Action Refinement in End-Based Choice Settings

Inauguraldissertation
zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften
der Universität Mannheim

vorgelegt von
Diplom-Mathematiker Harald Fecher
aus Offenbach

Mannheim, 2003
Dekan:     Professor Dr. Herbert Popp, Universität Mannheim
Referent:  Professor Dr. Mila Majster-Cederbaum, Universität Mannheim
Korreferent: Professor Dr. Franz Stetter, Universität Mannheim

## Contents

List of Figures .......................... 5

List of Tables .......................... 7

1 Introduction .......................... 9

  1.1 Reactive Systems for Concurrency .......................... 10
  1.2 Hierarchical System Design .......................... 12
  1.3 End-Based and Start-Based View of the Choice Operator .......................... 13
  1.4 Contributions of this Thesis .......................... 14
  1.5 Outline of this Thesis .......................... 15

2 Preliminaries .......................... 17

  2.1 Notations .......................... 17
    2.1.1 Set .......................... 17
    2.1.2 Relation and Function .......................... 18
  2.2 Transition System .......................... 20
  2.3 Complete Partial Orders .......................... 22
  2.4 Approximation Closedness .......................... 23
    2.4.1 Proofs .......................... 25

3 Standard Action Refinement .............. 29

  3.1 Different Approaches .......................... 29
  3.2 Syntax .......................... 30
  3.3 Denotational True Concurrency Semantics for \( \text{PA}_{sr} \) .......................... 31
    3.3.1 Bundle Event Structures .......................... 31
    3.3.2 Closed Bundle Event Structures (CBES) .......................... 34
    3.3.3 Operators on CBES .......................... 36
    3.3.4 Denotational Meaning for \( \text{PA}_{sr} \) .......................... 38
3.4 Operational Semantics for PA_{sr} .............................................. 40
  3.4.1 Modified Operational Semantics ......................................... 44
3.5 Discussion ................................................................. 45
3.6 Proofs ........................................................................... 46
  3.6.1 Proof of Theorem 3.10 ................................................................. 46
  3.6.2 Proof of Proposition 3.18 ......................................................... 47
  3.6.3 Proof of Theorem 3.25 ................................................................. 48

4 Modeling the End-Based View in CBES .................................... 57
  4.1 An End-Based Refinement Operator on CBES ........................... 57
  4.2 Equivalences ..................................................................... 60
    4.2.1 Standard Equivalence Notions ............................................ 60
    4.2.2 ICT-Equivalence on CBES ..................................................... 62
    4.2.3 UI-Bisimilarity on CBES ....................................................... 64
    4.2.4 FUI-Bisimilarity on CBES ..................................................... 65
    4.2.5 Comparison of Equivalences .................................................. 66
    4.2.6 Coarsest Congruence .......................................................... 67
  4.3 Discussion ....................................................................... 68
  4.4 Proofs ........................................................................... 69
    4.4.1 Proof of the Congruence Results .......................................... 69
    4.4.2 Proof of Proposition 4.20 ....................................................... 75
    4.4.3 Proof of Proposition 4.22 ....................................................... 77

5 Terminating by Action Execution ............................................. 79
  5.1 Motivation .................................................................... 79
  5.2 Syntax .......................................................................... 80
  5.3 Operational Semantics for PA_{st} ......................................... 81
  5.4 Denotational Semantics for PA_{st} ......................................... 83
    5.4.1 Termination Bundle Event Structure (TBES) ......................... 84
    5.4.2 Extended Termination Bundle Event Structure (ETBES) ........ 85
    5.4.3 Operators on ETBES ............................................................. 90
    5.4.4 Denotational Meaning for PA_{st} ............................................ 93
    5.4.5 Extended Termination Precursor Event Structures (ETPES) .... 94
    5.4.6 Correspondence between ETBES and ETPES ................. 97
  5.5 Discussion .................................................................... 98
  5.6 Proofs ........................................................................... 98
<table>
<thead>
<tr>
<th>CONTENTS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Bibliography</td>
<td>167</td>
</tr>
<tr>
<td>Index</td>
<td>179</td>
</tr>
<tr>
<td>Zusammenfassung</td>
<td>185</td>
</tr>
</tbody>
</table>
**List of Figures**

1.1 Nuclear Power Plant Example ............................................. 11
2.1 A Transition System Example ............................................. 21
2.2 Trace-Equivalent but not Bisimilar Transition Systems ............... 22
3.1 A Bundle Event Structure ................................................. 32
3.2 An Extended Bundle Event Structure .................................. 33
3.3 Transition System Derived from CBES ................................. 36
3.4 Illustration of the Ref operator ......................................... 38
3.5 Examples of the Denotational Semantics of EXPsr .................... 39
3.6 Example of a Process Derivation with respect to \( \rightarrow_{\text{decl}}^s \) 43
4.1 Start-Based versus End-Based Refinement ............................. 58
4.2 End-Based Refinement in CBES (1) .................................... 60
4.3 Trace Equivalent but not Bisimilar cbes ............................... 61
4.4 End-Based Refinement in CBES (2) .................................... 62
4.5 Some Closed Bundle Event Structures ................................ 62
4.6 Non ICT-Equivalent cbes ................................................. 63
4.7 ICT-Equivalent cbes ....................................................... 63
4.8 ICT-Equivalent and UI-Equivalent cbes ............................... 65
4.9 FUI-Equivalence Differs from UI-Equivalence ......................... 66
4.10 Relations Between the Equivalences ................................ 67
4.11 Counterexample of Coarsest Congruence ............................. 68
5.1 Some Extended Termination Bundle Event Structures ................. 87
5.2 Transition System Derived from ETBES ............................... 89
5.3 Some Extended Termination Precursor Event Structures ............. 95
5.4 Transition System Derived from ETPES ............................... 96
6.1 End-Based Refinement in ETBES ................................. 111
6.2 Non ICT-Equivalent eTbes .......................................... 112
6.3 ICT-Equivalent eTbes .................................................. 112
6.4 FUI-Equivalent eTbes (1) .............................................. 113
6.5 FUI-Equivalent eTbes (2) .............................................. 113
6.6 Relations Between the Equivalences .............................. 114

7.1 Some Start-End Bundle Event Structures ......................... 124
7.2 Illustration of the Stack Technique ................................. 128

8.1 Relations Between the Equivalences .............................. 165
8.2 Hierarchy of Event Structures ....................................... 166
# List of Tables

3.1 Transition Rules for $\rightarrow^{a}_{\text{decl}}$ .......................................................... 42
3.2 Modified Transition Rules for $\rightarrow^{s}_{\text{decl}}$ ......................................................... 45
3.3 Event-Based Transition Rules with respect to $\rightarrow^{a}_{\text{decl}}$ ............................... 50
5.1 Transition Rules for $\rightarrow^{t}_{\text{decl}}$ ..................................................................... 82
5.2 Event Based Transition Rules with respect to $\rightarrow^{t}_{\text{decl}}$ ................................ 100

7.1 Transition Rules for $\rightarrow^{c}_{\text{decl}}$ (1) ................................................................. 132
7.2 Transition Rules for $\rightarrow^{c}_{\text{decl}}$ (2) ................................................................. 133
7.3 Transition Rules for $\rightarrow^{c}_{\text{decl}}$ (3) ................................................................. 134
7.4 Transition Rules for $\rightarrow^{z}_{\text{decl}}$ (1) ................................................................. 140
7.5 Transition Rules for $\rightarrow^{z}_{\text{decl}}$ (2) ................................................................. 141
7.6 Axioms for the Non-Refinement Operators ............................................................. 143
7.7 Axioms for the Refinement Operators ................................................................. 144
7.8 Event Based Transition Rules with respect to $\rightarrow^{c}_{\text{decl}}$ (1) ......................... 147
7.9 Event Based Transition Rules with respect to $\rightarrow^{c}_{\text{decl}}$ (2) ......................... 148
7.10 Event Based Transition Rules with respect to $\rightarrow^{c}_{\text{decl}}$ (3) ......................... 149
Chapter 1

Introduction

Formal methods in computer science, like methods for specification and verification\(^1\), have become more and more important [51, 55], as the complexity of programs and processes that have to be controlled by machines has enormously increased over the years. No system designer can consider the totality of a program/process of a system in detail. Therefore, communication between different people about programs/processes has to take place. This communication usually subject to misunderstandings. Hence, it is important to possess formalisms that allow to talk more precisely about programs/processes, for example about the properties they should satisfy. Such languages are called specification formalisms. They allow to describe processes or properties of programs. Examples of specification languages are

- **descriptive/property-based** formalisms. Typical examples are logical frameworks, e.g. [100, 123, 124, 167]. They have the advantage of being intuitive, concise and abstract, i.e. they only consider the relevant details.

- **imperative/operational-based** formalisms. Examples are transition systems [121, 179], process algebras [18, 27, 84, 108, 138] or finite automata [110, 154]. They have the advantage of being close to actual implementations. Especially process algebras can be considered as action-based programming languages. Consequently, design descriptions are less likely to omit required attributes of the intended design, as it is possible in the property-based approaches to specification.

For safety critical tasks, for example air traffic control or the supervision of nuclear power plants, it is essential to guarantee the correctness of the involved program, i.e. verification (e.g. [83, 105]) becomes necessary. Furthermore, it is important to eliminate system design errors as early as possible, since they produce huge costs [161]. Such an error reduction can be achieved by verification based on the dual language approach (see for example [101, 151]), i.e. by using different description languages (a property- together with an operational-based one) in the system design phase and verifying their consistency, e.g. by using model checking [24, 52].

\(^1\)Verification means the correctness of a program relative to the considered mathematical model. It can never guarantee the correctness of a program running on a concrete computer, as this also depends on further circumstances.
CHAPTER 1. INTRODUCTION

For concurrent processes, i.e. systems where each process may proceed more or less independently, it is important to use formal methods, since their interaction is not easily handled, especially when communication or synchronization takes place. The formal methods that are important for this thesis are discussed in Section 1.1 and in Section 1.2. Section 1.1 introduces the kind of systems that are considered, namely reactive true concurrent systems. Section 1.2 motivates the necessity of hierarchical system design in specification formalism. The new formalism on which the thesis is based is illustrated in Section 1.3. The contribution of the thesis is given in Section 1.4, and the outline of the thesis is given in Section 1.5.

1.1 Reactive Systems for Concurrency

Reactive systems are systems whose behavior depends on the environment, which means that the environment is able to influence the future behavior of such a system. In other words, the environment interacts with the system. Consider for example a computer system that controls a nuclear power plant. It runs continuously until the environment demands a shut down, for example by human demand or because the temperature of the reactor reaches a critical state. Reactive systems are usually modeled by means of actions in order to describe the different activities of the system and of the environment, i.e. the system communicates with the environment via actions. Actions are usually considered to be atomic [34, 112] (they have no intermediate states) and instantaneous, i.e. durationless. Consequently, they can only be observed at a specific point in time. Furthermore, actions are divided into observable actions and internal actions. The execution of observable actions depends on the environment, whereas the environment has no influence on the execution of internal actions.

In the above nuclear power plant example the actions may be: \( a_1 \equiv \text{‘the reactor runs for another minute’}, \ a_2 \equiv \text{‘a human being demands the shut down of the reactor’}, \ a_3 \equiv \text{‘the temperature of the reactor has reached a critical state’} \) and \( a_4 \equiv \text{‘the reactor is shutting down’}. \) The reactive system can be described by the causal dependencies of these actions. This is done for example by:

- describing all possible (finite) execution sequences of the system. This description is called the trace semantics of a system, e.g. [48, 107].

- describing the actions each state allows (those which may be executed at a state) together with the information to which state the execution of the actions will lead. Typical models are labeled transition systems, which are introduced in Section 2.2. The labeled transition system obtained from the above nuclear power plant example is depicted in Figure 1.1.

In this thesis, we will consider concurrent reactive systems. This means that actions may be executed in parallel. Different models of concurrency are for example presented in [179]. These models can be divided into two different approaches:

- Interleaving: Here the parallelism between actions means that the execution of these actions can happen in any order. This is useful, if e.g. a sequential program has to be specified where the programmer is allowed to have a much greater degree of flexibility in order to carry out the implementation. So the programmer may decide the order of the execution of the actions. Transition systems are typical interleaving models.
1.1. REACTIVE SYSTEMS FOR CONCURRENCY

- **True concurrency**: Here, actions may be executed simultaneously. This is necessary, e.g. if physical processes, where actions may happen simultaneously, have to be specified. When an implementation is realized on a system with more than one processor, true concurrency within the model is essential for specification. Typical models for true concurrent computations are for example *petri nets* [155], *event structures* [177], *pomsets* [153] and *causal trees* [61].

Process algebras are typically used as specification languages for concurrent reactive systems. The best known process algebras are

- the *Calculus of Communicating Systems* (CCS) [138],
- the *Communicating Sequential Processes* (CSP) [106, 108],
- the *Theory of Communicating Sequential Processes* (TCSP) [47],
- the *Algebra of Communicating Processes* (ACP) [25, 26] and
- the *Language of Temporal Ordering Specification* (LOTOS) [32].

It is necessary to give meanings, i.e. semantics, to languages. In general, three different kinds of semantics are used:

*Operational semantics*: It expresses the meaning of terms by execution steps, for example the observable behavior of a machine when it runs the program. The operational semantics of process algebras is typically given by transition rules in the style of Plotkin [9, 150], which yield transition systems.

*Denotational semantics*: Process expressions are interpreted in a mathematical model where the meaning of an expression is defined in terms of the meanings of its components, i.e. it is derived compositionally. Denotational true concurrency semantics of process algebras are given, for example in terms of event structures [36, 125, 145, 178].

*Axiomatic semantics*: Here, properties of process terms can be derived. This is typically done by axiom systems. In particular, axiomatic semantics are used for verification.

Standard process algebras can only describe the functional behavior of a system, i.e. the order of the execution of the actions. This is only suitable for a limited number of applications, since applications might be influenced by time and probability aspects. The examination of these further aspects is usually called performance analysis. For the purpose of performance analysis, process algebras are extended by
• **time**: Here, time may pass between actions. Some approaches allow actions with a fixed duration [10, 57, 99], i.e. actions are not necessarily instantaneous. There are discrete time versions [17, 103, 143] and dense time versions [17, 77, 128, 164, 175]. Classification properties of timed process algebras are given in [142].

• **priority**: Formalisms are added to the basic process algebras such that some executions of actions have priority over others, as for examples in [50, 53, 129]. Such approaches can be classified by static versus dynamic priorities and by global pre-emption versus local pre-emption [54].

• **probability**: Here, a probabilistic choice operator is used to model probabilistic behavior. This choice operator contains numbers, which determines the probability that the first (respectively the second) process is chosen. Examples are [88, 115, 129, 165].

• **stochastic**: The actions are considered to have a duration and this duration may vary. The duration of an action is given by a distribution. Examples of stochastic process algebras can be found in [11, 45, 104].

• **combinations**: For example in [29, 43], the above features are modeled in single process algebras.

### 1.2 Hierarchical System Design

Whether a system satisfies a formula is automatically decidable by means of model checking techniques if the underlying transition model has a finite number of states. Nevertheless, most systems have too many states, hence model checking techniques cannot be used directly (the calculation takes too much time). This problem is usually called the state explosion problem [24, 52, 148].

Hierarchical system design [109], a design that is developed on different levels of abstraction, can be used in order to handle the state space explosion problem. For example, a developer divides the intended design (usually complex) into various ‘sub-designs’. He will develop the sub-designs by enriching them step by step with details, i.e. changing the level of abstraction. If the properties are preserved in some sense for every step between the levels of abstraction, it may be sufficient to verify the most abstract level, which has in general less states, to show the property of the concrete level. Design formalisms have to support hierarchical system design styles, as argued in [13, 55, 135, 169].

In reactive system models, hierarchical system design is usually done by action refinement operators\(^2\) [7, 91, 98]. Intuitively, action refinement means the refinement of actions \((a, b, \ldots)\), in a process \(P\) by more complex processes \((P_a, P_b, \ldots)\). In other words, the occurrence of action \(a\) in the behavior of \(P\) is replaced by the behavior of \(P_a\). How action refinement can be exploited for verification can be seen e.g. in [82, 111, 130, 132, 134, 162].

\(^2\)In some approaches action refinement is not considered to be a usual operator as in [158], where it is considered to be an implementation relation.
1.3  End-Based and Start-Based View of the Choice Operator

In models of concurrent systems, actions are usually considered to be instantaneous, i.e. durationless, as mentioned before. However, if real time aspects of systems have to be modeled and/or action refinement operators are employed, we have to take into account that actions consume time\(^3\).

A standard operator for modeling reactive systems is the choice between two processes \(P_1 + P_2\), i.e. if \(P_1\) executes the next action, \(P_2\) is disabled and vice versa. If durational actions are considered, it is not clear when an action has to be considered to be executed. In particular, it is not clear at which point in time an action triggers a choice – at the beginning, at the end or anywhere in the middle of its duration?

The consequence of this decision is illustrated by the following example. Consider a process that consists of a choice between actions \(a\) and \(b\). The duration of \(a\) is 3 and the duration of \(b\) is 1. In addition, action \(a\) may start at time 0 and action \(b\) may start at time 1. If the choice is triggered at the beginning, then \(a\) triggers the choice before \(b\) starts. On the other hand, if the choice is triggered by the end of an action, the choice is triggered by \(b\), i.e. \(a\) does not finish.

In the standard approach, a choice is triggered by the start of an involved event (action) \([10, 99, 133, 141, 174]\). But it is reasonable to consider approaches where choices are determined by the ending of actions:

- In stochastic approaches, it is common to consider a race policy approach \([12, 29, 104]\), i.e. the fastest action triggers the choice. Consequently, a choice has to be triggered at the end of the action’s duration, since it is usually not known a priori which action is the fastest.
- The end-based point of view is of interest for hierarchical system development, where complex activities are specified by single actions in the first system design steps. This is illustrated by the following example.

**Example 1.1** Consider the example of a plane that runs into problems and has to land as fast as possible. Two airports (in the same city) come into consideration for the emergency landing. The pilot sends an SOS-signal to both airports. Both airports start their preparations for the emergency landing. The pilot will choose the airport that is the first to respond to be ready. On an abstract level the pilot can be modeled by

\[
P_{ab} = \text{send}; ((ok_1; L_1) \+ (ok_2; L_2))
\]

where \(\text{send}\) denotes the sending of the SOS-Signal, \(ok_i\) is the response of the \(i\)-th airport \(A_i\) (that runs in parallel to \(P_{ab}\) synchronizing over \(ok_i\)), and \(L_i\) denotes the landing on the \(i\)-th airport. Furthermore, \(\;\) denotes the sequential composition of two processes. The choice in \(P_{ab}\) is either triggered by \(ok_1\) or by \(ok_2\), as usual.

In practice, the airports will send more detailed information, e.g. that the maneuvering area is free, fire service is ready, and so on. In other words, actions \(ok_1\) and \(ok_2\) are time-consuming. Then the choice in \(P_{ab}\) has to be considered as end-based, since the choice should be made when the first airport has completed its preparations.

---

\(^3\)Action refinement operators can, for example, split an action into a start- and an end-action, hence the action’s duration can be modeled in some sense.
This is easily understood when we consider the next system design phase, where the ok actions are specified in more detail, i.e. they are refined by a process Mi. The choice of the pilot is triggered when either M1 or M2 terminates and not when the first action is executed by M1 or M2. In particular, the actions of M2 that are executed before the termination of M1 remain visible, i.e. they are not made undone after the termination of M1, and vice versa. This makes clear that the end-based choice can be viewed as some kind of parallelism, where M1 and M2 run in parallel until one of them terminates.

The possibility of late decisions is also motivated and examined in Z [166].

1.4 Contributions of this Thesis

The goal of this thesis is to make a first step to establish the end-based choice operator in reactive true concurrent systems. More precisely, we want to establish an end-based choice operator in untimed reactive systems that contain action refinement operators. This is done by investigating a process algebra that contains these operators and by giving a semantic foundation (operational, denotational, axiomatic) to this process algebra. Furthermore, the consequences of considering an end-based rather than a start-based choice are examined, in particular with respect to equivalence notions.

The contributions of this thesis are explained in more detail in the following. We have to give operational semantics to a process algebra that contains end-based choice and action refinement operators. This leads to some problems which we first consider in the start-based setting: Action refinement operators in event structures where the choice is considered as start-based are well established. A corresponding definition for the operational semantics of process algebras is not obvious when non-atomic action refinement is considered. The typical approach to the substitution of the refining process (either statically or dynamically) [5, 144], sometimes called syntactical action refinement, does not always correspond to the refinement of event structures [93]. In this thesis, a new possibility to define an operational semantics for action refinement that corresponds to the denotational semantics is given in a start-based choice setting. Here, it is not necessary to introduce new syntactic terms in order to give the operational semantics.

Furthermore, by considering the end-based view in bundle event structures, we have recognized that bundle event structures [125, 126] fail to be a complete partial order. Therefore, we present a new technique in order to define complete partial orders for event structures that are based on the bundle technique. This is necessary in this thesis, since we use event structures that are based on the bundle technique as denotational semantics for our end-based process algebra. These new techniques will be applied first in the start-based setting.

The first step to the end-based approach is the definition of an action refinement operator on (extended) bundle event structures where the conflict relation is considered as end-based. Furthermore, new equivalences are defined in order to obtain equivalences that are congruences for the end-based action refinement operator, since the standard equivalences are no congruences for this operator. The new equivalences fail to be the coarsest with respect to trace and bisimulation equivalence. This results from the fact that processes terminate by the execution of a special termination action and not by the execution of the ‘final’ executed action.

---

4The theory of complete partial orders allows to define denotations of recursive processes.
Before we continue to present a process algebra with an end-based choice operator, we take
a closer look at the termination philosophy that a process should terminate by the execution
of its ‘final’ action and not (as usual) by an additional termination action. We will argue that
this termination philosophy is especially of interest if process algebras with a disrupt operator
(as it is implicitly the case for process algebras that contain action refinement and end-based
choice operators) are considered. Therefore, two new kinds of event structures, which allow
more general disabling, are given to obtain denotational semantics of process algebras that
contain disruption and model termination by the execution of the ‘final’ action. One event
structure models the ‘non-disabling’ of events rather than the disabling of events. This is done
by making use of a witness relation. The other event structure is a generalization of Winskel’s
event structures. We show that there is consistency [19, 68] between the operational and the
denotational semantics of a process algebra which contains disruption and which is based on
the new termination philosophy. The expressive power of our new kind of event structure is
also examined: We verify that these structures have the same expressive power and they are
more expressive than the standard event structures with respect to event traces. Furthermore,
we adapt the equivalences that are defined in the context of the end-based view to this new type
of event structures. We show that one of the adapted equivalences is the coarsest congruence for
the end-based refinement operator with respect to trace (respectively bisimulation) equivalence.

We argue that it is useful to have also a start-based choice operator whenever an end-based
choice operator is considered, since an end-based choice can model a kind of start-based choice
in the case of synchronized parallel execution. Therefore, we will finally consider a process
algebra that includes a start-based and an end-based choice together with an action refinement
operator. A class of event structures with two relations for disabling is introduced in order to
give a denotational semantics. An operational semantics, which is consistent with the denota-
tional semantics, is given. We define the coarsest congruence with respect to bisimulation
equivalence and we present an axiom system for this equivalence. Moreover, we show that the
axiom system is sound and complete for guarded and finite state processes.

Parts of this thesis are published in [78, 80].

1.5 Outline of this Thesis

Chapter 2 contains some preliminaries: First the notions used in this thesis are introduced.
Then transition systems and parts of the partial orders theory are presented. The final section
of Chapter 2 contains the results of the new approximation closedness property, which is later
used in order to define classes of event structures that yield complete partial orders.

In Chapter 3, the standard action refinement operator, i.e. the one that is based on the start-based
choice, is presented and examined. That chapter includes the new operational semantics and the
modification of bundle event structures that yield a complete partial order.

The action refinement operator that considers an end-based choice is given in Chapter 4. It
is defined on the modified extended bundle event structures mentioned before. Congruences
for this refinement operator will be introduced. We show that none of the new equivalences is
the coarsest one with respect to trace/bisimulation equivalence. It is also argued that extended
bundle event structures are not appropriate to model the end-based view. In addition, some
standard equivalences are summarized in this chapter.
In Chapter 5, we investigate a process algebra that includes a disrupt operator. Here, termination is determined by the execution of the ‘final’ action. In this chapter, the classes of event structures that allow more general disabling are introduced and their expressive power are examined. Chapter 5 also contains a denotational semantics in the class of event structures based on the witness approach. A consistency result between the operational and the denotational semantics is shown.

The end-based view, introduced in Chapter 4, is adapted in Chapter 6 to one of the classes of event structures presented in Chapter 5. Here, the adapted equivalences of Chapter 4 are the coarsest congruences with respect to trace (respectively bisimulation) equivalence. We also argue that some kinds of start-based choices can be modeled with an end-based choice operator together with a parallel operator where some actions have to be synchronized.

A process algebra that contains a start-based and an end-based choice operator at the same time is introduced in Chapter 7. A denotational semantics of this process algebra, which also contains a refinement operator, is presented there. Moreover, a consistent operational semantics is given. Chapter 7 also contains the definition and the axiom system of the coarsest congruence with respect to bisimulation equivalence.

Finally, a conclusion is given in Chapter 8.
Chapter 2

Preliminaries

2.1 Notations

In this section, we present some basic notations that are used in this thesis. Subsection 2.1.1 considers notions related to sets, whereas Subsection 2.1.2 considers notions related to relations and functions. In this section, $M$, $M_1$, $M_2$ and $M_3$ denote arbitrary sets.

2.1.1 Set

- $\mathbb{N}^+$ denotes the positive natural numbers, i.e. $\mathbb{N}$ without 0.
- $M_1 \setminus M_2 = \{m \in M_1 \mid m \notin M_2\}$
- $|M|$ denotes the cardinality of set $M$.
- $M$ is countable if $|M| \leq |\mathbb{N}|$.
- $\mathcal{P}(M) = \{A \mid A \subseteq M\}$
- $\mathcal{P}_{\text{fin}}(M) = \{A \subseteq M \mid |A| < |\mathbb{N}|\}$
- $\mathcal{P}_{\text{count}}(M) = \{A \subseteq M \mid |A| \leq |\mathbb{N}|\}$
- $M^n = \underbrace{M \times \cdots \times M}_{n\text{-times}} = \{(m_1, \ldots, m_n) \mid m_i \in M\}$ where $n \in \mathbb{N}^+$.
- $M^*$ denotes the set of all strings – including the empty string $\epsilon$ – over set $M$. Here, a string of $M$ is a finite sequence of elements of $M$. We sometimes write strings $m_1 \cdots m_n$ where $m_i = m$ for all $i$ as $m^n$.

Furthermore, if $m \in M$, $\sigma \in M^*$ with $\sigma = m_1 \cdots m_n$ and $i \in \mathbb{N}$ with $i \leq n$ then

- the $i$-th element of $\sigma$ is denoted by $\sigma[i]$.
- $m \cdot \sigma = mm_1 \cdots m_n$
CHAPTER 2. PRELIMINARIES

- $\sigma \setminus i$ denotes the deletion of the $i$-th element of $\sigma$, i.e.
  \[ \sigma \setminus i = m_1 \cdots m_{i-1} m_{i+1} \cdots m_n. \]
- $\sigma \pm (i,m)$ denotes the replacement of the $i$-th element of $\sigma$ by $m$, i.e.
  \[ \sigma \pm (i,m) = m_1 \cdots m_{i-1} mm_{i+1} \cdots m_n. \]
- the length of $\sigma$ is denoted by $|\sigma|$, i.e. $|\sigma| = n$.

The following definition introduces a universe of events. Events are used in the following to denote different occurrences of actions. We need this universe in order to guarantee that event structures, which are defined in the following chapters, are sets rather than classes. This enables us to apply the theory of complete partial orders, which is presented in Section 2.3, directly.

**Definition 2.1 (Universe of Events)** Let $\bullet, \ast_1, \ast_2$ and $\ast$ be arbitrary, but fixed, pairwise different symbols. Then the universe of events, denoted $\mathcal{U}$, is an arbitrary, but fixed, countable set such that $\bullet \in \mathcal{U}$, $\ast \notin \mathcal{U}$ and $\forall e, e' \in \mathcal{U} : (e, e'), (\ast_1, e), (\ast_2, e), (\ast, e), (e, \ast) \in \mathcal{U}^1$.

We use $s_1 \ldots s_n \bullet$ where $s_i \in \mathcal{U} \cup \{\ast_1, \ast_2, \ast\}$ as an abbreviation for $(s_1, (s_2, \ldots (s_n, \bullet) \ldots))$.

### 2.1.2 Relation and Function

- For any binary relation $\mathcal{I} \subseteq M_1 \times M_2$ we write
  - $m_1 \mathcal{I} m_2$ if and only if $(m_1, m_2) \in \mathcal{I}$ and
  - $\not\mathcal{I} m_2$ for the set $\{ m \in M_1 \mid m \not\mathcal{I} m_2 \}$.

- $\text{Id}^M \subseteq M \times M$ denotes the identity relation, i.e. $\text{Id}^M = \{(m, m) \mid m \in M \}$. The index $M$ is omitted if it is clear from the context.

- If $\mathcal{I}_1 \subseteq M_1 \times M_2$ and $\mathcal{I}_2 \subseteq M_2 \times M_3$ are two binary relations, then $\mathcal{I}_1 \circ \mathcal{I}_2$ denotes the binary relation given by $\{(m_1, m_3) \in M_1 \times M_3 \mid \exists m_2 \in M_2 : m_1 \mathcal{I}_1 m_2 \land m_2 \mathcal{I}_2 m_3 \}$

- $\langle M, \sqsubseteq \rangle$ denotes the set $M$ ordered by the partial order $\sqsubseteq$.

- $M_1 \to M_2$ (or $M_2^M$) denotes the set of all functions from $M_1$ to $M_2$. We denote that $f$ is a function from $M_1$ to $M_2$ by $f : M_1 \to M_2$. The function from $M_1$ to $M_2$ that maps every element of $M_1$ to $m \in M_2$ is denoted by $\text{cons}^M_{m} : M_1 \to M_2$, where index $M_1 \to M_2$ is omitted if it is clear from the context. Furthermore, if $f : M_1 \to M_2$ and $f'' : M_2 \to M_3$ then
  - $f'' \circ f$ is the function from $M_1$ to $M_3$ given by $(f'' \circ f)(m) = f''(f(m))$.
  - $f(M)$, where $M \subseteq M_1$, denotes the image of $M$ under $f$, i.e. $f(M) = \{ f(m) \mid m \in M \}$.
  - $f \upharpoonright M$ where $M \subseteq M_1$ is the function from $M$ to $M_2$ with $(f \upharpoonright M)(m) = f(m)$ for any $m \in M$.

\footnote{1It is clear that such an $\mathcal{U}$ exists.}
2.1. NOTATIONS

- \( f[m_1 \rightarrow m_2] \) where \( m_1 \in M_1, m_2 \in M_2 \) is the function from \( M_1 \) to \( M_2 \) with 
  \[
  (f[m_1 \rightarrow m_2])(m) = \begin{cases} 
  f(m) & \text{if } m \neq m_1 \\
  m_2 & \text{otherwise} 
  \end{cases} .
  \]
- if \( f \) is bijective, then \( f^{-1} \) denotes the inverse function of \( f \), i.e. \( f^{-1} : M_2 \rightarrow M_1 \) with \( f^{-1}(m_2) = m_1 \Leftrightarrow f(m_1) = m_2 \).

\( \bullet \) \( M_1 \rightarrow M_2 \) denotes the set of all partial functions from \( M_1 \) to \( M_2 \). We denote that \( f \) is a partial function from \( M_1 \) to \( M_2 \) by \( f : M_1 \rightarrow M_2 \). The partial function from \( M_1 \) to \( M_2 \) that is everywhere undefined is denoted by \( \bot_{M_1 \rightarrow M_2} \). The partial function from \( M_1 \) to \( M_2 \) that maps every element from \( M_1 \) to \( m \in M_2 \) is denoted by \( \text{cons}_{M_1 \rightarrow M_2} \). The index \( M_1 \rightarrow M_2 \) is omitted in both cases if it is clear from the context.

Furthermore, if \( f, f' : M_1 \rightarrow M_2, f'' : M_2 \rightarrow M_3 \) and \( f''' : M_3 \rightarrow M_2 \) then

- the domain of \( f \), denoted by \( \text{dom}(f) \), is the set \( \{ m \in M_1 \mid f(m) \text{ is defined} \} \).
- We define \( f \cup f' \) by considering \( f \) and \( f' \) as relations.
- \( f'' \circ f \) is the partial function from \( M_1 \) to \( M_3 \) given by \( (f'' \circ f)(m) = f''(f(m)) \) if \( m \in \text{dom}(f) \cap f(m) \in \text{dom}(f'') \) and undefined otherwise.
- \( f(M) \), where \( M \subseteq M_1 \), denotes the image of \( M \) under \( f \), i.e. \( f(M) = \{ f(m) \mid m \in M \} \).
- We write \( f(m_1) \simeq f'''(m_3) \) to denote that \( f(m_1) \) is defined \( \Leftrightarrow f'''(m_3) \) is defined \( \land f(m_1) \) is defined \( \Rightarrow f(m_1) = f'''(m_3) \).
- \( f \mid M \) where \( M \subseteq M_1 \) is the partial function from \( M_1 \) to \( M_2 \) with \( (f \mid M)(m) \simeq f(m) \) whenever \( m \in M \) and undefined otherwise.
- \( f[m_1 \rightarrow m_2] \) where \( m_1 \in M_1, m_2 \in M_2 \) is the partial function from \( M_1 \) to \( M_2 \) with 
  \[
  (f[m_1 \rightarrow m_2])(m) = \begin{cases} 
  f(m) & \text{if } m \neq m_1 \\
  m_2 & \text{otherwise} 
  \end{cases} .
  \]
- \( f \) is injective (surjective, bijective) from \( M'_1 \subseteq M_1 \) to \( M'_2 \subseteq M_2 \) if and only if \( \text{dom}(f) = M'_1, f(M'_1) \subseteq M'_2 \) and \( f \mid M'_1 \) is an injective function from \( M'_1 \) to \( M'_2 \) (respectively surjective, bijective). We call \( f \) injective if \( f \) is injective between \( \text{dom}(f) \) and \( M_2 \).
- if \( f \) is bijective, then \( f^{-1} \) denotes the inverse partial function of \( f \), i.e. \( f^{-1} : M_2 \rightarrow M_1 \) with \( f^{-1}(m_2) \simeq m_1 \iff f(m_1) \simeq m_2 \).

\( \bullet \) \( M \rightarrow^\text{fin} \mathbb{N} \) denotes the set of all functions from \( M \) to the natural numbers that differs only finitely often from 0, i.e. \( M \rightarrow^\text{fin} \mathbb{N} = \{ f : M \rightarrow \mathbb{N} \mid |\{ m \in M \mid f(m) \neq 0 \}| < \infty \} \).

\( \bullet \) \( M_1 \rightarrow M_2 \) denotes the set of all functions from \( M_1 \) to the set of all strings over \( M_2 \) that differs only finitely often from the empty string, i.e. \( M_1 \rightarrow M_2 = \{ f : M_1 \rightarrow M_2^* \mid |\{ m \in M_1 \mid f(m) \neq \epsilon \}| < \infty \} \).

The function that maps every element of \( M_1 \) to the empty string of \( M_2^* \) is denoted by \( \bot_{M_1 \rightarrow M_2} \). The index \( M_1 \rightarrow M_2 \) is omitted if it is clear from the context.

\( \bullet \) \( \pi_i \) denotes the projection to the \( i \)-th component of a Cartesian product


- $\cong$ is used to denote isomorphism of structures, i.e. a structure preserving bijective function.

- A binary relation $\vdash \subseteq M \times M$ is preserved by an operator $F : M \rightarrow M$ if and only if $m_1 \vdash m_2 \Rightarrow F(m_1) \vdash F(m_2)$. Furthermore, $\vdash$ is preserved by an operator $F_n : M^n \rightarrow M$ if and only if $F_n(m_1, ..., m_n) \vdash F_n(m_1', ..., m_n')$ whenever $m_i \vdash m_i'$ for $i = 1, .., n$. $F$ is called monotonic for the special case when $\vdash$ is a partial order.

**Definition 2.2** Let $\equiv \subseteq M \times M$ and let $\mathcal{F}$ be a set of operators where for every $f \in \mathcal{F}$ there exists $i \in \mathbb{N}$ such that $f : M^i \rightarrow M$. Then

- $\equiv$ is a congruence for $\mathcal{F}$ if and only if $\equiv$ is an equivalence relation and $\equiv$ is preserved by all operators of $\mathcal{F}$

- $\equiv_c \subseteq M \times M$ is the coarsest congruence for $\mathcal{F}$ with respect to $\equiv$ if and only if
  - $\equiv_c \subseteq \equiv$
  - $\equiv_c$ is a congruence for $\mathcal{F}$
  - whenever $\equiv' \subseteq M \times M$ is a congruence for $\mathcal{F}$ such that $\equiv' \subseteq \equiv$ then $\equiv' \subseteq \equiv_c$

### 2.2 Transition System

*Labeled transition systems*, originally introduced by [121] under the name ‘named transition systems’, represent a model to describe the behavioral character of a process. This is done by abstracting complex activities into a single action. The duration of actions is often neglected, i.e. the actions are considered to be instantaneous, in order to obtain a simpler, time independent model. Transition systems are used to describe to which state the execution of an action may lead.

**Definition 2.3 (Transition System)** A (labeled) transition system is a quadruple $(S, L, \rightarrow, \bar{s})$ with

- $S$, a non-empty set of states
- $L$, a set of labels
- $\rightarrow \subseteq S \times L \times S$, a transition relation
- $\bar{s} \in S$, the initial state.

We will write $p \xrightarrow{\gamma} q$ rather than $(p, \gamma, q) \in \rightarrow$. The class of all transition systems is denoted by $\text{TS}$. 
2.2. TRANSITION SYSTEM

The intuitive meaning of $p \xrightarrow{\gamma} q$ is that the execution of $\gamma$ in state $p$ may lead to state $q$. It is not necessarily uniquely determined to which state the execution of $\gamma$ in state $p$ leads, since it is possible that $p \xrightarrow{\gamma} q'$ is another transition of the transition system. Examples of transition systems are shown in Figure 1.1 and in Figure 2.1, where the initial state is marked by an arrow without a source.

Transition systems, as well as other models, are often considered as too concrete descriptions. Therefore, equivalences are defined on transition systems in order to identify those which we consider to display the same behavior in some sense.

Two basic equivalences have been defined for transition systems: trace equivalence [108], which considers the possible sequences of observable behavior, and bisimulation equivalence [136], which also takes the branching structure into account.

**Definition 2.4 (Trace Equivalence)** The set of traces of a transition system $(S, L, \rightarrow, \bar{s})$ is defined by

$$T(S, L, \rightarrow, \bar{s}) = \{ (\gamma_i)_{i<n} \mid n \in \mathbb{N} \land \exists s_0, \ldots, s_n \in S \land s_0 = \bar{s} \land \forall i < n : s_i \xrightarrow{\gamma_i} s_{i+1} \}.$$  

We sometimes write $T(\bar{s})$ if $S, L, \rightarrow$ are clear from the context.

Two transition systems $(S, L, \rightarrow, \bar{s})$ and $(S', L, \rightarrow', \bar{s}')$ over the same set of labels $L$ are trace equivalent, which is denoted by $(S, L, \rightarrow, \bar{s}) \sim_t (S', L, \rightarrow', \bar{s}')$ or $\bar{s} \sim_t \bar{s}'$ for short, if $T(\bar{s}) = T(\bar{s}')$.

**Definition 2.5 (Bisimilarity)** Two transition systems $(S, L, \rightarrow, \bar{s})$ and $(S', L, \rightarrow', \bar{s}')$ over the same set of labels are bisimilar (or bisimulation equivalent), denoted by $(S, L, \rightarrow, \bar{s}) \sim_b (S', L, \rightarrow', \bar{s}')$ or $\bar{s} \sim_b \bar{s}'$ for short, if there is a bisimulation, i.e. a relation $\mathcal{R} \subseteq S \times S'$ such that $(\bar{s}, \bar{s'}) \in \mathcal{R}$ and for each $(s_1, s_1') \in \mathcal{R}$ we have:

- if $s_1 \xrightarrow{\gamma} s_2$, then there is $s_2'$ such that $(s_2, s_2') \in \mathcal{R}$ and $s_1' \xrightarrow{\gamma'} s_2'$
- if $s_1' \xrightarrow{\gamma'} s_2'$, then there is $s_2$ such that $(s_2, s_2') \in \mathcal{R}$ and $s_1 \xrightarrow{\gamma} s_2$.

**Remark 2.6** Bisimilar transition systems are also trace equivalent.
The transition systems from Figure 1.1 and Figure 2.1 are bisimilar and therefore also trace equivalent. The transition systems presented in Figure 2.2 are trace equivalent but not bisimilar.

Various other equivalences have been defined for transition systems. An overview over equivalences that lie between trace and bisimulation equivalences with respect to their discriminating power is given in [86, 170]

### 2.3 Complete Partial Orders

To obtain denotational semantics for systems that include recursion, it is usually necessary to employ model with fixpoint theory. Complete metric spaces with contracting functions, e.g. [74], denote a class where fixpoints always exist. They are used for example in [21, 65, 117] to define a denotational semantics of recursive systems. Another class where fixpoints always exist is given by the (ω-)complete partial orders (cpo) with continuous functions. An overview of this theory is given in [3]. Complete partial orders are used for example in [4, 21, 76, 147, 149, 178] to obtain a denotational semantics of recursive systems. The definition and the results that are used in the following chapters are summarized in this section. For further details consult [3]. Here, the general cpo theory is restricted to the ω-cpo theory, because the latter is sufficient for our purpose.

**Definition 2.7 (ω-complete partial order)** A set $D$ with the partial order $\sqsubseteq$ is a (pointed) ω-complete partial order (cpo) if

- $D$ has a least element ($\perp$) with respect to $\sqsubseteq$, i.e. $\forall d \in D : \perp \sqsubseteq d$
- for every ω-chain $(d_i)_{i \in \mathbb{N}}$, i.e. $\forall j \in \mathbb{N} : d_j \sqsubseteq d_{j+1}$, there exists a least upper bound $(\bigsqcup_{i \in \mathbb{N}} d_i)$ in $D$, i.e.
  - $\forall j \in \mathbb{N} : d_j \sqsubseteq \bigsqcup_{i \in \mathbb{N}} d_i$
  - $\forall d \in D : (\forall j \in \mathbb{N} : d_j \sqsubseteq d) \Rightarrow \bigsqcup_{i \in \mathbb{N}} d_i \sqsubseteq d$

**Definition 2.8 (ω-continuous functions)** A function $f$ between cpo $D$ and cpo $E$, i.e. $f : D \rightarrow E$, is ω-continuous (or continuous for short) if it preserves the least upper bounds of ω-chains, i.e. for all ω-chains $(d_i)_{i \in \mathbb{N}}$ of $D$ we have $f(\bigsqcup_{i \in \mathbb{N}} d_i) = \bigsqcup_{i \in \mathbb{N}} f(d_i)$. 

---

**Figure 2.2: Trace-Equivalent but not Bisimilar Transition Systems**

The transition systems from Figure 1.1 and Figure 2.1 are bisimilar and therefore also trace equivalent. The transition systems presented in Figure 2.2 are trace equivalent but not bisimilar.
2.4. APPROXIMATION CLOSEDNESS

In particular, every continuous function is monotonic.

**Theorem 2.9** Let $M$ be a set and let $\langle D, \sqsubseteq_D \rangle$ and $\langle E, \sqsubseteq_E \rangle$ be cpos. Then

- the function space $M \to D$ with the pointwise order, i.e. $g \sqsubseteq g' \iff \forall m \in M : g(m) \sqsubseteq_D g'(m)$, is a cpo. Moreover, $\bigsqcup_{i \in \mathbb{N}} g_i = g$, where $g(m) = \bigsqcup_{i \in \mathbb{N}} g_i(m)$.

- and the Cartesian product $D \times E$ with the componentwise order, i.e. $(d, e) \sqsubseteq (d', e') \iff (d \sqsubseteq_D d' \land e \sqsubseteq_E e')$, is a cpo. Moreover, $\bigsqcup_{i \in \mathbb{N}} (d_i, e_i) = (\bigsqcup_{i \in \mathbb{N}} d_i, \bigsqcup_{i \in \mathbb{N}} e_i)$.

**Lemma 2.10** Let $D, D', E$ be cpos and $f : (D \times D') \to E$. Then $f$ is continuous if and only if it is componentwise continuous, i.e. for all $d' \in D'$: $f^{(1)}_{d'} : D \to E$, where $f^{(1)}_{d'}(d) = f(d, d')$, is continuous and for all $d \in D$: $f^{(2)}_d : D' \to E$, similarly defined, is continuous.

**Theorem 2.11** Let $D$ be a cpo and let $f : D \to D$ be a continuous function, then $f$ has a least fixpoint $\text{fix}(f)$, which is given by $\bigsqcup_{i \in \mathbb{N}} f^i(\bot)$.

## 2.4 Approximation Closedness

In this section, we define when a set $M \subseteq \mathcal{P}(E)$ is approximation closed. These sets are used to guarantee that an $\omega$-chain of event structures introduced in later chapters will have a least upper bound.

**Definition 2.12** Let $E$ be a countable set. A finite, monotone approximation of $E$ is a sequence $(E_i)_{i \in \mathbb{N}}$ such that $\bigcup_{i \in \mathbb{N}} E_i = E \land \forall k : E_k \subseteq E_{k+1} \land |E_k| < \infty$.

It is obvious that every countable set has a finite, monotone approximation.

**Definition 2.13** Let $E$ be a countable set and $M \subseteq \mathcal{P}(E)$. We say that $M$ is approximation closed with respect to $E$ if

- $X \in M$ whenever $X \subseteq E$ and there is a finite, monotone approximation $(E_i)_{i \in \mathbb{N}}$ of $E$ such that $\forall k \in \mathbb{N} : \exists X_k \in M : X_k \cap E_k = X \cap E_k$.

**Example 2.14** If $E$ is finite, then every $M \subseteq \mathcal{P}(E)$ is approximation closed with respect to $E$. Another example is $\mathcal{P}(\mathbb{N})$, which is approximation closed with respect to $\mathbb{N}$.

On the other hand, $\mathcal{P}_{\text{fin}}(\mathbb{N})$ is not approximation closed with respect to $\mathbb{N}$, since $\forall n \in \mathbb{N} : \{m \in \mathbb{N} \mid m \leq n\} \in \mathcal{P}_{\text{fin}}(\mathbb{N})$ but $\mathbb{N} \notin \mathcal{P}_{\text{fin}}(\mathbb{N})$. Also $\mathcal{P}(\mathbb{N})\setminus\{\emptyset\}$ is not approximation closed with respect to $\mathbb{N}$.

**Proposition 2.15** Suppose $M_1, M_2$ are approximation closed with respect to $E$. Then $M_1 \cap M_2$ and $M_1 \cup M_2$ are approximation closed with respect to $E$. 
Proof: The proof is given in Subsection 2.4.1.

For the following proofs, it is necessary to have further set constructions that yield approximation closed sets. In order to verify the approximation closedness, we construct a set $X$ out of a sequence $(X_n)_{n \in \mathbb{N}}$ of sets (where $X_n \in M$). We show that this constructed $X$ is in $M$ whenever $M$ is approximation closed. The construction of $X$ is generalized in the sense that $X$ is constructed out of two given sequences. This generalization is needed in some of the approximation closedness proofs.

Definition 2.16 Let $E_j$ be a set and $\kappa_j : \mathbb{N} \to E_j$ such that $\kappa_j$ is bijective for $j = 1, 2$. Furthermore, let $\vec{X}(j) = (X_n^{(j)})_{n \in \mathbb{N}}$ be a sequence of elements of $\mathcal{P}(E_j)$ for $j = 1, 2$.

Define $\mathcal{X}(\vec{X}(1), \kappa_1, \vec{X}(2), \kappa_2) = (\bigcup_{k \in \mathbb{N}} A_k^{(1)} \cup \bigcup_{k \in \mathbb{N}} A_k^{(2)})$ where $A_n^{(j)} \subseteq E_j$ and $N_n^{(j)} \subseteq \mathbb{N}$ is given by $A_0^{(1)} = A_0^{(2)} = \emptyset$, $N_0^{(0)} = N_0^{(1)} = N_0^{(2)} = \mathbb{N}$ and for $j = 1, 2$

$$N_{n+1}^{(0)} = N_n^{(2)}$$

$$N_{n+1}^{(j)} = \begin{cases} 
\{q \in N_{n+1}^{(j-1)} \mid \kappa_j(n) \in X_q^{(j)}\} & \text{if } \kappa_j(n) \in \bigcap_{k \geq k, i \in N_{n+1}^{(j-1)}} X_i^{(j)} \\
\{q \in N_{n+1}^{(j-1)} \mid \kappa_j(n) \notin X_q^{(j)}\} & \text{otherwise}
\end{cases}$$

$$A_{n+1}^{(j)} = \begin{cases} 
A_n^{(j)} \cup \{\kappa_j(n)\} & \text{if } \kappa_j(n) \in \bigcap_{k \geq k, i \in N_{n+1}^{(j-1)}} X_i^{(j)} \\
A_n^{(j)} & \text{otherwise}
\end{cases}$$

Note that $|N_n^{(j)}|$ is always infinite, since $\kappa_j(n) \in \bigcap_{k \geq k, i \in N_{n+1}^{(j-1)}} X_i^{(j)}$ holds exactly when $\kappa_j(n) \in X_i^{(j)}$ for infinitely many $i \in N_{n+1}^{(j-1)}$.

Proposition 2.17 Suppose $M_j$ is approximation closed with respect to $E_j$, $\vec{X}(j) = (X_n^{(j)})_{n \in \mathbb{N}}$ is a sequence of elements of $M_j$ and $\kappa_j : \mathbb{N} \to E$ is a bijective function for $j = 1, 2$. Then $\pi_j(\mathcal{X}(\vec{X}(1), \kappa_1, \vec{X}(2), \kappa_2)) \in M_j$ for $j = 1, 2$.

Proof: The proof is given in Subsection 2.4.1.

As a consequence of Proposition 2.17 we obtain the following corollaries. Their proofs are given in Subsection 2.4.1.

Corollary 2.18 Suppose $M$ is approximation closed with respect to $E$ and $E' \subseteq E$. Then

$$\{X \cap E' \mid X \in M\}$$

is approximation closed with respect to $E'$.

Corollary 2.19 Suppose $M_1, M_2$ are approximation closed with respect to $E$. Then

$$\{X_1 \cup X_2 \mid X_1 \in M_1 \land X_2 \in M_2\}$$

is approximation closed with respect to $E$. 
2.4. APPROXIMATION CLOSEDNESS

Corollary 2.20 Suppose $M_i$ is approximation closed with respect to $E_i$ for $i = 1, 2$. Then
\[
\{(e_1, e_2) \in E_1 \times E_2 \mid e_1 \in X_1 \land e_2 \in X_2 \} \cup \{(e_1, e_2) \in E_1 \times E_2 \mid i \in \{1, 2\} \land X_i \in M_i\}
\]
are approximation closed with respect to $E_1 \times E_2$.

Corollary 2.21 Suppose $M$ is approximation closed with respect to $E$ and for all $e \in E$ let $E_e$ be a set and $M_e$ be a collection of subsets such that $M_e$ is approximation closed with respect to $E_e$. Then
\[
\{(e, \hat{e}) \mid e \in X \land \hat{e} \in X_e \} \cup \{X \in M \land \forall e \in E : X_e \in M_e\}
\]
is approximation closed with respect to $\{(e, \hat{e}) \mid e \in E \land \hat{e} \in E_e\}$.

2.4.1 Proofs

Proof of Proposition 2.15: Let $X \subseteq E$ and $(E_n)_{n \in \mathbb{N}}$ be a finite monotone approximation of $E$ such that $\forall n \in \mathbb{N} : \exists X_n : X_n \in M_1 \land X_n \in M_2 \land X \cap E_n = X_n \cap E_n$. By the approximation closedness of $M_i$, we obtain that $X \in M_i$, which completes this case.

Proof of Proposition 2.17: Let $A^{(j)}_n$ and $N^{(j)}_n$ be defined as in Definition 2.16. Define $E^{(j)}_n = A^{(j)}_n \cup \left[ \bigcup_{r \leq n} \{\kappa_j(r)\} \right]$. Then $E^{(j)}_n \subseteq E^{(j)}_{n+1}$ and
\[
\forall n : \kappa_j(n) \in \bigcup_p E^{(j)}_p
\]
which is verified as follows. Suppose $\kappa_j(n) \in \bigcap_k \bigcup_{i \geq k, i \in N^{(j-1)}_n} X^{(j)}_i$, then $\kappa_j(n) \in A^{(j)}_{n+1}$. Hence, $\kappa_j(n) \in \bigcup_p E^{(j)}_p$. Now suppose $\kappa_j(n) \notin \bigcup_k \bigcup_{i \geq k, i \in N^{(j-1)}_n} X^{(j)}_i$. Then there is a $k$ such that $\kappa_j(n) \notin \bigcup_{i \geq k, i \in N^{(j-1)}_n} X^{(j)}_i$. Define $m = \max\{k, n\}$. We get $\kappa_j(n) \notin \bigcup_{i \geq m, i \in N^{(j-1)}_n} X^{(j)}_i$ since $N^{(j-1)}_{m+1} \subseteq N^{(j-1)}_{n+1}$. Hence, $\kappa_j(n) \in E^{(j)}_m$ which establishes (2.1).
\[
\forall n \in \mathbb{N} : \forall q \in N^{(j-1)}_{n+1} : A^{(j)}_n \subseteq X^{(j)}_q
\]
This can be proven by induction, where the claim is easily seen to hold in the base case. Now suppose $q \in N^{(j-1)}_{n+2}$ then $q \in N^{(j-1)}_{n+1}$ and so by induction $A^{(j)}_n \subseteq X^{(j)}_q$. If $A^{(j)}_{n+1} = A^{(j)}_n$, the
Suppose $A_{n+1} = A_n \cup \{\kappa_j(n)\}$. Then $\kappa_j(n) \in \cap_{i \geq k, i \in N_{n+1}} X_i$. From $N_{n+2} \subseteq N_{n+1}$ and the definition of $N_{n+1}$ we get $\kappa_j(n) \in X_j$, which verifies (2.2).

Now we show $\cap_{i \geq k, i \in N_{n+1}} X_i$ is infinite. Then by (2.2) and the fact that $|N_{n+1}|$ is infinite. Hence, $E_{n+1} \cap \cap_{i \geq k, i \in N_{n+1}} X_i = E_n \cap \cap_{i \geq k, i \in N_{n+1}} X_i$. Therefore, by the approximation closedness of $M$ we get $\cap_{i \geq k, i \in N_{n+1}} X_i$ is a finite, monotone approximation of $E$, which follows from (2.1).

**Proof of Corollary 2.18:** Define $M' = \{X \cap E' \mid X \in M\}$. Suppose $X' \subseteq E'$ and $(E'_n)_{n \in \mathbb{N}}$ is a finite, monotone approximation of $E'$ such that $\forall n : \exists X'_n \in M' : X'_n \cap E'_n = X' \cap E'_n$. By the definition of $M'$, for all $n \in \mathbb{N}$ there exists $X_n \in M$ such that $X_n \cap E'_n = X' \cap E'_n$. If $E$ is finite, the proof is trivial. Therefore, let $\kappa : \mathbb{N} \to E$ be bijective. Then by Proposition 2.17 we have $\pi_1(X(\bar{X}, \kappa, \bar{X}, \kappa)) \in M$, where $\bar{X} = (X_n)_{n \in \mathbb{N}}$. Let $A_{n+1}$ and $N_{n+1}$ be defined as in Definition 2.16.

It remains to prove that $E' \cap \pi_1(X(\bar{X}, \kappa, \bar{X}, \kappa)) = X'$ by which then $X' \in M'$.

$\subseteq$: Suppose $e \in E' \cap \cup_{i \geq 0} A_{i+1}$. Then there exists $g \in \mathbb{N}$ such that $\kappa(g) = e$. Moreover, there exists $n$ such that $\kappa(g) \in E'_n$, since $\kappa(g) \in E'$. From $\kappa(g) \in \cup_{i \geq 0} A_{i+1}$ we get $\kappa(g) \in \cap_{i \geq n, i \in N_{n+1}} X_i$, which is a subset of $\cup_{i \geq n, i \in N_{n+1}} X_i$. Then there is an $i \in \mathbb{N}$ such that $i \geq n$ and $\kappa(g) \in X_i$. Thus $\kappa(g) \in X_i \cap E_n \cap E'_n = X' \cap E'_n$. Hence, $\kappa(g) \in X'$.

$\supseteq$: Suppose $e \in X'$. Then there is $g \in \mathbb{N}$ such that $\kappa(g) = e$. Moreover, there is an $n$ such that $\kappa(g) \in E'_n$, since $X' \subseteq E'$. Then for all $i \geq n$ we have $\kappa(g) \in X_i \cap E'_n = X' \cap E'_n = X_i \cap E'_n = X_i \cap E'_n$. Thus $\forall k : \kappa(g) \in \cup_{i \geq n, i \in N_{n+1}} X_i$, since $|N_{n+1}|$ is infinite. Hence, $\kappa(g) \in \cap_{i \geq n, i \in N_{n+1}} X_i$. And so by definition $\kappa(g) \in A_{n+1} \subseteq \cup_{i \geq 0} A_{i+1}$.

**Proof of Corollary 2.19:** Suppose $X \subseteq E$ and $(E_n)_{n \in \mathbb{N}}$ is a finite monotone approximation of $E$ such that $\forall n \in \mathbb{N} : \exists X_n \in M_i : X \cap E_n = (X_n^{(1)} \cup X_n^{(2)}) \cap E_n$.

If $E$ is finite, the proof is trivial. Therefore, let $\kappa : \mathbb{N} \to E$ be bijective. Then by Proposition 2.17 we have $\pi_j(X(\bar{X}, \kappa, \bar{X}, \kappa)) \in M_j$, where $\bar{X} = (X_n)_{n \in \mathbb{N}}$. Let $A_{n+1}$ and $N_{n+1}$ be
2.4. APPROXIMATION CLOSEDNESS

defined as in Definition 2.16. It remains to prove that

$$\bigcup_{j \in \{1, 2\}} \pi_j(\mathcal{X}(\tilde{X}^{(1)}, \kappa, \tilde{X}^{(2)}, \kappa)) = X.$$  

\begin{itemize}
  \item \textbf{\subseteq:} Suppose $e \in (\bigcup_p A_p^{(1)}) \cup (\bigcup_p A_p^{(2)})$. Then $e \in \bigcup_p A_p^{(j)}$ for some $j \in \{1, 2\}$. Let $g \in \mathbb{N}$ such that $\kappa(g) = e$ and let $n \in \mathbb{N}$ such that $\kappa(g) \in E_n$. From $\kappa(g) \in \bigcup_p A_p^{(j)}$ we get $\kappa(g) \in \bigcap_k \bigcup_{q \geq k, q \in N_{g+1}^{(j-1)}} X_q^{(j)}$, which is a subset of $\bigcup_{q \geq n, q \in N_{g+1}^{(j-1)}} X_q^{(j)}$. Then there exists $q \in \mathbb{N}$ such that $q \geq n$ and $\kappa(g) \in X_q^{(j)}$. Thus $\kappa(g) \in (X_q^{(1)} \cup X_q^{(2)}) \cap E_n \cap E_q \cap E_n$. Hence, $e = \kappa(g) \in X$.

  \item \textbf{\supseteq:} Suppose $e \in X$. Let $g \in \mathbb{N}$ such that $\kappa(g) = e$ and let $n \in \mathbb{N}$ such that $\kappa(g) \in E_n$. Then for all $q \geq n$ we have $\kappa(g) \in X \cap E_n = X \cap E_q \cap E_n = (X_q^{(1)} \cup X_q^{(2)}) \cap E_q \cap E_n$. Hence, $\forall i \in M_{n+1}^{(1)} : i \geq n \Rightarrow \kappa(g) \in X_i^{(1)} \cup X_i^{(2)}$.

  Suppose $\kappa(g) \in X_i^{(2)}$ for infinitely many $i \in N_{g+1}^{(1)}$. Then $\forall k : \kappa(g) \in \bigcup_{i \geq k, i \in N_{g+1}^{(1)}} X_i^{(2)}$, hence $\kappa(g) \in \bigcup_p A_p^{(2)}$.

  Now suppose $\kappa(g) \in X_i^{(2)}$ for finitely many $i \in N_{g+1}^{(1)}$. Then $\kappa(g) \in X_i^{(1)}$ for infinitely many $i \in N_{g+1}^{(1)}$. Then $\forall k : \kappa(g) \in \bigcup_{i \geq k, i \in N_{g+1}^{(0)}} X_i^{(1)}$, since $N_{g+1}^{(1)} \subseteq N_{g+1}^{(0)}$. Hence $\kappa(g) \in \bigcup_p A_p^{(1)}$.
\end{itemize}

\textbf{Proof of Corollary 2.20:} Consider the first set: Suppose $X \subseteq E_1 \times E_2$ and $(\tilde{E}_n)_{n \in \mathbb{N}}$ is a finite monotone approximation of $E_1 \times E_2$ such that $\forall n \in \mathbb{N} : \exists X_n^{(1)} \in M_1, X_n^{(2)} \in M_2 : X \cap \tilde{E}_n = \{(e_1, e_2) \in \tilde{E}_n \mid e_1 \in X_n^{(1)} \wedge e_2 \in X_n^{(2)}\}$. If $E_j$ is finite, the proof is much simpler. Therefore, let $\kappa_j : \mathbb{N} \to E_j$ be bijective. Then by Proposition 2.17 we have $\pi_j(\mathcal{X}(\tilde{X}_1, \kappa_j, \tilde{X}_2, \kappa_j)) \in M_j$, where $\tilde{X}_j = (X_n^{(j)})_{n \in \mathbb{N}}$. Let $A^{(i)}$ and $N^{(i)}$ be defined as in Definition 2.16. It remains to prove that

$$X = \{(e_1, e_2) \mid e_1 \in \pi_1(\mathcal{X}(\tilde{X}_1, \kappa_j, \tilde{X}_2, \kappa_j)) \wedge e_2 \in \pi_2(\mathcal{X}(\tilde{X}_1, \kappa_j, \tilde{X}_2, \kappa_j))\}$$

\begin{itemize}
  \item \textbf{\subseteq:} Suppose $(\kappa_1(g_1), \kappa_2(g_2)) \in X$. Let $n \in \mathbb{N}$ such that $(\kappa_1(g_1), \kappa_2(g_2)) \in \tilde{E}_n$. Then for all $q \geq n$ we have $(\kappa_1(g_1), \kappa_2(g_2)) \in X \cap \tilde{E}_n = X \cap \tilde{E}_q \cap \tilde{E}_n = \{(e_1, e_2) \in \tilde{E}_q \mid e_1 \in X_q^{(1)} \wedge e_2 \in X_q^{(2)}\} \cap \tilde{E}_n$. Hence, $\forall i \in N^{(j-1)}_{g+1} : i \geq n \Rightarrow \kappa_j(g_j) \in X_i^{(j)}$. Then $\forall k : \kappa_j(g_j) \in \bigcup_{i \geq k, i \in N^{(j-1)}_{g+1}} X_i^{(j)}$, hence $\kappa_j(g_j) \in \bigcup_p A_p^{(j)}$.

  \item \textbf{\supseteq:} Suppose $\kappa_1(g_1) \in \bigcup_p A_p^{(1)}$ and $\kappa_2(g_2) \in \bigcup_p A_p^{(2)}$. Thus $\kappa_j(g_j) \in A_j^{(j)}$. From (2.2) we have

$$\forall n \in \mathbb{N} : \forall q \in N^{(j-1)}_{g+2} : \kappa_j(g_j) \in X_q^{(j)}.$$  

(2.3)

Let $n \in \mathbb{N}$ such that $(\kappa_1(g_1), \kappa_2(g_2)) \in \tilde{E}_n$. Then by (2.3) there is $q_j \geq n$ with $q_j \in N^{(j-1)}_{g+2}$ and $\kappa_j(g_j) \in X^{(j)}_{q_j}$. Define $q$ to be $q_1$ if $g_1 > g_2$ and otherwise to be $q_2$. Then

$$\kappa_1(g_1) \in X^{(1)}_q \wedge \kappa_2(g_2) \in X^{(2)}_q.$$  

(2.4)
which can be seen as follows. If \( g_1 > g_2 \) then \( N^{(1)}_{g_2+2} \subseteq N^{(0)}_{g_1+2} \) and if \( g_1 \leq g_2 \) then \( N^{(0)}_{g_1+2} \subseteq N^{(1)}_{g_2+2} \). Hence, (2.4) is an immediate consequence of (2.3).

Therefore, we have \((\kappa_1(g_1), \kappa_2(g_2)) \in \{(e_1, e_2) \mid e_1 \in X_1^{(1)} \land e_2 \in X_2^{(2)} \} \cap \tilde{E}_n \equiv n\) \{(e_1, e_2) \in \tilde{E}_n \mid e_1 \in X_1^{(1)} \land e_2 \in X_2^{(2)} \} \cap \tilde{E}_n = X \cap \tilde{E}_n.

For the proof of the approximation closedness of the second and the third set of Corollary 2.20 we define \( M_i' = \{(e_1, e_2) \in E_1 \times E_2 \mid e_i \in X_i \} \mid X_i \in M_i \}. Then from the approximation closedness of the first set of Corollary 2.20 and from the fact that the set \( \{E_i\} \) is approximation closed with respect to \( E_i \) we obtain that \( M_i' \) is approximation closed with respect to \( E_1 \times E_2 \). Then the approximation closedness of the second set follows from Proposition 2.15, since \( \{(e_1, e_2) \in E_1 \times E_2 \mid e_i \in X_i \} \mid i \in \{1, 2\} \land X_i \in M_i \} = M_1' \cup M_2'. \) Furthermore, the approximation closedness of the third set follows from Corollary 2.19, since \( \{(e_1, e_2) \in E_1 \times E_2 \mid e_1 \in X_1 \lor e_2 \in X_2 \} \mid X_1 \in M_1 \land X_2 \in M_2 \} = X_1' \cup X_2' \mid X_1' \in M_1' \land X_2' \in M_2'. \)

**Proof of Corollary 2.21:** Let \( \tilde{E} = \{(e, \hat{e}) \mid e \in E \land \hat{e} \in E_e \} \) and define \( \pi_e(\tilde{X}) = \{\hat{e} \mid (e, \hat{e}) \in \tilde{X} \} \) for \( e \in E \) and \( \tilde{X} \subseteq \tilde{E} \). Suppose \( X \subseteq E \) and \( (E_n)_{n \in \mathbb{N}} \) be a finite monotone approximation of \( \tilde{E} \) such that \( \forall n \in \mathbb{N} : \exists X_n \in M, X_n(e) \in M_e : X \cap \tilde{E}_n = \{(e, \hat{e}) \in \tilde{E}_n \mid e \in X_n \land \hat{e} \in X_n(e) \}. \)

Without loss of generality, let \( E \) be finite, since otherwise the proof is much simpler. Therefore, let \( k : \mathbb{N} \to E \). Then by Proposition 2.17 we have \( \pi_k(X(\tilde{X}, \kappa, \tilde{X}, \kappa)) \subseteq M \), where \( \tilde{X} = (X_n)_{n \in \mathbb{N}}. \) Let \( N_{n}^{(i)} \) be defined as in Definition 2.16. First we prove

\[
X = \{(e, \hat{e}) \in \tilde{E} \mid e \in \pi_{e_1}(X(\tilde{X}, \kappa, \tilde{X}, \kappa)) \land \hat{e} \in \pi_{e_2}(X)\}
\quad (2.5)
\]

From \( e', \hat{e}' \in \pi_e(X) \) we obtain \( (e', \hat{e}') \in X \), which establishes \( \supseteq \) of (2.5). Now suppose \( (e', \hat{e}') \in X \). Let \( q \in \mathbb{N} \) such that \( e' = \kappa(q) \) and let \( n \in \mathbb{N} \) such that \( (e', \hat{e}') \in \tilde{E}_n \). Then for all \( i \geq n \) we have \( \kappa(q), \hat{e}' \in X \cap \tilde{E}_n = X \cap \tilde{E}_i \cap \tilde{E}_n = \{(e, \hat{e}) \in \tilde{E}_i \mid e \in X_i \land \hat{e} \in X_i(e)\} \cap \tilde{E}_n \). Hence, \( \forall k : \kappa(q) \in \bigcup_{i \geq n, k \in N^{(i)}_{q+1}} X_i \). Thus \( \kappa(q) \in \bigcap_{k \in N^{(i)}_{q+1}} \bigcup_{i \geq n} X_i \). And so by definition \( \kappa(q) \in \pi_{e_1}(X(\tilde{X}, \kappa, \tilde{X}, \kappa)) \), which establishes (2.5).

It remains to prove that

\[
e' \in \pi_{e_1}(X(\tilde{X}, \kappa, \tilde{X}, \kappa)) \Rightarrow \pi_{e_2}(X) \in M
\quad (2.6)
\]

We have \( \pi_{e_2}(X) \cap \pi_{e_2}(\tilde{E}_n) = \pi_{e_2}(X \cap \tilde{E}_n) = \pi_{e_2}(\{(e, \hat{e}) \in \tilde{E}_n \mid e \in X_n \land \hat{e} \in X_n(e)\}) = X_n(e) \cap \pi_{e_2}(\tilde{E}_n) \) whenever \( e' \in X_n \). From \( e' \in \pi_{e_1}(X(\tilde{X}, \kappa, \tilde{X}, \kappa)) \) we obtain that \( e' \in X_n \) for infinitely many \( n \). And so (2.6) follows from the approximation closedness property, since \( (\pi_{e_2}(\tilde{E}_n))_{n \in \mathbb{N}} \), where \( N = \{n \in \mathbb{N} \mid e' \in X_n\} \), is a finite monotone approximation of \( E_{e'} \). \( \square \)
Chapter 3

Standard Action Refinement

In this chapter, we first sketch the different action refinement approaches in scientific literature. Then we illustrate the common approach on a concrete setting, i.e. we give a denotational and operational semantics to a process algebra that contains an action refinement operator. Before we present the action refinement operator on extended bundle event structures [125] (used as the denotational model), we will show that the event structures based on the bundle technique fail to yield a complete partial order (with the standard ordering). Therefore, we introduce a new subclass of extended bundle event structures that yields a complete partial order with respect to the standard ordering. This subclass is defined by using the approximation closedness techniques introduced in Section 2.4.

Furthermore, a new technique is used in this section to define an operational semantics that corresponds to the denotational semantics. This technique has the advantage of handling the disrupt expression in a feasible way and of avoiding any further syntactical expression.

3.1 Different Approaches

As mentioned in the introduction, in software design it is useful to have a top down system design [180], i.e. to change the level of abstraction until the implementation is obtained from the specification. Expressing simple actions by more concrete processes, called action refinement, reflects this methodology in the context of process algebraic settings.

Different approaches for action refinement can be distinguished:

- **atomic action refinement** [34, 64, 69, 94, 97], where the process $P$ to which the action is refined has to be considered atomic, i.e. there are no observable states in between the execution steps of $P$ (all-or-nothing). Motivations for this approach are given in [34, 69, 98].

- **non-atomic action refinement** [6, 31, 49, 60, 70, 90, 111, 114, 122, 144, 157, 173], where, as opposed to the above approach, the process to which the action is refined may interleave with the original system (or with other refinements). This approach is on the whole more popular than the atomic approach. For example, if two actions $a, b$ are completely independent, it seems unreasonable to impose a restriction stating that $b$ stays idle while the refinement of $a$ is executed.
• **relaxed forms** [113, 159, 176]. Here, the causal ordering after the refinement of actions is relaxed if the involved actions are considered to be independent. For example, suppose action \( b \) has to be preceded by action \( a \) and \( a \) is refined by the sequential composition of actions \( a_1 \) and \( a_2 \). Then \( a_1 \) may interleave with action \( b \) if \( a_1 \) and \( b \) are defined to be independent.

• **vertical action refinement** [94, 156, 158]. Here, refinement is regarded to be an implementation relation instead of being an operator, as it is done in the above cases.

For a more detailed overview over the different methods of action refinement consult [91, 98].

In this thesis, we consider non-atomic action refinement. The theory of action refinement is well established for denotational, true concurrency semantics of process algebras, i.e. refinement operators are employed in *event structures*, *petri nets* and in other models of concurrency [42, 62, 63, 72, 82, 91, 172]. When presenting an operational semantics of process algebras with a refinement operator in terms of transition systems, the action refinement operator is often handled on the process term level by *syntactic substitution* [5, 6, 144]. In general, this type of operational semantics is incompatible with the standard denotational semantics [93]. Approaches to obtain operational semantics that correspond to the denotational ones are given in [70, 98, 157].

The common non-atomic action refinement approach is illustrated in the rest of this chapter. It also contains a new technique to define an operational semantics that is consistent with the denotational semantics.

### 3.2 Syntax

We choose a process algebra that is close to *basic LOTOS* [32] except that the symbols are rather taken from [117] and that the process algebra contains an expression for action refinement.

Let \( \square \) and \( \tau \) be two different elements, which indicate the *termination* and the *internal action*. Furthermore, let \( \text{Obs} \) be a set such that \( \square, \tau \notin \text{Obs} \) and \( |\text{Obs}| > |\mathbb{N}| \). We call \( \text{Obs} \) the set of *observable actions*. The *set of all actions* \( \text{Act}_{\square} \) is defined by \( \text{Act}_{\square} = \{\square, \tau\} \cup \text{Obs} \). Assume a fixed countable set of *process variables* \( \text{Var} \) which is disjoint from \( \text{Act}_{\square} \).

The process algebra expressions \( \text{EXP}_{sr} \) (\( s \triangleq \text{start-based}, \quad r \triangleq \text{refinement} \)) are defined by the following BNF-grammar.

\[
B ::= 0 \mid 1 \mid a.B \mid \tau.B \mid B + B \mid B.B \mid B \triangleright B \mid B||A.B \mid B[(a \rightarrow B)^{a \in A}] \mid B \setminus A \mid x
\]

where \( x \in \text{Var}, \quad a \in \text{Obs} \) and \( A \subseteq \text{Obs} \) with \( |A| \leq |\mathbb{N}| \). A *process with respect to* \( \text{EXP}_{sr} \) is a pair \( \langle \text{decl}, B \rangle \) consisting of a *declaration* \( \text{decl} : \text{Var} \rightarrow \text{EXP}_{sr} \) and an expression \( B \in \text{EXP}_{sr} \).

Let \( \mathcal{PA}_{sr} \) denote the set of all processes with respect to \( \text{EXP}_{sr} \).

An expression \( B \in \text{EXP}_{sr} \) is also called a process if \( \text{decl} \) is clear from the context. We sometimes omit \( 1 \) in an expression, i.e. we write \( a \) instead of \( a.1 \).

\(^1\)The action relabeling operator [138] can be modeled by the action refinement operator in our process algebra, since \(|A|\) may be infinite.
The intuitive meanings of the expressions are as follows. 0 is the inactive process, i.e. it can not execute any action. 1 is the process that can terminate immediately. The action prefix process $a.B \ (\tau.B)$ is the process that executes $a$ (respectively $\tau$) and revolves to process $B$. Process $B_1 + B_2$ is a choice between the behaviors described by $B_1$ and $B_2$. The choice is determined by the first action that is executed. $B_1; B_2$ is the sequential composition, i.e. $B_1$ proceeds until it terminates, after which $B_2$ takes over. $B_1 \mid_A B_2$ is the disruption of $B_1$ by $B_2$, i.e. any action from $B_2$ may disable $B_1$ as long as $B_1$ has not terminated. On the other hand, the termination of $B_1$ disables $B_2$. $B_1 \parallel_A B_2$ describes the parallel composition of $B_1$ and $B_2$ where both processes have to synchronize on actions of $A$ and on $\sqrt{}$ (i.e. the parallel composition terminates if and only if both sides terminate). The intuitive meaning of the refinement expression $B[(a \to B_a)^{a \in A}]$ is that it behaves like process $B$ except that every execution of action $a$ in $A$ is substituted by the behavior of $B_a$. The hiding process $B \setminus A$ behaves like $B$ except that all actions of $A$ are renamed with $\tau$. The behavior of $x$ is given by the declaration.

3.3 Denotational True Concurrency Semantics for $\text{PA}_{sr}$

Event structures are typically used as denotational true concurrency models for process algebras. (Extended) bundle event structures are investigated in [126, 125] as a denotational model for LOTOS, on which our process algebra is based. Unfortunately, the set of all (extended) bundle event structures does not yield an $\omega$-complete partial order (cpo) with the standard order. We remedy this problem by defining a subclass of the class of extended bundle event structures that yields a cpo.

This section is organized as follows: Subsection 3.3.1 contains the definition of (extended) bundle event structures. We show that they do not yield a cpo with the standard order. The subset that yields a cpo is introduced in Subsection 3.3.2. The operators on these event structures are defined in Subsection 3.3.3. These operators are used in Subsection 3.3.4 in order to present the denotational meaning of a process.

3.3.1 Bundle Event Structures

(Extended) bundle event structures are introduced in [125, 126]. Later on, they are extended to timed versions [39, 40, 120], to stochastic versions [45, 118] and to a probabilistic version [119]. See also [46].

Definition 3.1 (Bundle Event Structure) A bundle event structure, bes for short, $(E, \sharp, \mapsto, l)$ is an element of $\mathcal{P}(U) \times \mathcal{P}(U \times U) \times \mathcal{P}(\mathcal{P}(U) \times U) \times (\mathcal{U} \Rightarrow \text{Act}_\sqrt{})$ such that

- $\sharp \subseteq (E \times E)$ and $\sharp$ is irreflexive and symmetric
- $\mapsto \subseteq \mathcal{P}(E) \times E$
- $\text{dom}(l) = E$
- $\forall X \subseteq E, e \in E : X \mapsto e \Rightarrow (\forall e', e'' \in X : e' \neq e'' \Rightarrow e' \not\sharp e'')$
CHAPTER 3. STANDARD ACTION REFINEMENT

Figure 3.1: A Bundle Event Structure

$E$ is called the set of events, $\sharp$ the (irreflexive) symmetric conflict relation, $\mapsto$ the bundle relation and $l$ the action-labeling function of the bes $(E, \sharp, \mapsto, l)$. $X$ is called a bundle of $e$ if and only if $X \mapsto e$.

The intuitive meaning of $X \mapsto e$ is that before $e$ is enabled, an event of $X$ has to be executed. $e' \sharp e$ means that the execution of $e$ disables $e'$ forever, and vice versa. The action that may be observed when an event is executed is given by the action-labeling function. The last constraint in the definition of bundle event structures is called bundle stability constraint. It guarantees the absence of causal ambiguity, i.e. exactly one event of a bundle of $e$ is executed before $e$ is enabled, and so no confusion which event causes $e$ arises. Bundle event structures and flow nets [35, 38] have exactly the same expressiveness [37].

Bundle event structures can be used as a semantic model for expressions of $\text{EXP}_{sr}$ that do not contain disrupt operators. For example, in [125] the expression $((a + b.0)||_{\{a\}}(a.c + a)) ; d$ is modeled by the bes depicted in Figure 3.1. There, events are illustrated by circles labeled with their corresponding action names; dashed lines indicate conflicts between events and for each bundle $X \mapsto e$ we draw one arrow from all events in $X$ to $e$.

A bundle event structures has a symmetric conflict relation. Therefore, it is not clear how disruption, for example $a.b [> c$, can be modeled as a bes, since disruption is not a symmetrical property. Therefore, [125] introduces extended bundle event structures, where the conflict relation does not need to be symmetric.

**Definition 3.2 (Extended Bundle Event Structure)** An extended bundle event structure, ebes for short, $\mathcal{E} = (E, \sim, \mapsto, l)$ is an element of $\mathcal{P}(U) \times \mathcal{P}(U \times U) \times \mathcal{P}(\mathcal{P}(U) \times U) \times (U \rightarrow \text{Act})$ such that

- $\sim \subseteq (E \times E)$ and $\sim$ is irreflexive
- $\mapsto \subseteq \mathcal{P}(E) \times E$
- $\text{dom}(l) = E$
- $\forall X \subseteq E, e \in E : X \mapsto e \Rightarrow (\forall e', e'' \in X : e' \neq e'' \Rightarrow e' \sim e'')$

Let $\text{EBES}$ denote the set of all extended bundle event structures.
3.3. DENOTATIONAL TRUE CONCURRENCY SEMANTICS FOR \( \text{PA}_{\text{SR}} \)

\[ \sim, \imply \] is called the (irreflexive) asymmetric conflict relation. Hereafter, we consider \( \mathcal{E} \) to be \( (E, \sim, \imply, l) \), \( \mathcal{E}_i \) to be \( (E_i, \sim_i, \imply_i, l_i) \) and in general \( \mathcal{E} \) to be \( (E_\mathcal{E}, \sim_\mathcal{E}, \imply_\mathcal{E}, l_\mathcal{E}) \).

The intuitive meaning of \( e' \sim e \) is that the execution of \( e \) disables \( e' \) forever, but not vice versa. Furthermore, an event can not be in conflict with itself, which is expressed by the irreflexivity of \( \sim \). In [125] the expression \( (a.b \triangleright c); d \) is modeled by the ebes shown in Figure 3.2. There the conflicts are depicted as dashed arrows or depicted as dashed lines if they are symmetrical.

**Remark 3.3** The bundle stability constraint of extended bundles event structures are dropped to obtain more general class of event structures. The event structures obtained are called dual event structures [116]. They allow causal ambiguity, which is examined in [127].

The standard order for (extended) bundle event structures is introduced as follows.

**Definition 3.4 (Restriction of a ebes)** Suppose \( \mathcal{E} \) is an extended bundle event structure and \( E' \subseteq E \). Then the restriction of \( \mathcal{E} \) to \( E' \), denoted by \( \mathcal{E} \upharpoonright E' \), is \( (E', \sim \cap (E' \times E'), \imply', l \upharpoonright E') \) where \( \imply' = \{ (X \cap E', e) \mid e \in E' \land X \imply e \} \).

**Remark 3.5** A restriction of a restricted ebes is equal to the restriction of that ebes, i.e. if \( \mathcal{E} = (E, \sim, \imply, l) \) is an ebes and \( E_1 \subseteq E_2 \subseteq E \), then \( \mathcal{E} \upharpoonright E_2 \upharpoonright E_1 = \mathcal{E} \upharpoonright E_1 \).

**Definition 3.6 (Order on EBES)** Let \( \mathcal{E}_i \in \text{EBES} \). Then \( \mathcal{E}_1 \preceq \mathcal{E}_2 \) if and only if \( E_1 \subseteq E_2 \) and \( \mathcal{E}_1 = \mathcal{E}_2 \upharpoonright E_1 \).

**Remark 3.7** Two different ebes which are comparable with respect to \( \preceq \) must have different sets of events, i.e. \( \mathcal{E}_1 \preceq \mathcal{E}_2 \land E_2 \subseteq E_1 \) implies \( \mathcal{E}_1 = \mathcal{E}_2 \).

Langerak [125] constructs the following minimal upper bound of an \( \omega \)-chain.

**Definition 3.8** Let \( (\mathcal{E}_i)_{i \in \mathbb{N}} \), where \( \mathcal{E}_i = (E_i, \sim_i, \imply_i, l_i) \), be an \( \omega \)-chain with respect to \( \preceq \). Then define \( \bigcup_i \mathcal{E}_i \) to be the ebes \( (\bigcup_i E_i; \bigcup_i \sim_i; \bigcup_i \imply_i; \bigcup_i l_i) \) where \( \imply = \{ (X, e) \mid \forall k : e \in E_k \Rightarrow (X \cap E_k) \imply_k e \} \).
CHAPTER 3. STANDARD ACTION REFINEMENT

Non-Completeness of \(\langle \text{EBES}, \preceq \rangle\).

\(\langle \text{EBES}, \preceq \rangle\) does not yield a cpo, as there are \(\omega\)-chains with more than one minimal upper bound. This situation can arise, for example, if \((X \cap E_k)_{k \in \mathbb{N}}\) is strictly increasing and each \((X \cap E_k, e)\) is a bundle of \(E_i\) whenever \(i \geq k\). We illustrate this by an example, where we consider \(\mathbb{N}\) to be a subset of \(U\) by identifying \(\star \)\(\cdots\) with \(n \in \mathbb{N}\).

Let \(\hat{E}_n\) be the ebes where the set of events consists of the elements of \(M_n = \{1, \ldots, n\}\) and an additional element \(e\). Furthermore, the bundles of \(e\) are all subsets of \(M\) whenever \(i \geq k\). We illustrate this by an example, where we consider \(\mathbb{N}\) to be a subset of \(U\) by identifying \(\star \)\(\cdots\) with \(n \in \mathbb{N}\).

\[
\hat{E}_n = (M_n \cup \{e\}, M_n \times M_n \setminus \text{Id}, \{(X, e) \mid X \in \mathcal{P}(M_n)\}, \text{cons}_a)
\]

for \(n \in \mathbb{N}\), where \(\text{Id}\) denotes the identity relation.

It is obvious that \((\hat{E}_n)_{n \in \mathbb{N}}\) is a chain in \(\langle \text{EBES}, \preceq \rangle\), hence \(\bigsqcup_n \hat{E}_n\) is a minimal upper bound of \((\hat{E}_n)_{n \in \mathbb{N}}\) with respect to \(\preceq\). From Definition 3.8 we get

\[
\bigsqcup_n \hat{E}_n = (\mathbb{N} \cup \{e\}, \mathbb{N} \times \mathbb{N} \setminus \text{Id}, \{(X, e) \mid X \in \mathcal{P}(\mathbb{N})\}, \text{cons}_a).
\]

In words, any subset of the natural numbers combined with \(e\) is a bundle in \(\bigsqcup_n \hat{E}_n\). If we restrict the bundles to the finite subsets of the natural numbers, i.e.

\[
\hat{E}^\text{fin}_n = (\mathbb{N} \cup \{e\}, \mathbb{N} \times \mathbb{N} \setminus \text{Id}, \{(X, e) \mid X \in \mathcal{P}_\text{fin}(\mathbb{N})\}, \text{cons}_a),
\]

we also get a minimal upper bound of \((\hat{E}_n)_{n \in \mathbb{N}}\) with respect to \(\preceq\). Furthermore, \(\hat{E}^\text{fin}_n\) and \(\bigsqcup_n \hat{E}_n\) are incomparable with respect to \(\preceq\) by Remark 3.7. Hence, the \(\omega\)-chain \((\hat{E}_n)_{n \in \mathbb{N}}\) does not have a least upper bound and therefore \(\langle \text{EBES}, \preceq \rangle\) fails to be a cpo.

The above counterexample is also a counterexample for the non-completeness of the class of bundle event structures under the given partial order, since every event structure in the example is a bundle event structure.

3.3.2 Closed Bundle Event Structures (CBES)

We want to restrict the ebes we have just considered in such a way that only one of the minimal upper bounds from the counterexample of Subsection 3.3.1 is allowed. Therefore, we only take those ebes into account which are closed under some special kind of finite approximation and thus rule out \(\hat{E}^\text{fin}_n\). We define this closedness property by using the results of approximation closedness introduced in Section 2.4.

**Definition 3.9 (Closed Bundle Event Structure)** A closed bundle event structure (cbes) \(\mathcal{E} = (E, \sim, \rightharpoonup, l)\) is an element of \(\mathcal{P}(U) \times \mathcal{P}(U \times U) \times \mathcal{P}(\mathcal{P}(U) \times U) \times (U \rightarrow \text{Act}_\vee)\) such that

- \(\sim \subseteq (E \times E)\) and \(\forall e \in E : \neg(e \sim e)\)
- \(\rightharpoonup \subseteq \mathcal{P}(E) \times E\)
- \(\text{dom}(l) = E\)
3.3. DENOTATIONAL TRUE CONCURRENCY SEMANTICS FOR $\text{PA}_{\text{SR}}$

- $\forall X \subseteq E, e \in E : X \mapsto e \Rightarrow (\forall e', e'' \in X : e' \neq e'' \Rightarrow e' \sim e'')$
- $\forall e \in E : e \mapsto e$ is approximation closed with respect to $E$

Let $\text{CBES}$ denote the set of all closed bundle event structures.

Since every cbes is also an ebes (only a further constraint is added), we have $\text{CBES} \subset \text{EBES}$. Furthermore, every ebes that has a finite set of events satisfies the closedness condition, and therefore is an element of $\text{CBES}$. Hence, the ebes shown in Figure 3.2 is also a cbes. Furthermore, $\bigcup_n \hat{E}_n$ from the counterexample of Subsection 3.3.1 is a cbes, whereas $\hat{E}^{\text{fin}}$ is not, which follows from Example 2.14.

**Theorem 3.10 (Complete Partial Order)** The ordered set $(\text{CBES}, \preceq)$ is an $\omega$-complete partial order, where $\bigcup_n \hat{E}_n$ from Definition 3.8 is the least upper bound.

**Proof:** The proof is given in Subsection 3.6.1. $\Box$

**Remark 3.11** The set of bundle event structures can be restricted in the same way as $\text{EBES}$ to obtain a cpo.

**Remark 3.12** There is another possibility of defining an order on $\text{EBES}$ that yields a complete partial order: An ebes is smaller than another one if it has less events but more bundles ($\{(X \cap E_1, e) \mid X \mapsto e \land e \in E_1 \} \subseteq \mapsto_1$), i.e. events in the greater ebes can be enabled earlier as in the smaller one. There, the ebes do not have to satisfy the approximation closedness constraints to yield a complete partial order, i.e. $\text{EBES}$ with this order yields a cpo. But not all standard operators (compare Subsection 3.3.3) are continuous with respect to this order.

**Transition system from a cbes.**

Here, we describe how to obtain a transition system from a cbes, which is later used to establish a consistency result for the denotational and the operational semantics. First, we specify the initial events, i.e. those events which do not have a causal constraint, then we specify the events that correspond to termination.

**Definition 3.13** The set of initial events of cbes $\mathcal{E}$ is defined by

$$\text{init}(\mathcal{E}) = \{ e \in E \mid \neg(\exists X : X \mapsto e) \}.$$  

The set of successful termination events of cbes $\mathcal{E}$ is defined by

$$\text{exit}(\mathcal{E}) = \{ e \in E \mid l(e) = \sqrt{\_} \}.$$  

In order to obtain a transition system from a cbes, the remainder [20, 126, 131] of a cbes with respect to an initial event is defined. The remainder denotes the event structure after the execution of this initial event. Remainders are used to obtain a transition relation for $\text{CBES}$. 
Definition 3.14 (Remainder of a cbes) Let $E \in \text{CBES}$ and $e \in \text{init}(E)$. Then the remainder $E_{[e]}$ is given by $(E', \sim', \mapsto', l')$ where

\begin{align*}
E' &= \{ e' \in E \mid e' \neq e \land \neg(e' \rightarrow e) \} \\
\sim' &= \sim \cap (E' \times E') \\
\mapsto' &= \{(X \cap E', e') \mid e' \in E' \land X \mapsto e' \land e \notin X \} \\
l' &= l \upharpoonright E'
\end{align*}

Lemma 3.15 Let $E \in \text{CBES}$ and $e \in \text{init}(E)$. Then $E_{[e]} \in \text{CBES}$.

Proof: Define $\tilde{E} = (E, \sim, \{(X, \tilde{e}) \in \mapsto | e \notin X\}, l)$. We will show that $\tilde{E} \in \text{CBES}$. Let $X \subseteq E$ and $(E_n)_{n \in \mathbb{N}}$ be a finite, monotone approximation of $E$ such that $\forall n : \exists X_n : X_n \mapsto \tilde{e} \land X \cap E_n = X_n \cap E_n$. Then there is $m \in \mathbb{N}$ such that $e \in E_m$. Thus, $e \notin X$. Furthermore, we have $X \mapsto \tilde{e}$ by the approximation closedness condition of $E$. Hence, $X \mapsto \tilde{e}$.

It is easy to check that $E_{[e]} = \tilde{E} \upharpoonright E_{[e]}$. And so the result follows by Lemma 3.27.

Definition 3.16 The transition relation $\mapsto \subseteq \text{CBES} \times \text{Act} \times \text{CBES}$ is defined by $\mapsto = \{(E, l(e), E_{[e]}) \mid E \in \text{CBES} \land e \in \text{init}(E)\}$.

An example of a transition system obtained from $\mapsto$ is given in Figure 3.3.

### 3.3.3 Operators on CBES

Here, we present the operators on CBES that are later used to define the denotational semantics. They are taken from [125] except for some slight modifications. For example, we model the disjoint union directly and we introduce more conflicts. This is done in order to obtain...
3.3. DENOTATIONAL TRUE CONCURRENCY SEMANTICS FOR PA_{SR}

a closer connection between the operational and the denotational semantics. The refinement operator, which does not appear in [125], is an adapted version of [79, 133].

**Definition 3.17 (Operators on \( E \))** Let \( A \subseteq \text{Obs} \). Then define

\[ \hat{\cdot} : (\text{Obs} \cup \{\tau\}) \times \text{CBES} \to \text{CBES} \] with \( a \hat{\cdot} E = (\{\bullet\} \cup (\{\star\} \times E), \sim, \i, \bar{l}) \) where

\[
\begin{align*}
\sim & = \{((\star_1, e), (\star_1, e')) | (e, e') \in \sim\} \\
\i & = \{((\{\star\} \times \text{init}(E)), (\star_2, e)) | e \in \text{init}(E)\} \\
\bar{l}(e) & = \begin{cases} l(e) & \text{if } e = (\star_1, e) \\
\text{a} & \text{if } e = \bullet
\end{cases}
\]

\[ \hat{+} : \text{CBES} \times \text{CBES} \to \text{CBES} \] with \( E_1 \hat{+} E_2 = (\tilde{E}, \tilde{\sim}, \tilde{i}, \bar{l}) \) where

\[
\tilde{E} = (\{\star\} \times E_1) \cup (\{\star\} \times E_2) \\
\tilde{\sim} = \{((\star_1, e), (\star_1, e')) | i \neq j \wedge e_j \in \text{init}(E_j)\} \\
\tilde{i} = \{((\{\star\} \times \text{init}(E_1)), (\star_2, e)) | e \in \text{init}(E_2)\}
\]

\[ \hat{?} : \text{CBES} \times \text{CBES} \to \text{CBES} \] with \( E_1 \hat{?} E_2 = (\tilde{E}, \tilde{\sim}, \tilde{i}, \bar{l}) \) where

\[
\tilde{E} = (\{\star\} \times E_1) \cup (\{\star\} \times E_2) \\
\tilde{\sim} = \{((\star_1, e), (\star_1, e')) | i \wedge e_i \wedge e_i' \in \text{exit}(E_i)\} \\
\tilde{i} = \{((\{\star\} \times \text{exit}(E_1)), (\star_2, e)) | e \in \text{exit}(E_2)\}
\]

\[ \hat{>}: \text{CBES} \times \text{CBES} \to \text{CBES} \] with \( E_1 \hat{>} E_2 = (\tilde{E}, \tilde{\sim}, \tilde{i}, \bar{l}) \) where

\[
\tilde{E} = (\{\star\} \times E_1) \cup (\{\star\} \times E_2) \\
\tilde{\sim} = \{((\star_1, e), (\star_1, e')) | e \wedge e_i' \wedge e_i' \in \text{exit}(E_i)\} \\
\tilde{i} = \{((\{\star\} \times \text{exit}(E_1)), (\star_2, e)) | e \in \text{exit}(E_2)\}
\]

\[ \hat{\|}_A : \text{CBES} \times \text{CBES} \to \text{CBES} \] with \( E_1 \hat{\|}_A E_2 = (\tilde{E}, \tilde{\sim}, \tilde{i}, \bar{l}) \) where

\[
\begin{align*}
\tilde{E} & = (E^f_1 \times \{\star\}) \cup (\{\star\} \times E^f_2) \cup E^s \\
E^f_1 & = \{e \in E_1 | l_1(e) \notin A \cup \{\sqrt{\cdot}\}\} \\
E^f_2 & = \{(e_1, e_2) \in E_1 \times E_2 | l_1(e_1) = l_2(e_2) \in A \cup \{\sqrt{\cdot}\}\} \\
\sim & = \{((e_1, e_2), (e'_1, e'_2)) | e_1 \sim e'_1 \wedge e_2 \sim e'_2 \wedge (e_1 = e'_1 \neq \star \wedge e_2 = e'_2 \neq \star \wedge e_1 \neq e'_2)\} \\
\i & = \{((e'_1, e'_2) \in \tilde{E} | e'_1 \in X_i), (e_1, e_2)) | X_i \MAP e_i\}
\bar{l}(e_1, e_2) = \begin{cases} l_1(e_1) & \text{if } e_2 = \star \\
l_2(e_2) & \text{otherwise}
\end{cases}
\]
\[
\begin{aligned}
\begin{array}{c}
\varepsilon^+ \\
\otimes
\end{array}
\quad
\begin{array}{c}
\varepsilon_{12} \\
\otimes
\end{array}
\quad
\begin{array}{c}
\text{Ref}^*_a(\varepsilon^+, (a \to \varepsilon_{12})) \\
\otimes
\end{array}
\end{aligned}
\]

Figure 3.4: Illustration of the \(\text{Ref}^*\) Operator

\[
\text{Ref}^*_a : \text{CBES} \times (A \to \text{CBES}) \to \text{CBES} \text{ with } \text{Ref}^*_a(\varepsilon, \theta) = (\bar{E}, \bar{\sim}, \bar{\rightarrow}, \bar{l}) \text{ where}
\]

\[
\bar{E} = \{(e, \hat{e}) \mid e \in E \land l(e) \in A \land \hat{e} \notin E_{\theta(l(e))}\} \cup \{(e, \hat{e}) \in E \times E \mid l(e) \notin A\}
\]

\[
\bar{\sim} = \{((e_1, \hat{e}_1), (e_2, \hat{e}_2)) \mid e_1 \sim e_2 \lor (e_1 = e_2 \land l(e_1) \in A \land \hat{e}_1 \neq \hat{e}_2 \land \hat{e}_1 \sim_{\theta(l(e_1))} \hat{e}_2 \land \hat{e}_2 \in \text{exit}(\theta(l(e_1))))\}
\]

\[
\bar{\rightarrow} = \{\{\theta \times X', (e, \hat{e})\} \mid l(e) \in A \land X' \Rightarrow_{\theta(l(e))} \hat{e}\} \cup \{(X, (e, \hat{e})) \mid \exists X : X \Rightarrow e \land (l(e) \in A \Rightarrow \hat{e} \notin \text{init}(\theta(l(e))))\} \land \hat{X} = \{(e', \hat{e}') \in \bar{E} \mid e' \in X \land (l(e') \in A \Rightarrow \hat{e}' \in \text{exit}(\theta(l(e'))))\}
\]

\[
\bar{l}((e, \hat{e})) = \begin{cases} 
\text{init}(\theta(l(e))) & \text{if } l(e) \notin A \\
\text{init}(\theta(l(e)))(\hat{e}) & \text{if } l(e) \in A \land \text{init}(\theta(l(e)))(\hat{e}) \neq \sqrt{1} \\
\sqrt{1} & \text{if } l(e) \in A \land \text{init}(\theta(l(e)))(\hat{e}) = \sqrt{1}
\end{cases}
\]

\[
\hat{A} : \text{CBES} \to \text{CBES} \text{ with } \varepsilon_{\hat{A}} = (E, \bar{\sim}, \bar{\rightarrow}, \bar{l}) \text{ where}
\]

\[
\bar{l}(e) = \begin{cases} 
\text{init}(\theta(l(e))) & \text{if } l(e) \notin A \\
\sqrt{1} & \text{if } l(e) \in A
\end{cases}
\]

A small example that illustrates how the refinement operator \(\text{Ref}^*\) behaves is given in Figure 3.4. There, \((a \to \varepsilon_{12})\) denotes the function from \{a\} to CBES that maps \(a\) to \(\varepsilon_{12}\).

**Proposition 3.18** All operators of Definition 3.17 are well defined, i.e. they really yield elements of CBES. Moreover all operators from Definition 3.17 are continuous with respect to \(\sqsubseteq\).

**Proof:** The proof is given in Subsection 3.6.2. \(\square\)

### 3.3.4 Denotational Meaning for \(\text{PA}_{sr}\)

First, we define the denotational semantics of expressions (\(\text{EXP}_{sr}\)) with respect to variable assignments, i.e. functions from \(\text{Var}\) to CBES. Then variable assignments are derived from declarations, which are used to define the denotational semantics of processes (\(\text{PA}_{sr}\)).
3.3. Denotational True Concurrency Semantics for $PA_{SR}$

Definition 3.19  Let $\llbracket \cdot \rrbracket_\rho : \text{EXP}_{sr} \times (\text{Var} \rightarrow \text{CBES}) \rightarrow \text{CBES}$ be defined as follows (where $\rho : \text{Var} \rightarrow \text{CBES}$)

\[
\begin{align*}
\llbracket 0 \rrbracket_\rho &= (\emptyset, \emptyset, \emptyset, \emptyset) \\
\llbracket a \cdot B \rrbracket_\rho &= a^\triangledown \llbracket B \rrbracket_\rho \\
\llbracket B_1 + B_2 \rrbracket_\rho &= \llbracket B_1 \rrbracket_\rho \uplus \llbracket B_2 \rrbracket_\rho \\
\llbracket B_1 \parallel B_2 \rrbracket_\rho &= \llbracket B_1 \rrbracket_\rho \parallel \llbracket B_2 \rrbracket_\rho \\
\llbracket B[(a \rightarrow B_a)^{\alpha \varepsilon A}] \rrbracket_\rho &= \text{Ref}_A^\alpha([B]_\rho, (a \rightarrow [B_a]_\rho)^{\alpha \varepsilon A}) \\
\llbracket B \setminus A \rrbracket_\rho &= [B]_\rho \setminus A \\
\llbracket x \rrbracket_\rho &= \rho(x)
\end{align*}
\]

Examples of how $\llbracket \cdot \rrbracket_\rho$ behaves are given in Figure 3.5.

To apply the cpo theory we need the following lemma.

Lemma 3.20  $\llbracket B \rrbracket_\rho$ is continuous for every $B \in \text{EXP}_{sr}$.

Proof:  By structural induction. We only present the case $B_1 + B_2$.

\[
\begin{align*}
\llbracket B_1 + B_2 \rrbracket_{\parallel, \rho_i} &= \hat{\lor}(\llbracket B_1 \rrbracket_{\parallel, \rho_i}, [B_2]_{\parallel, \rho_i}) \\
&= \hat{\lor}(\biguplus_i \llbracket B_1 \rrbracket_{\rho_i}, \biguplus_i [B_2]_{\rho_i}) \\
&= \hat{\lor}(\biguplus_i ([B_1]_{\rho_i} \parallel [B_2]_{\rho_i})) \\
&= \biguplus_i \hat{\lor}([B_1]_{\rho_i}, [B_2]_{\rho_i}) \\
&= \biguplus_i [B_1 + B_2]_{\rho_i}
\end{align*}
\]

The other cases follow analogously.  \qed

Now we are ready to give the meaning of a declaration.

Suppose decl : Var \rightarrow EXP_{sr}. Then define $\mathcal{F}_{\text{decl}} : (\text{Var} \rightarrow \text{CBES}) \rightarrow (\text{Var} \rightarrow \text{CBES})$ with $\mathcal{F}_{\text{decl}}(\rho)(x) = \llbracket \text{decl}(x) \rrbracket_\rho$. From Lemma 3.20 it follows that $\mathcal{F}_{\text{decl}}$ is continuous. Therefore, from Theorem 2.11 we get $\llbracket \cdot \rrbracket : (\text{Var} \rightarrow EXP_{sr}) \rightarrow (\text{Var} \rightarrow \text{CBES})$ with $\llbracket \text{decl} \rrbracket = \text{fix}(\mathcal{F}_{\text{decl}}) = \biguplus_n \mathcal{F}_{\text{decl}}^n(\bot)$ is well defined.

We define the denotation of a process as follows.
Definition 3.21 (Denotational Semantics)
Define \( [[\cdot]] : \text{PA}_{sr} \to \text{CBES} \) by \( [[\langle \text{decl}, B \rangle]] = [[B]]_{[\text{decl}]} \).

3.4 Operational Semantics for \( \text{PA}_{sr} \)

Operational semantics that coincide with the denotational semantics have been given for process algebras with action refinement in [70, 98, 157]. Operational semantics for the ST-approach\(^2\), which can be straightforwardly adjusted to operational semantics for action refinement, have been examined in [44, 49, 98, 102]. The papers based on the ST-approach do not explicitly examine consistency between denotational and operational semantics. The underlying languages used in the cited papers do not contain disruption. Moreover, most of the operational semantics defined in these papers are inappropriate for languages that contain disruption like our language does. This is argued as follows.

In [44, 49, 70], a refinement expression \( B[a \to Q] \) is modeled by invoking \( Q \) in parallel with the remaining process whenever \( a \) is activated in \( B \). This leads to problems if a disrupt operator is involved. For example, action \( a \) can be disrupted during its execution. But in the above approach \( Q \) is executed, even though \( a \) is disrupted.

On the other hand, in [98, 157] the execution of an action is only allowed if all active actions are still executable afterwards. This is not a reasonable approach if a language with a disrupt operator is considered, since this operator can remove active actions in a reasonable way.

The operational semantics mentioned above have to extend the syntax of the underlying languages in order to define the operational semantics for refinement: Further syntactical operators have to be added to the languages in [44, 49, 70]. And [98, 157] extends the syntactical expressions by moving to an event-based language, which leads to a very discriminating theory.

We will present a new approach to an operational semantics that corresponds to the denotational semantics. This approach can handle the disrupt operator and, moreover, no extension of the syntactical expressions is necessary.

The operational semantics is defined by using transition rules that simulate the execution of a refined action within the refinement construction. For example, the process \( (a.B)[a \to a_1.a_2] \) evolves into \( (a.B)[a \to a_2] \) by executing \( a_1 \) provided \( a \) does not occur in \( B \). If \( a \) occurs in \( B \) as in \( a.a \) and if we proceed in the same way, we lose the information to which term the \( a_s \) occurring in \( B \) have to be refined, i.e. \( (a.a)[a \to a_1.a_2] \) must not evolve to \( (a.a)[a \to a_2] \) by executing \( a_1 \). To circumvent this problem, we rename the executed action \( a \) by a fresh action name \( a' \) and extend the refinement by an additional refinement, i.e. \( (a.a)[a \to a_1.a_2] \) evolves into \( (a'.a)[(a \to a_1,a_2),(a' \to a_2)] \) by executing \( a_1 \), where \( a' \neq a \). This is still not sufficient, since we have to trigger the choice when an action is renamed in the choice operator, i.e. \( (b + (a.B))[a \to a_1.a_2] \) evolves into \( (a'.B)[(a \to a_1.a_2),(a' \to a_2)] \) by executing \( a_1 \). Otherwise, \( (b + (a.B))[a \to a_1.a_2] \) would be able to execute \( b \) after \( a_1 \). This is counterintuitive, and therefore the choice has to be triggered. We model the renaming of the action by extending the transition relation using additional transition labels. The choices are also triggered by those additional transition labels.

\(^2\)In the ST-approach, the execution of an action is split into the two different events of the start and the ending (termination) of an action, where the ending is uniquely related to its start.
Before we introduce the transition rules for EXP<sub>sr</sub>, we have to determine all actions which occur in a given expression. We need this in order to determine a fresh action name for the renaming of an action. Furthermore, we need \(|\text{Obs}| > |\mathbb{N}|\) to guarantee the existence of a fresh action name.

**Definition 3.22** The function \(\tilde{\mathcal{L}} : \text{EXP}_{\text{sr}} \to \mathcal{P}_{\text{count}}(\text{Obs})\) is defined as follows.

\[
\begin{align*}
\tilde{\mathcal{L}}(0) &= \tilde{\mathcal{L}}(1) = \tilde{\mathcal{L}}(x) = \emptyset & \tilde{\mathcal{L}}(a, B) &= \{a\} \cup \tilde{\mathcal{L}}(B) \\
\tilde{\mathcal{L}}(\tau, B) &= \tilde{\mathcal{L}}(B) & \tilde{\mathcal{L}}(B_1 \parallel A B_2) &= \tilde{\mathcal{L}}(B_1) \cup \tilde{\mathcal{L}}(B_2) \cup A \\
\tilde{\mathcal{L}}(B_1 + B_2) &= \tilde{\mathcal{L}}(B_1; B_2) = \tilde{\mathcal{L}}(B_1 \triangleright B_2) = \tilde{\mathcal{L}}(B_1) \cup \tilde{\mathcal{L}}(B_2) \\
\tilde{\mathcal{L}}(B \backslash A) &= \tilde{\mathcal{L}}(B) \cup A & \tilde{\mathcal{L}}(B[(a \rightarrow b)_{a \in A}]) &= \tilde{\mathcal{L}}(B) \cup A \cup \bigcup_{a \in A} \tilde{\mathcal{L}}(B_a)
\end{align*}
\]

Then, define \(\mathcal{L} : \text{PA}_{\text{sr}} \to \mathcal{P}_{\text{count}}(\text{Obs})\) by \(\mathcal{L}((\text{decl}, B)) = \tilde{\mathcal{L}}(B) \cup \bigcup_{x \in \text{Var}} \tilde{\mathcal{L}}(\text{decl}(x))\).

The set \(A\) has to be added in the cases of the parallel operator and the hiding operator, since otherwise confusion may arise. For example, if action \(a\) is renamed \(b\) in \(a\|_{\{a\}} 1\), we obtain a change in the behavior, since the process is deadlocked after the renaming.

The transition rules of \(\longrightarrow^*_{\text{decl}} \subseteq \text{EXP}_{\text{sr}} \times (\mathcal{A}c\triangleright \cup (\mathcal{O}bs \times \mathcal{O}bs)) \times \text{EXP}_{\text{sr}}\) are presented in Table 3.1, where \(\gamma\) is an element of \(\mathcal{A}c\triangleright \cup (\mathcal{O}bs \times \mathcal{O}bs)\). We write \(\longrightarrow^*\) if \(\text{decl}\) is clear from the context. Here \(\frac{(a,b)}{(s)}\) means that one executable action \(a\) is renamed \(b\) and that all choices which would be triggered by the execution of this \(a\) are taken. \(\frac{(a,b)}{(s)}\) can be interpreted that action \(a\) has been started and that this action \(a\) will finish by executing action \(b\).

In rule \(A_2\), action \(a\) is relabeled by \(b\). Rule \(C\) makes no difference between labels from \(\mathcal{A}c\triangleright\) and \((\mathcal{O}bs \times \mathcal{O}bs)\). Hence, in both cases the choice is triggered.

The disrupt operator \(\triangleright\) also disrupts the process if an \((a, b)\) transition takes place, since in this chapter the start of an action triggers the disruption.

The parallel operator works asynchronously exactly for those actions which are not in \(A \cup \{\sqrt{}\}\). For synchronization purposes, \((a, b)\) is regarded to be in \(A\) if and only if \(a \in A\). The rule for synchronization is divided into \(P_2\) and \(P_3\), since in the case where \((a, b)\) is executed, we have to take care that further derivations will synchronize on \(b\).

Rule \(Ref_1\) considers the case when an action is executed which is not refined. In this case, only the term which is refined is modified. Rule \(Ref_2\) considers the case when an action \(\hat{a}\) which is refined by a non-terminating process is executed. Here \(\hat{a}\) is renamed with a fresh action name, which is ensured by \(b \notin A \cup \mathcal{L}((\text{decl}, B))\). Such a \(b\) always exists, since \(|\text{Obs}| > |\mathbb{N}|\). Furthermore, we keep all present refinements and add a new refinement for the executed action \(\hat{a}\), which is now labeled with \(b\). The case when the refined process terminates is considered in rule \(Ref_3\). In this case, \(\hat{a}\) terminates. Therefore, it has to be removed, which is done by taking the transition labeled with \(\hat{a}\). The refinement remains unaffected.

The remaining rules are the standard ones [32].

Rules \(P_1, P_3, Ref_1, Res_1\) sometimes derive processes with undesired behavior by executing \((a, b)\). For example, \(a \|_{\{a\}} 1 \xrightarrow{(a,b)} b\|_{\{b\}} 1\), which deadlocks. The undesired behavior can only appear if action \(b\) occurs in the considered process. Such undesired behavior does not cause any problems, since we are only interested in transitions labeled with elements of \(\mathcal{A}c\triangleright\). The only situation where we need \((a, b)\) transitions is in rule \(Ref_2\), but there we take care that \(b\) is fresh.
Example 3.23 We illustrate the transition rules by presenting a derivation path of the process \((a \| (a + b.0)) [a \rightarrow c][c \rightarrow d]\) in Figure 3.6. In the first step, action \(a\) is renamed \(c\). This seems to be strange, since \(c\) will be refined to \(d\). But it does not cause a problem, because the operational semantics refines \(c\) to \(c'\) before it gets under the influence of \([c \rightarrow d]\). The choice is triggered by the renaming transition as it is illustrated by the first derivation step.
3.4. OPERATIONAL SEMANTICS FOR PA_{SR}  

\[
\begin{align*}
(a \parallel (a + b.0)) & \vdash [a \rightarrow c][c \rightarrow d] \\
(c \parallel (a,c)) & \vdash [a \rightarrow c, c \rightarrow c'][c \rightarrow d, c' \rightarrow 1] \\
(c'' \parallel (a,c,c')) & \vdash [a \rightarrow c, c \rightarrow c', c'' \rightarrow 1][c \rightarrow d, c' \rightarrow 1] \\
(1 \parallel (a,c,c')) & \vdash [a \rightarrow c, c \rightarrow c', c'' \rightarrow 1][c \rightarrow d, c' \rightarrow 1] \\
(0 \parallel (a,c,c')) & \vdash [a \rightarrow c, c \rightarrow c', c'' \rightarrow 1][c \rightarrow d, c' \rightarrow 1]
\end{align*}
\]

Figure 3.6: Example of a Process Derivation with respect to \(\longrightarrow_{\text{decl}}^{s}\)

Definition 3.24 (Operational Semantics) The operational semantics \(\mathcal{O}^{s}: \text{PA}_{sr} \rightarrow \text{TS}\) is given by \(\mathcal{O}^{s}((\text{decl}, B)) = (\text{EXP}_{sr}, \text{Act}_{\sqrt{\cdot}}, \longrightarrow_{\mathcal{O}^{s}}, B)\) where \(\longrightarrow_{\mathcal{O}^{s}} = \longrightarrow_{\text{decl}}^{s} \cap (\text{EXP}_{sr} \times \text{Act}_{\sqrt{\cdot}} \times \text{EXP}_{sr})\).

We still have to argue that the presented operational semantics is reasonable. The fact that the transitions labeled with elements of \(\text{Obs} \times \text{Obs}\) always generate an infinite transition system is problematic. More precisely, if process \(B\) can execute \(a \in \text{Obs}\) then it can execute \((a, b)\) for every \(b \in \text{Obs}\), which yields a transition system with uncountably infinite branches. Moreover, it is also possible to derive undesirable infinite derivations, for example \(a_{1}.1 \longrightarrow_{\mathcal{O}^{s}} \cdots (a_{i}.a_{i+1}) \cdots\). Nevertheless, this does not cause any problem since only the transitions labeled with elements of \(\text{Act}_{\sqrt{\cdot}}\) are used to define the operational semantics (Definition 3.24).

But we still have an infinitely branching transition system by rule \(\text{Ref}_{2}\), for example \(a[a \rightarrow a_{1}.a_{2}] \longrightarrow a_{1}b[a \rightarrow a_{1}.a_{2}, b \rightarrow a_{2}]\) for all \(b \in \text{Obs}\setminus\{a\}\). This is also no problem, since all the expressions are \(\alpha\)-equivalent [22], i.e. they can be translated into each other by action renaming of the bound actions. In other words, in rule \(\text{Ref}_{2}\), it is not important which \(b \notin A \cup \mathbb{L}((\text{decl}, B))\) is chosen, since all choices yield \(\alpha\)-equivalent expressions.

It is also possible to apply techniques, similar to [44], in order to obtain unique action renaming, which makes \(\alpha\)-conversion obsolete. This technique is used in Chapter 7 to obtain an operational semantics (Section 7.4).

Our operational semantics is a meaningful semantics with respect to the denotational semantics, since the transition system derived from the denotational semantics is bisimilar to the operational semantics.

Theorem 3.25 (Consistency) Let \((\text{decl}, B) \in \text{PA}_{sr}\). Then the transition systems \(\mathcal{O}^{s}((\text{decl}, B))\) and \((\text{CBES}, \text{Act}_{\sqrt{\cdot}}, \longrightarrow, [\langle\text{decl}, B\rangle])\) are bisimilar.

Proof: The proof is given in Subsection 3.6.3.
3.4.1 Modified Operational Semantics

The operational semantics we have just presented yields unnecessarily long terms, since we have to copy each refinement, even though it is used only once. Consider, for example, the process $(a.B')[a \rightarrow a_1.a_2.\cdots a_n.1]$. Then we have $(a.B')[a \rightarrow a_1.a_2.\cdots a_n.1] [a_1, \ldots a_i] (a^{(i)}.B'')[(a \rightarrow a_1.a_2.\cdots a_n.1), (a^{(i)} \rightarrow a_2.\cdots a_n.1), \ldots, (a^{(i)} \rightarrow a_{i+1}.\cdots a_n.1)]$ for $i \leq n$ and suitable actions $a^{(i)}$. Here, the refinement of $a^{(j)} \rightarrow a_{j+1}.\cdots a_n.1$ is an unnecessary information for all $j < i$, since $a^{(j)}$ does not occur in $(a^{(i)}.B'')$.

Our operational semantics does not only yield unnecessarily long terms, it also yields infinite transition systems in cases where finite ones would be sufficient. For example, let $\text{decl}(X) = a.X$, then $X[a \rightarrow b] \left( \frac{b}{\tau} \right)^i X[a \rightarrow b, a^{(1)} \rightarrow 1, \ldots a^{(i)} \rightarrow 1]$. Therefore, we get an infinite transition system. But it is sufficient that $X[a \rightarrow b]$ evolves to $X[a \rightarrow b]$ by executing $(b\tau)^i$, since $X[a \rightarrow b]$ contains all necessary information.

To circumvent the unnecessary copy of the refinement, we divide $\text{Obs}$ into two parts. One part ($\text{Obs}_a$) is used for active actions, i.e. actions which have been renamed. The other part ($\text{Obs}_P$) is used for actions in the original expression, i.e. in the expression on which the transition rules were applied. We then know that an executable action of $\text{Obs}_a$ only appears ‘once’ in the process. More precisely, suppose $B$ only contains actions of $\text{Obs}_P$ and $B \overset{\gamma_1}{\cdots} \overset{\gamma_i}{\cdots} B \overset{b,c}{\rightarrow} B''$, where $b \in \text{Obs}_a$ and $c \notin L(\text{decl}, B')$. Then $B'[b \rightarrow \tilde{B}] \sim B''[c \rightarrow \tilde{B}]$. This gives us the advantage of keeping actions of $\text{Obs}_a$ unrenamed.

Furthermore, when an action of $\text{Obs}_a$ finishes, we remove it from those positions in the expressions in which they were inserted by rule $P_3$ and $\text{Ref}_2$. Therefore we obtain the advantage of generating more finite state transition systems. This can also be achieved when no $\alpha$-conversion is considered, for example by choosing always the ‘smallest’ action of $\text{Obs}_a$ that does not occur for action renaming.

The ideas mentioned above are formalized as follows. Let $\text{Obs}_P, \text{Obs}_a \subseteq \text{Obs}$ with $|\text{Obs}_P| = |\text{Obs}_a| = |N|$ and $\text{Obs}_P \cap \text{Obs}_a = \emptyset$. Then the modified transition rules are those of Table 3.1 where $A_2, P_2, \text{Ref}_2$ and $\text{Ref}_3$ are replaced by the transition rules presented in Table 3.2.

In rule $A_2^m$, it is now only possible to start (renaming) actions of $\text{Obs}_P$, since actions of $\text{Obs}_a$ are considered to be active, and therefore they can not be started again. Furthermore, the action $a$ in rule $A_2^m$ can only be replaced by an action of $\text{Obs}_a$, since $a$ becomes active.

Rule $P_2$ is modified by removing action $\gamma$ from the synchronization set if it is an active action. This is possible, since $\gamma$ terminates in this rule, and therefore it does not appear in $B'[a] B'_2$ when it is from $\text{Obs}_a$.

The rule $\text{Ref}_2$ is split into two rules. Rule $\text{Ref}_2^m$ considers the case when an action of $\text{Obs}_P$ is refined. In this case the rule stays the same. And rule $\text{Ref}_2^m$ considers the case when an active action is refined. Here we do not rename the action nor do we change the expression which gets refined at all, i.e. we do not change $B$. We only check that $\hat{a}$ is an initial action, which is done by $B \overset{\hat{a}}{\rightarrow} B'$. This is necessary for the soundness, since we have a disrupt operator, and so not every started action which has not terminated has to be active. The change in the refinement in rule $\text{Ref}_2^m$ is directly done at $\hat{a}$, i.e. $\hat{a}$ is refined by $B''$.

In rule $\text{Ref}_3^m$ we remove, similar to rule $P_2^m$, action $\gamma$ from the expression if it is an active action.
3.5. DISCUSSION

In this chapter, we have reproduced the standard action refinement approach (the choice is considered to be start-based) in order to give an introduction to the subject of action refinement. We had to define a subclass of (extended) bundle event structures, called closed bundle event structures, to yield a reasonable complete partial order on these kinds of event structures. Furthermore, we have presented a new technique to define an operational semantics on process algebras containing action refinement operators such that the operational and the denotational semantics are consistent. With this technique it is not necessary to extend the syntactical expressions of the process algebra in order to define the operational semantics.

In the following chapter, we take a first look on the end-based view. More precisely, we define an action refinement operator on closed bundle event structures that considers the conflict relation to be end-based triggered. We show that the start-based and the end-based views lead to different theories, for example the standard equivalences are not preserved by the end-based refinement operator. Therefore, we introduce new equivalences that are congruences for the end-based

Table 3.2: Modified Transition Rules for \( \rightarrow_{\text{decl}} \)

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A_{2m} )</td>
<td>( a \in \text{Obs}_P \quad b \in \text{Obs}_a \quad a.B \xrightarrow{(a,b)} b.B )</td>
</tr>
<tr>
<td>( P_{2m} )</td>
<td>( B_1 \xrightarrow{\gamma} B'_1 \quad B_2 \xrightarrow{\gamma} B'_2 \quad \gamma \in {\sqrt{} } \cup A \quad A' = A \setminus ({\gamma} \cap \text{Obs}_a) )</td>
</tr>
<tr>
<td>( \text{Ref}<em>{f</em>{2,1}} )</td>
<td>( B \xrightarrow{(\hat{a},b)} B' \quad \hat{a} \in A \cap \text{Obs}<em>P \quad b \notin A \cup \mathcal{L}(\langle \text{decl}, B \rangle) \quad B</em>{\hat{a}} \xrightarrow{\gamma} B'' \quad \gamma \neq \sqrt{} )</td>
</tr>
<tr>
<td>( \text{Ref}<em>{f</em>{2,2}} )</td>
<td>( B \xrightarrow{\hat{a}} B' \quad \hat{a} \in A \cap \text{Obs}<em>a \quad B</em>{\hat{a}} \xrightarrow{\gamma} B'' \quad \gamma \neq \sqrt{} )</td>
</tr>
<tr>
<td>( \text{Ref}<em>{f</em>{3}} )</td>
<td>( B \xrightarrow{\hat{a}} B' \quad \hat{a} \in A \quad B_{\hat{a}} \xrightarrow{\sqrt{}} B'' \quad A' = A \setminus {\hat{a}} \cap \text{Obs}_a )</td>
</tr>
</tbody>
</table>

The modified transition rules and the original transition rules yield bisimilar transitions systems:

**Theorem 3.26** Suppose \( \langle \text{decl}, B \rangle \in \mathcal{PA}_{sr} \) and \( \mathcal{L}(\langle \text{decl}, B \rangle) \subseteq \text{Obs}_P \).

Then \( O^*_m(\langle \text{decl}, B \rangle) \) and \( (\text{CBES}, \text{Act}, \rightarrow, [\langle \text{decl}, B \rangle]) \) are bisimilar, where \( O^*_m \) is derived as \( O^* \) except that the modified transition rules are used.

**Proof:** Similar to the proof of Theorem 3.25. \( \square \)
refinement operator. Finally, we argue that closed bundle event structures are not appropriate to be used for the end-based view.

3.6 Proofs

3.6.1 Proof of Theorem 3.10

First we show that CBES is closed under restriction.

Lemma 3.27 Let $\mathcal{E} \in \text{CBES}$ and $E' \subseteq E$. Then $\mathcal{E} \upharpoonright E' \in \text{CBES}$.

Proof: Is an immediate consequence of Corollary 2.18. \qed

The following lemma is later used to verify the uniqueness of the minimal upper bounds in CBES.

Lemma 3.28 Two cbes are equal if and only if they coincide on every finite restriction, i.e. $\forall \mathcal{E}, \mathcal{E}' \in \text{CBES} : (E = E' \land \forall \mathcal{E} \in \text{P}_{\text{fin}}(E) : \mathcal{E} \upharpoonright \mathcal{E}' = \mathcal{E}' \upharpoonright \mathcal{E}) \Leftrightarrow \mathcal{E} = \mathcal{E}'$.

Proof: It is easy to check that the conflict relations and the action-labeling functions coincide.

Let $(E_n)_{n \in \mathbb{N}}$ be a finite, monotone approximation of $E$. Suppose $X \mapsto e$ (the other inclusion follows by symmetrical arguments). Then $e \in E_n \Rightarrow (X \cap E_n \mapsto \mathcal{E} \upharpoonright E_n e)$. From the assumption we get $\mapsto \mathcal{E} \upharpoonright E_n \mapsto \mapsto \mathcal{E}' \upharpoonright E_n$. Hence, $X \mapsto e$ by the approximation closedness condition of $\mathcal{E}'$. \qed

Proof of Theorem 3.10: We have to verify the following facts (where $\mathcal{E} = \bigsqcup_n \mathcal{E}_n$):

Reflexivity: Obvious.

Transitivity: Follows from Remark 3.5.

Antisymmetry: This is an immediate consequence of Remark 3.7.

Least element: It is easily seen that $(\emptyset, \emptyset, \emptyset, \emptyset)$ is a least element.

$\mathcal{E}$ is a cbes: The only non-trivial fact is the approximation closedness condition.

Suppose $e \in E$, $X \subseteq E$ and $(E'_k)_{k \in \mathbb{N}}$ is a finite, monotone approximation of $E$ where $X \cap E'_k \in \{X' \cap E'_k \mid X' \mapsto e\}$ holds.

From the definition of $\bigsqcup_n \mathcal{E}_n$ we get that for all $k$ there is a $X_k \subseteq \bigcup_{n \in \mathbb{N}} E_n$ such that $X_k \cap E'_k = X \cap E'_k$ and $\forall n : e \in E_n \Rightarrow (X_k \cap E_n) \mapsto_n e$. Now suppose $e \in E_n'$, then we have $(X_k \cap E_n \cap E'_k) \mapsto \mathcal{E}_n \cap (E_k \cap E'_k) e$ whenever $e \in E'_k$. Furthermore, $X_k \cap E_n \cap E'_k = X \cap E_n \cap E'_k$. Therefore, $X \cap E_n \mapsto_n e$, since $\mathcal{E}_n \in \text{CBES}$ and $(E_n \cap E'_k)_{k \in \mathbb{N}}$ is a finite, monotone approximation of $E_n'$. And so by definition $X \mapsto e$. 


\( \mathcal{E} \) is an upper bound: Trivial, except for the bundle relation.

Suppose \( X \mapsto e \) and \( e \in E_k \). Then \( (X \cap E_k) \mapsto_k e \) by definition of \( \bigcup_n \mathcal{E}_n \).

Suppose \( X_k \mapsto_k e \). Let \( \Phi : \mathbb{N} \to \mathcal{P}(E) \) such that \( \Phi(n) = X_k \cap E_n \) if \( n \leq k \) and if \( n > k \) then \( \Phi(n) \in \{ X' \mid X' \mapsto_n e \land X' \cap E_{n-1} = \Phi(n-1) \} \). The existence of such a kind of function follows from the axiom of choice, since \( \mathcal{E}_{n-1} \subseteq \mathcal{E}_n \) implies that the above set is nonempty.

Furthermore, we show by induction that \( E_n \cap \Phi(i) = \Phi(n) \) whenever \( i \geq n \). If \( i = n \), then the statement follows, since \( \Phi(j) \subseteq E_j \). Suppose \( i > n \) then \( E_n \cap \Phi(i) = E_n \cap E_{i-1} \cap \Phi(i) = E_n \cap \Phi(i-1) \) and the rest follows by induction.

Therefore, \( E_n \cap \bigcup_i \Phi(i) = \Phi(n) \), since \( (\Phi(i))_{i \in \mathbb{N}} \) is monotone. And from the definition of \( \Phi \) we have \( \Phi(n) \mapsto_n e \) whenever \( e \in E_n \). Hence, \( \bigcup_i \Phi(i) \mapsto e \) and \( E_k \cap \bigcup_i \Phi(i) = X_k \).

\( \mathcal{E} \) is the least upper bound: Suppose \( \forall n : \mathcal{E}_n \subseteq \mathcal{E}' \). Then \( E \subseteq E' \). Hence, \( \mathcal{E}' \uparrow E \in \text{CBES} \) by Lemma 3.27.

Suppose \( \tilde{E} \in \mathcal{P}_{\text{fin}}(E) \). Then there is an \( n \) such that \( \tilde{E} \subseteq E_n \). Therefore, \( (\mathcal{E}' \uparrow E) \uparrow \tilde{E} = \mathcal{E}' \uparrow \tilde{E} = (\mathcal{E}' \uparrow E_n) \uparrow \tilde{E} = \mathcal{E}_n \uparrow \tilde{E} = (\mathcal{E} \uparrow E_n) \uparrow \tilde{E} = \mathcal{E} \uparrow \tilde{E} \). Hence, \( \mathcal{E}' \uparrow E = \mathcal{E} \) by Lemma 3.28, i.e. \( \mathcal{E} \subseteq \mathcal{E}' \).

### 3.6.2 Proof of Proposition 3.18

First, we show the well definedness and then the continuity.

**Lemma 3.29** All operators of Definition 3.17 are well defined, i.e. they really yield elements of CBES.

**Proof:** We only present the cases of the parallel and the refinement operator. The well-definedness of the other operators can be easily checked.

\( \mathcal{E}_1 \parallel A \mathcal{E}_2 \): Let \( (\tilde{e}_1, \tilde{e}_2) \in E_{\mathcal{E}_1 \parallel A \mathcal{E}_2} \). Define \( M_i = \{ X \mid X \mapsto_i \tilde{e}_i \} \). Then \( M_i \) is approximation closed with respect to \( E_i \cup \{ \star \} \). Thus by Corollary 2.20 \( M' = \{ (e_1, e_2) \in (E_1 \cup \{ \star \}) \times (E_2 \cup \{ \star \}) \mid e_i \subseteq X_i \mid i \in \{ 1, 2 \} \land X_i \subseteq M_i \} \) is approximation closed with respect to \( (E_1 \cup \{ \star \}) \times (E_2 \cup \{ \star \}) \). And so the approximation closedness of \( \mapsto \mathcal{E}_1 \parallel A \mathcal{E}_2 \) follows from Corollary 2.18, since \( \{ X \mid X \mapsto_{\mathcal{E}_1 \parallel A \mathcal{E}_2} (\tilde{e}_1, \tilde{e}_2) \} = \{ X' \cap E_{\mathcal{E}_1 \parallel A \mathcal{E}_2} \mid X' \in M' \} \).

\( \text{Ref}^*_A(\mathcal{E}, \theta) \): Let \( (e', \tilde{e}') \in E_{\text{Ref}^*_A(\mathcal{E}, \theta)} \). Define \( M = \{ X \mid X \mapsto e' \} \) and

\[
M_1 = \left\{ \begin{array}{ll}
\{ \{ e' \} \times \hat{X} \mapsto \theta(l(e')) \tilde{e}' \} & \text{if } l(e') \in A \\
\emptyset & \text{otherwise}
\end{array} \right.
\]

Then \( M_1 \) is approximation closed with respect to \( E_{\text{Ref}^*_A(\mathcal{E}, \theta)} \). Furthermore, let \( E_e = \left\{ \begin{array}{ll}
E_{\theta(l(e))} & \text{if } l(e) \in A \\
\{ e \} & \text{otherwise}
\end{array} \right. \) and \( M_e = \left\{ \begin{array}{ll}
\{ \text{exit}(\theta(l(e))) \} & \text{if } l(e) \in A \\
\{ e \} & \text{otherwise}
\end{array} \right. \).

Obviously, \( M_e \) is approximation closed with respect to \( E_e \).

From Corollary 2.21 we obtain that

\[
M_2 = \left\{ \begin{array}{ll}
\emptyset & \text{if } l(e') \in A \land \tilde{e}' \notin \text{init}(\theta(l(e'))) \\
\{ \{ (e, \hat{e}) \mid e \in X \land \hat{e} \in X_e \} \mid X \in M \land X_e \in M_e \} & \text{otherwise}
\end{array} \right.
\]

is approximation closed with respect to \( E_{\text{Ref}^*_A(\mathcal{E}, \theta)} \). And so the approximation closedness of \( \mapsto \text{Ref}^*_A(\mathcal{E}, \theta) \) follows from Proposition 2.15, since \( \{ X' \mid X' \mapsto \text{Ref}^*_A(\mathcal{E}, \theta) (e', \tilde{e}') \} = M_1 \cup M_2 \).
The other conditions are easy to check.

To simplify the verification of the continuity, we use Winskel’s continuity on events [178].

**Definition 3.30** Let $D$ be a cpo. An operator $F : D \rightarrow \text{CBES}$ is continuous on events if and only if $F$ is monotonic and for every $\omega$-chain $(d_i)_{i \in \mathbb{N}}$ in $D$ we have $E_{F(\bigsqcup_i d_i)} \subseteq \bigsqcup_i F(d_i)$.

**Lemma 3.31** Let $D$ be a cpo and let $F : D \rightarrow \text{CBES}$. Then $F$ is continuous if and only if $F$ is continuous on events.

**Proof:** Let $\sqsubseteq$ be the partial order of $D$. Continuity on events is obviously implied by continuity. Now let $(d_i)_{i \in \mathbb{N}}$ be an $\omega$-chain in $D$. Then

$$
\forall i : d_i \sqsubseteq \bigsqcup_i d_i \quad \Rightarrow \quad \forall i : F(d_i) \sqsubseteq F(\bigsqcup_i d_i)
$$

$$
\Rightarrow \quad \bigsqcup_i F(d_i) \sqsubseteq F(\bigsqcup_i d_i)
$$

$$
\Rightarrow \quad \bigsqcup_i F(d_i) \sqsubseteq F(\bigsqcup_i d_i) \land E_{F(\bigsqcup_i d_i)} \subseteq \bigsqcup_i F(d_i)
$$

**Remark 3.7**

$$
\Rightarrow \quad \bigsqcup_i F(d_i) = F(\bigsqcup_i d_i),
$$

which completes the proof.

**Lemma 3.32** All operators from Definition 3.17 are continuous with respect to $\sqsubseteq$.

**Proof:** It is straightforward to check that every operator from Definition 3.17 considered componentwise is continuous on events. Hence, the statement follows by Lemma 3.31 and Lemma 2.10.

**3.6.3 Proof of Theorem 3.25**

We introduce an event-based transition relation. Then we show that its corresponding transition system is bisimilar to $O^*(⟨\text{decl}, B⟩)$ and that it is bisimilar to $⟨\text{CBES}, \text{Act}_\sqrt{}, \rightarrow, [[[\text{decl}, B]]]⟩$. And so Theorem 3.25 follows by the transitivity of bisimilarity [138, page 90]. The event-based transition relation is particularly introduced to handle unguarded recursion.

**Event-Based Transition Systems.**

We want to define event-based transition rules such that the corresponding action occurrence is denoted by the event which corresponds to this action occurrence in the denotational semantics. Therefore, the information of the original positions (events) has to be kept in the derived expressions. This is done by defining the expression set $\text{EXP}_{sr}$, which contains exactly those elements generated by

$$
C ::= B | C; B | C[B | C]|_{\text{Act}} C | C[(a \rightarrow C)^{\alpha \in A}] | C \setminus A | [C]_i
$$
where $a \in \text{Obs}$, $B \in \text{EXP}_{s,r}$, $i \in \{1, 2\}$ and $A \subseteq \text{Obs}$ with $|A| \leq |\mathbb{N}|$. The symbols in the definition of EXP$_{s,r}^e$, e.g. $[> ]$, are overloaded, since they are also used in the definition of EXP$_{s,r}$. Hence, the unique derivation of an expression of EXP$_{s,r}^e$ is contradicted. Nevertheless, it does not harm our theory (both have the same transition rule) and therefore, we use the same symbols, especially in order to reduce the numbers of the transition rules.

We do not need to extend the declaration, i.e. we define PA$_{s,r}^e = (\text{Var} \rightarrow \text{EXP}_{s,r}) \times \text{EXP}_{s,r}^e$.

The function $L$ is adapted to $L' : \text{PA}_{s,r}^e \rightarrow \mathcal{P}_{\text{count}}(\text{Obs})$ as follows.

$L'(\langle \text{decl}, B \rangle) = L(\langle \text{decl}, B \rangle)$

$L'(\langle \text{decl}, C; B \rangle) = L'(\langle \text{decl}, C [> B] \rangle) = L'(\langle \text{decl}, C \rangle) \cup L'(\langle \text{decl}, B \rangle)$

$L'(\langle \text{decl}, C_1C_2 \rangle) = L'(\langle \text{decl}, C_1 \rangle) \cup L'(\langle \text{decl}, C_2 \rangle) \cup A$

$L'(\langle \text{decl}, C[a \rightarrow C_a] \rangle) = L'(\langle \text{decl}, C \rangle) \cup A \cup \bigcup_{a \in A} L'(\langle \text{decl}, C_a \rangle)$

$L'(\langle \text{decl}, C\langle A \rangle \rangle) = L'(\langle \text{decl}, C \rangle) \cup A$

$L'(\langle \text{decl}, [C]_i \rangle) = L'(\langle \text{decl}, C \rangle)$

The event transition rules $\rightarrow_{\text{decl}} \subseteq \text{EXP}_{s,r} \times ((\text{Act} \cup \text{Obs} \times \text{Obs}) \times \mathcal{U}) \times \text{EXP}_{s,r}^e$ are presented in Table 3.3. The elements of $\mathcal{U}$ in the transitions labels encode the position of the execution such that they correspond exactly to the events labeled by the denotational semantics. The original event positions are kept by using the $[\_]_i$ expressions. Another possibility to encode this information is presented in [38].

The First Bisimilarity Result.

We define a relation between EXP$_{s,r}^e$ and EXP$_{s,r}$ which yields a bisimulation. An expression $C$ of EXP$_{s,r}^e$ and an expression $B$ of EXP$_{s,r}$ are related if $C$ results in $B$ by removing all $[\_]$ expressions. This is formalized by the following function, where we also count the $[\_]$ symbols in $C$.

**Definition 3.33** $\Xi : \mathbb{N} \times \text{EXP}_{s,r} \rightarrow \mathcal{P}(\text{EXP}_{s,r}^e)$ is defined as follows

$\Xi(0, B) = \{B\}$

$\Xi(n + 1, B) = \{[\tilde{C}]_i | i \in \{1, 2\} \land \tilde{C} \in \Xi(n, B)\}$

if $B \in \{0, 1, a.B_1, \tau.B_1, B_1 + B_2, x\}$

$\Xi(n + 1, B_1; B_2) = \{[\tilde{C}]_i | i \in \{1, 2\} \land \tilde{C} \in \Xi(n, B_1; B_2)\} \cup$

$\{C_1; B_2 | C_1 \in \Xi(n + 1, B_1)\}$

$\Xi(n + 1, B_1 [> B_2]) = \{[\tilde{C}]_i | i \in \{1, 2\} \land \tilde{C} \in \Xi(n, B_1 [> B_2])\} \cup$

$\{C_1 [> B_2 | C_1 \in \Xi(n + 1, B_1)\}$

$\Xi(n + 1, B_1 \parallel A B_2) = \{[\tilde{C}]_i | i \in \{1, 2\} \land \tilde{C} \in \Xi(n, B_1 \parallel A B_2)\} \cup$

$\{C_1 \parallel C_2 | \exists m \in \mathbb{N} : m \leq n + 1 \land C_1 \in \Xi(m, B_1) \land C_2 \in \Xi(n + 1 - m, B_2)\}$

$\Xi(n + 1, B[(a \rightarrow B_a)a\in A]) = \{[\tilde{C}]_i | i \in \{1, 2\} \land \tilde{C} \in \Xi(n, B[(a \rightarrow B_a)a\in A])\} \cup$

$\{C[(a \rightarrow C_a)a\in A] | \exists m \in \mathbb{N}, (m_a)_{a\in A} \in \mathbb{N}^A : m + \sum_{a\in A} m_a = n + 1 \land$

$C_a \in \Xi(m_a, B_a) \land C \in \Xi(m, B)\}$

$\Xi(n + 1, B \setminus A) = \{[\tilde{C}]_i | i \in \{1, 2\} \land \tilde{C} \in \Xi(n, B \setminus A)\} \cup$

$\{C \setminus A | C \in \Xi(n + 1, B)\}$
The well-definedness of \( \Xi \) is easily seen. Furthermore, \( \Xi \) has no effect on the action names occurred in the processes, i.e.

**Lemma 3.34** \( C \in \Xi(n, B) \Rightarrow \mathcal{L}(\langle \text{decl}, B \rangle) = \mathcal{L}'(\langle \text{decl}, C \rangle) \)

**Proof:** By structural induction on \( B \) combined with \( n \). \( \square \)
Lemma 3.35 Let $B \in \text{EXP}_{sr}$ then $O^s((\text{decl}, B))$ is bisimilar to $(\text{EXP}^e_{sr}, \text{Act}_{/}, \rightarrow O^s, B)$ where $C \xrightarrow{\alpha} O^s, C' \Leftrightarrow \exists e \in U : C \xrightarrow{\epsilon} \text{decl} C'$.

Proof: Define $R = \{(B, C) \in \text{EXP}_{sr} \times \text{EXP}^e_{sr} \mid \exists n : C \in \Xi(n, B)\}$. In order to verify that $R$ is a bisimulation, we show

$$\left( B \xrightarrow{\gamma} B' \land C \in \Xi(n, B) \right) \Rightarrow \exists e, C', m : C \xrightarrow{\epsilon} C' \land C' \in \Xi(m, B')$$

(3.1)

The proof of (3.1) works by induction on the depth of inference of $B \xrightarrow{\alpha} B'$, combined with the value of $n$. Then (3.1) can be easily checked with the following procedure:

- applying rule $N$ whenever $C = [\tilde{C}]$. In these cases $n$ is reduced by one and $B \xrightarrow{\alpha} B'$ is unaffected. Therefore the hypothesis concludes the result.

- applying the correspondent rules of $B \xrightarrow{\alpha} B'$ whenever $C$ is different from $[\tilde{C}]$. In these cases the depth of inference is reduced and $n$ gets not increased. Therefore the hypothesis concludes the result. In the case of rule $\text{Ref}_2$, we also use Lemma 3.34 for the conclusion.

Another fact is

$$\left( C \xrightarrow{\epsilon} C' \land C \in \Xi(n, B) \right) \Rightarrow \exists B', m : B \xrightarrow{\gamma} B' \land C' \in \Xi(m, B')$$

(3.2)

This equation can be seen by induction on the depth of inference of $C \xrightarrow{\epsilon} C'$. In the case of rule $\text{Ref}_2'$, we also use Lemma 3.34.

Now we are ready to verify that $R$ is a bisimulation:

- It is clear that $(B, B) \in R$.

- Suppose $(B_1, C_1) \in R$ and $B_1 \xrightarrow{\alpha} O^s B_2$. Then $B_1 \xrightarrow{\alpha} \text{decl} B_2$ by definition. Hence, $\exists e, C_2, m : C_1 \xrightarrow{\epsilon} C_2 \land C_2 \in \Xi(m, B_2)$ by (3.1). Thus $C_1 \xrightarrow{\epsilon} O^s, C_2$ and $(B_2, C_2) \in R$, as required.

- Suppose $(B_1, C_1) \in R$ and $C_1 \xrightarrow{\epsilon} O^s, C_2$. Then $C_1 \xrightarrow{\epsilon} C_2$ for some $\epsilon$. Hence, $\exists B_2, m : B_1 \xrightarrow{\epsilon} \text{decl} B_2 \land C_2 \in \Xi(m, B_2)$ by (3.2). Thus $B_1 \xrightarrow{\epsilon} O^s, B_2$ and $(B_2, C_2) \in R$, as required. \qed

Remark 3.36 The transition systems mentioned in Lemma 3.35 are not isomorphic. Consider, for example, $\text{decl}$ with $\text{decl}(x) = a; x$. Then the expression $x$ yields a finite transition system with respect to $O^s$, whereas an infinite one is derived with respect to $\rightarrow O^s$. 

The Second Bisimilarity Result.

First, we show that the denotation of a variable is the same as the denotation of its corresponding expression.

**Lemma 3.37** Let \( \text{decl} : \text{Var} \to \text{EXP} \) and \( x \in \text{Var} \). Then \( [\langle \text{decl}, x \rangle] = [\langle \text{decl}, \text{decl}(x) \rangle] \).

**Proof:** We have \( [\langle \text{decl}, x \rangle] = [x]_{[\text{decl}]} = [\langle \text{decl} \rangle](x) = \text{fix}(\mathcal{F}_{\text{decl}})(x) = \mathcal{F}_{\text{decl}}([\langle \text{decl} \rangle])(x) = [\text{decl}(x)]_{[\text{decl}]} = [\langle \text{decl}, \text{decl}(x) \rangle] \).

We extend the denotational semantics to \( \text{PA}^{e}_{sr} \).

**Definition 3.38** (Denotational semantics of \( \text{PA}^{e}_{sr} \))

\[ \text{Shift}_{1} : \text{CBES} \to \text{CBES} \] with \( \text{Shift}_{1}(\mathcal{E}) = (\{\ast_{i}\} \times E, \sim_{1}, \rightarrow_{1}, \bar{l}) \) where

\[ \sim_{1} = \{((x_{i}, e), (x_{i}, e')) \mid (e, e') \in \sim\} \]
\[ \rightarrow_{1} = \{\{(\ast_{i}) \times X, (x_{i}, e) \mid (X, e) \in \rightarrow\} \]
\[ \bar{l}(x_{i}, e) = l(e) \]

Furthermore, define \( [\_]' : \text{PA}^{e}_{sr} \to \text{CBES} \) by

\[ [\langle \text{decl}, B \rangle]' = [\langle \text{decl}, B \rangle] \]
\[ [\langle \text{decl}, C; B \rangle]' = [\langle \text{decl}, C \rangle]' \oplus [\langle \text{decl}, B \rangle]' \]
\[ [\langle \text{decl}, C \geq B \rangle]' = [\langle \text{decl}, C \rangle]' \geq [\langle \text{decl}, B \rangle]' \]
\[ [\langle \text{decl}, C \mid A \rangle]' = [\langle \text{decl}, C \rangle]' \mid A \]
\[ [\langle \text{decl}, C \setminus A \rangle]' = [\langle \text{decl}, A \rangle]' \setminus A \]
\[ [\langle \text{decl}, [C]_{i} \rangle]' = \text{Shift}_{1}([\langle \text{decl}, C \rangle]') \]

It is clear that \( [\_]' \) is well defined.

**Lemma 3.39** Suppose \( \langle \text{decl}, C \rangle \in \text{PA}^{e}_{sr}, b \in \text{Obs} \) and \( b \notin \mathcal{L}'(\langle \text{decl}, C \rangle) \). Then for all \( e \in E_{[\langle \text{decl}, C \rangle]}' \) we have \( l_{[\langle \text{decl}, C \rangle]}'(e) \neq b \).

**Proof:** First we show for any \( \text{decl} : \text{Var} \to \text{EXP} \) that for all \( n \in \mathbb{N} \) and \( B \in \text{EXP} \) we have

\[ b \notin \mathcal{L}(\langle \text{decl}, B \rangle) \Rightarrow \forall e \in \pi_{1}(\mathcal{F}_{\text{dec}}^{a}(\bot)) : b \neq \pi_{1}(\mathcal{F}_{\text{dec}}^{a}(\bot))(e) \quad (3.3) \]

This can be easily checked by induction on \( n \) combined with the structure of \( B \) where the lexicographic order is used.

The main statement follows by structural induction on \( C \). We only present the case \( C = B \in \text{EXP} \). By Lemma 3.20 we get that \( [\langle \text{decl}, B \rangle] = \bigsqcup_{n} [B]_{\mathcal{F}_{\text{dec}}^{n}(\bot)} \). Then there is \( m \) such that \( e \in \pi_{1}([B]_{\mathcal{F}_{\text{dec}}^{m}(\bot)}) \). And so the result follows by (3.3).

We introduce the following definition in order to obtain an adequate transition relation in \( \text{CBES} \) which is labeled with elements of \( \text{Obs} \times \text{Obs} \).
3.6. PROOFS

Let $E \in \text{CBES}$, $b \in \text{Obs}$ and $e \in \text{init}(E)$ such that $l(e) \in \text{Obs}$. Then $E_{[e/b]}$ of $E$ is given by $(E', \leadsto, \mapsto, l')$ where

\[
E' = \{ e' \in E \mid \neg(e' \leadsto e) \}
\]
\[
\leadsto = \leadsto \cap (E' \times E')
\]
\[
\mapsto = \{ (X \cap E', e') \mid e' \in E' \land X \mapsto e' \}
\]
\[
l'(e') = \begin{cases} 
  b & \text{if } e' = e \\
  l(e') & \text{otherwise}
\end{cases}
\]

Lemma 3.41 Let $E \in \text{CBES}$, $b \in \text{Obs}$ and $e \in \text{init}(E)$ such that $l(e) \in \text{Obs}$. Then $E_{[e/b]} \in \text{CBES}$.

Proof: Define $\tilde{E} = (E, \leadsto, \mapsto, \tilde{l})$ with $\tilde{l}(e) = b$ and $\forall e' \in E \setminus \{e\} : \tilde{l}(e') = l(e')$. It is obvious that $\tilde{E} \in \text{CBES}$. Furthermore, $E_{[e/b]} = \tilde{E} \uparrow \pi_1(\tilde{E}_{[e/b]})$. And so the result follows by Lemma 3.27. □

Lemma 3.42 Suppose $E, E_1, E_2, E_a \in \text{CBES}$. Then

\[
(a \cdot E)_{[*]} = \text{Shift}_{\downarrow}(E)
\]
\[
(E_1 + E_2)_{([*, e])} \simeq \text{Shift}_{\downarrow}(E_{[e]})
\]
\[
(E_1 \downarrow E_2)_{([*, e])} \simeq \begin{cases} 
  \text{Shift}_{\downarrow}(E_2) & \text{if } l_1(e) = \sqrt \land e \in \text{init}(E_1) \\
  E_{[e]} \downarrow E_2 & \text{otherwise}
\end{cases}
\]
\[
(E_1[>E_2])_{([*, e])} \simeq \begin{cases} 
  \text{Shift}_{\downarrow}(E_{[e]}) & \text{if } l_1(e) = \sqrt \land i = 1 \\
  E_{[e]}[>E_2] & \text{if } l_1(e) \neq \sqrt \land i = 1 \\
  \text{Shift}_{\downarrow}(E_{[e]}) & \text{otherwise}
\end{cases}
\]
\[
(E_1 \Box A E_2)_{([e_1, e_2])} \simeq \begin{cases} 
  E_{[e_1]} \Box A E_2 & \text{if } e_2 = * \land l_1(e_1) \notin A \cup \{\sqrt\} \\
  E_{[e_1]} \Box A E_{[e_2]} & \text{if } e_1 = * \land l_2(e_2) \notin A \cup \{\sqrt\} \\
  E_{[e_1]} \Box A E_{[e_2]} & \text{if } l_1(e_1) = l_2(e_2) \notin A \cup \{\sqrt\}
\end{cases}
\]
\[
\text{Ref}_{\downarrow}(E, \theta)_{([e, e])} \simeq \begin{cases} 
  \text{Ref}_{\downarrow}^*(E_{[e]}, \theta) & \text{if } (l_{\theta(l(e))}(\hat{e}) = \sqrt \land \hat{e} \in \text{init}(\theta(l(e)))) \lor \\
  (l(e) \notin A \land e = \hat{e}) & \text{if } l_{\theta(l(e))}(\hat{e}) \neq \sqrt \\
  \text{Ref}_{\downarrow}^*(E_{[e/b]}, \theta \cup \{(b, \theta(l(e))_{[e]}))\} & \text{if } l_{\theta(l(e))}(\hat{e}) \neq \sqrt \\
  \text{whenever } b \notin A \land \forall \hat{e} \in E : l(\hat{e}) \neq b
\end{cases}
\]
\[
(E \Box A)_{[e]} \simeq E_{[e]} \Box A
\]
\[
\text{Shift}_{\downarrow}(E)_{([*, e])} \simeq \text{Shift}_{\downarrow}(E_{[e]})
\]

Proof: Straightforward. □

Lemma 3.43 Suppose $E, E_1, E_2, E_a \in \text{CBES}$ and $b \in \text{Obs}$. Then

\[
(a \cdot E)_{[*]} = b \cdot E & \text{if } a \in \text{Obs}
\]
\[
(E_1 \downarrow E_2)_{([*, e])} \simeq \text{Shift}_{\downarrow}(E_{[e/b]})
\]
\[
(E_1 \downarrow E_2)_{([*, e])} \simeq E_{[e/b]} \downarrow E_2
\]
Suppose \( \mathcal{E}_1 \models \mathcal{E}_2 \) and \( \mathcal{E}_2 \models \mathcal{E}_3 \). Then

\[
\mathcal{E}_1 \models \mathcal{E}_3
\]

Proof: Straightforward.

Lemma 3.44 Suppose \( \langle \text{decl}, C \rangle \in \text{PA}_{sr}^e \) and \( C \rightarrow^{p, l} C' \). Then

\[
\langle \gamma = (a, b), b \notin \text{Act} \langle \langle \text{decl}, C \rangle \rangle \rangle \Rightarrow e \in \text{init}(\mathcal{E}) \land l(e) = a \land \mathcal{E}' = \mathcal{E}[e/b]
\]

\[
\gamma \in \text{Act} \Rightarrow e \in \text{init}(\mathcal{E}) \land l(e) = \gamma \land \mathcal{E}' = \mathcal{E}[e]
\]

with \( \mathcal{E} = [\langle \text{decl}, C \rangle]' \) and \( \mathcal{E}' = [[\langle \text{decl}, C' \rangle]'] \).

Proof: We use induction on the depth of inference of \( C \rightarrow^{p, l} C' \). Then the equation can be verified by case analysis on the derivation rules, where Lemma 3.42 and Lemma 3.43 are used. In the cases of \( P_i, \text{Ref}^i_2 \) and \( \text{Res}^i_1 \), we also use Lemma 3.39. And in the case of \( \text{Rec}' \), we make use of Lemma 3.37.

Lemma 3.45 Let \( \langle \text{decl}, C \rangle \in \text{PA}_{sr}^e, e \in \text{init}([\langle \text{decl}, C \rangle]' \rangle) \) and \( \alpha = l_{[\langle \text{decl}, C \rangle]'} \). Then

\[
\exists C'' \in \text{EXP}_{sr}^e : C \rightarrow^{p, l} C' \land (\alpha \in \text{Obs}) \Rightarrow \forall b \in \text{Obs} : \exists C''' \in \text{EXP}_{sr}^e : C \rightarrow^{p, l} C'''
\]

Proof: First we show for any decl : Var \( \rightarrow \) EXP_{sr}:

\[
\forall n \in \mathbb{N} : \forall B \in \text{EXP}_{sr} : e \in \text{init}(\llbracket B \rrbracket_{\pi_{\text{decl}}(\perp)}) \land \alpha = \pi_4(\llbracket B \rrbracket_{\pi_{\text{decl}}(\perp)})(e) \Rightarrow \exists C'' \in \text{EXP}_{sr}^e : B \rightarrow^{p, l} C' \land (\alpha \in \text{Obs}) \Rightarrow \forall b \in \text{Obs} : \exists C''' \in \text{EXP}_{sr}^e : B \rightarrow^{p, l} C'''
\] (3.4)

This is done by induction on \( n \) combined with the structure of \( B \), where the lexicographic order is used. Furthermore, a case analysis on the structure of \( B \) is made. Here, we only present the case \( B = x : e \in \text{init}(\llbracket x \rrbracket_{\pi_{\text{decl}}(\perp)}) \) implies that \( n > 0 \). Therefore, \( \llbracket x \rrbracket_{\pi_{\text{decl}}(\perp)} = \pi_{\text{decl}}(\perp)(x) = [\text{decl}(x)]_{\pi_{\text{decl}}(\perp)}^{-1} \). The rest follows by induction, since \( n \) is reduced. Thus (3.4) is established.

The main statement follows now by structural induction on \( C \). We only present the case \( C = B \in \text{EXP}_{sr} \). By Lemma 3.20 we get that \( \llbracket \langle \text{decl}, B \rangle \rrbracket = \bigsqcup_n \llbracket B \rrbracket_{\pi_{\text{decl}}(\perp)} \). Then it is easily seen that there is an \( m \) such that \( e \in \text{init}(\llbracket B \rrbracket_{\pi_{\text{decl}}(\perp)}) \) and \( \alpha = \pi_4(\llbracket B \rrbracket_{\pi_{\text{decl}}(\perp)}) \). And so the result follows by (3.4).
Lemma 3.46  Let \( \langle \text{decl}, C \rangle \in \text{PA}_{sr} \), then the transition systems \((\text{EXP}^e_{sr}, \text{Act}_v, \rightarrow'_{O^*}, C)\) and \((\text{CBES}, \text{Act}_v, \leftarrow, [\langle \text{decl}, C \rangle]'\)) are bisimilar, where \( \rightarrow'_{O^*} \) is defined as in Lemma 3.35.

Proof: Define \( R = \{ (C', [\langle \text{decl}, C' \rangle])' \mid C' \in \text{EXP}^e_{sr} \} \). Then \( (C, [\langle \text{decl}, C \rangle])' \in R \) by definition.

Suppose \( C_1 \in \text{EXP}^e_{sr} \) and \( C_1 \xrightarrow{\alpha'} C_2 \). Then \( C_1 \xrightarrow{\alpha} C_2 \) for some \( e \). Hence, by Lemma 3.44 \( [\langle \text{decl}, C_1 \rangle]' \xrightarrow{\alpha} [\langle \text{decl}, C_2 \rangle]' \), as required.

Suppose \( C_1 \in \text{EXP}^e_{sr} \) and \( [\langle \text{decl}, C_1 \rangle]' \xrightarrow{\alpha} E_2 \). Then there is \( e \in \text{init}([\langle \text{decl}, C_1 \rangle]') \) such that \( E_2 = [\langle \text{decl}, C_1 \rangle]'_e \) and \( \alpha = \pi_4([\langle \text{decl}, C_1 \rangle]')(e) \). From Lemma 3.45 we get the existence of \( C_2 \in \text{EXP}^e_{sr} \) such that \( C_1 \xrightarrow{\alpha} C_2 \). Moreover, \( [\langle \text{decl}, C_1 \rangle]'_e = [\langle \text{decl}, C_2 \rangle]' \) by Lemma 3.44, which concludes the proof. \( \square \)
Chapter 4

Modeling the End-Based View in CBES

A choice in concurrent systems is usually taken by the start of actions. In this chapter, we propose the alternative view that a choice is determined by the ending of actions, called end-based view, as this alternative has relevant applications and interesting implications, as illustrated in Section 1.3.

Another advantage of the end-based approach is that it can simplify the action refinement approach for timed systems. More precisely, an action refinement operator for timed bundle event structures [116, 120] is presented in [81, 133]. The authors extend each $Q$ in a refinement $P[a \rightarrow Q]$ by an additional internal event, which corresponds to the start of $Q$. This is necessary in order to guarantee the start-based choice, i.e. to guarantee that the choice is triggered at the start of $Q$ (at the time when action $a$ starts to be executed in $P$). In other words, the choice is not triggered when $Q$ executes its first action, which may be executed after a time period has passed. This approach is reasonable if generative systems\(^1\), rather than reactive systems, are considered. The problem is that an undesired internal choice may be introduced by action refinement in reactive systems, for example refining $a$ to $a'$ and $b$ to $b'$ in $a + b$ yields $\tau.a' + \tau.b'$ in [81, 133]. We have the opinion that the introduction of additional internal events can be avoided in an end-based choice setting.

The different points of view (start-based vs end-based) lead to different refinement operators on CBES. We introduce a refinement operator on closed bundle event structures for the end-based view. Furthermore, we show that the standard equivalences are not preserved by this refinement operator. Therefore, we introduce and study new equivalences that are preserved by our refinement operator.

4.1 An End-Based Refinement Operator on CBES

The decision at which time a choice is triggered does not influence the theory of most untimed semantic models. This situation changes when models that contain an action refinement operator are considered. Action refinement can, for example, split an action into a start- and an

\(^1\)Generative systems are systems that are considered independently of the environment, i.e. the behavior is generated. Consequently, the branching structure is considered with respect to the abstractions of the internal actions.
end-action. More precisely, if action $a$ is refined to $a_1$ followed by $a_2$, i.e. $a_2$ executes after $a_1$, in a process consisting of a choice between actions $a$ and $b$, then the execution sequence $a_1, b$ is allowed in an end-based setting, which is not the case in a start-based setting. This example is illustrated in Figure 4.1, where we depict extended (or closed) bundle event structures (see Subsection 3.3.1 and Subsection 3.3.2) after the refinement in the start-based and in the end-based approaches. It can be seen in this figure that $a_1$ is in conflict with $b$ in the start-based approach, whereas it is possible that $a_1$ precedes $b$ in the end-based approach (a non-symmetric relation is used, depicted by dashed arrows). Thus the sequence $a_1, b$ is only a trace of the event structure corresponding to the end-based view.

In the standard approach to action refinement, a choice is triggered by the start of actions, for example in [6, 63, 91, 98, 133, 172]. Here, we develop an action refinement operator for an untimed event structure with respect to the end-based point of view. Two constraints have to be imposed on the event structure in order to give a reasonable definition of such a refinement operator.

1. Each event in an event structure represents a unique occurrence of an action. This is necessary, since otherwise the occurrence of an action could be started more than once. In prime event structures [145] such a unique representation can not be guaranteed, as pointed out, for example, in [91, Section 2.3].

2. An event structure must allow to model disruption, since a disrupt operation can result from the end-based refinement operator, which will be discussed in more detail later in this section. This is for example not the case for prime event structures [36, 38] and stable event structures [178].

Therefore, we choose to define the end-based refinement operator on closed bundle event structures (CBES), which are introduced in Subsection 3.3.2.
The definition of the end-based refinement operator differs from the classical definition (Definition 3.17) with respect to the conflict relation: Only the termination events of the refining processes are used in our approach to define the conflict relation whereas every event (or every initial event) is used in the standard approach. More precisely, in the classical definition we have: If \( e \) is in conflict with \( e' \) and \( e \) (\( e' \) respectively) is refined to \( E_e \) (\( E_{e'} \) respectively), then every (initial) event of \( E_e \) is placed in conflict with every (initial) event of \( E_{e'} \) and vice versa. In our definition, we place every event of \( E_e \) in conflict with the termination events of \( E_{e'} \), i.e. they may only be executed if they are executed before every termination event of \( E_{e'} \). And, of course, we put every event of \( E_{e'} \) in conflict with the termination events of \( E_e \). By this approach, we guarantee that a choice is triggered by the ending of actions, i.e. by a termination event of the refining process. Formally:

Let \( \tau \), \( \text{Obs} \), \( \text{Act}_\vee \) and \( \text{Var} \) be defined as in Section 3.2.

**Definition 4.1 (Refinement Operator)** Let \( A \subseteq \text{Obs} \). Then define \( \text{Ref}^e_A : \text{CBES} \times (A \rightarrow \text{CBES}) \rightarrow \text{CBES} \) by \( \text{Ref}^e_A(E, \theta) = \left( \tilde{E}, \tilde{\sim}, \tilde{\rightarrow}, \tilde{I} \right) \) where

\[
\tilde{E} = \{(e, \tilde{e}) \mid e \in E \land l(e) \in A \land \tilde{e} \in E_{\theta(l(e))}\} \cup \{(e, \tilde{e}) \in E \land l(e) \notin A\}
\]

\[
\tilde{\sim} = \{(e_1, e_2) \mid (e_1 \sim e_2 \land (l(e_2) \in A \Rightarrow \tilde{e}_2 \in \text{exit}(\theta(l(e_2)))) \lor (e_1 = e_2 \land l(e_1) \in A \land \tilde{e}_1 \neq \tilde{e}_2 \land (\tilde{e}_1 \sim_{\theta(l(e_1))} \tilde{e}_2 \lor \tilde{e}_2 \in \text{exit}(\theta(l(e_1))))))\}
\]

\[
\tilde{\rightarrow} = \{(X, (e, \tilde{e})) \mid \exists X : X \hookrightarrow e \land (l(e) \in A \Rightarrow \tilde{e} \in \text{init}(\theta(l(e))))\} \land X = \{(\tilde{e}', \tilde{e}') \in \tilde{E} \mid \tilde{e}' \in X \land (l(e') \in A \Rightarrow \tilde{e}' \in \text{exit}(\theta(l(e'))))\}
\]

\[
\tilde{I}(e, \tilde{e}) = \begin{cases} l(e) & \text{if } l(e) \notin A \\ l_{\theta(l(e))}(\tilde{e}) & \text{if } l(e) \in A \land l_{\theta(l(e))}(\tilde{e}) \neq \sqrt{\tau} \\ \tau & \text{if } l(e) \in A \land l_{\theta(l(e))}(\tilde{e}) = \sqrt{\tau} \end{cases}
\]

**Lemma 4.2** The refinement operator \( \text{Ref}^e \) is well defined, i.e. it yields elements of \( \text{CBES} \).

**Proof:** The approximation closedness of \( \text{Ref}^e_A(E, \theta) \) follows from the approximation closedness of \( \text{Ref}^e_A(E, \theta) \) (Lemma 3.18), since they do not differ in their bundle relations. The other conditions are easy to check. \( \square \)

**Example 4.3** Figure 4.2 illustrates how the refinement operator (\( \text{Ref}^e \)) behaves. For a better understanding, we augment the examples by process term descriptions of the systems (see Section 3.2). Furthermore, \( (a \rightarrow E_{12}) \) denotes the function from \( \{a\} \) to \( \text{CBES} \) that maps \( a \) to \( E_{12} \). The effect of the classical (start-based) action refinement operator on this example is depicted in Figure 3.4.

Figure 4.2 illustrates that the events labeled by \( a_1 \) are not in conflict with each other in \( E_{ref}^+ \). Only the events that correspond to the termination of \( E_{12} \) (they are labeled by \( \tau \) in \( E_{ref}^+ \)) disable the other actions. In other words, both actions \( a \) in \( E_{ref}^+ \) may start and execute their actions independently until one of them terminates.

**Remark 4.4** Our refinement operator allows the modeling of a disrupt mechanism as it is used, for example, in LOTOS [32] and in Chapter 3: Suppose the process described by \( E_2 \) can disrupt
the process described by $E_1$, i.e. $E_1 \triangleright E_2$ where $\triangleright$ is introduced in Definition 3.17. Then let $E_1 \hat{\triangleright} E^a$ be the process that is obtained by taking a choice between $E_1$ and the event structure $E^a$ from Figure 4.4 where label $a$ does not appear in $E_1$. Then the event structures $E_1 \hat{\triangleright} E_2$ and $Ref_{\{a\}}(E_1 + E^a, (a \rightarrow E_2))$ possess the same behavior.

4.2 Equivalences

First, we give a short overview of some standard equivalence notions and argue that they are not reasonable for an end-based action refinement operator, since they are not preserved by the end-based refinement operator. Then we introduce new equivalences that are preserved by the end-based refinement operator. These equivalences are examined with respect to their discriminating power. Finally, the issue concerning the coarsest equivalence with respect to trace (respectively bisimulation) equivalence is discussed.

4.2.1 Standard Equivalence Notions

*Trace* and *strong bisimulation equivalences* for CBES are defined as follows.

**Definition 4.5 (Trace Equivalence)** Two $E, E' \in$ CBES are *trace equivalent*, denoted by $E \sim_t E'$, if and only if the transition systems $(CBES, Act_\vee, \hookrightarrow, E)$ and $(CBES, Act_\vee, \hookrightarrow, E')$ (where $\hookrightarrow$ is defined in Definition 3.16) are trace equivalent (Definition 2.4).

**Definition 4.6 (Strong Bisimulation Equivalence)** Two $E, E' \in$ CBES are *strong bisimilar* (or *strong bisimulation equivalent*), denoted by $E \sim_b E'$, if and only if the transition systems $(CBES, Act_\vee, \hookrightarrow, E)$ and $(CBES, Act_\vee, \hookrightarrow, E')$ are bisimilar (Definition 2.5).

The cbes obtained from $(a + b.0)_{\{a\}}(a_{\|a}a.c)$, depicted in Figure 3.5 (and in Figure 3.3), and the cbes obtained from $b.0 + a.c.0$ and from $b.0 + a.c.0 + a.0$, depicted in Figure 4.3 are all trace equivalent, whereas only the first and the third one are strong bisimilar.

Further equivalences that are based on interleaving [86, 89, 170] can be defined in a straightforward manner for cbes if one uses their corresponding transition system.
4.2. EQUIVALENCES

**Step equivalence:** Here, a finite multiset of actions, i.e., a finite set of events, may be executed at once, as opposed to the interleaving approach, where only single actions may be executed at once. In [152] trace and bisimulation versions of this equivalence have been proposed.

**Pomset equivalence:** Here, a finite partially ordered set of events, more precisely a pomset [153], may be executed. Pomsets are equivalence classes with respect to the action labeling and the order. The order of a pomset corresponds to the causality order of the events. In [33] trace and bisimulation versions of this equivalence have been proposed.

**History preserving equivalence:** Here, the causal order in which events have been executed is additionally taken into account. There are different versions depending on to what degree the past information is taken into account. There exists weak, normal and hereditary history preserving bisimulations [23, 67, 91].

[91] examines which equivalence notions are preserved by a start-based action refinement operator in a configuration structure setting. There it is shown that pomset trace equivalence, history preserving bisimulation and hereditary history preserving bisimulation are preserved by a start-based action refinement operator. As mentioned before, the ST-equivalence is also preserved by a start-based action refinement. All other equivalences of this subsection are not preserved by a start-based action refinement.

The action refinement operator in an end-based setting is not compatible with the equivalence notions mentioned. This can be seen as follows: In the case of the end-based refinement operator \((\text{Ref}^e)\) any equivalence that implies trace equivalence (Definition 4.5) and that identifies \(a\) and \(a + a\) (like all the equivalences mentioned do) is not preserved. This is the case, because \(\mathcal{E}_{\text{ref}}\) from Figure 4.4 and \(\mathcal{E}_{\text{ref}}^+\) from Figure 4.2 are not trace equivalent. Resource bisimulation [58, 59], which is defined on transition systems, is the only equivalence known to us that does not identify \(a + a\) and \(a\).

In the following subsections, we present new equivalences which are indeed congruences for our refinement operator. For simplicity, we introduce the following definition, which determines
Figure 4.4: End-Based Refinement in CBES (2)

Figure 4.5: Some Closed Bundle Event Structures

the initial events of an event structure with respect to their labels.

Definition 4.7 Define \init_a(\E) = \{ e \in \init(\E) \mid l(e) \in A \}, where A \subseteq \Act. Furthermore, we write \init_a(\E) as a short hand for \init_{\{a\}}(\E).

4.2.2 ICT-Equivalence on CBES

The first considered equivalence notion is derived from trace equivalence. An equivalence notion which is a congruence for the end-based refinement operator has to distinguish between \E and \E' from Figure 4.2 and \E^a from Figure 4.4. One way to achieve this is to guarantee that the number of the initial events which are labeled by the same action have to be equal, i.e. \E and \E' can only be equivalent if \init_a(\E) = \init_a(\E') for all a \in \Obs. Moreover, we also have to guarantee a relationship between the numbers of the initial events with the same label of the corresponding remainders of the event structures. For example, consider \E^b+ and \E^{ba} from Figure 4.5. Then \( (b, a_1, a_1) \in T(Ref^e_{\{a\}}(\E^{b+}, (a \rightarrow \E_{12}))) \) but \( (b, a_1, a_1) \notin T(Ref^e_{\{a\}}(\E^{ba}, (a \rightarrow \E_{12}))) \), where \T(\E) are the traces (Definition 2.4) of the derived transition system of \E (compare with Definition 4.5). Hence, \Ref^e_{\{a\}}(\E^{b+}, (a \rightarrow \E_{12})) and \Ref^e_{\{a\}}(\E^{ba}, (a \rightarrow \E_{12})) are not trace equivalent.

Further difficulties become evident by a closer look at \E^{a|b} and \E^{int} from Figure 4.5: \E^{a|b} and \E^{int} satisfy our above criterion, but \( (a_1, b, a_1) \in T(Ref^e_{\{a\}}(\E^{int}, (a \rightarrow \E_{12}))) \) and \( (a_1, b, a_1) \notin T(Ref^e_{\{a\}}(\E^{a|b}, (a \rightarrow \E_{12}))) \), i.e. our tentative relation is not a congruence for the refinement.

Therefore, we introduce the initial event traces of a cbes. They consist of an event execution sequence and of a subset of the initial events for every execution step. Two cbes, \E_1 and \E_2,
4.2. EQUIVALENCES

Equivalences are considered to be equivalent if every initial event trace of $E_1$ can be mapped by an injective function $f$ to an initial event trace of $E_2$ and vice versa. Furthermore, this function has to be labeling preserving, i.e. $\forall e_1 \in E_1 : l_1(e_1) = l_2(f(e_1))$. The equivalence is precisely given by the following definition, where $E_{[e]}$ is defined in Definition 3.14.

**Definition 4.8 (ICT-equivalence)** Let $E \in \text{CBES}$. Then the initial event traces of $E$ are defined by $T_{ic}(E) = \{(e_i, \gamma_i)_{i \leq n} | n \in \mathbb{N} \land \exists \mathcal{E}_0, \cdots, \mathcal{E}_{n+1} : \mathcal{E}_0 = \mathcal{E} \land \forall i \leq n : \mathcal{E}_{i[e]} = \mathcal{E}_{i+1} \land \gamma_i \in \mathcal{P}_{\text{fin}}(\text{init Obs}(\mathcal{E}_i))\}$.

Two cbes, $E$ and $E'$, are initial corresponding trace equivalent (ICT-equivalent), denoted by $E \sim_{ICT} E'$, if

- for every $(e_i, \gamma_i)_{i \leq n} \in T^{ic}(E)$ there exists an injective, labeling-preserving function $f : (\bigcup_{i \leq n}(\gamma_i \cup \{e_i\})) \rightarrow E'$ such that $(f(e_i), f(\gamma_i))_{i \leq n} \in T^{ic}(E')$ and

- for every $(e'_i, \gamma'_i)_{i \leq n} \in T^{ic}(E')$ there exists an injective, labeling-preserving function $f' : (\bigcup_{i \leq n}(\gamma'_i \cup \{e'_i\})) \rightarrow E$ such that $(f'(e'_i), f'(\gamma'_i))_{i \leq n} \in T^{ic}(E)$

$E^{ba}$ and $E^{ab}$, and also $E^{a|b}$ and $E^{int}$ from Figure 4.5 are not ICT-equivalent. In addition, the event structures from Figure 4.6 are not ICT-equivalent either. This holds, since in $E^{w1}$ it is possible that both events labeled by $b$ become enabled, which is not the case for $E^{w2}$. Examples of ICT-equivalent event structures are given in Figure 4.7 and in Figure 4.8.

**Proposition 4.9** Two ICT-equivalent cbes are also trace equivalent, i.e. $\sim_{ICT} \subset \sim_t$.

**Proof:** It follows from the fact that every trace is also an initial trace, where the second component is always empty.

\[\square\]
**Theorem 4.10** ICT-equivalence is a congruence for the refinement operator \( \text{Ref}^e \), i.e. \( \mathcal{E} \sim_{ICT} \mathcal{E}' \land \forall a \in A : \theta(a) \sim_{ICT} \theta'(a) \) implies that \( \text{Ref}^e_A(\mathcal{E}, \theta) \sim_{ICT} \text{Ref}^e_A(\mathcal{E}', \theta') \). Moreover, it is also a congruence for the operators \( \sim, \oplus, \ominus, \triangleright, \trianglerighteq, \bowtie \), and \( \leq \), which are defined in Definition 3.17.

**Proof:** The proof is given in Subsection 4.4.1. \( \square \)

We are also interested in a congruence which implies strong bisimilarity (Definition 4.6). ICT-equivalence does not yield such an equivalence.

**Lemma 4.11** Strong bisimilarity does not follow from ICT-equivalence.

**Proof:** The cbes \( \mathcal{E}'^h_1 \) and \( \mathcal{E}'^h_2 \) from Figure 4.7 are not bisimilar but ICT-equivalent. \( \square \)

### 4.2.3 UI-Bisimilarity on CBES

An equivalence that is derived from bisimulation equivalence and that is a congruence for the end-based refinement operator has to relate the initial events, as it is done by ICT-equivalence. Therefore, we extend the definition of a bisimulation relation by a third component which denotes a bijection between the initial events.

**Definition 4.12 (UI-Bisimulation)** A unique initial bisimulation (UI-bisimulation) \( \mathcal{R} \) is a subset of \( \text{CBES} \times \text{CBES} \times (\mathcal{U} \rightarrow \mathcal{U}) \) such that whenever \( (\mathcal{E}_1, \mathcal{E}_2, f) \in \mathcal{R} \), then

- \( \text{dom}(f) = \text{init}_{\text{Obs}}(\mathcal{E}_1) \),
- \( f \) is a labeling-preserving bijection between \( \text{init}_{\text{Obs}}(\mathcal{E}_1) \) and \( \text{init}_{\text{Obs}}(\mathcal{E}_2) \),
- \( e_1 \in \text{init}(\mathcal{E}_1) \) implies that there is \( e_2 \) and \( f' \) such that \( l_1(e_1) = l_2(e_2) \) and \( f \cup f' \) is an injective function and \( (\mathcal{E}_1[e_1]_1, \mathcal{E}_2[e_2]_1, f') \in \mathcal{R} \) and \( l_1(e_1) \in \text{Obs} \Rightarrow e_2 = f(e_1) \)
- \( e_2 \in \text{init}(\mathcal{E}_2) \) implies that there is \( e_1 \) and \( f' \) such that \( l_1(e_1) = l_2(e_2) \) and \( f \cup f' \) is an injective function and \( (\mathcal{E}_1[e_1]_1, \mathcal{E}_2[e_2]_1, f') \in \mathcal{R} \) and \( l_2(e_2) \in \text{Obs} \Rightarrow e_2 = f(e_1) \)

We say that \( \mathcal{E}_1, \mathcal{E}_2 \) are UI-bisimilar (or UI-equivalent), denoted by \( \mathcal{E}_1 \sim_{UI} \mathcal{E}_2 \), if and only if there is a UI-bisimulation \( \mathcal{R} \) and an \( f : \mathcal{U} \rightarrow \mathcal{U} \) such that \( (\mathcal{E}_1, \mathcal{E}_2, f) \in \mathcal{R} \).

The event structures from Figure 4.7 are not UI-equivalent, whereas the event structures from Figure 4.8 are UI-equivalent.

The condition in Definition 4.12 that \( f \cup f' \) has to be a function ensures that the identification of the initial events of \( \mathcal{E}_1 \) is preserved after the execution, i.e. \( f \upharpoonright (\text{init}_{\text{Obs}}(\mathcal{E}_1) \cap \text{init}_{\text{Obs}}(\mathcal{E}_1[e_1])) = f' \upharpoonright (\text{init}_{\text{Obs}}(\mathcal{E}_1) \cap \text{init}_{\text{Obs}}(\mathcal{E}_1[e_1])) \). The condition that \( f \cup f' \) is an injective function guarantees that an initial event \( e'_1 \) of \( \mathcal{E}_1 \) is kept after the execution if and only if \( f(e'_1) \) is kept after the corresponding execution, i.e. \( e'_1 \in \text{init}_{\text{Obs}}(\mathcal{E}_1) \cap \text{init}_{\text{Obs}}(\mathcal{E}_1[e_1]) \Leftrightarrow f(e'_1) \in \text{init}_{\text{Obs}}(\mathcal{E}_2) \cap \text{init}_{\text{Obs}}(\mathcal{E}_2[e_2]) \). This means that the identification of the initial events of \( \mathcal{E}_2 \) is also preserved after the execution, i.e. \( f^{-1} \upharpoonright (\text{init}_{\text{Obs}}(\mathcal{E}_2) \cap \text{init}_{\text{Obs}}(\mathcal{E}_2[e_2])) = f'^{-1} \upharpoonright (\text{init}_{\text{Obs}}(\mathcal{E}_2) \cap \text{init}_{\text{Obs}}(\mathcal{E}_2[e_2])) \).
4.2. EQUIVALENCEs

**Proposition 4.13** Two UI-equivalent cbes are also strong bisimilar, i.e. $\sim_{UI} \subset \sim_b$.

**Proof:** It follows from the fact that every UI-bisimulation restricted to its first and second component is a bisimulation. $\square$

**Theorem 4.14** UI-equivalence is a congruence for the refinement operator $Ref^e$, i.e. $\mathcal{E} \sim_{UI} \mathcal{E'} \land \forall a \in A : \theta(a) \sim_{UI} \theta'(a)$ implies that $Ref^e_\mathcal{A}(\mathcal{E}, \theta) \sim_{UI} Ref^e_\mathcal{A}(\mathcal{E'}, \theta')$. Moreover, it is also a congruence for the operators $\leq\land\lor\ltwo\rtwo\lra\lra\lra_{A}$ and $\lra_{A}$, which are defined in Definition 3.17.

**Proof:** The proof is given in Subsection 4.4.1. $\square$

### 4.2.4 FUI-Bisimilarity on CBES

UI-equivalence has to preserve the correspondence of all initial events. This condition is not necessary in order to obtain a congruence that is contained in strong bisimilarity\(^2\). It is sufficient to guarantee that the correspondence of any finite subset of the initial events is preserved. This is formalized by the following definition.

**Definition 4.15 (FUI-Bisimulation)** A finite unique initial bisimulation (FUI-bisimulation) $\mathcal{R}$ is a subset of $\text{CBES} \times \text{CBES} \times (\mathcal{U} \rightarrow \mathcal{U})$ such that whenever $(\mathcal{E}_1, \mathcal{E}_2, f) \in \mathcal{R}$, then

- $\text{dom}(f) = \text{init}_{\text{Obs}}(\mathcal{E}_1)$
- $f$ is a labeling-preserving bijection between $\text{init}_{\text{Obs}}(\mathcal{E}_1)$ and $\text{init}_{\text{Obs}}(\mathcal{E}_2)$
- $e_1 \in \text{init}(\mathcal{E}_1) \land I \in \mathcal{P}_{\text{fin}}(\text{init}_{\text{Obs}}(\mathcal{E}_1))$ implies that there exists $e_2$ and $f'$ such that $l_1(e_1) = l_2(e_2)$ and $(\mathcal{E}_1|e_1, \mathcal{E}_2|e_2, f') \in \mathcal{R}$ and $l_1(e_1) \in \text{Obs} \Rightarrow e_2 = f(e_1)$ and $f \upharpoonright (I \cap \text{init}_{\text{Obs}}(\mathcal{E}_1|e_1)) = f' \upharpoonright I$ and $f^{-1} \upharpoonright (f(I) \cap \text{init}_{\text{Obs}}(\mathcal{E}_2|e_2)) = f'^{-1} \upharpoonright f(I)$
- $e_2 \in \text{init}(\mathcal{E}_2) \land I \in \mathcal{P}_{\text{fin}}(\text{init}_{\text{Obs}}(\mathcal{E}_2))$ implies that there exists $e_2$ and $f'$ such that $l_1(e_1) = l_2(e_2)$ and $(\mathcal{E}_1|e_1, \mathcal{E}_2|e_2, f') \in \mathcal{R}$ and $l_2(e_2) \in \text{Obs} \Rightarrow e_2 = f(e_1)$ and $f \upharpoonright (f^{-1}(I) \cap \text{init}_{\text{Obs}}(\mathcal{E}_1|e_1)) = f' \upharpoonright f^{-1}(I)$ and $f^{-1} \upharpoonright (I \cap \text{init}_{\text{Obs}}(\mathcal{E}_2|e_2)) = f'^{-1} \upharpoonright I$

We say that $\mathcal{E}_1, \mathcal{E}_2$ are FUI-bisimilar (or FUI-equivalent), denoted by $\mathcal{E}_1 \sim_{FUI} \mathcal{E}_2$, if and only if there is a FUI-bisimulation $\mathcal{R}$ and an $f : \mathcal{U} \rightarrow \mathcal{U}$ such that $(\mathcal{E}_1, \mathcal{E}_2, f) \in \mathcal{R}$.

\(^2\)Nevertheless, UI-equivalence is of interest if infinite events can be executed at a single execution step.
CHAPTER 4. MODELING THE END-BASED VIEW IN CBES

Remark 4.16 Obviously, any cbes with a finite set of events is UI-equivalent if and only if it is FUI-equivalent. Furthermore, UI-equivalence differs from FUI-equivalence, since the event structures from Figure 4.9 are FUI-equivalent but not UI-equivalent.

Proposition 4.17 Two FUI-equivalent cbes are also strong bisimilar, i.e. $\sim_{FU1} \subset \sim_b$.

Proof: It follows from the fact that every FUI-bisimulation restricted to its first and second component is a bisimulation. \hfill \Box

Theorem 4.18 FUI-equivalence is a congruence for the refinement operator $\text{Ref}^e$, i.e. $\mathcal{E} \sim_{FU1} \mathcal{E}' \land \forall a \in A : \theta(a) \sim_{FU1} \theta'(a)$ implies that $\text{Ref}^e_A(\mathcal{E}, \theta) \sim_{FU1} \text{Ref}^e_A(\mathcal{E}', \theta')$. Moreover, it is also a congruence for the operators $\wedge, \wedge, \wedge, \wedge, \parallel_A$ and $\parallel_A$, which are defined in Definition 3.17.

Proof: The proof is given in Subsection 4.4.1. \hfill \Box

Remark 4.19 In Theorem 4.10, Theorem 4.14 and Theorem 4.18 the general parallel operator is allowed, i.e. no restriction on the synchronization set is made. But this operator has to be handled carefully, since it is not clear if it matches the intuitive meaning for an end-based setting: It is reasonable that the parallel operator introduces some start-based choices. For example, we expect that the expression $(a + a) \parallel \{a\}a.0$ can only start one $a$-action whereas its corresponding event structure, which is isomorphic to $\mathcal{E}^+$ depicted in Figure 4.2, can start two $a$-actions.

This problem can be solved by the requirement demanding that actions which can be potentially refined must not appear in the synchronization set. A solution where no restriction on the synchronization set is necessary is obtained if one considers start-based together with end-based disabling in a single setting, which is done in Chapter 7.

4.2.5 Comparison of Equivalences

First, the connection between ICT-equivalence, UI-equivalence and FUI-equivalence is presented.
4.2. EQUIVALENCES

\[ \sim_{UI} \supset \sim_{FUI} \supset \sim_{ICT} \]

Figure 4.10: Relations Between the Equivalences

**Proposition 4.20** If two cbes are UI-equivalent, then they are also FUI-equivalent and if two cbes are FUI-equivalent, then they are also ICT-equivalent, i.e.

\[ \sim_{UI} \subset \sim_{FUI} \subset \sim_{ICT} \]

**Proof:** The proof is given in Subsection 4.4.2.

From Proposition 4.17 and Lemma 4.11, we obtain that the second inclusion in Proposition 4.20 is strict. The strictness of the first inclusion follows from Remark 4.16. From Lemma 4.11 and from the fact that \( a \) and \( a + a \) are bisimilar but not ICT-equivalent, we obtain that ICT-equivalence is not comparable with strong bisimilarity. Furthermore, all equivalence notions from Section 4.2.1 can not be contained in \( \sim_{ICT} \). Consequently, they can not be contained in \( \sim_{UI} \) or in \( \sim_{FUI} \), since they identify \( a \) and \( a + a \). All connection between the equivalences that have been introduced is summarized in Figure 4.10: If two equivalences are connected via a line, then the lower one identifies more elements than the upper one.

4.2.6 Coarsest Congruence

In this subsection, we define the coarsest congruence for the refinement operator \( \text{Ref}^e \) with respect to strong bisimilarity. It is different from FUI-equivalence, i.e. FUI-equivalence is not the coarsest congruence with respect to strong bisimilarity. Furthermore, ICT-equivalence fails to be the coarsest congruence with respect to trace equivalence.

**Definition 4.21** Define \( \sim_c \subseteq \text{CBES} \times \text{CBES} \) by \( \mathcal{E}_1 \sim_c \mathcal{E}_2 \) if and only if \( \forall A \subseteq \text{Obs} : \forall \theta : A \rightarrow \text{CBES} : \text{Ref}^e_A(\mathcal{E}_1, \theta) \sim_b \text{Ref}^e_A(\mathcal{E}_2, \theta) \).

**Proposition 4.22** The relation \( \sim_c \) is the coarsest congruence for the refinement operator \( \text{Ref}^e \) with respect to \( \sim_b \).

**Proof:** The proof is given in Subsection 4.4.3.

Unfortunately, FUI-equivalence is a proper subset of \( \sim_c \), i.e. FUI-equivalence is not the coarsest congruence for \( \text{Ref}^e \). This is illustrated by the following example.
Example 4.23 Consider $\mathcal{E}^{c_1}$ and $\mathcal{E}^{c_2}$ from Figure 4.11. They are not ICT-equivalent. Therefore, they are not FUI-equivalent either; since after executing action $a$, the number of the initial actions does not coincide. But $\mathcal{E}^{c_1} \sim_c \mathcal{E}^{c_2}$, which can be seen as follows. If $a$ is not refined, then both cbes, which are bisimilar, keep unchanged under the refinement. Now suppose that $a$ is refined. Then we do not have a problem to find a corresponding bisimilar process for every execution step as long as the refinement of $a$ does not terminate. When the refinement terminates, the process executes $\tau$. If $\mathcal{E}^{c_1}$ executes this $\tau$, then $\mathcal{E}^{c_2}$ can execute its $\tau$ shown on the left to yield a bisimilar cbes. If $\mathcal{E}^{c_2}$ executes this $\tau$, then $\mathcal{E}^{c_1}$ can execute its $\tau$ shown on the right to yield a bisimilar cbes.

In addition, ICT-equivalence is not the coarsest congruence for $\text{Ref}^e$ with respect to $\sim_i$, since $\sim_c$ is a congruence for $\text{Ref}^e$ and $\sim_c \subseteq \sim_b \subseteq \sim_i$ but $\sim_c \not\subseteq \sim_{ICT}$ by Example 4.23.

4.3 Discussion

The reason why $\sim_{ICT}$ and $\sim_{FU}$ fail to be the coarsest congruences for $\text{Ref}^e$ stems from the fact that $\text{Ref}^e$ renames events labeled by $\sqrt{}$ to $\tau$. This renaming is necessary for the well-definedness of this operator, since a definition that removes these events will not result in an element of CBES or will not respect the intuitive meaning.

Nevertheless, it seems reasonable to have a refinement operator without such a relabeling. This corresponds to the philosophy that the ‘final’ executed action terminates the process [14, 28]. This kind of action refinement operator can only be defined in event structures where it is possible that sets of events rather than single events can disable other events. Therefore, we first
establish such a kind of event structures in Chapter 5 before we examine process algebras with an end-based choice operator. In Chapter 5, the new event structures are considered only in the context of disruption. The action refinement operator with the end-based choice view is applied on the new event structures in Chapter 6.

4.4 Proofs

4.4.1 Proof of the Congruence Results

We introduce an event-based refinement, which is used to verify Theorem 4.10. This refinement differs from \( \text{Ref}^e \), because it assigns event structures to each event and not only to action names.

**Definition 4.24** \( \text{Ref}^e_A : \text{CBES} \times (U \rightarrow \text{CBES}) \rightarrow \text{CBES} \) with \( \text{Ref}^e_A (E, \vartheta) = (\tilde{E}, \tilde{\vartheta}, \tilde{\sim}, \tilde{\rightarrow}, \tilde{l}) \) where

\[
\tilde{E} = \{ (e, \hat{e}) \mid e \in E \land l(e) \in A \land \hat{e} \in E_{\vartheta(e)} \} \cup \\
\{ (e, e) \in E \times E \mid l(e) \not\in A \}
\]

\[
\tilde{\sim} = \{ ((e_1, \hat{e}_1), (e_2, \hat{e}_2)) \mid (e_1 \sim e_2 \land (l(e_2) \in A \Rightarrow \hat{e}_2 \in \text{init}(\vartheta(e_2)))) \lor \\
(e_1 = e_2 \land l(e_1) \in A \land \hat{e}_1 \neq \hat{e}_2 \land (\hat{e}_1 \sim_{\vartheta(e_1)} \hat{e}_2 \lor \hat{e}_2 \in \text{exit}(\vartheta(e_1)))) \}
\]

\[
\tilde{\rightarrow} = \{ ((e, X'), (e, \hat{e})) \mid l(e) \in A \land X' \mapsto_{\vartheta(e)} \hat{e} \} \cup \\
\{ ((X, (e, \hat{e})) \mid \forall X : X \mapsto e \land (l(e) \in A \Rightarrow \hat{e} \in \text{init}(\vartheta(e))) \land \\
X = \{ (e, \hat{e})' \in \tilde{E} \mid e \in X \land l(e) \in A \Rightarrow \hat{e}' \in \text{exit}(\vartheta(e)) \} \}
\]

\[
\tilde{l}(e, \hat{e}) = \begin{cases} 
\text{l}_{\vartheta(e)}(\hat{e}) & \text{if } l(e) \in A \land l_{\vartheta(e)}(\hat{e}) \neq \sqrt{ } \\
\tau & \text{if } l(e) \in A \land l_{\vartheta(e)}(\hat{e}) = \sqrt{ }
\end{cases}
\]

The advantage of \( \text{Ref}^e_A \) is that event execution of \( \text{Ref}^e (E, \vartheta) \) can be reduced to the event execution of \( E \) and \( \vartheta \), as it is shown in the following lemma.

**Lemma 4.25** Suppose \( E \in \text{CBES} \) and \( \vartheta : U \rightarrow \text{CBES} \). Then

\[
\text{Ref}^e_A (E, \vartheta) \mid_{[e, \hat{e}]} \simeq \begin{cases} 
\text{Ref}^e_A (E[e], \vartheta) & \text{if } l(e) \not\in A \land e = \hat{e} \\
\text{Ref}^e_A (E[e], \vartheta[e \mapsto \vartheta(e)[e]]) & \text{if } l_{\vartheta(e)}(\hat{e}) = \sqrt{ } \land l(e) \in A \\
\text{Ref}^e_A (E, \vartheta[e \mapsto \vartheta(e)[e]]) & \text{if } l_{\vartheta(e)}(\hat{e}) \neq \sqrt{ } \land l(e) \in A
\end{cases}
\]

Furthermore, \( \text{Ref}^e_A (E[e], \vartheta[e \mapsto E']) \simeq \text{Ref}^e_A (E[e], \vartheta) \) holds for any \( E' \in \text{CBES} \).

**Proof:** Straightforward. \( \square \)

In order to simplify the proof of Theorem 4.10 we introduce a variant of the initial corresponding traces:

**Definition 4.26** Let \( E, E' \in \text{CBES} \). Then define \( \tilde{T}^{ic} (E) = \{ ((e_i, \gamma_i))_{i \leq n-1}, \gamma_n) \mid n \in \mathbb{N} \land \\
\exists \gamma_0, \ldots, \gamma_n : E_0 = E \land \forall i \leq n-1 : E_{i+1} = E_i \land \forall j \leq n : \gamma_j \in \mathcal{P}_{\text{fin}}(\text{init}_{\text{Obs}}(E_j)) \} \).

Furthermore, define \( E \simeq_{ ICT } E' \) if
for every \((e_i, \gamma_i)_{i \leq n} \in \hat{T}^{ic}(E)\) there is an injective, labeling-preserving function \(f : (\gamma_n \cup \bigcup_{i \leq n-1}(\gamma_i \cup \{e_i\})) \rightarrow E'\) such that \(((f(e_i), f(\gamma_i))_{i \leq n}, f(\gamma_n)) \in \hat{T}^{ic}(E')\)

- and symmetrically as in Definition 4.8.

**Lemma 4.27** The initial corresponding trace equivalence coincides with the equivalence defined in Definition 4.26, i.e. \(\sim_{ICT} = \approx_{ICT}\).

**Proof:** Suppose \((e_i, \gamma_i)_{i \leq n} \in T^{ic}(E)\) and \(E \approx_{ICT} E'\). Then \(((e_i, \gamma_i)_{i \leq n}, \emptyset) \in \hat{T}^{ic}(E)\), and so there is \(f : \left(\bigcup_{i \leq n}(\gamma_i \cup \{e_i\})\right) \rightarrow E'\) such that \(((f(e_i), f(\gamma_i))_{i \leq n}, \emptyset) \in \hat{T}^{ic}(E')\). This implies \((f(e_i), f(\gamma_i))_{i \leq n} \in T^{ic}(E')\), as required.

Now suppose \(((e_i, \gamma_i)_{i \leq n-1}, \gamma_n) \in \hat{T}^{ic}(E)\) and \(E \sim_{ICT} E'\). We proceed by making a case analysis:

\(\gamma_n = \emptyset\): Similar to the above reasoning.

\(\gamma_n \neq \emptyset\): Let \(e_n \in \gamma_n\). Then \((e_i, \gamma_i)_{i \leq n} \in T^{ic}(E)\) and so there is a function \(f : \left(\bigcup_{i \leq n}(\gamma_i \cup \{e_i\})\right) \rightarrow E'\) such that \(((f(e_i), f(\gamma_i))_{i \leq n} \in T^{ic}(E')\). This implies \(((f(e_i), f(\gamma_i))_{i \leq n-1}, \gamma_n) \in T^{ic}(E')\) which completes the proof, since \(\bigcup_{i \leq n}(\gamma_i \cup \{e_i\}) = \gamma_n \cup \bigcup_{i \leq n-1}(\gamma_i \cup \{e_i\})\). \(\square\)

**Proof of Theorem 4.10:** Here, we only present the proof for the refinement operator \(Ref^e\), since the other cases are easier.

Let \(\vartheta\) be a function such that \(\forall e \in E : l(e) \in A \Rightarrow \vartheta(e) = \theta(l(e))\), and let \(\vartheta'\) be a function such that \(\forall e' \in E' : l'(e') \in A \Rightarrow \vartheta'(e') = \theta'(l'(e'))\). Obviously, \(Ref^e_A(E, \vartheta) = Ref^e_A(E', \vartheta')\) and \(Ref^e_A(E', \vartheta') = Ref^e_A(E', \vartheta)\). For simplicity, we write \(\hat{E}\) for \(Ref^e_A(E, \vartheta)\) and \(\hat{E}'\) for \(Ref^e_A(E', \vartheta')\) respectively.

Suppose \(((e_i, \hat{e}_i, \gamma_i)_{i \leq n} \in T^{ic}(\hat{E})\). The case in which an element of \(T^{ic}(E')\) is taken follows by symmetrical arguments. As an immediate consequence of Lemma 4.25 we know that there exist \(E_0, ..., E_{n+1}\) and \(\vartheta_0, ..., \vartheta_{n+1}\) such that for \(i \leq n\) we have

\[
E_0 = E \land \vartheta_0 = \vartheta \land e_i \in \text{init}(E_i) \land (l_i(e_i) \in A \Rightarrow \hat{e}_i \in \text{init}(\vartheta_i(e_i))) \land (l_i(e_i) \notin A) \Rightarrow (E_0[E_i] = E_{i+1} \land \vartheta_i = \vartheta_{i+1}) \land (l_i(e_i) \in A \land l_{\vartheta_i(e_i)}(\hat{e}_i) = \sqrt{\vartheta}) \Rightarrow (E_0[E_i] = E_{i+1} \land \vartheta_i(e_i) \rightarrow \vartheta_{i+1}(e_i)) \land ((l_i(e_i) \in A \land l_{\vartheta_i(e_i)}(\hat{e}_i) \neq \sqrt{\vartheta}) \Rightarrow (E_i = E_{i+1} \land \vartheta_i(e_i) \rightarrow \vartheta_{i+1}(e_i)))
\]

Define \(I = \{i \in \{0, \ldots, n\} \mid l(e_i) \notin A \lor l_{\vartheta_i(e_i)}(\hat{e}_i) = \sqrt{\vartheta}\}\). Assume \(\{k_0, ..., k_{|I|}\} = I\) and \(k_i < k_i+1\). Then we have \(E_{k_0} = E\) and \(E_{k_{|I|+1}} = E_{k_{|I|+1}}\). Furthermore, define \(\gamma_{k_{|I|+1}} = (\bigcup_{i > k_{|I|+1}} \gamma_{\pi_i(e_i) \cup \{e_i\}}) \cap \text{init(Obs}_E)\) for \(j \leq |I|\) where \(k_{|I|} = -1\) and \(k_{|I|+1} = n + 1\).

Hence, \(((e_{k_i}, \gamma_{k_i})_{j \leq |I|, n+1} \in \hat{T}^{ic}(E)\). From \(E \sim_{ICT} E'\) and Lemma 4.27 we know that there exists an injective, labeling preserving function \(f : (\gamma_{n+1} \cup \bigcup_{j \leq |I|}(\gamma_{k_j} \cup \{e_{k_j}\})) \rightarrow E'\) such that \(((f(e_{k_j}), f(\gamma_{k_j}))_{j \leq |I|}, f(\gamma_{n+1})) \in \hat{T}^{ic}(E')\).

Now define \(I^e = \{j \in \{0, \ldots, n\} \mid e = e_j\}\) for \(e \in E\) with \(l(e) \in A\). Let \(\{k_0^e, ..., k_{|I^e|}\} = I^e\) with \(k_i^e < k_i^e+1\). Define \(\gamma_{k_j}^e = \bigcup_{i > k_j} \{e_i \mid (e, \hat{e}_i) \in \gamma_i \land l_{\vartheta_i(e_i)}(\hat{e}_i) \in \text{Obs}\}\) for \(j \leq |I^e|\), where
4.4. PROOFS

\[ k^e_{n+1} = -1 \text{ and } k^e_{|I|} = n + 1. \] Furthermore, we have \( \theta(l(e)) = \theta(l(e)) \) and \( \theta(l(e)) = \theta(l(e)) \).

Hence, \((\hat{e}^e_{k_j}, \gamma^e_{k_j})_{j < |I|}, \gamma^e_{|I|+1}) \in T_{ic}(\theta(l(e))).\) From \( \theta(l(e)) \sim_{icT} \theta(l(e)) \) and Lemma 4.27 we know that there exists an injective, labeling preserving \( f^e : (\gamma^e_{n+1} \cup \bigcup_{j < |I|} (\gamma^e \cup \{ \hat{e}^e_{k_j} \})) \to E_{ic}(\theta(l(e))) \) such that \((f^e(\hat{e}^e_{k_j}), f^e(\gamma^e_{k_j}))_{j < |I|}, f^e(\gamma^e_{|I|+1})) \in T_{ic}(\theta(l(e))).\)

Define \( \tilde{f} : (\bigcup_{i \leq n} (\gamma_i \cup \{(e_i, \hat{e}_i)\})) \to \tilde{E}' \) as follows:

\[
\tilde{f}(e, \hat{e}) = \begin{cases} (f(e), f(e)) & \text{if } l(e) \notin A \\ (f(e), f(e)) & \text{otherwise} \end{cases}
\]

Then it is easily seen that \( \tilde{f} \) is an injective and labeling-preserving function. Furthermore, define \( E'_0 = E', \hat{\theta}'_0 = \theta' \) and

\[
E'_{i+1} = \begin{cases} E'_{i[f(e_i)]} & \text{if } l'(f(e_i)) \notin A \lor l'(f(e_i))(f^e_i(\hat{e}_i)) = \checkmark \\ E'_i & \text{otherwise} \end{cases}
\]

\[
\hat{\theta}'_{i+1} = \begin{cases} \hat{\theta}'_i & \text{if } l'(f(e_i)) \notin A \\ \hat{\theta}'_i[f(e_i) \to \hat{\theta}'_i(f(e_i))_{f(e_i)}] & \text{otherwise} \end{cases}
\]

\( E'_j \) and \( \hat{\theta}'_j \) are well-defined, because \((f^e(\hat{e}^e_{k_j}), f^e(\gamma^e_{k_j}))_{j < |I|}, f^e(\gamma^e_{|I|+1})) \in T_{ic}(\theta'(l(e))) \) and \((f(e_k), f(\gamma_k))_{j < |I|}, f(\gamma_{|I|+1})) \in \tilde{T}_{ic}(\tilde{E}') \). More precisely, for any \( j \) and any \( e \in E \) with \( l(e) \in A \), we have \( E'_{k_j+1} = E'_{k_j[f(e_k)]} \) and \( \hat{\theta}'_{k_j+1}(f(e)) = \hat{\theta}'_{k_j}(f(e))_{f(e_k)} \).

Furthermore, define \( \tilde{E}'_i = \text{Ref}^{\tilde{E}'}_A(\tilde{E}'_i, \hat{\theta}'_i) \) for \( i \leq n + 1 \). Then \( \tilde{E}'_0 = \tilde{E}' \) and \( \tilde{E}'_{i+1} = \tilde{E}'_{i[f(e_i), \hat{e}_i]} \), which follows by Lemma 4.25.

Suppose \( (e, \hat{e}) \in \gamma_i \). Let \( m \in \{0, \ldots, |I|\} \) such that \( k_{m-1} < i \leq k_m \). Hence, \( e \in \gamma_m \), which implies \( f(e) \in f(\gamma_m) \subseteq \text{init}(E'_m) \). And so from the definition of \( E'_j \) we get \( f(e) \in \text{init}(E'_j) \), since \( k_{m-1} < i \leq k_m \). Furthermore, if \( l(e) \in A \), then take \( \tilde{m} \leq |I| \) such that \( k^e_{\tilde{m}-1} < i \leq k^e_{\tilde{m}} \). Hence, \( \hat{e} \in \gamma_{\tilde{m}} \), which implies \( f^e(\hat{e}) \in f^e(\gamma_{\tilde{m}}) \subseteq \text{init}(\hat{\theta}'_{k^e_{\tilde{m}}}(f(e))) \). And so from the definition of \( \hat{\theta}'_j \) we get \( f^e(\hat{e}) \in \text{init}(\hat{\theta}'_j(f(e))) \). Therefore, \( \tilde{f}(e, \hat{e}) \in \text{init}(\tilde{E}'_j) \).

Thus, we have shown \( (\tilde{f}(e_i, \hat{e}_i), \hat{f}(\gamma_i))_{i \leq n} \in T_{ic}(\tilde{E}') \), which completes this proof. \( \square \)

**Proof of Theorem 4.14:** We only present here the proof for the refinement operator \( \text{Ref}^e \), since the other cases are easier.

Let \( R \) and \( R_a \) be unique initial bisimulations such that \( (\theta(a), \theta'(a), g_a) \in R_a \) and \( (E, E', g) \in R \). Then define

\[
R_{\text{Ref}} = \{(\text{Ref}_A^e(\tilde{E}), \tilde{\theta}), \text{Ref}_A^e(\tilde{E}', \hat{\theta}') \mid \exists f : (\tilde{E}, \tilde{E}', f) \in R \land \\
(\forall e \in \tilde{E} : (l(e) \in A \land \hat{e} \notin \text{init}(\tilde{E})) \Rightarrow \tilde{\theta}(e) = \theta(l(e))) \land \\
(\forall e' \in \tilde{E}' : (l'(e') \in A \land e' \notin \text{init}(\tilde{E}')) \Rightarrow \hat{\theta}'(e') = \theta'(l'(e'))) \land \\
\forall e \in \tilde{E} : \exists f : (e \in \text{init}_A(\tilde{E}) \Rightarrow (\tilde{\theta}(e), \hat{\theta}'(f(e)), f_e) \in R_{l(e)}) \land \\
\tilde{f}(e, \hat{e}) \simeq \{(f(e), f(e)) \text{ if } e \in \text{init}(\tilde{E}) \land l(e) \in \text{Obs} \setminus A \\
(f(e), f_e(e)) \text{ if } e \in \text{init}(\tilde{E}) \land \hat{e} \in \text{init}_{\text{Obs}}(\tilde{\theta}(e)) \}
\]
Suppose \((\text{Ref}_A^e(\tilde{E}, \tilde{d}), \text{Ref}_A^e(\tilde{E}', \tilde{d}'), \tilde{f}) \in \mathcal{R}_{\text{Ref}}\) and let \(f, f_e\) be the corresponding functions.

Obviously, \(\tilde{f}\) is always an isomorphism between \(\text{init}_{\text{Obs}}(\text{Ref}_A^e(\tilde{E}, \tilde{d}))\) and \(\text{init}_{\text{Obs}}(\text{Ref}_A^e(\tilde{E}', \tilde{d}'))\) in the above definition. Furthermore, \(\tilde{f}\) is labeling-preserving.

In the following, we will show that \(\mathcal{R}_{\text{Ref}}\) is a UI-bisimulation. Therefore, suppose \((e, \hat{e}) \in \text{init}(\text{Ref}_A^e(\tilde{E}, \tilde{d}))\). We proceed by making a case analysis.

Let \(f_{e'}^f = \{\begin{array}{ll} g_a & \text{if } \tilde{l}(e) = a \land e' \notin \text{init}(\tilde{E}) \\ f_{e'} & \text{otherwise} \end{array}\) Define \(\tilde{f}\) as follows

\[
\tilde{f}(e', e'') \simeq \begin{cases} (f'(e'), f'(e'')) & \text{if } e' \in \text{init}(\tilde{E}) \land \tilde{l}(e') \in \text{Obs}\setminus A \\ (f'(e'), f'(e'')) & \text{if } e' \in \text{init}_A(\tilde{E}) \land e'' \in \text{init}_{\text{Obs}}(\tilde{d}(e')) \end{cases}
\]

Then \(\tilde{f}'\) and \(\tilde{f}\) coincide on \(\text{init}_{\text{Obs}}(\text{Ref}_A^e(\tilde{E}, \tilde{d})) \cap \text{init}_{\text{Obs}}(\text{Ref}_A^e(\tilde{E}', \tilde{d}')(e'|e'))\), since we have \(f \upharpoonright (\text{init}_{\text{Obs}}(\tilde{E}) \cap \text{init}_{\text{Obs}}(\tilde{E}')(e'|e')) = f' \upharpoonright (\text{init}_{\text{Obs}}(\tilde{E}) \cap \text{init}_{\text{Obs}}(\tilde{E}')(e'|e'))\). Thus, \(\tilde{f}' \cup \tilde{f}\) is a function. Suppose \((\tilde{f}' \cup \tilde{f})(e', e') = (\tilde{f}' \cup \tilde{f})(e'', e'')\). Then \((\tilde{f}' \cup \tilde{f})(e') = (\tilde{f}' \cup \tilde{f})(e'')\) and therefore \(e' = e''\). The equality of \(e'\) and \(e''\) can now be easily derived from the injectivity of \(\tilde{f}'\) or of \(\tilde{f}\). Thus, \(\tilde{f}' \cup \tilde{f}\) is an injective function.

Suppose \(\tilde{l}(e') \in A\), then \(e' \notin \text{init}(\tilde{E}) \Rightarrow \tilde{d}(e') = \theta(\tilde{l}(e'))\). Additionally, we conclude from \(e' \notin \text{init}(\tilde{E})\) that \(f'(e') \notin \text{init}(\tilde{E}'),\) since \(f' \cup f\) is injective. And so we get \(e' \notin \text{init}(\tilde{E}) \Rightarrow \tilde{d}(f'(e')) = \theta(\tilde{l}(f'(e'))).\) Thus, \(e' \in \text{init}(\tilde{E}) \land e' \in \text{init}_A(\tilde{E}'), \tilde{f}' \in \mathcal{R}_{\text{Ref}}\) by definition.

Moreover, \(\tilde{l}(e') \notin A\) implies \(\tilde{l}(e') \notin A\). From Lemma 4.25 we obtain that \(\text{Ref}_A^e(\tilde{E}', \tilde{d}') = \text{Ref}_A^e(\tilde{E}', \tilde{d}')(e'|e')\). Furthermore, \(l_{\text{Ref}_A^e(\tilde{E}, \tilde{d})}(e, \hat{e}) \in \text{Obs} \Rightarrow \tilde{f}(e, \hat{e}) = (f(e), f(e)) = (\hat{e}, \hat{e})\).

Now consider the case when \(\tilde{l}(e) \in A \land l_{\tilde{d}(e)}(e') = \sqrt{\land} \in \text{init}(\tilde{d}(e))\). Then we have \((\tilde{d}(e), \tilde{d}(f(e)), f_*) \in \mathcal{R}_{\tilde{l}(e)}\). Hence, there exists \(e' \in \text{init}(\tilde{d}(f(e)))\) such that \(l_{\tilde{d}(f(e))}(e') = \sqrt{\land}\). Thus \(\text{Ref}_A^e(\tilde{E}', \tilde{d}') = \text{Ref}_A^e(\tilde{E}', \tilde{d}'_f(e)) \quad \tilde{d}'_f(e) \rightarrow \tilde{d}'(f(e)|e) = \text{Ref}_A^e(\tilde{E}', \tilde{d}'_f(e)|e)\) by Lemma 4.25. This completes the case, since \(l_{\text{Ref}_A^e(\tilde{E}, \tilde{d})}(e, \hat{e}) \notin \text{Obs}\).

\[
\tilde{l}(e) \in A \land l_{\tilde{d}(e)}(e') \neq \sqrt{\land}: \text{ By Lemma 4.25 we get } \text{Ref}_A^e(\tilde{E}, \tilde{d})(e|e) = \text{Ref}_A^e(\tilde{E}, \tilde{d}[e \rightarrow \tilde{d}(e)|e]).
\]

Furthermore, \((\tilde{d}(e), \tilde{d}(f(e)), f_*) \in \mathcal{R}_{\tilde{l}(e)}\). And so there is \(e' \in \text{init}(\tilde{d}(f(e)))\) such that \(l_{\tilde{d}(f(e))}(e') = l_{\tilde{d}(f(e))}(e'')\) and \(l_{\tilde{d}(f(e))}(e') \in \mathcal{R}_{\tilde{l}(e)}\) and \(l_{\tilde{d}(f(e))}(e') \in \text{Obs} \Rightarrow \hat{e} = f_*(e')\) and \(f_* \cup f_*\) is an injective function. Define \(\tilde{f}'\) by

\[
\tilde{f}'(e', e'') \simeq \begin{cases} (f(e'), f(e'')) & \text{if } e' \in \text{init}_{\text{Obs}}(\tilde{E}) \land \tilde{l}(e') \notin A \\ (f(e'), f(e'')) & \text{if } e' \in \text{init}_A(\tilde{E}) \land e'' \in \text{init}_{\text{Obs}}(\tilde{d}(e')) \land e' \neq e \\ (f(e), f(e')) & \text{if } e \in \text{init}_{\text{Obs}}(\tilde{d}(e)) \land e' = e \end{cases}
\]

Then \(\tilde{f}'\) and \(\tilde{f}\) coincide on \(\text{init}_{\text{Obs}}(\text{Ref}_A^e(\tilde{E}, \tilde{d})) \cap \text{init}_{\text{Obs}}(\text{Ref}_A^e(\tilde{E}', \tilde{d}')(e'|e'))\). Thus, \(\tilde{f}' \cup \tilde{f}\) is a function. Suppose \((\tilde{f}' \cup \tilde{f})(e', e') = (\tilde{f}' \cup \tilde{f})(e'', e'').\) Then \(f(e') = f(e'')\) by definition,
hence \( e' = e'' \). If \( \tilde{l}(e') \not\in A \), then \( \tilde{e}' = \tilde{e}'' \) immediately follows. If \( \tilde{l}(e') \in A \land e' \neq e \), then \( \tilde{f}(e',\tilde{e}') = \tilde{f}'(e',\tilde{e}') \), and so \( \tilde{e}' = \tilde{e}'' \) follows by the injectivity of \( \tilde{f} \). Now suppose \( \tilde{l}(e') \in A \land e' = e \). Then \((f'_e \cup f_e)(\tilde{e}') = (f'_e \cup f_e)(\tilde{e}'')\), hence \( \tilde{e}' = \tilde{e}'' \). Therefore, \( \tilde{f}' \cup \tilde{f} \) is an injective function.

Thus, \((\overline{\text{Ref}}_A^e(\tilde{E}, \tilde{\vartheta}[e \to \tilde{\vartheta}(e)_{[\tilde{e}]]}], \overline{\text{Ref}}_A^e(\tilde{E}', \tilde{\vartheta}'[f(e) \to \tilde{\vartheta}(f(e))_{[\tilde{e}]]}], \tilde{f}')) \in \mathcal{R}_{\text{Ref}} \) by definition. Moreover, \( \tilde{l}(e) \in A \land \tilde{l} \tilde{\vartheta}(e)(\tilde{e}) \neq \sqrt{\land} \in \text{init}(\tilde{\vartheta}(e)) \) implies that \( \tilde{l}(f(e)) \in A \land \tilde{l} \tilde{\vartheta}(f(e))(\tilde{e}) \neq \sqrt{\land} \). And so by Lemma 4.25 we have \( \overline{\text{Ref}}_A^e(\tilde{E}', \tilde{\vartheta}'[f(e) \to \tilde{\vartheta}(f(e))_{[\tilde{e}]]]) = \overline{\text{Ref}}_A^e(\tilde{E}', \tilde{\vartheta})_{[f(e), \tilde{e}]} \).

This completes the case, since \( l_{\overline{\text{Ref}}_A^e(\tilde{E}, \tilde{\vartheta})}(e, \tilde{e}) \in \text{Obs} \Rightarrow \tilde{f}(e, \tilde{e}) = (f(e), \tilde{e}) \).

The last condition of the U!-bisimulation can be derived by symmetrical arguments. Thus, we proved that \( \mathcal{R}_{\text{Ref}} \) is a unique initial bisimulation.

Obviously, \( \overline{\text{Ref}}_A^e(\mathcal{E}, \tilde{\vartheta}) = \overline{\text{Ref}}_A^e(\mathcal{E}, \theta) \) whenever \( \forall e \in E : l(e) \in A \Rightarrow \tilde{\vartheta}(e) = \theta(l(e)) \). Define

\[
\tilde{g}(e', \tilde{e}') = \begin{cases} (g(e'), g(e')) & \text{if } e' \in \text{init}_{\text{Obs}}(\mathcal{E}) \land l(e') \notin A \\ (g(e'), g(t(e'))(\tilde{e}')) & \text{if } e' \in \text{init}(\mathcal{E}) \land l(e') \in A \land \tilde{e}' \in \text{init}_{\text{Obs}}(\theta(l(e'))) \end{cases}
\]

Then \( \overline{\text{Ref}}_A^e(\mathcal{E}, \theta), \overline{\text{Ref}}_A^e(\mathcal{E}', \theta'), \tilde{g}) \in \mathcal{R}_{\text{Ref}}, \) which completes this proof. \( \square \)

**Proof of Theorem 4.18**: We only present here the proof for the refinement operator \( \text{Ref}^e \), the other cases are easier.

Let \( \mathcal{R} \) and \( \mathcal{R}_a \) be unique initial bisimulations such that \((\theta(a), \theta'(a), g_a) \in \mathcal{R}_a \) and \((\mathcal{E}, \mathcal{E}', g) \in \mathcal{R} \). Then define

\[
\mathcal{R}_{\text{Ref}} = \{(\overline{\text{Ref}}_A^e(\tilde{E}, \tilde{\vartheta}), \overline{\text{Ref}}_A^e(\tilde{E}', \tilde{\vartheta}'), f) \mid \exists \tilde{f} : \exists \tilde{l} \in \mathcal{P}_{\text{fin}}(\text{init}_A(\tilde{E})) : \exists F : U \rightarrow (U \rightarrow U) : (\tilde{E}, \tilde{E}', \tilde{f}) \in \mathcal{R} \land \\
(\forall \tilde{e} \in \tilde{E} \setminus \tilde{l} : (\tilde{l}(\tilde{e}) \in A \Rightarrow \tilde{\vartheta}(\tilde{e}) = \theta(\tilde{l}(\tilde{e})) \land (\tilde{e} \in \text{init}(\tilde{E}) \Rightarrow F(\tilde{e}) = g_{l(\tilde{e}))}) \land \\
(\forall \tilde{e'} \in \tilde{E}' \setminus \tilde{l} : (\tilde{l}'(\tilde{e}') \in A \Rightarrow \tilde{\vartheta}'(\tilde{e}') = \theta'(\tilde{l}'(\tilde{e}'))) \land \\
(\forall \tilde{e} \in \tilde{l} : (\tilde{\vartheta}(\tilde{e}), \tilde{\vartheta}'(\tilde{f}(\tilde{e})), F(\tilde{e})) \in \mathcal{R}_{l(\tilde{e})}) \land \\
(\forall \tilde{e} \in \text{init}(\tilde{E}) : (\tilde{l}(\tilde{e}) \notin A \Rightarrow F(\tilde{e}) = \text{cons}_{l(\tilde{e})}) \land \\
f(e, \tilde{e}) = \begin{cases} (\tilde{f}(e), F(e)) & \text{if } (e, \tilde{e}) \in \text{init}_{\text{Obs}}(\overline{\text{Ref}}_A^e(\tilde{E}, \tilde{\vartheta})) \\ \text{undefined} & \text{otherwise} \end{cases} \}
\]

First, we observe that for all \((\overline{\text{Ref}}_A^e(\tilde{E}, \tilde{\vartheta}), \overline{\text{Ref}}_A^e(\tilde{E}', \tilde{\vartheta}')) \in \mathcal{R}_{\text{Ref}} \) we have: \( f \) is always an isomorphism between \( \text{init}_{\text{Obs}}(\overline{\text{Ref}}_A^e(\tilde{E}, \tilde{\vartheta})) \) and \( \text{init}_{\text{Obs}}(\overline{\text{Ref}}_A^e(\tilde{E}', \tilde{\vartheta}')) \), \( f \) is labeling-preserving and

\[
f^{-1}(e', \tilde{e}') = (\tilde{f}^{-1}(e'), F(\tilde{f}^{-1}(e'))^{-1}(\tilde{e}'))
\]

where \( \tilde{f}, F \) are its corresponding functions. This holds, since \( f(\tilde{f}^{-1}(e'), F(\tilde{f}^{-1}(e'))^{-1}(\tilde{e}')) = (e', \tilde{e}') \).

Suppose \((\overline{\text{Ref}}_A^e(\tilde{E}, \tilde{\vartheta}), \overline{\text{Ref}}_A^e(\tilde{E}', \tilde{\vartheta}'), f) \in \mathcal{R}_{\text{Ref}} \), let \( \tilde{I} \) be the corresponding set and let \( \tilde{f}, F \) be the corresponding functions.

In the following, we will show that \( \mathcal{R}_{\text{Ref}} \) is a FUI-bisimulation. Therefore, suppose \((e, \tilde{e}) \in \text{init}(\overline{\text{Ref}}_A^e(\tilde{E}, \tilde{\vartheta})) \land I \in \mathcal{P}_{\text{fin}}(\text{init}_{\text{Obs}}(\overline{\text{Ref}}_A^e(\tilde{E}, \tilde{\vartheta}))) \). We proceed by making a case analysis.
\( \tilde{l}(e) \notin A: \) Then \( \tilde{e} = e \) and from Lemma 4.25 we get \( \overline{\text{Ref}}_A^e(\tilde{E}, \tilde{o})_{[(e, \tilde{e})]} = \overline{\text{Ref}}_A^e(\tilde{E}[e], \tilde{o}) \).

Define \( \tilde{I}' = \tilde{I} \cup \pi_1(I) \). Since \( (\tilde{E}, \tilde{E}', \tilde{f}) \in \mathcal{R} \land \tilde{I}' \in \mathcal{P}_{\text{fin}}(\text{init}_{\text{Obs}}(\tilde{E})) \), there is \( e' \) and \( \tilde{f}' \) such that \( \tilde{l}(e) = \tilde{l}'(e') \) and \( (\tilde{E}[e], \tilde{E}'[e], \tilde{f}') \in \mathcal{R} \) and \( \tilde{l}(e) \in \text{Obs} \Rightarrow \tilde{f}(e) = e' \) and \( \tilde{f} \upharpoonright (\tilde{I}' \cap \text{init}_{\text{Obs}}(\tilde{E}[e])) = \tilde{f}' \upharpoonright \tilde{I}' \) and \( \tilde{f}^{-1} \upharpoonright (\tilde{f}(\tilde{I}) \cap \text{init}_{\text{Obs}}(\tilde{E}'[e])) = \tilde{f}'^{-1} \upharpoonright \tilde{f}(\tilde{I}) \).

We have \( \tilde{l}(e) \notin A \) implies \( \tilde{l}'(e') \notin A \). And so by Lemma 4.25 we obtain \( \overline{\text{Ref}}_A^e(\tilde{E}'[e], \tilde{o}) = \overline{\text{Ref}}_A^e(\tilde{E}, \tilde{o})_{[(e', \tilde{e}')]} \). Furthermore, the labels of \((e', e')\) and \((e, e)\) coincide and \( l_{\overline{\text{Ref}}_A^e(\tilde{E}, \tilde{o})}(e, e) \in \text{Obs} \Rightarrow f(e, e) = (\tilde{f}(e), \tilde{f}(e)) = (e', e') \).

Now we define \( f' \) and \( f'' \) by

\[
\begin{align*}
F'(e'') &= \begin{cases} 
\text{cons}\tilde{f}(e) & \text{if } l(e'') \notin A \land e'' \in \text{init}(\tilde{E}) \\
F(e) & \text{otherwise}
\end{cases} \\
F''(e'', \tilde{e}'') &= \begin{cases} 
(\tilde{f}'(e''), F'(e'')(\tilde{e}'')) & \text{if } (e'', \tilde{e}'') \in \text{init}_{\text{Obs}}(\overline{\text{Ref}}_A^e(\tilde{E}[e], \tilde{o})) \\
\text{undefined} & \text{otherwise}
\end{cases}
\end{align*}
\]

Then \( \overline{\text{Ref}}_A^e(\tilde{E}[e], \tilde{o}), \overline{\text{Ref}}_A^e(\tilde{E}'[e], \tilde{o}), f', \tilde{f}' \in \mathcal{R}_{\text{Ref}} \) where \( \tilde{I}' \cap \text{init}_{\text{Obs}}(\tilde{E}[e]) \) is its corresponding set and \( \tilde{f}' \), \( \tilde{f}'' \) are its corresponding functions.

Furthermore, from \( \tilde{f} \upharpoonright (\tilde{I}' \cap \text{init}_{\text{Obs}}(\tilde{E}[e])) = \tilde{f}' \upharpoonright \tilde{I}' \) we obtain that \( f' \) and \( f \) coincide on \( I \cap \text{init}_{\text{Obs}}(\overline{\text{Ref}}_A^e(\tilde{E}[e], \tilde{o}))) \).

Suppose \((e'', \tilde{e}'') \in \text{init}_{\text{Obs}}(\overline{\text{Ref}}_A^e(\tilde{E}', \tilde{o})_{[(e', \tilde{e}')]} \cap f(I)) \). Then from (4.1) and from the fact that \( \tilde{f}^{-1} \upharpoonright (f(\tilde{I}) \cap \text{init}_{\text{Obs}}(\tilde{E}'[e])) = \tilde{f}'^{-1} \upharpoonright f(\tilde{I}) \) we obtain the following equation \( f^{-1}(e'', \tilde{e}'') = (\tilde{f}'^{-1}(e'', \tilde{e}''), F'(\tilde{f}^{-1}(e'', \tilde{e}''))^{-1}(\tilde{e}'')) = (\tilde{f}^{-1}(e'', \tilde{e}''), F(\tilde{f}^{-1}(e'')))^{-1}(\tilde{e}'')) = f^{-1}(e'', \tilde{e}'') \), which completes this case.

\( \tilde{l}(e) \in A \land l_{\tilde{o}}(e) \tilde{e} = \sqrt{ } \): By Lemma 4.25 \( \overline{\text{Ref}}_A^e(\tilde{E}, \tilde{o})_{[(e, \tilde{e})]} = \overline{\text{Ref}}_A^e(\tilde{E}[e], \tilde{o}|e \rightarrow \tilde{o}(e)[e]) = \overline{\text{Ref}}_A^e(\tilde{E}[e], \tilde{o}) \).

Define \( \tilde{I}' = \tilde{I} \cup \pi_1(I) \). Since \( (\tilde{E}, \tilde{E}', \tilde{f}) \in \mathcal{R} \land \tilde{I}' \in \mathcal{P}_{\text{fin}}(\text{init}_{\text{Obs}}(\tilde{E})) \), there is \( e' \) and \( \tilde{f}' \) such that \( \tilde{l}(e) = \tilde{l}'(e') \) and \( (\tilde{E}[e], \tilde{E}'[e], \tilde{f}') \in \mathcal{R} \) and \( \tilde{l}(e) \in \text{Obs} \Rightarrow \tilde{f}(e) = e' \) and \( \tilde{f} \upharpoonright (\tilde{I}' \cap \text{init}_{\text{Obs}}(\tilde{E}[e])) = \tilde{f}' \upharpoonright \tilde{I}' \) and \( \tilde{f}^{-1} \upharpoonright (\tilde{f}(\tilde{I}) \cap \text{init}_{\text{Obs}}(\tilde{E}'[e])) = \tilde{f}'^{-1} \upharpoonright \tilde{f}(\tilde{I}) \).

Furthermore, \((\tilde{o}(e), \tilde{o}'(e'), F(e)) \in \mathcal{R}_{\tilde{o}(e)} \) and \( \tilde{e} \in \text{init}(\tilde{o}(e)) \). Hence, there exists \( \tilde{e} \in \text{init}(\tilde{o}) \) such that \( l_{\tilde{o}}(\tilde{e}) = l_{\tilde{o}'(e')}(\tilde{e}') \) and \( l_{\tilde{o}}(\tilde{e}) \in \text{Obs} \Rightarrow F(\tilde{e}) = \tilde{e}' \).

By Lemma 4.25 we get \( \overline{\text{Ref}}_A^e(\tilde{E}'[e], \tilde{o}'(e') \rightarrow \tilde{o}'(e') \rightarrow \tilde{o}'(e')) = \overline{\text{Ref}}_A^e(\tilde{E}, \tilde{o})_{[(e', \tilde{e}')]} \). Furthermore, the labels of \((e', \tilde{e}') \) and \((e, \tilde{e}) \) coincide and \( l_{\overline{\text{Ref}}_A^e(\tilde{E}, \tilde{o})}(e, \tilde{e}) \in \text{Obs} \) implies that \( f(e, \tilde{e}) = (\tilde{f}(e), F(\tilde{e}) \tilde{e}) = (e', e') \).

Now define \( F' \) and \( f' \) as in (4.2) and (4.3). Then \( (\overline{\text{Ref}}_A^e(\tilde{E}, \tilde{o})_{[(e, \tilde{e})]}, \overline{\text{Ref}}_A^e(\tilde{E}', \tilde{o})_{[(e', \tilde{e}')]}), f') = (\overline{\text{Ref}}_A^e(\tilde{E}[e], \tilde{o}), \overline{\text{Ref}}_A^e(\tilde{E}'[e], \tilde{o}), f') \in \mathcal{R}_{\text{Ref}} \) where \( \tilde{I}' \cap \text{init}_{\text{Obs}}(\tilde{E}[e]) \) is its corresponding set and \( \tilde{f}' \), \( \tilde{f}'' \) are its corresponding functions (please note that \( e \notin E_{\tilde{E}[e]} \)).

Furthermore, from \( \tilde{f} \upharpoonright (\tilde{I}' \cap \text{init}_{\text{Obs}}(\tilde{E}[e])) = \tilde{f}' \upharpoonright \tilde{I}' \) we obtain that \( f' \) and \( f \) coincide on \( \text{init}_{\text{Obs}}(\overline{\text{Ref}}_A^e(\tilde{E}, \tilde{o})_{[(e, \tilde{e})]}) \cap I \).
Suppose \((e'', e''') \in \text{init}_{\text{Obs}}(\tilde{R}^e(A, \tilde{\vartheta}, \tilde{\vartheta}'')_{((e'), (e'''))}) \cap f(I)\). Then from (4.1) and from the fact that 
\[
\tilde{f}^{-1} \upharpoonright (f(\tilde{I}')) \cap \text{init}_{\text{Obs}}(\tilde{E}'_{(e')}) = \tilde{f}^{-1} \upharpoonright f(\tilde{I}')
\]
we obtain the following equation 
\[
f^{-1}(e'', e''') = (\tilde{f}^{-1}(e'''), F'(\tilde{f}^{-1}(e'''))^{-1}(e'''')) = (\tilde{f}^{-1}(e'''), F(\tilde{f}^{-1}(e'''))^{-1}(e''')) = \tilde{f}^{-1}(e'', e'''),
\]
which completes this case.

\(\tilde{I}(e) \in A \land l_{\tilde{\theta}(e)}(\tilde{e}) \neq \sqrt{\cdot} \): By Lemma 4.25 we get \(\tilde{R}^e(A, \tilde{\vartheta}, \tilde{\vartheta}'')_{((e), (e'))} = \tilde{R}^e(A, \tilde{\vartheta}(e \rightarrow \tilde{\vartheta}(e)_{[e]}))\).

Define \(I^e = \{e'' \mid (e, e'') \in I\}\). Since \(I^e \in P_{\text{fin}}(\text{init}_{\text{Obs}}(\tilde{\vartheta}(e)))\) and \((\tilde{\vartheta}(e), \tilde{\vartheta}'(f(e)))\), \(F(e) \in R_{\tilde{I}(e)}\) there exists \(e' \in \text{init}(\tilde{\vartheta}'(f(e)))\) and \(\tilde{f}\) such that \((\tilde{\vartheta}(e)_{[e]}, \tilde{\vartheta}'(f(e))_{[e]}, \tilde{f}) \in R_{\tilde{I}(e)}\) and 
\[
l_{\tilde{\vartheta}(e)}(\tilde{e}) = l_{\tilde{\vartheta}'(f(e))}(\tilde{e}') \land l_{\tilde{\vartheta}(e)}(\tilde{e}) \in \text{Obs} \Rightarrow F(e)(\tilde{e}) = e' \land F(e) \upharpoonright (I^e \cap \text{init}_{\text{Obs}}(\tilde{\vartheta}(e)_{[e]})) = \tilde{f} \upharpoonright I^e \land F(e)^{-1} \upharpoonright (F(e)(I^e) \cap \text{init}_{\text{Obs}}(\tilde{\vartheta}'(f(e))_{[e']})) = \tilde{f}^{-1} \upharpoonright (F(e)(I^e)).
\]

From Lemma 4.25 we obtain \(\tilde{R}^e(A, \tilde{\vartheta}'(f(e)) \rightarrow \tilde{\vartheta}'(f(e))_{[e']})) = \tilde{R}^e(A, \tilde{\vartheta}'(f(e))_{[e']}\rangle\). Furthermore, the labels of \((\tilde{f}(e), e')\) and \((e, e')\) coincide and \(l_{\tilde{R}^e(A, \tilde{\vartheta})_{((e), (e'))}} \in \text{Obs} \Rightarrow f(e, \tilde{e}) = (\tilde{f}(e), F(e)(\tilde{e}')) = (\tilde{f}(e), e').\)

Now define \(f'\) by 
\[
f'(e'', e''') = \begin{cases} 
(\tilde{f}(e), \tilde{f}(e')) & \text{if } (e'', e''') \in \text{init}_{\text{Obs}}(\tilde{R}^e(A, \tilde{\vartheta}, \tilde{\vartheta}'')_{((e), (e'))}) \land e'' = e \\
(\tilde{f}(e), F(e')(e'')) & \text{if } (e'', e''') \in \text{init}_{\text{Obs}}(\tilde{R}^e(A, \tilde{\vartheta}, \tilde{\vartheta}'')_{((e), (e'))}) \land e''' \neq e \\
\text{undefined} & \text{otherwise}
\end{cases}
\]

Then \((\tilde{R}^e(A, \tilde{\vartheta})_{((e), (e'))})\), \((\tilde{R}^e(\tilde{E}', \tilde{\vartheta}' \tilde{\vartheta}'(f(e)) \rightarrow \tilde{\vartheta}'(f(e))_{[e']})\), \(f') \in R_{\tilde{f}_R}\) where \(\tilde{I} \cup \{e\}\) is its corresponding set and \(\tilde{f}, \tilde{f}' = F[e \rightarrow \tilde{f}]\) are its corresponding functions.

Furthermore, from \(F(e) \upharpoonright (I^e \cap \text{init}_{\text{Obs}}(\tilde{\vartheta}(e)_{[e]})) = \tilde{f} \upharpoonright I^e\) we obtain that \(f'\) and \(f\) coincide on \(\text{init}_{\text{Obs}}(\tilde{R}^e(A, \tilde{\vartheta})_{((e), (e'))}) \cap I\).

Suppose \((e'', e''') \in \text{init}_{\text{Obs}}(\tilde{R}^e(A, \tilde{\vartheta}'(f(e))_{[e']}))) \cap f(I)\). Then from (4.1) and from the fact that \(F(e)^{-1} \upharpoonright (F(e)(I^e) \cap \text{init}_{\text{Obs}}(\tilde{\vartheta}'(f(e)))_{[e']})) = \tilde{f}^{-1} \upharpoonright f(e)(I^e)\) we obtain 
\[
f^{-1}(e'', e''') = (\tilde{f}^{-1}(e'''), F(\tilde{f}^{-1}(e'''))^{-1}(e'''')) = (\tilde{f}^{-1}(e'''), \tilde{f}(\tilde{f}^{-1}(e'''))^{-1}(e'''')) = \tilde{f}^{-1}(e'', e''')
\]

The last condition of the FUI-bisimulation can be derived by symmetrical arguments. Thus, we have proved that \(R_{\tilde{f}_R}\) is a FUI-bisimulation.

Obviously, \(\tilde{R}^e(A, \tilde{\vartheta}) = R^e(A, \tilde{\vartheta})\) whenever \(\forall e \in E : l(e) \in A \Rightarrow \vartheta(e) = \theta(l(e))\). Define 
\[
F(e) = \begin{cases} 
\text{cons}_{g(e)} & \text{if } l(e) \in A \land e'' \in \text{init}(\tilde{E}) \\
g_{l(e)} & \text{if } l(e) \notin A \land e'' \in \text{init}(\tilde{E}) \\
\bot & \text{otherwise}
\end{cases}
\]

\[
f(e, \tilde{e}) = \begin{cases} 
(g(e), F(e)(\tilde{e})) & \text{if } (e, \tilde{e}) \in \text{init}_{\text{Obs}}(\tilde{R}^e(A, \tilde{\vartheta})) \\
\text{undefined} & \text{otherwise}
\end{cases}
\]

Then it is easy to check that \((\tilde{R}^e(A, \tilde{\vartheta}), \tilde{R}^e(\tilde{E}', \tilde{\vartheta}), f) \in R_{\tilde{f}_R}\) where \(\emptyset\) is its corresponding set and \(g, F\) are its corresponding functions.

\(\square\)

### 4.4.2 Proof of Proposition 4.20

Before we present the proof of Proposition 4.20, we establish the following lemma.
Lemma 4.28 For all $\mathcal{E}', \mathcal{E}_0, ..., \mathcal{E}_n, e_0, ..., e_{n-1}, f$ and for all $I \in \mathcal{P}_{\text{fin}}(U)$ and for all FUI-bisimulation $\mathcal{R}$ such that $\forall i \leq n - 1 : \mathcal{E}_{i+1} = \mathcal{E}_{i[e_i]}$ and $(\mathcal{E}_0, \mathcal{E}', f) \in \mathcal{R}$ there is a function $g : (I \cup \bigcup_{i \leq n} \text{init}_{\text{Obs}}(\mathcal{E}_i)) \cup \bigcup_{i \leq n-1} \{e_i\} \rightarrow \mathcal{E}'$ such that $g$ is injective, labeling-preserving and $\forall i \leq n : \exists g_i : (\mathcal{E}_i, \mathcal{E}_i'_{[g(e_0)]...[g(e_{i-1})]}, g_i) \in \mathcal{R} \land g \uparrow (\text{init}_{\text{Obs}}(\mathcal{E}_i) \cap I) = g_i \uparrow (\text{init}_{\text{Obs}}(\mathcal{E}_i) \cap I)$.

Proof: We use induction on $n$.

$n = 0$: The result follows immediately if we choose $g = f \uparrow I$.

$n + 1$: Define $\tilde{I} = (I \cap \bigcup_{i \leq n} \text{init}_{\text{Obs}}(\mathcal{E}_i)) \cup \bigcup_{i \leq n} \{e_i\}$. Then by induction there exists $\tilde{g} : ((\tilde{I} \cap \bigcup_{i \leq n} \text{init}_{\text{Obs}}(\mathcal{E}_i)) \cup \bigcup_{i \leq n-1} \{e_i\}) \rightarrow \mathcal{E}'$ such that $\tilde{g}$ is injective, labeling-preserving and $\forall i \leq n : \exists g_i : \tilde{g} \uparrow (\text{init}_{\text{Obs}}(\mathcal{E}_i) \cap \tilde{I}) = g_i \uparrow (\text{init}_{\text{Obs}}(\mathcal{E}_i) \cap \tilde{I}) \land (\mathcal{E}_i, \mathcal{E}_i', g_i) \in \mathcal{R}$, where $\mathcal{E}_i' = \mathcal{E}_i'_{[g(e_0)]...[g(e_{i-1})]}$.

From $(\mathcal{E}_n, \mathcal{E}_n', g_n) \in \mathcal{R}$ and $\tilde{I}' = \mathcal{P}_{\text{fin}}(\text{init}_{\text{Obs}}(\mathcal{E}_n))$, where $\tilde{I}' = \tilde{I} \cap \text{init}_{\text{Obs}}(\mathcal{E}_n)$, we obtain the existence of $e'$ and $g_{n+1}$ such that $l_0(e_n) = l'(e')$ and $(\mathcal{E}_{n+1}, \mathcal{E}_{n+1}', g_{n+1}) \in \mathcal{R}$ and $l_0(e_n) \in \text{Obs} \Rightarrow e' = g_n(e_n)$ and $g_n \uparrow (\tilde{I}' \cap \text{init}_{\text{Obs}}(\mathcal{E}_{n+1})) = g_{n+1} \uparrow \tilde{I}'$ and $g_{n+1} \uparrow (g_n(\tilde{I}') \cap \text{init}_{\text{Obs}}(\mathcal{E}_{n+1}')) = g_{n+1} \uparrow g_n(\tilde{I}')$. Define

$$g(e) = \begin{cases} \tilde{g}(e) & \text{if } e \in \tilde{I} \\ g_{n+1}(e) & \text{if } e \in (I \cap (\text{init}_{\text{Obs}}(\mathcal{E}_{n+1})) \setminus \tilde{I} \}.$$

It is easily seen that $g$ is labeling-preserving and that it coincides with $g_i$ on $\text{init}_{\text{Obs}}(\mathcal{E}_i) \cap I$ for any $i \leq n + 1$.

The injectivity of $g$ can be seen as follows: Suppose there are $e$ and $\tilde{e}$ such that $g(e) = g(\tilde{e})$ and $e \neq \tilde{e}$. This is only possible if $\tilde{e} \in \tilde{I}$ and $e \in (I \cap (\text{init}_{\text{Obs}}(\mathcal{E}_{n+1})) \setminus \tilde{I}$, since otherwise a contradiction to the injectivity of $\tilde{g}$ or $g_{n+1}$ follows. From $e \in \text{init}_{\text{Obs}}(\mathcal{E}_{n+1})$ we obtain $e \in \text{Obs}$, hence $\tilde{e} \in \text{Obs}$ by the labeling preserving of $\tilde{g}$ and $g_{n+1}$. Furthermore, there exists $j \leq n$ such that $\tilde{e} \in \text{init}(\mathcal{E}_j)$, thus $\tilde{g}(\tilde{e}) \in \text{init}_{\text{Obs}}(\mathcal{E}'_j)$. Moreover, $\tilde{g}(\tilde{e}) = g_{n+1}(e) \in \text{init}_{\text{Obs}}(\mathcal{E}_{n+1}')$. Hence, $\tilde{g}(\tilde{e}) \in \text{init}_{\text{Obs}}(\mathcal{E}_n')$ by the definition of the remainder. This implies that $\tilde{e} \in \text{init}_{\text{Obs}}(\mathcal{E}_n)$, hence $\tilde{g}(\tilde{e}) \in g_n(\tilde{I}') \cap \text{init}_{\text{Obs}}(\mathcal{E}_{n+1}')$. From the fact that $\tilde{g}(\tilde{e}) = g_{n+1}(e)$ and that $g_{n+1}$ and $g_{n+1}$ coincide on $g_n(\tilde{I}') \cap \text{init}_{\text{Obs}}(\mathcal{E}_{n+1}')$ we obtain $e = \tilde{e}$, which is a contradiction.

Proof of Proposition 4.20: The inclusion $\sim_{UI} \subseteq \sim_{\text{FUI}}$ is easily seen.

Suppose $\mathcal{E} \sim_{\text{FUI}} \mathcal{E}'$ and $(e_i, \gamma_i)_{i \leq n} \in T^{ic}(\mathcal{E})$. Define $\mathcal{E}_0 = \mathcal{E}$ and $\mathcal{E}_{i+1} = \mathcal{E}_{i[e_i]}$ for $i \leq n$, which is well defined since $(e_i, \gamma_i)_{i \leq n} \in T^{ic}(\mathcal{E})$. And let $\mathcal{R}$ be a FUI-bisimulation such that $(\mathcal{E}, \mathcal{E}', f) \in \mathcal{R}$. Furthermore, define $I = \bigcup_{i \leq n} (\gamma_i \cup \{e_i\})$. Then by Lemma 4.28 we obtain a function $g : I \rightarrow \mathcal{E}'$ such that $g$ is injective, labeling-preserving and $\forall i \leq n : \exists g_i : (\mathcal{E}_i', \mathcal{E}_i', g_i) \in \mathcal{R} \land g \uparrow (\text{init}_{\text{Obs}}(\mathcal{E}_i) \cap I) = g_i \uparrow (\text{init}_{\text{Obs}}(\mathcal{E}_i) \cap I)$, where $\mathcal{E}_i' = \mathcal{E}_i'_{[g(e_0)]...[g(e_{i-1})]}$.

From $(\mathcal{E}_i, \mathcal{E}_i', g_i) \in \mathcal{R}$ and from $\gamma_i \subseteq \text{init}_{\text{Obs}}(\mathcal{E}_i)$ we obtain that $g_i(\gamma_i) \subseteq \text{init}_{\text{Obs}}(\mathcal{E}_i')$. Hence, $(g(e_i), g(\gamma_i))_{i \leq n} \in T^{ic}(\mathcal{E}')$, since $g(\gamma_i) = g_i(\gamma_i)$.
4.4. PROOFS

4.4.3 Proof of Proposition 4.22

We use the following lemmas:

Lemma 4.29 Suppose \( A \subseteq \text{Obs} \) and \( \theta_1, \theta_2 : A \rightarrow \text{CBES} \) such that \( \forall a \in A : \theta_1(a) \sim_b \theta_2(a) \). Then \( \text{Ref}_A^e(\mathcal{E}, \theta_1) \sim_b \text{Ref}_A^e(\mathcal{E}, \theta_2) \) for any \( \mathcal{E} \in \text{CBES} \).

Proof: First, we show

\[
(\forall e \in \mathcal{U} : \vartheta_1(e) \sim_b \vartheta_2(e)) \Rightarrow \text{Ref}_A^e(\mathcal{E}, \vartheta_1) \sim_b \text{Ref}_A^e(\mathcal{E}, \vartheta_2) \tag{4.4}
\]

Let \( \mathcal{R}_e \) be a corresponding bisimulation for \( \vartheta_1(e) \sim_b \vartheta_2(e) \). Define

\[
\mathcal{R} = \{ (\text{Ref}_A^e(\mathcal{E}', \vartheta'_1), \text{Ref}_A^e(\mathcal{E}', \vartheta'_2)) \mid \mathcal{E}' \in \text{CBES} \land \forall e \in \mathcal{U} : (\vartheta_1(e), \vartheta_2(e)) \in \mathcal{R}_e \}
\]

By using Lemma 4.25, it is easy to check that \( \mathcal{R} \) is a bisimulation, which establishes (4.4).

Moreover, \( \vartheta_i \) can be easily derived from \( \theta_i \) such that \( \vartheta_i \) satisfies the requirement of (4.4) and that \( \text{Ref}_A^e(\mathcal{E}, \vartheta_i) = \text{Ref}_A^e(\mathcal{E}, \vartheta_i) \).

 lemma 4.30 Suppose \( A, A' \subseteq \text{Obs} \), \( \theta : A \rightarrow \text{CBES} \), \( \theta' : A' \rightarrow \text{CBES} \) and \( \theta'' : (A \cup A') \rightarrow \text{CBES} \) such that \( \theta''(a) = \begin{cases} \text{Ref}_A^e(\theta(a), \theta') & \text{if } a \in A \\ \theta'(a) & \text{if } a \in A' \setminus A \end{cases} \). Then \( \text{Ref}_A^e(\text{Ref}_A^e(\mathcal{E}, \theta), \theta') \) is isomorphic to \( \text{Ref}_A^e(\mathcal{E}, \theta'') \) for any \( \mathcal{E} \in \text{CBES} \).

Proof: The isomorphism \( \kappa : \mathcal{E}_{\text{Ref}_A^e(\mathcal{E}, \theta) \theta'} \rightarrow \mathcal{E}_{\text{Ref}_A^e(\mathcal{E}, \theta'')} \) is given by

\[
\kappa((e, \hat{e}), \hat{e}') = \begin{cases} (e, e) & \text{if } l(e) \notin A \cup A' \\ (e, \hat{e}') & \text{if } l(e) \in A' \setminus A \\ (e, (\hat{e}, \hat{e})) & \text{if } l(e) \in A \land \hat{e} \notin A' \\ (e, (\hat{e}, \hat{e}')) & \text{if } l(e) \in A \land \hat{e} \in A' \end{cases}
\]

The proof that \( \kappa \) is an isomorphism is straightforward.

Proof of Proposition 4.22: We have \( \sim_c \subseteq \sim_b \), since \( \mathcal{E} \) is isomorphic to \( \text{Ref}_A^e(\mathcal{E}, \theta) \). Therefore, it is only left to prove that \( \sim_c \) is a congruence, since every congruence which is below \( \sim_b \) has to satisfy the constraint of \( \sim_c \).

Suppose \( \mathcal{E}_1 \sim_c \mathcal{E}_2, A \subseteq \text{Obs} \) and \( \forall a \in A : \theta_1(a) \sim_c \theta_2(a) \). We have to verify \( \text{Ref}_A^e(\mathcal{E}_1, \theta_1) \sim_c \text{Ref}_A^e(\mathcal{E}_2, \theta_2) \). Therefore, let \( A' \subseteq \text{Obs} \) and \( \theta : A' \rightarrow \text{CBES} \).

Define \( \theta_i'(a) = \begin{cases} \text{Ref}_A^e(\theta_i(a), \theta) & \text{if } a \in A \\ \theta(a) & \text{if } a \in A' \setminus A \end{cases} \) for \( i \in \{1, 2\} \). From \( \forall a \in A : \theta_1(a) \sim_c \theta_2(a) \) we get

\[
\forall a \in A \cup A' : \theta_1'(a) \sim_b \theta_2'(a) \tag{4.5}
\]

Therefore, we obtain that \( \text{Ref}_{A' \cup A}(\mathcal{E}_1, \theta_1') \cong \text{Ref}_{A' \cup A}(\mathcal{E}_2, \theta_2') \) \( \text{Lemma 4.30} \) and \( \text{Lemma 4.29} \) \( \text{Ref}_{A' \cup A}(\mathcal{E}_1, \theta_1') \sim_b \text{Ref}_{A' \cup A}(\mathcal{E}_2, \theta_2') \) \( \text{Lemma 4.30} \) and \( \text{Lemma 4.29} \), as required.
Chapter 5

Terminating by Action Execution

In this chapter, an alternative approach to process termination, where a process is considered to terminate by executing its ‘final’ action [14, 28], is considered. This approach is called fa-approach in the following. In addition to the transition relation a predicate for termination with respect to action names is used. This termination approach is also used in many event-based models, like those mentioned in [91], where termination is indicated by maximal configurations and not by termination events.

Two new kinds of event structures are introduced in order to give a true concurrency model for process algebras based on the fa-philosophy. One type of event structure models the ‘non-disabling’ of events instead of disabling by using a witness relation. The other type of event structure models disabling by indicating sets of precursor events. We show that both types of event structures have the same expressive power and are more expressive with respect to event traces than the standard event structures. A consistency result of an operational and a denotational semantics is shown.

5.1 Motivation

Disrupt mechanisms are important in order to model many realistic systems. Hence they have found their way into various process algebras [14, 15, 73, 75]. The disrupt operator of LOTOS [32], called disabling operator, is denoted by $B_1 > B_2$. Here, any action executed by $B_2$ disables $B_1$, and the termination of $B_1$ disables $B_2$ (see also Chapter 3). Disrupt mechanisms are e.g. used to model timeouts, which represent an important concept for many applications.

In the definition of an operational semantics, the disrupt operator has to be described. Therefore it is necessary to specify when a process terminates. This can be achieved in two ways:

- By providing an additional syntactical expression 1 to indicate the process that may terminate immediately. For the operational description, an additional action $\sqrt{}$, which indicates termination, and a rule $1 \xrightarrow{\sqrt{}} \ldots$ are used. This approach is taken in Chapter 3 and in [47, 32]¹, for example.

¹These papers only differ with respect to the handling of sequential composition. For example, the sequential operator removes action $\sqrt{}$ in [47], whereas it is replaced with the internal action in LOTOS [32].
Unfortunately, some interesting and important ways of using disrupt mechanisms are excluded by the approach described above, as it is argued by the following generic example.

**Example 5.1** Consider a nuclear power plant. Let $P_{sd}$ be a process that controls the shutdown of the reactor and let $P_{nr}$ be the process that describes the normal running behavior of the reactor. In process $P_{nr}$ an action named stop may be activated by the environment. If stop is executed, the normal behavior $P_{nr}$ is terminated and $P_{nr}$ may only terminate in this way. The stop-action indicates the controlled shutdown demand of a worker and is in general not allowed at every execution step of $P_{nr}$, i.e. it can not be modeled by disruption. Then, a simple specification of a nuclear power plant is given by

$$P_{nr}; P_{sd}$$

where ; denotes the sequential composition. A more realistic specification of a nuclear power plant may invoke the shutdown process $P_{sd}$ for various other reason, e.g. if the temperature reaches a critical point. In addition, the shutdown may be combined with other activities, e.g. that of setting off an alarm.

Let us consider a nuclear power plant with the action named stop and a shutdown which is triggered by temperature and that also invokes an alarm. In any system run either a normal termination of $P_{nr}$ by stop or a disruption of $P_{nr}$ by a critical temperature message may happen, but it should be prevented that both occur. In particular, once the stop-action has been taken, no alarm should be set off. Let action $t$ denote that the temperature of the reactor reaches the critical point and let action $a$ denote that an alarm is set off. The natural representation of the reactor control in LOTOS is given by

$$((P_{nr} [> t]; P_{sd}) || \{t\} (t; a)$$

where $B_1 ||_A B_2$ denotes the parallel execution of $B_1$ and $B_2$ with synchronization on the actions in $A$. However, according to the semantics of LOTOS, $t$ can happen after stop. This originates in the fact that the $\sqrt{}$-action (and not the stop-action) terminates $P_{nr}$. Hence, an alarm with expensive consequences may be unnecessarily set off.

- An alternative approach to the problem of dealing with termination is to specify that a process terminates when it executes its ‘final’ action [14, 28]. For example, the process $a ||_0 b$ terminates by executing $a$ if $b$ was executed before or it terminates by executing $b$ if $a$ was executed before. This approach, which is called fa-approach in the following, leads to the expected behavior of the process considered in the above example. The fa-approach is modeled by using a predicate for termination with respect to action names in the transition system. There is no need to extend the syntactical expressions by further expressions, like 1, in order to handle termination.

### 5.2 Syntax

Let $\tau$, Obs and Var be defined as in Section 3.2. The set of all actions $\mathcal{A}ct$ is defined by $\mathcal{A}ct = \{\tau\} \cup \text{Obs}$. A relabeling function $f$ is a function from $\mathcal{A}ct$ to $\mathcal{A}ct$ such that $f(\tau) = \tau$. We denote the set of all labeling functions by $\mathcal{F}_L$. 
5.3. OPERATIONAL SEMANTICS FOR PA_{ST}

The process algebra expressions EXP_{st} (s ∆= start-based, t ∆= fa-termination) are defined by the following BNF-grammar.

\[
B \ ::= \ 0 \mid a \mid B + B \mid B; B \mid B > B \mid B|A B \mid B[t] \mid B \setminus A \mid x
\]

where \( f \in F_L, x \in \text{Var}, a \in \text{Act} \) and \( A \subseteq \text{Obs} \). A process with respect to EXP_{st} is a pair \( \langle \text{decl}, B \rangle \) consisting of a declaration \( \text{decl} : \text{Var} \rightarrow \text{EXP}_{st} \) and an expression \( B \in \text{EXP}_{st} \). Let PA_{st} denote the set of all processes. We sometimes call an expression \( B \in \text{EXP}_{st} \) also a process if \( \text{decl} \) is clear from the context.

The expressions have the following intuitive meaning: \( a \) is the process that executes \( a \) and terminates. \( B_1 > B_2 \) is the disruption of \( B_1 \) by \( B_2 \), i.e. any action from \( B_2 \) disables \( B_1 \) as long as \( B_1 \) has not terminated. On the other hand, the termination of \( B_1 \) disables \( B_2 \), \( B_1|A B_2 \) describes the parallel execution of \( B_1 \) and \( B_2 \), where both processes have to synchronize on actions of \( A \). The process terminates if \( B_1 \) and \( B_2 \) terminate in the case of synchronization or if one terminates and the other one has already terminated. The relabeling process \( B[t] \) executes action \( f(a) \) if \( B \) executes action \( a \). The restriction process \( B \setminus A \) executes action \( a \) if \( B \) executes action \( a \), provided \( a \) is not contained in \( A \). The behavior of the inactive process \( 0 \), of the choice operator +, of the sequential composition ; and of variable \( x \) is described in Section 3.2.

Process algebras, like [32, 98] or the one used in Chapter 3, that are based on the synchronization we have just presented and which contain an expression for a termination process (denoted by 1) can model a restriction operator in terms of the parallel operator \((B|A, 1)\). Since we do not introduce an expression for a termination process, we include the restriction operator, as in [25, 138]. As it turns out, the restriction operator plays also a crucial role for the operational definition of the parallel operator in our setting.

5.3 Operational Semantics for PA_{st}

As stated in the beginning of this chapter, we adopt the philosophy that the ‘final’ executed action terminates the process. Therefore, we have to distinguish between ‘final’ actions and ‘non-final’ actions. In transition systems, the fa-philosophy is usually modeled by using a predicate for every action \( a \) to determine that the process can terminate by executing \( a \) [14, 28]. The non-terminating action execution is modeled by the usual transition relation. We take a different approach using additional labels. More precisely, we allow transitions to be labeled with elements of \( \text{Act}^T = \text{Act} \cup (\text{Act} \times \{\sqrt{\_}\}) \), where label \((a, \sqrt{\_})\) indicates that the process terminates by executing action \( a \), i.e. \( a \) is a ‘final’ action of that process. Actions of \( \text{Act} \times \{\sqrt{\_}\} \) are called termination actions. Our approach leads to a decrease in the number of transition rules and allows a simplification of the definition of bisimulation and of the related proofs. The transition rules of \( \rightarrow_{\text{decl}} \subseteq \text{EXP}_{st} \times \text{Act}^T \times \text{EXP}_{st} \) with respect to \( \text{decl} : \text{Var} \rightarrow \text{EXP}_{st} \) are presented in Figure 5.1. We write \( a\sqrt{\_} \) instead of \((a, \sqrt{\_})\).

In the following, we explain the rules which deviate from the standard ones: Process \( a \) can execute \( a \) and terminates by executing this action. The process which results in a deadlock after executing \( a \) can be modeled by \( a; 0 \), for example. The transition rule for the choice operator is the standard CCS-rule [138]. Please note that no distinction between actions and termination actions is made. If the first process of the sequential composition terminates by executing \( a \)
In the following let $\gamma$ be an element of $\mathcal{A}ct^T$ and $a$ be an element of $\mathcal{A}ct$.

$A_1 : \frac{a \xrightarrow{\alpha^\gamma} 0}{a} \quad C : \frac{B_1 \xrightarrow{\gamma} B'}{B_1 + B_2 \xrightarrow{\gamma} B'} \quad \frac{B_2 + B_1 \xrightarrow{\gamma} B'}{B_2 + B_1}$

$S_1 : \frac{B_1 \xrightarrow{a} B_1'}{B_1; B_2 \xrightarrow{a} B_1'; B_2} \quad S_2 : \frac{B_1 \xrightarrow{\alpha^\gamma} B_1'}{B_1; B_2 \xrightarrow{a} B_2}$

$I_1 : \frac{B_1 \xrightarrow{a} B_1'}{B_1 [> B_2 \xrightarrow{a} B_1'][> B_2]} \quad I_2 : \frac{B_1 \xrightarrow{\alpha^\gamma} B_1'}{B_1 [> B_2 \xrightarrow{\alpha^\gamma} B_1'][> B_2]} \quad I_3 : \frac{B_2 \xrightarrow{\gamma} B_2'}{B_1 [> B_2 \xrightarrow{\gamma} B_2'][> B_2]}$

$P_1 : \frac{B_1 \xrightarrow{a} B_1'}{B_1 || A B_2 \xrightarrow{a} B_1' || A B_2} \quad P_2 : \frac{B_1 \xrightarrow{\alpha^\gamma} B_1'}{B_1 || A B_2 \xrightarrow{a} B_2 [> A]} \quad \frac{B_2 || A B_1 \xrightarrow{a} B_2 [> A]}{B_2 || A B_1 \xrightarrow{a} B_2 [> A]}$

$P_3 : \frac{B_1 \xrightarrow{a} B_1'}{B_1 || A B_2 \xrightarrow{a} B_1' || A B_2'} \quad a \in A \quad P_4 : \frac{B_1 \xrightarrow{\alpha^\gamma} B_1'}{B_1 || A B_2 \xrightarrow{a} B_2 [> A]} \quad \frac{B_2 || A B_1 \xrightarrow{a} B_2 [> A]}{B_2 || A B_1 \xrightarrow{a} B_2 [> A]}$

$P_5 : \frac{B_1 \xrightarrow{\alpha^\gamma} B_1'}{B_1 || A B_2 \xrightarrow{a} B_1' || A B_2'} \quad a \in A \quad \frac{B_1 || A B_2 \xrightarrow{a} 0}{B_1 || A B_2}$

$Lab_1 : \frac{B \xrightarrow{a} B'}{B[f] \xrightarrow{f(a)} B'[f]} \quad Lab_2 : \frac{B \xrightarrow{\alpha^\gamma} B'}{B[f] \xrightarrow{f(a)^\gamma} B'[f]}$

$Res_1 : \frac{B \xrightarrow{a} B'}{B \xrightarrow{a} B'} \quad a \notin A \quad Res_2 : \frac{B \xrightarrow{\alpha^\gamma} B'}{B \xrightarrow{a} B'} \quad a \notin A$

$Rec : \frac{\text{decl}(x) \xrightarrow{\gamma} B'}{x \xrightarrow{\gamma} B'}$

Table 5.1: Transition Rules for $\rightarrow^t_{\text{decl}}$

(rule $S_2$), then $B_1; B_2$ executes $a$ without termination. The rules for the disrupt operator are as they were expected, in particular if $B_1$ terminates by executing $a$, so does $B_1 [> B_2$.

In the case of the parallel operator, we have to distinguish whether or not the actions that are executed by the subprocess are termination actions. The second rule states that if a subprocess executes a non-synchronizing termination action, then this process has to be removed and
5.4. DENOTATIONAL SEMANTICS FOR $\text{PA}_{\text{st}}$

all actions in the synchronization set have to be forbidden for the remaining process. If both
processes execute a termination action of the synchronization set, the resulting process is 0,
since after the termination of both processes the parallel process has terminated, and no further
actions will be executed.

The rules for the relabeling operator and the restriction operator only depend on the action name
and preserve termination, as expected.

**Remark 5.2** It is easily seen by induction on the depth of inference that after the execution of
a termination action no further actions may be executed, i.e.

$$\forall B, B', a : B \xrightarrow{a^{\sqrt{\text{decl}}}} B' \Rightarrow (\forall \gamma, B'' : \neg(B' \xrightarrow{\gamma^{\text{decl}}} B'')).$$

5.4 Denotational Semantics for $\text{PA}_{\text{st}}$

A denotational semantics of $\text{PA}_{\text{st}}$ that corresponds to the operational semantics can not be given
in the standard event structures. This is argued as follows.

**Prime event structures** [145], **flow event structures** [36] and **stable event structures** [178] require
a symmetric conflict relation which makes it hard to model disruption. In addition **configura-
tions** [87, 177] do not provide a smooth way to model disruption. Consider, for example, pro-
cess $\tilde{B}$, which consists of the disruption of $a; b$ by action $c$ (i.e. $\tilde{B} = (a; b)[c]$). An intuitive
approach is to assume that $\tilde{B}$ has three events denoted by $a, b, c$. The sets $\emptyset, \{a\}, \{c\}, \{a, b\}$
can be considered to be configurations, but what about $\{a, c\}$? Assuming it is not a configura-
tion contradicts the existing execution $a \Rightarrow c$. On the other hand, assuming that $\{a, c\}$ is also
a configuration leads to the interpretation that the execution $c \Rightarrow a$ is legal\(^2\), which contra-
dicts the branching structure of $\tilde{B}$, since after the disruption (by $c$) no further actions from the
left-hand process may be executed.

**Closed bundle event structures** (Subsection 3.3.2), which can be used to give a denotational
semantics to LOTOS, and **dual event structures** [116] (Remark 3.3) allow the modeling of dis-
ruption, since the symmetry condition for the conflict relation is dropped. If (and how) these
types of event structures could be used to define a denotational semantics that also incorporates
the fa-philosophy is highly questionable. Consider, for example, the process $(a || \emptyset b)[c]:$ If we
put $c$ in conflict with $a$, then $c$ can be disabled before $b$ happens and by symmetry the same
argument holds for $b$. But $c$ has to be in conflict with some action, since otherwise $c$ would
remain enabled after the execution of $a$ and $b$.

Therefore, a conflict relation that is based on a binary relation on events is not appropriate to
model this kind of disrupt operator in the context of the fa-philosophy.

In this section, we present two new classes of event structures that are suitable to model the
fa-philosophy and study their properties. The first one, which is called **extended termination
bundle event structure**, moves from a conflict approach to a **witness** approach by introducing a
relation $(\succ)$ between sets of events and events, i.e. $\succ \subseteq \mathcal{P}(E) \times E$. A witness condition $(Z \succ e)$
indicates that event $e$ is not disabled in a system run if no event from the system run is contained

\(^2\)by the common definition [177, 87]
in $Z$. In other words, a system run disables an event $e$ if each witness-bundle $Z$ of $e$ $(Z \succ e)$ contains an element of the system run. As long as there is a witness condition $Z \prec e$, where $Z$ does not contain an element of the system run so far, $e$ is not disabled. $Z$ is considered to be a witness of this fact.

The extended termination bundle event structures follow the philosophy of bundle event structures [126] that each bundle either from causality or from the witness relation is interpreted existentially, i.e. the execution of any event of a bundle is sufficient to fulfill the requirement denoted by the bundle.

The other new class of event structures, which is called extended termination precursor event structure, follows the contrary approach that each bundle (called precursor in these event structures) is universally quantified, i.e. every event of a precursor-bundle has to be executed to fulfill the requirement. More precisely, the conflict relation $(\succ)$ is given as a relation between sets of events and events, i.e. $\succ \subseteq \mathcal{P}(E) \times E$. An event $e$ is disabled by a system run if and only if there is a precursor $Z$ of $e$ $(Z \succ e)$ such that all events of $Z$ are contained in the system run. The causality relation of an extended termination precursor event structure is also defined with the universal quantification philosophy, as in Winskel’s event structures [178].

The rest of this section is organized as follows: First we neglect disruption and only concentrate on the fa-philosophy. Therefore, termination bundle event structures, which represent a generalization of bundle event structures, are presented in Subsection 5.4.1. They allow the modeling of the fa-philosophy. In Subsection 5.4.2 the class of extended termination bundle event structures, which can also handle disruption, is introduced. This Subsection contains the result that this class of event structures is more expressive than the standard event structures with respect to event traces. Operators on extended termination bundle event structures are defined in Subsection 5.4.3. These operators are used in Subsection 5.4.4 to define the denotational semantics of $\text{PA}_{\text{st}}$. There the consistency between the denotational semantics and the operational semantics is also given. The class of extended termination precursor event structures is introduced in Subsection 5.4.5. In Subsection 5.4.6, it is shown that the class of extended termination bundle event structures and the class of extended termination precursor event structures have the same expressive power with respect to event traces.

### 5.4.1 Termination Bundle Event Structure (TBES)

Bundle event structures (Definition 3.1) indicate the termination of a process by additional events that are labeled with the termination symbol $\sqrt{\cdot}$. These events are maintained by sequential composition, where they are relabeled with the internal action. This is not appropriate for models of process algebras that are based on the fa-philosophy, since contrary to the semantics of LOTOS, no internal action is executed when the first process terminates in the sequential composition.

Therefore, we do not allow events labeled with the termination symbol. Consequently, we have to model termination in a different way. Our approach is to consider the termination event to be fictitious, and therefore we collect the bundles of the termination event without pointing directly to an event. This means that we add an additional component that consists of a collection of bundles, i.e. consists of a collection of subsets of events, to bundle event structures.
Definition 5.3 (Termination Bundle Event Structure) A termination bundle event structure, Tbes for short, \((E, \#, \to, T, l)\) is an element of \(\mathcal{P}(U) \times \mathcal{P}(U \times U) \times \mathcal{P}(\mathcal{P}(U) \times U) \times \mathcal{P}(\mathcal{P}(U)) \times (U \to \text{Act})\) such that

- \(\# \subseteq E \times E\) and \(\#\) is irreflexive and symmetric
- \(\to \subseteq \mathcal{P}(E) \times E\)
- \(T \subseteq \mathcal{P}(E)\) and \(T \neq \emptyset\)
- \(\text{dom}(l) = E\)
- \(\forall e \in E : \_ \to e\) is approximation closed with respect to \(E\)
- \(T\) is approximation closed with respect to \(E\)

Let \(\text{TBES}\) denote the set of all termination bundle event structures.

We call \(E\) the set of events, \(\#\) the conflict relation, \(\to\) the causality relation, \(T\) the termination set and \(l\) the action-labeling function.

The intuitive meaning of a Tbes is the following: If two events \(e, e'\) are in conflict, i.e. \(e \# e'\), then only one of them can appear in a system run. The meaning of \(X \to e\) is that before \(e\) may be executed, an event of \(X\) has to be executed. A system run of a Tbes is terminated if all bundles in the termination set are satisfied, i.e. every element of \(T\) contains an event of the system run. The constraint \(T \neq \emptyset\) on the termination set ensures that an Tbes is not able to terminate immediately, i.e. it can only terminate by executing an action. However, \(T\) might consist of the empty set only. The labeling function indicates which action is observable when the event is executed. The two approximation closedness constraints are used to guarantee that the standard order yields a complete partial order.

\(\text{TBES}\) can be used as a denotational semantics for the processes of \(\text{PA}_{\text{ST}}\) that do not contain disrupt expressions. The operators on \(\text{TBES}\) have to vary from those of the original bundle event structure (Definition 3.17 or [125, 126]). This is necessary, since the original bundle event structure (Definition 3.1) allows any number of events for termination, whereas our approach has exactly one event (the fictive one) for termination. We do not present the operators for \(\text{TBES}\). They can be easily obtained by adapting the operators for extended termination bundle event structures presented in Definition 5.20.

Remark 5.4 The original bundle event structures (Definition 3.1) contain an additional condition, called bundle stability constraint (compare with Remark 3.3). It can also be added to the condition of a Tbes, since it is an invariant for all operators needed in the denotational semantics. We omit this constraint, since it is not important for the theory presented here.

### 5.4.2 Extended Termination Bundle Event Structure (ETBES)

In extended termination bundle event structures the non-disabling of events rather than the disabling is modeled, which is done by a relation between sets of events and events. The non-disabling modeling follows the philosophy of bundle event structures, where each bundle (obtained from causality) is existentially quantified, i.e. the execution of any event of a bundle is
sufficient to fulfill the requirement denoted by the bundle. In Section 5.4.5 we present a different approach, where every bundle is universally quantified.

**Definition 5.5 (Extended Termination Bundle Event Structure)** An extended termination bundle event structure, eTbes for short, $\mathcal{E} = (E, \succ, \mapsto \rightarrow, T, l)$ is an element of $\mathcal{P}(\mathcal{U}) \times \mathcal{P}(\mathcal{P}(\mathcal{U}) \times \mathcal{U}) \times \mathcal{P}(\mathcal{P}(\mathcal{U}) \times \mathcal{U}) \times \mathcal{P}(\mathcal{P}(\mathcal{U})) \times (\mathcal{U} \rightarrow \mathcal{Act})$ such that

- $\succ \subseteq \mathcal{P}(E) \times E$ and $\forall e \in E : \exists Z : Z \succ e$ and $\forall (Z, e) \in \succ : e \in Z$
- $\mapsto \rightarrow \subseteq \mathcal{P}(E) \times E$
- $T \subseteq \mathcal{P}(E)$ and $T \neq \emptyset$
- $\text{dom}(l) = E$
- $\forall e \in E : \_ \succ e$ is approximation closed with respect to $E$
- $\forall e \in E : \_ \mapsto \rightarrow e$ is approximation closed with respect to $E$
- $T$ is approximation closed with respect to $E$

Let ETBES denote the set of all extended termination bundle event structures.

$\succ$ is called the witness relation. The attribute extended in the name of the event structures defined above is used to emphasize (as it is done for extended bundle event structures) that these event structures can model disruption.

The intuitive meaning of a witness-bundle $Z$ of $e$ ($Z \succ e$) is that event $e$ is not disabled in a system run if no event from the system run is contained in $Z$. A system run enables an event $e$ if all witness-bundles of $e$ contain an element of the system run. The constraints imposed on the witness relation are: Firstly, every event $e$ must have a witness-bundle, since otherwise $e$ would never be enabled and hence, could be omitted. Secondly, every witness-bundle of $e$ has to contain $e$, since the execution of an event disables itself, i.e. every event can be executed only once. Furthermore, the witness relation also has to satisfy approximation closedness constraints to guarantee that the standard order yields a complete partial order.

**Example 5.6** Some eTbes are shown in Figure 5.1. Here, the events are depicted as dots and their corresponding action names appears close by the dots (we do not name the events explicitly and we identify them with the action names if no confusion arises). The witness relation is illustrated by wavy lines. More precisely, a witness-bundle $Z \succ e$ is depicted by a wavy arrow from the elements of $Z \setminus \{e\}$ to $e$. In the special case when $Z$ consists only of $e$, we use a wavy arrow from the empty-set to $e$. Sometimes, the same wavy lines are used in different witness-bundles, for example the witness-bundles in $\mathcal{E}_6$ are $\{a, b, c\} \succ a$, $\{a, b, c\} \succ b$ and $\{a, b, c\} \succ c$. The causality relation is depicted similarly to the witness relation, except that straight lines are used instead of wavy lines. A termination set $X$ is displayed by surrounding its events by a closed line.
5.4. DENOTATIONAL SEMANTICS FOR PA\textsubscript{ST}  

Hereafter, we consider \( \mathcal{E} \) to be \((E, \succ, \rightarrow, T, l)\), \( \mathcal{E}_i \) to be \((E_i, \succ_i, \rightarrow_i, T_i, l_i)\) and in general \( \mathcal{E} \) to be \((E_\mathcal{E}, \succ_\mathcal{E}, \rightarrow_\mathcal{E}, T_\mathcal{E}, l_\mathcal{E})\). Furthermore, \( \text{init}(\mathcal{E}) \) denotes the set of events which are ready to execute and \( \Upsilon(T, e) \) holds if and only if \( e \) is a termination event with respect to \( T \), i.e. \( \mathcal{E} \) terminates by executing \( e \). Formally:

**Definition 5.7** Let \( \mathcal{E} \) be an eTbes. The set of initial events of \( \mathcal{E} \) is defined by

\[
\text{init}(\mathcal{E}) = \{ e \in E \mid \neg(\exists X : X \rightarrow e) \}.
\]

The termination predicate \( \Upsilon \subseteq \mathcal{P}(\mathcal{P}(U)) \times U \) is defined by

\[
\Upsilon(T, e) \iff \forall X \in T : e \in X.
\]

**Remark 5.8** Please note, that the events of an eTbes are labeled with elements of Act and not of Act\textsuperscript{T}, i.e. events must not be labeled, for example, by \( a\sqrt{.} \). This is necessary, since we do not know a priori if an event is a termination event of a system run. See \( \mathcal{E}_1 \) in Figure 5.1, for example.

**Transition system from an eTbes.**

Here, we describe how to obtain a transition system from an eTbes, which is later used to analyze the expressive power of ETBES and to establish a consistency result for the denotational and the operational semantics. In order to obtain a transition system from an eTbes, we define the remainder of an eTbes with respect to an initial event. The remainder with respect to event

---

Figure 5.1: Some Extended Termination Bundle Event Structures

Here, we consider \( \mathcal{E} \) to be \((E, \succ, \rightarrow, T, l)\), \( \mathcal{E}_i \) to be \((E_i, \succ_i, \rightarrow_i, T_i, l_i)\) and in general \( \mathcal{E} \) to be \((E_\mathcal{E}, \succ_\mathcal{E}, \rightarrow_\mathcal{E}, T_\mathcal{E}, l_\mathcal{E})\). Furthermore, \( \text{init}(\mathcal{E}) \) denotes the set of events which are ready to execute and \( \Upsilon(T, e) \) holds if and only if \( e \) is a termination event with respect to \( T \), i.e. \( \mathcal{E} \) terminates by executing \( e \). Formally:

**Definition 5.7** Let \( \mathcal{E} \) be an eTbes. The set of initial events of \( \mathcal{E} \) is defined by

\[
\text{init}(\mathcal{E}) = \{ e \in E \mid \neg(\exists X : X \rightarrow e) \}.
\]

The termination predicate \( \Upsilon \subseteq \mathcal{P}(\mathcal{P}(U)) \times U \) is defined by

\[
\Upsilon(T, e) \iff \forall X \in T : e \in X.
\]

**Remark 5.8** Please note, that the events of an eTbes are labeled with elements of Act and not of Act\textsuperscript{T}, i.e. events must not be labeled, for example, by \( a\sqrt{.} \). This is necessary, since we do not know a priori if an event is a termination event of a system run. See \( \mathcal{E}_1 \) in Figure 5.1, for example.

**Transition system from an eTbes.**

Here, we describe how to obtain a transition system from an eTbes, which is later used to analyze the expressive power of ETBES and to establish a consistency result for the denotational and the operational semantics. In order to obtain a transition system from an eTbes, we define the remainder of an eTbes with respect to an initial event. The remainder with respect to event
CHAPTER 5. TERMINATING BY ACTION EXECUTION

e describes the system after the execution of e. Therefore, we remove all events which are disabled by e, i.e. we only keep those events that have a witness-bundle which does not contain e. Please remember that an event has to be an element of all its witness-bundles. Hence, it disables itself. Furthermore, all bundles (from causality, witness or termination) that contain e are removed, since the execution of e fulfills the requirements specified by those bundles, i.e. the bundle contains an element of the system run. In the definition of the termination set, we consider the case that an eTbes terminates by executing e separately in order to guarantee that the remainder of an eTbes is an eTbes. This distinction is necessary, since otherwise the termination set would become the empty set, which is not allowed for an eTbes.

Definition 5.9 (Remainder of an eTbes) Let \( E \in \text{ETBES} \) and \( e \in \text{init}(E) \). Then the remainder \( E[e] \) of \( E \) is given by \( (E', \succ', \mapsto', T', l') \) where

\[
E' = \{ e' \in E \mid \exists Z : Z \succ e' \land e \notin Z \}
\]
\[
\succ' = \{ (Z \cap E', e') \mid e' \in E' \land Z \succ e' \land e \notin Z \}
\]
\[
\mapsto' = \{ (X \cap E', e') \mid e' \in E' \land X \mapsto e' \land e \notin X \}
\]
\[
T' = \{ \{ X \cap E' \mid X \in T \land e \notin X \} \}
\quad \text{if } \exists T, e
\]
\[
\{ \emptyset \}
\quad \text{otherwise}
\]
\[
l' = l \upharpoonright E'
\]

It can be shown that the remainder of an eTbes is also an eTbes.

Lemma 5.10 Let \( E \in \text{ETBES} \) and \( e \in \text{init}(E) \). Then \( E[e] \in \text{ETBES} \).

Proof: The approximation closedness conditions are an immediate consequence of Corollary 2.18. The other conditions can be easily checked. \( \square \)

The remainders are used in the following definition to obtain an interleaving semantics for ETBES.

Definition 5.11 The transition relation \( \leftrightarrow \subseteq \text{ETBES} \times \text{Act}^T \times \text{ETBES} \) is defined by

\[
\leftrightarrow = \{ (E, \gamma, E[e]) \mid E \in \text{ETBES} \land e \in \text{init}(E) \land \gamma = \{ l(e) \sqrt{\checkmark} \text{ if } \exists T, e \}
\}
\]

The transition system obtained from \( E_3 \) of Figure 5.1 is presented in Figure 5.2.

Remark 5.12 According to the definition of the remainder, it is possible that further events may be executed after the execution of a termination event. This effect also arose in the original bundle event structures. It is possible to circumvent this effect by considering only those eTbes where every event set that leads to termination also disables every event. Formally, eTbes \( \mathcal{E} \) has to satisfy (remember that \( \succ e = \{ Z \mid Z \succ e \} \)):

\[
\forall e \in E : \forall E' \subseteq E : (\forall X \in T : E' \cap X \neq \emptyset) \Rightarrow (\forall Z \in \succ e : Z \cap E' \neq \emptyset).
\]
5.4. DENOTATIONAL SEMANTICS FOR PA_{ST}  

Expressive Power.

The expressive power of event structures can be measured by comparing the set of event traces described by them. Here, we present the definition of event traces with respect to ETBES. The event traces for the other event structures are similarly defined.

**Definition 5.13** An event trace of $E_1 \in \text{ETBES}$ is a finite sequence of events $(e_1, ..., e_n)$ such that there are $e_{\text{Tes}} E_2, ..., E_{n+1}$ with $E_j[e_j] = E_{j+1}$ for all $j \leq n$. The set of all event traces of $E_1$ is denoted by $\text{Tr}^e(E_1)$.

A set $M$ of finite sequences of events (set of event traces) is described by ETBES if there exists $E \in \text{ETBES}$ such that $M = \text{Tr}^e(E)$.

For simplicity, we neglect the termination information when we compare the expressive power. The termination information can be easily included by dividing the set of event traces into terminated and non-terminated traces.

**Example 5.14** The set of event traces of $E_3$ from Figure 5.1 is

$$\text{Tr}^e(E_3) = \{(a), (b), (c), (a, b), (b, a), (a, c), (b, c)\}.$$

**Theorem 5.15** Every set of event traces that is described by prime [145], flow [36], stable [178], bundle, extended bundle or dual event structures [116] is also described by extended termination bundle event structures, but not vice versa.

**Proof:** From [116] we know that every set of event traces described by a cited class of event structures is also described by dual event structures. The inclusion of dual event structures in extended termination bundle event structures is shown by mapping the dual event structure $(E, \sim, \rightarrow, l)$ to $(E, \triangleright, \rightarrow, \{\emptyset\}, l)$ where $\triangleright = \{\{e' \in E \mid e \sim e'\} \cup \{e\}, e \mid e \in E\}$. Furthermore, the causality relation has to be extended such that it is approximation closed with respect to $E$. This extension does not change the set of event traces if the least extension is used.

On the other hand, the set of event traces obtained from $E_3$ of Figure 5.1 can not be described by a dual event structure. □
Complete Partial Order.

In order to give a semantics to PA<sub>et</sub>, we turn ETBES into an ω-complete partial order. First, we present the definition and results concerning the restriction of an eTbes, which is used to define an order on ETBES.

**Definition 5.16 (Restriction of an eTbes)** Suppose \( \mathcal{E} \in \text{ETBES} \) and \( E' \subseteq E \). Then the restriction of \( \mathcal{E} \) to \( E' \), denoted by \( \mathcal{E} \upharpoonright E' \), is given by \( (E', \succ', \mapsto', T', l') \) where

\[
\begin{align*}
\succ' & = \{ (Z \cap E', e') \mid e' \in E' \land Z \succ e' \} \\
\mapsto' & = \{ (X \cap E', e') \mid e' \in E' \land X \mapsto e' \} \\
T' & = \{ X \cap E' \mid X \in T \} \\
l' & = l \upharpoonright E'
\end{align*}
\]

**Lemma 5.17** Let \( \mathcal{E} \in \text{ETBES} \) and \( E' \subseteq E \). Then \( \mathcal{E} \upharpoonright E' \in \text{ETBES} \).

**Proof:** Is an immediate consequence of Corollary 2.18.

**Definition 5.18 (Order on ETBES)** Let \( \mathcal{E}_i \in \text{ETBES} \). Then \( \mathcal{E}_1 \preceq \mathcal{E}_2 \) if \( E_1 \subseteq E_2 \) and \( \mathcal{E}_1 = \mathcal{E}_2 \upharpoonright E_1 \).

**Theorem 5.19** The set of all eTbes ordered by \( \preceq \) is an \( \omega \)-complete partial order, where the least upper bound of an \( \omega \)-chain \((\mathcal{E}_i)_{i \in \mathbb{N}}\) is given by \( \bigsqcup_i \mathcal{E}_i = (\bigcup_i E_i, \succ, \mapsto, T, \bigcup_i l_i) \) with

\[
\begin{align*}
\succ & = \{ (Z, e) \mid \forall k : e \in E_k \Rightarrow (Z \cap E_k) \succ_k e \} \\
\mapsto & = \{ (X, e) \mid \forall k : e \in E_k \Rightarrow (X \cap E_k) \mapsto_k e \} \\
T & = \{ X \mid \forall k : X \cap E_k \in T_k \}
\end{align*}
\]

**Proof:** It works analogously to the proof of Theorem 3.10.

**5.4.3 Operators on ETBES**

Here, we present the operators on ETBES that are later used to define the denotational semantics.

**Definition 5.20 (Operators on ETBES)** Let \( A \subseteq \text{Obs} \). Then define

\[
\begin{align*}
\hat{\circ} : \text{ETBES} \times \text{ETBES} & \rightarrow \text{ETBES} \text{ with } \mathcal{E}_1 \hat{\circ} \mathcal{E}_2 = (\hat{E}, \hat{\succ}, \hat{\mapsto}, \hat{T}, \hat{l}) \text{ where} \\
\hat{E} & = \{ \{ \star_i \} \times E_1 \} \cup \{ \{ \star_j \} \times E_2 \} \\
\hat{\succ} & = \{ \{ \{ \star_i \} \times Z \} \cup \{ \{ \star_j \} \times \text{init}(\mathcal{E}_j) \}, (\star_i, e) \mid Z \succ_i e \land i \neq j \} \\
\hat{\mapsto} & = \{ \{ \{ \star_i \} \times X, (\star_i, e) \} \mid X \mapsto_i e \} \\
\hat{T} & = \{ \{ \{ \star_i \} \times X_1 \} \cup \{ \{ \star_j \} \times X_2 \} \mid X_1 \in T_1 \land X_2 \in T_2 \} \\
\hat{l}(\{ \star_i, e \}) & = l_i(e)
\end{align*}
\]
5.4. DENOTATIONAL SEMANTICS FOR \( \mathsf{PA}_{\mathsf{ST}} \)

\[ \uparrow : \mathsf{ETBES} \times \mathsf{ETBES} \to \mathsf{ETBES} \] with \( \mathcal{E}_1 \uparrow \mathcal{E}_2 = (\tilde{E}, \tilde{\succ}, \tilde{\mapsto}, \tilde{T}, \tilde{l}) \) where

\[
\begin{align*}
\tilde{E} &= \{(\{\star\} \times E_1) \cup (\{\star\} \times E_2) \}
\tilde{\succ} &= \{((\{\star\} \times (Z \cup X \cup (\{\star\} \times E_1) | Z \succ_1 e \wedge X \in T_1) \cup
(\{\star\} \times Z, (\star, e)) | Z \succ_2 e \}
\tilde{\mapsto} &= \{((\{\star\} \times X, (\star, e)) | X \mapsto_i e \}
\tilde{T} &= \{(\{\star\} \times X_2 | X_2 \in T_2 \}
\tilde{l}(\{\star\}, e)) = l_i(e)
\end{align*}
\]

\[ \downarrow : \mathsf{ETBES} \times \mathsf{ETBES} \to \mathsf{ETBES} \] with \( \mathcal{E}_1 \downarrow \mathcal{E}_2 = (\tilde{E}, \tilde{\succ}, \tilde{\mapsto}, \tilde{T}, \tilde{l}) \) where

\[
\begin{align*}
\tilde{E} &= \{(\{\star\} \times E_1) \cup (\{\star\} \times E_2) \}
\tilde{\succ} &= \{((\{\star\} \times Z) \cup (\{\star\} \times \text{init}(E_2)), (\star, e)) | Z \succ_1 e \}
\tilde{\mapsto} &= \{((\{\star\} \times X) \cup (\{\star\} \times Z), (\star, e)) | X \in T_1 \wedge Z \succ_2 e \}
\tilde{T} &= \{(\{\star\} \times X_1) \cup (\{\star\} \times X_2) | X_1 \in T_1 \wedge X_2 \in T_2 \}
\tilde{l}(\{\star\}, e)) = l_i(e)
\end{align*}
\]

\[ \llbracket \mathsf{A} : \mathsf{ETBES} \times \mathsf{ETBES} \to \mathsf{ETBES} \] with \( \mathcal{E}_1 \llbracket \mathsf{A} \mathcal{E}_2 = (\tilde{E}, \tilde{\succ}, \tilde{\mapsto}, \tilde{T}, \tilde{l}) \) where

\[
\begin{align*}
\tilde{E}^f &= \{(E_1^f \times \{\star\}) \cup (\{\star\} \times E_2^f) \cup E^s \}
E^s &= \{e \in E_i | l_i(e) \notin A \}
\tilde{\succ} &= \{((e_1, e_2) \in E_1 \times E_2 | l_1(e_1) = l_2(e_2)) \in A \}
\tilde{\mapsto} &= \{((e_1, e_2) \in E_1 \times Z_1 \cup X_1, (e_1, e_2)) | Z_1 \succ_1 e_1 \wedge X_1 \in T_1 \}
\tilde{T} &= \{(e_1, e_2) \in E_1 \cup \text{init}(E_2), (e_1, e_2)) | e_2 \in T_2 \}
\tilde{l}(e_1, e_2)) = \begin{cases} l_1(e_1) & \text{if } e_2 = \star \\
 l_2(e_2) & \text{otherwise} \end{cases}
\end{align*}
\]

\[ \widehat{\text{Lab}} : (\mathsf{ETBES} \times \mathcal{F}_{\mathcal{L}}) \to \mathsf{ETBES} \] with \( \widehat{\text{Lab}}(\mathcal{E}, f) = (E, \succ, \mapsto, T, f \circ l) \).

\[ \llbracket \mathsf{A} : \mathsf{ETBES} \to \mathsf{ETBES} \] with \( \mathcal{E} \llbracket \mathsf{A} \mathcal{E} = \mathcal{E} \cup \{e \in E | l(e) \notin A \} \).

We will give some comments on the definition of these operators: The definitions of the set of events, the causality relation and the relabeling function are the standard ones [125], except that the disjoint union is explicitly used (see Subsection 3.3.3). An event \((\star_1, e)\) from \( \mathcal{E}_1 \uparrow \mathcal{E}_2 \) has to be disabled if an event corresponding to \( E_2 \) is executed, where the event of \( E_2 \) has to be necessarily an initial event of \( \mathcal{E}_2 \). Therefore, the witness-bundles of \((\star_1, e)\) are obtained by extending the original ones with \( \text{init}(E_2) \). And, of course, similarly defined for \((\star_2, e)\) events.

The termination set of \( \mathcal{E}_1 \uparrow \mathcal{E}_2 \) is obtained by taking any combination of the elements of the two termination sets. This is intuitive, since an event is a termination event of \( \mathcal{E}_1 \uparrow \mathcal{E}_2 \) if and only if it is a termination event of \( \mathcal{E}_1 \) or \( \mathcal{E}_2 \).
An event \((*, e)\) from \(\mathcal{E}_1 \triangleright \mathcal{E}_2\) can not be disabled by events from \(\mathcal{E}_1\), and therefore we only take the original witness-bundles. On the other hand, we additionally disable an event \((*, e)\) by the termination of \(\mathcal{E}_1\). This is done in order to achieve consistency with the operational semantics. \(\mathcal{E}_1 \triangleright \mathcal{E}_2\) terminates if and only if a termination event from \(\mathcal{E}_2\) is executed. Hence, the termination set of \(\mathcal{E}_1 \triangleright \mathcal{E}_2\) is defined as the termination set of \(\mathcal{E}_2\).

The witness-bundle relation of \(\mathcal{E}_1 \succ \mathcal{E}_2\) is a combination of the ideas of \(\mathcal{E}_1 \triangleright \mathcal{E}_2\) and of \(\mathcal{E}_1 \triangleright \mathcal{E}_2\), since the \((*, e)\) events of \(\mathcal{E}_1 \succ \mathcal{E}_2\) are disabled by any \((*, e)\)-event and a \((*, e)\)-event is disabled by a termination event of \(\mathcal{E}_1\). Furthermore, the set of termination is similar to \(\mathcal{E}_1 \triangleright \mathcal{E}_2\).

An event \(e = (e_1, e_2)\) of \(\mathcal{E}_1 \parallel_A \mathcal{E}_2\) is disabled if \(e_1\) or \(e_2\) is disabled or if another event where one component is equal to a component of \(e\), like \((e_1, e_2')\), is executed. Furthermore, \(e\) is also disabled if one side of the parallel operator terminates and \(e\) is a synchronization event, i.e. \((e_1, e_2) \neq (*, e)\). This is intuitive, since a process that terminates can not execute any further action. Therefore, no synchronization can take place. The disabling mechanism described above is obtained by taking any combination of the corresponding sets, as it can be seen in the definition.

The set of termination for \(\mathcal{E}_1 \parallel_A \mathcal{E}_2\) is obtained by taking the union of the termination sets of its components, where we have to guarantee that all corresponding synchronization events are contained in this union.

The witness-bundles and the termination are unaffected by the labeling operator. The restriction operator removes all forbidden events, i.e. those labeled with elements of \(A\). This is also done in the witness-bundles and in the termination set.

**Lemma 5.21** All operators of Definition 5.20 are well defined, i.e. they really yield elements of ETBES.

**Proof:** The well-definedness of \(\mathcal{E}_1 \parallel_A \mathcal{E}_2\) follows from Lemma 5.17. The approximation closedness conditions of the witness relation of the sequential operator are a consequence of Corollary 2.19. The other conditions are easily seen except for the approximation closedness conditions of the parallel operator.

Therefore, let \(\tilde{\mathcal{E}} = \mathcal{E}_1 \parallel_A \mathcal{E}_2\).

\[\hat{\sim}: \text{Let } (e_1', e_2') \in E_{\mathcal{E}_1 \parallel_A \mathcal{E}_2}. \text{ Define } M_i = \begin{cases} \emptyset & \text{if } e_i' = * \\ \{ Z_i \cup X_i | Z_i \triangleright_i e_i' \land X_i \in T_i \} & \text{otherwise} \end{cases}.\]

Then \(M_i\) is approximation closed with respect to \(E_i \cup \{\ast\}\) by Corollary 2.19. From Corollary 2.20 we obtain that \(M' = \{(e_1, e_2) \in (E_1 \cup \{\ast\}) \times (E_2 \cup \{\ast\}) | e_1 \in X_1 \land e_2 \in X_2 \} | X_1 \in M_1 \land X_2 \in M_2\) is approximation closed with respect to \((E_1 \cup \{\ast\}) \times (E_2 \cup \{\ast\})\). And so the approximation closedness of \(\tilde{\sim}\) follows from Corollary 2.18, since \(\{X' \mid X' \tilde{\sim} (e_1', e_2')\} = \{X' \cap \tilde{E} \mid X' \in M'\}\).

\(\hookleftarrow\): The proof is similar to the one of Lemma 3.18.

\(\check{T}\): Analogous to \(\hookleftarrow\). \qed

**Lemma 5.22** All operators of Definition 5.20 are continuous with respect to \(\leq\).

**Proof:** Analogous to the proof of Lemma 3.18. \qed
The denotational semantics of some processes is illustrated in Figure 5.1. Suppose the is continuous with respect to: \[ \forall X \in T \cup \pi_1(\rightarrow) : \forall e, e' \in X : \forall Z : Z \succ e \Rightarrow e' \in Z. \]

But contrary to TBES, this constraint is to restrictive, since it is not preserved by the disrupt operator. This can be seen as follows. Consider \( E_3 \) of Figure 5.1. Then \( E_3 \) does not satisfy the bundle stability constraint, since \( \{a, c\} \in T_3 \) but \( \{b, c\} \succ c \). Furthermore, \( E_3 = E_1 \bowtie (\{\cdot\}, \{\{\cdot\}, \cdot\}, 0, \{\{\cdot\}\}, \{\cdot, c\}) \), where both components satisfy the bundle stability constraint.

### 5.4. Denotational Meaning for \( \text{PA}_{\text{st}} \)

First, we define the denotational semantics of expressions (\( \text{EXP}_{\text{st}} \)) with respect to variable assignments, i.e. functions from \( \text{Var} \) to \( \text{ETBES} \). Then variable assignments are derived from declarations, which are used to define the denotational semantics of processes (\( \text{PA}_{\text{st}} \)).

**Definition 5.24** Let \( \lbrack \cdot \rbrack : \text{EXP}_{\text{st}} \times (\text{Var} \rightarrow \text{ETBES}) \rightarrow \text{ETBES} \) be defined as follows (where \( \rho : \text{Var} \rightarrow \text{ETBES} \))

\[
\begin{align*}
\lbrack 0 \rbrack_\rho &= (\emptyset, \emptyset, \emptyset, \{\emptyset\}, \emptyset) \\
\lbrack B_1 + B_2 \rbrack_\rho &= \lbrack B_1 \rbrack_\rho \uplus \lbrack B_2 \rbrack_\rho \\
\lbrack B_1 > B_2 \rbrack_\rho &= \lbrack B_1 \rbrack_\rho \bowtie \lbrack B_2 \rbrack_\rho \\
\lbrack B[f] \rbrack_\rho &= \text{Lab}(\lbrack B \rbrack_\rho, f) \\
\lbrack x \rbrack_\rho &= \rho(x)
\end{align*}
\]

**Remark 5.25** \( \lbrack B \rbrack_\rho \) is continuous with respect to \( \preceq \) for every \( B \in \text{EXP}_{\text{st}} \).

Assume \( \text{decl} : \text{Var} \rightarrow \text{EXP}_{\text{st}} \). Then define \( \mathcal{F}_{\text{decl}} : (\text{Var} \rightarrow \text{ETBES}) \rightarrow (\text{Var} \rightarrow \text{ETBES}) \) with \( \mathcal{F}_{\text{decl}}(\rho)(x) = \lbrack \text{decl}(x) \rbrack_\rho \). From Remark 5.25 it follows that \( \mathcal{F}_{\text{decl}} \) is continuous. Therefore, from the complete partial order theory we get \( \lbrack \cdot \rbrack : (\text{Var} \rightarrow \text{EXP}_{\text{st}}) \rightarrow (\text{Var} \rightarrow \text{ETBES}) \) with \( \lbrack \{\text{decl}\} \rbrack = \text{fix}(\mathcal{F}_{\text{decl}}) = \bigsqcup_n \mathcal{F}_{\text{decl}}^n(\bot) \) is well defined.

**Definition 5.26 (Denotational Semantics)**

Define \( \lbrack \cdot \rbrack : \text{PA}_{\text{st}} \rightarrow \text{ETBES} \) by \( \lbrack \langle \text{decl}, B \rangle \rbrack = \lbrack B \rbrack_{\{\text{decl}\}} \).

**Example 5.27** The denotational semantics of some processes is illustrated in Figure 5.1.

The denotational semantics is consistent with the operational semantics, since the transition system derived from the denotational semantics is bisimilar to the operational semantics.

**Theorem 5.28 (Consistency)** Suppose \( \langle \text{decl}, B \rangle \in \text{PA}_{\text{st}} \). Then \( (\text{EXP}_{\text{st}}, \text{Act}^T, \rightarrow_{\text{decl}}^t, B) \) and \( (\text{ETBES}, \text{Act}^T, \\rightarrow, \lbrack \langle \text{decl}, B \rangle \rbrack) \) are bisimilar.

**Proof:** The proof is given in Subsection 5.6.1. \[\square\]
5.4.5 Extended Termination Precursor Event Structures (ETPES)

An event $e$ of an eTbes is caused (or disabled) in a system run if the run contains an element of every causality (respectively witness) bundle $X$ of $e$ (bundle approach). Another contrary way of modeling causality is Winkel's quantification approach [178]: An event $e$ is caused if there exists a 'causality set' $X$ of $e$ such that all elements of $X$ occur in the system run. Since this approach is a popular one, we introduce in this section another class of event structures, which again use a relation ($\prec$) between sets of events and events in order to model disabling. But this time, the causality and the disabling relation are interpreted with Winskel’s quantification approach. We show in Subsection 5.4.6 that these event structures and the extended termination bundle event structures are equivalent.

**Definition 5.29** $M$ is finitely determined with respect to $E$ if

- $E$ is a countable set
- $M$ is upper closed with respect to $E$, i.e. $\forall X, X' : (X \subseteq X' \land X \in M) \Rightarrow X' \in M$
- $\forall X \in M : \exists X' \in M : X' \subseteq X \land |X'| < |\mathbb{N}|$

**Definition 5.30 (Extended Termination Precursor Event Structure)** An extended termination precursor event structure, eTPes for short, $E = (\hat{\mathcal{E}}, \hat{\succ}, \hat{\rightarrow}, \hat{T}, \hat{l})$ is an element of $\mathcal{P}(\mathcal{E}) \times \mathcal{P}(\mathcal{P}(\mathcal{U}) \times \mathcal{U}) \times \mathcal{P}(\mathcal{P}(\mathcal{U}) \times \mathcal{U}) \times \mathcal{P}(\mathcal{P}(\mathcal{U})) \times (\mathcal{U} \rightarrow \text{Act})$ such that

- $\prec \subseteq \mathcal{P}(\hat{\mathcal{E}}) \times \hat{\mathcal{E}}$ and $\forall e \in \hat{\mathcal{E}} : \neg(\emptyset \prec e)$ and $\forall e \in \hat{\mathcal{E}} : \{e\} \prec e$
- $\rightarrow \subseteq \mathcal{P}(\hat{\mathcal{E}}) \times \hat{\mathcal{E}}$
- $\hat{T} \subseteq \mathcal{P}(\hat{\mathcal{E}}) \land \emptyset \notin T$
- $\text{dom}(\hat{l}) = \hat{\mathcal{E}}$
- $\forall e \in \hat{\mathcal{E}} : \prec e$ is finitely determined with respect to $\hat{\mathcal{E}}$
- $\forall e \in \hat{\mathcal{E}} : \rightarrow e$ is finitely determined with respect to $\hat{\mathcal{E}}$
- $\hat{T}$ is finitely determined with respect to $\hat{\mathcal{E}}$

Let ETPES denote the set of all extended termination precursor event structures.

$\prec$ is called the precursor conflict relation. The other components are called the same as those of the extended termination bundle event structures. The intuitive meaning of the precursor conflict relation is that event $e$ is disabled in a system run if there is a conflict precursor $Z$ of $e$ ($Z \prec e$) such that the system run contains all elements of $Z$. The intuitive meaning of the causality relation $\rightarrow$ and of the termination set $\hat{T}$ is similar to $\prec$. For example, a system run of an eTPes is terminated if there is an element $X$ of $\hat{T}$, where every element of $X$ appears in the system run.

The constraints imposed on the precursor conflict relation are: no event is immediately disabled ($\neg(\emptyset \prec e)$), since otherwise the event can be omitted. Furthermore, the execution of an event
5.4. DENOTATIONAL SEMANTICS FOR $\mathsf{PA_{ST}}$

5.4.1

Disables itself ($\{e\} \ni e$), i.e. every event can happen only once. The constraint $\emptyset \notin \check{T}$ on the termination set ensures that a precursor event structure may not terminate immediately, i.e. it can only terminate by executing an action. The three finitely determined constraints are used to guarantee that the order introduced later in this subsection yields a complete partial order.

Example 5.31 Some eTpes are shown in Figure 5.3. The five components are displayed similarly to the components of an eTbes (see Example 5.6), i.e. the conflict relation is depicted as wavy lines, the causality as straight lines. We depict a termination precursor by surrounding its events by a closed line. Furthermore, we do not draw the conflict precursors of the form $\{e\} \ni e$ and we omit the upper sets, e.g. in $\xi_3$ we do not draw the termination precursors $\{a, c\}$, $\{b, c\}$ and $\{a, b, c\}$, which can be derived from the termination precursor $\{c\}$.

Hereafter, $\xi$ is considered to be $(\hat{E}, \ni, \ni', \hat{T}, \hat{l})$, $\xi_i$ to be $(\hat{E}_i, \ni_i, \ni_i', \hat{T}_i, \hat{l}_i)$ and in general $\xi$ is considered to be $(\hat{E}_\xi, \ni_\xi, \ni_\xi', \hat{T}_\xi, \hat{l}_\xi)$.

Definition 5.32 Let $\xi$ be an eTpes. The set of initial events of $\xi$ is defined by

$$\hat{\text{init}}(\xi) = \{e \in \hat{E} \mid \emptyset \ni e\}.$$

The termination predicate $\hat{T} \subseteq \mathcal{P}(\mathcal{P}(\mathcal{U})) \times \mathcal{U}$ is defined by

$$\hat{T}(\hat{T}, e) \iff \{e\} \in \hat{T}.$$ 

In the following two subsections, we derive a transition system from an eTpes and provide a complete partial order on ETPES. These concepts will be used for the comparison of ETBES and ETPES.

Transition Systems from an eTpes.

The remainder of an eTpes is given as follows.

Definition 5.33 (Remainder of an eTpes) Let $\xi \in$ ETPES and $e \in \hat{\text{init}}(\xi)$. Then the remainder $\xi_{\lfloor e\rfloor}$ of $\xi$ is given by $(\hat{E}', \ni', \ni'_i, \hat{T}', \hat{l}')$ where
CHAPTER 5. TERMINATING BY ACTION EXECUTION

Figure 5.4: Transition System Derived from ETPES

\[
\begin{align*}
\hat{E}' &= \{ e' \in \hat{E} \mid \neg(\{e\} \Rightarrow e') \} \\
\hat{\succ}' &= \{(Z', e') \mid e' \in \hat{E}' \land Z' \subseteq \hat{E}' \land \exists Z : Z \Rightarrow e' \land Z' = Z \setminus \{e\} \} \\
\hat{\leadsto}' &= \{(X', e') \mid e' \in \hat{E}' \land X' \subseteq \hat{E}' \land \exists X : X \leadsto e' \land X' = X \setminus \{e\} \} \\
\hat{T}' &= \begin{cases} 
\{X' \mid X' \subseteq \hat{E}' \land \exists X : X' = X \setminus \{e\} \} & \text{if } \neg \hat{T}(\hat{T}, e) \\
\emptyset & \text{otherwise}
\end{cases} \\
\hat{t}' &= l \upharpoonright \hat{E}'
\end{align*}
\]

All events which are disabled by \( e \) are removed. Please remember that \( \{e\} \Rightarrow e \). Hence, \( e \) disables itself. After the execution of \( e \), we keep exactly those precursors that are completely contained in \( \hat{E}' \cup \{e\} \), since the other ones can not be contained in further system runs. In the definition of the termination set, we consider the case when an eTpes terminates by executing \( e \) separately in order to guarantee that the remainder of an eTpes is also an eTpes. This separation is necessary, since otherwise the empty set would be contained in the termination set, which is not allowed for an eTpes.

**Lemma 5.34** Let \( \xi \in \text{ETPES} \) and \( e \in \text{init}(\xi) \). Then \( \xi_{\{e\}} \in \text{ETPES} \).

**Proof:** Straightforward. \( \square \)

Analogous to Definition 5.11, the remainder can be used to define an interleaving semantics for ETPES, which is omitted here. The transition system obtained from \( \xi_3 \) of Figure 5.3 is presented in Figure 5.4.

**Complete Partial Order.**

We define the following order on ETPES.

**Definition 5.35** Let \( \xi_i \in \text{ETPES} \). Then \( \xi_1 \trianglelefteq \xi_2 \) if

- \( \hat{E}_1 \subseteq \hat{E}_2 \)
- \( \hat{\succ}_1 = \{(X, e) \in \hat{\succ}_2 \mid X \subseteq \hat{E}_1 \land e \in \hat{E}_1 \} \)
- \( \hat{\leadsto}_1 = \{(Z, e) \in \hat{\leadsto}_2 \mid Z \subseteq \hat{E}_1 \land e \in \hat{E}_1 \} \)
\[ \hat{T}_1 = \{ X \in \hat{T}_2 \mid X \subseteq \hat{E}_1 \} \]

\[ \hat{l}_1 = \hat{l}_2 \mid \hat{E}_1 \]

**Theorem 5.36** The set of all eTpes ordered by \( \hat{\leq} \) is an \( \omega \)-complete partial order, where the least upper bound of an \( \omega \)-chain \( (\mathcal{E}_i)_{i \in \mathbb{N}} \) is given by \( \bigcup_i \mathcal{E}_i = (\bigcup_i \hat{E}_i, \hat{\succ}, \hat{\mapsto}, \hat{T}, \bigcup_i \hat{l}_i) \) with

\[
\hat{\succ} = \{(Z, e) \in \mathcal{P}(\bigcup_i \hat{E}_i) \times (\bigcup_i \hat{E}_i) \mid \exists j : (Z \cap \hat{E}_j) \hat{\succ} e \}
\]

\[
\hat{\mapsto} = \{(X, e) \in \mathcal{P}(\bigcup_i \hat{E}_i) \times (\bigcup_i \hat{E}_i) \mid \exists j : (X \cap \hat{E}_j) \hat{\mapsto} e \}
\]

\[ \hat{T} = \{ X \in \mathcal{P}(\bigcup_i \hat{E}_i) \mid \exists j : (X \cap \hat{E}_j) \in \hat{T}_j \} \]

**Proof:** The proof is given in Subsection 5.6.2.

### 5.4.6 Correspondence between ETBES and ETPES.

We show that there is a continuous function from ETBES ordered by \( \leq \) to ETPES ordered by \( \hat{\leq} \) and vice versa. This result is used to show that ETBES and ETPES have the same expressive power with respect to event traces.

**Definition 5.37** Let \( F_E : \mathcal{P}(\mathcal{P}(E)) \to \mathcal{P}(\mathcal{P}(E)) \) be defined by

\[ F_E(M) = \{ \bar{X} \in \mathcal{P}(E) \mid \forall X \in M : X \cap \bar{X} \neq \emptyset \} \].

Define \( \mathcal{F} : \bar{M} \to \bar{M} \), where \( \bar{M} = \{(E, \hat{\succ}, \hat{\mapsto}, T, l) \mid T \subseteq \mathcal{P}(E) \wedge \text{dom}(l) = E \wedge \hat{\succ}, \hat{\mapsto} \subseteq \mathcal{P}(E) \times E \} \), by

\[
\mathcal{F}(E, \hat{\succ}, \hat{\mapsto}, T, l) = (E, \{ (\bar{Z}, e) \mid \bar{Z} \in F_E(\hat{\succ} e) \}, \{ (\bar{X}, e) \mid \bar{X} \in F_E(\hat{\mapsto} e) \}, F_E(T), l).
\]

**Example 5.38** The transformation of \( \mathcal{E}_3 \) of Figure 5.1 is \( \mathcal{E}_3 \) of Figure 5.3, i.e. \( \mathcal{F}(\mathcal{E}_3) = \mathcal{E}_3 \). The transformation of \( \mathcal{E}_3 \) yields \( \mathcal{E}_3 \) except that all upper sets are included in the conflict relation (respectively in the causality relation and the termination set), for example \( \{a, b, c\} \) is contained in the termination set.

**Proposition 5.39** Function \( \mathcal{F} \mid \text{ETBES} \) is a continuous function from \( \text{ETBES}, \leq \) into \( \text{ETPES}, \hat{\leq} \) and function \( \mathcal{F} \mid \text{ETPES} \) is a continuous function from \( \text{ETPES}, \hat{\leq} \) into \( \text{ETBES}, \leq \).

**Proof:** The proof is given in Subsection 5.6.3.

**Theorem 5.40** Every set of event traces that is described by ETBES is also described by ETPES and vice versa. More precisely, for all \( \mathcal{E} \in \text{ETBES} \) it holds that \( \mathcal{E} \) and \( \mathcal{F}(\mathcal{E}) \) describe the same set of event traces, and for all \( \mathcal{E} \in \text{ETPES} \) it holds that \( \mathcal{E} \) and \( \mathcal{F}(\mathcal{E}) \) describe the same set of event traces.

**Proof:** The proof is given in Subsection 5.6.4.
Corollary 5.41  Every set of event traces that is described by prime, flow, stable, bundle, extended bundle or dual event structures is also described by extended termination precursor event structures, but not vice versa.

**Proof:** Is an immediate consequence of Theorem 5.15 and Theorem 5.40.

Remark 5.42  Of course it is possible to use ETPES instead of ETBES as a model of the denotational semantics for our process algebra. The necessary operators can be defined explicitly, as it is done in Subsection 5.4.3 for ETBES. Another possibility is to define the operators on ETPES through the operators on ETBES, i.e. define, for example, the parallel operator \( \parallel \) on ETPES by \( \xi \parallel A \xi' = \mathcal{F} (\xi) \parallel A \mathcal{F} (\xi') \). These operators are continuous by Proposition 5.39 and Lemma 5.22. The denotational semantics that is obtained in this way is illustrated in Figure 5.3.

5.5  Discussion

In this chapter, we have investigated new kinds of event structures in order to give denotational semantics to process algebras that are based on the fa-philosophy and that contain disruption. The motivation of such an approach results from the fact that it is more reasonable to have an fa-philosophy in end-based settings, since otherwise the intuitive equivalences fail to be the coarsest (Subsection 4.2.6). Note that it is possible to model disruption with action refinement in end-based settings.

Sets of events may disable events in both event structures presented in this chapter. One of these event structures is based on the bundle technique the other one is based on Winskel’s approach. We have shown that these two event structures are equivalent approaches. Furthermore, we have used one of them to give a denotational semantics to a process algebra that is based on the fa-philosophy and that contains disruption. Moreover, we have shown that this denotational semantics is consistent with the standard operational semantics.

In the following chapter, we define the action refinement operator on ETBES with respect to the end-based view. In addition, we adapt two of the newly introduced equivalences of Chapter 4 to ETBES and show that they yield the coarsest equivalences with respect to trace (respectively bisimulation) equivalence for the end-based action refinement operator.

5.6  Proofs

5.6.1  Proof of Theorem 5.28.

The proof is analogous to the proof of Theorem 3.25, i.e. we introduce an event based transition relation. Then we show that this transition system is bisimilar to \( (\text{EXP}_{st}, \mathcal{A}ct^T, \xrightarrow{i_{\text{decl}}}, B) \) and that it is, in addition, bisimilar to \( (\text{ETBES}, \mathcal{A}ct^T, \xrightarrow{\langle \text{decl}, B \rangle}), \ldots \). Hence, Theorem 5.28 follows by the transitivity of bisimilarity.
5.6. PROOFS

Event Based Transition System.

Let $\text{EXP}_{\text{st}}$ be the set that contains exactly those elements generated by

$$C ::= B \mid C; B \mid C \mid B \mid C \mid B \mid C \mid C[f] \mid C \backslash A \mid [C]_i$$

where $B \in \text{EXP}_{\text{st}}$, $f \in F_L$, $i \in \{1, 2, l, r\}$ and $A \subseteq \text{Obs}$. We do not need to extend the declaration, i.e. we define $\text{PA}_{\text{st}} = (\text{Var} \to \text{EXP}_{\text{st}}) \times \text{EXP}_{\text{st}}$.

In Table 5.2, the event transition rules $\rightarrow_{\text{decl}}$ are presented.

The First Bisimilarity Result.

An expression $C$ of $\text{EXP}_{\text{st}}$ and an expression $B$ of $\text{EXP}_{\text{st}}$ are related if we obtain $B$ by removing all $\lfloor . \rfloor$ expressions from $C$. This is formalized by the following function, where we also count the $\lfloor . \rfloor$ symbols in $C$.

**Definition 5.43** $\Xi : \mathbb{N} \times \text{EXP}_{\text{st}} \to \mathcal{P}(\text{EXP}_{\text{st}})$ is defined as follows, where $I = \{1, 2, l, r\}$.

$$
\begin{align*}
\Xi(0, B) &= \{B\} \\
\Xi(n + 1, B) &= \{[\tilde{C}]_i \mid i \in I \land \tilde{C} \in \Xi(n, B)\} \quad \text{if } B \in \{0, a, B_1 + B_2, x\} \\
\Xi(n + 1, B_1; B_2) &= \{[\tilde{C}]_i \mid i \in I \land \tilde{C} \in \Xi(n, B_1; B_2)\} \cup \\
&\quad \{C_1; B_2 \mid C_1 \in \Xi(n + 1, B_1)\} \\
\Xi(n + 1, B_1; B_2) &= \{[\tilde{C}]_i \mid i \in I \land \tilde{C} \in \Xi(n, B_1; B_2)\} \cup \\
&\quad \{C_1 \mid B_2 \mid C_1 \in \Xi(n + 1, B_1)\} \\
\Xi(n + 1, B_1; B_2) &= \{[\tilde{C}]_i \mid i \in I \land \tilde{C} \in \Xi(n, B_1; B_2)\} \cup \\
&\quad \{C_1 \mid B_2 \mid C_1 \in \Xi(n + 1, B_1)\} \\
\Xi(n + 1, B_1; B_2) &= \{[\tilde{C}]_i \mid i \in I \land \tilde{C} \in \Xi(n, B_1; B_2)\} \cup \\
&\quad \{C_1 \mid B_2 \mid C_1 \in \Xi(n + 1, B_1)\} \\
\Xi(n + 1, B_1; B_2) &= \{[\tilde{C}]_i \mid i \in I \land \tilde{C} \in \Xi(n, B_1; B_2)\} \cup \\
&\quad \{C_1 \mid B_2 \mid C_1 \in \Xi(n + 1, B_1)\} \\
\Xi(n + 1, B) &= \{[\tilde{C}]_i \mid i \in I \land \tilde{C} \in \Xi(n, B)\} \cup \\
&\quad \{C_1 \mid B \mid C_1 \in \Xi(n + 1, B)\}
\end{align*}
$$

The well-definedness of $\Xi$ is easily seen.

**Lemma 5.44** Let $B \in \text{EXP}_{\text{st}}$, then $(\text{EXP}_{\text{st}}, \text{Act}^T, \rightarrow_{\text{decl}}^t, B)$ and $(\text{EXP}_{\text{st}}, \text{Act}^T, \rightarrow_{\text{decl}}^\prime, B)$ are bisimilar, where $C \xrightarrow{\gamma} \rightarrow_{\text{decl}}^t C' \Leftrightarrow \exists e \in U : C \xrightarrow{\gamma} e \rightarrow_{\text{decl}} C'$.

**Proof:** Define $\mathcal{R} = \{(B, C) \in \text{EXP}_{\text{st}} \times \text{EXP}_{\text{st}} \mid \exists n : C \in \Xi(n, B)\}$. In order to verify that $\mathcal{R}$ is a bisimulation, we show

$$
(B \xrightarrow{\gamma} B' \land C \in \Xi(n, B)) \Rightarrow \exists e, C', m : C \xrightarrow{\gamma} e \rightarrow_{\text{decl}}^t C' \land C' \in \Xi(m, B')
$$

The proof of (5.1) works by induction on the depth of inference of $B \xrightarrow{\gamma} B'$ combined with the value of $n$. Then (5.1) can be easily checked through the following procedure:

- applying rule $N_{12}$ or $N_{rl}$ whenever $C = [\tilde{C}]_i$. In these cases, $n$ is reduced by one and $B \xrightarrow{\gamma} B'$ remains unaffected. Therefore, the hypothesis yields the result.
In the following let $\gamma$ be an element of $\mathcal{Act}^T$ and $a$ be an element of $\mathcal{Act}$

$$A_1 : \quad a \xrightarrow{\ast} 0 \quad C : \quad \frac{B_1 \xrightarrow{\gamma} C' \quad B_1 + B_2 (\gamma_1, \gamma_2) [C']_1 \quad B_2 + B_1 (\gamma_2, \gamma_1) [C']_2}{C}$$

$$S_1 : \quad \frac{C \xrightarrow{a} C'}{C; B (\epsilon_1, \epsilon_2) C'} \quad B \quad S_2 : \quad \frac{C \xrightarrow{a} C'}{C; B (\epsilon_1, \epsilon_2) [B]_2}$$

$$I_1 : \quad \frac{C \xrightarrow{a} C'}{C [\Gamma] B (\epsilon_1, \epsilon_2) C'' [\Gamma]} \quad I_2 : \quad \frac{C \xrightarrow{a} C'}{C [\Gamma] B (\epsilon_1, \epsilon_2) [C']_1} \quad I_3 : \quad \frac{C \xrightarrow{a} C'}{C [\Gamma] B (\epsilon_1, \epsilon_2) [C']_2}$$

$$P_1 : \quad \frac{C_1 \xrightarrow{a} C_1'}{C_1 \parallel_C C_2 (\epsilon_1, \epsilon_2) C_1' \parallel_C C_2} \quad a \not\in A \quad P_2 : \quad \frac{C_1 \xrightarrow{a} C_1'}{C_1 \parallel_C C_2 (\epsilon_1, \epsilon_2) (\lfloor C_2 \rfloor_r) \backslash \\backslash A} \quad \frac{C_1 \xrightarrow{a} C_1'}{C_2 \parallel_C C_2 (\epsilon_1, \epsilon_2) (\lfloor C_2 \rfloor_l) \backslash \\backslash A}$$

$$P_3 : \quad \frac{C_1 \xrightarrow{a} C_1'}{C_1 \parallel_C C_2 (\epsilon_1, \epsilon_2) C_1' \parallel_C A C_2} \quad a \in A \quad P_4 : \quad \frac{C_1 \xrightarrow{a} C_1'}{C_1 \parallel_C C_2 (\epsilon_1, \epsilon_2) (\lfloor C_2 \rfloor_r) \backslash \\backslash A} \quad \frac{C_1 \xrightarrow{a} C_1'}{C_2 \parallel_C C_2 (\epsilon_1, \epsilon_2) (\lfloor C_2 \rfloor_l) \backslash \\backslash A}$$

$$P_5 : \quad \frac{C_1 \xrightarrow{a} C_1'}{C_1 \parallel_C C_2 (\epsilon_1, \epsilon_2) 0} \quad a \in A$$

$$\text{Lab}_1 : \quad \frac{C \xrightarrow{f(a)} C'}{C[f]} \quad \text{Lab}_2 : \quad \frac{C \xrightarrow{f(a)} C'}{C[f]}$$

$$\text{Res}_1 : \quad \frac{C \xrightarrow{a} C'}{C \backslash A} \quad a \not\in A \quad \frac{C \xrightarrow{a} C'}{C \backslash A} \quad \frac{C \xrightarrow{a} C'}{C \backslash A}$$

$$\text{Rec} : \quad \frac{\text{decl}(x) \xrightarrow{\gamma} C'}{x \xrightarrow{\gamma} C'} \quad N_{12} : \quad \frac{C \xrightarrow{\gamma} C'}{[C]_i} \quad N_{rt} : \quad \frac{C \xrightarrow{\gamma} C'}{[C]_r (\epsilon_1, \epsilon_2) [C']_r} \quad \frac{C \xrightarrow{\gamma} C'}{[C]_l (\epsilon_1, \epsilon_2) [C']_l}$$

Table 5.2: Event Based Transition Rules with respect to $\xrightarrow{\ast}_{\text{decl}}$

- applying the corresponding rules of $\xrightarrow{\gamma} B'$ whenever $C$ is different to $[\tilde{C}]_i$. In these cases, the depth of inference is reduced and $n$ gets not increased. Therefore, the hypothesis yields the result.
Another fact is

\[(C \xrightarrow{\gamma \cdot l_{\text{decl}}}' C' \land C \in \Xi(n, B)) \Rightarrow \exists B', m : B \xrightarrow{\gamma} B' \land C' \in \Xi(m, B') \tag{5.2}\]

This equation can be proved by induction on the depth of inference of \(C \xrightarrow{\gamma \cdot l_{\text{decl}}}' C'\).

Now we are ready to verify that \(\mathcal{R}\) is a bisimulation:

- It is clear that \((B, B) \in \mathcal{R}\).
- Suppose \((B_1, C_1) \in \mathcal{R}\) and \(B_1 \xrightarrow{\gamma} B_2\). Then \(\exists e, C_2, m : C_1 \xrightarrow{\gamma \cdot l_{\text{decl}}} C_2 \land C_2 \in \Xi(m, B_2)\) by (5.1). Thus \(C_1 \xrightarrow{\gamma} C_2\) and \((B_2, C_2) \in \mathcal{R}\), as required.
- Suppose \((B_1, C_1) \in \mathcal{R}\) and \(C_1 \xrightarrow{\gamma} C_2\). Then \(C_1 \xrightarrow{\gamma \cdot l_{\text{decl}}} C_2\) for some \(e\). Hence, \(\exists B_2, m : B_1 \xrightarrow{\gamma} B_2 \land C_2 \in \Xi(m, B_2)\) by (5.2). \(\square\)

**The Second Bisimilarity Result.**

First, we show that the denotation of a variable is the same as the denotation of its corresponding expression.

**Lemma 5.45** Let \(\text{decl} : \text{Var} \to \text{EXP}_{st}\) and \(x \in \text{Var}\). Then \([\langle \text{decl}, x \rangle] = [\langle \text{decl}, \text{decl}(x) \rangle]\).

**Proof:** Similar to the proof of Lemma 3.37. \(\square\)

We extend the denotational semantics to \(\text{PA}_{st}\).

**Definition 5.46 (Denotational semantics of \(\text{PA}_{st}\))** Let \(i \in \{1, 2\}\) then

\[
\text{Shift}_i : \text{ETBES} \to \text{ETBES} \text{ with } \tilde{\text{Shift}}_i(\mathcal{E}) = (\tilde{E}, \tilde{\prec}, \tilde{i}, \tilde{T}, \tilde{l}) \text{ where}
\]

- \(\tilde{E} = \{\ast_i\} \times E\)
- \(\tilde{\prec} = \{(\{\ast_i\} \times Z, (\ast_i, e)) \mid Z \succ e\}\)
- \(\tilde{i} = \{(\{\ast_i\} \times X, (\ast_i, e)) \mid X \mapsto_{i} e\}\)
- \(\tilde{T} = \{\{\ast_i\} \times X \mid X \in T\}\)
- \(\tilde{l}(\ast_i, e) = l(e)\)

\[
\tilde{\text{Shift}}_i : \text{ETBES} \to \text{ETBES} \text{ with } \tilde{\text{Shift}}_i(\mathcal{E}) = (\tilde{E}, \tilde{\prec}, \tilde{i}, \tilde{T}, \tilde{l}) \text{ where}
\]

- \(\tilde{E} = \{\ast\} \times E\)
- \(\tilde{\prec} = \{(\{\ast\} \times Z, (\ast, e)) \mid Z \succ e\}\)
- \(\tilde{i} = \{(\{\ast\} \times X, (\ast, e)) \mid X \mapsto_{i} e\}\)
- \(\tilde{T} = \{\{\ast\} \times X \mid X \in T\}\)
- \(\tilde{l}(\ast, e) = l(e)\)

\[
\tilde{\text{Shift}}_i : \text{ETBES} \to \text{ETBES} \text{ with } \tilde{\text{Shift}}_i(\mathcal{E}) = (\tilde{E}, \tilde{\prec}, \tilde{i}, \tilde{T}, \tilde{l}) \text{ where}
\]

- \(\tilde{E} = E \times \{\ast\}\)
- \(\tilde{\prec} = \{(Z \times \{\ast\}, (e, \ast)) \mid Z \succ e\}\)
- \(\tilde{i} = \{(X \times \{\ast\}, (e, \ast)) \mid X \mapsto_{i} e\}\)
- \(\tilde{T} = \{X \times \{\ast\} \mid X \in T\}\)
- \(\tilde{l}(e, \ast) = l(e)\)
Furthermore, define $[.]': \text{PA}_st^e \rightarrow \text{ETBES}$ by

$$
\begin{align*}
[(\text{decl}, B)]' &= [(\text{decl}, B)]' \\
[(\text{decl}, C; B)]' &= [(\text{decl}, C)]'[\leftarrow][(\text{decl}, B)]' \\
[(\text{decl}, C > B)]' &= [(\text{decl}, C)]'[>][(\text{decl}, B)]' \\
[(\text{decl}, C_1 \parallel_A C_2)]' &= [(\text{decl}, C_1)]'[\parallel_A][(\text{decl}, C_2)]' \\
[(\text{decl}, C \langle_1 f \rangle)]' &= \text{Lab}([(\text{decl}, C)], f) \\
[(\text{decl}, C \\setminus A)]' &= [(\text{decl}, C_2)]'[\setminus A] \\
\end{align*}
$$

It is easy to check that $[.]'$ is well defined.

**Lemma 5.47** Suppose $\mathcal{E}, \mathcal{E}_1, \mathcal{E}_2 \in \text{ETBES}$. Then

$$
\begin{align*}
(\mathcal{E}_1 \hat{\lor} \mathcal{E}_2)[e, e] &\cong \hat{\text{Shift}_1}(\mathcal{E}[e]) \\
(\mathcal{E}_1 \hat{\land} \mathcal{E}_2)[e, e] &\cong \begin{cases} 
\hat{\text{Shift}_2}(\mathcal{E}_2) & \text{if } e \in \text{init}(\mathcal{E}_1) \land \Upsilon(T_1, e) \\
\mathcal{E}[e] \hat{\land} \mathcal{E}_2 & \text{otherwise}
\end{cases} \\
(\mathcal{E}_1 \hat{\rightarrow} \mathcal{E}_2)[e, e] &\cong \begin{cases} 
\hat{\text{Shift}_2}(\mathcal{E}_2[e]) & \text{if } i = 2 \\
\hat{\text{Shift}_1}(\mathcal{E}[e]) & \text{if } i = 1 \land \Upsilon(T_1, e) \\
\mathcal{E}[e] \hat{\rightarrow} \mathcal{E}_2 & \text{otherwise}
\end{cases} \\
(\mathcal{E}_1 \hat{\parallel}_A \mathcal{E}_2)[e, e] &\cong \begin{cases} 
\mathcal{E}[e] \hat{\parallel}_A \mathcal{E}_2 & \text{if } \neg \Upsilon(T_1, e_1) \land \notA T_1, e_1 \land l_1(e_1) \notin A \\
\hat{\text{Shift}_r}(\mathcal{E}_2) & \text{if } e_1 \in \text{init}(\mathcal{E}_1) \land \Upsilon(T_1, e_1) \land l_1(e_1) \notin A
\end{cases} \\
(\mathcal{E}_1 \hat{\parallel}_A \mathcal{E}_2)[e_2, e_2] &\cong \begin{cases} 
\mathcal{E}[e_2] \hat{\parallel}_A \mathcal{E}_2[2] & \text{if } \neg \Upsilon(T_1, e_2) \land l_2(e_2) \notin A \\
\hat{\text{Shift}_r}(\mathcal{E}_2[2]) & \text{if } e_2 \in \text{init}(\mathcal{E}_1) \land \Upsilon(T_1, e_2) \land l_2(e_2) \notin A
\end{cases} \\
(\mathcal{E}_1 \hat{\parallel}_A \mathcal{E}_2)[e_1, e_2] &\cong \begin{cases} 
\mathcal{E}[e_1] \hat{\parallel}_A \mathcal{E}_2[e_2] & \text{if } \neg \Upsilon(T_1, e_1) \land \neg \Upsilon(T_2, e_2) \\
\hat{\text{Shift}_r}(\mathcal{E}_2[e_2]) & \text{if } e_1 \in \text{init}(\mathcal{E}_1) \land \Upsilon(T_1, e_1) \land \neg \Upsilon(T_2, e_2) \\
\hat{\text{Shift}_1}(\mathcal{E}[e_1]) & \text{if } e_2 \in \text{init}(\mathcal{E}_1) \land \Upsilon(T_1, e_1) \land \neg \Upsilon(T_2, e_2) \\
(\emptyset, \emptyset, \emptyset, \emptyset, \emptyset) & \text{if } \Upsilon(T_1, e_1) \land \Upsilon(T_2, e_2)
\end{cases}
\end{align*}
$$

whenever $l_1(e_1) = l_2(e_2) \in A$

$$
\begin{align*}
\text{Lab}(\mathcal{E}, f)[e] &\cong \text{Lab}(\mathcal{E}[e], f) \\
(\mathcal{E} \setminus A)[e] &\cong \begin{cases} 
\mathcal{E}[e] \setminus A & \text{if } l(e) \notin A \\
defined & \text{otherwise}
\end{cases} \\
\hat{\text{Shift}_1}(\mathcal{E})[e, e] &\cong \hat{\text{Shift}_1}(\mathcal{E}[e]) \\
\hat{\text{Shift}_1}(\mathcal{E})[e, e] &\cong \hat{\text{Shift}_1}(\mathcal{E}[e]) \\
\hat{\text{Shift}_1}(\mathcal{E})[e, e] &\cong \hat{\text{Shift}_1}(\mathcal{E}[e])
\end{align*}
$$

**Proof:** Straightforward.

**Lemma 5.48** Suppose $(\text{decl}, C) \in \text{PA}_st^e$ and $C \not\rightarrow_{\text{decl}} C'$. Then

$$
e \in \text{init}(\mathcal{E}) \land \mathcal{E}' = \mathcal{E}[e] \land (\gamma \notin \text{Act} \leftrightarrow \Upsilon(T, e)) \land l(e) = \begin{cases} 
\gamma & \text{if } \gamma \in \text{Act} \\
a & \text{if } \gamma = a \sqrt{a}
\end{cases}
$$

with $\mathcal{E} = [(\text{decl}, C)]'$ and $\mathcal{E}' = [(\text{decl}, C')]'$.  


Proof: We use induction on the depth of inference of \( C \overset{\gamma'}{\xrightarrow{\text{decl}}} C' \). Then the equation can be verified by case analysis on the derivation rules, where Lemma 5.47 is used. In the case of \( \text{Rec}' \) we make use of Lemma 5.45.

\( \square \)

**Lemma 5.49** Let \( \langle \text{decl}, C \rangle \in \text{PA}^e_{\text{st}}, E = [\{\text{decl}, C\}]' \) and \( e \in \text{init}(E) \). Then

\[
\exists C' \in \text{EXP}^e_{\text{st}} : C \overset{\gamma'}{\xrightarrow{\text{e}} \text{decl}} C' \wedge \gamma = \begin{cases}
  l(e) & \text{if } \neg \Upsilon(T, e) \\
  l(e) & \text{if } \Upsilon(T, e)
\end{cases}
\]

**Proof:** First we show for any \( \text{decl} : \text{Var} \rightarrow \text{EXP}_{\text{st}} \):

\[
\forall n \in \mathbb{N} : \forall B \in \text{EXP}_{\text{st}} : e \in \text{init}([B]_{\text{F}^n_{\text{decl}}(\bot)}) \Rightarrow \left( \exists C' \in \text{EXP}^e_{\text{st}} : B \overset{\gamma'}{\xrightarrow{\text{e}} \text{decl}} C' \wedge
\gamma = \begin{cases}
  \pi_5([B]_{\text{F}^n_{\text{decl}}(\bot)}(e)) & \text{if } \neg \Upsilon(\pi_4([B]_{\text{F}^n_{\text{decl}}(\bot)}), e) \\
  \pi_5([B]_{\text{F}^n_{\text{decl}}(\bot)}(e)) & \text{if } \Upsilon(\pi_4([B]_{\text{F}^n_{\text{decl}}(\bot)}), e)
\end{cases}
\right)
\]

(5.3)

This is done by induction on \( n \) combined with the structure of \( B \) where the lexicographical order is used. Furthermore, a case analysis on the structure of \( B \) is used. We only present here the case \( B = x : e \in \text{init}([x]_{\text{F}^n_{\text{decl}}(\bot)}) \) implies that \( n > 0 \). Therefore, \( [x]_{\text{F}^n_{\text{decl}}(\bot)} = \text{F}^{n-1}_{\text{decl}}(\bot)(x) = [\text{decl}(x)]_{\text{F}^{n-1}_{\text{decl}}(\bot)} \). The rest follows by induction, since \( n \) is reduced. Thus (5.3) is established.

The main statement follows now by structural induction on \( C \). We only present the case \( C = B \in \text{EXP}_{\text{st}} \). By Remark 5.25 we get \( [\langle \text{decl}, B \rangle] = \bigcup [B]_{\text{F}^n_{\text{decl}}(\bot)} \). Then it is easily seen that there is \( m \) such that \( e \in \text{init}([B]_{\text{F}^m_{\text{decl}}(\bot)}) \) and \( \gamma = \pi_5([B]_{\text{F}^m_{\text{decl}}(\bot)}) \). And so the result follows by (5.3).

\( \square \)

**Lemma 5.50** Let \( \langle \text{decl}, C \rangle \in \text{PA}^e_{\text{st}}, \text{then the transition systems } (\text{EXP}^e_{\text{st}}, \text{Act}^T, \rightarrow^\prime, C) \) and \( (\text{E}^T_{\text{BES}}, \text{Act}^T, \rightarrow, [\langle \text{decl}, C \rangle]') \) are bisimilar, where \( \rightarrow^\prime \) is defined as in Lemma 5.44.

**Proof:** Define \( \mathcal{R} = \{ ((C', [\langle \text{decl}, C' \rangle]')) : C' \in \text{EXP}^e_{\text{st}} \} \). Then \( (C, [\langle \text{decl}, C \rangle]') \in \mathcal{R} \) by definition.

Suppose \( C_1 \in \text{EXP}^e_{\text{st}} \) and \( C_1 \overset{\gamma'}{\xrightarrow{\text{e}} \text{decl}} C_2 \). Then \( C_1 \overset{\gamma}{\xrightarrow{\text{decl}}} C_2 \) for some \( e \). Hence, by Lemma 5.48 we get \( [\langle \text{decl}, C_1 \rangle] \overset{\gamma}{\xrightarrow{\text{decl}}} [\langle \text{decl}, C_2 \rangle] \), as required.

Suppose \( C_1 \in \text{EXP}^e_{\text{st}} \) and \( [\langle \text{decl}, C_1 \rangle] \overset{\gamma}{\xrightarrow{\text{decl}}} \mathcal{E}_2 \). Then there is \( e \in \text{init}([\langle \text{decl}, C_1 \rangle]) \) such that \( \mathcal{E}_2 = [\langle \text{decl}, C_1 \rangle]_e \) and \( \gamma = \begin{cases}
  l(e) & \text{if } \neg \Upsilon(T, e) \\
  l(e) & \text{if } \Upsilon(T, e)
\end{cases} \). From Lemma 5.49 we get the existence of \( C_2 \in \text{EXP}^e_{\text{st}} \) such that \( C_1 \overset{\gamma}{\xrightarrow{\text{decl}}} C_2 \). Moreover, \( [\langle \text{decl}, C_1 \rangle]_e = [\langle \text{decl}, C_2 \rangle]_e \) by Lemma 5.48, which concludes the proof.

\( \square \)

### 5.6.2 Proof of Theorem 5.36.

It is easily seen that \( \preceq \) is a partial order with \( (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset) \) as its least element. Furthermore, \( \bigcup E_i \) is an eTpe. In the following, we only consider \( \hat{T} \). The cases \( \hat{\rightarrow} \) and \( \rightarrow \) follow analogously.

Upper bound: Obviously, \( \hat{T}_j \subseteq \{ X \in \hat{T} : X \subseteq \hat{E}_j \} \).

Let \( X \subseteq \bigcup E_i \) such that \( X \subseteq \hat{E}_j \) and \( \exists i : (X \cap \hat{E}_i) \in \hat{T}_i \).

If \( i \geq j \) then \( \hat{E}_i \subseteq \hat{E}_j \). Thus \( X \cap \hat{E}_i = X \), hence \( X \in \hat{T}_i \). Moreover, \( X \in \hat{T}_j \), since \( \hat{E}_i \subseteq \hat{E}_j \).

If \( i < j \) then \( \hat{E}_i \subseteq \hat{E}_j \). Thus \( X \cap \hat{E}_i \in \hat{T}_j \). Since \( \hat{T}_j \) is finitely determined, we get \( X = (X \cap \hat{E}_j) \in \hat{T}_j \).
Least upper bound: Let \( \xi' \) be an eTpes such that \( \xi_i \preceq \xi' \) for all \( i \in \mathbb{N} \). Then \( \bigcup_i \hat{E}_i \subseteq \hat{E}' \).

Let \( X \in \hat{T} \). Then \( \exists j : (X \cap \hat{E}_j) \in \hat{T}_j \). Hence, \( (X \cap \hat{E}_j) \in \hat{T}' \), since \( \xi_j \preceq \xi' \). Thus \( X \in \hat{T}' \), since \( \hat{T}' \) is finitely determined.

Let \( X \in \hat{T}' \) such that \( X \subseteq \bigcup_i \hat{E}_i \). Since \( \hat{T}' \) is finitely determined, there is \( X' \subseteq \hat{T}' \) such that \( X' \subseteq X \land |X'| < |\mathbb{N}| \). Therefore, there exists \( j \in \mathbb{N} \) such that \( X' \subseteq \hat{E}_j \). Hence, \( X' \in \hat{T}_j \). Then by definition \( X' \in \hat{T} \). Thus \( X \in \hat{T} \), since \( \hat{T} \) is finitely determined.

Hence, Theorem 5.36 is established.

### 5.6.3 Proof of Proposition 5.39.

The following lemmas show that the constraints on eTbes are transformed into the constraints on eTpes and vice versa.

**Lemma 5.51** We have

(i) \( M \neq \emptyset \iff \emptyset \notin F_E(M) \)

(ii) \( (\forall X \in M : e \in X) \iff \{e\} \in F_E(M) \)

(iii) \( \emptyset \notin \hat{M} \iff F_E(\hat{M}) \neq \emptyset \)

(iv) \( \{e\} \in \hat{M} \Rightarrow (\forall X \in F_e(\hat{M}) : e \in X) \)

(v) \( (\{e\} \notin \hat{M} \land \emptyset \notin \hat{M}) \Rightarrow (\exists X \in F_e(\hat{M}) : e \notin X) \)

**Proof:** Straightforward. \( \square \)

**Lemma 5.52** If \( M \) is approximation closed with respect to \( E \) then \( F_E(M) \) is finitely determined with respect to \( E \).

**Proof:** Suppose \( \hat{X} \in F_E(M) \). From the definition of \( F_E \) it follows that all upper sets of \( \hat{X} \) are in \( F_E(M) \). Now suppose \( |\hat{X}| = |\mathbb{N}| \). Let \( \hat{X} = \{e_i : i \in \mathbb{N}\} \). Define \( \hat{X}' = \{e_i : \exists X \in M : e_i \in X \land \forall j < i : e_j \notin X\} \). Then \( \hat{X}' \subseteq \hat{X} \) and \( \hat{X}' \in F_E(M) \).

Assume \( |\hat{X}'| = |\mathbb{N}| \). Then for all \( e_i \in \hat{X}' \) there exists \( X_i \in M \) such that \( e_i \in X_i \) and \( \{e_1, ..., e_{i-1}\} \cap X_i = \emptyset \). Let \( \hat{X} = \mathcal{X}'((X_i)_{e_i \in \hat{X}'}, \kappa, (X_i)_{e_i \in \hat{X}'}, \kappa) \), where \( \kappa : \mathbb{N} \rightarrow E \) is bijective. Then \( \hat{X} \) only contains elements that are contained infinitely often in \( (X_i)_{e_i \in \hat{X}'} \), by the definition of \( \mathcal{X}' \) (Definition 2.16). Furthermore, \( \hat{X} \in M \) by Proposition 2.17. Therefore, \( \forall i : e_i \notin \hat{X} \), since \( e_i \) only appears finitely often in \( (X_i)_{e_i \in \hat{X}'} \). Hence, \( \hat{X} \cap \hat{X} = \emptyset \), which contradicts \( \hat{X} \in F_E(M) \). Thus \( |\hat{X}'| < |\mathbb{N}| \) as required. \( \square \)

**Lemma 5.53** If \( \hat{M} \) is finitely determined with respect to \( E \) then \( F_E(\hat{M}) \) is approximation closed with respect to \( E \).
Proof: Suppose $X \subseteq E$ and $(E_i)_{i \in \mathbb{N}}$ is a finite, monotone approximation of $E$ such that

$$\forall k \in \mathbb{N}: \exists X_k : X_k \cap E_k = X \cap E_k \land (\forall \hat{X} \in \hat{M} : X_k \cap \hat{X} \neq \emptyset).$$  \hspace{1cm} (5.4)

Let $\hat{X} \in \hat{M}$. Then there is $\hat{X}' \in \hat{M}$ such that $\hat{X}' \subseteq \hat{X} \land |\hat{X}'| < |\mathbb{N}|$, since $\hat{M}$ is finitely determined. Thus, there is $n \in \mathbb{N}$ such that $\hat{X}' \subseteq E_n$. From (5.4) we obtain $\emptyset \neq X_n \cap \hat{X}' \subseteq X_n \cap E_n \cap \hat{X}' = X \cap E_n \cap \hat{X}' \subseteq X \cap \hat{X}' \subseteq X \cap \hat{X}$. Hence, $\forall \hat{X} \in \hat{M} : X \cap \hat{X} \neq \emptyset$, as required. \hfill \Box

Proof of Proposition 5.39: That $\mathcal{F}(\text{ETBES}) \subseteq \text{ETPES}$ and $\mathcal{F}(\text{ETPES}) \subseteq \text{ETBES}$ is an immediate consequence of Lemma 5.51, Lemma 5.52 and Lemma 5.53.

It is left to show the continuity of $\mathcal{F} \upharpoonright \text{ETBES}$ and $\mathcal{F} \upharpoonright \text{ETPES}$, i.e. $\bigcup_i \mathcal{F}(E_i) = \mathcal{F}(\bigcup_i E_i)$ for every $\omega$-chain $(E_i)_{i \in \mathbb{N}}$ with respect to $\subseteq$ and $\bigcup_i \mathcal{F}(E_i) = \mathcal{F}(\bigcup_i E_i)$ for every $\omega$-chain $(E_i)_{i \in \mathbb{N}}$ with respect to $\subseteq$. The coincidence of the sets of events and the labeling function is easily seen. In the following, we only consider the termination set, since the conflict and the causality relation follow analogously. We have

$$T_{\bigcup_i \mathcal{F}(E_i)} = \{ \hat{X} \mid \exists j : \forall X \in T_j : \hat{X} \cap E_j \cap X \neq \emptyset \}$$  \hspace{1cm} (5.5)

$$T_{\mathcal{F}(\bigcup_i E_i)} = \{ X : \forall X \in \mathcal{P}(\bigcup_i E_i) : (\forall k : X \cap E_k \in E_k) \Rightarrow X \cap X \neq \emptyset \}$$  \hspace{1cm} (5.6)

$$T_{\bigcup_i \mathcal{F}(E_i)} = \{ X : \forall \hat{X} \in \bigcup_i \mathcal{E}_i : (\exists j : (\hat{X} \cap \mathcal{E}_j) \in \mathcal{T}_j) \Rightarrow X \cap \hat{X} \neq \emptyset \}$$  \hspace{1cm} (5.7)

$$T_{\mathcal{F}(\bigcup_i E_i)} = \{ X : \forall \hat{X} \in \mathcal{P}(\bigcup_i \mathcal{E}_i) : (\exists j : (\hat{X} \cap \mathcal{E}_j) \in \mathcal{T}_j) \Rightarrow X \cap \hat{X} \neq \emptyset \}$$  \hspace{1cm} (5.8)

$T_{\bigcup_i \mathcal{F}(E_i)} \subseteq T_{\mathcal{F}(\bigcup_i E_i)}$: Suppose $\hat{X} \in T_{\bigcup_i \mathcal{F}(E_i)}$. Let $X \in \mathcal{P}(\bigcup_i E_i)$ such that $\forall k : X \cap E_k \in E_k$. Then $\hat{X} \cap X \supseteq \hat{X} \cap E_j \cap (E_j \cap X) \neq \emptyset$. \hspace{1cm} (5.5)

$T_{\bigcup_i \mathcal{F}(E_i)} \supseteq T_{\mathcal{F}(\bigcup_i E_i)}$: Suppose $\hat{X} \in T_{\mathcal{F}(\bigcup_i E_i)}$. Since $\mathcal{F}(\bigcup_i E_i)$ is an eTpes, there is $\hat{X}' \in T_{\mathcal{F}(\bigcup_i E_i)}$ such that $\hat{X}' \subseteq \hat{X} \land |\hat{X}'| < |\mathbb{N}|$. Hence, there is $j$ such that $\hat{X}' \subseteq E_j$.

Let $X \in T_j$, then by Theorem 5.19 there is $X' \in \bigcup_i E_i$ (i.e. $\forall k : X' \cap E_k \in E_k$) such that $X' \cap E_j = X$. Thus $\hat{X} \cap E_j \cap X \supseteq \hat{X}' \cap E_j \cap X = \hat{X}' \cap E_j \cap X' \supseteq \hat{X}' \cap X' \neq \emptyset$. \hspace{1cm} (5.6)

$T_{\bigcup_i \mathcal{F}(E_i)} \subseteq T_{\mathcal{F}(\bigcup_i E_i)}$: Suppose $X \in T_{\bigcup_i \mathcal{F}(E_i)}$. Let $\hat{X} \in \mathcal{P}(\bigcup_i \mathcal{E}_i)$ such that $\exists j : (\hat{X} \cap \mathcal{E}_j) \in \mathcal{T}_j$. Then $X \cap \hat{X} \supseteq X \cap \mathcal{E}_j \cap (\mathcal{E}_j \cap \hat{X}) \neq \emptyset$. \hspace{1cm} (5.7)

$T_{\bigcup_i \mathcal{F}(E_i)} \supseteq T_{\mathcal{F}(\bigcup_i E_i)}$: Suppose $X \in T_{\mathcal{F}(\bigcup_i E_i)}$. Let $\hat{X} \in \mathcal{T}_k$, then $\hat{X} = \hat{X} \cap \mathcal{E}_k$. Hence, $X \cap \mathcal{E}_k \cap \hat{X} = X \cap \hat{X} \neq \emptyset$. \hfill \Box
5.6.4 Proof of Theorem 5.40.

In order to prove that ETBES and ETPES have the same expressive power with respect to event traces, we define remainders with respect to $\mathcal{P}(\mathcal{P}(E))$, as follows.

**Definition 5.54** Suppose $E' \subseteq E$. Define $\text{Rem}_{E,E'} : \mathcal{P}(\mathcal{P}(E)) \times E \rightarrow \mathcal{P}(\mathcal{P}(E'))$ by

$$\text{Rem}_{E,E'}(M, e) = \{X' \subseteq E' \mid \exists X \in M : X' = X \cap E' \land e \notin X\}$$

and $\widehat{\text{Rem}}_{E,E'} : \mathcal{P}(\mathcal{P}(E)) \times E \rightarrow \mathcal{P}(\mathcal{P}(E'))$ by

$$\widehat{\text{Rem}}_{E,E'}(M, e) = \{\hat{X}' \subseteq E' \mid \exists \hat{X} \in \hat{M} : \hat{X}' = \hat{X} \setminus \{e\}\}.$$  

The remainders defined above coincide with respect to $F_E$:

**Lemma 5.55** Suppose $e \notin E'$ and $E' \subseteq E$. Then

(i) $F_E'(\text{Rem}_{E,E'}(M, e)) = \widehat{\text{Rem}}_{E,E'}(F_E(M), e)$

(ii) $F_E'(\widehat{\text{Rem}}_{E,E'}(\hat{M}, e)) = \text{Rem}_{E,E'}(F_E(\hat{M}), e)$.

**Proof:**

(i): We have $\hat{X}' \in F_E'(\text{Rem}_{E,E'}(M, e))$ if and only if

$$\hat{X}' \subseteq E' \land \forall \hat{X}' \in \mathcal{P}(E') : (\exists X \in M : X' = X \cap E' \land e \notin X) \Rightarrow X' \cap \hat{X}' \neq \emptyset \tag{5.9}$$

and $\hat{X}' \in \widehat{\text{Rem}}_{E,E'}(F_E(M), e)$ if and only if

$$\hat{X}' \subseteq E' \land \exists \hat{X} \in \mathcal{P}(E) : (\forall \hat{X} \in M : \hat{X} \cap \hat{X}' \neq \emptyset) \land \hat{X}' = \hat{X} \setminus \{e\} \tag{5.10}$$

$\subseteq$: Suppose $\hat{X}' \in F_E'(\text{Rem}_{E,E'}(M, e))$. Define $\hat{X} = \hat{X}' \cup \{e\}$. Let $\hat{X} \in M$. If $e \in \hat{X}$ then $\hat{X} \cap \hat{X}' \neq \emptyset$. Therefore, suppose $e \notin \hat{X}$. From (5.9) we get $\emptyset \neq (\hat{X} \cap E') \cap \hat{X}' \subseteq E'$ and $\hat{X} \cap \hat{X}' \subseteq \hat{X} \cap \hat{X}'$, which establishes (5.10).

$\supseteq$: Suppose $\hat{X}' \in \widehat{\text{Rem}}_{E,E'}(F_E(M), e)$. Then by (5.10) there is $\hat{X} \in \mathcal{P}(E)$ such that $\forall \hat{X} \in M : \hat{X} \cap \hat{X}' \neq \emptyset$ and $\hat{X}' = \hat{X} \setminus \{e\}$. Let $X' \in \mathcal{P}(E')$ such that $\exists X \in M : X' = X \cap E' \land e \notin X$. Hence, $X' \cap \hat{X}' = (X \cap E') \cap \hat{X}' \subseteq E'$ and $X \cap \hat{X}' = X \cap (\hat{X} \setminus \{e\})$ which is non-empty by (5.10). Hence, (5.9) is concluded.

(ii): We have $X' \in F_E'(\widehat{\text{Rem}}_{E,E'}(\hat{M}, e))$ if and only if

$$X' \subseteq E' \land \forall \hat{X}' \in \mathcal{P}(E') : (\exists \hat{X} \in \hat{M} : \hat{X}' = \hat{X} \setminus \{e\}) \Rightarrow X' \cap \hat{X}' \neq \emptyset \tag{5.11}$$

and $X' \in \text{Rem}_{E,E'}(F_E(\hat{M}), e)$ if and only if

$$X' \subseteq E' \land \exists X \in \mathcal{P}(E) : (\forall \hat{X} \in \hat{M} : X \cap \hat{X} \neq \emptyset) \land X' = X \cap E' \land e \notin X \tag{5.12}$$
5.6. PROOFS

\(\subseteq\): Suppose \(X' \in F_E(\widehat{\text{Rem}}_{E,E'}(\hat{M}, e))\) Define \(X = X' \cup (E \setminus (E' \cup \{e\}))\). Then \(X \cap E' = X'\) and \(e \notin X\), since \(X' \subseteq E'\) and \(e \notin E'\). Let \(\hat{X} \in \hat{M}\).

If \(\hat{X} \subseteq E' \cup \{e\}\) then \(X' \cap (\hat{X} \setminus \{e\}) \neq \emptyset\) by (5.11). Furthermore, \(X \cap \hat{X} \supseteq X' \cap \hat{X} = X' \cap (\hat{X} \setminus \{e\})\), since \(X' \subseteq E'\) and \(e \notin E'\).

If \(\hat{X} \not\subseteq E' \cup \{e\}\) then there exists \(\hat{e} \in \hat{X}\) such that \(\hat{e} \notin E' \cup \{e\}\). Hence \(\hat{e} \in \hat{X} \cap (E \setminus (E' \cup \{e\})) \subseteq \hat{X} \cap X\).

Thus (5.12) is established.

\(\supseteq\): Suppose \(X' \in \text{Rem}_{E,E'}(F_E(\hat{M}), e)\). Then by (5.12) there is \(X \in \mathcal{P}(E)\) such that \(\forall \hat{X} \in \hat{M}: X \cap \hat{X} \neq \emptyset\) and \(X' = X \cap E'\) and \(e \notin X\).

Let \(\hat{X}' \in \mathcal{P}(E')\) such that \(\exists \hat{X} \in \hat{M}: \hat{X}' = \hat{X} \setminus \{e\}\). Then \(X' \cap \hat{X}' = X \cap E' \cap \hat{X}' \subseteq E'\), \(X \cap \hat{X}' = X \cap (\hat{X} \setminus \{e\})\) which is non-empty, since \(\forall \hat{X} \in \hat{M}: X \cap \hat{X} \neq \emptyset\).

Hence, (5.11) is concluded. \(\square\)

Before we continue, we give a modified version of the remainder on ETBES. There we guarantee in the case of termination that \(T\) is upper closed.

**Definition 5.56** Let \(\mathcal{E} \in \text{ETBES}\) and \(e \in \text{init}(\mathcal{E})\). Then \(\mathcal{E}_{[e]}\) is given by \((E', \succ', \mapsto', T', l')\), where \(E', \succ', \mapsto'\) and \(l'\) are defined as in Definition 5.9 and

\[
T' = \begin{cases} 
\{X \cap E' \mid X \in T \land e \notin X\} & \text{if } \neg \Upsilon(T, e) \\
\mathcal{P}(E') & \text{otherwise}
\end{cases}
\]

**Lemma 5.57** Let \(\mathcal{E} \in \text{ETBES}\). Then the event traces obtained by Definition 5.9 and by Definition 5.56 are identical. Moreover the corresponding labels of the event executions coincide.

**Proof:** This follows from the fact that \(T\) does not influence the event traces. It only determines the fact when an event becomes a termination event. After termination no further termination may happen by both remainders. This holds, since \(\emptyset\) is in the termination set. \(\square\)

The remainders on ETBES and ETPES coincide:

**Proposition 5.58** Let \(\mathcal{E} \in \text{ETBES}\) and \(\mathcal{E} \in \text{ETPES}\). Then \(\mathcal{F}(\mathcal{E}_{[e]}) \simeq \mathcal{F}(\mathcal{E})_{[\hat{e}]}\) and \(\mathcal{F}(\mathcal{E}_{\hat{e}}) \simeq \mathcal{F}(\mathcal{E})_{[e]}\).

**Proof:** Let \(\mathcal{E}' = \mathcal{E}_{[e]}\) and \(\mathcal{E}' = \mathcal{E}_{\hat{e}}\). Then it is easy to check that

\[
\succ' = \{(Z', e') \mid e' \in E' \land Z' \in \text{Rem}_{E,E'}(\succ, e')\}
\]
\[
\mapsto' = \{(X', e') \mid e' \in E' \land X' \in \text{Rem}_{E,E'}(\mapsto, e')\}
\]
\[
T' = \begin{cases} 
\text{Rem}_{E,E'}(T, e) & \text{if } \neg \Upsilon(T, e) \\
\mathcal{P}(E') & \text{otherwise}
\end{cases}
\]

and

\[
\hat{\succ}' = \{(Z', e') \mid e' \in \hat{E}' \land Z' \in \text{Rem}_{E,E'}(\hat{\succ}, e')\}
\]
\[
\hat{\mapsto}' = \{(X', e') \mid e' \in \hat{E}' \land X' \in \text{Rem}_{E,E'}(\hat{\mapsto}, e')\}
\]
\[
\hat{T}' = \begin{cases} 
\text{Rem}_{E,E'}(\hat{T}, e) & \text{if } \neg \Upsilon(\hat{T}, e) \\
\emptyset & \text{otherwise}
\end{cases}
\]
Furthermore, $e \in \text{init}(E) \iff \neg (\exists X : X \mapsto e) \iff \emptyset \mapsto_{E} e \iff e \in \text{init}(\mathcal{F}(E))$ and $e \in \text{init}(E) \iff \emptyset \mapsto e \iff (\exists X : X \mapsto_{E} e) \iff e \in \text{init}(\mathcal{F}(E))$. Hence, $\mathcal{F}(E')$ is defined if and only if $\mathcal{F}(E)_{[e]}$ is defined, and $\mathcal{F}(E'')$ is defined if and only if $\mathcal{F}(E''')_{[e]}$ is defined.

We have $e' \in E_{\mathcal{F}(E')}$ if and only if $e' \in E' \iff (\exists Z : Z \succ e' \land e \notin Z) \iff (\{e\} \succeq_{\mathcal{F}(E)} e') \iff e' \in E_{\mathcal{F}(E')}$ and $e' \in E_{\mathcal{F}(E')}$ if and only if $e' \in E' \iff (\exists Z : Z \succ e' \land e \notin Z) \iff (\{e\} \succeq e') \iff e' \in E_{\mathcal{F}(E'')}$.

The rest follows from Lemma 5.55 and the fact that $F_{E'}(\emptyset) = \mathcal{P}(E')$ and $F_{E'}(\mathcal{P}(E')) = \emptyset$. □

Theorem 5.40 is an immediate consequence of Lemma 5.57 and Proposition 5.58.
Chapter 6

End-Based View in ETBES

As in Chapter 4, we consider the view that a choice is determined by the ending of actions (end-based view), which is contrary to the usual approach, where the start of an action triggers the choice. A motivation for an end-based approach is given in Section 1.3.

In this chapter, we apply the end-based approach to ETBES and not to CBES, as it is done in Chapter 4. More precisely, we define an end-based refinement operator on ETBES such that the refinement terminates by its ‘final’ executed event (action) and not with an additional event, as it is done in Chapter 4. The end-based approach is adjusted to ETBES, because the intuitive equivalences (ICT- and FUI) fail to be the coarsest for the end-based refinement operator in the CBES-setting (Subsection 4.2.6).

We adjust the definition of the ICT- and the FUI-equivalence (Section 4.2) to the ETBES setting. The UI-equivalence can also be adjusted to the ETBES setting in a straightforward way, which is omitted here. We show that the ICT- (and the FUI-) equivalence is indeed the coarsest congruence for the end-based refinement operator with respect to trace (respectively bisimulation) equivalence in the ETBES setting. This circumstance underpins the fact that extended termination bundle event structures represent a reasonable extension of the standard event structures. Furthermore, we show that the hierarchy of the equivalences considered in the ETBES setting is the same as in the CBES setting.

6.1 An End-Based Refinement Operator on ETBES

The differences between refinement operators in start-based and in end-based settings is illustrated in Section 4.1. There, we also argue that an event structure suitable for an end-based refinement operator has to allow the modeling of disruption. This is true for extended termination bundle event structures (eTbes), which are introduced in Subsection 5.4.2. Hence, they represent a suitable model for introducing an end-based refinement operator. This operator is given in the following definition.

Let \( \tau \), Obs and Var be defined as in Section 3.2 and let \( \text{Act} \) be defined as in Section 5.2.

**Definition 6.1** Let \( A \subseteq \text{Obs} \). Then define \( \text{Ref}^{eT}_A : \text{ETBES} \times (A \rightarrow \text{ETBES}) \rightarrow \text{ETBES} \) by \( \text{Ref}^{eT}_A(\mathcal{E}, \theta) = (\tilde{E}, \tilde{\succ}, \tilde{\Rightarrow}, \tilde{T}, \tilde{l}) \) where
\[ \tilde{E} = \{ (e, \hat{e}) \mid e \in E \land l(e) \in A \land \hat{e} \in E_{\theta(l(e))} \} \cup \{ (e, e) \in E \times E \mid l(e) \notin A \} \]

\[ \sim = \{ \tilde{Z}, (e, \hat{e}) \mid \exists Z : Z \succ e \land \exists f : Z \to P(U) : \]

\[ (\forall e' \in Z : (l(e') \notin A \land f(e') = \{e'\}) \lor (l(e') \in A \land e' \neq e \land f(e') \in T_{\theta(l(e'))}) \land \]

\[ (l(e') \in A \land e' = e \land \exists \hat{X} \in T_{\theta(l(e'))}, \tilde{Z} : f(e') = \hat{Z} \cup \hat{X} \land \hat{Z} \succ_{T_{\theta(l(e'))}} e) \} \land \]

\[ \tilde{T} = \{ \tilde{X} \mid \exists e \in \tilde{T} \land \exists f : X \to P(U) : (\forall e' \in X : (l(e') \notin A \land f(e') = \{e'\}) \lor \]

\[ (l(e') \in A \land f(e') \in T_{\theta(l(e'))}) \land \tilde{X} = \{ (e', \hat{e'}) \in \tilde{E} \mid e' \in X \land \hat{e'} \in f(e') \} \} \]

\[ \tilde{l}(e, \hat{e}) = \begin{cases} l(e) & \text{if } l(e) \notin A \\ l_{\theta(l(e))}(\hat{e}) & \text{if } l(e) \in A \end{cases} \]

We give some comments on the definition. A witness-bundle derived from the witness-bundle \( Z \) of \( e \) (i.e. \( Z \succ e \)) of event \( (e, \hat{e}) \) has to contain all events of a termination-bundle of the refinement of an event \( e' \) (different to \( e \)) that is in \( Z \). This is done in order to guarantee that bundles derived from \( Z \) can not be used as a witness of \( e \) when \( e' \) terminates. If \( e' \) is equal to \( e \), then the witness-bundles (instead of the termination bundles) of the refinement are considered. This guarantees that the events of the refinement become disabled, as specified by the refinement. A termination-bundle of the refinement of \( e' \) is used in the causality relation whenever \( e' \) appears in the causality bundle \( X \) of the unrefined event structures, since the refinement has to terminate before the constraint specified by \( X \) is fulfilled. This is also the case in the definition of the termination set.

**Lemma 6.2** The refinement operator \( \text{Ref}^{\text{et}} \) is well defined, i.e. it really yields elements of ETBES.

**Proof:** The proof is given in Subsection 6.4.1. \( \square \)

An example that illustrates how the refinement operator \( \text{Ref}^{\text{et}} \) behaves is given in Figure 6.1. For a better understanding, we augment the examples by process term descriptions of the systems (see Section 5.2). Furthermore, \((a \to E_{12})\) denotes the function from \( \{a\} \) to ETBES that maps \( a \) to \( E_{12} \). Moreover, if an event \( e' \) is a necessary causality of \( e \), i.e. \( e \) can only be executed after the execution of \( e' \), we sometimes omit \( e \) in the witness bundles of \( e' \), since it has no consequence for the behavior. For example, in \( \text{Ref}^{\text{et}}_{\{a\}}(E_{1}^{+a}, (a \to E_{12})) \) of Figure 6.1, the witness bundles to events labeled with \( a_{1} \) have to contain both events labeled with \( a_{2} \).

As in Remark 4.4, the refinement operator \( \text{Ref}^{\text{et}} \) allows the modeling of the disrupt operator \( \triangleright_{D} \) of Definition 5.20, i.e. \( E_{1} \triangleright_{D} E_{2} \) and \( \text{Ref}^{\text{et}}_{\{a\}}(E_{1} + \lbrack a \rbrack, (a \to E_{2})) \) have the same behavior if label \( a \) does not appear in \( E_{1} \).

### 6.2 Equivalences for ETBES

As in Section 4.2, we examine congruence equivalences for \( \text{Ref}^{\text{et}} \). We are particularly interested in the coarsest congruence with respect to trace / strong bisimulation equivalences, where
6.2. EQUIVALENCES FOR ETBES

\[ \mathcal{E}^+ a \]

\[ \mathcal{E}_{12} \]

\[ \text{Ref}_{[a]}^T(\mathcal{E}^+ a, (a \rightarrow \mathcal{E}_{12})) \]

\[ a + a \]

\[ a_1; a_2 \]

Figure 6.1: End-Based Refinement in ETBES

trace and strong bisimulation equivalences is obtained, as usual, from the derived transition system introduced in Definition 5.11. Formally:

**Definition 6.3 (Trace Equivalence)** Two \( \mathcal{E}, \mathcal{E}' \in \text{ETBES} \) are trace equivalent, denoted by \( \mathcal{E} \sim_{\text{t}} \mathcal{E}' \), if and only if the transition systems \((\text{ETBES}, \text{Act}, \rightarrow, \mathcal{E})\) and \((\text{ETBES}, \text{Act}, \rightarrow, \mathcal{E}')\), where \( \rightarrow \) is defined in Definition 5.11, are trace equivalent (Definition 2.4).

**Definition 6.4 (Strong Bisimulation Equivalence)** Two \( \mathcal{E}, \mathcal{E}' \in \text{ETBES} \) are strong bisimilar (or strong bisimulation equivalent), denoted by \( \mathcal{E} \sim_{\text{b}} \mathcal{E}' \), if and only if the transition systems \((\text{ETBES}, \text{Act}, \rightarrow, \mathcal{E})\) and \((\text{ETBES}, \text{Act}, \rightarrow, \mathcal{E}')\) are bisimilar (Definition 2.5).

6.2.1 ICT-Equivalence on ETBES

ICT-equivalence from Subsection 4.2.2 is adapted to ETBES as follows.

**Definition 6.5 (ICT-equivalence)** Let \( \mathcal{E} \in \text{ETBES} \). Then the initial event traces of \( \mathcal{E} \) are defined by \( T_{\text{ic}}(\mathcal{E}) = \{ ((e_i, \nu_i), \gamma_i)_{i \leq n} \mid n \in \mathbb{N} \land \exists \mathcal{E}_0, \ldots, \mathcal{E}_{n+1} : \mathcal{E}_0 = \mathcal{E} \land \forall i \leq n : \mathcal{E}_{[i]} = \mathcal{E}_{i+1} \land (\Upsilon(\mathcal{E}_{i}, e_i) \Leftrightarrow \nu_i = \sqrt{\gamma}) \land \gamma_i \in \mathcal{P}_{\text{fin}}(\text{init}\text{Obs}(\mathcal{E}_{i})) \} \).

Two \( \mathcal{E}, \mathcal{E}' \in \text{ETBES} \) are initial corresponding trace equivalent (ICT-equivalent), denoted by \( \mathcal{E} \sim_{\text{ICT}} \mathcal{E}' \), if

- for every \( ((e_i, \nu_i), \gamma_i)_{i \leq n} \in T_{\text{ic}}(\mathcal{E}) \) there is an injective, labeling preserving function \( f : (\bigcup_{i \leq n}(\gamma_i \cup \{e_i\})) \rightarrow \mathcal{E}' \) such that \( ((f(e_i), \nu_i), f(\gamma_i))_{i \leq n} \in T_{\text{ic}}(\mathcal{E}') \) and

- for every \( ((e'_i, \nu_i), \gamma'_i)_{i \leq n} \in T_{\text{ic}}(\mathcal{E}') \) there is an injective, labeling preserving function \( f' : (\bigcup_{i \leq n}(\gamma'_i \cup \{e'_i\})) \rightarrow \mathcal{E} \) such that \( ((f'(e'_i), \nu_i), f'(\gamma'_i))_{i \leq n} \in T_{\text{ic}}(\mathcal{E}) \)

The eTbes from Figure 6.2 are not ICT equivalences, whereas the eTbes from Figure 6.3 are ICT equivalent.

**Proposition 6.6** Two ICT-equivalent eTbes are also trace equivalent, i.e. \( \sim_{\text{ICT}} \subset \sim_{\text{t}} \).

**Proof:** It follows from the fact that every trace is also an initial trace, where the second component is always empty. \( \Box \)
Theorem 6.7 ICT-equivalence is a congruence for the refinement operator $\text{Ref}^{eT}$, i.e. $\mathcal{E} \sim_{\text{ICT}} \mathcal{E}' \land \forall a \in A : \theta(a) \sim_{\text{ICT}} \theta'(a)$ implies that $\text{Ref}^{eT}_A (\mathcal{E}, \theta) \sim_{\text{ICT}} \text{Ref}^{eT}_A (\mathcal{E}', \theta')$.

**Proof:** It works analogously to the proof of Theorem 4.10. \qed

Contrary to the CBES setting, ICT-equivalence is the coarsest congruence for $\text{Ref}^{eT}$ with respect to trace equivalence.

Theorem 6.8 ICT-equivalence is the coarsest congruence for $\text{Ref}^{eT}$ with respect to trace equivalence. Moreover, if $\forall A, \theta : \text{Ref}^{eT}_A (\mathcal{E}, \theta) \sim_{t} \text{Ref}^{eT}_A (\mathcal{E}', \theta)$ then $\mathcal{E} \sim_{\text{ICT}} \mathcal{E}'$.

**Proof:** The proof is given in Subsection 6.4.2. \qed

### 6.2.2 FUI-Equivalence on ETBES

FUI-equivalence from Subsection 4.2.4 is adapted to ETBES as follows.

**Definition 6.9 (FUI-Bisimulation)** A finite unique initial bisimulation (FUI-bisimulation) $\mathcal{R}$ is a subset of $\text{ETBES} \times \text{ETBES} \times \{\mathcal{U} \rightarrow \mathcal{U}\}$ such that whenever $(\mathcal{E}_1, \mathcal{E}_2, f) \in \mathcal{R}$, then

- $\text{dom}(f) = \text{init}_{\text{Obs}}(\mathcal{E}_1)$,
- $f$ is a labeling preserving isomorphism between $\text{init}_{\text{Obs}}(\mathcal{E}_1)$ and $\text{init}_{\text{Obs}}(\mathcal{E}_2)$
- $e_1 \in \text{init}(\mathcal{E}_1) \land I \in \mathcal{P}_{\text{fin}}(\text{init}_{\text{Obs}}(\mathcal{E}_1))$ implies that there exist $e_2$ and $f'$ such that $l_1(e_1) = l_2(e_2)$ and $\Upsilon(T_1, e_1) \leftrightarrow \Upsilon(T_2, e_2)$ and $l_1(e_1) \in \text{Obs} \Rightarrow e_2 = f(e_1)$ and $(\mathcal{E}_1[e_1], \mathcal{E}_2[e_2], f') \in \mathcal{R}$ and $f \upharpoonright (I \cap \text{init}_{\text{Obs}}(\mathcal{E}_1[e_1])) = f' \upharpoonright I$ and $f^{-1} \upharpoonright (f(I) \cap \text{init}_{\text{Obs}}(\mathcal{E}_2[e_2])) = f'^{-1} \upharpoonright f(I)$
6.2. EQUIVALENCES FOR ETBES

We say that $E_1, E_2$ are FUI-bisimilar (or FUI-equivalent), denoted by $E_1 \sim_{FUI} E_2$, if and only if there is a FUI-bisimulation $R$ and an $f : U \rightarrow U$ such that $(E_1, E_2, f) \in R$.

The eTbes from Figure 6.3 are not FUI-equivalent, whereas the eTbes from Figure 6.4 are FUI-equivalent. Moreover, the eTbes from Figure 6.5 are also FUI-equivalent.

FUI-equivalence yields a congruence.

**Theorem 6.10** FUI-equivalence is a congruence for $\text{Ref}^e_T$, i.e. $E \sim_{FUI} E' \land \forall a \in A : \theta(a) \sim_{FUI} \theta'(a)$ implies that $\text{Ref}_A^e(\mathcal{E}, \theta) \sim_{FUI} \text{Ref}_A^e(\mathcal{E}', \theta')$.

**Proof:** It works analogously to the proof of Theorem 4.18.

Contrary to the CBES setting, FUI-equivalence is the coarsest congruence for $\text{Ref}^e_T$ with respect to bisimulation equivalence.


\begin{figure}
\centering
\begin{tikzpicture}
  \node (A) at (0,0) {$\sim_{\text{FUI}}$};
  \node (B) at (1,0) {$\sim_{\text{ICT}}$};
  \node (C) at (2,0) {$\sim_b$};
  \node (D) at (3,0) {$\sim_t$};
  \draw (A) -- (B);
  \draw (B) -- (C);
  \draw (C) -- (D);
\end{tikzpicture}
\caption{Relations Between the Equivalences}
\end{figure}

**Theorem 6.11** FUI-equivalence is the coarsest congruence for \( \text{Ref}^e_T \) with respect to bisimulation equivalence. Moreover, if \( \forall A, \theta : \text{Ref}^e_T (\mathcal{E}, \theta) \sim_b \text{Ref}^e_T (\mathcal{E'}, \theta) \) then \( \mathcal{E} \sim_{\text{FUI}} \mathcal{E'} \).

**Proof:** The proof is given in Subsection 6.4.2.

\[ \square \]

### 6.2.3 Comparison of Equivalences

**Theorem 6.12** All valid relations between the equivalences \( \sim_t, \sim_{\text{ICT}}, \sim_b, \sim_{\text{FUI}} \) are presented in Figure 6.6: If two equivalences are connected via a line, then the lower one identifies more elements than the upper one.

**Proof:** \( \sim_{\text{FUI}} \subseteq \sim_b \subseteq \sim_t \) and \( \sim_{\text{ICT}} \subseteq \sim_t \) is obvious.

Suppose \( \mathcal{E} \sim_{\text{FUI}} \mathcal{E'} \), then by Theorem 6.10 we have \( \forall A, \theta : \text{Ref}^e_T (\mathcal{E}, \theta) \sim_{\text{FUI}} \text{Ref}^e_T (\mathcal{E'}, \theta) \).

Since \( \sim_{\text{FUI}} \subseteq \sim_b \subseteq \sim_t \), we obtain \( \forall A, \theta : \text{Ref}^e_T (\mathcal{E}, \theta) \sim_t \text{Ref}^e_T (\mathcal{E'}, \theta) \). Thus by Theorem 6.8 it follows that \( \mathcal{E} \sim_{\text{ICT}} \mathcal{E'} \).

The strictness of \( \sim_{\text{FUI}} \subset \sim_{\text{ICT}} \) follows from the event structure depicted in Figure 6.5. The strictness of \( \sim_b \subset \sim_t \) is well known. And the other strictness follows from the fact that the event structures corresponding to \( a \) and \( a + a \) are bisimilar but not ICT-equivalent.

\[ \square \]

### 6.3 Discussion

It is now straightforward to give a denotational semantics to process algebras that contain end-based choice operators together with action refinement operators as long as no parallel operator with action synchronization is contained in the process algebra.

For process algebras that also contain a parallel operator with action synchronization, it is reasonable that some start-based choices are modeled. For example, we expect that the expression \( (a + a)\|\{a\}a \) may only start one \( a \)-action, since the process on the right hand side can only start one \( a \)-action. Hence, a start-based choice is modeled for the process on the left hand side. Such a circumstance arises, for example, if the left hand side demands processor resources and the right hand side specifies the administration of the processor resource, where only one \( a \) can be executed.

Therefore, it is more reasonable to consider process algebras that contain both choice operators, i.e. end-based and start-based choice operators. Such a process algebra is intensively examined in the following chapter.
6.4 Proofs

6.4.1 Proof of Lemma 6.2

The constraints which are different to the approximation closedness are easy to check. Let 
\[ \tilde{E} = \text{Ref}_A^T (\mathcal{E}, \theta) \text{ and } E_e = \begin{cases} E_{\theta(l(e))} & \text{if } l(e) \in A \\ \{e\} & \text{otherwise} \end{cases}. \]

\[ \tilde{\rightarrow} : \text{Suppose } (e', \tilde{e}') \in \tilde{E}. \text{ Define } M = \{Z | Z \succ e'\}. \text{ Furthermore, let } \]
\[ M_e = \begin{cases} \{\{e\}\} & \text{if } l(e) \notin A \\ \{T_{\theta(l(e))}\} & \text{if } l(e) \in A \land e' \neq e \\ \{\hat{Z} \cup \hat{X} | \hat{Z} \succ_{\theta(l(e))} \hat{e} \land \hat{X} \in T_{\theta(l(e)))}\} & \text{if } l(e) \in A \land e' = e \end{cases}. \]

From Corollary 2.19 we obtain that \( M_e \) is approximation closed with respect to \( E_e \).

Furthermore, \( \tilde{M} = \{(e, \tilde{e}) | e \in X \land \tilde{e} \in X_e\} | X \in M \land X_e \in M_e\} \) is approximation closed with respect to \( \tilde{E} \) by Corollary 2.21. Hence, \( \tilde{\rightarrow} \) is approximation closed with respect to \( \tilde{E} \), since \( \{\hat{Z} | \hat{Z} \tilde{\rightarrow} (e', \tilde{e}')\} = \tilde{M} \).

\[ \tilde{\leftarrow} : \text{Suppose } (e', \tilde{e}') \in \tilde{E}. \text{ Define } M = \{Z | Z \leftarrow e'\} \text{ and } \]
\[ M_1 = \begin{cases} \{\{e'\} \times \hat{X} | \hat{X} \tilde{\leftarrow}_{\theta(l(e'))} \tilde{e}'\} & \text{if } l(e') \in A \\ \emptyset & \text{otherwise} \end{cases}. \]

Then \( M_1 \) is approximation closed with respect to \( \tilde{E} \). Furthermore, let \( \]
\[ M_e = \begin{cases} \{\{e\}\} & \text{if } l(e) \notin A \\ \{T_{\theta(l(e))}\} & \text{if } l(e) \in A \end{cases}. \]

Obviously, \( M_e \) is approximation closed with respect to \( E_e \).

From Corollary 2.21 we obtain that \( M_2 = \emptyset \) if \( l(e') \in A \land e' \notin \text{init}(\theta(l(e'))) \) otherwise is approximation closed with respect to \( \tilde{E} \). And so the approximation closedness of \( \tilde{\leftarrow} \) follows from Proposition 2.15, since \( \{\hat{X} | \hat{X} \tilde{\leftarrow} (e', \tilde{e}')\} = M_1 \cup M_2 \).

\[ \tilde{T} : \text{Let } \]
\[ M_e = \begin{cases} \{\{e\}\} & l(e) \notin A \\ \{T_{\theta(l(e))}\} & l(e) \in A \end{cases}. \]

Obviously, \( M_e \) is approximation closed with respect to \( E_e \).

Furthermore, \( \tilde{M} = \{(e, \tilde{e}) | e \in X \land \tilde{e} \in X_e\} | X \in M \land X_e \in M_e\} \) is approximation closed with respect to \( \tilde{E} \) by Corollary 2.21. Hence, \( \tilde{T} \) is approximation closed with respect to \( \tilde{E} \), since \( \tilde{T} = \tilde{M} \).

Thus Lemma 6.2 is established.

6.4.2 Proofs of the Coarsest Congruence Results

We introduce an event-based refinement. This refinement differs from \( \text{Ref}_A^T \) by assigning event structures to each event and not only to action names.

\textbf{Definition 6.13} \( \text{Ref}_A^{eT} : \text{ETBES} \times (\mathcal{U} \rightarrow \text{ETBES}) \rightarrow \text{ETBES} \) with
\[ \text{Ref}^{eT}_A(E, \vartheta) = (\hat{E}, \hat{\vartheta}, \hat{\rightarrow}, \hat{T}, \hat{l}) \text{ where} \]
\[ \hat{E} = \{(e, \hat{e}) \mid e \in E \land l(e) \in A \land \hat{e} \in E_{\vartheta(e)}\} \cup \{(e, e) \in E \times E \mid l(e) \notin A\} \]
\[ \hat{\vartheta} = \{(\hat{Z}, (e, \hat{e})) \mid \exists Z : Z \ni e \land \exists f : Z \rightarrow \mathcal{P}(U) : (\forall e' \in Z : (l(e') \notin A \land f(e') = \{e'\}) \lor (l(e') \in A \land e' \neq e \land f(e') \in T_{\vartheta(e')}\)}\]
\[ \hat{l}(e, \hat{e}) = \begin{cases} l(e) & \text{if } l(e) \notin A \\ l_{\vartheta(e)}(\hat{e}) & \text{if } l(e) \in A \end{cases} \]

The advantage of \( \text{Ref}^{eT}_A \) is that the event execution of \( \text{Ref}^{eT}_A(E, \vartheta) \) can be reduced to the event execution of \( E \) and \( \vartheta \), as it is shown in the following lemma.

**Lemma 6.14** Suppose \( E \in \text{ETBES}, \vartheta : U \rightarrow \text{ETBES} \). Then

\[ \text{Ref}^{eT}_A(E, \vartheta)_{[e, \hat{e}]} \approx \begin{cases} \text{Ref}^{eT}_A(E_{[e]}, \vartheta) & \text{if } l(e) \notin A \land e = \hat{e} \\ \text{Ref}^{eT}_A(E_{[e]}, \vartheta[e \rightarrow \vartheta(e)[e]]) & \text{if } l(e) \in A \land \vartheta(T_{\vartheta(e)}, \hat{e}) \\ \text{Ref}^{eT}_A(E, \vartheta[e \rightarrow \vartheta(e)[e]]) & \text{if } l(e) \in A \land \neg \vartheta(T_{\vartheta(e)}, \hat{e}) \end{cases} \]

Furthermore, \( \text{Ref}^{eT}_A(E_{[e]}, \vartheta[e \rightarrow E']) \approx \text{Ref}^{eT}_A(E_{[e]}, \vartheta) \) holds for any \( E' \in \text{ETBES} \).

Moreover,

\[ \vartheta(T_{\text{Ref}^{eT}_A(E, \vartheta)}, (e, \hat{e})) \Leftrightarrow \begin{cases} \vartheta(T, e) & \text{if } l(e) \notin A \land e = \hat{e} \\ \vartheta(T, e) \land \vartheta(T_{\vartheta(e)}, \hat{e}) & \text{if } l(e) \in A \end{cases} \]

**Proof:** Straightforward and left to the reader. \( \square \)

**Proof of Theorem 6.8:** Suppose \( ((e_i, \nu_i), \gamma_i)_{i \leq n} \in T^{ic}(E) \). We define a refinement \( \theta' \) which is used to construct a corresponding trace.

Therefore, let \( \kappa : U \rightarrow \mathbb{N} \) be an isomorphism. Furthermore, define \( \hat{E} = \bigcup_{i \leq n} (\gamma_i \cup \{e_i\}) \) and \( \hat{E}_a = \{e \in \hat{E} \mid l(e) = a\} \). Additionally, define \( \delta : E \rightarrow \mathbb{N} \) by \( \delta(e) = 1 + |\{i \mid e \in \gamma_i\}| \).

Moreover, let \( A \subset Act \) be the set of all action-names occurring in \( E \) or in \( E' \), i.e. \( A = \{l(e) | e \in E\} \cup \{l'(e') | e' \in E'\} \). And let \( \mu : E \times \mathbb{N} \rightarrow \text{Obs} \setminus A \) be an injective function. Such a function exists.

Our idea of \( \theta' \) is that we replace any event \( e \) of \( \hat{E} \) by the sequential composition of actions \( \mu(e, 1), \ldots, \mu(e, \delta(e) + 1) \). Since \( \theta' \) only maps action-names instead of events, we take the sum
of all the corresponding events, i.e. \( \theta'(a) = \mathcal{E}'_a \), where

\[
\mathcal{E}'_a = \left\{ \{ x_2^{\kappa(e)} x_1^j \cdot \mid e \in \hat{E}_a \land 1 \leq j \leq \delta(e) \}, \right. \\
\left. \{ \{ x_2^k x_1 \cdot \mid k \neq \kappa(e) \} \cup \{ x_2^{\kappa(e)} x_1^j \cdot \mid e \in \hat{E}_a \land 1 \leq j \leq \delta(e) \}, \right. \\
\left. \{ \{ x_2^{\kappa(e)} x_1^j \cdot , x_2^{\kappa(e)} x_1^{j+1} \cdot \mid e \in \hat{E}_a \land 1 \leq j < \delta(e) \}, \right. \\
\left. \{ \{ x_2^{\kappa(e)} \delta(e) \cdot \mid e \in \hat{E}_a \}, \right. \\
\left. \{ \{ x_2^{\kappa(e)} x_1^j \cdot , \mu(e, j) \mid e \in \hat{E}_a \land 1 \leq j \leq \delta(e) \}. \right. \\
\]

The sequences \( x_2^{\kappa(e)} x_1^j \cdot \) are considered to be right bracketed and therefore to be elements of \( \mathcal{U} \).

Let \( m = n + \sum_{i=0}^n |\gamma_i| \) and define \( I_{-1} = \gamma_0 \) and \( s_{-1} = 0 \) and for \( i \in \{0, \ldots, m-1\} \)

- If \( I_{i-1} \neq \emptyset \), then \( s_i = s_{i-1}, I_i = I_{i-1} \setminus \{e\} \) and \( \tilde{e}_i = (e, x_2^{\kappa(e)} x_1^j \{j \in s_{i-1} \land e \in \gamma_i \} \cdot) \), where \( e \) is an element of \( I_{i-1} \).

- If \( I_{i-1} = \emptyset \), then \( s_i = s_{i-1} + 1, I_i = \gamma_{s_i} \) and

\[
\tilde{e}_i = \left\{ \begin{array}{ll}
(e_{s_{i-1}}, e_{s_{i-1}}) & \text{if } l(e_{s_{i-1}}) = \tau \\
(e_{s_{i-1}}, x_2^{\kappa(e_{s_{i-1}})} \cdot \delta(e_{s_{i-1}}) \cdot) & \text{otherwise}
\end{array} \right.
\]

and \( \tilde{e}_m = \left\{ \begin{array}{ll}
(e_n, e_n) & \text{if } l(e_n) = \tau \\
(e_n, x_2^{\kappa(e_{n-}}) \cdot \cdot) & \text{otherwise}
\end{array} \right. \)

It is easily seen that \( \{ \pi_1(\tilde{e}_i) \mid i \in \{0, \ldots, m\} \} = \hat{E} \). Define \( \tilde{E}_0 = \text{Ref}^T_A(\mathcal{E}, \theta') \) and \( \tilde{E}_{i+1} = \tilde{E}_{i[\tilde{e}_i]} \) for \( i \leq m \). The \( \tilde{E}_i \) are well defined which, can be seen by induction, as follows. Obviously for \( i = 0 \). Suppose \( \tilde{e}_i = (e, x_2^{\kappa(e)} x_1^j \cdot) \). By Lemma 6.14 we obtain that \( \tilde{E}_i = \text{Ref}^T_A(\mathcal{E}_{[e_0]}[e_{s_{i-1}} - 1], \theta'_i) \), where \( \theta'_i(e) = \theta'(l(e))_{x_2^{\kappa(e)} x_1^j \cdot \cdots x_2^{\kappa(e)} x_1^j \cdot} \) with \( q = \{ j \mid e = \pi_1(\tilde{e}_{j}) \land l(e) \neq \tau \land j < i \} \). It is easily seen that \( (e, x_2^{\kappa(e)} x_1^j \cdot) \), \( \ldots, (e, x_2^{\kappa(e)} x_1^j \cdot) \) appears in the sequence before \( \tilde{e}_i \). Thus \( x_2^{\kappa(e)} x_1^j \cdot \in \text{init}(\theta'_i(e)) \). Furthermore, \( e_{s_{i-1}} \in \text{init}(\mathcal{E}_{[e_0]}[e_{s_{i-1}} - 1]) \), since \( (e(e_j, \nu_j), \gamma_j)_{j \leq n} \in T^{ic}(\mathcal{E}) \). Hence, \( \tilde{e}_i \in \tilde{E}_i \). Furthermore, by Lemma 6.14 we obtain

\[
\Upsilon(\tilde{E}_i, \tilde{e}_i) \Leftrightarrow \exists j, \hat{e} : \tilde{e}_i = (e_j, \hat{e}) \land \nu_j = \sqrt{\land (l(e_j)) = \tau \lor \hat{e} = x_2^{\kappa(e_j)} \cdot \delta(e_j) \cdot) \quad (6.1)
\]

From the definition of \( \tilde{E}_i \) it follows that \((\alpha_i)_{(i \leq m)} \in T(\text{Ref}^T_A(\mathcal{E}, \theta')) \), where \( \alpha_i \) is defined by

\[
\alpha_i = \left\{ \begin{array}{ll}
l_{\text{Ref}^T_A(\mathcal{E}, \theta')} (\tilde{e}_i) & \text{if } \Upsilon(\tilde{E}_i, \tilde{e}_i) \\
l_{\text{Ref}^T_A(\mathcal{E}, \theta')} (\tilde{e}_i) \lor & \text{if } \Upsilon(\tilde{E}_i, \tilde{e}_i) \end{array} \right.
\]

Therefore, we get \((\alpha_i)_{(i \leq m)} \in T(\text{Ref}^T_A(\mathcal{E}', \theta')) \), since \( \text{Ref}^T_A(\mathcal{E}, \theta') \sim_t \text{Ref}^T_A(\mathcal{E}', \theta') \). Hence, there exists \((\tilde{e}_i')_{(i \leq m)} \) such that \( \tilde{E}'_0 = \text{Ref}^T_A(\mathcal{E}', \theta') \) and \( \tilde{E}'_{i+1} = \tilde{E}'_{i[\tilde{e}_i']} \) are well defined and \( \alpha_i = \left\{ \begin{array}{ll}
l_{\text{Ref}^T_A(\mathcal{E}', \theta')} (\tilde{e}_i') & \text{if } \Upsilon(\tilde{E}'_i, \tilde{e}_i') \\
l_{\text{Ref}^T_A(\mathcal{E}', \theta')} (\tilde{e}_i') \lor & \text{if } \Upsilon(\tilde{E}'_i, \tilde{e}_i') \end{array} \right.
\]

From the injectivity of \( \mu \) we get \( \forall i \leq m : \pi_1(\tilde{e}_i') \neq \tau \Rightarrow \pi_2(\tilde{e}_i') = \pi_2(\tilde{e}_i) \). Define \( e_j' = \pi_1(\tilde{e}_i') \) where \( i \) is chosen such that \( \tilde{e}_i = (e_j, x_2^{\kappa(e_j)} \cdot \delta(e_j) \cdot) \lor (l(e_j) = \tau \lor \tilde{e}_i = (e_j, e_j)) \). Now, we verify by induction that

\[
\tilde{E}'_i = \text{Ref}^T_A(\mathcal{E}'_{[e_0]}[e_{s_{i-1}} - 1], \theta''_i),
\]

where \( \theta''_i(e) = \left\{ \begin{array}{ll}
\theta'_i(\pi_1(\tilde{e}_j)) & \text{if } (e', x_2^{\kappa(e')} \cdot \cdot) = \tilde{e}_j' \\text{otherwise} \\
\theta'(l'(e')) & \end{array} \right. \)

Obviously for \( i = 0 \). We proceed by making a case analysis:
By induction and Lemma 6.14 we get
\[ \bar{e}_{i+1} = \text{Ref}^e_{\mathcal{A}}(\mathcal{E}'_{[e_0] \cdots [e_{s_{i-1}}-1]}[\pi_1(\bar{e}_i)], \nu_i), \]
which is equal to
\[ \text{Ref}^e_{\mathcal{A}}(\mathcal{E}'_{[e_0] \cdots [e_{s_{i-1}}-1]}, \nu_{i+1}). \]

Then \( \Upsilon(\nu_i, \pi_2(\bar{e}_i)) \), since \( \bar{e}_i \) and \( \bar{e}'_i \) have the same label and \( \nu \) is injective. Thus by induction and Lemma 6.14 we get
\[ \tilde{\mathcal{E}}_{i+1} = \text{Ref}^e_{\mathcal{A}}(\mathcal{E}'_{[e_0] \cdots [e_{s_{i-1}}-1]}[\pi_1(\bar{e}_i)], \nu_i[\pi_1(\bar{e}_i)] \rightarrow \nu_i'(\pi_1(\bar{e}_i))[\pi_2(\bar{e}_i)]). \]

Furthermore, there is \( k < i \) such that \( \bar{e}_k = (\pi_1(\bar{e}_i), *_{2}^{k(e)} \star_1 \bullet) \), since otherwise \( \bar{e}'_i \notin \text{init}(\mathcal{E}'_i) \). From the injectivity of \( \nu \) we obtain that \( \pi_1(\bar{e}_k) = \pi_1(\bar{e}_i) \), since \( \bar{e}_k \) and \( \bar{e}'_i \) have the same label. Hence, \( \mathcal{E}'_{i+1} = \text{Ref}^e_{\mathcal{A}}(\mathcal{E}'_{[e_0] \cdots [e_{s_{i-1}}-1]}, \nu_{i+1}) \), as required.

Otherwise: Then \( \neg \Upsilon(\nu'_i, \pi_2(\bar{e}_i)) \). Thus by induction and Lemma 6.14 it follows that \( \tilde{\mathcal{E}}_{i+1} = \text{Ref}^e_{\mathcal{A}}(\mathcal{E}'_{[e_0] \cdots [e_{s_{i-1}}-1]}[\pi_1(\bar{e}_i)], \nu_i'[\pi_1(\bar{e}_i)] \rightarrow \nu_i'(\pi_1(\bar{e}_i))[\pi_2(\bar{e}_i)]). \) Furthermore, there is \( k < i \) such that \( \bar{e}'_k = (\pi_1(\bar{e}_i), *_{2}^{k(e)} \star_1 \bullet) \), since otherwise \( \tilde{\mathcal{E}} \notin \text{init}(\mathcal{E}'_i) \). From the injectivity of \( \nu \) we obtain that \( \pi_1(\bar{e}_k) = \pi_1(\bar{e}_i) \). Hence, \( \mathcal{E}'_{i+1} = \text{Ref}^e_{\mathcal{A}}(\mathcal{E}'_{[e_0] \cdots [e_{s_{i-1}}-1]}, \nu_{i+1}) \), as required.

From (6.2) we obtain that \( \pi_1(\bar{e}_i) = \pi_1(\bar{e}_j) \Leftrightarrow \pi_1(\bar{e}'_i) = \pi_1(\bar{e}'_j) \). Hence, the function \( f : \hat{E} \rightarrow E' \) with \( f(e) = \pi_1(\bar{e}'_i) \) whenever \( e = \pi_1(\bar{e}_i) \) is well defined, labeling preserving and injective.

Additionally, (6.2) becomes \( \tilde{\mathcal{E}}_i = \text{Ref}^e_{\mathcal{A}}(\mathcal{E}'_{[f(e_0)] \cdots [f(e_{s_{i-1}}-1)]}, \nu_i) \), where \( \nu \) is defined by \( \nu_i'(e') = \begin{cases} \nu_i'(f^{-1}(e')) & \text{if } f^{-1}(e') \text{ is defined} \\ \nu'(e') & \text{otherwise} \end{cases} \). Furthermore, by Lemma 6.14 we get
\[ \Upsilon(\mathcal{E}'_{[f(e_0)] \cdots [f(e_{s_{i-1}}-1)]}, f(e_{s_{i-1}})) \Leftrightarrow \exists j : \pi_1(\bar{e}_j) = e_{s_{i-1}} \land \Upsilon(\mathcal{E}'_j, \bar{e}'_j) \land \left( (\lambda(\pi_1(\bar{e}_j)) = \tau \lor \pi_2(\bar{e}_j) = *_{2} \mu(e_{s_{i-1}}) \star_1 \bullet) \right). \]

Therefore, by (6.1) we obtain \( \Upsilon(\mathcal{E}'_{[f(e_0)] \cdots [f(e_{s_{i-1}}-1)]}, f(e_{s_{i-1}})) \Leftrightarrow \nu_i = \sqrt{.} \). From this and (6.2) it follows that \( ((f(e_i), \nu_i), f(\nu_i))_{i=1}^{n} \in T^e(\mathcal{E}) \).

The other case can be shown by symmetrical arguments. \( \square \)

Proof of Theorem 6.11: We verify the stronger statement claiming that there is a refinement function \( \theta' \) such that \( \text{Ref}^e_{\mathcal{A}}(\mathcal{E}, \theta') \sim_{\rho} \text{Ref}^e_{\mathcal{A}}(\mathcal{E}, \theta') \) implies \( \mathcal{E} \sim_{FU1} \mathcal{E}' \).

Define \( \mathcal{A} \subseteq \mathcal{A}ct \) to be the set of all action-names occurring in \( \mathcal{E} \) or in \( \mathcal{E}' \), i.e. \( A = \{ l(e) | e \in E \} \cup \{ l'(e') | e' \in E' \} \). Let \( \mu : \{ 1, 2 \} \times \mathcal{A} \times \mathbb{N} \rightarrow \mathcal{A}ct \setminus \mathcal{A} \) be an injective function. Such a function exists. We define for all \( a \in A \) an ETBES \( \mathcal{E}_a \), which corresponds to the process algebra term \( X = \mu(1, a, 0) ; \mu(2, a, 0) + X[f] \), where \( f(\mu(i, a, n)) = \mu(i, a, n + 1) \). In the definition the sequences \( *_{2} \star_1 \bullet \) are considered to be right bracketed and therefore to be elements of \( \mathcal{U} \).

\[ \mathcal{E}_a = ( \{ \star_{2}^{n} \star_{1} \star_{1} \bullet | n \in \mathbb{N} \land i \in \{ 1, 2 \} \}, \{ \{ \star_{2}^{n} \star_{1} \star_{1} \bullet | n \in \mathbb{N} \land j \in \{ 1, 2 \} \}, \{ \{ \star_{2}^{n} \star_{1} \star_{1} \bullet, \star_{2}^{n} \star_{1} \star_{1} \bullet | n \in \mathbb{N} \}, \{ \{ \star_{2}^{n} \star_{1} \star_{1} \bullet | n \in \mathbb{N} \}, \{ \{ \star_{2}^{n} \star_{1} \star_{1} \bullet, \mu(i, a, n) | n \in \mathbb{N} \land i \in \{ 1, 2 \} \} \} \} \}

Define \( \theta' : A \rightarrow \mathcal{ETBES} \) by \( \theta'(a) = \mathcal{E}_a \). Furthermore, define \( \mathcal{E}^{a,n} \) by
\[ \mathcal{E}^{a,n} = ( \{ \star_{2}^{n} \star_{1} \star_{1} \bullet \}, \{ \{ \star_{2}^{n} \star_{1} \star_{1} \bullet, \star_{2}^{n} \star_{1} \star_{1} \bullet \}, \emptyset, \{ \{ \star_{2}^{n} \star_{1} \star_{1} \bullet \}, \{ \star_{2}^{n} \star_{1} \star_{1} \bullet, \mu(2, a, n) \} \}) \).
Let $\mathcal{R}_b$ be a strong bisimulation such that $(\text{Ref}_A^eT(\mathcal{E}, \theta'), \text{Ref}_A^eT(\mathcal{E}', \theta')) \in \mathcal{R}_b$. Without loss of generality, $R_b$ contains only elements which can be derived from $\text{Ref}_A^eT(\mathcal{E}, \theta'), \text{Ref}_A^eT(\mathcal{E}', \theta')$. Furthermore, let $\kappa : \mathcal{U} \to \mathbb{N}$ be an isomorphism. We define the relation $\text{Ref}_{\text{FUI}}$ by

$$
\mathcal{R}_{\text{FUI}} = \{(\mathcal{E}, \mathcal{E}', \tilde{f}) \mid \tilde{f} : \text{init}_{\text{Obs}}(\mathcal{E}) \to \text{init}_{\text{Obs}}(\mathcal{E}')\text{ is a labeling preserving isomorphism} \land \\
\forall \tilde{I} \in \mathcal{P}_{\text{fin}}(\text{init}_{\text{Obs}}(\mathcal{E})) : \exists \tilde{J} \in \mathcal{P}_{\text{fin}}(\text{init}_{\text{Obs}}(\mathcal{E})): \tilde{I} \subseteq \tilde{J} \land \exists \tilde{\vartheta}, \tilde{\vartheta}' : \\
\exists e \in \tilde{E} : \tilde{\vartheta}(e) = \begin{cases} E_{\tilde{\vartheta}(e)} & \text{if } e \in \tilde{E} \land \tilde{J} \\
\tilde{E}(\tilde{f}(e)) & \text{if } e \in \tilde{J} \end{cases} \\
\land \\
\exists e' \in \tilde{E}' : \tilde{\vartheta}'(e') = \begin{cases} E_{\tilde{\vartheta}'(e')} & \text{if } e' \in \tilde{E}' \land \tilde{f}(\tilde{J}) \\
\tilde{E}'(\tilde{f}(\tilde{J})) & \text{if } e' \in \tilde{f}(\tilde{J}) \end{cases} \\
(\text{Ref}_A^eT(\mathcal{E}, \tilde{\vartheta}), \text{Ref}_A^eT(\mathcal{E}', \tilde{\vartheta}')) \in \mathcal{R}_b
$$

In the following we show that $\text{Ref}_{\text{FUI}}$ is a FUI-bisimulation. Therefore, suppose $(\mathcal{E}, \mathcal{E}', \tilde{f}) \in \text{Ref}_{\text{FUI}}$.

Then $\tilde{f}$ is such a required isomorphism by definition. Now suppose $\tilde{e} \in \text{init}(\mathcal{E})$ and $\tilde{I} \in \mathcal{P}_{\text{fin}}(\text{init}_{\text{Obs}}(\mathcal{E}))$. Then there is $\tilde{J}$ such that $\tilde{I} \cup \{\tilde{e}\} \subseteq \tilde{J}$, and $(\text{Ref}_A^eT(\mathcal{E}, \tilde{\vartheta}), \text{Ref}_A^eT(\mathcal{E}', \tilde{\vartheta}')) \in \mathcal{R}_b$, where $\tilde{\vartheta}, \tilde{\vartheta}'$ are the corresponding functions. We proceed by making a case analysis:

1. $\tilde{I}(\tilde{e}) \in \text{Obs} \land \neg \Upsilon(\mathcal{E}, \tilde{e})$: Then $\text{Ref}_A^eT(\mathcal{E}, \tilde{\vartheta}) \cup (2, \tilde{I}(\tilde{e}), \kappa(\tilde{f}(\tilde{e}))) \overrightarrow{\text{Ref}_A^eT(\mathcal{E}', \tilde{\vartheta}')} \overrightarrow{\text{Ref}_A^eT(\mathcal{E}, \tilde{\vartheta})}$ by Lemma 6.14, since $\tilde{I}(\tilde{e}) \in A$. Furthermore, because of the fact that $\mathcal{R}_b$ is a strong bisimulation, there exists $e'$ such that $(\text{Ref}_A^eT(\mathcal{E}[\tilde{e}], \tilde{\vartheta}), \text{Ref}_A^eT(\mathcal{E}', \tilde{\vartheta}')) \in \mathcal{R}_b$ and $\text{l}_{\text{Ref}_A^eT(\mathcal{E}', \tilde{\vartheta}')}^{\text{e}'}(e') = \mu(2, \tilde{I}(\tilde{e}), \kappa(\tilde{f}(\tilde{e})))$.

Thus, $\pi_1(e') = f(\tilde{e})$ and $\tilde{I}(\pi_1(e')) = \tilde{I}(\tilde{e})$ by the injectivity of $\kappa$. Moreover, we have $\text{Ref}_A^eT(\mathcal{E}, \tilde{\vartheta})^{e'} = \text{Ref}_A^eT(\mathcal{E}', \tilde{\vartheta}').$

2. $\tilde{I}(\tilde{e}) = \tau \land \neg \Upsilon(\mathcal{E}, \tilde{e})$: Then $\text{Ref}_A^eT(\mathcal{E}, \tilde{\vartheta}) \overset{\tau}{\rightarrow} \text{Ref}_A^eT(\mathcal{E}, \tilde{\vartheta})$ by Lemma 6.14. From the fact that $\mathcal{R}_b$ is a strong bisimulation, there exists $e'$ such that $(\text{Ref}_A^eT(\mathcal{E}[\tilde{e}], \tilde{\vartheta}), \text{Ref}_A^eT(\mathcal{E}', \tilde{\vartheta}')) \in \mathcal{R}_b$ and $\text{l}_{\text{Ref}_A^eT(\mathcal{E}', \tilde{\vartheta}')}^{\text{e}'}(e') = \tau$. Thus, $\tilde{I}(\tilde{e}') = \tau$ and $\text{Ref}_A^eT(\mathcal{E}', \tilde{\vartheta})^{e'} = \text{Ref}_A^eT(\mathcal{E}', \tilde{\vartheta})$.}

The cases when $\Upsilon(\mathcal{E}, \tilde{e})$ holds are carried out analogously. Furthermore, we have $\Upsilon(\mathcal{E}', \tilde{e}) \leftrightarrow \Upsilon(\mathcal{E}', \tilde{e}')$ by Lemma 6.14.

So it remains to find a function $\hat{f} : \text{init}_{\text{Obs}}(\mathcal{E}[\tilde{e}]) \times \mathbb{N} \to \text{init}_{\text{Obs}}(\mathcal{E}'[\tilde{e}'])$ that satisfies the necessary constraints. Therefore, define functions $\hat{f}_i$ and eTbes $\hat{\mathcal{E}}_i, \hat{\mathcal{E}}_i$ with $i \in \mathbb{N}$ as follows: $\hat{f}_0 = \hat{f} \cap (\hat{J} \cap \text{init}_{\text{Obs}}(\hat{\mathcal{E}}[\tilde{e}]))$, $\hat{f}_0 = \text{Ref}_A^eT(\mathcal{E}[\tilde{e}], \tilde{\vartheta})$ and $\hat{\mathcal{E}}_0' = \text{Ref}_A^eT(\mathcal{E}'[\tilde{e}], \tilde{\vartheta}')$.

1. for $2n+1$: If $\kappa^{-1}(n) \notin \text{init}_{\text{Obs}}(\mathcal{E}[\tilde{e}])$ or $\hat{f}_{2n}(\kappa^{-1}(n))$ is defined, then $\hat{f}_{2n+1} = \hat{f}_{2n}$, $\hat{\mathcal{E}}_{2n+1} = \hat{\mathcal{E}}_n$ and $\hat{\mathcal{E}}_{2n+1}' = \hat{\mathcal{E}}_{2n}'$.

   If $\kappa^{-1}(n) \notin \text{init}_{\text{Obs}}(\mathcal{E}[\tilde{e}]) \land \hat{f}_{2n}(\kappa^{-1}(n))$ is undefined, then $(\kappa^{-1}(n), \star^* \star^* \star \bullet) \in \text{init}_{\text{Obs}}(\hat{\mathcal{E}}_{2n})$.

Hence, there is $\tilde{e}'_{2n}$ with the same label such that $(\hat{\mathcal{E}}_{2n}[\tilde{e}'_{2n}], \hat{\mathcal{E}}_{2n}[\tilde{e}'_{2n}]) \in \mathcal{R}_b$, where $\tilde{e}'_{2n} = (\kappa^{-1}(n), \star^* \star^* \star \bullet)$. Define $\hat{f}_{2n+1} = \hat{f}_{2n} \cup \{((\kappa^{-1}(n), \pi_1(\tilde{e}'_{2n})), \hat{\mathcal{E}}_{2n+1} = \hat{\mathcal{E}}_{2n}[\tilde{e}'_{2n}]\}$.

2. for $2n+2$: If $\kappa^{-1}(n) \notin \text{init}_{\text{Obs}}(\mathcal{E}'[\tilde{e}])$ or $\hat{f}^{-1}_{2n}(\kappa^{-1}(n))$ is defined, then $\hat{f}_{2n+1} = \hat{f}_{2n}$, $\hat{\mathcal{E}}_{2n+1} = \hat{\mathcal{E}}_n$ and $\hat{\mathcal{E}}_{2n+1}' = \hat{\mathcal{E}}_{2n}'$. \}
If $\kappa^{-1}(n) \in \text{init}_{\text{Obs}}(\tilde{E}_{[\varepsilon]}') \land \tilde{f}^{-1}(\kappa^{-1}(n))$ is undefined, then $(\kappa^{-1}(n), \ast_2^2 \ast_1 \ast_1 \bullet) \in \text{init}_{\text{Obs}}(\tilde{E}_{2n})$.

Hence, there is $\tilde{e}_{2n}$ with the same label such that $(\tilde{E}_{2n}[\tilde{e}_{2n}], \tilde{E}_{2n}[\tilde{e}_{2n}']) \in \mathcal{R}_b$, where $\tilde{e}_{2n} = (\kappa^{-1}(n), \ast_2^2 \ast_1 \ast_1 \bullet)$. Define $\tilde{f}_{2n+1} = \tilde{f}_{2n} \cup \{ (\pi_1(\tilde{e}_{2n}), \kappa^{-1}(n)) \}$, $\tilde{E}_{2n+1} = \tilde{E}_{2n}[\tilde{e}_{2n}]$ and $\tilde{E}_{2n+1}' = \tilde{E}_{2n}[\tilde{e}_{2n}]'$. Define $\hat{f}$ is defined by $\hat{f} = \bigcup_{n \in \mathbb{N}} \hat{f}_n$.

$\hat{f}$ is a partial function, since $\hat{f}_i(\kappa^{-1}(n))$ is defined implies that its corresponding $\tilde{E}_i$ does not possess events labeled by $\mu(1, \tilde{l}(\kappa^{-1}(n)), n)$. Hence, it is not defined twice. Moreover, $\hat{f}$ is a labeling preserving isomorphism between $\text{init}_{\text{Obs}}(\tilde{E}_{[\varepsilon]}) \to \text{init}_{\text{Obs}}(\tilde{E}_{[\varepsilon]'})$, since every event of both sides will be considered in the definition.

By definition, the restriction of $\hat{f}$ to $\tilde{J} \cap \text{init}_{\text{Obs}}(\tilde{E}_{[\varepsilon]})$ is equal to $f$ if it is restricted to this set. This restriction constraint also holds for $\hat{f}^{-1}$, since otherwise $\overline{\text{Ref}_{FUI}}(\tilde{E}_{[\varepsilon]}, \check{\vartheta})$ and $\overline{\text{Ref}_{FUI}}(\tilde{E}_{[\varepsilon]'}, \check{\vartheta}')$ can not be bisimilar.

It remains to prove that $(\tilde{E}_{[\varepsilon]}, \tilde{E}_{[\varepsilon]'}, \hat{f}) \in \mathcal{R}_{FUI}$. Therefore, let $\hat{I} \in \mathcal{P}_{\text{fin}}(\text{init}_{\text{Obs}}(\tilde{E}_{[\varepsilon]}))$. Define $m = \max(\{2n + 1 \in \mathbb{N} \mid \kappa^{-1}(n) \in \hat{I} \})$ and $\hat{J} = \text{dom}(\hat{f}_m)$. Furthermore, $(\tilde{E}_m, \tilde{E}_m') \in \mathcal{R}_b$ and $\tilde{E}_m, \tilde{E}_m'$ satisfies the requirements. Hence, $(\tilde{E}_{[\varepsilon]}, \tilde{E}_{[\varepsilon]'}, \hat{f}) \in \mathcal{R}_{FUI}$.

The third requirement of the FUI-bisimulation follows by symmetrical arguments. Thus we have proved that $\text{Ref}_{FUI}$ is a FUI-bisimulation.

The construction of a function $f$ such that $(\mathcal{E}, \mathcal{E}', f) \in \text{Ref}_{FUI}$ is analogous to the construction of $\hat{f}$. Hence, $\mathcal{E} \sim_{FUI} \mathcal{E}'$. $\square$
Chapter 7

Start-Based Choice together with End-Based Choice

In this chapter, a process algebra that contains action refinement together with three choice operators is introduced. Similar to Chapter 5, termination is determined by the ‘final’ executed action.

In order to give a true concurrency denotational semantics, event structures with two different conflict relations corresponding to the start-based and respectively to the end-based determination are introduced. An operational semantics, which corresponds to the denotational semantics, is presented.

Furthermore, the coarsest congruence with respect to bisimilarity is given. An axiom system that is sound and complete for finite state processes is investigated for this equivalence.

7.1 Motivation

Motivations of an end-based choice (⊕), i.e. a choice that is triggered when an action finishes, have already been given in Section 1.3. It is reasonable that process algebras which contain an end-based choice operator and a parallel operator with action synchronization lead to some start-based choices, as it is discussed in Section 6.3.

A start-based choice (+), i.e. a choice that is triggered as soon as an action starts, is the usual kind of choice in process algebras, e.g. as in [10, 99, 133, 141, 174]. An example of a start-based choice is the following: a person standing in front of a fork has to decide immediately which direction she should take, i.e. she does not follow both directions at the same time and then make a decision depending on where she arrives.

Sometimes, it is also useful to have a choice (⊤, called end-start choice) that is end-based and start-based triggered. More precisely, it is a choice that is triggered when its right process starts an action and it is triggered when its left process finishes an action. This is for example useful to model some special kinds of disruption, described as follows. Consider the process that executes $a$ followed by $b$ ($a; b$). This process should be allowed to be disrupted by the start of action $c$ as long as action $a$ runs, i.e. after the ending of $a$ no disruption by $c$ is allowed. This disruption is modeled by $(a; b) ⊤c$. 

121
CHAPTER 7. START-BASED TOGETHER WITH END-BASED CHOICE

The usefulness of these three kinds of choices is illustrated in the following example.

**Example 7.1** Let us consider a nuclear power plant that consists of two reactors which can burn uranium or plutonium fuel rods. When the power plant gets the instructions to produce electricity, it warms up its two reactors and starts to burn either uranium or plutonium in the reactor that warms up first. The process can be disrupted during the warm-up phase. The phase during which the fuel rods are carried into the reactor is critical and may not be disrupted. This process is specified as follows in a process algebra setting. The actions considered are:

\[
\begin{align*}
  w_i &\triangleq \text{‘warm up the } i\text{-th reactor’}, \\
  c &\triangleq \text{‘cancel the warm-up phase’}, \\
  u_i &\triangleq \text{‘carry a uranium fuel rod to the } i\text{-th reactor’}, \text{ and} \\
  p_i &\triangleq \text{‘carry a plutonium fuel rod to the } i\text{-th reactor’}.
\end{align*}
\]

All actions are considered as abstractions of more concrete processes, i.e. they are refined by more concrete processes in a next specification level. Furthermore, \( N_i \) denotes the specification of the working behavior of reactor \( i \), which includes, for example, disruption possibilities. Then the nuclear power plant is specified by the process algebra expression

\[
P = \left( (w_1; (u_1 + p_1); N_1) \oplus (w_2; (u_2 + p_2); N_2) \right) \downarrow c.
\]

### 7.2 Syntax

Let \( \tau \), \( \text{Obs} \) and \( \text{Var} \) be defined as in Section 3.2 and let \( \text{Act} \) be defined as in Section 5.2. The process algebra expressions \( \text{EXP}_{se} (s \triangleq \text{start-based, } e \triangleq \text{end-based}) \) are defined by the following BNF-grammar.

\[
B ::= 0 \mid a \mid B + B \mid B\uplus B \mid B \parallel B \mid B; B \mid B \downarrow A \mid B\uparrow A \mid B[(a \rightarrow B)^{a \in A}] \mid x
\]

where \( x \in \text{Var}, a \in \text{Act} \) and \( A \subseteq \text{Obs} \). A process with respect to \( \text{EXP}_{se} \) is a pair \( \langle \text{decl}, B \rangle \) consisting of a declaration \( \text{decl} : \text{Var} \rightarrow \text{EXP}_{se} \) and an expression \( B \in \text{EXP}_{se} \). Let \( \text{PA}_{se} \) denote the set of all processes. We sometimes call an expression \( B \in \text{EXP}_{se} \) also a process if \( \text{decl} \) is clear from the context.

The intuitive meaning of the end-based choice (\( \oplus \)) and of the end-start choice (\( \downarrow \)) is given in Section 7.1. The intuitive meaning of the refinement expression \( B[(a \rightarrow B)^{a \in A}] \) is that it behaves like process \( B \) except that every execution of action \( a \) in \( A \) is substituted by the behavior of \( B_a \). The other operators are explained in Section 5.2.

### 7.3 Denotational Semantics for \( \text{PA}_{se} \)

#### 7.3.1 Start-End Bundle Event Structures (SEBES)

Event structures that are used as denotational models of \( \text{PA}_{se} \) have to handle a start-based and an end-based choice. Therefore, we introduce two relations for conflicts, one for the start-based
and another for the end-based conflict. We use the witness approach (Subsection 5.4.2) for the end-based conflict, since we follow the fa-approach, i.e. termination is determined by the action that is finally executed (see Chapter 5). The start-based conflict is modeled in the classical way, i.e. by a binary relation between events. Therefore, the start-end bundle event structures are a combination of closed bundle event structures (Definition 3.9) and extended termination bundle event structures (Definition 5.5):

**Definition 7.2 (Start-End Bundle Event Structure)** A start-end bundle event structure, sebes for short, $\mathcal{E} = (E, \sim, \succ, \mapsto, T, l)$ is an element of $\mathcal{P}(U) \times \mathcal{P}(\mathcal{P}(U) \times U) \times \mathcal{P}(\mathcal{P}(U) \times U) \times \mathcal{P}(\mathcal{P}(U)) \times \mathcal{P}(\mathcal{P}(U)) \times (U \rightarrow \text{Act})$ such that

- $\sim \subseteq E \times E$ and $\forall e \in E : \neg(e \sim e)$
- $\succ \subseteq \mathcal{P}(E) \times E$ and $\forall e \in E : \exists Z : Z \succ e$ and $\forall (Z, e) \in \succ : e \in Z$
- $\mapsto \subseteq \mathcal{P}(E) \times E$
- $T \subseteq \mathcal{P}(E)$ and $T \neq \emptyset$
- $\text{dom}(l) = E$
- $\forall e \in E : _{\sim} \succ e$ is approximation closed with respect to $E$
- $\forall e \in E : _{\sim} \mapsto e$ is approximation closed with respect to $E$
- $T$ is approximation closed with respect to $E$

Let SEBES denote the set of all start-end bundle event structures.

We call $E$ the set of events, $\sim$ the (irreflexive) (start) conflict relation, $\succ$ the (end) witness relation, $\mapsto$ the causality relation, $T$ the termination set and $l$ the action-labeling function.

The intuitive meaning of the components of a sebes is given in Section 5.4 and in Section 3.3.

**Remark 7.3** A tuple $(E, \sim, \succ, \mapsto, T, l)$ is a sebes if and only if $(E, \succ, \mapsto, T, l)$ is an eTbes (Definition 5.5) and $\sim \subseteq E \times E$ and $\forall e \in E : \neg(e \sim e)$.

**Example 7.4** Some sebes are depicted in Figure 7.1. The different components of a sebes is depicted as described in Example 5.6 and Subsection 3.3.1.

Hereafter, we consider $\mathcal{E}$ to be $(E, \sim, \succ, \mapsto, T, l)$, $\mathcal{E}_i$ to be $(E_i, \sim_i, \succ_i, \mapsto_i, T_i, l_i)$ and in general $\mathcal{E}$ to be $(E_\mathcal{E}, \sim_\mathcal{E}, \succ_\mathcal{E}, \mapsto_\mathcal{E}, T_\mathcal{E}, l_\mathcal{E})$. Furthermore, init($\mathcal{E}$) denotes the set of events which are ready to be executed and $\Upsilon(T, e)$ holds if and only if $e$ is a termination event with respect to $T$, i.e. $\mathcal{E}$ terminates by executing $e$. Formally:

**Definition 7.5** Let $\mathcal{E}$ be a sebes. The set of initial events of $\mathcal{E}$ is defined by

$$\text{init}(\mathcal{E}) = \{ e \in E \mid \neg(\exists X : X \mapsto e) \}.$$ 

The termination predicate $\Upsilon \subseteq \mathcal{P}(\mathcal{P}(U)) \times U$ is defined by

$$\Upsilon(T, e) \iff \forall X \in T : e \in X.$$
CHAPTER 7. START-BASED TOGETHER WITH END-BASED CHOICE

(\(a \oplus a\))|_{\{a\}}((a + b) \oplus c)

[diagram]

Figure 7.1: Some Start-End Bundle Event Structures

Complete Partial Order

First, we present the definition and the properties of the restriction of a sebes, which are used to define an order on SEBES.

Definition 7.6 (Restriction of a sebes) Suppose \(E \in \text{SEBES}\) and \(E' \subseteq E\). Then the restriction of \(E\) to \(E'\), denoted by \(E \upharpoonright E'\), is \((E', \sim', \succ', \mapsto', T', l')\) where

\[
\begin{align*}
\sim' &= \sim \cap (E' \times E') \\
\succ' &= \{(Z \cap E', e') \mid e' \in E' \land Z \succ e'\} \\
\mapsto' &= \{(X \cap E', e') \mid e' \in E' \land X \mapsto e'\} \\
T' &= \{X \cap E' \mid X \in T\} \\
l' &= l \upharpoonright E'
\end{align*}
\]

Lemma 7.7 Let \(E \in \text{SEBES}\) and \(E' \subseteq E\). Then \(E \upharpoonright E' \in \text{SEBES}\).

Proof: Is an immediate consequence of Corollary 2.18.

Definition 7.8 (Order on SEBES) Let \(E_i \in \text{SEBES}\). Then \(E_1 \preceq E_2\) if and only if \(E_1 \subseteq E_2\) and \(E_1 = E_2 \upharpoonright E_1\).

Remark 7.9 Suppose \(E_1, E_2 \in \text{SEBES}\), then \(E_1 \preceq E_2\) if and only if \(\sim_1 = \sim_2 \cap (E_1 \times E_1)\) and \((E_1, \succ_1, \mapsto_1, T_1, l_1)\) is less than or equal to \((E_2, \succ_2, \mapsto_2, T_2, l_2)\) with respect to the order defined in Definition 5.18.

Theorem 7.10 The set of all sebes ordered by \(\preceq\) is an \(\omega\)-complete partial order, where the least upper bound of an \(\omega\)-chain \((E_i)_{i \in \mathbb{N}}\) is \(\bigcup_i E_i = (\bigcup_i E_i; \bigcup_i \sim_i, \succ_i, \mapsto_i, T, \bigcup_i l_i)\) with

\[
\begin{align*}
\succ &= \{(Z, e) \mid \forall k : e \in E_k \Rightarrow (Z \cap E_k) \succ_k e\} \\
\mapsto &= \{(X, e) \mid \forall k : e \in E_k \Rightarrow (X \cap E_k) \mapsto_k e\} \\
T &= \{X \mid \forall k : X \cap E_k \in T_k\}
\end{align*}
\]

Proof: Is an immediate consequence of Theorem 5.19, Remark 7.3 and Remark 7.9.
7.3. DENOTATIONAL SEMANTICS FOR PA_{SE}

7.3.2 Operators on SEBES

Here, we present the operators on SEBES that will be used later to define the denotational semantics.

Definition 7.11 (Operators on SEBES) Let $A \subseteq \text{Obs}$. Then define

$\triangleright: \text{SEBES} \times \text{SEBES} \to \text{SEBES}$ with $\mathcal{E}_1 \triangleright \mathcal{E}_2 = (\tilde{E}, \tilde{\sim}, \tilde{\succ}, \tilde{\iota}, \tilde{T}, \tilde{l})$ where

$$
\begin{align*}
\tilde{E} &= (\{\ast_1\} \times E_1) \cup (\{\ast_2\} \times E_2) \\
\tilde{\sim} &= \{((\ast_i, e_i), (\ast_j, e_j)) \mid i \neq j \land e_i \in \text{init}(E_i)\} \cup \{((\ast_i, e), (\ast_i, e')) \mid e \sim_i e'\} \\
\tilde{\succ} &= \{((\{\ast_i\} \times Z) \cup (\{\ast_j\} \times \text{init}(E_j)), (\ast_i, e)) \mid Z \succ_i e \land i \neq j\} \\
\tilde{\iota} &= \{((\{\ast_i\} \times X, (\ast_i, e)) \mid X \mapsto_i e\} \\
\tilde{T} &= \{((\{\ast_1\} \times X_1) \cup (\{\ast_2\} \times X_2) \mid X_1 \in T_1 \land X_2 \in T_2\} \\
\tilde{l}((\ast_i, e)) &= l_i(e)
\end{align*}
$$

$\bowtie: \text{SEBES} \times \text{SEBES} \to \text{SEBES}$ with $\mathcal{E}_1 \bowtie \mathcal{E}_2 = (\tilde{E}, \tilde{\sim}, \tilde{\succ}, \tilde{\iota}, \tilde{T}, \tilde{l})$ where

$$
\begin{align*}
\tilde{E} &= (\{\ast_1\} \times E_1) \cup (\{\ast_2\} \times E_2) \\
\tilde{\sim} &= \{((\ast_i, e_i), (\ast_j, e_j)) \mid e_i \in \text{init}(E_i)\} \cup \{((\ast_i, e), (\ast_i, e')) \mid e \sim_i e'\} \\
\tilde{\succ} &= \{((\{\ast_i\} \times Z) \cup (\{\ast_j\} \times \text{init}(E_j)), (\ast_i, e)) \mid Z \succ_i e \land i \neq j\} \\
\tilde{\iota} &= \{((\{\ast_i\} \times X, (\ast_i, e)) \mid X \mapsto_i e\} \\
\tilde{T} &= \{((\{\ast_1\} \times X_1) \cup (\{\ast_2\} \times X_2) \mid X_1 \in T_1 \land X_2 \in T_2\} \\
\tilde{l}((\ast_i, e)) &= l_i(e)
\end{align*}
$$

$\triangleright\bowtie: \text{SEBES} \times \text{SEBES} \to \text{SEBES}$ with $\mathcal{E}_1 \triangleright\bowtie \mathcal{E}_2 = (\tilde{E}, \tilde{\sim}, \tilde{\succ}, \tilde{\iota}, \tilde{T}, \tilde{l})$ where

$$
\begin{align*}
\tilde{E} &= (\{\ast_1\} \times E_1) \cup (\{\ast_2\} \times E_2) \\
\tilde{\sim} &= \{((\ast_i, e), (\ast_i, e')) \mid e \sim_i e'\} \\
\tilde{\succ} &= \{((\{\ast_1\} \times (Z \cup X), (\ast_1, e)) \mid Z \succ_1 e \land X \in T_1\} \cup \{((\{\ast_2\} \times Z, (\ast_2, e)) \mid Z \succ_2 e\} \\
\tilde{\iota} &= \{((\{\ast_i\} \times X, (\ast_i, e)) \mid X \mapsto_i e\} \cup \{((\{\ast_1\} \times X_1, (\ast_1, e)) \mid e \in \text{init}(E_2) \land X_1 \in T_1\} \\
\tilde{T} &= \{\{\ast_2\} \times X_2 \mid X_2 \in T_2\} \\
\tilde{l}((\ast_i, e)) &= l_i(e)
\end{align*}
$$
\[ \hat{\parallel}_A : \text{SEBES} \times \text{SEBES} \rightarrow \text{SEBES} \text{ with } \mathcal{E} \hat{\parallel}_A \mathcal{E}_2 = (\tilde{E}, \tilde{\sim}, \tilde{\asymp}, \tilde{\rightarrow}, \tilde{T}, \tilde{l}) \text{ where} \]

\[
\tilde{E} = (E' \times \{\#\}) \cup (\{\#\} \times E_2^s) \cup E^s
\]

\[
E_i'^{f} = \{ e \in E_i \mid l_i(e) \notin A \}
\]

\[
E_2^s = \{ (e_1, e_2) \in E_1 \times E_2 \mid l_1(e_1) = l_2(e_2) \in A \}
\]

\[
\tilde{\sim} = \{ ((e_1, e_2), (e'_1, e'_2)) \mid e_1 \sim_1 e'_1 \lor e_2 \sim_2 e'_2 \lor (e_1 = e'_1 \neq \# e_2 \neq e'_2) \lor (e_2 = e'_2 \neq \# e_1 \neq e'_1) \}
\]

\[
\tilde{\asymp} = \{ ((e'_1, e'_2) \in E \mid e'_1 \in Z_1 \cup X_1), (e_1, \#)) \mid Z_1 \sim_1 e_1 \land X_1 \in T_1 \} \cup
\{ ((e'_1, e'_2) \in E \mid e'_2 \in Z_2 \cup X_2), (\#, e_2)) \mid Z_2 \sim_2 e_2 \land X_2 \in T_2 \} \cup
\{ ((e'_1, e'_2) \mid e'_1 \in Z_1 \cup X_1 \lor e'_2 \in Z_2 \cup X_2), (e_1, e_2)) \mid
\}
\]

\[
\tilde{\rightarrow} = \{ ((e'_1, e'_2) \in E \mid e'_1 \in X_i), (e_1, e_2)) \mid X_i \rightarrow_i e_i \}
\]

\[
\tilde{\mathcal{T}} = \{ (e_1, e_2) \in E \mid e_1 \in X_i \} \mid X_i \in T_i \}
\]

\[
\tilde{l}(e_1, e_2) = \begin{cases} l_1(e_1) & \text{if } e_2 = \# \\ l_2(e_2) & \text{otherwise} \end{cases}
\]

\[ \sqcap A : \text{SEBES} \rightarrow \text{SEBES} \text{ with } \mathcal{E} \sqcap A = \mathcal{E} \upharpoonright \{ e \in E \mid l(e) \notin A \} \]

\[\text{Ref}^e_A : \text{SEBES} \times (A \rightarrow \text{SEBES}) \rightarrow \text{SEBES} \text{ by } \text{Ref}^e_A (\mathcal{E}, \theta) = (E, \sim, \asymp, \rightarrow, T, l) \text{ where} \]

\[
E = \{ (e, \#) \mid e \in E \land l(e) \in A \land \# \in E_{\theta(l(e))} \} \cup
\{ (e, e) \in E \times E \mid l(e) \notin A \}
\]

\[
\sim = \{ ((e, \#), (e', \#')) \mid e \sim\# e' \lor (e = e' \land l(e) \in A \land \# \sim\# l(e)) \}
\]

\[
\asymp = \{ ((Z, (e, \#)) \mid \exists Z : Z \ni e \land \exists f : Z \rightarrow \mathcal{P}(U) :
\}
\]

\[
\left( \forall e' \in Z : (l(e') \notin A \land f(e') = \{ e' \}) \lor (l(e') \in A \land e' \neq \# \land f(e') \in T_{\theta(l(e'))} \} \lor
\right)
\]

\[
\left( l(e') \in A \land e' = e \land \exists X \in T_{\theta(l(e'))}, \hat{Z} : f(e') = \hat{Z} \cup \hat{X} \land \hat{Z} \ni \theta(l(e')) \right) \}
\]

\[
\rightarrow = \{ ((X, (e, \#)) \mid l(e) \in A \land X' \ni \#) \Rightarrow \hat{e} \in \text{init}(\theta(l(e))) \land \exists X : X \rightarrow e \land \exists f : X \rightarrow \mathcal{P}(U) :
\]

\[
\left( \forall e' \in X : (l(e') \notin A \land f(e') = \{ e' \}) \lor (l(e') \in A \land f(e') \in T_{\theta(l(e'))} \} \land
\right)
\]

\[
\hat{X} = \{ (e', \hat{e}) \in \hat{E} \mid e' \in X \land \hat{e} \in \hat{f}(e') \} \}
\]

\[
\hat{T} = \{ X \mid \exists X \in T \land \exists f : X \rightarrow \mathcal{P}(U) : (\forall e' \in X : (l(e') \notin A \land f(e') = \{ e' \}) \lor
\}
\]

\[
(l(e') \in A \land f(e') \in T_{\theta(l(e'))}) \land \hat{X} = \{ (e', \hat{e}) \in \hat{E} \mid e' \in X \land \hat{e} \in \hat{f}(e') \} \}
\]

\[
\hat{l}(e, \#) = \begin{cases} l(e) & \text{if } l(e) \notin A \\ l_{\theta(l(e))}(\#) & \text{if } l(e) \in A \end{cases}
\]

The definitions of these operators are similar to those presented in Subsection 5.4.3, Subsection 3.3.3 and Section 6.1. For comments on these operators, please consult the subsections mentioned.

**Lemma 7.12** All operators of Definition 7.11 are well defined, i.e. they really yield elements of SEBES.

**Proof:** The conditions of the start conflict relation are easy to check. The rest is an immediate consequence of Lemma 5.21, Lemma 6.2 and Remark 7.3 □

**Lemma 7.13** All operators of Definition 7.11 are continuous with respect to \( \subseteq \).

**Proof:** Analogous to the proof of Lemma 3.18. □
7.3.3 Denotational Meaning for PA$_{se}$

As in Subsection 3.3.4, we define the denotational semantics of expressions (EXP$_{se}$) relatively to variable assignments, i.e. functions from Var to SEBES. Variable assignments are derived from declarations, which are used to define the denotational semantics of processes (PA$_{se}$).

**Definition 7.14** Let $[[\_]] : \text{EXP}_se \times (\text{Var} \rightarrow \text{SEBES}) \rightarrow \text{SEBES}$ be defined as follows (where $\rho : \text{Var} \rightarrow \text{SEBES}$)

\[
\begin{align*}
[0]_\rho &= (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset) \\
[a]_\rho &= (\{a\}, \emptyset, \{\{a\}\}, \emptyset, \{\{a\}\}) \\
[B_1 + B_2]_\rho &= [B_1]_\rho + [B_2]_\rho \\
[B_1 \oplus B_2]_\rho &= [B_1]_\rho \oplus [B_2]_\rho \\
[B_1 ; B_2]_\rho &= [B_1]_\rho ; [B_2]_\rho \\
[B_1 \parallel A B_2]_\rho &= [B_1]_\rho \parallel [A B_2]_\rho \\
[B(a \rightarrow B_a)^a \in A]_\rho &= \text{Ref}^T_A([B]_\rho, (a \rightarrow [B_a]_\rho)^a \in A) \\
[x]_\rho &= \rho(x)
\end{align*}
\]

**Remark 7.15** $[[B]]_\rho$ is continuous for every $B \in \text{EXP}_se$. This follows analogously to Lemma 3.20, where Lemma 7.13 is used.

Assume $\text{decl} : \text{Var} \rightarrow \text{EXP}_se$. Then define $\mathcal{F}_{\text{decl}} : (\text{Var} \rightarrow \text{SEBES}) \rightarrow (\text{Var} \rightarrow \text{SEBES})$ with $\mathcal{F}_{\text{decl}}(\rho)(x) = [[\text{decl}(x)]_\rho]$. From Remark 7.15 it follows that $\mathcal{F}_{\text{decl}}$ is continuous. Therefore, we get $[[\_]] : (\text{Var} \rightarrow \text{EXP}_se) \rightarrow (\text{Var} \rightarrow \text{SEBES})$ with $[[\text{decl}]] = \text{fix}(\mathcal{F}_{\text{decl}}) = \bigsqcup_n \mathcal{F}_{\text{decl}}^n(\perp)$ is well defined from the cpo theory (Section 2.3).

**Definition 7.16** (Denotational Semantics)

Define $[[\_]] : \text{PA}_se \rightarrow \text{SEBES}$ by $[[\langle \text{decl}, B \rangle]] = [[B]]_{[[\text{decl}]]}$.

**Example 7.17** The denotational semantics of some processes is illustrated in Figure 7.1.

7.4 Operational Semantics for PA$_{se}$

Similarly to the ST semantics, we distinguish between the start and the ending of actions and relate the ending uniquely to the start of the corresponding action. There are different techniques of encoding the history (executed events or started events) in operational semantics: via static names [6, 44], via pointers [49, 61, 95, 96], via dynamic names [44, 140] and via the stack technique [44]. We adapt the stack technique to our process algebra, since it has the following advantages:

- it produces finite transition systems for a wide class of processes. Hence bisimulation equivalence is decidable for this class of processes. Moreover, the transition system derived from the stack technique needs less states than the transition system derived from the other techniques.
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 (see Section 7.7). More precisely, the standard axiom set developed by Milner [138] can be extended in order to obtain an axiomatization.

it yields an appropriate method to handle refinement operators, as it can be seen in [98].

The intuitive idea behind the stack techniques is the following: the start of an action \(a \in \text{Obs}\) is denoted in the transition system by \(a^+\); and the termination of an action \(a \in \text{Obs}\) is denoted by \(a^-\), where the natural number \(n\) indicates that exactly \(n - 1\) many \(a\)-actions that are started after the start of the corresponding \(a^-\) action are still active. In other words, if an \(a\)-action starts at position \(t_s^1\) and finishes with \(a^-\) at position \(t_f\), then the number of the \(a\)-actions that are started after position \(t_s\) and that are not finished before position \(t_f\) is exactly \(n - 1\). An illustration that may help to understand this approach is given in Figure 7.2. The number \(n\) is called the relative active number of the action corresponding to \(a^-\). We do not split internal actions (\(\tau\)), i.e. they execute \(\tau\) as usual. This is different to [44], where an internal action is split into two internal actions.

In order to define transition rules, we have to encode the information stating when the active actions were started in the expressions. For example, we have to know whether the left \(a\)-action started before the right \(a\) in \(a^-; 0 || a^-; 0\) or not (compare with Figure 7.2). This can be encoded by extending each operator that allows more than one process to be active by the information indicating to which subcomponent (and also to which relative position of the subcomponents)
the \( n \)-th active \( a \)-action corresponds. In other words, the operator has to be extended by a function from \( \text{Obs} \times \mathbb{N} \) to the natural numbers combined with the possible subcomponents. For example, the parallel operator has to be extended by a function \( \text{Obs} \times \mathbb{N} \to \{l, r\} \), where \( l \) indicates the left process and \( r \) indicates the right process of the parallel operator. Such functions have to be changed dynamically after every execution. In order to simplify the dynamical changes, these functions are encoded by strings, i.e. by a function from \( \text{Obs} \) to the set of strings over the possible subcomponents. For example, the parallel operator is extended by a function \( M : \text{Obs} \to \{l, r\}^* \). Then \( a^-; 0 \parallel M a^-; 0 \) where \( M(a) = lr \) indicates that the \( a \)-action on the right hand side has been started before the \( a \)-action on the left hand side.

The process algebra expressions \( \text{EXP}^\text{O} \) for the operational semantics are defined by the following BNF-grammar:

\[
\begin{align*}
C & ::= \ B \mid b^- \mid C + B \mid C \oplus_M C \mid C; B \mid C\|_{A,M} C \mid C\setminus_M A \\
\tilde{C} & ::= \ C \mid C \cdot \tilde{C}
\end{align*}
\]

where \( B \in \text{EXP}^\text{se}, b \in \text{Obs}, A \subseteq \text{Obs}, \tilde{A} \in \mathcal{P}_{\text{fin}}(A), M : \text{Obs} \to \{l, r\} \) and \( M_A : \text{Obs} \to ((\text{Obs} \times \mathbb{N}^+) \cup \{0, b\}) \), where \( \to \) is defined in Subsection 2.1.2. We consider function \( (a \to \tilde{C})^{a \in \tilde{A}} \) to be the function \( (a \to (\tilde{C} \cup \varepsilon))^{a \in \tilde{A}} \) where \( a \) maps onto the empty string \( \varepsilon \) if and only if \( a \notin \tilde{A} \). The symbols in the definition of \( \text{EXP}^\text{O} \) are overloaded, since they are also used in the definition of \( \text{EXP}^\text{se} \). Hence, the unique derivation of an expression of \( \text{EXP}^\text{O} \) is contradicted. Nevertheless, it does not harm our theory (both have the same transition rule) and therefore, we use the same symbols, especially to reduce the number of transition rules.

The intuitive meaning that differs from those given in Section 7.2 is the following: \( b^- \) indicates that action \( b \) is active, i.e. it has been started, but it has not been finished yet. It is able to execute \( b^- \). The end-based choice operator is extended by \( M \), since both components may contain active actions. This is not the case for the end-start choice, where only the left hand side may be an active process. The parallel operator is also extended by \( M \). Actions of the synchronization set are not relevant in \( M \), since they have to be uniquely executed on both sides. The restriction operator \( C\setminus_M A \) is also extended by \( M \), since it is used in some cases to maintain the \( M \) information of the parallel and the end-based choice expressions. The refinement operator \( C[(a \to B_a)^{a \in A}, (a \to \tilde{C}_a)^{a \in \tilde{A}}]_{M_A} \) contains additional strings of active processes \( \tilde{C}_a \) for each \( a \in A \) to encode the execution state of each active action in \( C \). Furthermore, the refinement operator has to be extended by \( M_A \) to encode the corresponding position of the active actions, where \( (a, i) \) refers to the \( i \)-th position in \( \tilde{C}_a \), \( 0 \) refers to \( C \) and \( b \) is a default value used when active actions are disrupted.

As mentioned in the beginning of this section, we have to adapt \( M \) and \( M_A \) after every action execution. Therefore, the following functions are defined, where \( w \cdot \sigma \) and \( \sigma \setminus i \) are defined in Subsection 2.1.1.

\[
\begin{align*}
\cdot \cdot \cdot : (\text{Obs} \times W) \times (\text{Obs} \to W^*) \to (\text{Obs} \to W^*) \text{ with} \\
([a, w] \cdot M)(b) & = \begin{cases} \\
\quad w \cdot M(a) & \text{if } a = b \\
\quad M(b) & \text{otherwise}
\end{cases} \\
\setminus \cdot \cdot \cdot : (\text{Obs} \to W^*) \times (\text{Obs} \times \mathbb{N}) \to (\text{Obs} \to W^*) \text{ with} \\
(M \setminus (a, i))(b) & = \begin{cases} \\
\quad M(a)\setminus i & \text{if } a = b \\
\quad M(b) & \text{otherwise}
\end{cases}
\end{align*}
\]
Function \( \cdot \) puts value \( w \) in front of string \( M(a) \) and is used when an action is started. Function \( \_ \backslash _i \) removes the \( i \)-th position in string \( M(a) \) and is used when an action finishes.

In the case of the refinement operator \( C[[a \to B_a]_{a \in A}, (a \to C_a)_{a \in A}]_{M_A} \), further functions are needed. They are necessary, since the relative pointers have to be changed when the length of \( C_a \) changes, which is the case as soon as an action of \( A \) is started in \( C \) or a process in \( C_a \) terminates. These functions are given as follows, where the \( i \)-th element of a string \( \sigma \) is denoted by \( \sigma[i] \).

\[
\cdot : (\text{Obs} \to ((A \times \mathbb{N}^+) \cup \{0, \beta\})^*) \times A \to (\text{Obs} \to ((A \times \mathbb{N}^+) \cup \{0, \beta\})^*) \quad \text{with}
(M_A \cdot a)(b)[k] \simeq \begin{cases} (a, i + 1) & \text{if } M_A(b)[k] = (a, i) \\ M_A(b)[k] & \text{otherwise} \end{cases}
\]

\[
\_ \backslash _i : (\text{Obs} \to ((A \times \mathbb{N}^+) \cup \{0, \beta\})^*) \times (A \times \mathbb{N}^+) \to (\text{Obs} \to ((A \times \mathbb{N}) \cup \{0, \beta\})^*) \quad \text{with}
(M_A \_ \backslash _i (a, j))[k] \simeq \begin{cases} (a, i - 1) & \text{if } M_A(b)[k] = (a, i) \land j < i \\ M_A(b)[k] & \text{otherwise} \end{cases}
\]

Function \( \cdot \) is used to add a process to \( \tilde{C}_a \). Function \( \_ \backslash _i \) is used when an action is removed. It reduces the relative pointers to \( \tilde{C}_a \) by 1 if it is greater than \( j \).

It is also necessary to obtain the active number of an action \( a \) from \( M \) when the active number \( i \) of its subcomponent \( p \) is given. This is done by counting the elements that are in front of the \( i \)-th occurrence of \( p \) in \( M(a) \). Formally:

Suppose \( \sigma \in W^* \) then define

\[
\tilde{\sigma} : W \times \mathbb{N}^+ \to \mathbb{N}^+ \quad \text{with}
\tilde{\sigma}(w, i) \simeq \begin{cases} \min\{j \mid \sigma[j] = w\} & \text{if } i = 1 \\ \min\{j \mid \sigma[j] = w \land j > \tilde{\sigma}(w, i - 1)\} & \text{otherwise} \end{cases}
\]

We will also use the function that permutes \( r \) and \( l \) in \( M \):

\[
\_ : (\text{Obs} \to \{l, r\}^*) \to (\text{Obs} \to \{l, r\}^*) \quad \text{with}
\overline{M(a)}[i] = \begin{cases} l & \text{if } M(a)[i] = r \\ r & \text{if } M(a)[i] = l \end{cases}
\]

Furthermore, \( \tilde{C} \) is considered to be a string. Hence, \( \tilde{C}[i] \) determines the \( i \)-th component if it exists, \( \tilde{C} \backslash i \) removes the \( i \)-th component and \( \tilde{C} \pm (i, C) \) replaces the \( i \)-th component by \( C \) if it exists.

The operational semantics for a process with respect to \( \text{EXP}_{se} \) is given by a transition system where the set of states consists of the elements of \( \text{EXP}_{se}^O \), and the transition labels are \( \mathcal{L}_{se} = \{\tau, (\tau, \sqrt{\cdot})\} \cup (\text{Obs} \times \{+, \cap \} \cup \mathbb{N} \cup (\mathbb{N} \times \{\sqrt{\cdot}\})) \).

**Remark 7.18** In [44] the internal action \( \tau \) is split into its start and into its end. This is not necessary, since the internal action can not be refined and therefore the observer can not detect the start and the ending of this action.
The transition rules $\xrightarrow{\tau_{\text{decl}}} \text{with respect to decl : Var} \to \text{EXP}_{\text{se}}$ are presented in Table 7.1, except for the transition rules of the parallel operator and those of the restriction operator, which are presented in Table 7.2, and except for the transition rules of the refinement operator, which are presented in Table 7.3. In these tables, $\gamma$ denotes an element of $\mathcal{L}_{\text{se}}$, $\bot$ denotes the function that maps every action to the empty string and $D$ is either an element of $\text{EXP}_{\text{se}}^0$ or of $\text{EXP}_{\text{se}}$. Furthermore, we write $a^+$ instead of $(a, +)$, $a^n$ instead of $(a, n)$, $a^n\sqrt{}$ instead of $(a, n, \sqrt{})$ and $\tau\sqrt{}$ instead of $(\tau, \sqrt{})$. To reduce the number of rules, we use the notation where $\sqrt{}$ is in brackets. It means that either all bracketed $\sqrt{}s$ are considered as $\sqrt{}$ or all bracketed $\sqrt{}s$ are ignored. For example, rule $Ch_3^e$ of Table 7.1 encodes the following two rules

$$
\begin{align*}
C_1 \xrightarrow{\tau} C_1' \\
C_1 \circ M C_2 \xrightarrow{\tau} C_1' \setminus_M \emptyset \\
C_2 \circ M C_1 \xrightarrow{\tau} C_1' \setminus_M \emptyset
\end{align*}
$$

$$
\begin{align*}
C_1 \xrightarrow{\tau\sqrt{}} C_1' \\
C_1 \circ M C_2 \xrightarrow{\tau\sqrt{}} C_1' \setminus_M \emptyset \\
C_2 \circ M C_1 \xrightarrow{\tau\sqrt{}} C_1' \setminus_M \emptyset
\end{align*}
$$

We give some comments on the transition rules: In $Ac_1$ an observable action starts by executing $a^+$ and results into the process that can finish this $a$ by executing $a_1\sqrt{}$, which is described in $Ac_2$. The internal action is handled in rule $Ac_3$.

The transition rule for the start-based choice ($Ch^b$) is the standard one. Any action from the right term triggers the end-start choice ($Ch_1^b$), whereas only the non-starting actions of the left term trigger the end-start choice ($Ch_2^b$, $Ch_3^b$).

String $M(a)$ of the end-based choice expression is extended whenever an $a$-action starts ($Ch^e$). The case when an action finishes is described in $Ch_2^e$, where the corresponding relative active number is calculated by $\overline{M(a)}(l, i)$. Furthermore, the relative active numbers have to be kept after this execution, which triggers the choice. This is done by using the restriction operator, where the information of the executed action is removed in $M$ and no action is forbidden. In the case where the right process triggers the choice, $M$ has to be transposed, i.e. $\overline{M}$ is taken, since the restriction operator considers the encoding by left to be active. The internal action is handled in a similar way ($Ch_3^e$). Rule $Ch_1^e$ embeds $\text{EXP}_{\text{se}}$ expressions by adding the information that no action is active.

The transition rules of the sequential operator ($S_1$, $S_2$) are the standard ones, i.e. the execution is given to the second process if and only if the first one terminates. The rules of the parallel operator have to change $M$, as it is done in the end-based choice rules. Furthermore, if one side terminates, it is removed (compare with the transition rules presented in Section 5.3). In this case, the relative active numbers are kept as in the case of the end-based choice, except that the actions of the synchronization set is forbidden. The restriction operator deals with $M$ like the end-based and the parallel operator.

Rules $R_1$, $R_2$ and $R_3$ consider the cases when the execution of the action is independent of the refinement. The case when an action that is not in $A$ starts, is considered in $R_1$. There, $M_A$ is extended by the information that the started action results from the process that gets refined. In $R_2$ the ending of an action is considered, where $M_A$ is modified in the usual way.

The case when an action that gets refined starts, is considered in $R_4$, $R_5$ and $R_6$. Rule $R_6$ considers the special case when the refinement terminates because of the execution of the internal action. In this case, also $C$ has to finish the started action, which is described by $C' \xrightarrow{a_1(\sqrt{\cdot})} C''$. 

7.4. OPERATIONAL SEMANTICS FOR PA\textsubscript{se}
Furthermore, \( \varphi \) does not change, since the started process is terminated. In the other cases (\( R_4 \) and \( R_6 \)) the process that corresponds to \( a \) (\( B_a \)) is activated and therefore attached to \( \varphi \). Furthermore, the pointers corresponding to \( a \) have to be increased by one in \( M_A \), since their positions in \( \varphi \) is changed by one. This is done by \( M_A \circ a \). Furthermore, we only consider here starting actions and internal actions for \( B_a \), since an expression of \( \text{EXP}_{se} \) has no active actions. The case when an observable action of the refinement is started (\( B_a \xrightarrow{b^+} C'_a \)), is considered in \( R_4 \). The active function is adapted as usual.

Rules \( R_7 \), \( R_8 \) and \( R_9 \) consider the case when the active refinement (\( \varphi \)) executes an action that is different to a termination action, i.e. the process remains active. Before the active process may
Table 7.2: Transition Rules for $\rightarrow^{\text{decl}}_{\beta}$ (2)

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_0$</td>
<td>$B_1 \parallel A B_2 \xrightarrow{\gamma} C' \quad B_1 \parallel A B_2 \xrightarrow{\gamma} C'$</td>
</tr>
<tr>
<td>$P_1$</td>
<td>$C \xrightarrow{a^+} C_1' \quad a \notin A \quad C \parallel_{A,M} C_2 \xrightarrow{a^+} C_1' \parallel_{A,[a,l]} M C_2 \quad C_2 \parallel_{A,M} C_1 \xrightarrow{a^+} C_2 \parallel_{A,[a,r]} M C_1'$</td>
</tr>
<tr>
<td>$P_2$</td>
<td>$C_1 \xrightarrow{a^-} C_1' \quad a \notin A \quad C_1 \parallel_{A,M} C_2 \xrightarrow{a^-<em>{M(a)(l,i)}} C_1' \parallel</em>{A,M \setminus (a,M(a)(l,i))} C_2 \quad C_2 \parallel_{A,M} C_1 \xrightarrow{a^-<em>{M(a)(r,i)}} C_2' \parallel</em>{A,M \setminus (a,M(a)(r,i))} C_1'$</td>
</tr>
<tr>
<td>$P_3$</td>
<td>$C_1 \xrightarrow{a^\gamma} C_1' \quad a \notin A \quad C_1 \parallel_{A,M} C_2 \xrightarrow{a^\gamma_{M(a)(l,i)}} C_2' \parallel_{A,M \setminus (a,M(a)(l,i))} A \quad C_2 \parallel_{A,M} C_1 \xrightarrow{a^\gamma_{M(a)(r,i)}} C_2' \parallel_{A,M \setminus (a,M(a)(r,i))} A$</td>
</tr>
<tr>
<td>$P_4$</td>
<td>$C_1 \xrightarrow{a^+} C_1' \quad C_2 \xrightarrow{a^+} C_2' \quad a \in A \quad C_1 \parallel_{A,M} C_2 \xrightarrow{a^+} C_1' \parallel_{A,M} C_2'$</td>
</tr>
<tr>
<td>$P_5$</td>
<td>$C_1 \xrightarrow{a^\gamma} C_1' \quad C_2 \xrightarrow{a^\gamma} C_2' \quad a \in A \quad C_1 \parallel_{A,M} C_2 \xrightarrow{a^\gamma} C_1' \parallel_{A,M} C_2'$</td>
</tr>
<tr>
<td>$P_6$</td>
<td>$C_1 \xrightarrow{a^-} C_1' \quad C_2 \xrightarrow{a^-} C_2' \quad a \in A \quad C_1 \parallel_{A,M} C_2 \xrightarrow{a^-} C_1' \parallel_{A,M} C_2'$</td>
</tr>
<tr>
<td>$P_7$</td>
<td>$C_1 \xrightarrow{a^\gamma} C_1' \quad C_2 \xrightarrow{a^\gamma} C_2' \quad a \in A \quad C_1 \parallel_{A,M} C_2 \xrightarrow{a^\gamma} C_1' \parallel_{A,M} C_2'$</td>
</tr>
<tr>
<td>$P_8$</td>
<td>$C_1 \xrightarrow{\tau} C_1' \quad C_2 \xrightarrow{\tau} C_2' \parallel_{A,M} C_1 \xrightarrow{\tau} C_2 \parallel_{A,M} C_1'$</td>
</tr>
<tr>
<td>$P_9$</td>
<td>$C_1 \xrightarrow{\tau} C_1' \quad C_2 \xrightarrow{\tau} C_2' \parallel_{A,M} C_1 \xrightarrow{\tau} C_2 \parallel_{A,M} C_1'$</td>
</tr>
<tr>
<td>$Res_0$</td>
<td>$B \parallel A \xrightarrow{\gamma} C' \quad B \parallel A \xrightarrow{\gamma} C'$</td>
</tr>
<tr>
<td>$Res_1$</td>
<td>$C \xrightarrow{a^+} C' \quad a \notin A \quad C \parallel_{M} A \xrightarrow{a^+} C' \parallel_{M \setminus [a,l]} A$</td>
</tr>
<tr>
<td>$Res_2$</td>
<td>$C \xrightarrow{a_m^\gamma(l,i)} C' \quad a \notin A \quad C \parallel_{M} A \xrightarrow{a_m^\gamma(l,i)} C' \parallel_{M \setminus (a,M(a)(l,i))} A$</td>
</tr>
<tr>
<td>$Res_3$</td>
<td>$C \xrightarrow{\tau^\gamma} C' \quad C \parallel_{M} A \xrightarrow{\tau^\gamma} C' \parallel_{M} A$</td>
</tr>
</tbody>
</table>
For simplicity, let $\phi = (a \rightarrow B_a)^{a \in A}$ and $\varphi = (a \rightarrow \bar{C}_a)^{a \in A}$

\[
R_0 : \frac{B[\phi, (a \rightarrow \varepsilon)^{a \in A}]}{B[\phi]} \xrightarrow{\gamma} C' \quad R_1 : \frac{C \xrightarrow{a^+} C' \quad a \notin A}{C[\phi, \varphi]_{MA} \xrightarrow{a^+} C'[\phi, \varphi]_{[a,0]\cdot MA}}
\]

\[
R_2 : \frac{C \xrightarrow{\alpha_i(\vee)} C' \quad a \notin A \quad m = \overline{M_{A}(a)}(0,i)}{C[\phi, \varphi]_{MA} \xrightarrow{\alpha_m(\vee)} C'[\phi, \varphi]_{MA\backslash(a,m)}} \quad R_3 : \frac{C \xrightarrow{\tau(\vee)} C'}{C[\phi, \varphi]_{MA} \xrightarrow{\tau(\vee)} C'[\phi, \varphi]_{MA}}
\]

\[
R_4 : \frac{C \xrightarrow{a^+} C' \quad a \in A \quad B_a \xrightarrow{b^+} C'}{C[\phi, \varphi]_{MA} \xrightarrow{\tau_{b^+}} C'[\phi, \varphi[a \rightarrow C'_a \cdot \bar{C}_a]_{[b,(a,1)]\cdot MA^\infty}}}
\]

\[
R_5 : \frac{C \xrightarrow{a^+} C' \quad a \in A \quad B_a \xrightarrow{\tau} C'_a \quad C' \xrightarrow{a_i(\vee)} C''}{C[\phi, \varphi]_{MA} \xrightarrow{\tau(\vee)} C'[\phi, \varphi[a \rightarrow C'_a \cdot \bar{C}_a]_{MA^\infty}}}
\]

\[
R_6 : \frac{C \xrightarrow{a^+} C' \quad a \in A \quad B_a \xrightarrow{\tau(\vee)} C'_a \quad C' \xrightarrow{a_i(\vee)} C''}{C[\phi, \varphi]_{MA} \xrightarrow{\tau(\vee)} C'[\phi, \varphi]_{MA}}
\]

\[
R_7 : \frac{C \xrightarrow{\alpha_i(\vee)} C' \quad a \in A \quad \bar{C}_a[i] \xrightarrow{b^+} C}{C[\phi, \varphi]_{MA} \xrightarrow{b_m} C[\phi, \varphi]_{MA \backslash(a,i)\cdot MA}}
\]

\[
R_8 : \frac{C \xrightarrow{\alpha_i(\vee)} C' \quad a \in A \quad \bar{C}_a[i] \xrightarrow{b^+} C_a'}{C[\phi, \varphi]_{MA} \xrightarrow{b_m} C[\phi, \varphi[a \rightarrow \bar{C}_a \pm (i, C'_a)]_{MA\backslash(b,m)}}}
\]

\[
R_9 : \frac{C \xrightarrow{\alpha_i(\vee)} C' \quad a \in A \quad \bar{C}_a[i] \xrightarrow{\tau} C_a'}{C[\phi, \varphi]_{MA} \xrightarrow{\tau} C[\phi, \varphi[a \rightarrow \bar{C}_a \pm (i, C'_a)]_{MA}}}
\]

\[
R_{10} : \frac{C \xrightarrow{\alpha_i(\vee)} C' \quad a \in A \quad \bar{C}_a[i] \xrightarrow{b^+ \vee} C_a'}{C[\phi, \varphi]_{MA} \xrightarrow{b_m(\vee)} C'[\phi, \varphi[a \rightarrow \bar{C}_a \pm (i, C'_a)]_{MA\backslash(b,m)}}}
\]

\[
R_{11} : \frac{C \xrightarrow{\alpha_i(\vee)} C' \quad a \in A \quad \bar{C}_a[i] \xrightarrow{\tau(\vee)} C_a'}{C[\phi, \varphi]_{MA} \xrightarrow{\tau(\vee)} C'[\phi, \varphi[a \rightarrow \bar{C}_a \pm (i, C'_a)]_{MA^\infty(a,i)}}}
\]

Table 7.3: Transition Rules for $\rightarrow_{\text{decl}}^a$ (3)
execute an action, one has to check that it is really an active action, i.e. that the corresponding action in $C$ is not disrupted. This is done by $C \xrightarrow{a_i^{-}((/))} C'$.

Please note that $C$ rather than $C'$ is used in the resulting expression, since $a_i^{-}$ has to remain active. The rest is handled as expected.

Rule $R_{10}$ and $R_{11}$ consider the case when the active process terminates. Similar as above it has to be checked that the corresponding action is still active. Furthermore, the active process is removed. As a result, some pointers corresponding to $a$ in $M_A$ have to be decreased by one, since their positions in $\varphi$ are changed. Only those pointers which are greater than $i$ have to be changed, since the other ones point to the correct position. This modification in $M_A$ is done by $M_A \xleftarrow{(a,i)}$. The handling of the relative active numbers is as usual. Please note that $C'$ rather than $C$ is used in the resulting process, since action $a_i^{-}$ finishes when its active process terminates.

The refinement process $C[\phi, \varphi]M_A$ terminates if and only if a termination action that is not refined is executed in $C$ (rule $R_2$ and $R_3$) or if the refinement terminates and its corresponding action in $C$ is a termination action (rule $R_6$, $R_{10}$ and $R_{11}$).

Remark 7.19 To reduce the state space, i.e. the expressions which are derived from a process, it is possible to modify the transition rules in such a way that every expression results in 0 if it executes a termination action. This is reasonable, since every expressions that results from the execution of a termination action is equivalent to the inactive process.

7.5 Consistency of the Operational and the 3Denotational Semantics for $\mathcal{PA}_{se}$

In this section, we show that a transition system can be derived from a sebes such that the denotational and the operational semantics yield bisimilar transition systems.

We define the remainder of a sebes similarly to Definition 5.9 and Definition 3.14.

Definition 7.20 (Remainder of a sebes) Let $E \in \text{SEBES}$ and $e \in \text{init}(E)$. Then the remainder $E[e]$ of $E$ is given by $(E', \sim', \succ', \mapsto', T', l')$ where

\[
E' = \{e' \in E \mid -(e' \sim e) \land \exists Z : Z \succ e' \land e \notin Z\}
\]

\[
\sim' = \sim \cap (E' \times E')
\]

\[
\succ' = \{(Z \cap E', e') \mid e' \in E' \land Z \succ e' \land e \notin Z\}
\]

\[
\mapsto' = \{(X \cap E', e') \mid e' \in E' \land X \mapsto e' \land e \notin X\}
\]

\[
T' = \begin{cases} 
\{X \cap E' \mid X \in T \land e \notin X\} & \text{if } \neg \Upsilon(T, e) \\
\emptyset & \text{otherwise}
\end{cases}
\]

\[
l' = l \upharpoonright E'
\]

Lemma 7.21 Let $E \in \text{SEBES}$ and $e \in \text{init}(E)$. Then $E[e] \in \text{SEBES}$.

Proof: It is an immediate consequence of Remark 7.3 and Lemma 5.10, since the remainder correspondence to the other remainder on all relevant components. 

The remainder is used to obtain an interleaving semantics for $\text{SEBES}$:
Definition 7.22 The transition relation $\rightarrow \subseteq SEBES \times \text{Act} \times SEBES$ is defined by $\rightarrow = \{(E, \gamma, E[e]) \mid E \in SEBES \land e \in \text{init}(E) \land (\Upsilon(T, e) \Rightarrow \gamma = l(e)) \land (\neg \Upsilon(T, e) \Rightarrow \gamma = l(e))\}$.

The transition system derived from the denotational semantics is bisimilar to the operational semantics that is restricted to pure action execution. Formally:

Theorem 7.23 (Consistency) Suppose $\langle decl, B \rangle \in PA_{se}$. Then the two transition systems $(\text{EXP}_{se}, \text{Act} \cup (\text{Act} \times \{\sqrt{}\}), \text{\rightarrow}_{\text{decl}}, B)$ and $(SEBES, \text{Act} \cup (\text{Act} \times \{\sqrt{}\}), \rightarrow, [[\langle decl, B \rangle]])$ are bisimilar,

where $C \xrightarrow{a(\sqrt{})}_{\text{decl}} C' \Leftrightarrow \begin{cases} C \xrightarrow{a^+}_{\text{decl}} a_{\sqrt{}}^C C' & \text{if } a \in \text{Obs} \\ C \xrightarrow{\tau(\sqrt{})}_{\text{decl}} C' & \text{if } a = \tau \end{cases}$.

Before we verify Theorem 7.23, which is done in Subsection 7.8.1, we show the stronger bisimilarity-result which says that the operational semantics is bisimilar to a $L_{se}$ labeled transition system derived from the denotational semantics. The transition system from SEBES that has labels from $L_{se}$ is defined as follows. We define the start-remainder with respect to event $e$ in order to describe the system that remains after the start of event $e$.

Definition 7.24 (Start-remainder of a sebes) Let $E \in SEBES$ and $e \in \text{init}(E)$. Then define $E[e]$ by $E[e] = E \upharpoonright \{e' \in E \mid \neg(e' \sim e)\}$.

For the $L_{se}$ labeled transition system derived from SEBES, it is necessary to save the information of the relative start of the events, i.e. we have to determine for each started event $e$ how many active events labeled with the same action as $e$ started after $e$. This is done by taking a SEBES combined with the set of partial functions from $U$ to $\mathbb{N}^+$. The partial functions are also used to encode which actions are active, i.e. have already been started. Moreover, we restrict this set further to guarantee that:

- only a finite number of events may be active,
- every started event can not be start-based in conflict with another event, i.e. each start-based choice is triggered and
- each started event has a unique relative active number, i.e. for all $a, n$ there exists at most one event $e$ labeled with $a$ that has exactly $n$ active events labeled with $a$ that started after $e$.

This is formalized by defining

$$SEBES_M = \{(E, \mathcal{M}) \in SEBES \times (U \rightarrow \mathbb{N}^+) \mid \text{dom}(\mathcal{M}) \in \mathcal{P}_{fin}(\text{init}_{\text{obs}}(E)) \land$$

$$\sim \cap (U \times \text{dom}(\mathcal{M})) = \emptyset \land \forall a \in \text{obs} : \mathcal{M} \mid \text{init}_a(E) \text{ is injective},$$

where $\text{init}_a(E) = \{e \in \text{init}(E) \mid l(e) \in A\}$ and $\text{init}_a(E)$ is a short hand for $\text{init}_{\{a\}}(E)$.

Furthermore, we need the following function to define the $L_{se}$ labeled transition system from SEBES. Function $\hat{}$ moves the relative active number of active actions by one. This function
Suppose is used to reduce the relative active number of the affected active actions by one if an event finishes.

\[ \hat{\tau}_\sim : (\mathcal{U} \rightarrow \mathbb{N}^+) \times \mathcal{P}(\mathcal{U}) \rightarrow (\mathcal{U} \rightarrow \mathbb{N}^+) \text{ with} \]

\[ (\mathcal{M} \hat{\tau} E)(e) = \begin{cases} 
\mathcal{M}(e) + 1 & \text{if } e \in E \\
\mathcal{M}(e) & \text{otherwise} 
\end{cases} \]

\[ \hat{\tau}_\sim : (\mathcal{U} \rightarrow \mathbb{N}^+) \times (\mathcal{P}(\mathcal{U}) \times \mathbb{N}^+) \rightarrow (\mathcal{U} \rightarrow \mathbb{N}^+) \text{ with} \]

\[ (\mathcal{M} \hat{\tau} E)(e) = \begin{cases} 
\mathcal{M}(e) - 1 & \text{if } e \in E \land \mathcal{M}(e) > j \\
\mathcal{M}(e) & \text{otherwise} 
\end{cases} \]

Now we are ready to present the \( L_{\text{se}} \) labeled transition system from SEBES.

**Definition 7.25** Suppose \( \mathcal{E} = (\mathcal{E}, \mathcal{M}) \in \text{SEBES}_M \) then define \( \mathcal{E}_{e(}\), which is an element of \( \text{SEBES}_M \), by

\[ \mathcal{E}_{e(} \simeq \begin{cases} 
(\mathcal{E}[e], \mathcal{M} \upharpoonright E_{\mathcal{E}[e]}) & \text{if } e \in \text{init}_\tau(\mathcal{E}) \\
(\mathcal{E}[e], (\mathcal{M} \upharpoonright (\text{init}_l(e)(\mathcal{E}), \mathcal{M}(e)))) \upharpoonright E_{\mathcal{E}[e]} & \text{if } e \in \text{init}_\text{obs}(\mathcal{E}) \land e \in \text{dom}(\mathcal{M}) \\
(\mathcal{E}(e), \{\{e, 1\} \cup (\mathcal{M} \circ \text{init}_l(e)(\mathcal{E}))) \upharpoonright E_{\mathcal{E}(e)}) & \text{if } e \in \text{init}_\text{obs}(\mathcal{E}) \land e \notin \text{dom}(\mathcal{M}) 
\end{cases} \]

The transition relation \( \hat{\sim}_c \subseteq \text{SEBES}_M \times L_{\text{se}} \times \text{SEBES}_M \) is defined by

\[ (\mathcal{E}, \mathcal{M}) \hat{\sim}_c (\mathcal{E}', \mathcal{M}') \text{ if and only if} \]

\[ (\mathcal{E}', \mathcal{M'}) = (\mathcal{E}, \mathcal{M})_{e(} \land \gamma = \begin{cases} 
\tau & \text{if } e \in \text{init}_\tau(\mathcal{E}) \land \neg \Upsilon(T, e) \\
\tau \sqrt{T} & \text{if } e \in \text{init}_\tau(\mathcal{E}) \land \Upsilon(T, e) \\
l(e)^+ & \text{if } e \in \text{init}_\text{obs}(\mathcal{E}) \land e \notin \text{dom}(\mathcal{M}) \\
l(e)_{\mathcal{M}(e)} & \text{if } e \in \text{init}_\text{obs}(\mathcal{E}) \land e \in \text{dom}(\mathcal{M}) \land \neg \Upsilon(T, e) \\
l(e)_{\mathcal{M}(e) \sqrt{T}} & \text{if } e \in \text{init}_\text{obs}(\mathcal{E}) \land e \in \text{dom}(\mathcal{M}) \land \Upsilon(T, e) 
\end{cases} \]

We have that the operational semantics is bisimilar to the \( L_{\text{se}} \) labeled transition system derived from the denotational semantics.

**Theorem 7.26** Suppose \( \langle \text{decl}, B \rangle \in \text{PA}_{\text{se}} \). Then \( \text{SEBES}_M, L_{\text{se}}, \hat{\sim}_c, ([\langle \text{decl}, B \rangle], \bot) \) is bisimilar to \( \text{EXP}_{\text{se}}, L_{\text{se}}, \hat{\sim}_{\text{decl}, B} \).

**Proof:** The proof is given in Subsection 7.8.1. \( \square \)

### 7.6 Equivalence

We introduce an equivalence relation where differences between the start and the ending of an event is made. Furthermore, the ending of an event has to be related uniquely to its start. Therefore, we call this equivalence ST-equivalence (compare with Subsection 4.2.1). In its definition, we make use of the \( L_{\text{se}} \) labeled transition systems defined in Section 7.4 and in Section 7.5.
CHAPTER 7. START-BASED TOGETHER WITH END-BASED CHOICE

Definition 7.27 \( \langle \text{decl}, B \rangle \) and \( \langle \text{decl}', B' \rangle \) of \( \text{PA}_{se} \) are ST-bisimilar (or ST-equivalent), denoted \( \langle \text{decl}, B \rangle \sim_{ST} \langle \text{decl}', B' \rangle \), if \( (\text{EXP}^{O}_{se}, L_{se}, \xrightarrow{c} \text{decl}, B) \) and \( (\text{EXP}^{O}_{se}, L_{se}, \xrightarrow{c} \text{decl}', B') \) are bisimilar (Definition 2.5).

Two sebes \( E \) and \( E' \) are ST-bisimilar, denoted \( E \sim_{ST} E' \), if \( (\text{SEBES}_{M}, L_{se}, \xrightarrow{c}, (E, \bot)) \) and \( (\text{SEBES}_{M}, L_{se}, \xrightarrow{c}, (E', \bot)) \) are bisimilar.

From Theorem 7.26 we obtain the correspondence of the ST-equivalences on \( \text{PA}_{se} \) and \( \text{SEBES} \), i.e. for all \( \langle \text{decl}, B \rangle \), \( \langle \text{decl}', B' \rangle \in \text{PA}_{se} \) we have \( \langle \text{decl}, B \rangle \sim_{ST} \langle \text{decl}', B' \rangle \) if and only if \( [[\langle \text{decl}, B \rangle]] \sim_{ST} [[\langle \text{decl}', B' \rangle]] \).

As in the standard event structures settings, ST-equivalence is the coarsest congruence for our refinement operator with respect to bisimulation equivalence.

Theorem 7.28 ST-equivalence is a congruence for the refinement operator \( \text{Ref}^{se} \), i.e. \( E \sim_{ST} E' \land \forall a \in A : \theta(a) \sim_{ST} \theta'(a) \) implies that \( \text{Ref}^{se}_{A}(E, \theta) \sim_{ST} \text{Ref}^{se}_{A}(E', \theta') \). Moreover, it is also a congruence for the operators \( \hat{+}, \hat{\downarrow}, \hat{\circ}, \hat{\cdot}, \hat{\parallel}, \hat{\setminus}, A \) and \( \hat{\setminus} A \), which are defined in Definition 7.11.

Proof: The proof is given in Subsection 7.8.2. \( \Box \)

Theorem 7.29 ST-equivalence is the coarsest congruence for \( \text{Ref}^{se} \) with respect to bisimulation equivalence, i.e. if \( \forall A, \theta : \text{Ref}^{se}_{A}(E, \theta) \sim_{b} \text{Ref}^{se}_{A}(E', \theta) \) then \( E \sim_{ST} E' \).

Proof: The proof is given in Subsection 7.8.2. \( \Box \)

Remark 7.30 All transition rules for \( \xrightarrow{c} \text{decl} \) are in panth-format [171]. The transition rules are also complete [9], since no negative literates are used. Hence, ST-equivalence is a congruence for all expression constructions of \( \text{PA}_{se} \) [9].

This fact does not immediately follow from Theorem 7.28, since processes can contain recursion.

7.7 Axiomatization

In this section, we present an axiom system for ST-equivalence with respect to \( \text{PA}_{se} \). We follow the idea of [44], where some modifications are made, since the declaration technique is used. Further modifications are necessary, since termination is determined by the final executed action and since the internal action \( \tau \) is not split into its start and end. We also use a different definition of the refinement operator taken from [98].

Following the approach mentioned, we extend the syntax by further expressions, for example by the left merge (\( \parallel \cdot \)) and the synchronization merge (\( \cdot \)) operators [8, 25]. We also add expressions considering the start and ending of actions. The equality of \( \text{PA}_{se} \) processes can be concluded from the equality of the newly introduced processes, since they are a conservative extension [9] of \( \text{PA}_{se} \).
The process algebra expressions $\text{EXP}^{Ax}_{se}$ are defined by the following BNF-grammar.

$$H ::= 0 \mid a \mid b^+ \mid H \mid b^- \mid b_q^- \mid b_q^\vee \mid H \mid \tau \sqrt{;} \mid H \mid H + H \mid H_H \mid H_H^\perp \mid H_H^\perp H \mid H_H^\perp \mid H_H^\perp \mid H_H^\perp$$

$$H \oplus M H \mid H \oplus M H \mid H \mid H \mid (a \rightarrow H)\in A, (a \rightarrow H)\in A_{\text{M}}$$

$$\text{H}'[(a \rightarrow H)\in A', (a \rightarrow H)\in A, (a \rightarrow H)\in A_{\text{M}}] \mid H \mid H \mid H \mid H \mid H \mid \| \mid H \mid \| \mid M \mid A \mid x$$

$$\bar{H} ::= H \mid H \cdot \bar{H}$$

where $x \in \text{Var}$, $a \in \text{Act}$, $b \in \text{Obs}$, $A \subseteq \text{Obs}$, $\bar{A} \in \mathcal{P}_{\text{fin}}(A)$, $M : \text{Obs} \rightarrow \{l, r\}$ and $M_A : \text{Obs} \rightarrow \{(A \times \mathbb{N}^+) \cup \{0, b\}\}$. As in Section 7.4, we consider function $(a \rightarrow \bar{H})\in A$ to be function $(a \rightarrow (\bar{H} \cup \varepsilon))\in A$ where $a$ maps to $\varepsilon$ if and only if $a \notin \bar{A}$. A process with respect to $\text{EXP}^{Ax}_{se}$ is a pair $(\text{decl}, H)$ consisting of a declaration $\text{decl} : \text{Var} \rightarrow \text{EXP}^{Ax}_{se}$ and an expression $H \in \text{EXP}^{Ax}_{se}$. Let $\text{PA}^{Ax}_{se}$ denote the set of all processes. We sometimes call an expression $H \in \text{EXP}^{Ax}_{se}$ also a process if decl is clear from the context.

The intuitive meaning of the new expressions are as follows: $b^+; H$ is the process that evolves into $H$ by starting a $b$-action. Process $b_q^-$ terminates the active $b$-action that started before the last $q - 1$ active $b$-actions. $b_q^\vee; H$ is similar to $b_q^-$, except that also the whole process is terminated, i.e. it executes $b_q^\vee$ and evolves into 0 (and not into $H$).

The end-start ($+/\cdot$) and the end choice ($\oplus$) together with the parallel operator are extended with left merge ones ($+^- \mid, \oplus^- \mid, \|^- \mid A, \|^- M$). In these operators the left process has to execute the next action. A synchronization merge operator for the parallel operator is also introduced ($\|^- A, \|^- M$). Here, the next action that is executed has to be obtained from communication.

Process $H_1[(a \rightarrow H_a)\in A, (a \rightarrow H_a')\in A, (a \rightarrow \bar{H}_a)\in A_{\text{M}}]$. $H_2$ has the same behavior as $H_2[(a \rightarrow H_a')\in A, (a \rightarrow \bar{H}_a)\in A_{\text{M}}]$ except that $H_2$ and $(a \rightarrow H_a')\in A$ are replaced by $H_1$ (respectively by $(a \rightarrow H_a)\in A$). $H_1$ and $(a \rightarrow H_a)\in A$ are used to determine the next actions and $H_1$ and $(a \rightarrow H_a')\in A$ are used to determine the future behavior of the process. Please note that $H_1$ is only used for the future behavior if $H_2$ does not execute the action. This new refinement expression is introduced for the axiom system, since we have to expand the refinement to determine the next action of the refinement (expand $U[\phi|a \rightarrow T + R|, \varphi]_{\text{MA}}$ to $U[\phi|a \rightarrow T|, \varphi]_{\text{MA}} + U[\phi|a \rightarrow R|, \varphi]_{\text{MA}}$). But if we use this expansion, we forget the original refinement function, which is essential for the future behavior, for example if $U$ is equal to $a; a$. Therefore, the original refinement function is kept in the expression and only the copy of this function may expand further. Similar arguments hold for $H_1$, since we have to check, whether an action is still active, which destroys the original process.

The operational semantics ($\rightarrow^\tau$) for $\text{PA}^{Ax}_{se}$ is given as follows. The transition rules for the operators that exist in $\text{EXP}^{Ax}_{se}$ are the old ones presented in Table 7.1, Table 7.2 and Table 7.3. The transition rules for the newly introduced operators are given in Table 7.4 and Table 7.5.

We give some comments on the transition rules. One may expect that the processes in rule $Ac_0$ and $Ac_7$ should evolve into $H$. This is not reasonable, since the execution of a termination action has to lead to an inactive process. This is also respected in the axioms $b_q^+ = b_q^\vee; T$ and $\tau = \tau \sqrt{;} T$ from Table 7.6, where the processes evolve to 0 for all these cases. The other rules are just as expected.

ST-equivalence is adapted to $\text{PA}^{Ax}_{se}$ in the straightforward way, i.e. $\langle \text{decl}, H \rangle$ and $\langle \text{decl}', H' \rangle$ are $\text{ST}$-bisimilar (or $\text{ST}$-equivalent), denoted $\langle \text{decl}, H \rangle \sim_{ST} \langle \text{decl}', H' \rangle$, if $(\text{EXP}^{Ax}_{se}, L_{se}, \rightarrow^\tau_{\text{decl}}, H)$
\[ \begin{align*}
A_c_4 & : \quad b^+; H \xrightarrow{b^+} H \\
A_c_5 & : \quad b_q^+ b_q \xrightarrow{\varnothing} 0 \\
A_c_6 & : \quad b_q^+; H \xrightarrow{b_q^+} 0 \\
A_c_7 & : \quad \tau^\varnothing; H \xrightarrow{\tau^\varnothing} 0 \\
C_h_1^{Lh} & : \quad H_1 \xrightarrow{a^+} H_1' \\
 & \quad H_1 \oplus_M H_2 \xrightarrow{\alpha^+} H_1' \oplus_\{\alpha,\beta\} H_2 \\
C_h_2^{Lh} & : \quad H_1 \xrightarrow{\gamma} H_1' \\
 & \quad H_1 \oplus_M H_2 \xrightarrow{\gamma} H_1' \\
C_h_1^{Le} & : \quad H_1 \xrightarrow{a^+} H_1' \\
C_h_2^{Le} & : \quad H_1 \xrightarrow{a^-}(\varnothing) H_1' \\
 & \quad H_1 \oplus_M H_2 \xrightarrow{\alpha^+_M(a_M(i,l))} H_1' M(\{a,M(a)l,i\}) \varnothing \\
C_h_3^{Le} & : \quad H_1 \xrightarrow{\tau(\varnothing)} H_1' \\
 & \quad H_1 \oplus_M H_2 \xrightarrow{\tau(\varnothing)} H_1' M(\emptyset) \\
P_1^L & : \quad H_1 \xrightarrow{a^+} H_1' \\
 & \quad a \notin A \\
 & \quad H_1 M(A,M) H_2 \xrightarrow{a^+} H_1' \parallel A,M H_2 \\
P_2^L & : \quad H_1 \xrightarrow{a^-} H_1' \\
 & \quad a \notin A \\
 & \quad H_1 M(A,M) H_2 \xrightarrow{a^-_{M(a)(i,l)}} H_1' \parallel A,M \{a,M(a)l,i\} H_2 \\
P_3^L & : \quad H_1 \xrightarrow{a^- \varnothing} H_1' \\
 & \quad a \notin A \\
 & \quad H_1 M(A,M) H_2 \xrightarrow{a^-_{M(a)(i,l)}} H_1' \parallel \varnothing \{a,M(a)l,i\} A \\
P_4^L & : \quad H_1 \xrightarrow{\tau} H_1' \\
 & \quad \tau \parallel A,M H_2 \xrightarrow{\tau} H_1' \parallel A,M H_2 \\
P_5^L & : \quad H_1 \xrightarrow{\tau \varnothing} H_1' \\
 & \quad \tau \parallel A,M H_2 \xrightarrow{\tau \varnothing} H_1' \parallel \varnothing A \\
P_1^S & : \quad H_1 \xrightarrow{\tau} H_1', H_2 \xrightarrow{\alpha^+} H_2', a \in A \\
 & \quad H_1 M(A,M) H_2 \xrightarrow{\alpha^+} H_1' \parallel A,M H_2 \\
P_2^S & : \quad H_1 \xrightarrow{\alpha^- \varnothing} H_1', H_2 \xrightarrow{\alpha^-} H_2', a \in A \\
 & \quad H_1 M(A,M) H_2 \xrightarrow{\alpha^- \varnothing} H_1' \parallel \varnothing A \\
P_3^S & : \quad H_1 \xrightarrow{\alpha^-} H_1', H_2 \xrightarrow{\alpha^-} H_2', a \in A \\
 & \quad H_1 M(A,M) H_2 \xrightarrow{\alpha^-} H_1' \parallel A,M H_2 \\
P_4^S & : \quad H_1 \xrightarrow{\alpha^- \varnothing} H_1', H_2 \xrightarrow{\alpha^-} H_2', a \in A \\
 & \quad H_1 M(A,M) H_2 \xrightarrow{\alpha^- \varnothing} H_1' \parallel \varnothing A \\
 & \quad H_2 M(A,M) H_1 \xrightarrow{\alpha^-} H_2' \parallel M A \\
\end{align*} \]

Table 7.4: Transition Rules for \( \xrightarrow{\text{decl}} (1) \)
For simplicity, let $\phi' = (a \rightarrow H_a)^{a \in A}$ and $\varphi = (a \rightarrow \bar{H}_a)^{a \in A}$

\begin{align*}
R'_1 : & \quad H \xrightarrow{a^+} H' \quad a \notin A \\
D[\phi, \phi', \varphi]_{M_A} H \xrightarrow{a^+} H'[\phi, \varphi]_{[a,0], M_A} \\
R'_2 : & \quad H \xrightarrow{a_\gets (\sqrt{\cdot})} H' \quad a \notin A \\
D[\phi, \phi', \varphi]_{M_A} H \xrightarrow{a_\gets (\sqrt{\cdot})} H'[\phi, \varphi]_{M_A \setminus (a,m)} \\
R'_3 : & \quad H \xrightarrow{\tau(\sqrt{\cdot})} H' \\
D[\phi, \phi', \varphi]_{M_A} H \xrightarrow{\tau(\sqrt{\cdot})} H'[\phi, \varphi]_{M_A} \\
R'_4 : & \quad H \xrightarrow{a^+} H' \quad a \in A \\
H_a \xrightarrow{b^+} H'_a \\
D[\phi, \phi', \varphi]_{M_A} H \xrightarrow{b^+} H'[\phi, \varphi[a \rightarrow H'_a \cdot H_a]]_{[b,(a,1); (M_A \setminus a)]} \\
R'_5 : & \quad H \xrightarrow{a^+} H' \quad a \in A \\
H_a \xrightarrow{\tau} H'_a \\
D[\phi, \phi', \varphi]_{M_A} H \xrightarrow{\tau} H'[\phi, \varphi[a \rightarrow H'_a \cdot H_a]]_{M_A \setminus a} \\
R'_6 : & \quad H \xrightarrow{a^+} H' \quad a \in A \\
H_a \xrightarrow{\tau \sqrt{\cdot}} H'_a \\
H' \xrightarrow{a_\gets (\sqrt{\cdot})} H'' \\
D[\phi, \phi', \varphi]_{M_A} H \xrightarrow{\tau \sqrt{\cdot}} H''[\phi, \varphi]_{M_A} \\
R'_7 : & \quad H \xrightarrow{a_\gets (\sqrt{\cdot})} H' \quad a \in A \\
\bar{H}_a[i] \xrightarrow{b^+} H'_a \\
D[\phi, \phi', \varphi]_{M_A} H \xrightarrow{b^+} D[\phi, \varphi[a \rightarrow \bar{H}_a \pm (i, H'_a)]]_{[b,(a,i); M_A]} \\
R'_8 : & \quad H \xrightarrow{a_\gets (\sqrt{\cdot})} H' \quad a \in A \\
\bar{H}_a[i] \xrightarrow{b^j} H'_a \\
m = M_A(b)((a, i), j) \\
D[\phi, \phi', \varphi]_{M_A} H \xrightarrow{b^j} D[\phi, \varphi[a \rightarrow \bar{H}_a \pm (i, H'_a)]]_{M_A \setminus (b,m)} \\
R'_9 : & \quad H \xrightarrow{a_\gets (\sqrt{\cdot})} H' \quad a \in A \\
\bar{H}_a[i] \xrightarrow{\tau} H'_a \\
D[\phi, \phi', \varphi]_{M_A} H \xrightarrow{\tau} D[\phi, \varphi[a \rightarrow \bar{H}_a \pm (i, H'_a)]]_{M_A} \\
R'_{10} : & \quad H \xrightarrow{a_\gets (\sqrt{\cdot})} H' \quad a \in A \\
\bar{H}_a[i] \xrightarrow{b^j \sqrt{\cdot}} H'_a \\
m = M_A(b)((a, i), j) \\
D[\phi, \phi', \varphi]_{M_A} H \xrightarrow{b^j \sqrt{\cdot}} H'[\phi, \varphi[a \rightarrow \bar{H}_a \setminus i]]_{(M_A \setminus (a,i)) \setminus (b,m)} \\
R'_{11} : & \quad H \xrightarrow{a_\gets (\sqrt{\cdot})} H' \quad a \in A \\
\bar{H}_a[i] \xrightarrow{\tau \sqrt{\cdot}} H'_a \\
D[\phi, \phi', \varphi]_{M_A} H \xrightarrow{\tau \sqrt{\cdot}} H'[\phi, \varphi[a \rightarrow \bar{H}_a \setminus i]]_{M_A \setminus 1(a,i)} \\
\end{align*}

Table 7.5: Transition Rules for $\xrightarrow{\varphi} (2)$
and \((\text{EXP}^{\text{Ax}}, \mathcal{L}_{\text{se}}, \rightarrow_{\text{decl}'}, H')\) are bisimilar. Furthermore, ST-equivalence remains a congruence for \(\text{PA}^{\text{Ax}}_{\text{se}}\), since the transition rules are in panth format and complete (compare with Remark 7.30).

As in [44], we restrict our axioms for recursion to sequential guarded processes. The definition of sequential guarded has to be adjusted to process algebras that are based on the declaration technique. This adjustment is done with respect to a subset of \(\text{Var}\), since we restrict our axioms to those processes that use only a finite number of variables. For example, we do not allow processes like \(\langle \text{decl}, x_1 \rangle\) where \(\text{decl}(x_i) = x_{i+1}\). This restriction is done, since we are only interested in derivation rules with a finite number of pre-conditions.

**Definition 7.31** A declaration \(\text{decl} : \text{Var} \rightarrow \text{EXP}^{\text{Ax}}_{\text{se}}\) is sequentially guarded with respect to \(V \in \mathcal{P}(\text{Var})\) if \(\forall x \in V : \exists n_x \in \mathbb{N}, u_x : \mathbb{N} \rightarrow \text{Act}, f_x : \mathbb{N} \rightarrow V : \text{decl}(x) = \sum_{j=1}^{n_x} u_x(j); f_x(j)\).

\(\text{SeqG}\) denotes the set of all declarations that are sequentially guarded by some \(V\), i.e. \(\text{SeqG} = \{(\text{decl}, V) \mid \text{decl} \text{ is sequentially guarded with respect to } V\}\).

We introduce the following axiom systems for \(\text{PA}^{\text{Ax}}_{\text{se}}\). This is done by presenting axioms (Table 7.6 and Table 7.7) to conclude equality with respect to the same declaration. Additionally, we present a derivation rule that relates processes with possibly different declarations.

**Definition 7.32** Let \(\text{decl} : \text{Var} \rightarrow \text{EXP}^{\text{Ax}}_{\text{se}}\) and let \(H, H' \in \text{EXP}^{\text{Ax}}_{\text{se}}\). Then we write \(\vdash_{\text{decl}} H = H'\) if \(H = H'\) can be derived from the axioms presented in Table 7.6 and Table 7.7.

Furthermore, the equality deduction of \(\text{PA}^{\text{Ax}}_{\text{se}}\) is given by the following rule

\[
\frac{(\text{decl}', \{x_0, ..., x_n\}) \in \text{SeqG} \quad \forall i \leq n : \vdash_{\text{decl}} H_i = \text{decl}'(x_i)\{((H_j/x_j)\}_{j \in \{0, ..., n\}})}{\vdash \langle \text{decl}, H_0 \rangle = \langle \text{decl}', x_0 \rangle} \tag{7.1}
\]

where \(H\{((H_j/x_j)\}_{j \in \{0, ..., n\}}\) denotes the simultaneous replacement (substitution) of every occurrence of \(x_j\) by \(H_j\) in expression \(H\).

### 7.7.1 Soundness

In this subsection, we show that the deductive system presented is sound with respect to ST-equivalence. First, we consider the soundness of \(\vdash_{\text{decl}}:\)

**Lemma 7.33** If \(\vdash_{\text{decl}} H = H'\) then \(\langle \text{decl}, H \rangle\) and \(\langle \text{decl}, H' \rangle\) are ST-equivalent.

**Proof:** It can be straightforwardly checked that all axioms are sound. The rest follows from the fact that ST-equivalence is a congruence. \(\square\)

Now we are ready to verify the soundness of our equality deduction:

**Theorem 7.34 (Soundness)** If \(\vdash \langle \text{decl}, H \rangle = \langle \text{decl}', H' \rangle\) then \(\langle \text{decl}, H \rangle\) and \(\langle \text{decl}', H' \rangle\) are ST-equivalent.

**Proof:** The proof is given in Subsection 7.8.3. \(\square\)
Let $a \in A$, $b \in \text{Obs}$, $c \in \text{Obs} \setminus A$, $R, T, U \in \text{EXP}^{Ax}_{se}$ and $q, k \in \mathbb{N}$ with $q \neq k$

\[
b = b^+; b^- = b_1^-; b_q^- = b_q^-; T; \tau = \tau \sqrt{T}
\]

\[
T + U = U + T \quad (T + U) + R = T + (U + R) \quad T + T = T \quad T + 0 = T
\]

\[
T \uparrow U = (T \uparrow U) + U \quad (T + U) \uparrow R = (T \uparrow R) + (U \uparrow R)
\]

\[
(b^+; T) \uparrow U = b^+; (T \uparrow U) \quad (b_q^-; T) \downarrow U = b_q^-; T
\]

\[
\tau; T \uparrow U = \tau; T \quad 0 \uparrow T = 0
\]

\[
T \uparrow_M U = (T \uparrow_M U) \uparrow_M U \quad (T + U) \uparrow_M R = (T \uparrow_M R) + (U \uparrow_M R)
\]

\[
(b^+; T) \uparrow_M R = b^+; (T \uparrow_M[M,b]) \quad (\tau; T) \uparrow_M R = \tau; (T \setminus_M 0)
\]

\[
(b_q^-; T) \uparrow_M R = b_q^-; (T \setminus_M[M,b]) \quad (\tau; T) \uparrow_M R = \tau; (T \setminus_M A)
\]

\[
(a^+; T) \uparrow_M R = 0 \quad (a_q^-; T) \uparrow_M R = 0 \quad 0 \uparrow_M R = 0
\]

\[
T \uparrow A_{\overline{M}} R = R \uparrow A_{\overline{M}} T \quad (T + U) \uparrow A_{\overline{M}} R = (T \uparrow A_{\overline{M}} R) + (U \uparrow A_{\overline{M}} R)
\]

\[
(c^+; T) \uparrow A_{\overline{M}} R = 0 \quad (c_q^-; T) \uparrow A_{\overline{M}} R = 0 \quad R \uparrow A_{\overline{M}} R = 0
\]

\[
(a^+; T) \uparrow A_{\overline{M}} (a^+; R) = a^+; (T \uparrow A_{\overline{M}} R) \quad (a^+; T) \uparrow A_{\overline{M}} (a^-; R) = 0
\]

\[
(a_q^-; T) \uparrow A_{\overline{M}} (a_q^-; R) = 0 \quad (a_q^-; T) \uparrow A_{\overline{M}} (a^-; R) = a_q^-; (T \setminus_M A)
\]

\[
(a_q^-; T) \uparrow A_{\overline{M}} (a_q^-; R) = a_q^-; (T \setminus_M A) \quad 0 \uparrow A_{\overline{M}} R = 0
\]

\[
(T + U) \setminus M A = (T \setminus M A) + (U \setminus M A)
\]

\[
(c^+; T) \setminus M A = c^+; (T \setminus M A) \quad (a^+; T) \setminus M A = 0
\]

\[
(c_q^-; T) \setminus M A = c_q^-; (T \setminus M A) \quad (a_q^-; T) \setminus M A = 0
\]

\[
(\tau; T) \setminus M A = \tau; (T \setminus M A) \quad 0 \setminus M A = 0
\]

\[
x = \text{decl}(x)
\]
Let $\phi = (z \rightarrow G_z)^{\varphi \in A}$, $\varphi = (z \rightarrow G_z)^{\varphi \in A}$, $\varphi \in A$, $c \in \text{Obs} \setminus A$, $R, T, U \in \text{EXP}_{se}$ and $q_k \in \mathbb{N}$

$$U[\phi, \varphi]_M = U[\phi, \varphi]_M U$$

$$U[\phi, \varphi']_M (T + R) = U[\phi, \varphi']_M T + U[\phi, \varphi']_M R$$

$$U[\phi, \varphi'[a \rightarrow T + R], \varphi']_M = U[\phi, \varphi'[a \rightarrow T], \varphi']_M + U[\phi, \varphi'[a \rightarrow R], \varphi']_M V$$

$$U[\phi, \varphi', \varphi]_M (c^+; R) = c^+; R[\phi, \varphi]_{[c, 0]; M}$$

$$U[\phi, \varphi', \varphi]_M (c^+; R) = c^+; R[\phi, \varphi]_{[c, 0]; M}$$

$$U[\phi, \varphi', \varphi]_M (\tau(\sqrt{\cdot}); R) = \tau(\sqrt{\cdot}); R[\phi, \varphi]_{M}$$

Table 7.7: Axioms for the Refinement Operators
7.7. AXIOMATIZATION

7.7.2 Completeness

We will show that our axioms are complete with respect to guarded processes that are finite state. Before we present the definition of guarded and finite state, we restrict the processes to those that use only a finite number of variables.

Definition 7.35 A process $\langle \text{decl}, H \rangle \in \mathbb{P}_{\text{se}}^\text{Ax}$ is specified by $V \in \mathcal{P}($Var$)$ if $|V| < |\mathbb{N}|$ and every variable occurrence in $H$ is an element of $V$ and every variable occurrence in $\text{decl}(x)$ is an element of $V$ for any $x \in V$. VarSp denotes the set of all processes that are specified by some $V$, i.e. $\text{VarSp} = \{\langle \text{decl}, H \rangle \mid \langle \text{decl}, H \rangle \text{ is specified by } V \}.$

A process is guarded if every variable $x$ used by the process is behind an action $(a, \tau; H)$ or behind a started action $(b^−, b_q^−, b_q^−; H)$ in the scope of $\text{decl}(x)$. It is not sufficient that $x$ is behind a starting action $(b^+)$, since otherwise it would be possible for infinitely many actions to start without any of them finishing. For example, $\langle \text{decl}, x \rangle$ with $\text{decl}(x) = b^+; x$ starts infinitely many $b$-actions. The possibility to start infinitely many actions without finishing one is problematic for our verifications. We define guardedness only for processes that use only finitely many variables, i.e. processes that are in VarSp.

Definition 7.36 (Guarded) Define $G \subseteq \text{VarSp} \times \mathcal{P}($Var$)$ by

\[
\begin{align*}
G((\langle \text{decl}, H \rangle, V), \bar{V}) & \quad \text{if } H \in \{0, a, b^−, b_q^−, b_q^−; H', \tau; H'\} \\
G((\langle \text{decl}, H \rangle, V), \bar{V}) & \quad \Leftrightarrow \quad G((\langle \text{decl}, H' \rangle, V), \bar{V}) \quad \text{if } H \in \{b^+; H', H''; H'\setminus M.A\} \\
G((\langle \text{decl}, H \rangle, V), \bar{V}) & \quad \Leftrightarrow \quad G((\langle \text{decl}, H_1 \rangle, V), \bar{V}) \wedge G((\langle \text{decl}, H_2 \rangle, V), \bar{V}) \\
& \quad \text{if } H \in \{H_1 + H_2, H_1 + H_2, H_1 + H_2, H_1 + H_2, H_1 + H_2, H_1 + H_2, \\
& \quad \quad H_1|_{A,M} H_2, H_1|_{A,M} H_2, H_1|_{A,M} H_2\} \\
G((\langle \text{decl}, H[a \rightarrow H_a]^{a \in A}, (a \rightarrow H_a)_{_{A,M,A}}), V), \bar{V}) & \quad \Leftrightarrow \quad G((\langle \text{decl}, H \rangle, V), \bar{V}) \wedge \\
& \quad \forall a \in A : G((\langle \text{decl}, H_a \rangle, V), \bar{V}) \wedge \\
& \quad \forall a \in A : \forall i \leq |H_a| : G((\langle \text{decl}, H_a[i] \rangle, V), \bar{V}) \\
G((\langle \text{decl}, H[a \rightarrow H_a]^{a \in A}, (a \rightarrow H_a)_{_{A,M,A}}), V), \bar{V}) & \quad \Leftrightarrow \quad G((\langle \text{decl}, H \rangle, V), \bar{V}) \wedge G((\langle \text{decl}, H' \rangle, V), \bar{V}) \wedge \\
& \quad \forall a \in A : G((\langle \text{decl}, H_a \rangle, V), \bar{V}) \wedge G((\langle \text{decl}, H'_a \rangle, V), \bar{V}) \wedge \\
& \quad \forall a \in A : \forall i \leq |H_a| : G((\langle \text{decl}, H_a[i] \rangle, V), \bar{V}) \\
G((\langle \text{decl}, x \rangle, V), \bar{V}) & \quad \Leftrightarrow \quad G((\langle \text{decl}, \text{decl}(x) \rangle, V), V \cup \{x\}) \wedge x \notin \bar{V}
\end{align*}
\]

A process $\langle \text{decl}, H \rangle \in \mathbb{P}_{\text{se}}^\text{Ax}$ is guarded if there is $V \in \mathcal{P}($fin$($Var$)$) such that $\langle \text{decl}, H \rangle, V \in \text{VarSp}$ and for all $x \in V : G((\langle \text{decl}, x \rangle, V), \emptyset)$.

The set of all guarded processes is denoted by $\mathbb{P}_{\text{se}}^\text{Gu}$.

The guarded predicate ($G$) can be applied directly to the expression of a guarded process, as it is illustrated by the following lemma.

Lemma 7.37 Suppose $\langle \text{decl}, H \rangle \in \mathbb{P}_{\text{se}}^\text{Ax}$ is guarded then there is $V \in \mathcal{P}($fin$($Var$)$) such that $\langle \text{decl}, H \rangle, V \in \text{VarSp}$ and $G((\langle \text{decl}, H \rangle, V), \emptyset)$.
**Proof:** By definition, there exists \( V \in \mathcal{P}_{\text{fin}}(\text{Var}) \) such that \( \langle \langle \text{decl}, H \rangle, V \rangle \in \text{VarSp} \) and for all \( x \in V : \mathcal{G}(\langle \langle \text{decl}, x \rangle, V \rangle, \emptyset) \). The rest can be easily checked by structural induction on \( H \). \( \square \)

Guardedness is preserved by the transition rules, i.e. if \( \langle \text{decl}, H \rangle \in \text{PA}_{\text{Gu}} \) and \( H \overset{\gamma}{\rightarrow}^{z_{\text{decl}}} H' \) then \( \langle \text{decl}, H' \rangle \in \text{PA}_{\text{Gu}} \). This just results from the fact that no new variables can be used by \( H' \) and guardedness only depends on \( \text{decl} \).

Finite state processes are introduced as follows:

**Definition 7.38** An element of \( \text{PA}_{\text{se}}^{\text{Ax}} \) is finite state if its corresponding transition system has a finite set of states.

We have the following completeness result:

**Theorem 7.39 (Completeness)** Suppose \( \langle \text{decl}, H \rangle, \langle \text{decl}', H' \rangle \in \text{PA}_{\text{se}}^{\text{Gu}} \) are finite state and ST-equivalent. Then \( \vdash \langle \text{decl}, H \rangle = \langle \text{decl}', H' \rangle \).

**Proof:** The proof is given in Subsection 7.8.4. \( \square \)

### 7.8 Proofs

#### 7.8.1 Proof of the Consistency Results

The proof of Theorem 7.26 is analogous to the proof of Theorem 3.25, i.e. we introduce an event based transition relation. Then we show that this transition system is bisimilar to \( (\text{EXP}_{\text{se}}^{\text{O}}, \mathcal{L}_{\text{se}}, \rightarrow^{c_{\text{decl}}}, B) \) and that it is in addition bisimilar to \( (\text{SEBES}, \mathcal{L}_{\text{se}}, \leftarrow^{c}, \llbracket \langle \text{decl}, B \rangle \rrbracket) \). And so Theorem 7.26 follows by the transitivity of bisimilarity. Finally, we conclude Theorem 7.23 from Theorem 7.26.

**Event Based Transition System for \( \text{EXP}_{\text{se}}^{\text{O}} \)**

The process algebra expressions \( \text{EXP}_{\text{se}}^{\text{O}} \) are defined by the following BNF-grammar.

\[
G ::= B \mid b^{-} \mid G + B \mid G \oplus M G \mid G ; B \mid G \parallel A, M G \mid G \backslash M A \mid G[(a \rightarrow B)^{a \in A}, (a \rightarrow \vec{G})^{a \in A}]_{M_{A}} \mid [G]_{q} \\
\vec{G} ::= \varepsilon \mid G \cdot \vec{G}
\]

where \( B \in \text{EXP}_{\text{se}}, b \in \text{Obs}, A \subseteq \text{Obs}, M : \text{Obs} \rightarrow \{l, r\}^{*}, M_{A} : \text{Obs} \rightarrow ((A \times \mathbb{N}) \cup \{0\})^{*} \) and \( q \in \{1, 2, l, r\} \). The event based transition rules \( \rightarrow^{c_{\text{decl}}} \) with respect to \( \text{decl} : \text{Var} \rightarrow \text{EXP}_{\text{se}} \) are presented in Table 7.8, Table 7.9 and Table 7.10.
7.8. PROOFS

<table>
<thead>
<tr>
<th>Rule</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Ac'_1$</td>
<td>$\frac{a \in \text{Obs}}{a \xrightarrow{a^+} a}$</td>
</tr>
<tr>
<td>$Ac'_2$</td>
<td>$\frac{b - b_1 \sqrt{\cdot}}{b_1 \sqrt{\cdot}}$</td>
</tr>
<tr>
<td>$Ac'_3$</td>
<td>$\frac{\tau - \tau \sqrt{\cdot}}{\tau}$</td>
</tr>
<tr>
<td>$Ch'_1$</td>
<td>$\frac{B_1 \xrightarrow{\gamma} G'_1}{B_1 + B_2 (r_1, \tau) [G'_1]_1}$</td>
</tr>
<tr>
<td>$Ch'_2$</td>
<td>$\frac{D_1 \xrightarrow{a^+} G'_1}{D_1</td>
</tr>
<tr>
<td>$Ch'_3$</td>
<td>$\frac{D_1 \xrightarrow{\gamma} G'_1}{D_1</td>
</tr>
<tr>
<td>$Ch'_4$</td>
<td>$\frac{G_1 \xrightarrow{\gamma \in \text{Obs}} G'_1}{G_1 \oplus_M G_2 (r_1, \tau) [G'_1]_1}$</td>
</tr>
<tr>
<td>$Ch'_5$</td>
<td>$\frac{G_1 \xrightarrow{\gamma \in \text{Obs}} G'_1}{G_1 \oplus_M G_2 (r_1, \tau) [G'_1]_1}$</td>
</tr>
<tr>
<td>$S'_1$</td>
<td>$\frac{D_1 \xrightarrow{\alpha} G'_1}{D_1; B_2 (\alpha, \tau) [G'_1]_1}$</td>
</tr>
<tr>
<td>$S'_2$</td>
<td>$\frac{D_1 \xrightarrow{\alpha \sqrt{\cdot}} G'_1}{D_1; B_2 (\alpha, \tau) [G'_1]_1}$</td>
</tr>
<tr>
<td>$Rec'$</td>
<td>$\frac{\text{decl}(x) \xrightarrow{\alpha} G'}{x \xrightarrow{\alpha} G'}$</td>
</tr>
<tr>
<td>$N'_1$</td>
<td>$\frac{G \xrightarrow{\gamma} G'}{[G]_i (r_1, \tau) [G']_i}$</td>
</tr>
<tr>
<td>$N'_2$</td>
<td>$\frac{G \xrightarrow{\gamma} G'}{[G]_i (r_1, \tau) [G']_i}$</td>
</tr>
<tr>
<td>$N'_3$</td>
<td>$\frac{G \xrightarrow{\gamma} G'}{[G]_i (r_1, \tau) [G']_i}$</td>
</tr>
</tbody>
</table>

Table 7.8: Event Based Transition Rules with respect to $\xrightarrow{\text{decl}} (1)$
\[ P_0': \frac{B_1 \parallel A \parallel B_2 \xrightarrow{\tau_e} G'}{B_1 \parallel A \parallel B_2 \xrightarrow{\tau_e} G'} \quad P_1': \frac{G_1 \xrightarrow{a^+_e} G'_1 \ a \notin A}{G_1 \parallel A,M G_2 \xrightarrow{a^+_e} G'_1 \parallel A,a,l \cdot M G_2} \quad G_2 \parallel A,M G_1 \xrightarrow{a^+_e} G'_2 \parallel A,a,r \cdot M G'_1} \\
\]
\[ P_2': \frac{G_1 \xrightarrow{a^-_e} G'_1 \ a \notin A}{G_1 \parallel A,M G_2 \xrightarrow{a^-_e} G'_1 \parallel A,M \backslash (a,M(a)(l,i)) G_2} \quad G_2 \parallel A,M G_1 \xrightarrow{a^-_e} G'_2 \parallel A,M \backslash (a,M(a)(r,i)) G'_1} \\
\]
\[ P_3': \frac{G_1 \xrightarrow{a^-_\sqrt{e}} G'_1 \ a \notin A}{G_1 \parallel A,M G_2 \xrightarrow{a^-_\sqrt{e}} G'_1 \parallel A,M \backslash (a,M(a)(l,i)) G_2} \quad G_2 \parallel A,M G_1 \xrightarrow{a^-_\sqrt{e}} G'_2 \parallel A,M \backslash (a,M(a)(r,i)) G'_1} \\
\]
\[ P_4': \frac{G_1 \xrightarrow{a^+_t} G'_1 \ G_2 \xrightarrow{a^+_t} G'_2 \ a \in A}{G_1 \parallel A,M G_2 \xrightarrow{a^+_t} G'_1 \parallel A,M G'_2} \quad P_5': \frac{G_1 \xrightarrow{a^-_t} G'_1 \ G_2 \xrightarrow{a^-_t} G'_2 \ a \in A}{G_1 \parallel A,M G_2 \xrightarrow{a^-_t} G'_1 \parallel A,M G'_2} \quad G_2 \parallel A,M G_1 \xrightarrow{a^-_t} G'_2 \parallel A,M G'_1} \\
\]
\[ P_6': \frac{G_1 \xrightarrow{a^-_t} G'_1 \ G_2 \xrightarrow{a^-_t} G'_2 \ a \in A}{G_1 \parallel A,M G_2 \xrightarrow{a^-_t} G'_1 \parallel A,M G'_2} \quad P_7': \frac{G_1 \xrightarrow{a^-_t} G'_1 \ G_2 \xrightarrow{a^-_t} G'_2 \ a \in A}{G_1 \parallel A,M G_2 \xrightarrow{a^-_t} G'_1 \parallel A,M G'_2} \quad G_2 \parallel A,M G_1 \xrightarrow{a^-_t} G'_2 \parallel A,M G'_1} \\
\]
\[ P_8': \frac{G_1 \xrightarrow{\tau e} G'_1}{G_1 \parallel A,M G_2 \xrightarrow{\tau e} G'_1 \parallel A,M G'_2} \quad P_9': \frac{G_1 \xrightarrow{\tau e} G'_1}{G_1 \parallel A,M G_2 \xrightarrow{\tau e} G'_1 \parallel A,M G'_2} \quad G_2 \parallel A,M G_1 \xrightarrow{\tau e} G'_2 \parallel A,M G'_1} \\
\]
\[ \text{Res}_0': \frac{B \parallel A \xrightarrow{\tau_e} G'}{B \parallel A \xrightarrow{\tau_e} G'} \quad \text{Res}_1': \frac{G \xrightarrow{a^+_e} G' \ a \notin A}{G \parallel M A \xrightarrow{a^+_e} G' \parallel [a,l] \cdot M A} \quad \text{Res}_2': \frac{G \xrightarrow{a^-_\sqrt{e}} G' \ a \notin A}{G \parallel M A \xrightarrow{a^-_\sqrt{e}} G' \parallel [a,l] \cdot M A} \quad \text{Res}_3': \frac{G \xrightarrow{\tau(t)} G'}{G \parallel M A \xrightarrow{\tau(t)} G' \parallel M A} \]

Table 7.9: Event Based Transition Rules with respect to \( \xrightarrow{e}_{\text{decl}} (2) \)
For simplicity, let $\phi = (a \rightarrow B_a)_{a \in A}$ and $\varphi = (a \rightarrow \vec{G}_a)_{a \in A}$

$$R_0' : \frac{B[\phi, (a \rightarrow \varepsilon)_{a \in A}]}{B[\phi]} \xrightarrow{\frac{\gamma}{\varepsilon}} G'$$

$$R_1' : \frac{G \xrightarrow{a^+} G'}{G[\phi, \varphi]_{M_A} \xrightarrow{a^+} G'[\phi, \varphi]_{[a,0], M_A}}$$

$$R_2' : \frac{G \xrightarrow{a^-} G', a \notin A, m = \overline{M_A}(a)(0, i)}{G[\phi, \varphi]_{M_A} \xrightarrow{a^-} G'[\phi, \varphi]_{[a, i \rightarrow M_A \setminus \{a, i\}]}(M_A \cup \varphi)}$$

$$R_3' : \frac{G \xrightarrow{\tau} G'}{G[\phi, \varphi]_{M_A} \xrightarrow{\tau} G'[\phi, \varphi]_{M_A}}$$

$$R_4' : \frac{G \xrightarrow{a^+} G', a \in A}{G[\phi, \varphi]_{M_A} \xrightarrow{a^+} G'[\phi, \varphi]_{[a \rightarrow G_a G_a]}}$$

$$R_5' : \frac{G \xrightarrow{\tau} G', a \in A}{G[\phi, \varphi]_{M_A} \xrightarrow{\tau} G'[\phi, \varphi]_{[a \rightarrow G_a G_a]}(M_A \cup \varphi)}$$

$$R_6' : \frac{G \xrightarrow{a^-} G', a \in A}{G[\phi, \varphi]_{M_A} \xrightarrow{a^-} G'[\phi, \varphi]_{[a \rightarrow G_a G_a]}}$$

$$R_7' : \frac{G \xrightarrow{a^-} G', a \in A}{G[\phi, \varphi]_{M_A} \xrightarrow{a^-} G'[\phi, \varphi]_{[a \rightarrow G_a G_a]}(M_A \cup \varphi)}$$

$$R_8' : \frac{G \xrightarrow{a^-} G', a \in A}{G[\phi, \varphi]_{M_A} \xrightarrow{a^-} G'[\phi, \varphi]_{[a \rightarrow G_a G_a]](M_A \cup \varphi)}}$$

$$R_9' : \frac{G \xrightarrow{a^-} G', a \in A}{G[\phi, \varphi]_{M_A} \xrightarrow{a^-} G'[\phi, \varphi]_{[a \rightarrow G_a G_a]}(M_A \cup \varphi)}$$

$$R_{10} : \frac{G \xrightarrow{a^-} G', a \in A}{G[\phi, \varphi]_{M_A} \xrightarrow{a^-} G'[\phi, \varphi]_{[a \rightarrow G_a G_a]}(M_A \cup \varphi)}}$$

$$R_{11} : \frac{G \xrightarrow{a^-} G', a \in A}{G[\phi, \varphi]_{M_A} \xrightarrow{a^-} G'[\phi, \varphi]_{[a \rightarrow G_a G_a]}(M_A \cup \varphi)}}$$

Table 7.10: Event Based Transition Rules with respect to $\xrightarrow{\gamma_{\text{decl}}}$ (3)
First Bisimulation Result.

An expression \( G \) of \( \text{EXP}_\text{se}' \) and an expression \( C \) of \( \text{EXP}_\text{se} \) are related if \( G \) results in \( C \) through the removal of all \([\cdot]\) expressions. This is formalized by the following function, where we also count the \([\cdot]\) symbols in \( G \).

**Definition 7.40** \( \Xi : \mathbb{N} \times \text{EXP}_\text{se} \rightarrow \mathcal{P}(\text{EXP}_\text{se}') \) is defined as follows, where \( Q = \{1, 2, l, r\} \).

\[
\Xi(0, C) = \{ C \}
\]

\[
\Xi(n + 1, B) = \{ [\tilde{G}]_i \mid i \in Q \land \tilde{G} \in \Xi(n, B) \}
\]

\[
\Xi(n + 1, C\uplus B) = \{ [\tilde{G}]_i \mid i \in Q \land \tilde{G} \in \Xi(n, C\uplus B) \} \cup \{ G \mid G \in \Xi(n + 1, C) \}
\]

\[
\Xi(n + 1, C\oplus M C_2) = \{ [\tilde{G}]_i \mid i \in Q \land \tilde{G} \in \Xi(n, C\oplus M C_2) \} \cup \\
\{ G_1 \oplus_M C_2 \mid \exists m \in \mathbb{N} : m \leq n + 1 \land G_1 \in \Xi(m, C_1) \land G_2 \in \Xi(n + 1 - m, C_2) \}
\]

\[
\Xi(n + 1, C ; B) = \{ [\tilde{G}]_i \mid i \in Q \land \tilde{G} \in \Xi(n, C ; B) \} \cup \{ G \mid G \in \Xi(n + 1, C) \}
\]

\[
\Xi(n + 1, C\setminus M A) = \{ [\tilde{G}]_i \mid i \in Q \land \tilde{G} \in \Xi(n, C\setminus M A) \} \cup \\
\{ G \setminus M A \mid G \in \Xi(n + 1, C) \}
\]

\[
\Xi(n + 1, C[(a \rightarrow B_a)^{a \in A}, (a \rightarrow \tilde{C}_a)^{a \in A}]_{M A}) = \{ G[(a \rightarrow B_a)^{a \in A}, (a \rightarrow \tilde{C}_a)^{a \in A}] \mid \\
\exists m \in \mathbb{N}, (m_1^a)^{a \in A} \in \mathbb{N}^A : m + \sum_{a \in A} m_1^a = n + 1 \land G \in \Xi(m, C) \land \\
\left( \tilde{C}_a[j] \in \Xi(m_1^a, \tilde{C}_a[j]) \lor (\tilde{C}_a[j] \text{ is undefined} \land \tilde{C}_a[j] \text{ is undefined}) \right) \} \cup \\
\{ [\tilde{G}]_i \mid i \in Q \land \tilde{G} \in \Xi(n, C[(a \rightarrow B_a)^{a \in A}, (a \rightarrow \tilde{C}_a)^{a \in A}]_{M A}) \}
\]

The well-definedness of \( \Xi \) is easily seen.

**Lemma 7.41** Let \( B \in \text{EXP}_\text{se} \) then \( (\text{EXP}_\text{se} ; \longrightarrow_{\text{decl}}^r, B) \) and \( (\text{EXP}_\text{se} ; \longrightarrow''_{\text{decl}}^r, B) \) are bisimilar, where \( G \xrightarrow{\gamma} G' \Leftrightarrow \exists e \in \mathcal{U} : G \xrightarrow{\gamma} G' \).

**Proof:** Define \( \mathcal{R} = \{ (C, G) \in \text{EXP}_\text{se} \times \text{EXP}_\text{se}' \mid \exists n : G \in \Xi(n, C) \} \). In order to verify that \( \mathcal{R} \) is a bisimulation, we show

\[
(C \xrightarrow{\gamma} C' \land G \in \Xi(n, C)) \Rightarrow \exists e, G', m : G \xrightarrow{\gamma} G' \land G' \in \Xi(m, C') \quad (7.2)
\]

The proof of (7.2) works by induction on the depth of inference of \( C \longrightarrow^c C' \) combined with the value of \( n \). Then, (7.2) can be easily checked with the following procedure:

- applying rule \( N_j \) whenever \( C = [\tilde{C}]_q \). In these cases \( n \) is reduced by one and \( C \longrightarrow^c C' \) keeps unaffected. Therefore the result follows by induction.

- applying the correspondent rules of \( C \longrightarrow^c C' \) whenever \( C \) is different to \([\tilde{C}]_q \). In these cases the depth of inference is reduced and \( n \) gets not increased. Therefore the result follows by induction.
Another fact is
\[(G \xrightarrow{e} G' \land G \in \Xi(n, C')) \Rightarrow \exists C', m : C' \xrightarrow{e} C' \land G' \in \Xi(m, C') \quad (7.3)\]

This equation can be seen by induction on the depth of inference of $G \xrightarrow{e} G'$.
Now we are ready to verify that $\mathcal{R}$ is a bisimulation:

- It is clear that $(B, B) \in \mathcal{R}$.
- Suppose $(C_1, G_1) \in \mathcal{R}$ and $C_1 \xrightarrow{e} C_2$. Then $\exists e, G_2, m : G_1 \xrightarrow{e} G_2 \land G_2 \in \Xi(m, C_2)$ by (7.2). Thus $G_1 \xrightarrow{e} G_2$ and $(C_2, G_2) \in \mathcal{R}$, as required.
- Suppose $(C_1, G_1) \in \mathcal{R}$ and $G_1 \xrightarrow{e} G_2$. Then $G_1 \xrightarrow{e} G_2$ for some $e$. Hence, $\exists C_2, m : C_1 \xrightarrow{e} C_2 \land G_2 \in \Xi(m, C_2)$ by (7.3).

**Second Bisimulation Result.**

First, we show that the denotation of a variable is the same as the denotation of its corresponding expression.

**Lemma 7.42** Let $\text{decl} : \text{Var} \to \text{EXP}_{se}$ and $x \in \text{Var}$. Then $\llbracket \text{decl}, x \rrbracket = \llbracket \text{decl}, \text{decl}(x) \rrbracket$.

**Proof:** Similar to the proof of Lemma 3.37. \qed

We extend the denotational semantics to $\text{EXP}^{O'}_{se}$. We define an event based refinement operator, which refines events rather than actions.

**Definition 7.43** Define $\text{Ref}_{se}^{\uparrow} : \text{SEBES} \times (\mathcal{U} \to \text{SEBES}) \to \text{SEBES}$ by $\text{Ref}_{se}^{\uparrow}(\mathcal{E}, \vartheta) = (\hat{E}, \hat{\sim}, \hat{\leadsto}, \hat{T}, \hat{l})$ where

\[
\hat{E} = \{(e, \hat{e}) \mid e \in E \land l(e) \in A \land \hat{e} \in E_{\vartheta(e)}\} \cup
\{(e, e) \in E \times E \mid l(e) \notin A\}
\]

\[
\hat{\sim} = \{(e, e) \mid e \leadsto e' \lor (e' = e' \land l(e) \in A \land \hat{e} \sim_{\vartheta(e)} \hat{e}')\}
\]

\[
\hat{\leadsto} = \{(e', \hat{e}') \in \hat{E} \mid e' \in E \land \hat{e}' \in E_{\vartheta(e')}\}
\]

\[
\hat{T} = \{X \mid \exists Y \in T \land \exists f : X \to \mathcal{P}(U) \land \forall e' \in \hat{E} \land l(e') \in A \land f(e') = \{e'\} \lor (l(e') \in A \land e' \neq e \land f(e') \in \vartheta(e')\}
\]

\[
l(e, \hat{e}) = \begin{cases}
\hat{e} & \text{if } l(e) \notin A \\
l_{\vartheta(e)}(\hat{e}) & \text{if } l(e) \in A
\end{cases}
\]
For the operators on $\mathsf{EXP}^O_{se}$ we need the following function, which adapts function $\mathcal{M}$ when the events are renamed.

Define $\Psi : (\mathcal{U} \rightarrow \mathbb{N}) \times \{1, 2, l, r\} \rightarrow (\mathcal{U} \rightarrow \mathbb{N})$ by

$$
\Psi(\mathcal{M}, q)(e) = \begin{cases} 
\mathcal{M}(e') & \text{if } q \in \{1, 2\} \land e = (q, e') \\
\mathcal{M}(e') & \text{if } q = l \land e = (*, e') \\
\mathcal{M}(e') & \text{if } q = r \land e = (e', *)
\end{cases}
$$

The operators on $\mathsf{EXP}^O_{se}$ are given as follows:

**Definition 7.44 (Operators on $\mathsf{EXP}^O_{se}$)** Let $A \subseteq \mathsf{Obs}$, $\mathcal{M} : \mathsf{Obs} \rightarrow \{l, r\}^*$, $\mathcal{M}_A : \mathsf{Obs} \rightarrow ((\mathcal{U} \times \mathbb{N}) \cup \{0\})^*$ and $q \in \{1, 2, l, r\}$ Furthermore, suppose $\mathcal{E} = (\mathcal{E}, \mathcal{M})$ and $\mathcal{E}_j = (\mathcal{E}_j, \mathcal{M}_j)$. Then define

$$
\oplus : \mathsf{SEBES}_M \times \mathsf{SEBES} \rightarrow \mathsf{SEBES}_M \text{ with } \mathcal{E}_1 \oplus \mathcal{E}_2 = (\mathcal{E}_1|\mathcal{E}_2, \Psi(\mathcal{M}_1, 1))
$$

$$
\ominus : \mathsf{SEBES}_M \times \mathsf{SEBES}_M \rightarrow \mathsf{SEBES}_M \text{ with } \mathcal{E}_1 \ominus \mathcal{E}_2 = (\mathcal{E}_1|\mathcal{E}_2, \Psi(\mathcal{M}_1, 1))
$$

$$
\|_{A,M} : \mathsf{SEBES}_M \times \mathsf{SEBES}_M \rightarrow \mathsf{SEBES}_M \text{ with } \mathcal{E}_1\|_{A,M}\mathcal{E}_2 = (\mathcal{E}_1|\mathcal{E}_2, \mathcal{E}', \mathcal{M}')
$$

$$
\forall_{A,M} : \mathsf{SEBES}_M \rightarrow \mathsf{SEBES}_M \text{ with } \mathcal{E} \forall_{A,M} \mathcal{E} = \mathcal{E} \forall A \rightarrow \mathcal{E}' \rightarrow \mathcal{M}' \forall E'
$$

$$
\overline{\mathcal{M}}^*_{A} : \mathsf{SEBES}_M \rightarrow \mathsf{SEBES}_M \text{ with } \mathcal{E} \overline{\mathcal{M}}_{A} \mathcal{E} = \mathcal{E} \overline{\mathcal{M}} A \rightarrow \mathcal{E}' \rightarrow \mathcal{M}' \forall E'
$$

$$
\overline{\text{Ref}}_{A,M}^{sc} : \mathsf{SEBES}_M \times (A \rightarrow \mathsf{SEBES}) \times (A \rightarrow \mathsf{SEBES}_M^*) \rightarrow \mathsf{SEBES}_M
$$

with $\overline{\text{Ref}}_{A,M}^{sc}(\mathcal{E}, \theta, \tilde{\theta}) = (\overline{\text{Ref}}^{sc}(\mathcal{E}, \vartheta) \rightarrow E', \mathcal{M}')$

where $\vartheta(e) = \begin{cases} 
\pi_1(\vartheta(e)([\mathcal{M}])) & \text{if } l(e) \in A \land e \notin \text{dom}(\mathcal{M}) \\
(0, 0, 0, 0, \{0\}, 0) & \text{otherwise}
\end{cases}$

and $E' = \{(e, e) \mid l(e) \notin A \land (e \in \text{dom}(\mathcal{M}) \Rightarrow M_A(l(e))(0, \mathcal{M}(e)) \text{ is defined}) \cup \{(e, \hat{e}) \mid \hat{e} \in \text{dom}(\mathcal{M}^\prime) \Rightarrow M_A(l(e))(l(e), \mathcal{M}(e), \mathcal{M}'(\hat{e})) \text{ is defined})\}$.
Lemma 7.46

Suppose \( E_1, E_2 \in \text{SEBES} \) and \( \vartheta : U \to \text{SEBES} \). Then

\[
\begin{align*}
\langle E_1 \oplus E_2 \rangle_{[\ast, \ast]} & \approx \overbrace{\text{Shift}_l(E_{[\ast]}E_{[\ast]})} \\
\langle E_1 \oplus E_2 \rangle_{[[\ast, \ast]]} & \approx \overbrace{\text{Shift}_l(E_{[[\ast]]}E_{[[\ast]]})} \\
\langle E_1 \circ E_2 \rangle_{[[\ast, \ast]]} & \approx \overbrace{\text{Shift}_l(E_{[[\ast]]}E_{[[\ast]]})} \\
\langle E_1 \mid A \rangle_{[e_1, \ast]} & \approx \begin{cases} 
\langle E_1 \rangle_{[e_1]} \mid A \langle E_2 \rangle & \text{if } e_1 \in \text{init}(E_1) \land \top(T_1, e_1) \\
\overbrace{\text{Shift}_r(E_{[e_1]}E_{[\ast]}) \mid A} & \text{otherwise}
\end{cases} \\
\langle E_1 \mid A \rangle_{[[e_1, \ast]]} & \approx \begin{cases} 
\langle E_1 \rangle_{[[e_1]]} \mid A \langle E_2 \rangle_{[e_2]} & \text{if } e_2 \in \text{init}(E_2) \land \bot(T_1, e_1) \land l_1(e_1) \notin A \\
\overbrace{\text{Shift}_r(E_{[[e_1]]}E_{[e_2]}) \mid A} & \text{if } e_1 \in \text{init}(E_1) \land \bot(T_1, e_1) \land l_1(e_1) \notin A
\end{cases} \\
\langle E_1 \mid A \rangle_{[[e_1, e_2]]} & \approx \begin{cases} 
\langle E_1 \rangle_{[[e_1]]} \mid A \langle E_2 \rangle_{[e_2]} & \text{if } e_2 \in \text{init}(E_2) \land \top(T_2, e_2) \land l_2(e_2) \notin A \\
\overbrace{\text{Shift}_r(E_{[[e_1]]}E_{[e_2]}) \mid A} & \text{if } e_1 \in \text{init}(E_1) \land \top(T_1, e_1) \land l_1(e_1) \notin A
\end{cases} \\
\langle E_1 \mid A \rangle_{[[e_1, e_2]]} & \approx \begin{cases} 
\langle E_1 \rangle_{[[e_1]]} \mid A \langle E_2 \rangle_{[e_2]} & \text{if } e_2 \in \text{init}(E_2) \land \top(T_2, e_2) \land l_2(e_2) \notin A \\
\overbrace{\text{Shift}_r(E_{[[e_1]]}E_{[e_2]}) \mid A} & \text{if } e_1 \in \text{init}(E_1) \land \top(T_1, e_1) \land l_1(e_1) \notin A
\end{cases}
\end{align*}
\]
whenever \( l_1(e_1) = l_2(e_2) \in A \)

\[
(\mathcal{E} \backslash \backslash A)[e] \approx \begin{cases} 
\mathcal{E}[e] \backslash \backslash A & \text{if } l(e) \notin A \\
\text{undefined} & \text{otherwise}
\end{cases}
\]

\[
\text{Ref}^\text{se}_A (\mathcal{E}, \vartheta)[(e, \hat{e})] \approx \begin{cases} 
\text{Ref}^\text{se}_A (\mathcal{E}[e], \vartheta) & \text{if } l(e) \notin A \land e = \hat{e} \\
\text{Ref}^\text{se}_A (\mathcal{E}[e], \vartheta[e \rightarrow \vartheta(e)[e]]) & \text{if } l(e) \in A \land \Upsilon(T, \vartheta(e), \hat{e}) \\
\text{Ref}^\text{se}_A (\mathcal{E}(e), \vartheta[e \rightarrow \vartheta(e)[e]]) & \text{if } l(e) \in A \land \neg \Upsilon(T, \vartheta(e), \hat{e})
\end{cases}
\]

\[
\text{Shift}_i(\mathcal{E})[(\ast, e)] \approx \text{Shift}_i(\mathcal{E}[e]) \quad \text{whenever } i \in \{1, 2\}
\]

\[
\text{Shift}_r(\mathcal{E})[(\ast, e)] \approx \text{Shift}_r(\mathcal{E}[e])
\]

\[
\text{Shift}_t(\mathcal{E})[(e, \ast)] \approx \text{Shift}_t(\mathcal{E}[e])
\]

and \( \text{Ref}^\text{se}_A (\mathcal{E}[e], \vartheta[e \rightarrow \mathcal{E}']) \approx \text{Ref}^\text{se}_A (\mathcal{E}[e], \vartheta) \) holds for any \( \mathcal{E}' \in \text{SEBES} \).

Furthermore

\[
(\mathcal{E}_1 \oplus \mathcal{E}_2)[(\ast, e)] \approx \text{Shift}_i(\mathcal{E}_i[e])
\]

\[
(\mathcal{E}_1 \oplus \mathcal{E}_2)[(\ast, e)] \approx \begin{cases} 
\mathcal{E}_1(e) \oplus \mathcal{E}_2 & \text{if } i = 1 \\
\text{Shift}_2(\mathcal{E}_2[e]) & \text{if } i = 2
\end{cases}
\]

\[
(\mathcal{E}_1 \oplus \mathcal{E}_2)[(\ast, e)] \approx \begin{cases} 
\mathcal{E}_1(e) \oplus \mathcal{E}_2 & \text{if } i = 1 \\
\text{Shift}_2(\mathcal{E}_2[e]) & \text{if } i = 2
\end{cases}
\]

\[
(\mathcal{E}_1 \downarrow A \mathcal{E}_2)[(e_1, e_2)] \approx \begin{cases} 
\mathcal{E}_1(e_1) \downarrow A \mathcal{E}_2 & \text{if } l_1(e_1) \notin A \land e_2 = \ast \\
\mathcal{E}_1(e_1) \downarrow A \mathcal{E}_2 & \text{if } l_2(e_2) \notin A \land e_1 = \ast \\
(\mathcal{E}_1(e_1) \downarrow A \mathcal{E}_2)[e'] & \text{if } l_1(e_1) = l_2(e_2) \in A \land e' = \{(e_1', e_2') \mid (e_1', e_2') \neq (e_1, e_2) \lor (e_1', e_2') = (e_1, e_2)\}
\end{cases}
\]

\[
(\mathcal{E} \backslash \backslash A)[e] \approx \begin{cases} 
\mathcal{E}[e] \backslash \backslash A & \text{if } l(e) \notin A \\
\text{undefined} & \text{otherwise}
\end{cases}
\]

\[
\text{Ref}^\text{se}_A (\mathcal{E}, \vartheta)[(e, \hat{e})] \approx \begin{cases} 
\text{Ref}^\text{se}_A (\mathcal{E}[e], \vartheta) & \text{if } l(e) \notin A \land e = \hat{e} \\
\text{Ref}^\text{se}_A (\mathcal{E}[e], \vartheta[e \rightarrow \vartheta(e)[e]]) & \text{if } l(e) \in A \land \Upsilon(T, \vartheta(e), \hat{e}) \\
\text{Ref}^\text{se}_A (\mathcal{E}(e), \vartheta[e \rightarrow \vartheta(e)[e]]) & \text{if } l(e) \in A \land \neg \Upsilon(T, \vartheta(e), \hat{e})
\end{cases}
\]

\[
\text{Shift}_i(\mathcal{E})[(\ast, e)] \approx \text{Shift}_i(\mathcal{E}[e]) \quad \text{whenever } i \in \{1, 2\}
\]

\[
\text{Shift}_r(\mathcal{E})[(\ast, e)] \approx \text{Shift}_r(\mathcal{E}[e])
\]

\[
\text{Shift}_t(\mathcal{E})[(e, \ast)] \approx \text{Shift}_t(\mathcal{E}[e])
\]

Moreover,

\[
\Upsilon(T, \mathcal{E}_1 \oplus \mathcal{E}_2)[(\ast, e)] \Leftrightarrow \Upsilon(T, e)
\]

\[
\Upsilon(T, \mathcal{E}_1 \oplus \mathcal{E}_2)[(\ast, e)] \Leftrightarrow \Upsilon(T, e)
\]

\[
\Upsilon(T, \mathcal{E}_1 \oplus \mathcal{E}_2)[(\ast, e)] \Leftrightarrow \Upsilon(T, e)
\]

\[
\Upsilon(T, \mathcal{E}_1 \oplus \mathcal{E}_2)[(\ast, e)] \Leftrightarrow \Upsilon(T, e) \land i = 2
\]

\[
\Upsilon(T, \mathcal{E}_1 \downarrow A \mathcal{E}_2)[(e_1, e_2)] \Leftrightarrow \big(\Upsilon(T_1, e_1) \land \Upsilon(T_2, e_2) \land l_1(e_1) = l_2(e_2) \in A\big)
\]

\[
\Upsilon(T, \mathcal{E} \backslash \backslash A[e]) \Leftrightarrow \big(\Upsilon(T, e) \land l(e) \notin A\big)
\]
\[
\begin{align*}
\forall (T_{\text{Ref}^*_{A}}(\varepsilon, \vartheta), (e, \hat{e})) & \iff \left\{ \begin{array}{ll}
\Upsilon(T, e) & \text{if } l(e) \notin A \land e = \hat{e} \\
\Upsilon(T, e) \land \Upsilon(T_{\vartheta(e)}, \hat{e}) & \text{if } l(e) \in A \\
\end{array} \right. \\
\forall (T_{\text{Shift}^i}(\varepsilon), (\star_i, e)) & \iff \Upsilon(T, e) \text{ whenever } i \in \{1, 2\} \\
\forall (T_{\text{Shift}^r}(\varepsilon), (\star, e)) & \iff \Upsilon(T, e)
\end{align*}
\]

**Proof:** Straightforward. \(\Box\)

Now we are ready to show how the remainder on \text{SEBES}_{\mathcal{M}}\ defined in Definition 7.25 is determined with respect to the different operators of Definition 7.44.

**Lemma 7.47** Let \(A \subseteq \text{Obs}, \; M : \text{Obs} \to \{l, r\}^*, \; M_A : \text{Obs} \to ((A \times \mathbb{N}) \cup \{0\})^*\) and \(q \in \{1, 2, l, r\}\). Furthermore, suppose \(\varepsilon \equiv (\varepsilon, \mathcal{M})\) and \(\varepsilon_j \equiv (\varepsilon_j, \mathcal{M}_j)\). Then

\[
(\varepsilon_1 \vdash \varepsilon_2)_{(\star_i, e)} \simeq \left\{ \begin{array}{ll}
\varepsilon_{1e_i} \vdash \varepsilon_2 & \text{if } i = 1 \land e \notin \text{dom}(\mathcal{M}_1) \land l(e) \in \text{Obs} \\
\text{Shift}_1(\varepsilon_{1e_i}) \text{\small \bigl(\text{\small Shift}_1(\varepsilon_{1e_i}) \text{\small Shift}_2((\varepsilon_2, \perp)_{e_i}) \bigr)} & \text{if } i = 1 \land (e \in \text{dom}(\mathcal{M}_1) \lor l(e) = \tau) \\
\text{Shift}_2(\varepsilon_{1e_i}) \text{\small \bigl(\text{\small Shift}_2(\varepsilon_{1e_i}) \text{\small Shift}_1((\varepsilon_1, \perp)_{e_i}) \bigr)} & \text{if } i = 2
\end{array} \right.
\]

Let \(\varepsilon_c \equiv (\varepsilon_1 \vdash_{\mathcal{M}} \varepsilon_2)_{(\star_i, e)}\) then

\[
\varepsilon_c \simeq \left\{ \begin{array}{ll}
\varepsilon_{1e_i} \vdash [l_1(e), l_2(e), l_3(e)]_{\mathcal{M}} \varepsilon_2 & \text{if } i = 1 \land l_1(e) \in \text{Obs} \land e \notin \text{dom}(\mathcal{M}_1) \\
\varepsilon_{1e_i} \vdash [l_1(e), l_2(e), l_3(e)]_{\mathcal{M}} \varepsilon_2 & \text{if } i = 2 \land l_2(e) \in \text{Obs} \land e \notin \text{dom}(\mathcal{M}_2) \\
\text{Shift}_1(\varepsilon_{1e_i}) \text{\small \bigl(\text{\small Shift}_1(\varepsilon_{1e_i}) \text{\small Shift}_2((\varepsilon_2, \perp)_{e_i}) \bigr)} & \text{if } i = 1 \land l_1(e) \in \text{Obs} \land m = M(l_1(e))(l, \mathcal{M}_1(e)) \\
\text{Shift}_2(\varepsilon_{1e_i}) \text{\small \bigl(\text{\small Shift}_2(\varepsilon_{1e_i}) \text{\small Shift}_1((\varepsilon_1, \perp)_{e_i}) \bigr)} & \text{if } i = 1 \land l_1(e) = \tau \\
\text{Shift}_2(\varepsilon_{1e_i}) \text{\small \bigl(\text{\small Shift}_2(\varepsilon_{1e_i}) \text{\small Shift}_1((\varepsilon_1, \perp)_{e_i}) \bigr)} & \text{if } i = 2 \land l_2(e) \in \text{Obs} \land m = M(l_2(e))(r, \mathcal{M}_2(e)) \\
\text{Shift}_1(\varepsilon_{1e_i}) \text{\small \bigl(\text{\small Shift}_1(\varepsilon_{1e_i}) \text{\small Shift}_2((\varepsilon_2, \perp)_{e_i}) \bigr)} & \text{if } i = 2 \land l_2(e) = \tau \\
\end{array} \right.
\]

Let \(\varepsilon_p \equiv (\varepsilon_1 \vdash_{\mathcal{M}} \varepsilon_2)_{(e_1, e_2)}\) then

\[
\varepsilon_p \simeq \left\{ \begin{array}{ll}
\varepsilon_{1e_i} \vdash [l_1(e), l_2(e), l_3(e)]_{\mathcal{M}} \varepsilon_2 & \text{if } l_1(e) \in \text{Obs} \land e \notin \text{dom}(\mathcal{M}_1) \\
\varepsilon_{1e_i} \vdash [l_1(e), l_2(e), l_3(e)]_{\mathcal{M}} \varepsilon_2 & \text{if } l_1(e) \in \text{Obs} \land \neg \Upsilon(T_1, e) \land e \in \text{dom}(\mathcal{M}_1) \\
\text{Shift}_r(\varepsilon_2) \text{\small \bigl(\text{\small Shift}_r(\varepsilon_2) \text{\small Shift}_1((\varepsilon_1, \perp)_{e_i}) \bigr)} & \text{if } l_1(e) = \tau \land \neg \Upsilon(T_1, e) \\
\text{Shift}_r(\varepsilon_2) \text{\small \bigl(\text{\small Shift}_r(\varepsilon_2) \text{\small Shift}_1((\varepsilon_1, \perp)_{e_i}) \bigr)} A & \text{if } e \in \text{init}_{\text{Obs}}(\varepsilon_1) \land \Upsilon(T_1, e) \land e \in \text{dom}(\mathcal{M}_1) \\
\text{Shift}_r(\varepsilon_2) \text{\small \bigl(\text{\small Shift}_r(\varepsilon_2) \text{\small Shift}_1((\varepsilon_1, \perp)_{e_i}) \bigr)} A & \text{if } e \in \text{init}_{\tau}(\varepsilon_1) \land \Upsilon(T_1, e)
\end{array} \right.
\]

whenever \(e_1 = e \land e_2 = \star\).
whenever $e_1 = \ast \land e_2 = e$

$\forall l(e) \in A (\text{please note that } M_1(e_1) = M_2(e_2) \Rightarrow e_i \in \text{dom}(M_i))$

Let $\forall l(e) \in A \land e \notin \text{dom}(M)$

whenever $l(e) \notin A$

whenever $l(e) \in \text{Obs} \land e = e \land e \notin \text{dom}(M)$
The following lemma states that every transition of $\rightarrow'$ is matched by $\rightarrow^c$.

**Lemma 7.48** Suppose $G \in \text{EXP}_{se}'$ and $\text{decl} : \text{Var} \rightarrow \text{EXP}_{se}$. Then for all $G' \in \text{EXP}_{se}'$ and $\gamma \in L_{se}$ we have

$$G \xrightarrow{\text{decl}} G' \Rightarrow ([\text{decl}, G')]' \xrightarrow{\gamma} ((\text{decl}, G'))' \wedge ([\text{decl}, G'])' = [[(\text{decl}, G)]]'_{\gamma(e)}$$

**Proof:** It follows by induction on the depth of inferences of $G \xrightarrow{\text{decl}} G'$, where Lemma 7.46 and Lemma 7.47 are used. Furthermore, in the case of $\text{Rec}$ also Lemma 7.42 is applied.

Every started action can be immediately finished in $\rightarrow'_{\text{decl}}$:

**Lemma 7.49** Suppose $G, G' \in \text{EXP}_{se}'$, $\text{decl} : \text{Var} \rightarrow \text{EXP}_{se}$, $a \in \text{Obs}$ and $e \in \mathcal{U}$. Then

$$G \xrightarrow{a, e} G' \Rightarrow \exists G'' : (G' \xrightarrow{a + \sqrt{e}, e} G'') \vee G' \xrightarrow{a - \sqrt{e}, e} G''$$

**Proof:** It follows straightforwardly by induction on the depths of inferences of $G \xrightarrow{a, e} G'$ and is hence omitted.

The following lemma states that every transition of $\rightarrow^c$ is matched by $\rightarrow'$.

**Lemma 7.50** Suppose $G \in \text{EXP}_{se}'$, $\text{decl} : \text{Var} \rightarrow \text{EXP}_{se}$, $(\mathcal{E}, M) = [[(\text{decl}, G)]]'$ and $e \in \text{init}(\mathcal{E})$. Then there exists $G' \in \text{EXP}_{se}'$ and $\gamma \in L_{se}$ such that

$$G \xrightarrow{\text{decl}} G' \wedge \gamma \in \begin{cases} \{l(e) \rightarrow M(e), l(e) \rightarrow M(e)\} & \text{if } l(e) \in \text{Obs} \wedge e \in \text{dom}(M) \\ \{l(e)\} & \text{if } l(e) \in \text{Obs} \wedge e \notin \text{dom}(M) \\ \{\tau, \tau\sqrt{e}\} & \text{if } l(e) = \tau \end{cases}$$
Proof: First we show for any decl : Var → EXP_{se}:

$$\forall n \in \mathbb{N} : \forall B \in \text{EXP}_{se} : \forall \mathcal{E} : (\mathcal{E} = \{B\}^{n}_{\text{decl}}(\perp) \wedge e \in \text{init}(\mathcal{E})) \Rightarrow \exists G', \gamma : (B \xrightarrow{\gamma} \text{decl} G' \wedge (l(e) = \tau \Rightarrow \gamma \in \{\tau, \sqrt{\gamma}\}) \wedge (l(e) \in \text{Obs} \Rightarrow \gamma = l(e)+))$$ (7.4)

This is done by induction on n combined with the structure of B where the lexicographical order is used. We only present case B = x in detail: $e \in \text{init}([x]^{n}_{\text{decl}}(\perp))$ implies that $n > 0$. Therefore, $[x]^{n}_{\text{decl}}(\perp) = [\text{decl}(x)]^{n-1}_{\text{decl}}(\perp)$. The rest follows by induction, since n is reduced. In the case of the refinement operator, Lemma 7.49 is applied. Thus (7.4) is established.

An immediate consequence of (7.4) is

$$\forall B \in \text{EXP}_{se} : \forall \mathcal{E} : (\mathcal{E} = \{\langle \text{decl}, B \rangle \} \wedge e \in \text{init}(\mathcal{E})) \Rightarrow \exists G', \gamma : (B \xrightarrow{\gamma} \text{decl} G' \wedge (l(e) = \tau \Rightarrow \gamma \in \{\tau, \sqrt{\gamma}\}) \wedge (l(e) \in \text{Obs} \Rightarrow \gamma = l(e)+))$$ (7.5)

The main statement follows now by structural induction on G, where (7.5) is used. In the case of the refinement operator, Lemma 7.49 is applied. \qed

Now we are ready to obtain the second bisimulation result, which establishes Theorem 7.26.

Lemma 7.51 Let $\langle \text{decl}, B \rangle \in \text{PA}_{se}$, then the transition system $(\text{EXP}_{se}^{O}, L_{se}, \rightarrow^{\prime})$, where $\rightarrow^{\prime}$ is defined as in Lemma 7.41, is bisimilar to $(\text{SEBES}_{M}, L_{se}, \leftrightarrow, [\langle \text{decl}, B \rangle]_{\prime})$, where $\rightarrow^{\prime}$ is defined as in Lemma 7.41.

Proof: Define $\mathcal{R} = \{(G, [\langle \text{decl}, G \rangle]_{\prime}) | G \in \text{EXP}_{se}^{O}\}$. Then $(B, [\langle \text{decl}, B \rangle]_{\prime}) \in \mathcal{R}$ by definition.

Suppose $G_1 \in \text{EXP}_{se}^{O}$ and $G_1 \xrightarrow{\gamma^{\prime}} G_2$. Then $G_1 \xrightarrow{\gamma^{\prime}} G_2$ for some $e$. Hence, by Lemma 7.48 we get $\langle \text{decl}, G_1, G_2 \rangle$ as required.

Suppose $G_1 \in \text{EXP}_{se}^{O}$ and $\langle \text{decl}, G_1 \rangle$ $\xrightarrow{\gamma} \mathcal{E}_2$. Then there is $e \in \text{init}(\langle \text{decl}, G_1 \rangle)$ such that $\mathcal{E}_2 = [\langle \text{decl}, G_1 \rangle]_{e(\ell)}$ and $\langle \text{decl}, G_1 \rangle \xrightarrow{\gamma} [\langle \text{decl}, G_1 \rangle]_{e(\ell)}$. From Lemma 7.50 we get the existence of $G_2 \in \text{EXP}_{se}^{O}$ and $\gamma'$ such that $G_1 \xrightarrow{\gamma'} G_2$. Moreover, $\langle \text{decl}, G_1 \rangle_{e(\ell)} = [\langle \text{decl}, G_2 \rangle]_{e(\ell)}$ and $\gamma = \gamma'$ by Lemma 7.48, which concludes the proof. \qed

Final conclusions.

The only proof left is that of Theorem 7.23. Before we do this, we introduce the following lemma, which shows a correspondence between $\leftrightarrow$ and $\leftrightarrow^{c}$.

Lemma 7.52 For all $a \in \text{Act}$, $\mathcal{E}_1 \in \text{SEBES}$ and $(\mathcal{E}_2, M_2) \in \text{SEBES}_{M}$ we have

$$(M_2 = \perp \wedge \mathcal{E}_1 \xrightarrow{a^{(\vee)}} \mathcal{E}_2) \Leftrightarrow \begin{cases} (\mathcal{E}_1, \perp) \xrightarrow{a^{+}} (\mathcal{E}_2, M_2) & \text{if } a \in \text{Obs} \\ (\mathcal{E}_1, \perp) \xrightarrow{\tau^{(\vee)}} (\mathcal{E}_2, M_2) & \text{if } a = \tau \end{cases}$$
7.8. PROOFS

**Proof:** It can be easily verified, since $\mathcal{E}_{(e)[e]} = \mathcal{E}_{[e]}$. □

Now we are ready to verify Theorem 7.23.

**Proof of Theorem 7.23:** From Lemma 7.41, Lemma 7.51 and from the transitivity of bisimilarity it follows that $(\text{EXP}_se, \mathcal{L}_se, \sim_{\text{decl}}^e, B)$ and $(\text{SEBES}_M, \mathcal{L}_se, \sim^e, [\{(\text{decl}, B)\}]^e)$ are bisimilar (Theorem 7.26). Let $\mathcal{R}$ be such a corresponding bisimulation.

Define $\mathcal{R}$ by $\mathcal{R} = \{(C, \mathcal{E}) \in \text{EXP}_se \times \text{SEBES} \mid (C, (\mathcal{E}, \perp)) \in \tilde{\mathcal{R})}$. Then $(B, [\{(\text{decl}, B)\}]) \in \mathcal{R}$ by definition.

Suppose $(C_1, \mathcal{E}_1) \in \mathcal{R}$ and $C_1 \xrightarrow{a(\triangledown)} C_2$. We assume that $a \in \text{Obs}$ (the case when $a = \tau$ follows analogously). By definition $C_1 \xrightarrow{\tau}^e \xrightarrow{\tau}^e C_2$. Then there is $(\mathcal{E}_2, M_2)$ such that $(\mathcal{E}_1, \perp) \xrightarrow{\tau}^e \xrightarrow{\tau}^e (\mathcal{E}_2, M_2) \seesim (C_2, (\mathcal{E}_2, M_2)) \in \tilde{\mathcal{R}}$. Thus, $\mathcal{E}_1 \xrightarrow{a(\triangledown)} \mathcal{E}_2 \seesim (C_2, (\mathcal{E}_2, \perp)) \in \tilde{\mathcal{R}}$ by Lemma 7.52.

Suppose $(C_1, \mathcal{E}_1) \in \mathcal{R}$ and $\mathcal{E}_1 \xrightarrow{a(\triangledown)} \mathcal{E}_2$. We assume that $a \in \text{Obs}$ (the case when $a = \tau$ follows analogously). From Lemma 7.52 we obtain that $(\mathcal{E}_1, \perp) \xrightarrow{\tau}^e \xrightarrow{\tau}^e (\mathcal{E}_2, \perp)$. Then there is $C_2$ such that $C_1 \xrightarrow{a(\triangledown)} C_2 \seesim (C_2, (\mathcal{E}_2, \perp)) \in \tilde{\mathcal{R}}$. Hence, $C_1 \xrightarrow{a(\triangledown)} C_2$ by definition. □

### 7.8.2 Proof of the Congruence Results

**Proof of Theorem 7.28:** Let $\mathcal{R}$ be a bisimulation such that $((\mathcal{E}, \perp), (\mathcal{E}', \perp)) \in \mathcal{R}$ and let $\mathcal{R}_a$ be bisimulations such that $((\theta(a), \perp), (\theta(a), \perp)) \in \mathcal{R}_a$. Then define

\[\mathcal{R}_{\text{Ref}} = \{(\text{Ref}_{A,M}^{se}(\mathcal{E}, \theta, \bar{\theta}), \text{Ref}_{A,M}^{se}(\mathcal{E}', \theta, \bar{\theta}')) \mid (\mathcal{E}, \mathcal{E}') \in \mathcal{R} \wedge \forall a \in A : \forall i \in \mathbb{N}^+ : (\theta(a)[i] \text{ is defined } \iff \bar{\theta}(a)[i] \text{ is defined}) \wedge (\theta(a)[i] \text{ is defined } \implies (\bar{\theta}(a)[i], \bar{\theta}(a)[i] \in \mathcal{R}_a))\} \]

It is clear that $(\text{Ref}_{A}^{se}(\mathcal{E}, \theta), \perp) = \text{Ref}_{A,M}^{se}(\mathcal{E}, \theta, \perp)$, and therefore we obtain the fact that $((\text{Ref}_{A}^{se}(\mathcal{E}, \theta), \perp), (\text{Ref}_{A}^{se}(\mathcal{E}, \theta), \perp)) \in \mathcal{R}_{\text{Ref}}$ as required.

The verification that $\text{Rel}_{\text{Ref}}$ is a bisimulation is a straightforward consequence of Lemma 7.46 and Lemma 7.47 and is omitted here.

The proof of the other operators is straightforward. □

**Proof of Theorem 7.29:** The idea of this proof is the same as in Lemma 6.11:

Define $A \subseteq \mathcal{A}$ to be the set of all action-names occurring in $\mathcal{E}$ or in $\mathcal{E}'$, i.e. $A = \{l(e) \mid e \in E \} \cup \{l'(e') \mid e' \in E'\}$. Let $\mu : \{1, 2\} \times A \times \mathbb{N} \rightarrow \mathcal{A} \setminus A$ be an injective function. Such a function exists. We define for all $a \in A$ a sebes $\mathcal{E}_a$, which corresponds to the process algebra term $X = \mu(1, a, 0) \oplus X[f] \text{ where } f(\mu(i, a, n)) = \mu(i, a, n + 1)$.\(^2\)

\(\text{\footnotesize Here we use a relabeling operator as defined in Section 5.2. An isomorphic event structure can also be derived by the refinement operator.}\)
In the following definition, sequences $\star_2^{n} \star_1 i \bullet$ are considered to be right bracketed and therefore elements of $\mathcal{U}$.

$$\mathcal{E}_a = \{ \{ \star_2^n \star_1 i \bullet \mid n \in \mathbb{N} \land i \in \{1, 2\} \}, \emptyset, \{(\{\star_2^n \star_1 i \bullet \mid n \in \mathbb{N} \backslash \{j\}\} \cup \{\star_2^j \star_1 i \bullet \mid j \in \mathbb{N} \land i \in \{1, 2\}\}, \{\{\star_2^n \star_1 i \bullet, \star_2^j \star_1 i \bullet \mid n \in \mathbb{N}\}, \{\{\star_2^n \star_1 i \bullet \mid n \in \mathbb{N}\}, \{\{\star_2^n \star_1 i \bullet, \mu(i, a, n) \mid n \in \mathbb{N} \land i \in \{1, 2\}\})$$

Define $\theta' : A \rightarrow \text{SEBES}$ by $\theta'(a) = \mathcal{E}_a$. Furthermore, define $\mathcal{E}^{(a, n)}$ by

$$\mathcal{E}^{(a, n)} = (\{\star_2^n \star_1 i \bullet, \emptyset, \{\{\star_2^n \star_1 i \bullet, \star_2^m \star_1 i \bullet \mid m \in \mathbb{N}\}, \emptyset, \{\{\star_2^n \star_1 i \bullet \mid n \in \mathbb{N}\}, \{\{\star_2^n \star_1 i \bullet, \mu(2, a, n)\})$$

Let $\mathcal{R}_b$ be a strong bisimulation such that $(\text{Ref}^{se}_A(\mathcal{E}, \theta'), \text{Ref}^{se}_A(\mathcal{E}', \theta')) \in \mathcal{R}_b$. Without loss of generality, $\mathcal{R}_b$ contains only elements which can be derived from $\text{Ref}^{se}_A(\mathcal{E}, \theta'), \text{Ref}^{se}_A(\mathcal{E}', \theta')$. Furthermore, let $\kappa : \mathcal{U} \rightarrow \mathbb{N}$ be an isomorphism. We define the relation $\mathcal{R}_{ST}$ by

$$\mathcal{R}_{ST} = \{((\mathcal{E}, \mathcal{M}), (\mathcal{E}', \mathcal{M}'), \mathcal{E}_a) \mid \exists \emptyset, \emptyset : \emptyset \in \mathcal{E}, \emptyset : \emptyset \notin \mathcal{E}' : \emptyset : \emptyset \in \mathcal{E}_a : \emptyset : \emptyset \notin \mathcal{E}_a : \emptyset$$

Obviously, $((\mathcal{E}, \bot), (\mathcal{E}', \bot)) \in \mathcal{R}_{ST}$.

The verification that $\text{Ref}_{\text{Re}f}$ is a bisimulation is straightforward, where Lemma 7.46 and Lemma 7.47 are used.

### 7.8.3 Proof of Theorem 7.34

It is only necessary to check the correctness of (7.1): From Lemma 7.33 we get that the transition systems obtained from $\langle \text{decl}, H_i \rangle$ and $\langle \text{decl}, \text{decl}'(x_i)\rangle \{H_j/x_j\}_{j \in \{0, \ldots, n\}}$ are bisimilar for any $i \leq n$. Therefore, let $R_i$ be corresponding bisimulations. Without loss of generality, let every $R_i$ be reflexive. In the following, we write $(H, H') \in R_i$ instead of $((\text{decl}, H), (\text{decl}, H')) \in R_i$. Define

$$R = \{((\text{decl}, H), (\text{decl}', x_q) \mid q \leq n \land \exists i, f : \mathbb{N} \rightarrow \{1, \ldots, n\} : \langle H, H_q \rangle \in R_{f(1)} \circ \cdots \circ R_{f(i)} \}$$

Now we show that $R$ is a bisimulation. Therefore, let $((\text{decl}, H), (\text{decl}', x_q)) \in R$ such that $(H, H_q) \in R'$ where $R' = R_{f(1)} \circ \cdots \circ R_{f(i)}$. Then

$$(H, \text{decl}'(x_q)\{H_j/x_j\}_{j \in \{0, \ldots, n\}}) \in R' \circ R_q, \quad (7.6)$$

since $(H_q, \text{decl}'(x_q)\{H_j/x_j\}_{j \in \{0, \ldots, n\}}) \in R_q$. 

Then there is $m \leq n$ such that $\tilde{H}' = x_m$ since $(\text{decl}', \{x_0, \ldots, x_n\}) \in \text{SeqG}$. Thus decl$'(x_q)(\{H_j/x_j\}_{j \in \{0,\ldots,n\}}) \xrightarrow{\gamma} z_{\text{decl}} H_m$. Since $R' \circ R_q$ is a bisimulation, we obtain from (7.6) the existence of $\tilde{H}$ such that $H \xrightarrow{\gamma} z_{\text{decl}} \tilde{H}$ and $(\tilde{H}, H_m) \in R' \circ R_q$. Thus $(\langle \text{decl}, \tilde{H} \rangle, \langle \text{decl}', \tilde{H}' \rangle) \in R$.

$H \xrightarrow{\gamma} z_{\text{decl}} \tilde{H}$: Since $R' \circ R_q$ is a bisimulation we obtain from (7.6) the existence of $\tilde{H}'$ such that decl$'(x_q)(\{H_j/x_j\}_{j \in \{0,\ldots,n\}}) \xrightarrow{\gamma} z_{\text{decl}} \tilde{H}'$ and $(\tilde{H}, \tilde{H}') \in R' \circ R_q$. Thus $x_q \xrightarrow{\gamma} z_{\text{decl}} x_m \wedge \tilde{H}' = H_m$ for some $m \leq n$, since decl$'$ is sequential guarded with respect to $\{x_0, \ldots, x_n\}$. Hence, $(\langle \text{decl}, H_0 \rangle, \langle \text{decl}', x_0 \rangle) \in R$ follows from the reflexivity of $R_0$, which establish Theorem 7.34.

### 7.8.4 Proof of Theorem 7.39

The verification of the completeness is similar to [44], which uses the technique from [137, 139]. The verification of our completeness result differs a little bit from [44], since we do not split the internal action, and termination is determined by the final action. More precisely, the above conditions lead to rules $R_6$ and $R'_6$, where we have a transition $(C' \xrightarrow{a} C'' \xrightarrow{b})$ in which the left process ($C'$) is no subterm of the original process ($C[\phi, \psi]_{M_A}$). Consequently, structural induction can not be used to verify completeness.

For this reason, we introduce a weight function on $\text{PA}_{\text{se}}^{\text{Cu}}$, which is used for induction. Weight function $\Lambda$ counts an upper bound of the possible numbers of actions that can start when no action finishes. This is done respectively for every action to guarantee the well-definedness of the weight function for the refinement operators. $\Lambda$ also counts the numbers of the reachable variables ($\Lambda(\tilde{u})$). Function $\Lambda^c$ counts the reachable variables together with the upper bound of the possible number of actions that can start when no action ends, i.e. $\Lambda^c$ is the sum over $\Lambda$.

**Definition 7.53** Define $\Lambda : \text{PA}_{\text{se}}^{\text{Cu}} \rightarrow ((\text{Obs} \cup \{\tilde{u}\}) \xrightarrow{\text{fin}} \mathbb{N})$ as follows, where we do not mention decl explicitly

\[
\begin{align*}
\Lambda(H)(c) &= 0 & \text{if } H \in \{0, \tau, b^-, b_q^-, b_q^+; H', \tau; H'\} \\
\Lambda(b)(c) &= \begin{cases} 1 & \text{if } c = b \\ 0 & \text{otherwise} \end{cases} \\
\Lambda(b^+; H')(c) &= \begin{cases} 1 + \Lambda(H')(c) & \text{if } c = b \\ \Lambda(H')(c) & \text{otherwise} \end{cases} \\
\Lambda(H)(c) &= \Lambda(H')(c) & \text{if } H \in \{H'; H'', H'\} \setminus \{M_A\} \\
\Lambda(H)(c) &= \Lambda(H_1)(c) + \Lambda(H_1)(c) & \text{if } H \in \{H_1 + H_2, H_1 + H_2, H_1 - H_2, H_1 \oplus_M H_2, H_1 \parallel_M H_2, H_1 \parallel_{A,M} H_2, H_1 \parallel_{A,M} H_2\}
\end{align*}
\]
CHAPTER 7. START-BASED TOGETHER WITH END-BASED CHOICE

\[ \Lambda(H[(a \rightarrow H_a)^{a \in A}, (a \rightarrow \vec{H}_a)^{a \in \vec{A}}_M])(c) = \]
\[ \Lambda(H)(c) + \left( \sum_{a \in A} \Lambda(H)(a) \cdot \Lambda(H_a)(c) \right) + \sum_{a \in \vec{A}} \sum_{i \leq |\vec{H}_a|} \Lambda(\vec{H}_a[i])(c) \]
\[ \Lambda(H[(a \rightarrow H_a)^{a \in A}, (a \rightarrow H_a')[a \in A], (a \rightarrow \vec{H}_a)^{a \in \vec{A}}_{M', H'}])(c) = \]
\[ \Lambda(H)(c) + \left( \sum_{a \in A} \Lambda(H)(a) \cdot \Lambda(H_a)(c) + \Lambda(H')(a) \cdot \Lambda(H'_a)(c) \right) + \]
\[ \sum_{a \in \vec{A}} \sum_{i \leq |\vec{H}_a|} \Lambda(\vec{H}_a[i])(c) \]
\[ \Lambda(x)(c) = \begin{cases} 
1 + \Lambda(decl(x))(c) & \text{if } c = \emptyset \\
\Lambda(decl(x))(c) & \text{otherwise} 
\end{cases} \]

Furthermore, define \( \Lambda^c : PA_{\text{se}} \rightarrow \mathbb{N} \) by \( \Lambda^c(H) = \sum_{c \in \text{Obs} \cup \{\emptyset\}} \Lambda(H)(c) \).

**Lemma 7.54** Function \( \Lambda \) is well defined and consequently function \( \Lambda^c \) is also well defined.

**Proof:** We show that
\[ \forall ((\text{decl}, H), V) \in \text{VarSp} : G(((\text{decl}, H), V), \vec{V}) \Rightarrow \Lambda(H) \text{ is well defined}. \]

This is done by induction on \(|V \setminus \vec{V}|\) combined with the structure of \( H \) where the lexicographic order is used.

The rest is an immediate consequence of Lemma 7.37. \( \square \)

The nice property that the start of an action reduces the weight holds:

**Lemma 7.55** Suppose \( (\text{decl}, H) \in PA_{\text{se}}, a \in \text{Obs} \) and \( H^+ \xrightarrow{a^+ \text{ decl}} H' \). Then
\[ \Lambda(H)(a) > \Lambda(H')(a) \land \forall c \in (\text{Obs} \cup \{\emptyset\}) : \Lambda(H)(c) \geq \Lambda(H')(c) \]

and consequently \( \Lambda^c(H) > \Lambda^c(H') \).

**Proof:** This can be verified by induction on the depth of inference of \( H \xrightarrow{a^+ \text{ decl}} H' \). \( \square \)

Guardedness implies finitely branching:

**Lemma 7.56** Suppose \( (\text{decl}, H) \in PA_{\text{se}} \) then \( H \) is finitely branching with respect to \( \xrightarrow{\text{decl}} \).

**Proof:** It follows by induction on \( \Lambda^c(H) \) combined with the structure of \( H \) where the lexicographic order is used. In the case of rules \( R_6 \) and \( R_6' \), we make use of Lemma 7.55. \( \square \)

It can be derived from our axioms that a process is equivalent to the choice of its branches. This is illustrated by the following lemma, where \( \sum_H \gamma ; H' \rightarrow^\gamma \text{ decl} H' \) is really an expression, i.e. it has a finite choice, by Lemma 7.56.

**Lemma 7.57** Suppose \( (\text{decl}, H) \in PA_{\text{se}} \), then
\[ \vdash_{\text{decl}} H = \sum_{H \rightarrow^\gamma \text{ decl} H'} \gamma ; H' \]
7.8. PROOFS

Proof: It follows by induction on $\Lambda^e(H)$ combined with the structure of $H$ where the lexicographic order is used. In the case of the refinement operator, we make use of Lemma 7.55. □

As a consequence of the previous lemma, we can derive from our axioms that every guarded and finite state process is equivalent to a sequentially guarded process.

**Corollary 7.58** Suppose $\langle \decl, H \rangle \in \PA_{\text{Gu}}^\text{se}$ is finite state, then there is $\langle \decl', \{x_0, \ldots, x_n\} \rangle \in \SeqG$ such that $\vdash \langle \decl, H \rangle = \langle \decl', x_0 \rangle$.

Proof: There is only a finite number of different expressions reachable from $H$ by the transition rules, since $\langle \decl, H \rangle$ is finite state. Let $\{H_0, \ldots, H_n\}$ be the set of expressions reachable from $H$ with $H_0 = H$.

Define $\decl'(x_i) = \sum_{H_i} \gamma_i \cdot x_j$, which is well defined since $\langle \decl, H \rangle$ is finitely branching by Lemma 7.56. It is easily seen that $\langle \decl', \{x_0, \ldots, x_n\} \rangle \in \SeqG$. Furthermore, $\vdash \decl H_i = \decl'(x_i) \{ (H_j / x_j)^{j \in \{0, \ldots, n\}} \}$ by Lemma 7.57. Thus $\vdash \langle \decl, H \rangle = \langle \decl', x_0 \rangle$ follows by rule (7.1).

Our derivation system derives that two processes are equivalent whenever they are sequentially guarded and ST-equivalent.

**Lemma 7.59** Suppose $\langle \decl, \{y_0, \ldots, y_m\} \rangle, \langle \decl', \{x_0, \ldots, x_n\} \rangle \in \SeqG$ such that $\langle \decl, y_0 \rangle$ is ST-equivalent to $\langle \decl', x_0 \rangle$. Then there is $\langle \decl'', V \rangle \in \SeqG$ such that $|V| < |\IN|$ and $\vdash \langle \decl, y_0 \rangle = \langle \decl'', z_00 \rangle$ and $\vdash \langle \decl', x_0 \rangle = \langle \decl'', z_00 \rangle$ for some $z_00 \in V$.

Proof: Let $R \subseteq \{y_0, \ldots, y_m\} \times \{x_0, \ldots, x_n\}$ be a bisimulation such that $(y_0, x_0) \in R$. Define $V_z = \{z_{ij} \mid i \leq n \land j \leq m \land (x_i, x_j) \in R\}$. Let

$$\decl''(z_{ij}) = \sum_{y_i \rightarrow^\gamma \decl y_k, x_j \rightarrow^\gamma \decl x_l, (y_k, x_l) \in R} \gamma; z_{kl}.$$ 

Then $\langle \decl'', V \rangle \in \SeqG$. Furthermore, $\vdash \decl y_i = \decl''(z_{ij}) \{ (\decl(y_k) / z_{kl})^{k \in \{0, \ldots, m\}, l \in \{0, \ldots, n\}} \}$ for any $z_{ij} \in V_z$, since $R$ is a bisimulation. Thus $\vdash \langle \decl, y_0 \rangle = \langle \decl'', z_00 \rangle$ by rule (7.1). With symmetrical arguments we obtain $\vdash \langle \decl', x_0 \rangle = \langle \decl'', z_00 \rangle$.

Now, we are ready to show the completeness result for guarded and finite state processes.

**Proof of Theorem 7.39:** It is an immediate consequence of Corollary 7.58, Lemma 7.59 and Theorem 7.34. □
Chapter 8

Conclusion

In this thesis, we motivated the approach that considers a choice operator as end-based triggered, in particular in the context of action refinement. We established the end-based choice operator by investigating a process algebra, which contains an end-based choice and an action refinement operator, and by developing a denotational, an operational and an axiomatical semantics for this process algebra. We showed that these semantics are consistent. More precisely, the operational and the denotational semantics are bisimilar, and the axiomatical semantics is sound and complete (for guarded and finite state processes) with respect to the bisimulation equivalence obtained from the operational (denotational) semantics.

We had to investigate a new technique (approximation closedness) to restrict event structures that are based on the bundle technique in order to obtain a complete partial order. Furthermore, we used a new technique to show the bisimilarity between the operational and the denotational semantics. This new technique can handle unguarded recursion.

We also investigated new equivalences in the extended bundle event structures setting. These new equivalences are congruences for the action refinement operator that considers the conflict relation in extended bundle event structures to be end-based triggered. The valid relations between the trace equivalence, the bisimulation equivalence and these new equivalences are summarized in Figure 8.1 (if two equivalences are connected via a line, then the lower one identifies more elements than the upper one).

We pointed out that extended bundle event structures are not appropriate to model the end-based view, since the intuitive congruence equivalences fail to be the coarsest for the end-based action

![Figure 8.1: Relations Between the Equivalences](image)

165
refinement operator with respect to trace (respectively bisimulation) equivalence. Therefore, we investigated new event structures, namely extended termination bundle and extended termination precursor event structures, which have more general disabling relations. These event structures were examined in the context of a process algebra that contains a disrupt operator. The expressive power with respect to the set of event traces, describable by the classes of event structures, was examined. The hierarchy of this expressive power is depicted in Figure 8.2, where prime event structures can describe less set of event traces than the other event structures. The ICT-equivalence is the coarsest equivalence with respect to trace equivalence, and the FUI-equivalence is the coarsest equivalence with respect to bisimulation equivalence for the end-based action refinement operator in the extended termination event structures setting.

We introduced a choice operator where one side triggers the choice by ending actions and the other side triggers the choice by starting actions. Furthermore, we argued that it is useful to have this kind of choice as well as a start-based and an end-based choice in a single setting.

In the context of a process algebra with a start-based choice, we investigated a new technique of defining an operational semantics. This technique can handle action refinement and disruption in a reasonable way. Moreover, it is not necessary to introduce new syntactic terms in order to give the operational semantics.
Bibliography


[78] Harald Fecher and Mila Majster-Cederbaum. Taking decisions late: End-based choice combined with action refinement. In Derrick et al. [71].


[81] Harald Fecher, Mila Majster-Cederbaum, and Jinzhao Wu. Refinement of actions in a real-time process algebra with a true concurrency model. In Derrick et al. [71].


[95] Roberto Gorrieri and Cosimo Laneve. The limit of split
1


Index

Symbols.
[

\[ \], 39, 93, 127
\(|\), 142
\(\sim_b\), 21, 60, 111
\(\sim_c\), 67
\(\sim_{FUI}\), 65, 113
\(\sim_{ICT}\), 63, 111
\(\sim_{ST}\), 138
\(\sim_t\), 21, 60, 111
\(\sim_{UI}\), 64
\(\rightarrow\), 18
\(\to\), 19
\(\Rightarrow\), 19
\(\rightarrow_{fin}\), 19
\(\cong\), 20
\(\preceq\), 33, 90, 124
\(\preceq^\ast\), 96
0, 30, 81, 122, 139
1, 30
\(\ldots\), 30
\(+\), 30, 81, 122, 139
\(\vdash\), 122, 129, 139
\(\sqsubseteq\), 139
\(\oplus\), 122, 129, 139
\(\oplus^\ast\), 139
\(\ldots\), 30, 81, 122, 129, 139
\(\triangleright\), 30, 81
\(\parallel\), 30, 81, 122, 129, 139
\(\parallel\), 139
\(\parallel\), 30
\(\parallel\), 81, 122, 129, 139
\(\parallel\), 37
\(\checkmark\), 37, 90, 125
\(\checkmark\), 125
\(\checkmark\), 125
\(\checkmark\), 37, 91, 125
\(\checkmark\), 37, 91
\(\parallel\), 37, 91, 126

\(\hat{A}\), 38
\(\hat{\hat{A}}\), 91, 126
*, 17
\(\rightarrow_{\sigma^*}\), 43
\(\rightarrow_{c^*}\), 131
\(\rightarrow_{s^*}\), 42
\(\rightarrow_{t^*}\), 82
\(\rightarrow_{z^*}\), 139
\(\rightarrow\), 136
\(\leftarrow\), 36, 88, 136
\(\leftarrow^c\), 137

ACP, 11
Act, 80
Act\(\sqrt{.}\), 30
Act\(T\), 81
action, 10
atomic, 10
instantaneous, 10
internal, 10, 30
observable, 10, 30, 44
set of, 30, 80
termination, 30, 81
action refinement, 12, 29–30
atomic, 29
non-atomic, 29
syntactical, 14
vertical, 30
\(\alpha\)-equivalence, 43
approximation
closed, 23
finite, monotone, 23
automaton
finite, 9
axioms, 142

bijective, 19
bisimilar, 21
FUI, 65, 113

179
ST, 138, 139
strong, 60, 111
UI, 64
bisimulation, 21
  finite unique initial, see bisimulation, FUI
  FUI, 65, 112
  UI, 64
  unique initial, see bisimulation, UI
bundle stability constraint, 32, 85, 93
cardinality, 17
causal ambiguity, 32
causal trees, 11
CBES, 35
cbes, see event structures, closed bundle
CCS, 11
chain, 22
choice
  end-based, 57, 109, 121
  end-start, 121
  start-based, 121
coarsest, see congruence, coarsest
completeness, 146
configurations, 83
congruence, 20
  coarsest, 20, 67, 112, 114, 138
  FUI, 66, 113
  ICT, 64, 112
  ST, 138
  UI, 65
consistency, 15, 43, 93, 136
continuous, 22
  componentwise, 23
  on events, 48
countable, 17
cpo, see partial order, complete
CSP, 11
decl, 30, 81, 122, 139
declaration, see decl
design errors, 9
dom, 19
domain, 19
dual language approach, 9
dynamic names, 127
E, 32, 34, 86, 123
ε, 137
ξ, 94
EBES, 32
ebes, see event structures, extended bundle
environment, 10
equivalence, 60–61
  bisimulation, see bisimilar
  FUI, see bisimilar, FUI
  history preserving, 61
  ICT, 63, 111
  initial corresponding trace, see equivalence, ICT
  pomset, 61
  resource bisimulation, 61
  ST, 61, 138, 139
  step, 61
  trace, 21, 60, 111
  UI, see bisimilar, UI
ETBES, 86
eTbes, see event structures, extended termination bundle
ETPES, 94
eTpes, see event structures, extended termination precursor
event, 18
  initial, 35, 87, 95, 123
  set of, 32, 85, 123
  termination, 35, 87, 123
trace, 89
  initial, 63
  set of, 89, 97
universe of, 18
event structures, 11
  bundle, 31, 89
  closed bundle, 34, 83
dual, 33, 83, 89
  extended bundle, 32, 89
  extended termination bundle, 86
  extended termination precursor, 94
flow, 83, 89
prime, 83, 89
restriction of, 33, 90, 124
stable, 83, 89
start-end, 123
time bundle, 57
exit, 35
INDEX

\[\text{EXP}_{se}, \ 122\]
\[\text{EXP}^{Ax}_{se}, \ 139\]
\[\text{EXP}_{sr}, \ 30\]
\[\text{EXP}_{st}, \ 81\]

\[\mathcal{F}, \ 97\]
\[\mathcal{F}_{\text{decl}}, \ 127\]
\[\mathcal{F}_{\text{decl}}, \ 39, \ 93\]
\[F_E, \ 97\]
\[F_L, \ 80\]

fa-approach, 79, 80
 finite state, 146
 finitely determined, 94
 formal methods, 9
 function, 18–19
  action-labeling, 32, 85, 123
  inverse, 19
  labeling preserving, 63
  partial, see partial function
  relabeling, 80

\[G, \ 145\]
guarded, 145
  sequentially, 142

\[\text{Id}, \ 18\]
\[\text{init}, \ 35, \ 62, \ 87, \ 123, \ 136\]
\[\hat{\text{init}}, \ 95\]
injective, 19
interleaving, 10

\[L, \ 41\]
\[L_{se}, \ 130\]
\[\hat{\text{Lab}}, \ 91\]
label, 20
\[\Lambda^c, \ 162\]
least
  element, 22
  fixpoint, 23
  upper bound, 22
literals
  negative, 138
logic, 9
LOTOS, 11

metric space, 22
model checking, 9, 12
monotonic, 20

\[\mathbb{N}^+, \ 17\]
natural numbers
  positive, 17

\[O^a, \ 43\]
Obs, 30
\[\text{Obs}_a, \ 44\]
\[\text{Obs}_p, \ 44\]
\[\omega\text{-chain, see chain}\]
\[\omega\text{-continuous, see continuous order, see partial order}\]
  componentwise, 23
  pointwise, 23

\[P, \ 17\]
\[P_{\text{count}}, \ 17\]
\[P_{\text{fin}}, \ 17\]
\[\text{PA}_{se}, \ 122\]
\[\text{PA}^{Ax}_{se}, \ 139\]
\[\text{PA}^{Gu}_{se}, \ 145\]
\[\text{PA}_{sr}, \ 30\]
\[\text{PA}_{st}, \ 81\]
panth, 138
partial function, 19
  bijective, 19
  domain, 19
  injective, 19
  inverse, 19
  surjective, 19
partial order
  complete, 22, 35, 90, 97, 124
  \(\omega\)-complete, see partial order, complete
performance analysis, 11
petri net, 11
pointers, 127
pomsets, 11
preserved
  relation, 20, 61
process, 30
  action prefix, 31
  choice, 31, 81
  end-based, 122
  end-start, 122
  start-based, 121
disrupt, 31, 81
hiding, 31
inactive, 31, 81
parallel, 31, 81
refinement, 31, 122
relabeling, 81
restriction, 81
sequential, 31, 81
specified by, 145
terminate, 31
variable, 30, 81

process algebra, 9, 11
priority, 12
probability, 12
stochastic, 12
time, 12
projection, 19

race policy, 13
reactive system, 10
concurrent, 10
$Ref^e_A$, 59
$Ref^e_A^r$, 69
$Ref^e_A^T$, 109
$Ref^e_T$, 115
$Ref^e_A$, 38
$Ref^e_A^{st}$, 126

relation
bundle, 32
causality, 85, 123
conflict, 85
asymmetric, 33
precursor, 94
start, 123
symmetric, 32
identity, 18, 34
transition, see transition, relation
witness, 86
end, 123
relative active number, 128
remainder, 36, 88, 95, 135

SEBES, 123
$SEBES_M$, 136
sebes, see event structures, start-end
semantics
axiomatic, 11
denotational, 11, 40, 93, 127
operational, 11, 42, 81, 130
SeqG, 142

soundness, 142
specification, 9
descriptive/property-based, 9
imperative/operational-based, 9

stack technique, 127
state, 20
initial, 20
state explosion problem, 12
static names, 127
string, 17
empty, 17
substitution, 142
syntactic, 30
surjective, 19
system design
hierarchical, 12
top down, 29

$T$, 21
$T^{ic}$, 63, 111
$\tau$, 30, see action, internal
TBES, 85
Tbes, see event structures, termination bundle
TCSP, 11
termination
action, see action, termination
event, see event, termination
predicate, 87, 95, 123
set, 85, 123
$Tr^e$, 89
trace, 21
initial event, 111
semantics, 10
transition
relation, 20, 36, 88, 137
rules, 11, 41, 81, 131, 139
complete, 138
system, 9–11, 20
true concurrency, 11
TS, 20

$\mathcal{U}$, 18
upper closed, 94
$\Upsilon$, 87, 123
$\Upsilon^*$, 95

Var, 30
variable assignment, 38, 93, 127
VarSp, 145
verification, 9
\mathcal{X}', 24
Z, 14
Zusammenfassung


Beim Definieren einer denotationalen Semantik hatte man seither das Problem, dass die Ereignisstrukturen (engl: event structures), welche auf der Bundle-Technik basieren, keine cpo mit der üblichen Ordnung liefern. Deshalb wurde eine Technik zur Definition von Einschränkungen dieser Ereignisstrukturen entwickelt, so dass diese Einschränkungen cpo mit der üblichen Ordnung liefern. Weiterhin wurde auch eine neue Technik vorgestellt, um operational Semantiken, die den denotationalen Semantiken entsprechen, für Prozess-Algebren mit einem Aktionsverfeinerungsoperator zu entwickeln. Mit dieser Technik ist es nicht nötig, die Prozess-Algebren um weitere Ausdrücke zu erweitern, um die operationale Semantik zu definieren.


Kongruenzäquivalenzen für den Aktionsverfeinerungsoperator, der die Konfliktrelation als end-basiert betrachtet, tatsächlich die kleinsten Kongruenzen bezüglich der Bisimulations- und der Trace-Äquivalenz liefern.


Danksagungen

Vor allem möchte ich Frau Prof. Dr. Mila Majster-Cederbaum für Ihre Betreuung danken. Sie ließ mir den nötigen Freiraum, stand allzeit für meine Fragen bereit und half mir beim Formulieren mathematischer Texte in englischer Sprache. Ich möchte mich auch herzlich bei Herrn Prof. Dr. Franz Stetter, meinem Zweitkorrektor, bedanken. Mein Dank gilt außerdem meinen Kollegen, unter anderem Dr. Jinzhao Wu, Jürgen Jaap, Dr. Naijun Zhan, Dr. Sven Helmer und insbesondere meinem Zimmerkollegen Dr. Frank Salger. Bedanken möchte ich mich auch bei Frau Jackowski für ihre hilfreiche Unterstützung bei den anfallenden Formalitäten. Zuletzt möchte ich meiner Frau Annette danken, ohne die ich das alles nicht geschafft hätte.