrightarrow{ext}, n & . & C^0 \longrightarrow^n_{\Delta,\sigma,ext} C^n \wedge /safety(\underrightarrow{C}, n) \\ \quad \Downarrow & & \\ \exists \underrightarrow{\hat{C}}, \underrightarrow{\hat{\sigma}}, \underrightarrow{e\hat{x}t}, \hat{n} \leq n & . & \bigwedge \left\{ \begin{array}{l} \hat{C}^0 \longrightarrow^{\hat{n}}_{\Delta,\hat{\sigma},e\hat{x}t} \hat{C}^{\hat{n}}, \quad \hat{C}^0 = C^0, \\ /safety(\underrightarrow{\hat{C}}, \hat{n}), \quad \mathcal{IO}sched(\underrightarrow{\hat{C}}, \hat{n}) \end{array} \right\} \end{array}$$

An illustrating example of the reordering can be found in Figure 1. The external inputs can be interpreted as external interrupts which are triggered by a device or interrupt controller and handled by processor machines 2 and 3. The grey boxes represent $\mathcal{IO}$ steps which keep their order. Local steps of some machine



$\underrightarrow{ext}$
I_2
I_3
$\underrightarrow{\sigma}$
1
3
2
0
$\underrightarrow{\hat{\sigma}}$
$\underrightarrow{\hat{ext}}$

Figure 1: Example for reordering an arbitrary *Cosmos* model schedule $\underrightarrow{\sigma}, \underrightarrow{ext}$ into an $\mathcal{IO}$-block schedule $\underrightarrow{\hat{\sigma}}, \underrightarrow{\hat{ext}}$ with $np = 4$; boxes marked with 0 to 3 are steps of machines with the respective identifier; greyish steps are representing $\mathcal{IO}$ steps; lightnings denote reactions to external inputs; I_2 and I_3 are singleton sets containing input signals handled only by machine 2 and 3 respectively; arrows illustrate the reordering of $\mathcal{IO}$ steps

p can be reordered across steps of other machines. Nevertheless the order in which p executes them is preserved.

Note also that in the figure it appears that external inputs obey the common external interrupt convention which states that interrupt signals should be stable until they are acknowledged by the receiver. We maintained this convention in both schedules $\underrightarrow{\sigma}$ and $\underrightarrow{\hat{\sigma}}$ for the steps where the respective receiving machines where not scheduled. As we mentioned before external inputs for machine p in $\underrightarrow{ext}$ at step i are redundant when $\sigma^i \neq p$ and we can actually omit them. However in the example we kept active the inputs for 2, and 3 respectively, also during steps of other machines in order to keep the picture simple and convey the actual continuity of the input in a real application where all machines are truly running in parallel. In a concurrent system with a scheduling function there is only an abstract notion of timing for inputs and interrupts, tied to the steps of machines reacting to them. This connection between machine steps and external inputs wrt. $\underrightarrow{C}$ and $\underrightarrow{\hat{C}}$ is formulated in $u\surd_{reord}$.

Observe that we do not need the valid reordering property for soundness. In our setting *Cosmos* model executions must be safe for all schedules and external input sequences, hence it is sufficient to find an arbitrary unsafe $\mathcal{IO}$-block schedule starting in the same configuration to guarantee soundness. We believe that our formulation of this order reduction theorem is correct and usable. For demonstration in Section 4.5 we will show how to apply it in a system verification task and transfer safety and verified program properties from $\mathcal{IO}$-Block schedules to arbitrary ones. Before we give the proof of the theorem we have to introduce some helpful notation and lemmata.

4.3 Auxilliary Definitions and Lemmata

In order to prove the theorem we split the equality between CSO configurations into two subrelations $\approx_p$ and $\approx_{\overline{p}}$ which distinguish the machine-local and the global state wrt. some machine p. Based on these relations we introduce helpful lemmata which we will use in the reordering proof later on. Firstly we define the two equivalence relations $\approx_p$ and $\approx_{\overline{p}}$ on the set of *Cosmos* model configurations taking into account the memory contents of the ownership domain and the register values of a certain machine $p < np$.

$$\begin{array}{lcl} C \approx_p C' & \equiv & C \sim_p C' \wedge C \overset{o}{\sim}_p C' \\ C \approx_{\overline{p}} C' & \equiv & \forall p' \neq p.\, C \sim_{p'} C' \wedge C \overset{sv}{\sim} C' \wedge \mathcal{O}_p = \mathcal{O}'_p \end{array}$$

The relations shall imply that from equivalent configurations a machine p will take the same local step for $\approx_p$, or any other machine than p will take the same step for $\approx_{\overline{p}}$ respectively. Below we make some useful observations.

Corollary 2 *From the definitions it directly follows for any $p \in \mathbb{N}_{np}$ that*

1. *$C \approx_p C' \wedge C \approx_{\overline{p}} C'$ is equivalent to $C = C'$.*
2. *$C \approx_{\overline{p}} C'$ implies $C \approx_{p'} C'$ for all $p' \neq p$.*
3. *$C \approx_{\overline{p}} C'$ implies $C \overset{o}{\sim} C'$.*
4. *$C \approx_{\overline{p}} C'$ and $\omega_p(C) = \omega_p(C')$ implies $in(C, p', ext) = in(C', p', ext)$ due to $constr_\omega(p')$ for all $p' \in \mathbb{N}_{np}$ and $ext \subseteq \mathfrak{E}$.*
5. *$C \approx_p C'$ and $C \overset{sv}{\sim} C'$ implies $C \sim_p^A C'$ for the set $A = \mathcal{O}_p \cup C.\mathcal{S} \cup \mathcal{R}$.*

Assuming the parameter constraint we can prove the following lemmata for our relations $\approx_p$ and $\approx_{\overline{p}}$.

Lemma 2 *The relations $C \approx_p C'$ and $C \approx_{\overline{p}} C'$ are equivalence relations for any machine $p \in \mathbb{N}_{np}$.*

PROOF: Obviously $C \sim_p C'$ is reflexive. Moreover it is symmetric and transitive in conjunction with $\mathcal{O}_p = \mathcal{O}'_p$ which is guaranteed by $C \overset{o}{\sim}_p C'$. The latter is an equivalence relation because it only uses unquantified equalities in its definition. Hence $C \approx_p C'$ is an equivalence relation as well. $C \approx_{\overline{p}} C'$ is an equivalence relation because its definition is based exclusively on equalities, $C \overset{o}{\sim}_p C'$ and the equivalence relation $\overset{v}{\sim}_p$. □

We need a lemma on the safety of steps in equivalent *Cosmos* configurations.

Lemma 3 *(Safety of Equivalent Configurations) Given two related CSO configurations C^i and $\hat{C}^i$ and sets of external inputs $ext^i, \hat{ext}^i \subseteq \mathfrak{E}$. If any machine p performs a safe step wrt. C^i and ext^i, then the same step is also safe wrt. $\hat{C}^i$ and $\hat{ext}^i$ according to the following statements.*

1. *If p takes a safe local step, then the same step is safe and local in an equivalent system if there the machine does not see more inputs.*

$$\bigwedge \left\{ \begin{array}{ll} C^i \approx_p \hat{C}^i, & /\mathcal{IO}_p(C^i), \\ safe_p(C^i), & in_p(\hat{C}^i) \subseteq in_p(C^i) \end{array} \right\} \Longrightarrow safe_p(\hat{C}^i) \wedge /\mathcal{IO}_p(\hat{C}^i)$$

2. *If two system configurations are equivalent with respect to all machines $p' \neq p$, and such a machine takes a safe step in one system, then the same step is also safe in the other one if the same outputs of p are seen.*

$$\bigwedge \left\{ \begin{array}{ll} C^i \approx_{\overline{p}} \hat{C}^i, & safe_{p'}(C^i), \\ \omega_p(C^i) = \omega_p(\hat{C}^i) & \end{array} \right\} \Longrightarrow \bigwedge \left\{ \begin{array}{l} safe_{p'}(\hat{C}^i), \\ \mathcal{IO}_{p'}(C^i) = \mathcal{IO}_{p'}(\hat{C}^i) \end{array} \right\}$$

PROOF: There are two parts of the Lemma to be shown.

1. Machine p is performing a local step and $C^i \approx_p \hat{C}^i$ holds. By Lemma 1 we know that the step of p from C^i is sound, in particular it does not react to any of the active inputs. Since machine p sees in $\hat{C}^i$ at most the same inputs as in C^i, we have $/reacts(C^i, p, in(\hat{C}^i, p, \hat{ext}^i))$ by constraint $constr_{react}(p)$ and because $c^i_p = \hat{c}^i_p$ is implied by $C^i \approx_p \hat{C}^i$ also $/reacts(\hat{C}^i, p, in(\hat{C}^i, p, \hat{ext}^i))$. Thus in $\hat{C}^i$ machine p does not react to any input, too. With the definition of $reacts$ we obtain:

$$\begin{array}{rcl} \delta_p(C^i, p, in(C^i, p, ext^i)) & = & \delta_p(C^i, p, \emptyset) \\ \delta_p(\hat{C}^i, p, in(\hat{C}^i, p, \hat{ext}^i)) & = & \delta_p(\hat{C}^i, p, \emptyset) \end{array}$$

   Therefore we can assume $in(C^i, p, ext^i) = \emptyset = in(\hat{C}^i, p, \hat{ext}^i)$ in the following. Then in $\hat{C}^i$ machine p also does not perform an $\mathcal{IO}$ step according to $constr_{\mathcal{IO}}(p)$. From $constr_{safe}(p)$ and $C^i \overset{o}{\sim}_p \hat{C}^i$ we deduce $safe_p(\hat{C}^i)$ and our claim holds.

2. From $C^i \approx_{\overline{p}} \hat{C}^i$ we get $C^i \approx_{p'} \hat{C}^i$ by Corollary 2.2 and thus $C^i \sim_{p'} \hat{C}^i$. Consequently the $\mathcal{IO}$ predicate has the same value in both configurations due to $constr_{\mathcal{IO}}(p')$. By Corollary 2.4 we know that p' sees the same inputs in both systems. Since $C^i \overset{o}{\sim} \hat{C}^i$ holds by Corollary 2.3, the ownership configuration is equal. Finally we obtain the safety of the step in the equivalent system $safe_{p'}(\hat{C}^i)$ from $constr_{safe}(p')$. □

Furthermore we can prove two lemmata concerning machine steps on equivalent configurations wrt. $\approx_p$ and $\approx_{\overline{p}}$.

Lemma 4 *(Local Processor Step) Assuming machine p takes a safe, local step under external inputs ext^i in a configuration C^i where $inv(C^i)$ holds, we execute the same step under inputs $\hat{ext}^i$ on a configuration $\hat{C}^i$ which is equivalent wrt. processor p. Then the resulting configurations are equivalent again and the environment is unaffected by the local step of p if in $\hat{C}^i$ not more inputs that in C^i are seen by p.*

$$\bigwedge \left\{ \begin{array}{lll} /\mathcal{IO}_p(C^i), & C^i \approx_p \hat{C}^i, & in_p(\hat{C}^i) \subseteq in_p(C^i) \\ C^i \longrightarrow_{\Delta,p,ext^i} C^{i+1}, & \hat{C}^i \longrightarrow_{\Delta,p,\hat{ext}^i} \hat{C}^{i+1}, & \\ inv(C^i), & inv(\hat{C}^i), & safe_p(C^i) \end{array} \right\} \Longrightarrow \bigwedge \left\{ \begin{array}{l} C^{i+1} \approx_p \hat{C}^{i+1}, \\ C^i \approx_{\overline{p}} C^{i+1}, \\ \hat{C}^i \approx_{\overline{p}} \hat{C}^{i+1} \end{array} \right\}$$

PROOF: By Lemma 3.1 the step of p in $\hat{C}^i$ is also safe and local. From the soundness of both steps (Lemma 1) and the definition of $\approx_p$ we see that all information accessible to the machines in a local step is equal. By $constr_\delta(p)$ the same local operation is executed by p in both system configurations C^i and $\hat{C}^i$. Subsequently we prove the three claims one by one.

1. $C^{i+1} \approx_p \hat{C}^{i+1}$ - Inserting the definitions for $C^i \approx_p \hat{C}^i$ we obtain:

   $$c_p^i = \hat{c}_p^i \qquad C^i \overset{o}{\sim}_p \hat{C}^i \qquad \forall adr \in \mathcal{O}_p^i \cup \mathcal{R}.\ C^i.m(adr) = \hat{C}^i.m(adr)$$

   Both systems perform a local step according to δ_p. Sound accesses may only access memory addresses from the set $\mathcal{O}_p^i \cup \mathcal{R}$ which is equal to $\hat{\mathcal{O}}_p^i \cup \mathcal{R}$ due to $C^i \overset{o}{\sim}_p \hat{C}^i$. However all these memory cells have the same content. Additionally both machines may only access their own machine configurations $c_p^i = \hat{c}_p^i$. Therefore by the $constr_\delta(p)$ we know that both machines will perform the same local step. Hence we have:

   $$c_p^{i+1} = \hat{c}_p^{i+1}, \qquad \forall adr \in \mathcal{O}_p^i \cup \mathcal{R}.\ C^{i+1}.m(adr) = \hat{C}^{i+1}.m(adr)$$

   The ownership state also does not change for local steps. From this observation $C^i \overset{o}{\sim} C^{i+1}$, $\hat{C}^i \overset{o}{\sim} \hat{C}^{i+1}$ follows $C^i \overset{o}{\sim}_p C^{i+1}$ and $\hat{C}^i \overset{o}{\sim}_p \hat{C}^{i+1}$ by Corollary 1. As we have $C^i \overset{o}{\sim}_p \hat{C}^i$ and $\overset{o}{\sim}_p$ is an equivalence relations we obtain:

   $$C^{i+1} \overset{o}{\sim}_p \hat{C}^{i+1}$$

   Since also $\mathcal{O}_p^i \cup \mathcal{R} = \mathcal{O}_p^{i+1} \cup \mathcal{R}$ the condition on local memory above now matches the definition of $C^{i+1} \sim_p \hat{C}^{i+1}$ and we conclude $C^{i+1} \approx_p \hat{C}^{i+1}$.

2. $C^i \approx_{\overline{p}} C^{i+1}$ - Again we know that the ownership state does not change for local steps i.e., $C^i \overset{o}{\sim} C^{i+1}$ and in particular $\mathcal{O}_p^i = \mathcal{O}_p^{i+1}$. Furthermore the step of p is sound wrt. the ownership policy. We prove the memory consistency condition first. Memory soundness conditions forbid local writes to shared addresses. Hence $C^i \overset{s}{\sim} C^{i+1}$ holds and also $C^i \overset{sv}{\sim} C^{i+1}$ as a sound local step of p cannot alter its visible components or other machine states. Moreover it is forbidden to write to addresses owned by other processors. Ownership invariants guarantee that these sets are disjoint from the owned addresses of p. Therefore

   $$\forall adr, p' \neq p.\ adr \in \mathcal{O}_{p'}^i \implies C^i.m(adr) = C^{i+1}.m(adr)$$

   and also $\forall adr \in \mathcal{R}.\ C^i.m(adr) = C^{i+1}.m(adr)$ as read-only memory is never modified. Additionally $\forall p' \neq p.\ C^i.s[p'] = C^{i+1}.s[p']$ holds because other machine states cannot be altered by steps of p, thus

   $$\forall p' \neq p.\ C^i \sim_p^{\mathcal{O}_{p'}^i \cup \mathcal{R}} C^{i+1}$$

   which is the definition of:

   $$\forall p' \neq p.\ C^i \sim_p C^{i+1}$$

   Now our claim $C^i \approx_{\overline{p}} C^{i+1}$ follows from the definition of $\approx_{\overline{p}}$.

3. $\hat{C}^i \approx_{\overline{p}} \hat{C}^{i+1}$ - Lemma 3 yields $safe_p(\hat{C}^i)$ and $/\mathcal{IO}_p(\hat{C}^i)$. The rest of the proof is analogous to the one above. □

Lemma 5 *(Environment Steps) Given is a* Cosmos *model performing n consecutive steps according to schedule $\underset{\rightarrow}{\sigma}$ under external inputs $\underset{\rightarrow}{ext}$ starting in the consistent configuration C^i with $inv(C^i)$.*

$$C^i \longrightarrow_{\Delta,\sigma^i,ext^i} C^{i+1} \longrightarrow_{\Delta,\sigma^{i+1},ext^{i+1}} \cdots \longrightarrow_{\Delta,\sigma^{i+n-1},ext^{i+n-1}} C^{i+n}$$

Now assume that $\forall j \in [i : i+n) = I.\ \sigma^j \neq p$ i.e., machine p is never scheduled in the interval I, and all steps of the other machines are $safe_{\sigma^j}(C^j)$, then it holds:

1. $C^i \approx_p C^{i+n} \wedge inv(C^{i+n})$
2. $\forall \underset{\rightarrow}{\hat{C}}.\ \bigwedge \begin{Bmatrix} \hat{C}^i \approx_{\overline{p}} C^i, inv(\hat{C}^i), \\ \hat{C}^i \longrightarrow^n_{\Delta,\sigma,ext} \hat{C}^{i+n}, \\ \omega_p(\hat{C}^i) = \omega_p(C^i) \end{Bmatrix} \Longrightarrow \bigwedge \begin{Bmatrix} \hat{C}^{i+n} \approx_{\overline{p}} C^{i+n}, inv(\hat{C}^{i+n}), \\ \forall j \in I.\ safe_{\sigma^j}(\hat{C}^j), \\ \omega_p(\hat{C}^{i+n}) = \omega_p(C^{i+n}) \end{Bmatrix}$

PROOF: Both parts can be proven by induction on n.

1. *Induction Start*: $n = 0$. The claim becomes trivial:

$$\begin{aligned} C^i = C^{i+0} &\Longrightarrow C^i \approx_p C^{i+n} \\ inv(C^i) &\Longrightarrow inv(C^{i+n}) \end{aligned}$$

Induction Hypothesis: The claim holds for an arbitrary but fixed n.

$$\bigwedge \begin{Bmatrix} C^i \longrightarrow^n_{\Delta,\sigma,ext} C^{i+n}, & inv(C^i), \\ \forall j \in [i : i+n).\ \sigma^j \neq p \wedge safe_{\sigma^j}(C^j) \end{Bmatrix} \Longrightarrow \bigwedge \begin{Bmatrix} C^i \approx_p C^{i+n}, \\ inv(C^{i+n}) \end{Bmatrix}$$

Induction Step: $n \rightarrow n+1$. We can directly apply the induction hypothesis on the first n steps of $C^i \longrightarrow^{n+1}_{\Delta,\sigma,ext} C^{i+n+1}$ obtaining:

$$C^i \approx_p C^{i+n} \wedge inv(C^{i+n})$$

By Lemma 1 and the prerequisite $safe_{\sigma^{i+n}}(C^{i+n})$ we get the invariant $inv(C^{i+n+1})$ and the definition of $\approx_p$ gives us:

$$\text{IH} \Longrightarrow \bigwedge \begin{Bmatrix} c^i_p = c^{i+n}_p, \quad C^i \overset{o}{\sim}_p C^{i+n}, \\ \forall adr \in \mathcal{O}^i_p \cup \mathcal{R}.\ C^i.m(adr) = C^{i+n}.m(adr) \end{Bmatrix}$$

Since $\sigma^{i+n} = p' \neq p$ we immediately have $c^i_p = c^{i+n+1}_p$ and $\mathcal{O}^i_p = \mathcal{O}^{i+n+1}_p$. The step of machine $p' \neq p$ is safe, thus it is also sound by Lemma 1 and p' cannot alter data which is owned by p. We know also that read-only addresses are never written by sound accesses, therefore

$$\forall adr \in \mathcal{O}^i_p \cup \mathcal{R}.\ C^i.m(adr) = C^{i+n+1}.m(adr)$$

and $C^i \sim_p C^{i+n+1}$ holds. The equivalence of the terms $\mathcal{O}^i_p \cap C^i.\mathcal{S}$ and $\mathcal{O}^{i+n+1}_p \cap C^{i+n+1}.\mathcal{S}$ is left to show. From induction hypothesis we have:

$$\mathcal{O}^i_p \cap C^i.\mathcal{S} = \mathcal{O}^{i+n}_p \cap C^{i+n}.\mathcal{S}$$

Again we argue using the soundness of the n+1-st step and the ownership invariants. Then it follows:

$$\mathcal{O}^{i+n+1}_p \cap C^{i+n+1}.\mathcal{S} = \mathcal{O}^{i+n}_p \cap C^{i+n}.\mathcal{S} = \mathcal{O}^i_p \cap C^i.\mathcal{S}$$

Only p can share or unshare the addresses it owns, consequently we have $C^i \overset{o}{\sim}_p C^{i+n+1}$ and thus $C^i \approx_p C^{i+n+1}$ which completes our proof of the first part of the lemma.

2. *Induction Start*: $n = 0$. We directly have:

$$\begin{aligned} C^i = C^{i+0} \wedge \hat{C}^i = \hat{C}^{i+0} \wedge C^i \approx_{\overline{p}} \hat{C}^i &\implies C^{i+n} \approx_{\overline{p}} \hat{C}^{i+n} \\ \hat{C}^i = \hat{C}^{i+0} \wedge inv(\hat{C}^i) &\implies inv(\hat{C}^{i+n}) \end{aligned}$$

The safety statement becomes trivial as there is no step taken at all.

Induction Hypothesis: The claim shall hold for an arbitrary but fixed n given that the steps from configuration i to $i + n$ are safe. For all $\underset{\rightarrow}{\hat{C}}$ we then have:

$$\bigwedge \left\{ \begin{array}{l} \hat{C}^i \approx_{\overline{p}} C^i, inv(\hat{C}^i), \\ \hat{C}^i \longrightarrow^n_{\Delta,\sigma,ext} \hat{C}^{i+n}, \\ \omega_p(\hat{C}^i) = \omega_p(C^i) \end{array} \right\} \Longrightarrow \bigwedge \left\{ \begin{array}{l} \hat{C}^{i+n} \approx_{\overline{p}} C^{i+n}, inv(\hat{C}^{i+n}), \\ \forall j \in [i : i+n).\ safe_{\sigma^j}(\hat{C}^j), \\ \omega_p(\hat{C}^{i+n}) = \omega_p(C^{i+n}) \end{array} \right\}$$

Induction Step: $n \rightarrow n + 1$. by Induction Hypothesis the claim already holds for configuration $\hat{C}^{i+n}$ and the steps from i to $i + n - 1$. From Lemma 3.2 we get the safety of step $i + n$. Hence:

$$\forall j \in [i : i+n+1).\ safe_{\sigma^j}(\hat{C}^j)$$

With $inv(\hat{C}^{i+n})$ and Lemma 1 we have $inv(\hat{C}^{i+n+1})$. The definition of $\approx_{\overline{p}}$ gives us:

$$\forall p' \neq p.\ C^{i+n} \sim_{p'} \hat{C}^{i+n} \qquad C^{i+n} \overset{sv}{\sim} \hat{C}^{i+n} \qquad \mathcal{O}_p^{i+n} = \hat{\mathcal{O}}_p^{i+n}$$

Let $p'' \neq p$ perform the n+1-st step in the computation starting in C^i or $\hat{C}^i$ respectively. For any safe step of p'' we get by Lemma 5.1 that relation $\approx_{p'}$ is preserved for all $p' \notin \{p, p''\}$, thus by Corollary 2.2 and the transitivity of $\approx_{p'}$ we have:

$$\forall p' \notin \{p, p''\}.\ \hat{C}^{i+n+1} \approx_{p'} C^{i+n+1}$$

Conforming to $constr_\omega$ for p' and p'' all these machines have the same outputs to other machines in both configurations $\hat{C}^{i+n}$ and C^{i+n} since their local state and all visible machine components are consistent. Also for p outputs are identical by IH. Using the same external input sequence inputs to p'' are identical by Corollary 2.4. Since all state and memory accessible to p'' is equal in both systems, we know that the same step is executed according to $constr_\delta(p'')$, yielding the same results $C^{i+n+1} \overset{sv}{\sim} \hat{C}^{i+n+1}$ and $C^{i+n+1} \sim_{p''} \hat{C}^{i+n+1}$. In combination we have:

$$\forall p' \neq p.\ \hat{C}^{i+n+1} \sim_{p'} C^{i+n+1} \qquad C^{i+n+1} \overset{sv}{\sim} \hat{C}^{i+n+1}$$

Note that we are applying soundness of the step of p'' here. By induction hypothesis all values that p' may safely read have the same value, therefore the same results are written. All other memory contents preserve their value by $constr_{mem}(p'')$. In the construction of Δ we see that the owned set of p can not be altered by p'', hence:

$$\mathcal{O}_p^{i+n+1} = \mathcal{O}_p^{i+n} \overset{\text{IH}}{=} \hat{\mathcal{O}}_p^{i+n} = \hat{\mathcal{O}}_p^{i+n+1}$$

Consequently we can apply the definition of $\approx_{\overline{p}}$ again and conclude:

$$\hat{C}^{i+n+1} \approx_{\overline{p}} C^{i+n+1}$$

By the construction of Δ we get $c_p^{i+n} = c_p^{i+n+1}$ and $\hat{c}_p^{i+n} = \hat{c}_p^{i+n+1}$ as well as $c_{p'}^{i+n} \overset{v}{\sim}_{p'} c_{p'}^{i+n+1}$ and $\hat{c}_{p'}^{i+n} \overset{v}{\sim}_{p'} \hat{c}_{p'}^{i+n+1}$ for $p' \neq p''$. Thus by $constr_\omega(p)$ we get for all $\alpha \in \mathfrak{O}_p \setminus \mathfrak{I}_{p''}$:

$$\omega_p(C^{i+n+1}) =_\alpha \omega_p(C^{i+n}) = \omega_p(\hat{C}^{i+n}) =_\alpha \omega_p(\hat{C}^{i+n+1})$$

For outputs $\alpha \in \mathfrak{O}_p \cap \mathfrak{I}_{p''}$ targetting machine p'' we have by the second condition of $constr_\omega(p)$ and $c_{p''}^{i+n+1} \overset{v}{\sim}_{p''} \hat{c}_{p''}^{i+n+1}$:

$$\begin{aligned} \omega_p(C^{i+n}) \overset{\text{IH}}{=} \omega_p(\hat{C}^{i+n}) \quad &\Longrightarrow \quad \omega_p(C^{i+n}) =_\alpha \omega_p(\hat{C}^{i+n}) \\ &\Longrightarrow \quad \omega_p(C^{i+n+1}) =_\alpha \omega_p(\hat{C}^{i+n+1}) \end{aligned}$$

As $\omega_p(C^{i+n+1}) =_\alpha \omega_p(\hat{C}^{i+n+1})$ holds for all $\alpha \in \mathfrak{O}_p$ we can deduce the desired $\omega_p(C^{i+n+1}) = \omega_p(\hat{C}^{i+n+1})$.

Having shown its two parts to be correct the lemma holds. □

4.4 Reordering Proof

In the following we will present proofs of the two parts of our $\mathcal{IO}$-block schedule reordering theorem.

4.4.1 Existence

First we need to show that there is a valid reordering for all possible safe ISA traces. Later we can use this property in the soundness proof.

$$\begin{array}{lcl} \forall \underrightarrow{C}, \underrightarrow{\sigma}, \underrightarrow{ext}, n & . & safety(\underrightarrow{C}, n) \\ \quad\Downarrow & & \\ \exists \underrightarrow{\hat{C}}, \underrightarrow{\hat{\sigma}}, \underrightarrow{e\hat{x}t}, u & . & \bigwedge \left\{ \begin{array}{l} safety(\underrightarrow{\hat{C}}, n), \quad \mathcal{IO}sched(\underrightarrow{\hat{C}}, n), \\ \hat{C}^0 = C^0, \quad \hat{C}^n = C^n, \\ \forall i < n.\ \underrightarrow{C}, \underrightarrow{\hat{C}}, i \vdash u\surd_{reord} \end{array} \right\} \end{array}$$

<span style="font-variant:small-caps">Proof</span>: By induction on n.

Induction Start: For $n = 0$ the $\mathcal{IO}sched$, $extconv$ and $u\surd_{reord}$ claims hold trivially for arbitrary $\hat{\sigma}$, $e\hat{x}t$ and u. Since no step is taken at all the safety claim reduces to $inv(\hat{C}^0)$. It holds by construction $C^0 = \hat{C}^0$ which also proves the remaining $\hat{C}^n = C^n$.

Induction Hypothesis: Given an arbitrary schedule $\underrightarrow{\sigma}$ and external interrupt sequence $\underrightarrow{ext}$ we assume a safe *Cosmos* model computation $\underrightarrow{C}$, then we can obtain the induction hypothesis.

$$\exists \underrightarrow{\bar{C}}, \underrightarrow{\bar{\sigma}}, \underrightarrow{e\bar{x}t}, \bar{u}.\ \bigwedge \left\{ \begin{array}{l} safety(\underrightarrow{\bar{C}}, n), \quad \mathcal{IO}sched(\underrightarrow{\bar{C}}, n), \\ \bar{C}^0 = C^0, \quad \bar{C}^n = C^n, \\ \forall i < n.\ \underrightarrow{C}, \underrightarrow{\bar{C}}, i \vdash \bar{u}\surd_{reord} \end{array} \right\}$$

Induction Step: $n \to n+1$, Let us now consider a safe n+1-step execution trace $\underrightarrow{C}$ under external interrupts $\underrightarrow{ext}$ and the schedule $\underrightarrow{\sigma}$. By induction hypothesis, for the n-step subsequence

$$C^0 \longrightarrow_{\Delta,\sigma^0,ext^0} C^1 \longrightarrow_{\Delta,\sigma^1,ext^1} \cdots \longrightarrow_{\Delta,\sigma^{n-1},ext^{n-1}} C^n$$

we can find a safe and equivalent $\mathcal{IO}$-block schedule $\underrightarrow{\bar{C}}$, $\underrightarrow{\bar{\sigma}}$ under external inputs $\underrightarrow{e\bar{x}t}$ and inverse reordering function $\bar{u}$ according to induction hypothesis. Now we perform a case distinction over the nature of the operation to be executed on machine σ^n in the n+1-st step of the execution sequence.

1. $\mathcal{IO}_{\sigma^n}(C^n)$: The n+1-st step starts a new $\mathcal{IO}$-block. Independent of the machine previously executing this is always allowed in an $\mathcal{IO}$-block schedule. Thus $\hat{\sigma}^n = \sigma^n$ and the same step is executed in both schedules. The new inverse reordering function u is defined as follows.

$$u(i) = \begin{cases} \bar{u}(i) & : \quad i < n \\ i & : \quad \text{otherwise} \end{cases}$$

We create the new schedule and external event sequence

$$\underrightarrow{\hat{\sigma}} = \underrightarrow{\bar{\sigma}}[0:n), \sigma^n \qquad\qquad \underrightarrow{e\hat{x}t} = \underrightarrow{e\bar{x}t}[0:n), ext^n$$

by appending the final step of $\underrightarrow{\sigma}$. Thus $\hat{C}^0 = C^0$ holds by construction. We can easily show the remaining five claims.

Because the first n steps of our reordered schedule are identical to the IH execution trace we get $\hat{C}^0 \longrightarrow^n_{\Delta,\hat{\sigma},e\hat{x}t} \hat{C}^n$ for free. Naturally we can extend the trace for one more step using Δ, $\hat{\sigma}$ and $e\hat{x}t$.

$$\bigwedge \left\{ \begin{array}{l} \hat{C}^0 \longrightarrow^n_{\Delta,\hat{\sigma},e\hat{x}t} \hat{C}^n \quad \text{(IH)}, \\ \hat{C}^{n+1} = \Delta(\hat{C}^n, \hat{\sigma}, e\hat{x}t) \end{array} \right\} \implies \hat{C}^0 \longrightarrow^{n+1}_{\Delta,\hat{\sigma},e\hat{x}t} \hat{C}^{n+1}$$

By induction hypothesis the configuration $\hat{C}^n$ is identical to C^n and the n+1-st step is scheduled on the same processor under the same external interrupts. Since machine execution is deterministic also the resulting configurations are equal after $n+1$ steps for the two different schedules.

$$\bigwedge \left\{ \begin{array}{l} \hat{C}^n = \bar{C}^n \overset{\text{IH}}{=} C^n, \\ \hat{\sigma}^n = \sigma^n, \\ e\hat{x}t^n = ext^n \end{array} \right\} \implies \hat{C}^{n+1} = C^{n+1}$$

All steps in the IH schedule are safe and the reordered schedule equals the former for the first n steps. Hence this part is safe. As the last step equals the safe step in the original schedule the complete new one is safe.

$$\bigwedge \left\{ \begin{array}{l} \forall i < n.\ safe_{\hat{\sigma}^i}(\hat{C}^i) \quad \text{(IH)}, \\ \hat{C}^n \overset{\text{IH}}{=} C^n,\ \hat{\sigma}^n = \sigma^n, \\ e\hat{x}t^n = ext^n,\ safe_{\hat{\sigma}^n}(C^n) \end{array} \right\} \implies \forall i < n+1.\ safe_{\hat{\sigma}^i}(\hat{C}^i)$$

By construction $\hat{C}^0 = C^0$ we also have $inv(\hat{C}^0)$. What is left to show is that the resulting interleaving of processor steps is a valid $\mathcal{IO}$-block schedule reordering. For the first n steps this holds trivially by IH.

$$\bigwedge \left\{ \begin{array}{l} \mathcal{IO}sched(\underrightarrow{\bar{C}}, n) \quad \text{(IH)}, \\ \underrightarrow{\hat{C}}[0:n) = \underrightarrow{\bar{C}}[0:n) \\ \underrightarrow{\hat{\sigma}}[0:n) = \underrightarrow{\bar{\sigma}}[0:n) \\ \underrightarrow{e\hat{x}t}[0:n) = \underrightarrow{e\bar{x}t}[0:n) \end{array} \right\} \implies \mathcal{IO}sched(\underrightarrow{\hat{C}}, n)$$

Since the last step starts a new $\mathcal{IO}$-block on machine σ^n with an $\mathcal{IO}$ step and both traces perform the same step, the $\mathcal{IO}sched$ property holds also for the complete reordered trace.

$$\bigwedge \left\{ \begin{array}{l} \mathcal{IO}sched(\underrightarrow{\hat{C}}, n), \quad \mathcal{IO}_{\sigma^n}(C^n), \\ \hat{C}^n \overset{\text{IH}}{=} C^n,\ \hat{\sigma}^n = \sigma^n, \\ e\hat{x}t^n = ext^n,\ \text{Def.}\, \mathcal{IO}sched \end{array} \right\} \implies \mathcal{IO}sched(\underrightarrow{\hat{C}}, n+1)$$

By induction hypothesis and construction of u we know already that the first n steps of the new schedule are a valid reordering. Due to the identity of configurations and external inputs and since the same step is scheduled the last claim holds trivially:

$$\forall i < n+1.\ \underrightarrow{C}, \underrightarrow{\hat{C}}, i \vdash u\surd_{reord}$$

2. $/\mathcal{IO}_{\sigma^n}(C^n) \wedge \sigma^n = \bar{\sigma}^{n-1}$: The additional n+1-st step belongs to the same machine as the last one in the reordered IH schedule. According to the definition of $\mathcal{IO}$-block schedules we can add it to the last $\mathcal{IO}$-block in the induction hypothesis schedule with no need for further reordering. We construct the $\mathcal{IO}$-block schedule and prove all claims analogously to the previous case. The only difference is made for the proof of the property $\mathcal{IO}sched(\underrightarrow{\hat{C}}, n+1)$. The definition of $\mathcal{IO}sched$ gives us here:

   $$\mathcal{IO}sched(\underrightarrow{\hat{C}}, n) \wedge (\hat{\sigma}^n \neq \hat{\sigma}^{n-1} \implies (\mathcal{IO}_{\hat{\sigma}^n}(\hat{C}^n) \vee \forall i < n.\ \hat{\sigma}^i \neq \hat{\sigma}^n))$$

   The first requirement is fulfilled by induction hypothesis and the construction $\underrightarrow{\hat{C}}[0:n) = \underrightarrow{\bar{C}}[0:n)$. The second requirement is irrelevant for our case since:

   $$\hat{\sigma}^n = \sigma^n = \bar{\sigma}^{n-1} = \hat{\sigma}^{n-1}$$

   Therefore the first part of the reordering theorem holds for this case, too.

3. $\forall i < n.\ \sigma^n \neq \bar{\sigma}^i$: The n+1-st step is executed by a machine which was never scheduled before. Since the first n steps of the IH schedule are already representing an $\mathcal{IO}$-block schedule, we can simply append the additional step to the IH trace and obtain a valid $\mathcal{IO}$-block schedule of $n+1$ steps. This case is explicitely treated in the definition of $\mathcal{IO}sched$. The claims of the theorem are proven completely alike to the above cases.

4. $/\mathcal{IO}_{\sigma^n}(C^n) \wedge \sigma^n \neq \bar{\sigma}^{n-1} \wedge \exists j.\, j = \max\{k+1 \mid k < n \wedge \bar{\sigma}^k = \sigma^n\}$:
The n+1-st step does not belong to the previous $\mathcal{IO}$-block in the IH schedule, it does not start a new $\mathcal{IO}$-block and machine σ^n was run before already. Hence we have to reorder the step to the end of the last $\mathcal{IO}$-block of σ^n, namely we insert it before step j in $\underrightarrow{\bar{\sigma}}$ which is the first step after the last $\mathcal{IO}$-block of $\sigma^n = p$.

$$u(i) = \begin{cases} \bar{u}(i) & : \quad i < j \\ n & : \quad i = j \\ \bar{u}(i-1) & : \quad \text{otherwise} \end{cases}$$

We construct our $\mathcal{IO}$-block schedule with $\hat{\sigma}$ and $e\hat{x}t$ as follows.

$$\hat{\sigma}^i \quad = \quad \begin{cases} \bar{\sigma}^i & : \quad i < j \\ \sigma^n & : \quad i = j \\ \bar{\sigma}^{i-1} & : \quad i > j \end{cases} \qquad e\hat{x}t^i \quad = \quad \begin{cases} e\bar{x}t^i & : \quad i < j \\ ext^n & : \quad i = j \\ e\bar{x}t^{i-1} & : \quad i > j \end{cases}$$

The reordering is implemented by setting $\hat{\sigma}^j = \sigma^n = p$ in the schedule and using the same external inputs as in the original schedule. Recall that the external inputs for other machines are redundant in this step and also no other mahine than the dedicated receiver p can react to or acknowledge its external inputs .Thus we can just move them to step j in the new schedule such that p observes the same external inputs. We will later argue that due to soundness conditions machine p will also see at most the same (if not less) internal inputs from other machines.

The execution sequence $\underrightarrow{\hat{C}}$ is computed accordingly.

$$\underbrace{\hat{C}^0}_{=\bar{C}^0} \longrightarrow^{j}_{\Delta,\bar{\sigma},e\bar{x}t} \underbrace{\hat{C}^j}_{=\bar{C}^j} \longrightarrow_{\Delta,\sigma^n,ext^n} \hat{C}^{j+1} \longrightarrow^{n-j}_{\Delta,\hat{\sigma},e\hat{x}t} \hat{C}^{n+1}$$

In order to show the validity of the new schedule we compare it to another schedule created by just appending the n+1-st step of $\underrightarrow{\sigma}$ to the IH schedule $\underrightarrow{\bar{\sigma}}$ as in the three previous cases.

$$\bar{\sigma}^n \quad = \quad \sigma^n \qquad\qquad e\bar{x}t^n \quad = \quad ext^n \qquad\qquad \bar{C}^{n+1} \quad = \quad C^{n+1}$$

For such a trace all desired properties besides $\mathcal{IO}sched$ obviously hold as shown before. On the other hand $\underrightarrow{\hat{C}}$, $\underrightarrow{\hat{\sigma}}$ and $\underrightarrow{e\hat{x}t}$ represent an $\mathcal{IO}$-block schedule. We conduct the reordering proof below in five consecutive steps as shown in Figure 2.

I. We have to show the safety and equivalence of $\underrightarrow{\hat{C}}$ wrt. $\underrightarrow{\bar{C}}$. For all $i < j$ we have $\hat{\sigma}^i = \bar{\sigma}^i$ and $e\hat{x}t^i = e\bar{x}t^i$ by construction. Since also $\hat{C}^0 = \bar{C}^0$ induction hypothesis gives us $\hat{C}^j = \bar{C}^j$ and $safety(\underrightarrow{\hat{C}}, j)$.
Examining the recursive definition of $\mathcal{IO}sched$ we see that in general we have the following inductive property.

$$(\exists x > 0.\, \mathcal{IO}sched(\underrightarrow{C}, x)) \Longrightarrow \forall y < x.\, \mathcal{IO}sched(\underrightarrow{C}, y)$$

$$\begin{tikzcd}
\bar{D}^0 \arrow[r, "\bar{\sigma}[0\,:\,j-1]", "e\bar{x}t[0\,:\,j-1]"'] \arrow[d, dashed, <->, "="] & \bar{D}^j \arrow[rr, "\bar{\sigma}[j\,:\,n-1]", "e\bar{x}t[j\,:\,n-1]"'] \arrow[d, dashed, <->, "="', "\text{I.}"] \arrow[rr, dashed, <->, bend left, "{\approx_p}", "\text{II.}"'] \arrow[dr, dashed, <->, "\approx_{\overline{p}}", "\text{III.}"'] & & \bar{D}^n \arrow[r, "p", "ext^n"'] \arrow[r, dashed, <->, bend left, "\approx_{\overline{p}}", "\text{III.}"'] \arrow[dll, dashed, <->, "\approx_p", "\text{II.}"'] \arrow[dr, dashed, <->, "\approx_{\overline{p}}", "\text{IV.}"'] & \bar{D}^{n+1} \arrow[d, dashed, <->, "=", "\text{V.}"'] \arrow[dll, dashed, <->, "\approx_p", "\text{III.}"'] \\
\hat{D}^0 \arrow[r, "\bar{\sigma}[0\,:\,j-1]", "e\bar{x}t[0\,:\,j-1]"'] & \hat{D}^j \arrow[r, "p", "e\hat{x}t^j"'] \arrow[r, dashed, <->, bend right, "\approx_{\overline{p}}"', "\text{III.}"] & \hat{D}^{j+1} \arrow[rr, "\bar{\sigma}[j\,:\,n-1]", "e\bar{x}t[j\,:\,n-1]"'] \arrow[rr, dashed, <->, bend right, "\approx_p"', "\text{V.}"] & & \hat{D}^{n+1}
\end{tikzcd}$$

Figure 2: Proof sketch for consistency and safety of the lower reordered *Cosmos* model trace $\underset{\rightarrow}{\hat{C}}, \underset{\rightarrow}{\hat{\sigma}}, \underset{\rightarrow}{e\hat{x}t}$ wrt. the upper extended IH trace $\underset{\rightarrow}{\bar{C}}, \underset{\rightarrow}{\bar{\sigma}}, \underset{\rightarrow}{e\bar{x}t}$. Step I. relies on the induction hypothesis and the equality of the traces, Step II. applies Lemma 5.1, in Step III. we use Lemma 3.1 and 4, Lemma 5.2 enables Step IV., we conclude by applying Lemma 5.1 on the $\mathcal{IO}$-block schedule in Step V. Straight arrows indicate Δ state transitions using the given schedule and external input parameters. Dashed arrows represent equivalence relations as annotated.

Thus $\mathcal{IO}sched(\underset{\rightarrow}{\bar{C}}, j)$ holds by IH i.e., the first j steps of the trace leading into configuration $\hat{C}^j$ are an $\mathcal{IO}$-block schedule and since these steps are identical for both schedules we immediately obtain $\mathcal{IO}sched(\underset{\rightarrow}{\hat{C}}, j)$. Also we know that $j-1$ was the last step when p was running in the IH schedule. As we now schedule the original n+1-st step we have $\hat{\sigma}^j = p = \hat{\sigma}^{j-1}$. Therefore $\mathcal{IO}sched(\underset{\rightarrow}{\hat{C}}, j+1)$ holds by the definition of $\mathcal{IO}sched$.

Similarly we know that the first j steps of the new schedule are a valid reordering according to induction hypothesis and the identity of the execution traces.

II. From $\hat{C}^j = \bar{C}^j$ we directly obtain $\hat{C}^j \approx_p \bar{C}^j$ and $\hat{C}^j \approx_{\overline{p}} \bar{C}^j$ using Corollary 2.1. In the current case we know that p is not scheduled in $\bar{\sigma}$ until the n+1-st step and from the safety of the steps until $j-1$ we deduce $inv(\bar{C}^j)$ by inductive application of Lemma 1. Therefore and since the next $n-j$ steps in $\underset{\rightarrow}{\bar{C}}$ are safe by induction hypothesis we can apply Lemma 5.1 obtaining:

$$\hat{C}^j \approx_p \bar{C}^n \qquad inv(\bar{C}^n)$$

Also $\hat{C}^j \sim_p \bar{C}^n$ holds by definition of $\approx_p$. Since the steps j to $n-1$ are safe in $\underset{\rightarrow}{\bar{C}}$ other machines $p' \neq p$ do not deactivate any outputs that p observes during their steps. They might activate outputs for p though. When they are not stepped their outputs for p are constant due to $constr_\omega(p')$ and the fact that p's visible components do not change. Therefore and by $e\hat{x}t^j = e\bar{x}t^n$ we have $in(\hat{C}^j, p, e\hat{x}t^j) \subseteq in(\bar{C}^n, p, e\bar{x}t^n)$. The construction $\hat{\sigma}^j = \bar{\sigma}^n = \sigma^{u(j)} = p$ and the third condition of $constr_{\mathcal{IO}}(p)$ then gives us:

$$\mathcal{IO}_{\hat{\sigma}^j}(\hat{C}^j) = \mathcal{IO}_{\bar{\sigma}^n}(\bar{C}^n) = \mathcal{IO}_{\sigma^{u(j)}}(C^{u(j)})$$

Since the step is local we deduce that the first $j+1$ steps of the schedule are a valid reordering of the original trace.

$$\forall i < j+1.\ \underset{\rightarrow}{C}, \underset{\rightarrow}{\hat{C}}, i \vdash u\surd_{reord}$$

III. In both systems p is performing the next step in these configurations because $\hat{\sigma}^j = \bar{\sigma}^n = p$. Moreover $safe_{\bar{\sigma}^n}(\bar{C}^n)$ holds by precondition, IH and Lemma 3.1. We already argued above that $in_p(\hat{C}^j) = in_p(\bar{C}^j) \subseteq in_p(\bar{C}^n)$ thus by Lemma 4 we know:

$$\hat{C}^{j+1} \approx_p \bar{C}^{n+1} \qquad \hat{C}^j \approx_{\overline{p}} \hat{C}^{j+1} \qquad \bar{C}^n \approx_{\overline{p}} \bar{C}^{n+1}$$

From Lemma 3.1 we get $safe_{\hat{\sigma}^j}(\hat{C}^j)$ and Lemma 1 yields $inv(\hat{C}^{j+1})$. With the soundness of the local step and $constr_\omega(p')$ we also see that $\forall p' \in \mathbb{N}_{np}.\ \omega_{p'}(\hat{C}^{j+1}) = \omega_{p'}(\hat{C}^j) = \omega_{p'}(\bar{C}^j)$ as we have $\hat{C}^j \overset{v}{\sim} \hat{C}^{j+1}$.

IV. For $i \le j$ we already have $safe(\hat{C}^i, \hat{\sigma}^i, \hat{ext}^i)$ as shown above. From $\hat{C}^{j+1}$ or $\bar{C}^j$, respectively, both systems perform the same $n-j$ steps as $\hat{\sigma}(i) = \bar{\sigma}(i-1)$ and $\hat{ext}^i = \bar{ext}^{i-1}$ holds for all $i \in [j+1:n]$. Using $\hat{C}^{j+1} \approx_{\overline{p}} \hat{C}^j = \bar{C}^j$ Lemma 5.2 implies $\hat{C}^i \approx_{\overline{p}} \bar{C}^{i-1}$ and $\omega_p(\hat{C}^i) = \omega_p(\bar{C}^{i-1})$ for each i. The former yields $\hat{C}^i \sim_{\hat{\sigma}^{i-1}} \bar{C}^{i-1}$ hence $\hat{c}^i_{\hat{\sigma}^i} = \bar{c}^{i-1}_{\bar{\sigma}^{i-1}}$ and for all $p' \neq p$ also $\omega_{p'}(\hat{C}^i) = \omega_{p'}(\bar{C}^{i-1})$ by definition of $\approx_{\overline{p}}$ and $constr_\omega(p')$.

Thus in the corresponding configurations all machines but p have the same configuration and the same inputs are observed in the system. Consequently we have $\mathcal{IO}_{\hat{\sigma}^i}(\hat{C}^i) = \mathcal{IO}_{\bar{\sigma}^{i-1}}(\bar{C}^{i-1})$ by the first condition of $constr_{\mathcal{IO}}(\hat{\sigma}^i)$ and with IH as well as the definition of $\mathcal{IO}sched$ induction on i yields:

$$\mathcal{IO}sched(\underset{\rightarrow}{\hat{C}}, i+1)$$

For $i = n$ we finally obtain the desired $\mathcal{IO}sched(\underset{\rightarrow}{\hat{C}}, n+1)$ and by Lemma 5.2 we obtain:

$$\hat{C}^{n+1} \approx_{\overline{p}} \bar{C}^n \qquad safety(\underset{\rightarrow}{\hat{C}}, n+1)$$

As stated above we have $\hat{C}^i \sim_{\hat{\sigma}^{i-1}} \bar{C}^{i-1}$ and the steps i in $\underset{\rightarrow}{\hat{C}}$ and steps $i-1$ in $\underset{\rightarrow}{\bar{C}}$ are consistent wrt. predicate $\mathcal{IO}$. By induction hypothesis and construction, consistency holds also with the original schedule steps $u(i) = \bar{u}(i-1)$. Hence $\hat{C}^i \approx_{\overline{p}} C^{u(i)}$. By Lemma 5.2 we have $\omega_p(\hat{C}^i) = \omega_p(C^{u(i)})$, i.e., outputs are consistent and by $\hat{ext}^i = ext^{u(i)}$ the respective machines see the same inputs during these steps.

$$in_{\hat{\sigma}^i}(\hat{C}^i) = in_{\hat{\sigma}^i}(\bar{C}^{i-1}) \overset{\text{IH}}{=} in_{\hat{\sigma}^i}(C^{\bar{u}(i-1)}) = in_{\hat{\sigma}^i}(C^{u(i)})$$

With the construction $\hat{\sigma}^i = \bar{\sigma}^{i-1}$ and $u(i) = \bar{u}(i-1)$ for $j < i \leq n$ we get the desired $\hat{\sigma}^i = \sigma^{u(i)}$ by induction hypothesis.

$$\hat{\sigma}^i = \bar{\sigma}^{i-1} \overset{\text{IH}}{=} \sigma^{\bar{u}(i-1)} = \sigma^{u(i)}$$

From $\hat{C}^i \approx_{\overline{p}} \bar{C}^{(i-1)}$ we deduce $\hat{C}^i \overset{sv}{\sim} C^{u(i-1)}$ and $\hat{C}^i \overset{o}{\sim} C^{u(i-1)}$ by Corollary 2.3. From $\hat{C}^i \sim_{\hat{\sigma}^{i-1}} \bar{C}^{i-1}$ we get $\hat{C}^i \sim_{\hat{\sigma}^{i-1}} C^{u(i-1)}$. Moreover $\hat{C}^j \approx_p \bar{C}^n = C^n = C^{u(j)}$ yields $\hat{C}^j \sim_p C^{u(j)}$ as well as $\hat{C}^j \overset{o}{\sim}_p C^{u(j)}$. Thus we obtain the validity of the reordered schedule.

$$\forall i < n+1.\ \underrightarrow{C}, \underrightarrow{\hat{C}}, i \vdash u\surd_{reord}$$

V. By applying Lemma 5.1 on $\hat{C}^{j+1} \approx_p \bar{C}^{n+1}$ for the steps from configuration $\hat{C}^{j+1}$ to $\hat{C}^{n+1}$ we get $\hat{C}^{n+1} \approx_p \bar{C}^{n+1}$ using the safety of all these steps shown above. For the last step in $\underrightarrow{\bar{C}}$ we know the environment is preserved from the invocation of Lemma 4 in Step III. yielding $\bar{C}^n \approx_{\overline{p}} \bar{C}^{n+1}$. By the transitivity of $\approx_{\overline{p}}$ we obtain $\hat{C}^{n+1} \approx_{\overline{p}} \bar{C}^{n+1}$. Consequently by Corollary 2.1 we conclude:

$$\hat{C}^{n+1} \approx_p \bar{C}^{n+1} \wedge \hat{C}^{n+1} \approx_{\overline{p}} \bar{C}^{n+1} \iff \hat{C}^{n+1} = \bar{C}^{n+1}$$

Since all four cases are correct we have shown that for every safe $n+1$-step *Cosmos* model computation $\underrightarrow{C}$ we can find a safe and equivalent $\mathcal{IO}$-block schedule $\underrightarrow{\hat{C}}, \underrightarrow{\hat{\sigma}}, \underrightarrow{e\hat{x}t}$ that is a valid reordering of the original one. □

4.4.2 Soundness

We have to show the following statement to ensure soundness of reordering.

$$\begin{array}{rcl} \forall \underrightarrow{C}, \underrightarrow{\sigma}, \underrightarrow{ext}, n & . & C^0 \longrightarrow^n_{\Delta,\sigma,ext} C^n \wedge /safety(\underrightarrow{C}, n) \\ \Downarrow & & \\ \exists \underrightarrow{\hat{C}}, \underrightarrow{\hat{\sigma}}, \underrightarrow{e\hat{x}t}, \hat{n} \leq n & . & \bigwedge \left\{ \begin{array}{ll} \hat{C}^0 \longrightarrow^{\hat{n}}_{\Delta,\hat{\sigma},e\hat{x}t} \hat{C}^{\hat{n}}, & \hat{C}^0 = C^0, \\ /safety(\underrightarrow{\hat{C}}, \hat{n}), & \mathcal{IO}sched(\underrightarrow{\hat{C}}, \hat{n}) \end{array} \right\} \end{array}$$

PROOF: We distinguish the cases $n = 0$ and $n > 0$. In the first case the term $/safety(\underrightarrow{C}, n)$ collapses to $/inv(C^0)$ by definition which follows for $\underrightarrow{\hat{C}}$ from construction $\hat{C}^0 = C^0$. The $\mathcal{IO}sched$ condition holds trivially for arbitrary $\underrightarrow{\hat{\sigma}}$, $\underrightarrow{e\hat{x}t}$ and u as no step is taken if $\hat{n} = 0$.

For $n > 0$ we examine the trace closer and search for the first step j that is unsafe assuming that $inv(C^0)$ holds. Otherwise we set $\hat{n} = 0$ and conclude analoguously to the first case.

$$j = \min\{i \mid /safe_{\sigma^i}(C^i)\}$$

Observe that $safety(\underrightarrow{C}, j)$ holds, thus we can apply the existence theorem we proved above on the first j steps. We obtain inverse reordering function u and a safe $\mathcal{IO}$-block schedule $\underrightarrow{\bar{C}}, \underrightarrow{\bar{\sigma}}, \underrightarrow{e\bar{x}t}$ with $\bar{C}^0 = C^0$:

$$\bar{C}^0 \longrightarrow_{\Delta,\bar{\sigma}^0,e\bar{x}t^0} \bar{C}^1 \longrightarrow_{\Delta,\bar{\sigma}^1,e\bar{x}t^1} \cdots \longrightarrow_{\Delta,\bar{\sigma}^{j-1},e\bar{x}t^{j-1}} \bar{C}^j$$

Again we have $safety(\underrightarrow{\bar{C}}, j)$ and in addition $\mathcal{IO}sched(\underrightarrow{\bar{C}}, j)$ and $\bar{C}^j = C^j$. Because of the latter we also have $/safe_{\bar{\sigma}^j}(\bar{C}^j)$ if we set $\bar{\sigma}^j = \sigma^j$ and $e\bar{x}t^j = ext^j$. Now there are two cases.

Either $\mathcal{IO}sched(\underrightarrow{\bar{C}}, j+1)$, then our soundness claim holds with

$$\underrightarrow{\hat{C}} = \underrightarrow{\bar{C}}, C^{j+1} \qquad \underrightarrow{\hat{\sigma}} = \underrightarrow{\bar{\sigma}}, \sigma^j \qquad \underrightarrow{e\hat{x}t} = \underrightarrow{e\bar{x}t}, ext^j$$

and $\hat{n} = j+1$ using induction hypothesis.

Otherwise we have to reorder the unsafe j+1-st step to get an $\mathcal{IO}$-block schedule. Note that by the definition of $\mathcal{IO}sched$ this step must be a local operation of a processor $p = \sigma^j \neq \bar{\sigma}^{j-1}$ which was scheduled before already. We reorder it after the last preceding step $k-1$ of p in the schedule.

$$k = \max\{i+1 \mid i < j \wedge \bar{\sigma}^i = p\}$$

The reordered schedule $\underrightarrow{\hat{\sigma}}$ and the corresponding external input sequence $\underrightarrow{e\hat{x}t}$ are defined as follows.

$$\hat{\sigma}^i = \begin{cases} \bar{\sigma}^i & : \quad i < k \\ p & : \quad i = k \end{cases} \qquad e\hat{x}t^i = \begin{cases} e\bar{x}t^i & : \quad i < k \\ ext^j & : \quad i = k \end{cases}$$

For the resulting system trace $\underrightarrow{\hat{C}}$ we have $\hat{C}^i = \bar{C}^i$ for $i \leq k$. Configuration $\hat{C}^{k+1}$ is the result of the first unsafe step on processor p and we do not care about any subsequent system states. Obviously $\underrightarrow{\hat{C}}, \underrightarrow{\hat{\sigma}}, \underrightarrow{e\hat{x}t}[0 : k-1]$ is an $\mathcal{IO}$-block schedule and from $\hat{\sigma}^k = \hat{\sigma}^{k-1}$ we obtain $\mathcal{IO}sched(\underrightarrow{\hat{C}}, k+1)$. By $\hat{C}^k = \bar{C}^k$ and Corollary 2.1 we have $\hat{C}^k \approx_p \bar{C}^k$. Since all steps until j are safe in schedule $\underrightarrow{\bar{\sigma}}$ we can apply Lemma 5.1 and it follows:

$$\hat{C}^k \approx_p \bar{C}^j \overset{\text{IH}}{=} C^j \qquad \overset{\text{Def.}}{\Longrightarrow} \qquad \hat{C}^k \sim_p C^j \wedge \hat{C}^k \overset{o}{\sim}_p C^j$$

We have a soundness condition for outputs of machines $p' \neq p$. Thus all machines keep their active outputs for p stable during the steps $\underrightarrow{\bar{\sigma}}[k : j-1]$ because p is not running then and the steps are safe.

$$\forall i \in [k : j-1], p', \alpha \in \mathfrak{O}_{p'}.\ \alpha \in \mathfrak{I}_p \cap \omega_{p'}(\bar{C}^i) \Longrightarrow \alpha \in \omega_{p'}(\bar{C}^{i+1})$$

By construction p also sees the same external inputs in $\hat{C}^k$ and C^j. Then we have $in_p(\hat{C}^k) \subseteq in_p(C^j)$ and $constr_{safe}(p)$ yields the unsafeness of step k in $\underrightarrow{\hat{\sigma}}$. We conclude $/safety(\underrightarrow{\hat{C}}, \hat{n})$ with $\hat{n} = k+1$. $\square$

4.5 Application in System Verification

The value of a theorem shows itself in its application. As a sanity check, i.e., to prove that the order reduction theorem we have just presented is useful, we consider a scenario where we have verified the safety of all system traces that are $\mathcal{IO}$-block schedules of length $\hat{n}$. In addition we assume that an invariant P

has been shown for all configurations of the trace. We sum up the verification result in the following predicate for an initial state $\hat{C}^0$ and the property P.

$$\begin{array}{l} verified_{\mathcal{IO}}(C_0, \hat{n}, P) \equiv \\ \quad \forall \underrightarrow{\hat{C}}, \underrightarrow{\hat{\sigma}}, \underrightarrow{e\hat{x}t}. \\ \qquad \bigwedge \left\{ \begin{array}{l} \hat{C}_0 \longrightarrow^{\hat{n}}_{\Delta,\hat{\sigma},e\hat{x}t} \hat{C}^{\hat{n}},\ \hat{C}^0 = C^0, \\ \mathcal{IO}sched(\underrightarrow{\hat{C}}, \hat{n}) \end{array} \right\} \Longrightarrow \bigwedge \left\{ \begin{array}{l} safety(\underrightarrow{\hat{C}}, \hat{n}), \\ \forall i \leq \hat{n}.\ P(\hat{C}^i) \end{array} \right\} \end{array}$$

Observe that $verified_{\mathcal{IO}}(\hat{C}^0, \hat{n}, P)$ implies $\forall i \leq \hat{n}.\ verified_{IO}(\hat{C}^0, i, P)$. Also among the most interesting properties are invariants on shared resources. Hence we assume for P here that it only depends on shared memory, the ownership configuration and visible machine components.

$$\forall C, C'.\ C \overset{sv}{\sim} C' \wedge C \overset{o}{\sim} C' \Longrightarrow P(C) = P(C')$$

Now it would be desirable if we could use the reordering theorem to infer the safety as well as the validity of P for all traces independent of scheduling.

$$\begin{array}{l} \forall \underrightarrow{C}, \underrightarrow{\sigma}, \underrightarrow{ext}, n. \\ \quad \bigwedge \left\{ \begin{array}{l} verified_{\mathcal{IO}}(C^0, n, P), \\ C^0 \longrightarrow^{n}_{\Delta,s,ext} C^n \end{array} \right\} \Longrightarrow \bigwedge \left\{ \begin{array}{l} safety(\underrightarrow{C}, n), \\ \forall i \leq n.\ P(C^i) \end{array} \right\} \end{array}$$

PROOF: We concentrate on the safety property first. If we look at the soundness part of our reordering theorem and take the complement we get:

$$\begin{array}{l} \forall \underrightarrow{C}, \underrightarrow{\sigma}, \underrightarrow{ext}, n. \\ \left(\begin{array}{l} \forall \hat{n} \leq n. \forall \underrightarrow{\hat{C}}, \underrightarrow{\hat{\sigma}}, \underrightarrow{e\hat{x}t}, u.\ \bigwedge \left\{ \begin{array}{l} \hat{C}^0 \longrightarrow^{\hat{n}}_{\Delta,\hat{\sigma},e\hat{x}t} C^{\hat{n}},\ \hat{C}^0 = C^0, \\ \mathcal{IO}sched(\underrightarrow{\hat{C}}, \hat{n}) \end{array} \right\} \\ \qquad\qquad\qquad\qquad\qquad \Longrightarrow safety(\underrightarrow{\hat{C}}, \hat{n}) \end{array} \right) \\ \qquad\qquad \Downarrow \\ \quad safety(\underrightarrow{C}, n) \vee \overline{C^0 \longrightarrow^{n}_{\Delta,\sigma,ext} C^n} \end{array}$$

The implication in the bracket term above is derived from the negation of the big conjunction and by application of DeMorgan's Law pulling out the $safety$ term. The term $\overline{C^0 \longrightarrow^{n}_{\Delta,s,e\hat{x}t} C^n}$ can be transformed into an antecedent of the overall implication as well. For all logical propositions A, B and C the following implication holds.

$$(A \Rightarrow B \wedge C) \Longrightarrow (A \Rightarrow B)$$

Then the bracket implication is implied by $verified_{IO}(\hat{C}^0, \hat{n}, P)$ for all $\hat{n} \leq n$ if we replace C with $\forall i \leq n.\ P(C^i)$. Hence the soundness theorem implies:

$$\begin{array}{l} \forall \underrightarrow{C}, \underrightarrow{\sigma}, \underrightarrow{ext}, n. \\ \quad \bigwedge \left\{ \begin{array}{l} \forall \hat{n} \leq n.\ verified_{IO}(\hat{C}^0, \hat{n}, P), \\ C^0 \longrightarrow^{n}_{\Delta,\sigma,ext} C^n \end{array} \right\} \Longrightarrow safety(\underrightarrow{C}, n) \end{array}$$

As $\forall \hat{n} \leq n.\ verified_{IO}(\hat{C}^0, \hat{n}, P)$ is implied by $verified_{IO}(\hat{C}^0, n, P)$ we have:

$$\begin{array}{l} \forall \underrightarrow{C}, \underrightarrow{\sigma}, \underrightarrow{ext}, n. \\ \quad \bigwedge \left\{ \begin{array}{l} verified_{\mathcal{IO}}(C^0, n, P), \\ C^0 \longrightarrow^{n}_{\Delta,\sigma,ext} C^n \end{array} \right\} \Longrightarrow safety(\underrightarrow{C}, n) \end{array}$$

For the proof of property transfer we can use the safety of all traces of the system which follows from the verified safety of all $\mathcal{IO}$-block schedules as we just have shown. Now for the sake of contradiction we assume a safe trace $\underset{\rightarrow}{C}$ and search the minimal index $j \leq n$ for which $P(C^j)$ does not hold. Then by existence of the reordering there must exist an equivalent n-step $\mathcal{IO}$-block schedule with configurations $\underset{\rightarrow}{\hat{C}}$ such that $\hat{C}^0 = C^0$ and $\hat{C}^n = C^n$. Moreover that schedule is a valid reordering of the original one and with the inverse reordering function u we have:

$$\forall i < n.\ \underset{\rightarrow}{C}, \underset{\rightarrow}{\hat{C}}, i \vdash u\surd_{reord}$$

If $j = 0$ or $j = n$ we immediately have $\hat{C}^0 = C^0$ and $\neg P(\hat{C}^0)$ or $\hat{C}^n = C^n$ and $\neg P(\hat{C}^n)$ respectively.

If $0 < j < n$ and the next machine σ^j in the original schedule is about to perform an $\mathcal{IO}$ step i.e. $\mathcal{IO}_{\sigma^j}(C^j)$ holds, then we know that there exists a corresponding $\mathcal{IO}$ step $k < n$ in the reordered schedule such that $u(k) = j$. Also from $u\surd_{reord}$ we get $\hat{C}^k \overset{sv}{\sim} C^j$ and $\hat{C}^k \overset{o}{\sim} C^j$ and it follows $\neg P(\hat{C}^k)$.

In the case of a local step the shared and visible system state must have been altered by a preceding $\mathcal{IO}$ step of machine σ^{j-1}, since C^j is the earliest configuration where P does not hold and only $\mathcal{IO}$ steps may modify shared information. Thus $\mathcal{IO}_{\sigma^{j-1}}(C^{j-1})$ and we search for the step $k-1$ such that we have $u(k-1) = j-1$. Application of $u\surd_{reord}$ gives us:

$$\hat{C}^{k-1} \sim_{\hat{\sigma}^{j-1}} C^{j-1} \qquad \hat{\sigma}^{k-1} = \hat{\sigma}^{j-1} \qquad \hat{C}^{k-1} \overset{sv}{\sim} C^{j-1}$$

Consequently we have $\hat{C}^k \overset{sv}{\sim} C^j$ by constraint $constr_\delta(\sigma^{j-1})$ and since the ownership sets of other machines remain unchanged due to construction of Δ we have $\hat{C}^k \overset{o}{\sim} C^j$, too. Hence we obtain again that P doesn't hold in the $\mathcal{IO}$-block schedule. Overall we have:

$$\exists k \leq n. \quad \neg P(\hat{C}^k)$$

However the $verified_{\mathcal{IO}}(C^0, n, P)$ predicate gives us

$$\forall i \leq n.\ P(\hat{C}^i)$$

exposing the contradiction. Thus a trace $\underset{\rightarrow}{C}$ with $\neg P(C^j)$ cannot exist and the verified property P holds for all schedules and configurations. □

5 $\mathcal{IO}$-Block Machine Semantics

Based on the order reduction we now want to explore how to apply local simulation theorems in a concurrent context. Our goal is to state and prove a global *Cosmos* model simulation theorem which argues that the local simulation theorems still hold on individual machines. In particular the simulation relation holds for a machine when it reaches an $\mathcal{IO}$ point. Since we may assume $\mathcal{IO}$ block schedules, semantics can first be simplified. When introducing simulation theorems on *Cosmos* models later on it is convenient to define semantics where we consecutively execute $\mathcal{IO}$-blocks. We call the machine implementing such semantics the *$\mathcal{IO}$-block machine* or short the *block machine*.

5.1 Definition

The block machine execution depends on the follow parameters.

- $\underrightarrow{\kappa} : \mathbb{N}^*_{np}$ - the block schedule
- $\underrightarrow{\lambda} : \mathbb{N}^*$ - the length of the blocks to be executed
- $\underrightarrow{ext} :^* 2^{\mathfrak{E}}$ - a sequence of external inputs for every block

The transition function $\Delta_B : \mathbb{C} \times \mathbb{N}_{np} \times 2^{\mathfrak{E}} \times \mathbb{N} \to \mathbb{C}$ is defined as follows.

$$\Delta_B(C, p, l, ext) = \begin{cases} \Delta_B(\Delta(C, p, ext), p, l-1, \emptyset) & : \quad l > 0 \\ C & : \quad l = 0 \end{cases}$$

We simply execute step by step until the end of the block. The first step of the block is usually an $\mathcal{IO}$ step so it might depend on the external input. All other steps must be non-$\mathcal{IO}$ steps which do not need any inputs because they are by construction never reacting to them. We define a validity prdicate for block machine executions $\underrightarrow{C}, \underrightarrow{\kappa}, \underrightarrow{ext}, \underrightarrow{\lambda}$ by defining the $\mathcal{IO}sched_B$ predicate.

$$\begin{aligned} &\mathcal{IO}sched_B(\underrightarrow{C}, \underrightarrow{\kappa}, \underrightarrow{\lambda}, \underrightarrow{ext}, n) \equiv \\ &\qquad \forall i < n. \bigwedge \left\{ \begin{array}{l} \mathcal{IO}_{\kappa^i}(C^i) \vee \forall j < i.\ \kappa^j \neq \kappa^i, \quad \lambda^i \neq 0, \\ \forall j \in (0, \lambda^i).\ /\mathcal{IO}_{\kappa^i}(\Delta_B(C^i, \kappa^i, j, ext^i), p, \emptyset) \end{array} \right\} \end{aligned}$$

For further definitions we introduce some notation to talk about $\mathcal{IO}$ steps of machines p in *Cosmos* model computations.

$$\begin{aligned} nio(\underrightarrow{C}, \underrightarrow{\sigma}, \underrightarrow{ext}, n) &= \begin{cases} 0 & : \quad n = 0 \\ nio(n-1) + 1 & : \quad n > 0 \wedge \\ & \qquad \bigvee \left\{ \begin{array}{l} \mathcal{IO}_{\sigma^{n-1}}(C^{n-1}), \\ \forall i < n-1.\ \sigma^i \neq \sigma^{n-1} \end{array} \right\} \\ nio(n-1) & : \quad \text{otherwise} \end{cases} \\ iop(\underrightarrow{C}, \underrightarrow{\sigma}, \underrightarrow{ext}, i) &= \max\{j \mid nio(j) = i\} \end{aligned}$$

Here nio counts the $\mathcal{IO}$ steps passed until configuration n in a computation $\underrightarrow{C}, \underrightarrow{\sigma}, \underrightarrow{ext}$. Conversely iop returns the step index of the $\mathcal{IO}$ step i. Note that in this definition first steps of machines are also counted as $\mathcal{IO}$ steps. Basically $nio(n)$ tells us how many blocks must be formed out of an n-step $\mathcal{IO}$-block schedule. We use the following shorthands.

$$iop(i) \mathrel{\hat{=}} iop(\underrightarrow{C}, \underrightarrow{\sigma}, \underrightarrow{ext}, i) \qquad\qquad nio(n) \mathrel{\hat{=}} nio(\underrightarrow{C}, \underrightarrow{\sigma}, \underrightarrow{ext}, n)$$

Also here predicate parameters of the form $\underrightarrow{C}, \underrightarrow{\kappa}, \underrightarrow{\lambda}, \underrightarrow{ext}$ are abbreviated by $\underrightarrow{C}$.

5.2 Simulation Theorem and Proof

In order to justify the block machine semantics we conduct a simulation proof between safe $\mathcal{IO}$-block schedules and safe executions of the block machine.

Furthermore we show that for every unsafe $\mathcal{IO}$-block trace there exists a consistent unsafe block execution. First we need to introduce safety for the block machine.

$$safe_B(C_0, p, l, ext_0) \equiv$$
$$\forall \underrightarrow{C}, \underrightarrow{\sigma}, \underrightarrow{ext}. \bigwedge \left\{ \begin{array}{l} C^0 = C_0,\ \sigma^0 = p,\ ext^0 = ext_0, \\ \forall i \in (0, l).\ \sigma^i = p \wedge ext^i = \emptyset \end{array} \right\} \Longrightarrow safety(\underrightarrow{C}, l)$$

According to the definition a block step out of a configuration C_0 is safe if all fine-grained steps inside the executed block are safe. The $safety$ predicate is then defined similarly to the one for *Cosmos* model traces.

$$\frac{C^0 \longrightarrow^n_{\Delta_B, \kappa, \lambda, ext} C^n \qquad inv(C^0) \qquad \forall i < n.\ safe_B(C^i, \kappa^i, \lambda^i, ext^i)}{safety_B(\underrightarrow{C}, \underrightarrow{\kappa}, \underrightarrow{\lambda}, \underrightarrow{ext}, n)}$$

Theorem 2 *(Block Machine Simulation) We consider the simulation between an n-step $\mathcal{IO}$-block schedule $\underrightarrow{C}, \underrightarrow{\sigma}, \underrightarrow{ext}$ and a corresponding $\mathcal{IO}$-block machine computation $\underrightarrow{\hat{C}}, \underrightarrow{\kappa}, \underrightarrow{\lambda}, \underrightarrow{e\hat{x}t}$ starting in the same configuration and show the properties below.*

1. *(Existence) For any safe $\mathcal{IO}$-block schedule we can find a safe block machine execution trace that simulates the first one at all passed $\mathcal{IO}$ points.*

$$\begin{array}{lcl} \forall \underrightarrow{C}, \underrightarrow{\sigma}, \underrightarrow{ext}, n & . & safety(\underrightarrow{C}, n) \wedge \mathcal{IO}sched(\underrightarrow{C}, n) \\ \Downarrow & & \\ \exists \underrightarrow{\hat{C}}, \underrightarrow{\kappa}, \underrightarrow{\lambda}, \underrightarrow{e\hat{x}t} & . & \bigwedge \left\{ \begin{array}{l} \hat{C}^0 = C^0, \quad safety_B(\underrightarrow{\hat{C}}, nio(n)), \\ \mathcal{IO}sched_B(\underrightarrow{\hat{C}}, nio(n)), \\ \forall i < nio(n). \\ \quad \bigwedge \left\{ \begin{array}{l} \hat{C}^i = C^{iop(i)},\ \kappa^i = \sigma^{iop(i)}, \\ e\hat{x}t^i = ext^{iop(i)},\ \sum_{j=0}^{i-1} \lambda^j = iop(i) \end{array} \right\} \end{array} \right\} \end{array}$$

2. *(Soundness) For any unsafe $\mathcal{IO}$-block schedule there exists an unsafe block machine execution starting in the same configuration consisting of at most the same total number of machine steps.*

$$\begin{array}{lcl} \forall \underrightarrow{C}, \underrightarrow{\sigma}, \underrightarrow{ext}, n & . & \bigwedge \left\{ \begin{array}{l} \forall i < n.\ C^0 \longrightarrow^i_{\Delta, \sigma, ext} C^i, \\ /safety(\underrightarrow{C}, n), \quad \mathcal{IO}sched(\underrightarrow{C}, n) \end{array} \right\} \\ \Downarrow & & \\ \exists \begin{array}{c} \underrightarrow{\hat{C}}, \underrightarrow{\kappa}, \underrightarrow{\lambda}, \underrightarrow{e\hat{x}t}, \\ \hat{n} \leq nio(n) \end{array} & . & \bigwedge \left\{ \begin{array}{l} \forall i < \hat{n}.\ \hat{C}^0 \longrightarrow^i_{\Delta_B, \kappa, \lambda, e\hat{x}t} \hat{C}^i, \\ /safety_B(\underrightarrow{\hat{C}}, \hat{n}), \quad \mathcal{IO}sched_B(\underrightarrow{\hat{C}}, \hat{n}), \\ \sum_{i=0}^{\hat{n}-1} \lambda^i \leq n \end{array} \right\} \end{array}$$

PROOF: We prove the simulation by induction on n and construct $\underrightarrow{\kappa}$ and $\underrightarrow{e\hat{x}t}$ as follows. For the schedule of the block machine we simply schedule machines in the order of their $\mathcal{IO}$ steps in the $\mathcal{IO}$ schedule with the same external inputs.

$$\kappa^i = \sigma^{iop(i)} \qquad\qquad e\hat{x}t^i = ext^{iop(i)}$$

The existence property becomes trivial in the induction start with $n = 0$. Observe that we also have $nio(n) = 0$ and $iop(0) = 0$ by definition. All claims are discharged by the identity of the initial configurations and the fact that no step is taken. For induction hypothesis we assume the claims to hold for a fixed but arbitrary n. In the induction step from $n \to n+1$ we assume a new $\mathcal{IO}$-block schedule with one step more. We have to make a case distinction on the new last step of the $n+1$-step trace.

1. $/\mathcal{IO}_{\sigma^n}(C^n)$ - the $\mathcal{IO}$-block schedule was extended by a local step of some machine. Because every prefix of a safe $\mathcal{IO}$-block schedule is also a safe $\mathcal{IO}$-block schedule, for the n steps into configuration C^n all claims hold by induction hypothesis. As no new $\mathcal{IO}$ step is added to the schedule the block machine can not make another step and there is nothing further to prove. In particular we have $nio(n+1) = nio(n)$ and the block machine schedule, step numbers and external input sequence equal the ones we get from induction hypothesis.

2. $\mathcal{IO}_{\sigma^n}(C^n)$ - Machine $\sigma^n = p'$ performs an $\mathcal{IO}$ step, therefore we have $nio(n+1) = nio(n) + 1$. For all $\mathcal{IO}$ steps $iop(i) < n$ we have simulation and safety as well as the $\mathcal{IO}$-block schedule property for the block machine execution $\underrightarrow{\hat{C}}, \underrightarrow{\kappa}, \underrightarrow{\lambda}, \underrightarrow{e\hat{x}t}$ by induction hypothesis.

$$\text{IH} \Longrightarrow \bigwedge \begin{Bmatrix} \hat{C}^0 = C^0, \quad safety_B(\underrightarrow{\hat{C}}, nio(n)), \quad \mathcal{IO}sched_B(\underrightarrow{\hat{C}}, nio(n)), \\ \forall i < nio(n).\ \hat{C}^i = C^{iop(i)} \wedge \sum_{j=0}^{i-1} \lambda^j = iop(i) \end{Bmatrix}$$

In the block machine execution we set $\kappa^{nio(n)} = p'$, $e\hat{x}t^{nio(n)} = ext^n$ and examine the last executed block step $m = nio(n) - 1$. By induction hypothesis we also have $\hat{C}^m = C^{iop(m)}$ for the start of that last block by machine $\kappa^m = p$. Because we have an $\mathcal{IO}$-block schedule, all l steps from $C^{iop(m)}$ into C^n are executed by p, hence we set $\lambda^m = l = n - iop(m) > 0$. Let $\underrightarrow{\bar{C}}, \underrightarrow{\bar{\sigma}}, \underrightarrow{e\bar{x}t}$ be the internal computation of the block starting in $\bar{C}^m$.

$$\forall i \in [0:l].\ \hat{C}^m \longrightarrow^i_{\Delta,\bar{\sigma},e\bar{x}t} \bar{C}^i \wedge \bar{\sigma}^i = p \wedge e\bar{x}t^i = \begin{cases} ext^m & : \quad i = 0 \\ \emptyset & : \quad i > 0 \end{cases}$$

Note that by the definition of Δ_B we have

$$\forall i \in [0:l].\ \bar{C}^i = \Delta_B(\hat{C}^m, p, i, e\hat{x}t^m)$$

and consequently:

$$\bar{C}^l = \Delta_B(\hat{C}, \kappa^m, \lambda^m, e\hat{x}t^m) = \hat{C}^{m+1} = C^{nio(n)} = C^{nio(n+1)-1}$$

Repeated applications of Lemmata 3.2 and 3.1 yield the safety of these steps as we execute the same number steps of p as in $\underrightarrow{C}$ with consistent inputs. For the starting $\mathcal{IO}$ step into $\bar{C}^1$ we have the same inputs by construction and IH. Hence by $constr_\delta(p)$ and the identity $C^m = C^{iop(m)}$ it follows:

$$\bar{C}^1 = C^{iop(m)+1}$$

As the following non-$\mathcal{IO}$ steps are safe and the inputs are consistent we can repeatedly apply Lemma 4 giving us:

$$\forall i \in [0:l].\ \bigwedge \left\{ \begin{array}{l} \bar{C}^i \approx_p C^{iop(m)+i}, \\ C^{iop(m)+i} \approx_{\overline{p}} C^{iop(m)}, \\ \bar{C}^i \approx_{\overline{p}} \hat{C}^m \end{array} \right\}$$

Thus from $\hat{C}^{nio(n)} = \bar{C}^l \approx_p C^{iop(m)+l} = C^n$ and

$$\hat{C}^{nio(n)} = \bar{C}^l \approx_{\overline{p}} \hat{C}^m \overset{\text{IH}}{=} C^{iop(m)} \approx_{\overline{p}} C^{iop(m)+l} = C^n$$

we conclude $\hat{C}^{nio(n)} = C^n = C^{iop(nio(n))}$ by Corollary 2.1. Also we have:

$$\sum_{j=0}^{nio(n)-1} \lambda^j = \sum_{j=0}^{m} \lambda^j = \sum_{j=0}^{m-1} \lambda^j + l \overset{\text{IH}}{=} iop(m) + l = n = iop(nio(n))$$

Recall that $nio(n) = nio(n+1) - 1$. The safety $safe_B(C^m, p, l, ext^m)$ of block step m follows from the observation derived before from induction hypothesis.

$$\forall i \in [0:l).\ safe_p(\bar{C}^i)$$

If we set $\lambda^{nio(n)} = 1$ the safety of the last block step follows directly from the equality of configurations and inputs by $constr_{safe}(p)$. Therefore we also have the block trace safety property for $nio(n+1)$.

Similarly $\forall i \in (0:l).\ /\mathcal{IO}_p(\bar{C}^i, p, \emptyset)$ due to Lemma 3.1, $\mathcal{IO}_p(\hat{C}^m)$ by induction hypothesis, and by $constr_{\mathcal{IO}}(p')$ we have $\mathcal{IO}_{p'}(\hat{C}^{nio(n)})$ because $\hat{C}^{nio(n)} = C^n$ and $e\hat{x}t^{nio(n)} = ext^n$. Hence also $\mathcal{IO}sched_B(\underrightarrow{\hat{C}}, nio(n+1))$ holds. This finishes the block machine existence proof.

For the soundness proof we proceed in a way similar to that of the $\mathcal{IO}$-block reordering soundness proof. We need to find the first step $j < n$ in $\underrightarrow{C}$ that is unsafe and use the existence theorem to build an equivalent block machine trace $\underrightarrow{\hat{C}}, \underrightarrow{\kappa}, \underrightarrow{\lambda}, \underrightarrow{e\hat{x}t}$ leading into configuration $\hat{C}^k$ where:

$$k = \begin{cases} nio(j) & : \quad \mathcal{IO}_{\sigma^j}(C^j) \\ nio(j) - 1 & : \quad \text{otherwise} \end{cases} \qquad \sum_{i=0}^{k-1} \lambda_i = iop(k)$$

In the $\mathcal{IO}$-block schedule $\underrightarrow{C}$ machine $\sigma^j = p$ executes $l = j - iop(k)$ steps from $C^{iop(k)}$ until the unsafe step j, hence we set $\lambda^k = l+1$. By the existence theorem we have $\hat{C}^k = C^{iop(k)}$. By Lemmata 3.1 and 3.2, the remaining l steps are safe and with $constr_\delta(p)$ as well as Lemma 4 we deduce for $C' = \Delta_B(\hat{C}^k, p, l, e\hat{x}t^k)$:

$$C' \sim_p C^j \qquad C' \overset{o}{\sim} C^j$$

As also the inputs are consistent by construction, $constr_{safe}(p)$ yields that the next step is $/safe(C', p, \emptyset)$. Consequently we set $\hat{n} = k+1$ and get:

$$/safety_B(\underrightarrow{\hat{C}}, k+1) \qquad \sum_{i=0}^{\hat{n}-1} \lambda_i = iop(k) + l + 1 = j + 1 \leq n$$

All other claims follow analoguously to the existence proof and from the construction of the block trace. This completes our proof. □

5.3 Property Transfer

To demonstrate the usefulness of the block simulation theorem we again consider its application in a property transfer scenario. In particular we want to focus on safety as well as local functional properties $P_f : \mathbb{C} \times \mathbb{N}_{np} \times \mathbb{C} \to \mathbb{B}$ and global invariants on shared and visible state $P_g : \mathbb{C} \to \mathbb{B}$. The functional properties shall specify transitions on local state, e.g., the effect of one complete $\mathcal{IO}$ block of a machine p. In contrast the global invariants should hold at all times and be only dependent on shared memory and visible system components.

$$\forall C, C'.\, P_g(C) \wedge C \overset{sv}{\sim} C' \Longrightarrow P_g(C')$$

By the predicate $verified_B$ we denote to have verified safety and above properties for all blocks of block machine computations going out of start configuration C_0 with a total number of at most n machine steps.

$$\begin{array}{l}
verified_B(C_0, n, P_f, P_g) \equiv \\
\quad \forall \underrightarrow{\hat{C}}, \underrightarrow{\kappa}, \underrightarrow{\lambda}, \underrightarrow{e\hat{x}t}, \hat{n} \quad . \quad \bigwedge \left\{ \begin{array}{ll} C_0 \longrightarrow^{\hat{n}}_{\Delta_B, \kappa, \lambda, e\hat{x}t} \hat{C}^{\hat{n}}, & \sum_{i=0}^{\hat{n}-1} \lambda^i \leq n, \\ \mathcal{IO}sched_B(\underrightarrow{\hat{C}}, \hat{n}), & \end{array} \right\} \\
\qquad\qquad \Downarrow \\
\qquad\quad \forall i \leq \hat{n} \quad . \quad \bigwedge \left\{ \begin{array}{ll} safety_B(\underrightarrow{\hat{C}}, \hat{n}), & P_g(\hat{C}^i), \\ P_f(\hat{C}^{last(\kappa^i, i)}, \kappa^i, \hat{C}^i) & \end{array} \right\}
\end{array}$$

Here $last(\kappa^i, i)$ is an abbreviation for the function $last(\underrightarrow{\hat{C}}, \underrightarrow{\kappa}, \underrightarrow{e\hat{x}t}, \kappa^i, i)$ which searches the last $\mathcal{IO}$ step of machine κ^i before step i and is defined as follows.

$$last(\underrightarrow{\hat{C}}, \underrightarrow{\kappa}, \underrightarrow{e\hat{x}t}, p, i) \equiv \begin{cases} 0 & : \quad i = 0 \\ i - 1 & : \quad i > 0 \wedge \kappa^{i-1} = p \wedge \\ & \qquad \bigvee \left\{ \begin{array}{l} \mathcal{IO}_p(\hat{C}^{i-1}), \\ \forall j < i - 1.\, \kappa^j \neq p \end{array} \right\} \\ last(p, i-1) & : \quad \text{otherwise} \end{cases}$$

We demand that the functional properties hold for all finished $\mathcal{IO}$ blocks. Now we would like to conclude from the verification of all block traces with a total machine step number of n, that also all $\mathcal{IO}$-block schedules of this length are safe and fulfill the verified properties. We formalize this claim by

$$\begin{array}{l}
\forall \underrightarrow{C}, \underrightarrow{\sigma}, \underrightarrow{ext}, n. \\
\bigwedge \left\{ \begin{array}{l} verified_B(C^0, n, P_f, P_g), \\ C^0 \longrightarrow^n_{\Delta, \sigma, ext} C^n, \\ \mathcal{IO}sched(\underrightarrow{C}, n) \end{array} \right\} \Longrightarrow \bigwedge \left\{ \begin{array}{l} safety(\underrightarrow{C}, n), \\ \forall i \leq iop(nio(n) - 1).\, P_g(C^i) \wedge \\ \quad (\mathcal{IO}_{\sigma^i}(C^i) \Longrightarrow \\ \qquad\quad P_f(C^{last(\sigma^i, i)}, \sigma^i, C^i)) \end{array} \right\}
\end{array}$$

where $last(\sigma^i, i)$ is an abbreviation for $last(\underrightarrow{C}, \underrightarrow{\sigma}, \underrightarrow{ext}, \sigma^i, i)$.

PROOF: The safety property is proven by assuming the contrary in order to show a contradiction. If there existed an unsafe computation $\underrightarrow{C}, \underrightarrow{\sigma}, \underrightarrow{ext}$ of length n although we have verified all block computations containing up to that number of individual machine steps, by the soundness property of out block simulation theorem we know that there exists a corresponding block machine computation $\underrightarrow{\tilde{C}}$ of length $\tilde{n} \leq nio(n)$ that is also unsafe. Additionally $\sum_{i=0}^{\tilde{n}-1} \lambda_i \leq n$ holds and we have verified that all block machine computations are safe which are starting in C^0 and are taking up to $\hat{n}$ steps where $\sum_{i=0}^{\hat{n}-1} \lambda^i \leq n$. Hence we see that traces of length $\tilde{n}$ are included in this set of safe traces and there cannot exist an unsafe block machine computation $\underrightarrow{\tilde{C}}$. Consequently all $\mathcal{IO}$ schedules of length n must be safe.

Then we can apply the existence part of the block simulation theorem yielding for every n-step $\mathcal{IO}$-block schedule an equivalent block machine execution $\underrightarrow{\hat{C}}, \underrightarrow{\kappa}, \underrightarrow{\lambda}, \underrightarrow{e\hat{x}t}$ of length $nio(n)$ where we have corresponding configurations $\hat{C}^i = C^{iop(i)}$ at all $\mathcal{IO}$ and first steps of machines in the system with $i < nio(n)$. By the condition on the sum of the step numbers in the existence theorem we see that the first $nio(n)$ blocks in verified block machine computations contain $iop(nio(n)-1)$ small steps including all $\mathcal{IO}$ steps of the n-step $\mathcal{IO}$-block schedule. In the theorem we claim to transfer the verified properties exactly for this range of steps.

$$\forall i < nio(n).\, P_g(C^{iop(i)})$$

The global properties directly hold for all the corresponding $\mathcal{IO}$ and first steps. For all local steps before the last $\mathcal{IO}$ point they are preserved by Lemma 4 and the definition of $\approx_{\overline{p}}$ implying that shared and visible contents are not modified by non-$\mathcal{IO}$ steps. For the functional properties we see that

$$\forall i < nio(n), j \in [0, i).\, j = last(\kappa^i, i) \implies iop(j) = last(\sigma^{iop(i)}, iop(i))$$

because the number and order of $\mathcal{IO}$ points is preserved by the block machine simulation as well as the scheduling of machines. All these corresponding configurations are identical by the simulation theorem and by the precondition $C^0 = \hat{C}^0$. Thus from $\forall i < nio(n).\, P_f(\hat{C}^{last(\kappa^i,i)}, \kappa^i, \hat{C}^i)$ we can deduce:

$$\forall i < nio(n).\, P_f(C^{last(\sigma^{iop(i)},iop(i))}, \sigma^{iop(i)}, C^{iop(i)})$$

Moreover the definitions of $iop(i)$ and $nio(n)$ are consistent, that means that any $\mathcal{IO}$ point up to $nio(n) - 1$ must be one of the first $nio(n)$ $\mathcal{IO}$ points.

$$\forall j \leq iop(nio(n) - 1).\, \mathcal{IO}_{\sigma^j}(C^j) \implies \exists i < nio(n).\, j = iop(i),$$

Then we also get

$$\forall j \leq iop(nio(n) - 1).\, \mathcal{IO}_{\sigma^j}(C^j) \implies P_f(C^{last(\sigma^j,j)}, \sigma^j, C^j)$$

completing the proof of property transfer. □

6 Simulation Theorems for Concurrent Systems

In computer systems we find several layers of abstractions. For instance we can look on programs on the C or the compiled assembly level or even go down to the level of ISA execution. Between different levels there are simulation theorems justified by e.g. compiler correctness. Such simulation theorems are usually proven locally for an execution trace where no environment steps are interleaved. However it is desirable to have the simulation relation hold also in the context of the concurrent system. Thus we need to be able to transfer local simulation theorems into a system wide simulation proof between two *Cosmos* model instantiations.

6.1 Local Simulation Theorems

In the following we develop a generalized theory of local simulation theorems. We consider the simulation between computations $\underrightarrow{D} : \mathbb{C}_d{}^*$ and $\underrightarrow{E} : \mathbb{C}_e{}^*$ with inputs $\underrightarrow{din}, \underrightarrow{ein} : \mathfrak{I}_p^*$ for every machine p. For simplicity we assume that the two systems have compatible memory and the same input types.

$$\mathfrak{A}^d \supseteq \mathfrak{A}^e \qquad\qquad \mathfrak{V}^d = \mathfrak{V}^e$$

Observe that the address range of $\underrightarrow{D}$ might be larger than that of $\underrightarrow{E}$. This means, that the latter may abstract from certain memory regions in the former. For example a stack region might be abstracted to a stack of local memories when we consider compilation.

To specify a local simulation theorem for machine p we need the following ingredients.

- $sim_p : \mathbb{C}_d \times \mathfrak{P}_p \times \mathbb{C}_e \to \mathbb{B}$ - a simulation relation between the two instantiations of p, depending on
- $\mathfrak{P}_p$ - the set of simulation parameters,
- $\mathcal{CP}_p : \mathfrak{C}_p^e \to \mathbb{B}$ - a predicate identifying consistency points of machine p,
- $valid_p^e : \mathbb{C}_e \to \mathbb{B}$ - a validity predicate, collecting all constraints on the configuration of p in $\underrightarrow{E}$ so that execution and simulation is possible,
- $valid_p^d : \mathbb{C}_d \to \mathbb{B}$ - a similar predicate for p in $\underrightarrow{D}$
- δ_p, η_p - *Cosmos* model transition functions for machine p in $\underrightarrow{D}$, or $\underrightarrow{E}$ respectively

The simulation relation shall hold on all consistency points. We demand that all $\mathcal{IO}$-points are also consistency points.

$$\forall ein \in \mathfrak{I}_p.\, \mathcal{IO}_p(E, p, ein) \Longrightarrow \mathcal{CP}_p(c_p)$$

That means that we only consider such systems where reactions to inputs, e.g., interrupts occur at consistency points exclusively. Thus there is some additional reordering necessary to instantiate this theorem for proving simulation on systems where this condition is generally not fulfilled. We define local semantics Δ_p for configurations $D \in \mathbb{C}_d$, $E \in \mathbb{C}_e$, and inputs $din, ein \subseteq \mathfrak{I}_p$.

$$\Delta_p(D, din) = D\lceil \delta_p(D, p, din)\rceil_p \qquad\qquad \Delta_p(E, ein) = E\lceil \eta_p(E, p, ein)\rceil_p$$

Similarly we need a local definition of safety:

$$safety_p(\underrightarrow{D}, \underrightarrow{din}, n) \;\equiv\; inv(D^0) \Longrightarrow \forall i < n.\, \mathcal{R}, \mathfrak{A}, np, p, din^i \vdash D^i \surd_p$$

In order to be able to integrate the simulation proof into the concurrent system later on all $\mathcal{IO}$-points must be preserved and there may not be extra $\mathcal{IO}$-points within the simulation steps according to step numbers s and t.

$$\frac{\begin{array}{c}\mathcal{IO}_p(D^s) \Longrightarrow \mathcal{IO}_p(E^t) \qquad \forall j \in (0,s).\quad /\mathcal{IO}_p(D^j) \\ \forall j \in (0,t).\quad /\mathcal{CP}_p(c_p(E^j))\end{array}}{\mathcal{IO}atCP_p(\underrightarrow{D}, \underrightarrow{din}, \underrightarrow{E}, \underrightarrow{ein}, s, t)}$$

For clarity we abbreviate $safety_p(\underrightarrow{D}, \underrightarrow{din}, n)$ by $safety_p(\underrightarrow{D}, n)$ and we use the shorthand $\mathcal{IO}atCP_p(\underrightarrow{D}, \underrightarrow{E}, s, t)$ in place of $\mathcal{IO}atCP_p(\underrightarrow{D}, \underrightarrow{din}, \underrightarrow{E}, \underrightarrow{ein}, s, t)$. Also here $\mathcal{IO}_p(D^i) \equiv \mathcal{IO}(D^i, p, din^i)$ for all i and local computations $\underrightarrow{D}, \underrightarrow{din}$. The generalized local simulation theorem is then formulated as follows.

Theorem 3 *(Generalized Local Simulation Theorem) For every two consistent start configurations D_0 and E_0 which are valid for machine p and any input sequence $\underrightarrow{din}$ for D we can find an input sequence $\underrightarrow{ein}$ for E, non-trivial step numbers s, t and a simulation parameter par such that the corresponding execution traces $\underrightarrow{D}$, $\underrightarrow{E}$ of p out of D_0 and E_0 preserve the simulation relation of p at all consistency points. There we have consistent inputs. Moreover all reached configurations are valid for p and $\mathcal{IO}$-points do only occur at consistency points. Finally the safety of $\underrightarrow{E}$ implies the safety of the simulated trace.*

$$\forall \begin{array}{l} D_0 \in \mathbb{C}_d, \\ E_0 \in \mathbb{C}_e, \\ par_0 \in \mathfrak{P}_p, \\ \underrightarrow{din} : 2^{\mathfrak{I}_p}{}^* \end{array} . \bigwedge \left\{ \begin{array}{ll} valid_p^d(D_0), & valid_p^e(E_0), \\ \mathcal{CP}_p(c_p(E_0)), & sim_p(D_0, par_0, E_0) \end{array} \right\}$$

$$\Downarrow$$

$$\exists \begin{array}{l} s, t \in \mathbb{N}, \\ \underrightarrow{D} : \mathbb{C}_d{}^*, \\ \underrightarrow{E} : \mathbb{C}_e{}^*, \\ \underrightarrow{ein} : 2^{\mathfrak{I}_p}{}^*, \\ par \in \mathfrak{P}_p \end{array} . \bigwedge \left\{ \begin{array}{l} D_0 \longrightarrow^s_{\Delta_p, din} D^s, \quad E_0 \longrightarrow^t_{\Delta_p, ein} E^t, \\ \forall j \leq s.\ valid_p^d(D^j), \quad \forall j \leq t.\ valid_p^e(E^j), \\ sim_p(D^s, par, E^t), \quad din^0 = ein^0, \\ \mathcal{CP}_p(c_p(E^t)), \quad \mathcal{IO}atCP_p(\underrightarrow{D}, \underrightarrow{E}, s, t), \\ s > 0, \quad \mathcal{IO}_p(D_0) \Longrightarrow t > 0, \\ safety_p(\underrightarrow{E}, t) \Longrightarrow safety_p(\underrightarrow{D}, s) \end{array} \right\}$$

Note that for the abstracted computation $\underrightarrow{E}$ we only demand progress in case of $\mathcal{IO}$ steps. In contrast we only consider such computations $\underrightarrow{D}$ that are progressing in every step.

6.2 *Cosmos* Model Simulation Theorem

Now if we assume that for all machines in a concurrent system we have proven local simulation theorems as stated above, we want to combine them to obtain a system-wide simulation for all interleaved execution traces. For this simulation we consider two block machine computations $\underrightarrow{D}, \underrightarrow{\kappa}, \underrightarrow{\lambda}, \underrightarrow{ext}$ and $\underrightarrow{E}, \underrightarrow{\sigma}, \underrightarrow{\tau}, \underrightarrow{ext}$. Note that the simulation uses the same external input sequence. First we define the simulation relation between two configurations $D \in \mathbb{C}_d$ and $E \in \mathbb{C}_e$ which are at the start of a block of machine p.

$$sim(D, par, p, E) \equiv sim_p(D, \epsilon(par \cap \mathfrak{P}_p), E)$$

Here $par \in \mathfrak{P}$ is the union of all local simulation parameters where for each machine there is at most one parameter given.

$$\mathfrak{P} = \bigcup_{p \in \mathbb{N}_{np}} \mathfrak{P}_p \quad \text{s.t. } \forall par \in \mathfrak{P}, p \in \mathbb{N}_{np}.\ \#(par \cap \mathfrak{P}_p) \leq 1$$

Thus at the beginning of each block we demand the local simulation relation for that particular proof to hold. In a concurrent system all participants should agree on the set of shared resources. Moreover there may be an abstraction of these resources between the two simulation layers we are considering and an access policy for the shared resources. To capture this notion we introduce a shared simulation relation *glob* on the shared and visible components, as well as the ownership configuration. The global properties covered by this relation must be maintained by all local simulation relations and must be defined individually for a given system.

$$glob : \mathbb{C}_d \times \mathbb{C}_e \rightarrow \mathbb{B}$$

For a successful integration of local simulation theorems we need several further constraints on the predicates and the simulation relation introduced above.

Assumption 1 *The shared simulation relation glob may only depend on the shared memory, visible components and ownership configuration.*

$$\forall D, D' \in \mathbb{C}_d, E, E' \in \mathbb{C}_e.\ \bigwedge \left\{ \begin{array}{ll} glob(D, E), & \\ D \overset{sv}{\approx} D', & E \overset{sv}{\approx} E', \\ D \overset{o}{\approx} D', & E \overset{o}{\approx} E' \end{array} \right\} \Longrightarrow glob(D', E')$$

Thus *glob* encodes the relation between the shared memories, the visible components and ownership in both systems. Ideally shared memories are identical or there is only a subset of the addresses visible in the abstract simulation layer, however if two addresses are present in both systems their memory contents should be consistent. The shared simulation relation is a meant to encapsulate these system-specific properties in an instantiable entity. Moreover here we can state specific ownership system invariants, e.g., restrict the ownership on private memory regions or specify lock-protected data structures.

Assumption 2 *We assume that the consistency statement implies the relation on shared memory, visible components and ownership between D and E.*

$$\forall p \in \mathbb{N}_{np}, D \in \mathbb{C}_d, E \in \mathbb{C}_e, par \in \mathfrak{P}_p. \quad sim_p(D, par, p, E) \implies glob(D, E)$$

Note that we do not demand that the owned sets and shared memory is equal. As $\underset{\rightarrow}{E}$ might be an abstraction of $\underset{\rightarrow}{D}$ where we hide certain portions of memory it is enough if the consistency only depends on local parts of p and consistent shared components.

Assumption 3 *The simulation relation for machine p only depends on p's local state and the shared simulation relation.*

$$\forall D, D' \in \mathbb{C}_d, E, E' \in \mathbb{C}_e, par \in \mathfrak{P}_p.$$
$$\bigwedge \left\{ \begin{array}{lll} sim_p(D, par, E), & & \\ valid_p^d(D), & valid_p^e(E), & \\ D \approx_p D', & E \approx_p E', & glob(D', E') \end{array} \right\} \Longrightarrow sim_p(D', par, E')$$

This assumption allows us to maintain simulation during environment steps. Additionally all $\mathcal{IO}$-steps should be atomic wrt. the shared simulation relation and consistent $\mathcal{IO}$ steps should agree on safety.

Assumption 4 *$\mathcal{IO}$ steps of machine p preserve the shared simulation relation. A consistent $\mathcal{IO}$ step is safe in $\underrightarrow{D}$ if it is safe in $\underrightarrow{E}$.*

$$\forall D \in \mathbb{C}_d, E \in \mathbb{C}_e, ext \subseteq \mathfrak{E}, par \in \mathfrak{P}_p . \\ \bigwedge \left\{ \begin{array}{l} sim_p(D, par, E), \\ valid_p^d(D), \\ valid_p^e(E), \\ \mathcal{IO}_p(D),\ \mathcal{IO}_p(E) \end{array} \right\} \Longrightarrow \bigwedge \left\{ \begin{array}{l} glob(\Delta(D, p, ext), \Delta(E, p, ext)), \\ safe_p(E) \Longrightarrow safe_p(D) \end{array} \right\}$$

This means, if we consider for instance C compiler consistency, that an $\mathcal{IO}$ C statement either needs to be compiled into a single $\mathcal{IO}$ instruction, or in case it is implemented by more than one instruction the first one needs to implement the $\mathcal{IO}$ access. If this is not the case, $\mathcal{IO}$ points do not match consistency points and the simulation relation may need to be relaxed.

Assumption 5 *Validity predicates only depend on the local state of their respective machines. For all $par \in \mathfrak{P}_p$:*

$$\begin{array}{lcl} \forall D, D' \in \mathbb{C}_d & . & valid_p^d(D) \wedge D \approx_p D' \implies valid_p^d(D') \\ \forall E, E' \in \mathbb{C}_e & . & valid_p^e(E) \wedge E \approx_p E' \implies valid_p^e(E') \end{array}$$

Consequently validity cannot be broken by safe steps of other participants in the system.

Assumption 6 *The safety of execution has to be verified for every machine in the system separately. We encapsulate this idea in the predicate $verified^e(p)$ which is defined below for $p \in \mathbb{N}_{np}$.*

$$\forall E_0 \in \mathbb{C}_e, \underrightarrow{E} : \mathbb{C}_e^*, \underrightarrow{ein} : \mathfrak{I}_p^*, n \in \mathbb{N}. \\ \bigwedge \left\{ \begin{array}{ll} E_0 \longrightarrow^n_{\Delta_p, ein} E^n, & valid_p^e(E_0), \\ \mathcal{CP}_p(E_0), & n = \min\{i > 0 \mid \mathcal{CP}_p(E^i)\} \end{array} \right\} \Longrightarrow safety_p(\underrightarrow{E}, \underrightarrow{ein}, n)$$

This is a crucial prerequesite to enable a safe composition of computations. From a valid consistency point a stand-alone computation of p into the next consistency point must be safe. As the validity predicate only depends on the local state no assumptions on the shared unowned memory can be made besides the properties guaranteed by *glob* and the safety has to be proven for all possibilities.

Assuming all local simulation theorems to be proven for the simulation relations and validity predicates constrained as presented above and all machines to be verified, we can now show the desired global simulation theorem.

Theorem 4 (Cosmos *Model Simulation Theorem) For all consistent and valid* Cosmos *model start configurations D^0 and E_0 which fulfil ownership invariants and any computation $\underrightarrow{D}$ going out of D^0 under block schedule $\underrightarrow{\kappa}$, $\underrightarrow{\lambda}$ and external input sequence $\underrightarrow{ext}$, we can find an equivalent block schedule $\underrightarrow{\kappa}$, $\underrightarrow{\nu}$ as well as a sequence of simulation parameters $\underrightarrow{par}$, s.t. we have a simulation between $\underrightarrow{D}$ and the resulting* *computation $\underrightarrow{E}$ out of E_0 at the beginning of all blocks. The validity properties still hold after every simulation step and $\underrightarrow{D}$ is safe wrt. ownership if we have verified all machines of E_0 separately.*

$$\forall \begin{array}{l} \underrightarrow{D} : \mathbb{C}_d^*, \underrightarrow{\kappa} : \mathbb{N}_{np}^*, \\ \underrightarrow{\lambda} : \mathbb{N}^*, \underrightarrow{ext} : 2^{\mathfrak{E}^*}, \\ E_0 \in \mathbb{C}_e, n \in \mathbb{N}, \\ par_0 \in \mathfrak{P} \end{array} . \bigwedge \left\{ \begin{array}{l} D^0 \longrightarrow^n_{\Delta_B,\kappa,\lambda,ext} D^n, \quad inv(D^0), \quad inv(E_0), \\ \mathcal{IO}sched_B(\underrightarrow{D}, n), \\ \forall p \in \mathbb{N}_{np}. \bigwedge \left\{ \begin{array}{l} valid_p^d(D^0), \quad valid_p^e(E_0), \\ sim(D^0, par_0, p, E_0), \\ verified^e(p) \end{array} \right\} \end{array} \right\}$$

$$\Downarrow$$

$$\exists \begin{array}{l} \underrightarrow{E} : \mathbb{C}_e^*, \\ \underrightarrow{\nu} : \mathbb{N}^*, \\ \underrightarrow{par} : \mathfrak{P}^* \end{array} . \bigwedge \left\{ \begin{array}{l} E_0 \longrightarrow^n_{\Delta_B,\kappa,\nu,ext} E^n, \quad par^0 = par_0, \\ \forall i < n. \bigwedge \left\{ \begin{array}{l} valid_{\kappa^i}^d(D^i), \quad valid_{\kappa^i}^e(E^i), \\ sim(D^i, par^i, \kappa^i, E^i) \end{array} \right\}, \\ \mathcal{IO}sched_B(\underrightarrow{E}, n), \quad safety_B(\underrightarrow{D}, n) \end{array} \right\}$$

Observe that from the safety of all block schedules for $\mathbb{C}_d$ we can deduce the $verified$ property for all p in any new simulation where $\mathbb{C}_d$ is on top. This is because wherever the consistency points lay for this new simulation, they must be at least at the $\mathcal{IO}$ points. As we prove the safety for any $\mathcal{IO}$-block schedule of $\mathbb{C}_d$ we also cover all traces from any consistency point to the next one.

PROOF: By induction on n. For $n = 0$ we set $E^0 = E_0$ and $par^0 = par_0$ to conclude the first line of the claim. Validity and simulation follows directly from the precondition. All other claims are trivial for $n = 0$.
As the induction hypothesis we assume the claim to hold for an arbitrary but fixed n. Thus for the n-step prefix of $\underrightarrow{D}$ a computation $\underrightarrow{\hat{E}}, \underrightarrow{\kappa}, \underrightarrow{\hat{\nu}}, \underrightarrow{ext}$ exists such that all desired properties already hold until $\hat{E}^n$ and $\underrightarrow{D}$ is safe until configuration D^n. Taking the induction step $n \to n+1$ we construct a new trace $\underrightarrow{E}, \underrightarrow{\kappa}, \underrightarrow{\nu}, \underrightarrow{ext}$ by extending $\underrightarrow{\hat{E}}$ with a block step of machine $\kappa^n = p$. Now let the last block executed by p before step n be scheduled at step j.

$$j = \max\{i \mid i < n \wedge \kappa^i = p\}$$

In case no such j exists than p is scheduled for the first time and we see that $\underrightarrow{\nu}[0:n) = \underrightarrow{\hat{\nu}}[0:n)$, thus $\underrightarrow{E}[0:n] = \underrightarrow{\hat{E}}[0:n]$. By Lemma 5.1 we have

$$D^n \approx_p D^0 \qquad E^n \approx_p E^0$$

and from the precondition $sim_p(D^0, par_0, E^0)$ as well as $valid_p^d(D^0)$ and $valid_p^e(E^0)$. Therefore with Assumption 5 we obtain

$$valid_p^d(D^n) \qquad valid_p^e(E^n)$$

Let the previous block from D^{n-1} to D^n or E^{n-1} to E^n be executed by machine $\kappa^{n-1} \overset{\text{IH}}{=} \kappa^{n-1} = p'$. By induction hypothesis we have

$$sim_{p'}(D^{n-1}, par', E^{n-1})$$

with $par' = \epsilon(par^{n-1} \cap \mathfrak{P}_{p'})$. Moreover we know from the $\mathcal{IO}sched$ property that p in both systems is about to execute an $\mathcal{IO}$ step and that both configurations are valid wrt. machine p. Therefore we have by Assumption 4:

$$glob(D', E') \qquad D' = \Delta(D^{n-1}, p', ext^{n-1}) \quad E' = \Delta(E^{n-1}, p', ext^{n-1}))$$

Since we know from IH and Assumption 6 that $\underrightarrow{D}$ and $\underrightarrow{E}$ are safe until configuration n the following $\lambda^{n-1}-1$ steps, or $\nu^{n-1}-1$ steps resp., are safe as well. Thus we can apply Lemma 4 obtaining

$$D^n \approx_{\overline{p'}} D' \qquad E^n \approx_{\overline{p'}} E'$$

By the definition of $\approx_{\overline{p'}}$ as well as Assumptions 1 and 2 we then know that the shared and visible components between these configurations are consistent. Moreover safe local steps do not alter the ownership configuration.

$$glob(D',E') \wedge \bigwedge \left\{ \begin{array}{ll} D^n \overset{sv}{\sim} D', & D^n \overset{o}{\sim} D', \\ E^n \overset{sv}{\sim} E', & E^n \overset{o}{\sim} E' \end{array} \right\} \Longrightarrow glob(D^n, E^n)$$

Thus by Assumption 3 and with $par^n = par_0$ we conclude:

$$sim(D^n, par^n, p, E^n)$$

Since $\kappa^n = p$ was not scheduled before and by induction hypothesis, $\underrightarrow{E}$ is a valid $\mathcal{IO}$-block machine computation. We set $\lambda^n = s$ and $\nu^n = t$ to the step numbers we obtain from the local simulation theorem. The safety of the new block in $\underrightarrow{D}$ then follows directly from the safety transfer property and $verified(p)^e$. Combination with IH yields:

$$safety_B(\underrightarrow{D}, n+1)$$

If p was scheduled before, i.e., j is well-defined, we know by induction hypothesis that $sim_p(D^j, par^j|_p, E^j)$ holds. In order to apply the local simulation theorem for p we need to construct an input sequence $\underrightarrow{din}$ for the steps in the block under consideration. We assume that machine p performs $l = \lambda^j > 0$ steps in this block and set up the sequence of length $l+1$ as follows.

$$din^i = \begin{cases} in_p(D^j) & : \quad i = 0 \\ in(\Delta(D^j, p, ext^j), p, \emptyset) & : \quad i \in (0, l) \\ in_p(D^n) & : \quad i = l \end{cases}$$

If we execute l steps of machine p from D^j using these inputs we get a trace $\underrightarrow{\tilde{D}}$, such that

$$\forall i \in [0:l].\; D^j \longrightarrow^i_{\Delta_p, din} \tilde{D}^i$$

and where we have $\forall i \in [0:l].\; \tilde{D}^i = \Delta_B(D^j, p, i, ext^j)$ by $\tilde{D}^0 = D^j$ and the determinism of Δ. Consequently $\tilde{D}^l = D^{j+1}$ and since we have $D^{j+1} \approx_p D^n$ by Lemma 5.1 and machine p is at an $\mathcal{IO}$ point in the latter configuration we know due to $constr_{\mathcal{IO}}(p)$ that $\mathcal{IO}_p(\tilde{D}^l)$ holds, too. From the local simulation theorem we know by induction that there exists step functions s, t, a computation $\underrightarrow{\tilde{E}}$, $\underrightarrow{ein}$, and simulation parameters $\underrightarrow{par_p}$ such that for all $i > 0$:

$$\bigwedge \left\{ \begin{array}{l} \hat{E}^j \longrightarrow^{t(i)}_{\Delta_p, ein} \tilde{E}^{t(i)}, \quad din^{s(i-1)} = ein^{t(i-1)}, \quad par_p^0 = \epsilon(par^j \cap \mathfrak{P}_p), \\ valid_p^d(\tilde{D}^{s(i)}, par_p^i), \quad valid_p^e(\tilde{E}^{t(i)}, par_p^i), \quad sim_p(\tilde{D}^{s(i)}, par_p^i, \tilde{E}^{t(i)}), \\ \mathcal{IO}at\mathcal{CP}_p(pop(\underrightarrow{\tilde{D}}, s(i-1)), pop(\underrightarrow{\tilde{E}}, t(i-1)), s(i)-s(i-1), t(i)-t(i-1)), \\ \mathcal{CP}_p(c_p(\tilde{E}^{t(i)})), \quad \mathcal{IO}_p(\tilde{D}^{s(i-1)}) \Longrightarrow t(i) > 0, \quad s(i) > 0, \\ safety_p(\underrightarrow{\tilde{E}}, t(i)) \Longrightarrow safety_p(\underrightarrow{\tilde{D}}, s(i)), \end{array} \right\}$$

Note that here we know due to IH that the simulation also holds initially for $i = 0$ where $s(0) = t(0) = 0$. Furthermore by $\mathcal{IO}at\mathcal{CP}$ we know that simulation must hold at all $\mathcal{IO}$ points of $\underrightarrow{\tilde{D}}$. Since $\tilde{D}^l$ is such an $\mathcal{IO}$ point and since the simulation always makes progress, we know that there must exist a simulation step number x such that $s(x) = l$ and we set the length of the simulating trace $\tilde{E}$ into that consistency point as $m = t(x)$.

$$sim_p(\tilde{D}^l, par_p^x, \tilde{E}^m)$$

We construct the simulating block machine computation $\underrightarrow{E}, \underrightarrow{\kappa}, \underrightarrow{\nu}, \underrightarrow{ext}$ from the IH trace $\underrightarrow{\hat{E}}, \underrightarrow{\kappa}, \underrightarrow{\hat{\nu}}, \underrightarrow{ext}$ by inserting the completed block of p and appending the new unfinished block, where we only execute the starting $\mathcal{IO}$ step.

$$\nu^i = \begin{cases} \hat{\nu}^i & : \quad i \in [0:j) \cup (j:n) \\ m & : \quad i = j \\ 1 & : \quad j = n \end{cases}$$

As a direct result we get $E^i = \hat{E}^i$ for all $i \in [0:j]$. Also we have:

$$\Delta(E^j, p, ext^j) = \Delta(\hat{E}^j, p, ext^j)$$

With Assumption 6 we know that the both the new block starting in E^j and the old block starting in $\hat{E}^j$ are safe. As $m \geq 1$ due to the progress property of the local simulation theorem and since $\hat{\nu}^j \geq 1$ by the $\mathcal{IO}sched_B$ property we deduce

$$E^{j+1} \approx_{\overline{p}} \Delta(E^j, p, ext^j) = \Delta(\hat{E}^j, p, ext^j) \approx_{\overline{p}} \hat{E}^{j+1}$$

by inductive application of Lemma 4. Applying Lemma 5.2 for the remaining block steps we see that:

$$\forall i \in (j:n].\ E^i \approx_{\overline{p}} \hat{E}^i$$

Since the definition of $\approx_{\overline{p}}$ implies $E^i \approx_{p'} \hat{E}^i$ and $E^i \overset{sv}{\sim} \hat{E}^i$ for all $p' \neq p$ and the ownership state is not affected by local steps of p, i.e., $E^i \overset{o}{\sim} \hat{E}^i$, it follows

$$\forall i < n.\ sim(D^i, par^i, \kappa^i, E^i)$$

by Assumptions 1 and 3 and induction hypothesis. Similarly we deduce the validity of all configurations using Assumption 5. We obtain $glob(D^n, E^n)$ from E^{n-1} in the same manner as above for the case where p was not scheduled before. Lemma 5.1 yields $E^n \approx_p E^{j+1} = \tilde{E}^m$ and $D^n \approx_p D^{j+1} = \tilde{D}^l$. Thus using the consistency between $\tilde{D}^l$ and $\tilde{E}^m$ as well as Assumption 3 we have also $sim(D^n, par^n, \kappa^i, E^n)$ where we set $par^n = par_p^x$.

It remains to be shown that $\underrightarrow{D}$ is safe and that $\underrightarrow{E}$ is a correct $\mathcal{IO}$ schedule block execution. Concerning safety we get the desired property until block step j directly from IH. The safety of the finished block j is transfered from $\underrightarrow{E}$ as shown above via the safety transfer property of the local simulation theorem and $verified(p)^e$. For the steps $i \in (j:n)$ we have already seen that $E^i \approx_{\overline{p}} \hat{E}^i$

which implies $E^i \approx_{\kappa^i} \hat{E}^i$ and $E^i \overset{o}{\sim} \hat{E}^i$. Furthermore by construction these steps have the same inputs thus their safety follows from IH and $constr_{safe}(\kappa^i)$. For the last step by Assumption 4 we know that consistent configurations at $\mathcal{IO}$ points agree on safety. Since by Assumption 6 we have $safe_p(E^n)$ Assumption 4 yields $safe_p(D^n)$ and $safe_B(D^n, p, \lambda^n, ext^n)$ with $\lambda^n = 1$. Therefore the complete trace is safe.

$$safety_B(\underset{\rightarrow}{D}, n+1)$$

As we showed above the computation $\underset{\rightarrow}{E}$ is consistent to $\underset{\rightarrow}{\hat{E}}$ for all blocks except those with indices j and n. Therefore the $\mathcal{IO}sched_B$ property holds for these blocks by IH. For block j we see that only the first step is an $\mathcal{IO}$ operation with the $\mathcal{IO}at\mathcal{CP}$ property and the fact that $\underset{\rightarrow}{D}$ is a correct $\mathcal{IO}$ schedule block computation. For the last block we only schedule the initial $\mathcal{IO}$ step, thus we have for the complete trace:

$$\mathcal{IO}sched_B(\underset{\rightarrow}{E}, n+1)$$

This finishes the overall *Cosmos* model simulation proof. □

Thus we have shown how to lift up local simulation theorems fulfilling Assumptions 1 to 6 forming a system-wide concurrent simulation. If the simulation theorems hold locally, any $\mathcal{IO}$-blockwise interleaving of steps on the implementation level ($\underset{\rightarrow}{D}$) will be consistent to some simulating block trace on the abstract level ($\underset{\rightarrow}{E}$). Also it suffices to show the safety on the abstract level when it can be transfered down, guaranteeing the safety of all implementation block schedules. By the reordering theorem we then know that arbitrarily interleaved executions on this level are safe. Global properties proven on the abstract level hold in all $\mathcal{IO}$ points on the lower level in conjunction with the simulation relation. Whatever can be deduced from this conjuction holds for all implementation executions, since between $\mathcal{IO}$ points there are only safe local steps which do not interfere with shared data.

7 Conclusion

We have presented our approach to facilitate the formal specification and verification of concurrent systems with shared memory. It is based on the instantiable *Cosmos* model model of concurrent state machines and the principle of ownership which defines a policy to enforce memory safety. We have used these safety properties to prove a reduction theorem which enables us to assume coarse scheduling. Machines may thus be assumed to take turns in executing blocks of steps, where each block starts with an $\mathcal{IO}$ step, i.e., a step that implements communication with other machines, e.g., by a shared memory access. In addition for simulation theorems where each $\mathcal{IO}$ point is also a consistency point we have shown how and under which assumptions one can combine local simulation theorems into a system-wide *Cosmos* model simulation theorem that states that the individual simulation relation of a machine holds as soon it reaches another $\mathcal{IO}$ point. Our results lay an important foundation for the development of a semantics stack for system specification. Moreover our reduction theorem generalizes existing ones and can be instantiated to show the soundness of coarse scheduling assumed, e.g. by the verification too VCC.

Further applications of our theorem exist, e.g., in the treatment of interrupts as ordinary concurrent threads. Here the execution of the interrupt handler can be reordered to occur only when the interrupted program is in a consistency point. Moreover we have not yet shown how to instantiate our models with existing machine and programming languages. This is ongoing work and results will be published elsewhere.

We would like to thank Prof. Wolfgang J. Paul and Sabine Schmaltz for fruitful discussions and valuable advice during the development of the theory presented above.

References

[AHL+09] E. Alkassar, M. A. Hillebrand, D. C. Leinenbach, N. W. Schirmer, A. Starostin, and A. Tsyban. Balancing the load: Leveraging semantics stack for systems verification. *Journal of Automated Reasoning: Special Issue on Operating Systems Verification*, 42, Numbers 2-4:389–454, 2009.

[AL91] Martín Abadi and Leslie Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284, 1991.

[AL95] Martín Abadi and Leslie Lamport. Conjoining specifications. *ACM Transactions on Programming Languages and Systems*, 17(3):507–534, 1995.

[Ash75] E. A. Ashcroft. Proving assertions about parallel programs. *Journal of Computer and System Sciences*, 10(1):110 – 135, 1975.

[BBBB09] Christoph Baumann, Bernhard Beckert, Holger Blasum, and Thorsten Bormer. Formal verification of a microkernel used in dependable software systems. In Bettina Buth, Gerd Rabe, and Till Seyfarth, editors, *Computer Safety, Reliability, and Security (SAFECOMP 2009)*, volume 5775 of *Lecture Notes in Computer Science*, pages 187–200, Hamburg, Germany, 2009. Springer.

[BG03] Andreas Blass and Yuri Gurevich. Abstract state machines capture parallel algorithms. *ACM Transactions on Computation Logic*, 4(4):578–651, 2003.

[Bro04] Stephen D. Brookes. A semantics for concurrent separation logic. In *CONCUR 2004 - Concurrency Theory, 15th International Conference, London, UK, August 31 - September 3, 2004, Proceedings*, volume 3170 of *Lecture Notes in Computer Science*, pages 16–34. Springer, 2004.

[CAB+09] E. Cohen, A. Alkassar, V. Boyarinov, M. Dahlweid, U. Degenbaev, M. Hillebrand, B. Langenstein, D. Leinenbach, M. Moskal, S. Obua, W. Paul, H. Pentchev, E. Petrova, T. Santen, N. Schirmer, S. Schmaltz, W. Schulte, A. Shadrin, S. Tobies, A. Tsyban, and S. Tverdyshev. Invariants, modularity, and rights. In Amir Pnueli, Irina Virbitskaite, and Andrei Voronkov, editors, *Perspectives of Systems Informatics (PSI 2009)*, volume 5947 of *Lecture Notes in Computer Science*, pages 43–55. Springer, 2009.

[CDH$^+$09] Ernie Cohen, Markus Dahlweid, Mark Hillebrand, Dirk Leinenbach, Michał Moskal, Thomas Santen, Wolfram Schulte, and Stephan Tobies. VCC: A practical system for verifying concurrent C. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Markus Wenzel, editors, *Theorem Proving in Higher Order Logics (TPHOLs 2009)*, volume 5674 of *Lecture Notes in Computer Science*, pages 23–42, Munich, Germany, 2009. Springer. Invited paper.

[CL98] Ernie Cohen and Leslie Lamport. Reduction in tla. In *CONCUR'98 Concurrency Theory*, volume 1466 of *Lecture Notes in Computer Science*, pages 317–331, 1998.

[CMST09] Ernie Cohen, Michał Moskal, Wolfram Schulte, and Stephan Tobies. A practical verification methodology for concurrent programs. Technical Report MSR-TR-2009-15, Microsoft Research, February 2009. Available from `http://research.microsoft.com/pubs`.

[Doe77] Thomas W. Doeppner Jr. Parallel program correctness through refinement. In *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, Los Angeles, California, USA, January 1977*, pages 155–169, 1977.

[FFQ05] Cormac Flanagan, Stephen N. Freund, and Shaz Qadeer. Exploiting purity for atomicity. *IEEE Transactions on Software Engineering*, 31(4):275–291, 2005.

[FG05] Cormac Flanagan and Patrice Godefroid. Dynamic partial-order reduction for model checking software. In *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2005, Long Beach, California, USA, January 12-14, 2005*, pages 110–121, 2005.

[Flo67] R. W. Floyd. Assigning meaning to programs. In *Proceedings of the Symposium on Applied Maths*, volume 19, pages 19–32. AMS, 1967.

[FQ04] Stephen N. Freund and Shaz Qadeer. Checking concise specifications for multithreaded software. *Journal of Object Technology*, 3(6):81–101, 2004.

[GM82] Joseph A. Goguen and José Meseguer. Security policies and security models. In *IEEE Symposium on Security and Privacy*, pages 11–20, 1982.

[GRS05] Yuri Gurevich, Benjamin Rossman, and Wolfram Schulte. Semantic essence of asml. *Theoretical Computer Science*, 343(3):370–412, 2005.

[Gur00] Yuri Gurevich. Sequential abstract-state machines capture sequential algorithms. *ACM Transactions on Computation Logic*, 1(1):77–111, 2000.

[Gur04] Yuri Gurevich. Abstract state machines: An overview of the project. In *Foundations of Information and Knowledge Systems, Third International Symposium, FoIKS 2004, Wilhelminenberg Castle, Austria, February 17-20, 2004, Proceedings*, pages 6–13, 2004.

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

[Hoa78] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[Kle09] Gerwin Klein. Operating system verification — an overview. *Sādhanā*, 34(1):27–69, February 2009.

[Lam90] Leslie Lamport. A theorem on atomicity in distributed algorithms. *Distributed Computing*, 4:59–68, 1990.

[Lam93] Leslie Lamport. Verification and specifications of concurrent programs. In *A Decade of Concurrency, Reflections and Perspectives, REX School/Symposium*, pages 347–374, 1993.

[Lam94] Leslie Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, 1994.

[Lip75] Richard J. Lipton. Reduction: A method of proving properties of parallel programs. *Commun. ACM*, 18(12):717–721, 1975.

[LS89] Leslie Lamport and Fred B. Schneider. Pretending atomicity. Technical report, SRC Research Report 44, 1989.

[LS09] Dirk Leinenbach and Thomas Santen. Verifying the Microsoft Hyper-V Hypervisor with VCC. In *Formal Methods (FM 2009)*, volume 5850 of *Lecture Notes in Computer Science*, pages 806–809, Eindhoven, the Netherlands, 2009. Springer. Invited paper.

[LT87] Nancy A. Lynch and Mark R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing, Vancouver, British Columbia, Canada*, pages 137–151, 1987.

[Lyn96] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[OG76] Susan S. Owicki and David Gries. An axiomatic proof technique for parallel programs i. *Acta Informatica*, 6:319–340, 1976.

[Rus92] John Rushby. Noninterference, transitivity, and channel-control security policies. Technical report, dec 1992. http://www.csl.sri.com/papers/csl-92-2/.