ELECTRONIC WORKSHOPS IN COMPUTING

Series edited by Professor C.J. van Rijsbergen

**D. J. Duke , University of York, UK and A.S. Evans, University of Bradford,
UK (Eds)**

2nd BCS-FACS Northern Formal Methods Workshop

Proceedings of the 2nd BCS-FACS Northern Formal Methods
Workshop, Ilkley, 14-15 July 1997

Relaxing Property Preservation in the Refinement of Concurrent Systems

M. Siegel

Springer

Published in collaboration with the
British Computer Society

©Copyright in this paper belongs to the author(s)

Relaxing Property Preservation in the Refinement of Concurrent Systems

Michael Siegel

Weizmann Institute of Science

Rehovot, Israel

Email: mis@wisdom.weizmann.ac.il

Abstract

One of the major development strategies for concurrent systems suggests to start the system development from a so-called functional design of the envisaged system and to distribute/parallelize this design in subsequent development steps towards a concurrent system.

In this paper we argue that this strategy is not supported by the standard state-based refinement approaches. This phenomenon is traced back to the fact that these approaches are constructed such that necessarily *all* temporal properties of the refined system are preserved during refinement.

We explain that the key feature of a suitable refinement notion for the above strategy has to relax this strict preservation of properties. Rather than preserving all temporal properties of the refined system the required refinement notion has to support the exclusive preservation of specific properties. We present such a refinement approach and prove that the standard state-based refinement relations are particular instances of the advocated notion.

1 Introduction

Since its introduction by [6, 31] (stepwise) refinement is a prominent approach to the development of systems that provably satisfy their specification. An abstract system which satisfies some requirement specification is stepwise refined into a concrete system such that the specification is preserved in each refinement step.

However, formal frameworks that support stepwise refinement [5] not only have to be based on sound mathematical concepts but also have to support those practical development strategies that are used by actual system developers.

One such strategy for concurrent systems, advocated by [2, 3, 4, 10, 30] and used by many protocol designers (see, e.g., [25]), originates from the observation that the functionality of most large concurrent algorithms can be described by a *sequentialized* variant of the algorithm [4], called its *functional design*. This strategy suggests to start the development of a concurrent system from its functional design and to distribute/parallelize this design in subsequent refinement steps. The functional design is considerably easier to develop and verify than the concurrent system and is thus an adequate starting point for system development. As explained in [28, 27, 30] and briefly recalled below, standard state-based refinement approaches, such as [1, 12, 14, 18, 21], do not support this development pattern.

In this paper we present a parameterized refinement notion between *fair transition systems* [12, 23] which supports the development of concurrent systems from a functional design and we show that Abadi & Lamport refinement [1], trace refinement [12, 21] and partial order refinement [13, 17] are particular instances of the proposed refinement relation. These results allow one to mix refinement steps according to the above development strategy and the usually supported strategy of decreasing non-determinism during system development [27].

Assuming that we are given an abstract system that satisfies some specification, the idea of (stepwise) refinement is the following: Rather than verifying directly that the more concrete system also satisfies the specification, one proves that the concrete system refines the abstract one w.r.t. a refinement relation which preserves the properties, in particular the specification, of the abstract system.

According to this informal explanation we can identify the three essential ingredients of state-based refinement approaches. Below, $\mathbf{TS}$ denotes a set of transition systems and LTL a set of linear temporal logic formulae [23].

1. A binary refinement relation $\text{ref} \subseteq \mathbf{TS} \times \mathbf{TS}$, defined on the basis of the semantics of systems.

2. A satisfaction relation $\models \subseteq \mathbf{TS} \times \mathbf{LTL}$ defining which system satisfies what temporal property.
3. A method to prove that $(C, A) \in \text{ref}$ holds for two given systems. In the field of state-based reasoning commonly simulation techniques are used to prove refinement between systems [1, 7, 8, 12, 14, 18, 20, 22]. Such methods induce a relation $\text{sim} \subseteq \mathbf{TS} \times \mathbf{TS}$ with the two compatibility conditions $\text{sim} \subseteq \text{ref}$ (soundness) and $\text{sim} \supseteq \text{ref}$ (completeness).

Concerning the relation between ref and the satisfaction relation $\models$ one can distinguish two pattern in state-based refinement approaches:

1. The refinement relation is chosen such that *all* properties of the abstract system are preserved during refinement, formally expressed by $(\text{ref}; \models) \subseteq \models$ where ‘;’ denotes relation composition. This is the approach employed in, e.g., [1, 12, 14, 18, 21].
2. A parameterized refinement relation ref_φ is chosen, cf. [30, 28], which only preserves a *specific property* φ , formally expressed by $\text{ref}_\varphi \subseteq \{(C, A) \mid A \models \varphi \text{ implies } C \models \varphi\}$.

Remark: The second pattern also comprises the case where the refinement relation preserves a finite set $\{\varphi_1, \dots, \varphi_n\}$ of properties. The respective refinement relation is defined as intersection of relations ref_{φ_i} , $i \in \{1, \dots, n\}$, chosen according to the second pattern.

When starting a system development from the envisaged functional design, stepwise refinement amounts to increasing the degree of parallelism until a concurrent variant of the functional design is obtained. Due to concurrent execution of tasks, the concurrent variant generates, in general, *more* observable behaviors than the functional design [28]. Hence, the functional design obeys *different* temporal properties than the concurrent variant, such as a stronger temporal ordering of events (see [27] for concrete examples from the field of stabilizing algorithms).

Thus, a refinement setup for the above development strategy must have the capability to preserve only a specific set of *desirable* properties, such as the original specification, while ignoring undesired properties such as the too strong temporal ordering of events in the functional design [30].

We present a refinement setup, elaborated in full detail in [27], that relaxes property preservation during refinement. It allows a combination of refinement steps which preserve *all* temporal properties and refinement steps preserving only specific properties in the development of concurrent systems.

The paper is organized as follows. In the next section we present fair transition systems and a stutter closed fragment of LTL. Section 3 presents the advocated refinement relation and in Section 4 we demonstrate that Abadi & Lamport refinement, trace refinement and partial order refinement are particular instances of the suggested refinement notion. We end with some conclusions and future work in Section 5.

Remark: All omitted technical details and proofs can be found in [27].

2 Preliminaries

In this section we introduce fair transition systems (cf. [12, 19, 23]), linear time temporal logic, as well as some technical notions and conventions.

We assume a countable set Var of typed variables in which each variable is associated with a domain describing the possible values of that variable. A *state* is a type consistent interpretation of a subset $V \subseteq \text{Var}$ of variables. The set of all states interpreting variables in V is denoted by Σ_V .

Definition 1 A fair transition system (fts) $A = (\Sigma, I, T, WF, SF)$ consists of a set Σ of states, a set $I \subseteq \Sigma$ of initial states, a finite set $T = \{t_1, \dots, t_n\}$ of transitions with $t_i \subseteq \Sigma \times \Sigma$ including the idling transition $t_{idle} \stackrel{\text{def}}{=} id_\Sigma$, as well as weak and strong fairness constraints $WF \subseteq T$, $SF \subseteq T$. $\square$

The set of all fair transition systems is denoted by $\mathbf{FTS}$. Fair transition systems are typically denoted by $A, B, C, \dots$ and states of a system by the corresponding lower case letter. To refer to the components of system A we use A as index with the additional convention that V_A refers to the set of *variables* evaluated by states in Σ_A . A transition $t \in T$ is *enabled* in state $s \in \Sigma$ if there exists $s' \in \Sigma$ with $(s, s') \in t$. We also write $s \xrightarrow{t} s'$ for $(s, s') \in t$.

Definition 2 A computation of a system $A = (\Sigma, I, T, WF, SF)$ is an infinite sequence $\sigma = \langle s_0, s_1, s_2, \dots \rangle$ of states $s_i \in \Sigma$, s.t.

1. $s_0 \in I$,
2. for all $i \in \mathbf{N}$ there exists a transition $t \in T$ with $(s_i, s_{i+1}) \in t$,
3. for all $t \in WF$ that are continuously enabled from some point onwards in σ , there are infinitely many $i \in \mathbf{N}$ with $(s_i, s_{i+1}) \in t$,
4. for all $t \in SF$ that are infinitely often enabled in σ , there are infinitely many $i \in \mathbf{N}$ with $(s_i, s_{i+1}) \in t$. $\square$

We use $\llbracket A \rrbracket$ to denote the set of computations generated by fts A .

As specification language we consider a stutter-closed [19] fragment of linear time temporal logic [23], referred to as LTL^- . Formulas are constructed from first-order state predicates and temporal operators $\Box$ (always), $\Diamond$ (eventually), and U (strong until). Temporal formulas are denoted by φ, ψ and state formulas by p . We write V_p , resp. V_φ , for the set of free variables of predicate p , resp. formula φ .

Temporal formulas are interpreted over infinite sequences of states [23]. A fair transition system A satisfies formula $\varphi \in LTL^-$, denoted by $A \models \varphi$, if all its computations satisfy φ .

We use the following notations for sequences of states. Σ^* , resp. Σ^ω , denotes the set of all finite, resp. all finite and infinite, state sequences over some state space Σ . Operator $\circ$ denotes concatenation of sequences. For a sequence σ , $|\sigma|$ returns the number of elements in σ ($|\sigma| \stackrel{\text{def}}{=} \omega$ if σ is infinite), $last(\sigma)$ returns the last element of σ if $1 \leq |\sigma| < \omega$ (otherwise $last(\sigma) \stackrel{\text{def}}{=} \perp$), and $rem(\sigma)$ is defined by $\sigma = rem(\sigma) \circ last(\sigma)$ if $1 \leq |\sigma| < \omega$ (otherwise $rem(\sigma) \stackrel{\text{def}}{=} \perp$). The empty sequence is denoted by $\langle \rangle$. For a sequence $\sigma = \langle s_0, s_1, s_2, \dots \rangle$ and $i \in \mathbf{N}$ we define the prefix of σ up to index i by $\sigma_{\leq i} \stackrel{\text{def}}{=} \langle s_0, s_1, \dots, s_i \rangle$. By σ_i we refer to the $(i+1)$ -st state of σ .

3 The Refinement Relation

Refinement approaches provide a formal basis to conclude correctness of a concrete system from correctness of an abstract system. Rather than verifying directly that the more concrete system also satisfies the specification, one exploits the *similarity* of the abstract and the concrete system concerning the observable behavior that both systems display. The notion of similarity is formally captured by means of a refinement relation. Such a refinement relation states e.g. that every observable behavior of the concrete system has to be a possible observable behavior of the abstract system. The direct proof that the concrete system satisfies the specification is replaced by a proof that the concrete system refines the abstract one w.r.t. a predefined refinement relation.

3.1 Standard Refinement

In the state-based world there are several prominent refinement approaches based on Linear Temporal Logic and (fair) transition systems e.g. [1, 11, 14, 18, 21, 29]. These approaches are defined such that they preserve all temporal properties of the abstract system. This means that besides the desired properties contained in the specification also all other temporal properties of the abstract system, which are not mentioned in the specification, are preserved [30].

In case of refining a functional design *we are definitely interested in preserving its specification but we are certainly not interested in preserving the strict temporal ordering of events due to sequential composition of tasks*. In general the functional design obeys *other* temporal properties than the concurrent system and consequently a refinement approach which necessarily preserves all temporal properties of the abstract system is not able to express the relation between both systems. Examples illustrating this situation are given in [27]. We use a parameterized refinement approach which supports the preservation of specific properties during refinement instead of necessarily all.

Though standard refinement approaches do not support the desired notion of refinement, they have been successfully applied for more than a decade for refinement of concurrent systems. So, we want the new refinement approach to allow both:

- refinement steps which preserve specific properties in order to refine a functional design into a concurrent system, and

- standard refinement steps which preserve all temporal properties and allow one to refine the concurrent system towards a concurrent implementation.

Such a refinement framework has been elaborated in [27]. The following presentation of this refinement theory is split into two parts:

1. Defining the parameterized refinement relation, and
2. proving that the refinement relations used in [1, 12, 14, 18, 13, 17, 21] are special instances of the suggested refinement relation.

3.2 Parameterized Refinement

The most general relation that we could choose for property preservation has been indicated in the introduction: Suppose φ is the property that we want to preserve, then the most general refinement relation which preserves φ obviously is $\text{ref}_\varphi \stackrel{\text{def}}{=} \{(C, A) \mid A \models \varphi \text{ implies } C \models \varphi\}$ (a similar definition has been investigated in [30]). Indeed, this is the largest relation that preserves φ ; however we would not succeed in finding a proof technique which avoids proving *directly* that $C \models \varphi$ if $A \models \varphi$ holds since there is not necessarily any similarity between system C and A , besides the fact that both satisfy φ . This means, if we choose ref_φ as refinement relation we would not be able to exploit the knowledge that the abstract system satisfies φ in order to prove that also the concrete system satisfies φ . Consequently we look for a more restrictive relation which allows the preservation of specific properties but nevertheless is provable by standard proof techniques for state-based refinement, namely simulation.

We investigate a generalization of Abadi & Lamport's refinement theory and show that the generalization actually meets the above requirements. First we briefly recall Abadi & Lamport's refinement notion.

3.3 Abadi & Lamport Refinement

In most state-based refinement setups, C *refines* A is defined as inclusion of observable behaviors, i.e. the set of observable behaviors generated by C has to be a subset of the observable behaviors generated by A . Also the refinement relation in [1] is defined as inclusion of observable behavior under the following definition of observable behavior.

In [1] the set of variables of a system is partitioned into a set E of so-called *externally visible* variables and a set of *internal* variables. The observable behavior of a computation is defined relative to an observer that only observes (the values of variables in) set E in the states of the computation. This is captured in the next definition, where operator Π_E denotes restriction of states to the set E of variables. Operator Π_E acts point-wise on sequences of states.

Definition 3 Let σ be a computation of some fair transition system A and $E \subseteq V_A$. The observable behavior of σ w.r.t. E is defined by $\text{beh}_E(\sigma) \stackrel{\text{def}}{=} \Pi_E(\sigma)$.

□

Each definition of observable behavior for computations induces a definition of observable behavior for systems: $\text{beh}_E(A) \stackrel{\text{def}}{=} \{\text{beh}_E(\sigma) \mid \sigma \in \llbracket A \rrbracket\}$. So also for subsequent notions of observable behavior we use $\text{beh}(A)$, rather than $\text{beh}(\llbracket A \rrbracket)$ to denote this lifting to systems.

Refinement in [1] is defined as inclusion of observable behaviors modulo *stuttering*. We say that a sequence of states $\text{seq} = \langle s_0, s_1, s_2, s_3 \dots \rangle$ is *stutter free* if $s_{i-1} \neq s_i$ for all $0 < i < |\text{seq}|$ where $|\text{seq}|$ denotes the length of seq . Operator $\natural$ assigns to each sequence seq its stutter free version, i.e. the stutter free sequence obtained by replacing every maximal subsequence $\langle s_i, s_{i+1} \dots \rangle$ of identical elements in seq by element s_i . Two sequences σ, τ are *stutter equivalent*, denoted by $\sigma \simeq \tau$, if $\natural(\sigma) = \natural(\tau)$. To obtain a simpler proof method, operator $\natural$ removes final stuttering in contrast to the stutter operator in [1]. No matter whether final stuttering is preserved or removed, the induced stutter equivalence $\simeq$ remains unchanged in the Abadi & Lamport setup. The *stutter closure* of a set A of sequences is defined as $\Gamma(A) \stackrel{\text{def}}{=} \{\tau \mid \exists \sigma \in \llbracket A \rrbracket. \sigma \simeq \tau\}$. The refinement relation in [1] can now be defined as follows.

Definition 4 Given systems A, C and set $E \subseteq V_A \cap V_C$. C *refines* A w.r.t. E , denoted by $C \text{ ref}_E A$, if $\Gamma(\text{beh}_E(C)) \subseteq \Gamma(\text{beh}_E(A))$.

□

The usage of the stutter closure Γ in Definition 4 makes relation ref_E invariant under stuttering. The following point-wise characterization of ref_E is of use in Section 4, where we prove that Abadi & Lamport refinement is a special instance of our refinement notion.

Proposition 5 *Given systems A, C and set $E \subseteq V_A \cap V_C$.*

$$C \text{ ref}_E A \text{ iff } \forall \sigma \in \llbracket C \rrbracket. \exists \tau \in \llbracket A \rrbracket. \text{beh}_E(\sigma) \simeq \text{beh}_E(\tau)$$

□

So, system C refines system A w.r.t set E of externally visible variables iff for every computation of C there exists a computation in A which displays, up to stuttering, the same observable behavior. The satisfaction relation $\models \subseteq \text{FTS} \times \text{LTL}^-$ is insensitive to stuttering [19] hence we have that relation ref_E preserves all LTL^- formulas φ with $V_\varphi \subseteq E$.

3.4 Generalizing Abadi & Lamport Refinement

In [1] there is one fixed definition of observable behavior (up to set E) just as in most existing refinement setups. We make this definition a parameter of our refinement relation. We demonstrate that this idea supports the preservation of specific temporal properties during refinement as well as gives the flexibility to capture standard refinement relations.

Technically, the generalization is based on the following observation: The amount of information necessary to decide whether a computation σ satisfies some temporal formula is often *strictly less* than the complete behavior $\text{beh}_E(\sigma)$ of that computation. Consider for example the property $\Box((x = 0) \rightarrow \Diamond(x = 10))$ (informally: it is always the case that a state where x evaluates to 0 is eventually followed by a state where the value of x equals 10). To decide whether a computation satisfies this formula the only relevant information is, whether every state occurring in the computation where $x = 0$ holds is eventually followed by a state where $x = 10$ holds; the values of x in intermediate states where x is different from 0 and 10 are in fact *irrelevant*. So in case that we have a computation satisfying a temporal property there exists a well defined *subsequence* of states witnessing this satisfaction. We generalize relation ref_E by defining that a concrete system refines an abstract system iff: for every computation of the concrete system there exists a computation in the abstract system that displays the same sequence of witnessing states. So also the generalization is defined as inclusion of observable behavior, however with a more liberal notion of observability. Actually, the definition of observability becomes a parameter in our refinement definition since it depends on the property of interest whether a state is a witnessing state or not. So, every property requires a specific notion of observability for its preservation.

We explained that the refinement definition of observable behavior in [1] is based on a notion of observer who observes values of variables in set E in the states of a computation. We use the following more liberal notion of observer.

Definition 6 *An observer $\mathcal{O}$ is characterized by a tuple $\mathcal{O} = (p, E)$ consisting of a state predicate p and a set E of externally visible variables. We define the set of free variables of $\mathcal{O}$ as $V_{\mathcal{O}} \stackrel{\text{def}}{=} V_p \cup E$.*

□

An observer observes a set E of variables but only in those states of a computation that satisfy the so-called *filter predicate* p . The idea is that the filter predicate is used to characterize witnessing states. Using state predicates as observation criterion for states of a computation is a powerful concept; by adding auxiliary variables to the state space of a program (cf. [1]), observability of states may be defined e.g. dependent on occurrences of actions or the computation history.

The observer notion in Definition 6 induces the following adapted definitions of observable behavior and refinement. Here, operator Π_p , for state predicate p , projects a sequence of states to its subsequence of states which satisfy p .

Definition 7

- Let σ be a computation of system A . For observer $\mathcal{O} = (p, E)$ with $V_{\mathcal{O}} \subseteq V_A$ the observable behavior of σ w.r.t. $\mathcal{O}$ is defined by $\text{beh}_{\mathcal{O}}(\sigma) \stackrel{\text{def}}{=} \Pi_E(\Pi_p(\sigma))$.
- Given systems A, C and observer $\mathcal{O} = (p, E)$ with $V_{\mathcal{O}} \subseteq V_A \cap V_C$. C refines A w.r.t. $\mathcal{O}$, denoted by $C \text{ ref}_{\mathcal{O}} A$, iff $\Gamma(\text{beh}_{\mathcal{O}}(C)) \subseteq \Gamma(\text{beh}_{\mathcal{O}}(A))$.

□

Also relation $\text{ref}_{\mathcal{O}}$ is invariant under stuttering due to the use of operator Γ . As in the case of ref_E we give a point-wise characterization of $\text{ref}_{\mathcal{O}}$.

Proposition 8 *Given systems A, C and observer $\mathcal{O}$ with $V_{\mathcal{O}} \subseteq V_A \cap V_C$.*

$$C \text{ ref}_{\mathcal{O}} A \text{ iff } \forall \sigma \in \llbracket C \rrbracket. \exists \tau \in \llbracket A \rrbracket. \text{beh}_{\mathcal{O}}(\sigma) \simeq \text{beh}_{\mathcal{O}}(\tau)$$

□

Proof:

$C \text{ ref}_{\mathcal{O}} A$ **implies** $\forall \sigma \in \llbracket C \rrbracket. \exists \tau \in \llbracket A \rrbracket. \text{beh}_{\mathcal{O}}(\sigma) \simeq \text{beh}_{\mathcal{O}}(\tau)$:

$$\begin{aligned} & C \text{ ref}_{\mathcal{O}} A \\ \text{iff} & \quad \Gamma(\text{beh}_{\mathcal{O}}(C)) \subseteq \Gamma(\text{beh}_{\mathcal{O}}(A)) && \text{Def. 7b)} \\ \text{iff} & \quad \{\delta \in \Sigma_E^\omega \mid \exists \rho \in \text{beh}_{\mathcal{O}}(C). \sharp \delta = \sharp \rho\} \subseteq \\ & \quad \{\delta \in \Sigma_E^\omega \mid \exists \rho \in \text{beh}_{\mathcal{O}}(A). \sharp \delta = \sharp \rho\} && \text{def. of } \Gamma \text{ and } \simeq \\ \text{then} & \quad \{\sharp(\text{beh}_{\mathcal{O}}(\sigma)) \mid \sigma \in \llbracket C \rrbracket\} \subseteq \\ & \quad \{\delta \in \Sigma_E^\omega \mid \exists \rho \in \text{beh}_{\mathcal{O}}(A). \sharp \delta = \sharp \rho\} && \text{set theory} \\ \text{iff} & \quad \{\sharp(\text{beh}_{\mathcal{O}}(\sigma)) \mid \sigma \in \llbracket C \rrbracket\} \subseteq \\ & \quad \{\sharp(\text{beh}_{\mathcal{O}}(\sigma)) \mid \sigma \in \llbracket A \rrbracket\} && \text{def. of } \sharp \\ \text{iff} & \quad \forall \sigma \in \llbracket C \rrbracket. \exists \tau \in \llbracket A \rrbracket. \text{beh}_{\mathcal{O}}(\sigma) \simeq \text{beh}_{\mathcal{O}}(\tau) && \text{def. } \simeq \end{aligned}$$

$\forall \sigma \in \llbracket C \rrbracket. \exists \tau \in \llbracket A \rrbracket. \text{beh}_{\mathcal{O}}(\sigma) \simeq \text{beh}_{\mathcal{O}}(\tau)$ **implies** $C \text{ ref}_{\mathcal{O}} A$:

The fact that Γ is idempotent proves that all steps in the proof above are actually equivalences. We have for all sets A, C of computations that

$$C \subseteq \Gamma(A) \text{ iff } \Gamma(C) \subseteq \Gamma(\Gamma(A)) \text{ iff } \Gamma(C) \subseteq \Gamma(A)$$

by idempotency of Γ . Instantiating $C = \{\sharp(\text{beh}_{\mathcal{O}}(\sigma)) \mid \sigma \in \llbracket C \rrbracket\}$, and $A = \{\delta \in \Sigma_E^\omega \mid \exists \rho \in \text{beh}_{\mathcal{O}}(A). \sharp \delta = \sharp \rho\}$ and observing that

$$\Gamma\{\sharp(\text{beh}_{\mathcal{O}}(\sigma)) \mid \sigma \in \llbracket C \rrbracket\} = \{\delta \in \Sigma_E^\omega \mid \exists \rho \in \text{beh}_{\mathcal{O}}(C). \sharp \delta = \sharp \rho\}$$

we conclude that we can replace the ‘then’ by an ‘iff’ in the proof above.

□

Following [28] we call two computations σ, τ *equivalent w.r.t. observer $\mathcal{O}$* , denoted by $\sigma \stackrel{\mathcal{O}}{=} \tau$, if $\text{beh}_{\mathcal{O}}(\sigma) \simeq \text{beh}_{\mathcal{O}}(\tau)$ holds. From reflexivity and transitivity of the subset relation we obtain the following proposition.

Proposition 9 *For any observer $\mathcal{O} = (p, E)$, $\text{ref}_{\mathcal{O}}$ is a pre-order on the set of systems A with $V_{\mathcal{O}} \subseteq V_A$.*

□

A new aspect in refinement relations that allow one to preserve specific temporal properties is the preservation of an increasing set of properties during program development [30]. The following investigation serves to build a basis for the preservation of an increasing set of properties during successive refinement steps.

Definition 10

- Relation ref preserves property $\varphi \in LTL^-$ if for all systems A, C we have that $(A \models \varphi \text{ and } C \text{ ref } A) \text{ implies } C \models \varphi$.
- Given two observers $\mathcal{O}_1$ and $\mathcal{O}_2$. Observer $\mathcal{O}_2$ is more discriminating than $\mathcal{O}_1$, denoted by $\mathcal{O}_1 \preceq \mathcal{O}_2$, iff $\text{ref}_{\mathcal{O}_2} \subseteq \text{ref}_{\mathcal{O}_1}$.

□

Obviously, $\mathcal{O}_1 \preceq \mathcal{O}_2$ implies that all properties preserved by $\text{ref}_{\mathcal{O}_1}$ are also preserved by $\text{ref}_{\mathcal{O}_2}$. As a consequence from Proposition 9 and Definition 10b) we get the following generalized transitivity.

Composition: If $C \text{ ref}_{\mathcal{O}} B$ and $B \text{ ref}_{\mathcal{O}'} A$ with $\mathcal{O}' \preceq \mathcal{O}$ then $C \text{ ref}_{\mathcal{O}'} A$. $\square$

This composition is an important property for refinement relations that allow the preservation of specific properties during refinement. It justifies the usage of more discriminating observers in subsequent refinement steps in order to preserve an increasing set of properties.

The next proposition gives an explicit characterization when $\mathcal{O}_1 \preceq \mathcal{O}_2$ holds. Here, $s =_V s'$ abbreviates the expression $\Pi_V(s) = \Pi_V(s')$, i.e. $s =_V s'$ holds iff s and s' coincide on the set V of variables.

Proposition 11 Given observers $\mathcal{O}_1 = (p_1, E_1)$ and $\mathcal{O}_2 = (p_2, E_2)$.

$$\begin{array}{l} \models p_1 \rightarrow p_2, \text{ and} \\ E_1 \subseteq E_2, \text{ and} \\ \forall s, s'. (s \models p_1 \wedge s =_{E_2} s' \wedge s' \models p_2) \rightarrow s' \models p_1 \end{array} \quad \text{then } \mathcal{O}_1 \preceq \mathcal{O}_2.$$

$\square$

Proof: First we show that it suffices to prove that the three premises and $beh_{\mathcal{O}_2}(\sigma) \simeq beh_{\mathcal{O}_2}(\tau)$ imply $beh_{\mathcal{O}_1}(\sigma) \simeq beh_{\mathcal{O}_1}(\tau)$ for all finite sequences σ, τ of states; then we prove this claim by induction on the length of σ .

$$\begin{array}{ll} \mathcal{O}_1 \preceq \mathcal{O}_2 & \\ \text{iff } \text{ref}_{\mathcal{O}_2} \subseteq \text{ref}_{\mathcal{O}_1} & \text{Def. 10b} \\ \text{iff } \forall A, C \in \mathcal{S}. C \text{ ref}_{\mathcal{O}_2} A \rightarrow C \text{ ref}_{\mathcal{O}_1} A & \text{set theory} \\ \text{if } \forall V \subseteq \text{Var}. \forall \sigma, \tau \in \Sigma_V^* & \\ \quad beh_{\mathcal{O}_2}(\sigma) \simeq beh_{\mathcal{O}_2}(\tau) \rightarrow beh_{\mathcal{O}_1}(\sigma) \simeq beh_{\mathcal{O}_1}(\tau) & \text{Def. 7b and Prop. 8} \\ \text{iff } \forall V \subseteq \text{Var}. \forall \sigma, \tau \in \Sigma_V^* & \\ \quad beh_{\mathcal{O}_2}(\sigma) \simeq beh_{\mathcal{O}_2}(\tau) \rightarrow beh_{\mathcal{O}_1}(\sigma) \simeq beh_{\mathcal{O}_1}(\tau) & \text{def. of } \simeq \text{ and } beh_{\mathcal{O}} \end{array}$$

So, given two finite sequences σ, τ of states, it remains to prove the following:

$$\begin{array}{l} \text{Ass1: } \models p_1 \rightarrow p_2 \\ \text{Ass2: } E_1 \subseteq E_2 \\ \text{Ass3: } \forall s, s'. (s \models p_1 \wedge s =_{E_2} s' \wedge s' \models p_2) \rightarrow s' \models p_1 \\ \text{Ass4: } beh_{\mathcal{O}_2}(\sigma) \simeq beh_{\mathcal{O}_2}(\tau) \\ \text{Claim1: } beh_{\mathcal{O}_1}(\sigma) \simeq beh_{\mathcal{O}_1}(\tau) \end{array}$$

In order to prove Claim1 we first prove the special case $|\sigma| = 0$, then we make an induction on the length of σ with base case $|\sigma| = 1$.

case: $|\sigma| = 0$

$$\begin{array}{ll} 1. beh_{\mathcal{O}_1}(\sigma) = \langle \rangle & \text{Def. 7a} \\ 2. beh_{\mathcal{O}_2}(\sigma) = \langle \rangle & \text{Def. 7a} \\ 3. beh_{\mathcal{O}_2}(\tau) = \langle \rangle & \text{Ass4 and the definition of } \simeq \\ 4. beh_{\mathcal{O}_1}(\tau) = \langle \rangle & \text{Ass1 and Def. 7a} \\ 5. beh_{\mathcal{O}_1}(\sigma) \simeq beh_{\mathcal{O}_1}(\tau) & 1., 4., \text{ and the definition of } \simeq \end{array}$$

case: $|\sigma| = 1$. Let $\sigma = \langle s \rangle$

sub-case: $s \models p_1$

$$\begin{array}{ll} 1. s \models p_2 & \text{Ass1} \\ 2. \exists i \in \{1, \dots, |\tau|\}. \tau_i \models p_2 \wedge \tau_i =_{E_2} s & \text{Ass4} \\ 3. \forall i \in \{1, \dots, |\tau|\}. \tau_i \models p_2 \rightarrow \tau_i =_{E_2} s & \text{Ass4} \\ 4. \exists i \in \{1, \dots, |\tau|\}. \tau_i \models p_1 \wedge \tau_i =_{E_1} s & \text{Ass2, Ass3 and 2.} \\ 5. \forall i \in \{1, \dots, |\tau|\}. \tau_i \models p_1 \rightarrow \tau_i =_{E_1} s & \text{Ass1, Ass2, and 3.} \\ 6. beh_{\mathcal{O}_1}(\sigma) \simeq beh_{\mathcal{O}_1}(\tau) & 4., 5., \text{ Def. 7a) and def. } \simeq \end{array}$$

sub-case: $s \not\models p_1$

$$\begin{array}{ll} 1. beh_{\mathcal{O}_1}(\sigma) = \langle \rangle & \text{Def. 7a} \\ 2. \forall i \in \{1, \dots, |\tau|\}. \tau_i \not\models p_1 & \text{Ass3, Ass4 and 1.} \\ 3. beh_{\mathcal{O}_1}(\tau) = \langle \rangle & 2. \text{ and Def. 7a} \\ 4. beh_{\mathcal{O}_1}(\sigma) \simeq beh_{\mathcal{O}_1}(\tau) & 1., 3., \text{ and the definition of } \simeq \end{array}$$

Note, that the base case $|\sigma| = 0$ on its own is not sufficient for the following induction step.

case: $|\sigma| = n + 1, n > 0$. Let $\sigma = \sigma' \circ \langle s_n \rangle$

1. $beh_{\mathcal{O}_1}(\sigma) = beh_{\mathcal{O}_1}(\sigma') \circ beh_{\mathcal{O}_1}(\langle s_n \rangle)$ Def. 7a
2. $\exists \tau_1, \tau_2. \tau \simeq \tau_1 \circ \tau_2$
 $\wedge beh_{\mathcal{O}_2}(\sigma') \simeq beh_{\mathcal{O}_2}(\tau_1)$
 $\wedge beh_{\mathcal{O}_2}(\langle s_n \rangle) \simeq beh_{\mathcal{O}_2}(\tau_2)$ Ass4, definition of $\simeq$, and Def. 7a
3. $\exists \tau_1, \tau_2. \tau \simeq \tau_1 \circ \tau_2$
 $\wedge beh_{\mathcal{O}_1}(\sigma') \simeq beh_{\mathcal{O}_1}(\tau_1)$
 $\wedge beh_{\mathcal{O}_1}(\langle s_n \rangle) \simeq beh_{\mathcal{O}_1}(\tau_2)$ 2. and induction hypothesis
4. $beh_{\mathcal{O}_1}(\sigma) \simeq beh_{\mathcal{O}_1}(\tau)$ 3., $\sigma = \sigma' \circ s_n$, Def. 7a, and the definition of $\simeq$

□

The above proposition states that observer $\mathcal{O}_2$ is more discriminating than $\mathcal{O}_1$ if the following holds:

1. $\mathcal{O}_2$ observes at least those states which are observed by $\mathcal{O}_1$.
2. $\mathcal{O}_2$ observes at least those variables which are observed by $\mathcal{O}_1$.
3. If there is a state s' observed by $\mathcal{O}_2$ ($s' \models p_2$) which $\mathcal{O}_2$ cannot distinguish from a state s observed by $\mathcal{O}_1$ (i.e. $s \models p_1 \wedge s =_{E_2} s'$), then s' has to be an observable state of $\mathcal{O}_1$.

The last property states that $\mathcal{O}_1$ may not use variables in set $V_{p_1} \setminus E_2$ to judge observability of states. A simple heuristic to guarantee that the third condition holds is thus to choose E_2 to be a superset of V_{p_1} . See [27] for more details and examples.

As demonstrated in [27] it is convenient to lift $\text{ref}_{\mathcal{O}}$ to *tuples* of observers. In the following section we prove that this lifting allows to capture the notion of partial order refinement – as used in approaches for phased reasoning and design of distributed systems – but not vice versa. The previous results for $\text{ref}_{\mathcal{O}}$ can be lifted to ref_{Ω} .

Definition 12

- a) Let σ be a computation of system A . For a tuple $\Omega = (\mathcal{O}_1, \dots, \mathcal{O}_n)$ of observers with $V_{\mathcal{O}_i} \subseteq V_A$ for $i \in \{1, \dots, n\}$ the observable behavior of σ w.r.t. Ω is defined by $beh_{\Omega}(\sigma) \stackrel{\text{def}}{=} (beh_{\mathcal{O}_1}(\sigma), \dots, beh_{\mathcal{O}_n}(\sigma))$.
- b) Given systems A, C and $\Omega = (\mathcal{O}_1, \dots, \mathcal{O}_n)$ with $V_{\mathcal{O}_i} \subseteq V_A \cap V_C$ for $i \in \{1, \dots, n\}$. C refines A w.r.t. Ω , denoted by $C \text{ ref}_{\Omega} A$, iff $\Gamma(beh_{\Omega}(C)) \subseteq \Gamma(beh_{\Omega}(A))$.

□

In part b) of this definition operator Γ is defined component-wise on tuples. The corresponding element-wise characterization is:

Proposition 13 Given systems A, C and $\Omega = (\mathcal{O}_1, \dots, \mathcal{O}_n)$ with $V_{\mathcal{O}_i} \subseteq V_A \cap V_C$ for $i \in \{1, \dots, n\}$.

$$C \text{ ref}_{\Omega} A \text{ iff } \forall \sigma \in \llbracket C \rrbracket. \exists \tau \in \llbracket A \rrbracket. \forall i \in \{1, \dots, n\}. beh_{\mathcal{O}_i}(\sigma) \simeq beh_{\mathcal{O}_i}(\tau).$$

□

Note, that the order of observers in Ω does not matter, i.e. if $C \text{ ref}_{\Omega} A$ holds and Ω' is a permutation of Ω then also $C \text{ ref}_{\Omega'} A$ holds. Nevertheless, due to Definition 12, Ω is defined as tuple of observers.

Proposition 14 Given $\Omega = \langle \mathcal{O}_1, \dots, \mathcal{O}_n \rangle$ and $\Omega' = \langle \mathcal{O}'_1, \dots, \mathcal{O}'_m \rangle$. If for all $i \in \{1, \dots, n\}$ there exists a $j \in \{1, \dots, m\}$ such that $\mathcal{O}_i \preceq \mathcal{O}'_j$ then $\text{ref}_{\Omega'}$ preserves all properties that are preserved by ref_{Ω} .

□

We get the corresponding composition property for ref_{Ω} .

Composition: If $C \text{ ref}_{\Omega'} B$ and $B \text{ ref}_{\Omega} A$ with $\forall \mathcal{O} \in \Omega. \exists \mathcal{O}' \in \Omega'. \mathcal{O} \preceq \mathcal{O}'$ then $C \text{ ref}_{\Omega} A$.

Proposition 14 and composition of ref_{Ω} taken together state that it is a sound strategy to strengthen the sets of observers used in successive refinement steps by either choosing more discriminating observers or by adding new observers in order to preserve an increasing set of properties during the development of a system.

3.5 Linking Properties and Observers

Due to the generality of the proposed refinement relation the following question arises: How do we prove that a specific temporal property is actually preserved by the chosen refinement relation?

This is an aspect which can be dealt with once and forever in standard refinement approaches such as [1, 11, 21, 29, 14, 18] by proving that the chosen refinement relation preserves *all* temporal properties. Refinement relations that preserve specific properties inevitably have to deal with the additional proof obligation to show that the property of interest is actually preserved.

In [27] we employed the advocated refinement notion for the development of stabilizing systems [26]. Since the crucial part of the behavior of stabilizing systems is defined by so-called *convergence* properties we had to establish a link between convergence properties and observers preserving these properties.

The following theorem states such a link. Here, $p \leadsto q$ (pronounced p leads-to q) is an abbreviation for $\Box(p \rightarrow \Diamond q)$.

Theorem 15 *Given system A, C and $\varphi = p \leadsto q$ for state predicates p, q . If $A \models p \leadsto q$ and $C \text{ ref}_{\mathcal{O}_\varphi} A$ for $\mathcal{O}_\varphi = (p \vee q, V_p \cup V_q)$ then $C \models p \leadsto q$.* □

This theorem states that the notion of observability induced by observer $\mathcal{O}_\varphi$ suffices to conclude from $A \models p \leadsto q$ and $C \text{ ref}_{\mathcal{O}_\varphi} A$ that also $C \models p \leadsto q$ holds. Observer $\mathcal{O}_\varphi$ observes the sequence of states in a computation that witness or disprove the satisfaction of φ . These relevant states in a computation are those satisfying $p \vee q$ and the relevant state information which needs to be preserved during refinement are the values of variables in set $V_p \cup V_q$.

Note, that the induced refinement relation $\text{ref}_{\mathcal{O}_\varphi}$ is not complete in the sense that we do not have for all systems A, C that $A \models p \leadsto q$ and $C \models p \leadsto q$ implies $C \text{ ref}_{\mathcal{O}_\varphi} A$.

Example 16 *Consider two systems A, C where:*

- $\llbracket A \rrbracket = \{ \langle a_0, a_1, a_2, \dots \rangle \}$ with $a_i(x) = 1$ for $i \in \mathbf{N}$, and
- $\llbracket C \rrbracket = \{ \langle c_0, c_1, c_2, \dots \rangle \}$ with $c_i(x) = 2$ for $i \in \mathbf{N}$.

Both systems satisfy the convergence property $\varphi \stackrel{\text{def}}{=} \text{true} \leadsto (x = 1 \vee x = 2)$ but $(C, A) \notin \text{ref}_{\mathcal{O}_\varphi}$.

It is not difficult to see that the reason why C does not refine A w.r.t. $\text{ref}_{\mathcal{O}_\varphi}$ lies in the fact that observed states in C and A have to be identical in order for $\text{ref}_{\mathcal{O}_\varphi}$ to hold. In [28] we suggested observers $\mathcal{O} = (p, E, \equiv)$ where $\equiv \subseteq \Sigma_E \times \Sigma_E$ is an equivalence relation which defines when two observed states are indistinguishable for $\mathcal{O}$. In this example we might have defined $s \equiv s'$ iff $[s \models (x = 1 \vee x = 2) \text{ iff } s' \models (x = 1 \vee x = 2)]$. The resulting observer is easily proven to preserve φ and we also have $C \text{ ref}_{\mathcal{O}} A$. Since we do not need this generalization for subsequent results we refer to [28] for details. In [27] the proof method of *delayed simulation* is elaborated which can be used to actually prove that $C \text{ ref}_{\mathcal{O}} A$ for two given fts and observer $\mathcal{O}$.

A complete characterization of observers for all properties of the temporal hierarchy [23] is work in progress. In the next section we prove that Abadi & Lamport refinement, trace refinement as well as partial order refinement are definable in our setup.

4 Embedding Standard Refinement

4.1 Abadi & Lamport Refinement

In the previous chapter we have given alternative point-wise characterizations of ref_E and $\text{ref}_{\mathcal{O}}$. These characterizations are now used in order to prove that Abadi & Lamport refinement is a particular instance of observer refinement.

Proposition 17 *Given systems A, C and set $E \subseteq V_A \cap V_C$. For all computations $\sigma \in \llbracket C \rrbracket$ and all computations $\tau \in \llbracket A \rrbracket$ we have for $\mathcal{O} \stackrel{\text{def}}{=} (\text{true}, E)$:*

$$\text{beh}_E(\sigma) \simeq \text{beh}_E(\tau) \text{ iff } \text{beh}_{\mathcal{O}}(\sigma) \simeq \text{beh}_{\mathcal{O}}(\tau)$$

□

As a consequence of this proposition we get the desired result, which comes as no surprise since we have defined $\text{ref}_{\mathcal{O}}$ as generalization of relation ref_E .

Theorem 18 Given systems A, C , set $E \subseteq V_A \cap V_C$, and observer $\mathcal{O} \stackrel{\text{def}}{=} (\text{true}, E)$.

$$C \text{ ref}_E A \text{ iff } C \text{ ref}_{\mathcal{O}} A.$$

□

This result is immediate since the observer underlying the definition of ref_E observes *all* states in a computation but inspects only the set E of externally visible variables. So formally this observer can be captured by $\mathcal{O} \stackrel{\text{def}}{=} (\text{true}, E)$ as stated above.

4.2 Trace Refinement

In order to define and characterize trace refinement [12, 13, 21] and also partial order refinement [13, 17] we use a slightly different model, namely fair *named* transition systems [13]. In this model, event names from some set $\mathcal{E}$ are associated with the transitions of a system. We denote the set of events generated by system S by $\mathcal{E}_S$. Whenever a transition is taken the corresponding event is generated. Computations of fair named transition systems are now alternating sequences of states and events. As notational convention we write $s_0 \xrightarrow{e_1} s_1 \xrightarrow{e_2} s_2 \xrightarrow{e_3} \dots$ instead of $\langle s_0, e_1, s_1, e_2, s_2, \dots \rangle$. The e_i record the fact that the occurrence of the transition corresponding to e_i caused the state change from s_{i-1} to s_i . For the definition of trace refinement, the set $\mathcal{E}$ is partitioned into *externally visible* events $\text{Ext} \subseteq \mathcal{E}$ and into *internal* events $\text{Int} \subseteq \mathcal{E}$. This separation in internal and external events induces a corresponding separation of the transitions into external and internal transitions [13].

Extracting the sequence of externally visible events of a computation yields its observable behavior. We use operator Π_{Ext} to project a computation to the sequence of events belonging to set Ext , generally called the *trace* of the computation [22].

Definition 19 Given computation $\sigma = s_0 \xrightarrow{e_1} s_1 \xrightarrow{e_2} s_2 \xrightarrow{e_3} \dots$ of some system A and set $\text{Ext} \subseteq \mathcal{E}_A$ of externally visible events of A . The observable behavior of σ w.r.t. Ext is defined as:

$$\text{beh}_{\text{Ext}}(\sigma) = \Pi_{\text{Ext}}(\sigma).$$

□

Trace refinement can now be defined as:

Definition 20 Given systems A, C with a common set $\text{Ext} \subseteq \mathcal{E}_A \cap \mathcal{E}_C$ of externally visible events. System C refines A w.r.t. Ext , denoted by $C \text{ ref}_{\text{Ext}} A$, is defined as

$$C \text{ ref}_{\text{Ext}} A \text{ iff } \text{beh}_{\text{Ext}}(C) \subseteq \text{beh}_{\text{Ext}}(A).$$

□

In order to define trace refinement in our refinement setup we assume that the state space of the systems under investigation contains a distinct variable *taken* recording the event generated by the occurrence of the transition which caused the last state change. Initially *taken* has a designated initial value. We obtain the following proposition.

Proposition 21 Given systems A, C and set $\text{Ext} \subseteq \mathcal{E}_A \cap \mathcal{E}_C$. For computations $\sigma \in \llbracket C \rrbracket$ and computations $\tau \in \llbracket A \rrbracket$ we have for $\mathcal{O} \stackrel{\text{def}}{=} (\text{taken} \in \text{Ext}, \{\text{taken}\})$:

$$\text{beh}_{\text{Ext}}(\sigma) = \text{beh}_{\text{Ext}}(\tau) \text{ iff } \text{beh}_{\mathcal{O}}(\sigma) = \text{beh}_{\mathcal{O}}(\tau)$$

□

As in case of Abadi & Lamport refinement we have as a consequence of Proposition 21 the following characterization of trace refinement.

Theorem 22 Given systems A, C with a common set $\text{Ext} \subseteq \mathcal{E}_A \cap \mathcal{E}_C$ of externally visible events and observer $\mathcal{O} \stackrel{\text{def}}{=} (\text{taken} \in \text{Ext}, \{\text{taken}\})$. Then the following holds

$$C \text{ ref}_{\text{Ext}} A \text{ iff } C \text{ ref}_{\mathcal{O}} A.$$

□

This theorem states that the observer, underlying the definition of observable behavior in trace refinement, only observes the sequence of externally visible events of a computation which is a well-known fact.

4.3 Partial Order Refinement

The two previous refinement relations ref_E and ref_{Ext} have extensively been studied in the literature. *Partial order refinement* is a comparatively new refinement notion, see e.g. [13, 17], which gained considerable interest in the automated verification community as a means to tackle the state explosion problem in enumerative model-checkers. The characterization of partial order refinement in our setup is far more involved than the characterization of the two previous refinement notions. The reason is the more complicated and rather implicit notion of observability which underlies partial order approaches, as explained next.

As explained in [24], in concurrent systems there is only a *partial ordering* between events of parallel processes, since there is no means to decide which one from two independent events occurs earlier than the other. The only way to establish objective ordering between events is to find their mutual dependencies and to define that a cause must always be earlier than its effect. Therefore an *independence relation* on events is the basis for the description of concurrent behaviors in partial order approaches. These approaches allow elegant and concise descriptions of phenomena occurring in concurrent systems such as serializability and sequential consistency as well as interesting development techniques for such systems [9]. Though the underlying idea is the same in all partial order approaches their presentation differs considerably. In the following we concentrate on the definition and presentation of partial order refinement as given in [13].

The very basis of all partial order approaches that deal with linear computations rather than directly with partial orders, is the definition of a *permutation equivalence* amongst computations. The idea is that two computations are permutation equivalent iff an observer who only observes the relative order of dependent events does not observe any difference between the two computations. An equivalence class of permutation equivalent computations is an abstract representation of a partial order. Considering all computations of an equivalence class as total orderings on events, the intersection of these total orderings yield the partial order on events represented by the equivalence class.

As mentioned above, permutation equivalences are defined relative to a symmetrical, irreflexive independence relation I [24] on a set of external events. Two computations σ, τ are permutation-equivalent w.r.t. I , denoted by $\sigma \approx_I \tau$, if their traces differ only by permutations of independent events [24]. Recall, that $\sigma_{\leq i}$ denotes the prefix of state sequence σ up to and including index i .

Definition 23 Given a set $\mathcal{E}$ of events and independence relation $I \subseteq \mathcal{E} \times \mathcal{E}$.

a) Two finite sequences $\iota, \kappa \in \mathcal{E}^*$ are permutation equivalent w.r.t. I , denoted by $\iota \sim_I \kappa$, if

$$\begin{aligned} \iota &= \langle e_1, e_2, \dots, e_{i-1}, e_i, \dots, e_n \rangle \\ \kappa &= \langle e_1, e_2, \dots, e_i, e_{i-1}, \dots, e_n \rangle \end{aligned}$$

for some $i \in 1, \dots, n$ and $(e_{i-1}, e_i) \in I$. Relation $\sim_I^* \subseteq \mathcal{E}^* \times \mathcal{E}^*$ denotes the reflexive and transitive closure of $\sim_I$.

b) For infinite sequences $\iota, \kappa \in \mathcal{E}^\omega$ we define relation $\preceq_I \subseteq \mathcal{E}^\omega \times \mathcal{E}^\omega$ by

$$\iota \preceq_I \kappa \text{ iff } \forall i \in \mathbf{N}. \exists j \geq i. \exists \rho \in \mathcal{E}^*. \kappa_{\leq j} \sim_I^* \rho \wedge \iota_{\leq i} = \rho_{\leq i}.$$

c) Now, permutation equivalence $\approx_I \subseteq \mathcal{E}^\omega \times \mathcal{E}^\omega$ is defined by

$$\approx_I \stackrel{\text{def}}{=} (\preceq_I \cap (\preceq_I)^{-1}) \cup \sim_I^*.$$

□

The permutation closure of a set A of sequences of events relative to an independence relation I is defined by $\Gamma_I(A) \stackrel{\text{def}}{=} \{\kappa \mid \exists \iota \in A. \iota \approx_I \kappa\}$. For a given sequence ι , $\Gamma_I(\iota)$ is the *run* [16] corresponding to ι . In [15] a run is also referred to as an *interleaving set*.

Definition 24 Given systems A, C , set $Ext \subseteq \mathcal{E}_A \cap \mathcal{E}_C$, and independence relation $I \subseteq Ext \times Ext$. System C refines A w.r.t. Ext, I , denoted by $C \text{ ref}_{Ext}^I A$, is defined by

$$C \text{ ref}_{Ext}^I A \text{ iff } \Gamma_I(\text{beh}_{Ext}(C)) \subseteq \Gamma_I(\text{beh}_{Ext}(A)).$$

□

The interesting point in this definition is that system A may generate *less* traces than C .

Example 25 Consider two systems A, C such that:

- $beh_{Ext}(A) = \{\langle a, b \rangle\}$, and
- $beh_{Ext}(C) = \{\langle a, b \rangle, \langle b, a \rangle\}$.

Let $Ext \stackrel{\text{def}}{=} \{a, b\}$ and $I \stackrel{\text{def}}{=} \{(a, b), (b, a)\}$.

Then we have $C \text{ ref}_{Ext}^I A$ since $\Gamma_I(beh_{Ext}(C)) \subseteq \Gamma_I(beh_{Ext}(A))$.

This can be exploited for phased reasoning about concurrent systems [28]. The point-wise characterization of partial order refinement is stated in the following proposition:

Proposition 26 Given systems A, C , set $Ext \subseteq \mathcal{E}_A \cap \mathcal{E}_C$, and independence relation $I \subseteq Ext \times Ext$.

$$C \text{ ref}_{Ext}^I A \text{ iff } \forall \sigma \in \llbracket C \rrbracket. \exists \tau \in \llbracket A \rrbracket. beh_{Ext}(\sigma) \approx_I beh_{Ext}(\tau)$$

□

An alternative characterization of permutation equivalence, inspired by Mazurkiewicz [24], is the key to formulate partial order refinement in our setup. For this characterization we need the following definition where D denotes the *dependence relation* corresponding to a given independence relation $I \subseteq \mathcal{E} \times \mathcal{E}$, i.e. $D \stackrel{\text{def}}{=} (\mathcal{E} \times \mathcal{E}) \setminus I$.

Definition 27 Given set $\mathcal{E}$ of events and independence relation $I \subseteq \mathcal{E} \times \mathcal{E}$. A collection of non-empty sets $D_i \subseteq \mathcal{E}, i \in \{1, \dots, n\}$, is a *dependence collection* of I iff

1. $\bigcup_{i \in \{1, \dots, n\}} D_i = \mathcal{E}$, and
2. $\bigcup_{i \in \{1, \dots, n\}} D_i \times D_i = D$.

□

Note, that the sets D_i , resp. $D_i \times D_i$, form in general not a partition but a covering of $\mathcal{E}$, resp. D . The following theorem is a generalization of a main result from trace theory [24] to sequences of arbitrary (in particular infinite) length. We use operator Π_E with $E \subseteq \mathcal{E}$ to project sequences $\iota \in \mathcal{E}^\omega$ to the set E of events.

Theorem 28 Given set $\mathcal{E}$ of events and independence relation $I \subseteq \mathcal{E} \times \mathcal{E}$. For every dependence collection $D_i, i \in \{1, \dots, n\}$, of I the following holds for all $\iota, \kappa \in \mathcal{E}^\omega$:

$$\iota \approx_I \kappa \text{ iff } \forall i \in \{1, \dots, n\}. \Pi_{D_i}(\iota) = \Pi_{D_i}(\kappa).$$

□

The proof of this theorem extends the following result from trace theory for finite sequences of events.

Theorem 29 (Mazurkiewicz) Given set $\mathcal{E}$ of events and independence relation $I \subseteq \mathcal{E} \times \mathcal{E}$. For every dependence collection $D_i, i \in \{1, \dots, n\}$, of I the following holds for all $\iota, \kappa \in \mathcal{E}^*$:

$$\iota \sim_I^* \kappa \text{ iff } \forall i \in \{1, \dots, n\}. \Pi_{D_i}(\iota) = \Pi_{D_i}(\kappa).$$

□

Note, that Theorem 29 is based on the simpler notion of permutation equivalence for finite sequence of events, see part a) of Definition 23. Below, we prove that this result can be lifted to infinite sequences of events using the more general notion of permutation equivalence given in part c) of Def. 23. The proof mainly consists of the two following lemmata. We use $\preceq$ to denote the prefix relation on sequences of events.

Lemma 30 Given set $\mathcal{E}$ of events and $\mathcal{E}' \subseteq \mathcal{E}$. For all $\iota, \kappa \in \mathcal{E}^\omega$ the following holds:

$$\Pi_{\mathcal{E}'}(\iota) \preceq \Pi_{\mathcal{E}'}(\kappa) \text{ iff } \forall i \in \mathbb{N}. \exists j \in \mathbb{N}. \Pi_{\mathcal{E}'}(\iota_{\leq i}) = \Pi_{\mathcal{E}'}(\kappa_{\leq j})$$

□

Lemma 31 Given set $\mathcal{E}$ of events and independence relation $I \subseteq \mathcal{E} \times \mathcal{E}$. For every dependence collection $D_i, i \in \{1, \dots, n\}$, of I , the following holds for all $\iota, \kappa \in \mathcal{E}^\omega$:

$$\iota \preceq_I \kappa \text{ iff } \forall i \in \{1, \dots, n\}. \Pi_{D_i}(\iota) \preceq \Pi_{D_i}(\kappa).$$

□

Proof: Assume we are given relation I , some dependence collection $D_i, i \in \{1, \dots, n\}$, of I and $\iota, \kappa \in \mathcal{E}^\omega$.

$$\begin{aligned}
 & \iota \preceq_I \kappa \\
 \text{iff } & \forall i \in \mathbf{N}. \exists j \geq i. \exists \rho \in \mathcal{E}^*. \iota_{\leq i} = \rho_{\leq i} \wedge \rho \sim_I^* \kappa_{\leq j} && \text{by Def. 23b)} \\
 \text{iff } & \forall i \in \mathbf{N}. \exists j \geq i. \exists \rho \in \mathcal{E}^*. \iota_{\leq i} = \rho_{\leq i} \wedge \\
 & \quad (\forall k \in \{1, \dots, n\}. \Pi_{D_k}(\rho) = \Pi_{D_k}(\kappa_{\leq j})) && \text{by Theorem 29} \\
 \text{iff } & \forall i \in \mathbf{N}. \exists j \geq i. \forall k \in \{1, \dots, n\}. \Pi_{D_k}(\iota_{\leq i}) \preceq \Pi_{D_k}(\kappa_{\leq j}) && \text{by } \iota_{\leq i} \preceq \rho \\
 \text{iff } & \forall k \in \{1, \dots, n\}. \Pi_{D_k}(\iota) \preceq \Pi_{D_k}(\kappa) && \text{by Lemma 30}
 \end{aligned}$$

□

Now it is straightforward to complete the proof of Theorem 28.

Proof: We consider the case that ι, κ are infinite; the finite case is covered by Theorem 29.

$$\begin{aligned}
 & \iota \approx_I \kappa \\
 \text{iff } & \iota \preceq_I \kappa \text{ and } \kappa \preceq_I \iota && \text{by Def. 23c)} \\
 \text{iff } & \forall i \in \{1, \dots, n\}. \Pi_{D_i}(\iota) \preceq \Pi_{D_i}(\kappa) \text{ and } \\
 & \quad \forall i \in \{1, \dots, n\}. \Pi_{D_i}(\kappa) \preceq \Pi_{D_i}(\iota) && \text{by Lemma 31} \\
 \text{iff } & \forall i \in \{1, \dots, n\}. \Pi_{D_i}(\iota) = \Pi_{D_i}(\kappa) && \text{by anti-symmetry of } \preceq
 \end{aligned}$$

□

Theorem 28 states that two sequences of events are permutation equivalent iff the relative order of events belonging to the same dependence set D_i is the same.

From the previous section on trace-refinement we known how to define observers that guarantee that two computations coincide when projected on a common set of externally visible events. So, we again assume that the state space of the systems under consideration evaluates a distinct variable *taken* which records the event that caused the last state change. Then we obtain the following characterization of partial order refinement in our setup.

Theorem 32 *Given systems A, C , set $Ext \subseteq \mathcal{E}_A \cap \mathcal{E}_C$, and independence relation $I \subseteq Ext \times Ext$. For each dependence collection $D_i, i \in \{1, \dots, n\}$, of I and corresponding $\Omega \stackrel{\text{def}}{=} (\mathcal{O}_1, \mathcal{O}_2, \dots, \mathcal{O}_n)$ with observers $\mathcal{O}_i \stackrel{\text{def}}{=} (taken \in D_i, \{taken\})$ we have:*

$$C \text{ ref}_{Ext}^I A \text{ iff } C \text{ ref}_\Omega A.$$

□

Proof: The proof of this theorem is based on Theorem 22 and 28. Assume we are given systems A, C , a dependence collection $D_i, i \in \{1, \dots, n\}$, of independence relation I and the corresponding $\Omega \stackrel{\text{def}}{=} (\mathcal{O}_1, \mathcal{O}_2, \dots, \mathcal{O}_n)$.

$$\begin{aligned}
 & C \text{ ref}_{Ext}^I A \\
 \text{iff } & \forall \sigma \in \llbracket C \rrbracket. \exists \tau \in \llbracket A \rrbracket. beh_{Ext}(\sigma) \approx_I beh_{Ext}(\tau) && \text{by Prop. 26} \\
 \text{iff } & \forall \sigma \in \llbracket C \rrbracket. \exists \tau \in \llbracket A \rrbracket. \forall i \in \{1, \dots, n\}. \\
 & \quad \Pi_{D_i}(beh_{Ext}(\sigma)) = \Pi_{D_i}(beh_{Ext}(\tau)) && \text{by Theorem 28} \\
 \text{iff } & \forall \sigma \in \llbracket C \rrbracket. \exists \tau \in \llbracket A \rrbracket. \forall i \in \{1, \dots, n\}. beh_{D_i}(\sigma) \simeq beh_{D_i}(\tau) && \text{def. } beh_{Ext}(\cdot) \\
 & \quad \text{and } D_i \subseteq Ext && \text{by Theorem 22} \\
 \text{iff } & \forall \sigma \in \llbracket C \rrbracket. \exists \tau \in \llbracket A \rrbracket. \forall i \in \{1, \dots, n\}. beh_{\mathcal{O}_i}(\sigma) \simeq beh_{\mathcal{O}_i}(\tau) && \text{by Def. of } \mathcal{O}_i \\
 \text{iff } & \forall \sigma \in \llbracket C \rrbracket. \exists \tau \in \llbracket A \rrbracket. \sigma \stackrel{\mathcal{O}_i}{=} \tau && \text{by Def. of } \mathcal{O}_i \\
 \text{iff } & C \text{ ref}_\Omega A && \text{by Def. 13}
 \end{aligned}$$

□

Summarizing Theorem 18, 22, and 32 we note that Abadi & Lamport refinement, trace refinement, and partial order refinement are definable in our setup. Note, that none of the theorems needs heavy coding but rather an explicit definition of the notion of observer underlying the various refinement approaches.

5 Conclusion

We have presented a refinement notion which allows a combination of refinement steps which preserve specific temporal properties of the refined system and those which preserve all its temporal properties.

The preservation of specific properties is crucial in order to support the development of concurrent systems from a functional design as applied by many protocol designer. The key feature of the advocated refinement relation is its parameterization with so-called observers. This parameterization gives the flexibility to adjust the notion of observability to the properties which are to be preserved in the current refinement step. We furthermore proved that standard state-based refinement notions can be expressed in our framework by an explicit characterization of their underlying notion of observability.

In [27] a formal development framework for stabilizing systems [26] has been elaborated which is based on temporal logic and the advocated refinement relation. This framework comprises besides simulation techniques to prove the parameterized refinement also temporal proof and development rules employing this kind of refinement.

Currently we use a variant of the refinement approach to express and prove refinement between declarative, synchronous SIGNAL-programs and their imperative, asynchronous implementation in C. Furthermore, we investigate further constructions and heuristics how to generate appropriate observers from given sets of properties which are to be preserved during refinement.

Acknowledgment: I thank the anonymous referee for careful reading and a lot of valuable comments.

References

- [1] M. Abadi and L. Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2), 1991.
- [2] R. J. R. Back. Refinement calculus: Parallel and reactive systems. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems*, volume 430 of *LNCS*, 1990.
- [3] K. M. Chandy and J. Misra. *Parallel Program Design: a Foundation*. Addison-Wesley, 1988.
- [4] C. T. Chou and E. Gafni. Understanding and verifying distributed algorithms using stratified decomposition. In *7th ACM Symp. on Principles of Distributed Computing*, 1988.
- [5] J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors. *Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness*, volume 430 of *LNCS*. Springer Verlag, 1990.
- [6] E. W. Dijkstra. *Structured Programming*. Academic Press, 1971.
- [7] K. Engelhardt, W.P. de Roever, et al. *Data Refinement*. Prentice-Hall, 1997. to appear.
- [8] R. Gerth. Foundations of compositional program refinement. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems*, volume 430 of *LNCS*, 1990.
- [9] W. Janssen. *Layered Design of Parallel Systems*. PhD thesis, University of Twente, 1994.
- [10] W. Janssen and J. Zwiers. Protocol design by layered decomposition, a compositional approach. In *Proceedings Formal Techniques in Real Time and Fault Tolerant Systems*, volume 571 of *LNCS*, 1992.
- [11] B. Jonsson. On decomposing and refining specifications of distributed systems. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems*, volume 430 of *LNCS*, 1990.
- [12] B. Jonsson. Simulations between specifications of distributed systems. In J. C. M. Baeten and J. F. Groote, editors, *CONCUR '91*, volume 527 of *LNCS*, 1991.
- [13] B. Jonsson, A. Pnueli, and C. Rump. Proving refinement using transduction. In R. Gerth, editor, *Verifying sequentially consistent memory*. 1993. To be published as a special issue of Distributed Computing.
- [14] M. B. Josephs. A state-based approach to communicating processes. *Distributed Computing*, 3, 1988.
- [15] S. Katz and D. Peled. Interleaving set temporal logic. *Theoretical Computer Science*, 75(3), 1990.
- [16] S. Katz and D. Peled. Verification of distributed programs using representative interleaving sequences. *Distributed Computing*, 6, 1992.

- [17] S. Katz, J. Zwiers, and W. Janssen. Proving sequential consistency by loosening convenient executions to partial orders. In R. Gerth, editor, *Verifying sequentially consistent memory*. 1993. To be published in a special issue of Distributed Computing.
- [18] S. S. Lam and A. U. Shankar. Refinement and projection of relational specifications. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems*, volume 430 of *LNCS*, 1990.
- [19] L. Lamport. The temporal logic of actions. Technical Report 79, DEC, Systems Research Center, December 1991. To appear in Transactions on programming Languages and Systems.
- [20] N. Lynch and M. Tuttle. Hierarchical correctness proofs for distributed algorithms. Proc. of the 6th ACM Symposium on Distributed Computing, 1987.
- [21] N. Lynch and M. Tuttle. An introduction to input/output automata. Technical Report MIT/LCS/TM-373, Massachusetts Institute of Technology, 1988.
- [22] N. Lynch and F. Vaandrager. Forward and backward simulations for timing based systems. In *Real-Time: Theory in Practice*, volume 600 of *LNCS*, 1991.
- [23] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer Verlag, 1991.
- [24] A. Mazurkiewicz. Trace semantics, proceedings of an advanced course. volume 354 of *LNCS*, Bad Honnef, 1989.
- [25] M. Raynal and J.-P. Helary. *Synchronization and control of distributed systems and programs*. Wiley, 1990.
- [26] M. Schneider. Self-stabilization. *ACM Computing Surveys*, 25, 1993.
- [27] M. Siegel. *Phased Design and Verification of Stabilizing Systems*. PhD thesis, University of Kiel, 1996.
- [28] M. Siegel and F. Stomp. Extending the limits of sequentially phased reasoning. In P. S. Thiagarajan, editor, *FST&TCS 14*, volume 880 of *LNCS*, 1994.
- [29] E. W. Stark. Proving entailment between conceptual state specifications. *Theoretical Computer Science*, 56, 1988.
- [30] F. Stomp. Preserving specific properties in program development. In Cleaveland, editor, *CONCUR '92*, volume 630 of *LNCS*, 1992.
- [31] N. Wirth. Program development by stepwise refinement. *Communications of the ACM*, 14, 1971.