Truly Concurrent Logic via In-Between Specification

Harald Fecher[1]

Christian-Albrechts-University Kiel, Germany
hf@ informatik. uni−kiel. de

Abstract

In order to obtain a formalism for the specification of true concurrency in reactive systems, we modify the μ-calculus such that properties that are valid during the execution of an action can be expressed. The interpretation of this logic is based on transition systems that are used to model the ST-semantics. We show that this logic and step equivalence have an incomparable expressive power. Furthermore, we show that the logic characterizes the ST-bisimulation equivalence for finite process algebra expressions that do not contain synchronization mechanisms.

Keywords: specification, true concurrency, μ-calculus, bisimulation, action splitting

1 Introduction

Concurrency is a fundamental feature in reactive systems. There are two ways of handling concurrency: via *interleaving*, where parallel executions are transformed to nondeterministic sequential ones, or via *true concurrency*, where the simultaneity of events is really observable. A typical indication of a truly concurrent observation is that it can distinguish processes $a.b + b.a$ from $a\|b$ (where $a.b$ denotes the sequential execution of a and b, $+$ denotes the choice and $\|$ the parallel operator). These two processes are not distinguishable in an interleaving approach. Formalisms that allow true concurrency are, e.g., petri nets [23], event structures [26] and process algebras [12,15].

---

[1] The work started when the author worked at the University of Mannheim, Germany.

Modal logics are useful formalisms for specification. An example is the μ-calculus [14], which is based on an interleaving approach, i.e., it cannot express true concurrency. One possibility to obtain a truly concurrent logic is to use a different action set in the μ-calculus, e.g., to allow multi-sets of actions, which corresponds to step semantics [21]. But the step observation is not always sufficient, e.g., it is not preserved by action refinement [7]. Moving to more complex action sets, e.g., pomsets [22], makes the logical expression very complex and unreadable, which is unacceptable for a specification language.

In order to investigate a comprehensible truly concurrent logic that can be automatically checked, we extend the μ-calculus such that properties that are valid during the execution of an action can be described. More precisely, instead of using formulas $\langle a \rangle \varphi_1$ we take formulas of the form $\langle a - \phi_1 \rangle \phi_2$ with the meaning that ϕ_2 has to hold after the execution of action a (i.e., after the start and the immediate termination of a) and ϕ_1 has to hold immediately after the start of a. This new logic, which is called *in-between logic* (IB-logic for short), is interpreted over transition systems corresponding to the ST-semantics. The IB-logic has the advantages that:

- It is a truly concurrent logic, e.g., $\langle a - \langle b - true \rangle true \rangle true$ says that after the start of action a another action b can be started, which is only possible if a and b are concurrent.

- Its underlying semantical model is based on the well-understood transition system paradigm.

- It is an intuitive and comprehensible extension of the well-understood μ-calculus. In particular, the μ-calculus can be straightforwardly embedded into the IB-logic, i.e., the IB-logic has more expressive power than the μ-calculus.

- It can be embedded in the μ-calculus that uses an extended action set. Hence, all the results and the tools existing for the μ-calculus can be used for the IB-logic. In particular, validity of IB-formulas can be automatically checked.

- It is easy to understand and is therefore less error-prone than for example logic approaches that deal with an extended action set.

An interesting question is the expressive power of this logic. For example is it more expressive than step observation? Or how it is related to the ST-approach [8], which was investigated to obtain the coarsest equivalence with respect to interleaving bisimulation equivalence that is preserved under action refinement. In the ST-approach an action is no longer considered to be atomic and is split into two uniquely related actions corresponding to its start, respectively, its termination.

In order to answer these questions, we present an equivalence notion, called IB-bisimulation, and show that two processes are IB-equivalent iff they satisfy the same set of IB-formulas. We show that the expressive power of the IB-equivalence is incomparable to the step bisimulation. We also show that the ST-bisimulation is strictly more expressive than the IB-equivalence. On the other hand, we verify that the IB-formulas are expressive enough, indeed, to characterize ST-equivalence for processes that are obtained from a finite process algebra and that do not contain action synchronization.

The outline of the paper is as follows: Stack-based transition systems are introduced in Section 2. A process algebra together with its ST-based semantics in terms of stack-based transition systems is also presented there. Section 3 introduces the IB-logic together with the embedding with respect to the μ-calculus. In Section 4, IB-bisimulation is introduced and its correspondence to the ST-bisimulation is examined. Related work is discussed in Section 5 and a conclusion together with a discussion of future work is given in Section 6.

2 Adapted Transition Systems

In order to illustrate the usage of stack-based transition systems in the context of semantics based on action splitting, we present a process algebra and give it an ST-based operational semantics. Our processes algebra uses a parallel operator similar to CSP [12].

2.1 The Syntax of the Process Algebra

Let $\mathcal{A}ct$ be a set of actions. A relabelling function f is a function from $\mathcal{A}ct$ to $\mathcal{A}ct$. The set of all relabelling functions is denoted by F_L. Furthermore, we assume a fixed countable set of *process variables* $\mathcal{V}ar^P$ which is disjoint from $\mathcal{A}ct$. The process algebra expressions EXP are defined by the following BNF-grammar:

$$B ::= \mathbf{0} \mid a.B \mid B + B \mid B\|_A B \mid B[f] \mid x,$$

where $f \in F_L$, $x \in \mathcal{V}ar^P$, $a \in \mathcal{A}ct$ and $A \subseteq \mathcal{A}ct$. A *process* with respect to EXP is a pair $\langle \text{decl}, B \rangle$ consisting of a declaration $\text{decl} : \mathcal{V}ar^P \to$ EXP and an expression $B \in$ EXP. Let PA denote the set of all processes. We sometimes call an expression $B \in$ EXP also a process if the decl part is clear from the context.

The expressions have the following intuitive meaning: $\mathbf{0}$ is the inactive process, i.e., it cannot execute any action; $a.B$ is the action prefix process, which can execute a and evolves to B afterwards. In the following, we write a

for the process $a.\mathbf{0}$. Process $B_1 + B_2$ denotes the choice between the behaviors described by B_1 and B_2. As usual, the choice is triggered when actions are starting, which is contrary to [5], where the choice is triggered by the termination of actions. Process $B_1\|_A B_2$ describes the parallel execution of B_1 and B_2 where both processes have to synchronize on actions from A. The relabelling process $B[f]$ executes action $f(a)$ if B executes action a. The behavior of x is given by the declaration.

Remark 2.1 We only consider an action prefix operator instead of the more general sequential operator $B_1; B_2$ in order to avoid the introduction of process termination in our setting. This is done to increase readability.

2.2   Stack-Based Transition Systems

In the ST-semantics [8,7] an action is not considered to have an atomic execution. More precisely, an action is split into two actions corresponding to its start, respectively, its termination where additionally to the pure split-semantics of [10] the termination of an action is uniquely connected to the start of this action. There are different techniques of encoding the ST-semantics in terms of transition systems. See [2] for an overview. We choose to use the *stack technique* [2], since it has the following advantages:

- It produces finite transition systems for a wide class of processes. Hence, ST-bisimulation equivalence is decidable for more processes. Moreover, transition systems obtained from a process algebra expression have less states when the stack-based technique is used [2].

- It is compositional, i.e., the transition system of a process can be derived from the transition system of its components. This has the advantage of simplifying the derivation of an axiomatization [2].

- It yields an appropriate method to handle the operational semantics of the action refinement operator, as illustrated in [9].

   The intuitive idea behind the stack technique is the following, where an *active action* denotes an action that has been started but has not terminated yet: the start of an action a is denoted in the transition system labels by a^+ and the termination of an action a is denoted by a_n^-, where the natural number n indicates that exactly $n - 1$ many a-actions that were started after the start of the a^+ action corresponding to the a_n^- action are still active. In other words, if an a-action starts at execution position t_s and terminates with a_n^- at execution position t_f, then the number of the a-actions that were started after position t_s and that are not yet terminated before position t_f is exactly $n - 1$. An illustration that may help to understand this approach
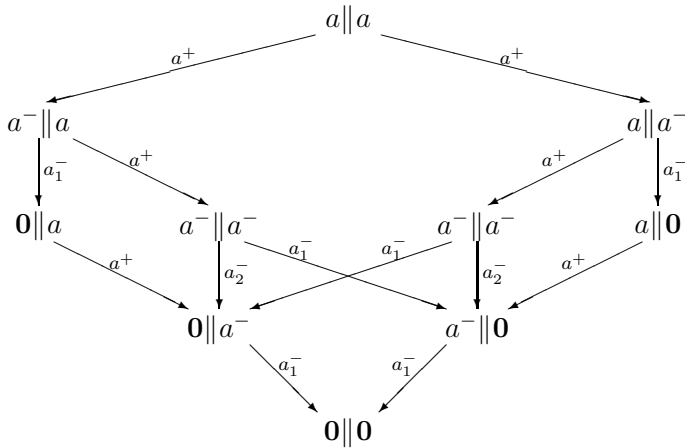
Fig. 1. Illustration of the Stack Technique

is given in Figure 1, where the stack-based transition system derived from process $a\|a$ is presented. Number n is called the *relative active number* of the action corresponding to a_n^-. In the following, $\mathbb{N}^+$ denotes the set of all natural numbers different from zero.

Definition 2.2 [Stack-Based Transition System] A *stack-based transition system* over $\mathcal{A}ct$ is a transition system with labels from $\mathcal{L} = (\mathcal{A}ct \times \{+\}) \cup (\mathcal{A}ct \times \mathbb{N}^+)$, i.e., a tuple $(S, \mathcal{A}ct, \longrightarrow)$ with

- S a non-empty set of *states* and
- $\longrightarrow \subseteq S \times \mathcal{L} \times S$ a *stack-based transition relation.*

We write $s \xrightarrow{\gamma} s'$ rather than $(s, \gamma, s') \in \longrightarrow$. Furthermore, elements of $\mathcal{A}ct \times \{+\}$ are denoted by a^+ and elements of $\mathcal{A}ct \times \mathbb{N}^+$ are denoted by a_n^-. The class of all stack-based transition systems is too general for the description of processes. Useful restrictions are presented informally in the following definition.

Definition 2.3 A stack-based transition system is:

- *atomic sensitive* if the start of every action execution can be immediately terminated,
- *interrupt free* if every active action can be immediately terminated,
- *action termination deterministic* if every state can have at most one transition for every termination action,
- *termination disabling free* if 'everything' that can be executed before the termination of an action is also possible afterwards, e.g., the termination of an action cannot disable a choice, and

- *start enabling free* if 'everything' that can be executed after the start of an action is also possible before, e.g., the start of an action cannot be a causality of another one.

The atomic sensitive and action termination deterministic property usually hold for all processes. The termination disabling and the start enabling freeness are satisfied usually by processes derived from process algebras. Interrupt freeness is normally satisfied when no disrupt/interrupt operator exists in the process algebras.

States that do not have active actions are denoted as *non-active states*. More precisely, a state s is defined to be non-active if every state s' that can terminate an action and that is reachable from s by an execution sequence implies that this action must be started during the execution sequences that lead to s'.

How a transition system that has action labels instead of start and termination action labels can be derived from a stack-based transition system is described as follows:

Definition 2.4 Let $(S, \mathcal{A}ct, \longrightarrow)$ be a stack-based transition system. Then its *un-splitting transition system* is the transition system $(S, \mathcal{A}ct, \longmapsto)$ with $s \xmapsto{a} s'$ iff there is s'' such that $s \xrightarrow{a^+} s'' \xrightarrow{a_1^-} s'$.

The ST-bisimulation equivalence is defined for stack-based transition systems as follows:

Definition 2.5 [ST-Bisimilarity] Let $(S, \mathcal{A}ct, \longrightarrow)$ be a stack-based transition system. An ST-bisimulation with respect to this stack-based transition system is a symmetric relation $R \subseteq S \times S$ such that for all $(s_1, s_1') \in R$ and $\gamma \in \mathcal{L}$ we have
$\forall s_2 : s_1 \xrightarrow{\gamma} s_2 \Rightarrow (\exists s_2' : s_1' \xrightarrow{\gamma} s_2' \wedge (s_2, s_2') \in R)$.
Two elements $s, s' \in S$ are *ST-bisimilar* (or ST-equivalent), written as $s \sim_{ST} s'$, if there is an ST-bisimulation R such that $(s, s') \in R$.

Remark 2.6 The Hennessy-Milner logic [11] and, therefore, also the μ-calculus [14] can characterize bisimulation over labelled transition systems, in the sense that two states are bisimilar iff they satisfy the same formulas. Therefore, these logics can be used to characterize the ST-equivalence just by using $(\mathcal{A}ct \times \{+\}) \cup (\mathcal{A}ct \times \mathbb{N}^+)$ as their underlying action set. But then the formulas become very difficult to read, since there are a lot of indices denoting the relative active numbers and it is hard to see in a formula to which start action a termination action corresponds. Furthermore, these logics do not take into account that the natural appearing stack-based transition systems
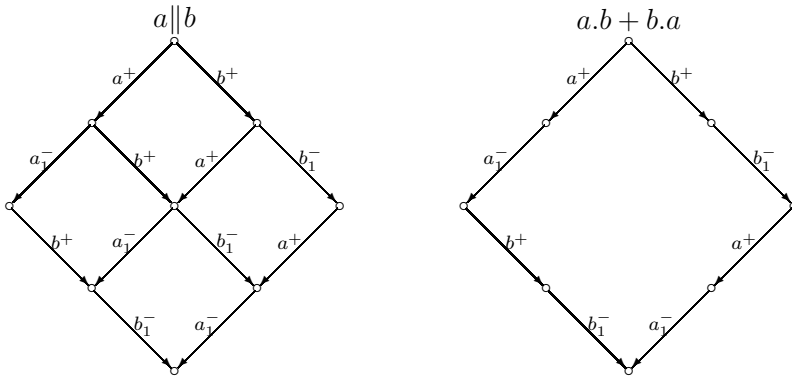
Fig. 2. Some Stack-Based Transition Systems

have special features (compare with Definition 2.3). Hence, these logics are maybe unnecessarily too expressive.

2.3 The ST-Based Operational Semantics of the Process Algebra

The definition of the operational semantics is as in [2]. Here, we only present examples showing how the operational semantics is derived from a process algebra expression. The derived stack-based transition system of process $a\|a$ is given in Figure 1 (where the parallel operator must have the additional information to which side the different relative active number corresponds). The derived stack-based transition system of processes $a\|b$ and $a.b + b.a$ are presented in Figure 2, where it is assumed that a and b are different action names.

A stack-based transition system derived from an expression of EXP satisfies all properties introduced in Definition 2.3. Furthermore, the transition system derived from a process of EXP with pure action execution is the same as the unsplitting transition system derived from the stack-based operational semantics of this process (restricted to the reachable states). Moreover, a stack-based transition system derived from an expression of EXP has finitely many states iff its transition system obtained by pure action execution has finitely many states.

2.4 Step Transition System

The step transition system is a transition system with finite multi-sets of actions as its labels. Formally:

Definition 2.7 [Step Transition System] A *step transition system* over $\mathcal{A}ct$ is a transition system $(S, \mathcal{A}ct, \leadsto)$ with labels from $\mathcal{L}^{\text{step}} = \{\omega : \mathcal{A}ct \to \mathbb{N} \mid \infty > \sum_{a \in \mathcal{A}ct} \omega(a)\}$, and so $\leadsto \subseteq S \times \mathcal{L}^{\text{step}} \times S$.

Step-equivalence is similarly defined as ST-equivalence, except that step instead of stack-based transition systems are used. A corresponding step transition system is derived from a stack-based transition system as follows:

Definition 2.8 Suppose $(S, \mathcal{A}ct, \longrightarrow)$ is a stack-based transition system. Then its *step transition system* is the transition system $(S, \mathcal{A}ct, \rightsquigarrow)$ with $s \overset{\omega}{\rightsquigarrow} s'$ iff $\forall a \in \mathcal{A}ct : \omega(a) > 0 \Rightarrow \exists s_1, s_2 : s \xrightarrow{a^+} s_1 \overset{\omega[a \mapsto (\omega(a)-1)]}{\rightsquigarrow} s_2 \xrightarrow{a_1} s'$, where $\omega[a \mapsto (\omega(a) - 1)$ is the function that is everywhere equal to ω except on a where its value is reduced by 1.

The step transition system derived directly from a process of EXP is the same as the step transition system derived from the stack-based operational semantics of this process (restricted to the reachable states).

3   In-Between Logic

In this section, we present a modification of the μ-calculus [14] such that properties that have to hold during the execution of an action can be described.

3.1   Syntax of the Logic

Let $\mathcal{V}ar^L$ be a set of *logic variables*. The IB-formulas are generated according to the following grammar:

$$\phi ::= false \mid true \mid X \mid \langle a - \phi \rangle \phi \mid [a - \phi]\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \mu X.\phi \mid \nu X.\phi,$$

where $X \in \mathcal{V}ar^L$ and $a \in \mathcal{A}ct$. The set of all formulas is denoted by $\mathcal{F}$. A formula is *closed* if every occurrences of variable X appears inside a formula of form $\mu X.\phi$ or $\nu X.\phi$.

The intuition of most of these formulas is the same as for the μ-calculus, where, e.g., $\mu X.\phi$ denotes the least and $\nu X.\phi$ denotes the greatest fix-point formula. The intuition of $\langle a - \phi_1 \rangle \phi_2$ is that there is an execution of action a such that ϕ_2 has to hold after the complete execution of this a whereas ϕ_1 has to hold after the start of this a. In particular, the formula states that there is an action a that can be started and that can be immediately terminated. Please note that ϕ_2 only has to hold immediately after the termination of a, i.e., there is no statement which has to hold after the termination of a when further executions took place during the execution of a. The intuition of $[a - \phi_1]\phi_2$ is that whenever an action a is started which can be terminated immediately, then ϕ_1 holds after the start of this a or ϕ_2 holds immediately after the termination of this a. This definition is on the first sight strange, since one would expect a definition like 'after every start ϕ_1 has to hold and

immediately after every termination ϕ_2 has to hold'. The presented interpretation of the box formula is chosen, since the box formula yields the dual of the diamond formula in our interpretation. The above described formula can be described within our interpretation by $([a - \phi_1]false) \wedge ([a - false]\phi_2)$. We do not introduce a negation formula explicitly, since negation can be modelled by a duality operator, which will be shown later.

3.2 Semantics of the Logic

Suppose $T = (S, \mathcal{A}ct, \longrightarrow)$ is a stack-based transition system. Then the semantics $[\![\_]\!]_T : \mathcal{F} \times (\mathcal{V}ar^L \to \mathcal{P}(S)) \to \mathcal{P}(S)$ is defined as follows, where $\mathcal{P}(S)$ denotes the power set of S.

$$[\![false]\!]_T^\varsigma = \emptyset \qquad\qquad [\![true]\!]_T^\varsigma = S \qquad\qquad [\![X]\!]_T^\varsigma = \varsigma(X)$$

$$[\![\langle a - \phi_1 \rangle \phi_2]\!]_T^\varsigma = \{s \in S \mid \exists s' \in [\![\phi_1]\!]_T^\varsigma, s'' \in [\![\phi_2]\!]_T^\varsigma : s \xrightarrow{a^+} s' \xrightarrow{a_1^-} s''\}$$

$$[\![[a - \phi_1]\phi_2]\!]_T^\varsigma = \{s \in S \mid \forall s', s'' \in S : (s \xrightarrow{a^+} s' \wedge s' \xrightarrow{a_1^-} s'') \Rightarrow$$
$$(s' \in [\![\phi_1]\!]_T^\varsigma \vee s'' \in [\![\phi_2]\!]_T^\varsigma)\}$$

$$[\![\phi_1 \wedge \phi_2]\!]_T^\varsigma = [\![\phi_1]\!]_T^\varsigma \cap [\![\phi_2]\!]_T^\varsigma \qquad\qquad [\![\phi_1 \vee \phi_2]\!]_T^\varsigma = [\![\phi_1]\!]_T^\varsigma \cup [\![\phi_2]\!]_T^\varsigma$$

$$[\![\mu X.\phi]\!]_T^\varsigma = \bigcap \{M \in \mathcal{P}(S) \mid [\![\phi]\!]_T^{\varrho[X \mapsto M]} \subseteq M\}$$
$$[\![\nu X.\phi]\!]_T^\varsigma = \bigcup \{M \in \mathcal{P}(S) \mid [\![\phi]\!]_T^{\varrho[X \mapsto M]} \supseteq M\},$$

where $\varsigma[X \mapsto M]$ denotes the function that is everywhere equal to ς except on X where it is equal to M. A state $s \in S$ is a model of a closed IB-formula ϕ, written as $s \models \phi$, if $s \in [\![\phi]\!]_T^\varsigma$ for some ς. In the rest of the paper, $(S, \mathcal{A}ct, \longrightarrow)$ denotes a stack-based transition system. Furthermore, we omit the index T in $[\![\ ]\!]_T$ if it is clear from the context.

Example 3.1 Consider the IBL-formula $\phi = \langle a - \langle b - true \rangle true \rangle true$. Then the stack-based transition system on the left hand side of Figure 2 satisfies ϕ, whereas the stack-based transition system on the right hand side of Figure 2 does not satisfy ϕ. In other words, $a\|b$ and $a.b + b.a$ can be distinguished by an IB-formula. Thus the IBL logic can be considered as a truly concurrent logic.

Example 3.2 The processes $a\|b$ and $(a\|b) + a.b$, which are step-bisimilar, are distinguished by the IB-formula $\langle a - [b - false]false \rangle true$.

Remark 3.3 The semantics of the diamond (and the box) operator can be defined differently, e.g., take $\{s \in S \mid \exists s' \in [\![\phi_1]\!]^\varsigma : s \xrightarrow{a^+} s' \wedge \forall s'' \in S : s' \xrightarrow{a_1^-} s'' \Rightarrow s'' \in [\![\phi_2]\!]^\varsigma\}$ as the interpretation of $\langle a - \phi_1 \rangle \phi_2$. Nevertheless,

this definition is equivalent to the presented one if atomic sensitive and action termination deterministic stack-based transition systems are considered.

In the following, we present the duality operator, which models negation.

Definition 3.4 The duality operator $\mathcal{D} : \mathcal{F} \to \mathcal{F}$ is inductively defined as follows:

$$\mathcal{D}(false) = true \qquad \mathcal{D}(true) = false \qquad \mathcal{D}(X) = X$$

$$\mathcal{D}(\langle a - \phi_1 \rangle \phi_2) = [a - \mathcal{D}(\phi_1)]\mathcal{D}(\phi_2) \quad \mathcal{D}([a - \phi_1]\phi_2) = \langle a - \mathcal{D}(\phi_1) \rangle \mathcal{D}(\phi_2)$$

$$\mathcal{D}(\phi_1 \wedge \phi_2) = \mathcal{D}(\phi_1) \vee \mathcal{D}(\phi_2) \qquad \mathcal{D}(\phi_1 \vee \phi_2) = \mathcal{D}(\phi_1) \wedge \mathcal{D}(\phi_2)$$

$$\mathcal{D}(\mu X.\phi) = \nu X.\mathcal{D}(\phi) \qquad \mathcal{D}(\nu X.\phi) = \mu X.\mathcal{D}(\phi)$$

Proposition 3.5 *The duality operator corresponds to negation, i.e., for any stack-based transition system $T = (S, \mathcal{A}ct, \longrightarrow)$ and for any IB-formula ϕ we have $S \setminus [\![\phi]\!]_T^\varsigma = [\![\mathcal{D}(\phi)]\!]_T^{S \setminus \varsigma}$, where $S \setminus M = \{s \in S \mid s \notin M\}$ and $S \setminus \varsigma$ denotes the function with $(S \setminus \varsigma)(X) = S \setminus \varsigma(X)$.*

3.3 Correspondence to the μ-Calculus

We describe how the μ-calculus can be embedded in the IB-logic. The syntax of the μ-calculus is similar to the IB-logic except that the first formulas in the diamond and box expressions are omitted. The semantics function $\{\_\}^\varsigma$ of the μ-calculus is defined over a transition system $(S, \mathcal{A}ct, \longmapsto)$. Its interpretation is similar to the IB-logic interpretation except that $\{\langle a \rangle \varphi\}^\varsigma = \{s \in S \mid \exists s' \in \{\varphi\}^\varsigma : s \overset{a}{\longmapsto} s'\}$ and $\{[a]\varphi\}^\varsigma = \{s \in S \mid \forall s' \in S : (s \overset{a}{\longmapsto} s' \Rightarrow s' \in [\![\varphi]\!]^\varsigma)\}$. The transformation that maps μ-calculus formulas to IB-logic formulas is given as follows:

$$\Psi(false) = false \qquad \Psi(true) = true \qquad \Psi(X) = X$$

$$\Psi(\langle a \rangle \varphi) = \langle a - true \rangle \Psi(\varphi) \qquad \Psi([a]\varphi) = [a - false]\Psi(\varphi)$$

$$\Psi(\varphi_1 \wedge \varphi_2) = \Psi(\varphi_1) \wedge \Psi(\varphi_2) \quad \Psi(\varphi_1 \vee \varphi_2) = \Psi(\varphi_1) \vee \Psi(\varphi_2)$$

$$\Psi(\mu X.\varphi) = \mu X.\Psi(\varphi) \qquad \Psi(\nu X.\varphi) = \nu X.\Psi(\varphi)$$

Proposition 3.6 *The μ-calculus is encoded in the IB-logic via Ψ. More precisely, for any stack-based transition system $T = (S, \mathcal{A}ct, \longrightarrow)$ and any μ-calculus formula φ we have $\{\varphi\}_{\widetilde{T}}^\varsigma = [\![\Psi(\varphi)]\!]_T^\varsigma$, where $\widetilde{T}$ denotes the un-splitting transition system of T.*

Proposition 3.6 together with the fact that $a \| b$ cannot be distinguished by the μ-calculus illustrates that the IB-logic has more expressive power than the

μ-calculus that has labels from $\mathcal{A}ct$. In the following, we present an embedding transformation of the IB-logic into the μ-calculus that has labels from $\mathcal{L}$:

$$\Phi(false) = false \qquad \Phi(true) = true \qquad \Phi(X) = X$$

$$\Phi(\langle a - \phi_1 \rangle \phi_2) = \langle a^+ \rangle (\Phi(\phi_1) \wedge \langle a_1^- \rangle \Phi(\phi_2)) \qquad \Phi(\phi_1 \wedge \phi_2) = \Phi(\phi_1) \wedge \Phi(\phi_2)$$

$$\Phi([a - \phi_1] \phi_2) = [a^+](\Phi(\phi_1) \vee [a_1^-] \Phi(\phi_2)) \qquad \Phi(\phi_1 \vee \phi_2) = \Phi(\phi_1) \vee \Phi(\phi_2)$$

$$\Phi(\mu X.\phi) = \mu X.\Phi(\phi) \qquad\qquad\qquad \Phi(\nu X.\phi) = \nu X.\Phi(\phi)$$

Proposition 3.7 *The IB-logic is encoded in the μ-calculus with labels from $\mathcal{L}$ via Φ. More precisely, for any stack-based transition system $(S, \mathcal{A}ct, \longrightarrow)$ and any IB-formula ϕ we have $[\![\phi]\!]^\varsigma_{(S,\mathcal{A}ct,\longrightarrow)} = \{\!|\Phi(\phi)|\!\}^\varsigma_{(S,\mathcal{L},\longrightarrow)}$.*

By Proposition 3.7, all existing results of the μ-calculus, such as decidability of finite state systems, can be applied to the IB-logic. Furthermore, the existing model-checking tools for the μ-calculus can be used for the IB-logic. As already described in Remark 2.6, it is not in general useful to use the μ-calculus that has labels from $\mathcal{L}$ as a specification language.

4 Characterization of the IB-Logic

4.1 IB-Bisimulation

In this subsection, we introduce a bisimulation approach that characterizes the equivalence derived from the IB-logic.

Definition 4.1 [IB-Bisimilarity] Let $(S, \mathcal{A}ct, \longrightarrow)$ be a stack-based transition system. An IB-bisimulation with respect to this stack-based transition system is a symmetric relation $R \subseteq S \times S$ such that for all $(s_1, s_1') \in R$ and $a \in \mathcal{A}ct$ we have

$$\forall s_2, s_3 : (s_1 \xrightarrow{a^+} s_2 \wedge s_2 \xrightarrow{a_1^-} s_3) \Rightarrow \Big(\exists s_2', s_3' : s_1' \xrightarrow{a^+} s_2' \wedge (s_2, s_2') \in R \wedge$$

$$s_2' \xrightarrow{a_1^-} s_3' \wedge (s_3, s_3') \in R \Big).$$

Two elements $s, s' \in S$ are *IB-bisimilar* (or IB-equivalent), written as $s \sim_{IB} s'$, if there exists an IB-bisimulation R such that $(s, s') \in R$.

Lemma 4.2 *The IB-equivalence follows from the ST-equivalence, i.e., $\sim_{ST} \subseteq \sim_{IB}$.*

Theorem 4.3 *Suppose $s, s' \in S$.*

- *If $s \sim_{IB} s'$, then for every closed IB-formulas ϕ we have $s \models \phi \Leftrightarrow s' \models \phi$.*
- *If for every closed IB-formulas ϕ we have $(s \models \phi \Leftrightarrow s' \models \phi)$ and $(S, \mathcal{A}ct, \longrightarrow)$ is finitely branching, then $s \sim_{IB} s'$.*

Corollary 4.4 *Suppose* $(S, \mathcal{A}ct, \longrightarrow)$ *is a stack-based transition system and* $s, s' \in S$ *are ST-bisimilar. Then for every closed IB-formula* ϕ *we have* $s \models \phi \Leftrightarrow s' \models \phi$.

Proof. From Lemma 4.2 we obtain that s and s' are IB-equivalent. The rest is an immediate consequence of Theorem 4.3.   □

The opposite direction of Corollary 4.4, i.e., "Do the IB-formulas characterize the ST-equivalence?" does not hold, which is shown in the next subsection. But in Section 4.3 we show that the opposite of Corollary 4.4 holds for a suitable subset of stack-based transition systems.

4.2 Step-Equivalence does not follow from IB-Equivalence

Obviously, the ST-equivalence does not follow from the IB-equivalence if general stack-based transition systems are allowed. This results from the fact that actions which are already active cannot be detected by the IB-semantics. Consider, for example, the transition system $(\{s, s'\}, \mathcal{A}ct, \{(s', a^-, s')\})$, where state s cannot execute any action and state s' can only execute action a^+ and evolves to itself. Then states s and s' are IB-equivalent but not ST-equivalent.

This is not problematic, since we are only interested in comparing states in which no actions are active, i.e., in comparing non-active states. But also for this case the characterization fails, since the IB-equivalence can only consider the start of actions that can terminate immediately. Just modify the previous counterexample by choosing action a^+ instead of a^-, i.e., consider $(\{s, s'\}, \mathcal{A}ct, \{(s', a^+, s')\})$. Then again s and s' are IB-equivalent but not ST-equivalent.

The characterization also fails for stack-based transition systems that are atomic sensitive, since the IB-equivalence cannot detect disruption or interruption of active actions. Consider, e.g., the stack-based transition system given in Figure 3, where, e.g., the already started action a^+ is disrupted by the start of action b. State $\tilde{s}$ of this transition system is IB-equivalent to the transition system obtained from $a\|b$, which is shown in Figure 2. But they are not ST-equivalent.

But what about states that corresponds to processes of our process algebra, where the correspondent stack-based transition systems are, e.g., interrupt free? Unfortunately, a characterization also fails in that case. Moreover, IB-equivalence does not imply step-equivalence, which is implied by ST-equivalence. Consider the expressions

$$\tilde{B}_1 = \big(\big((a.b.c + d.e)\|_{\{c,d,e\}}(b.d.e + c)\big) \,\|_{\{b\}} b\big)\,[d \mapsto a][e \mapsto c]$$
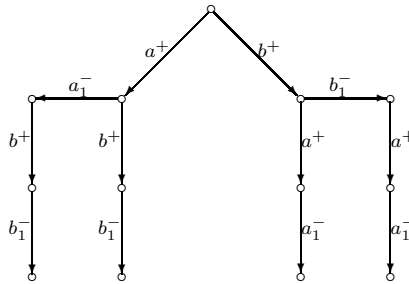$$\tilde{B}_2 = (a.(b + c)\|_{\{c\}} b.(a.c + c))\|_{\{a,b\}}(a\|_\emptyset b)$$

Fig. 3. A Stack-Based Transition System with Disruption

Then $\tilde{B}_1$ and $\tilde{B}_1 + \tilde{B}_2$ are IB-equivalent, which can be seen as follows:
The only non obvious case is how the execution from $\tilde{B}_2$ can be matched.
Suppose $\tilde{B}_2$ executes action a. Then the process

$$(1) \qquad (a^-.(b+c)\|_{\{c\}}b.(a.c+c))\|_{\{a,b\}}(a^-\|_\emptyset b)$$

has to be matched after the start of a and the process

$$(2) \qquad ((b+c)\|_{\{c\}}b.(a.c+c))\|_{\{a,b\}}(\mathbf{0}\|_\emptyset b)$$

has to be matched after the termination of a. Since the IB-equivalence cannot
observe the expression behind an active action, (1) is IB-equivalent to

$$(\mathbf{0}\|_{\{c\}}b.(a.c+c))\|_{\{a,b\}}(\mathbf{0}\|_\emptyset b)$$

Furthermore, because of the synchronization mechanism (1) is IB-equivalent
to b and (2) is IB-equivalent to $b.c + c$. On the other hand, it is straight
forwardly seen that the start (and that the start and termination) of action a
in $\tilde{B}_1$ also yield an expression that is IB-equivalent to b (respectively $b.c + c$).
Furthermore, the start of action b in $\tilde{B}_2$ yields a process IB-equivalent to a
and after the termination of b it yields a process IB-equivalent to $(a.c\|_{\{c\}}(a.c+$
$c))\|_{\{a\}}a$, which is IB-equivalent to $a + a.c$. Moreover, the start and the start
and finishing of b in $\tilde{B}_1$ yields a and respectively $\big((a + d.e)\|_{\{c,d,e\}}d.e\big)[d \mapsto$
$a][e \mapsto c]$, which is IB-equivalent to $a + a.c$. Hence $\tilde{B}_1$ and $\tilde{B}_1 + \tilde{B}_2$ are
IB-equivalent.

On the other hand, $\tilde{B}_1$ and $\tilde{B}_1 + \tilde{B}_2$ are not step equivalent, which is argued
as follows: The execution of $\{a, b\}$ in $\tilde{B}_1$ yields 0. But the execution of $\{a, b\}$
in $\tilde{B}_1$ yields a process step equivalent to c.

The possibility for synchronization of actions is essential in the above coun-
terexample. We can prove that the IB-equivalence and the ST-equivalence
coincide for finite process algebra expressions that do not contain synchro-
nization mechanism. This is done in the following subsection.

4.3   IB Characterizes ST for a Finite Process Algebra without Synchronization

The expressions EXP^{fp} of a finite process algebra without synchronization are defined by the following BNF-grammar, where $a \in \mathcal{A}ct$:

$$P ::= \mathbf{0} \mid a.P \mid P + P \mid P \| P$$

The meaning of these expression is obtained by straightforward embedding into the expression of EXP from Section 2.1, where $\|$ is interpreted as $\|_\emptyset$. For this process algebra, we obtain that the IB-logic characterizes the ST-equivalence:

Theorem 4.5 *Let $P, P' \in \mathsf{EXP}^{fp}$. Then $P \sim_{ST} P'$ iff $P \sim_{IB} P'$.*

Corollary 4.6 *Let $P, P' \in \mathsf{EXP}^{fp}$. Then $P \sim_{ST} P'$ iff for every closed IB-formula ϕ we have $s \models \phi \Leftrightarrow s' \models \phi$.*

5   Related Work

We are not aware of any truly concurrent logic that is based on the splitting of actions. There are logics defined over partial orders taking causality and concurrency into account. They do not distinguish between different interleavings of the same partially ordered execution. This property can be used to reduce state spaces by applying so-called partial order reductions [18]. The *TLC* - Temporal Logic of Concurrency [1], which is interpreted over causal structures, is such a logic. Other temporal logics exist, for which expressive completeness for Mazurkiewicz's traces has been shown; these are either local logics, interpreted at the events of a trace [3,4,6], or global logics interpreted at its cuts [24,25]. There are also approaches that interpret logics on event structures, e.g., [16,20].

Another reason of introducing logics on partial orders is that these allow a direct representation of properties involving causality and concurrency: Logics that are interpreted on trace systems (or runs) are presented, e.g., in [13,17] the logic ISTL is interpreted over partially ordered runs of global states. The logic CTL is extended by a past operator, called CTL$_P$, in [19], where it is interpreted over global states of trace systems.

6   Conclusion and Future Work

We have presented an extension of the μ-calculus, called IB-logic, that can specify some truly concurrent properties. Its semantics is based on stack-based transition systems, which are used as a model of the ST-semantics.

Tool support of the IB-logic is achieved by embedding the IB-logic into the μ-calculus that uses a start/termination-action label set. We give a characterization of the equivalence introduced by the IB-logic via IB-bisimulation. We showed that the IB-logic characterizes the ST-bisimulation equivalence for processes obtained form a finite process algebra without synchronization. Furthermore, we showed that the IB-logic and step execution have an incomparable expressiveness.

Future work will be the investigation of a more detailed distinction of the IB- from the ST-semantics, e.g., to clarify whether IB- and ST-bisimulation coincide on general process algebra expressions without synchronization. It is also of interest whether more efficient tools than the transformation into the μ-calculus can be investigated for the IB-logic. And the most interesting work will be to find a comprehensible logic that characterizes the ST-bisimulation for general processes.

Acknowledgement

References

[1] R. Alur, D. Peled, and W. Penczek. Model checking of causality properties. In *Proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science*, pages 90–100. IEEE Comp. Soc. Press, 1995.

[2] M. Bravetti and R. Gorrieri. Deciding and axiomatizing weak ST bisimulation for a process algebra with recursion and action refinement. *ACMTCL: ACM Transactions on Computational Logic*, 3, 2002.

[3] V. Diekert and P. Gastin. Pure future local temporal logics are expressively complete for Mazurkiewicz traces. To appear in LNCS.

[4] V. Diekert and P. Gastin. LTL is expressively complete for Mazurkiewicz traces. *Journal of Computer and System Sciences*, 64:396–418, 2002.

[5] H. Fecher and M. Majster-Cederbaum. Taking decisions late: End-based choice combined with action refinement. In J. Derrick, E. Boiten, J. Woodcock, and J. von Wright, editors, *REFINE 2002*, volume 70 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 2002.

[6] P. Gastin, M. Mukund, and K. N. Kumar. Local LTL with past constants is expressively complete for Mazurkiewicz traces. In *MFCS'03*, volume 2747 of *LNCS*, pages 429–438. Springer-Verlag, 2003.

[7] R. v. Glabbeek and U. Goltz. Refinement of actions and equivalence notions for concurrent systems. *Acta Informatica*, 37:229–327, 2001.

[8] R. v. Glabbeek and F. Vaandrager. Petri net models for algebraic theories of concurrency. In J. de Bakker, A. Nijman, and P. Treleaven, editors, *PARLE, Parallel Architectures and Languages Europe (Volume II)*, volume 259 of *LNCS*, pages 224–242. Springer-Verlag, 1987.

[9] R. Gorrieri and A. Rensink. Action refinement. In J. A. Bergstra, A. Ponse, and S. A. Smolka, editors, *Handbook of Process Algebra*, pages 1047–1147. North-Holland, 2001.

[10] M. Hennessy. Axiomatising finite concurrent processes. *SIAM Journal on Computing*, 17(5):997–1017, 1988.

[11] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, 1985.

[12] C. A. R. Hoare. *Communications Sequential Processes*. International Series in Computer Science. Prentice Hall, 1985.

[13] S. Katz and D. Peled. Interleaving set temporal logic. *Theoretical Computer Science*, 75:263–287, 1990.

[14] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[15] R. Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1989.

[16] M. Mukund and P. S. Thiagarajan. A logical characterization of well branching event structures. *Theoretical Computer Science*, 96:35–72, 1992.

[17] D. Peled. Proving partial order properties. *Theoretical Computer Science*, 126:143–182, 1994.

[18] D. Peled. Ten years of partial-order reductions. In *CAV'98*, volume 1427 of *LNCS*, pages 17–28. Springer-Verlag, 1998.

[19] W. Penczek. Temporal logics for trace systems: On automated verification. *International Journal of Foundations of Computer Science*, 4:31–67, 1993.

[20] W. Penczek. Model-checking for a subclass of event structures. In E. Brinksma, editor, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1217 of *LNCS*, pages 145–164. Springer-Verlag, 1997.

[21] L. Pomello. Some Equivalence Notions for Concurrent Systems. An Overview. In G. Rozenberg, editor, *Advances in Petri Nets 1985*, volume 222 of *LNCS*, pages 381–400. Springer-Verlag, 1986.

[22] V. Pratt. Modeling concurrency with partial orders. *International Journal of Parallel Programming*, 15:33–71, 1986.

[23] W. Reisig. *Petri Nets: An Introduction*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1985.

[24] P. S. Thiagarajan and I. Walukiewicz. An expressively complete linear time temporal logic for mazurkiewicz traces. *Information and Computation*, 179(2):230–249, 2002.

[25] I. Walukiewicz. Difficult configurations - on the complexity of LTrL. In K. Larsen, S. Skyum, and G. Winskel, editors, *ICALP'98*, volume 1443 of *LNCS*, pages 140–151. Springer-Verlag, 1998.

[26] G. Winskel. Event structures. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Applications and Relationship to Other Models of Concurrency, Advances in Petri Nets 1986, Part II*, volume 255 of *LNCS*, pages 325–392. Springer-Verlag, 1987.