DISSERTATIONES
MATHEMATICAE
UNIVERSITATIS
TARTUENSIS
100

ABEL ARMAS CERVANTES

Diagnosing Behavioral Differences between Business Process Models

DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS
100

DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS
100

ABEL ARMAS CERVANTES

Diagnosing Behavioral Differences between Business Process Models

UNIVERSITY OF TARTU
PRESS

Institute of Computer Science, Faculty of Mathematics and Computer Science, University of Tartu, Estonia.

Dissertation has been accepted for the commencement of the degree of Doctor of Philosophy (Ph.D.) in informatics on June 15, 2015, by the Council of the Institute of Computer Science, Faculty of Mathematics and Computer Science, University of Tartu.

Supervisors:

Prof. PhD.	Marlon Dumas Institute of Computer Science University of Tartu, Tartu, Estonia
Assoc. Prof. PhD.	Luciano García Bañuelos Institute of Computer Science University of Tartu, Tartu, Estonia

Opponents:

Res. Assoc. PhD.	Eric Badouel INRIA Rennes, France
Assoc. Prof. PhD.	Josep Carmona Department of Computer Science Universitat Politecnica de Catalunya (UPC), Barcelona, Spain

Commencement will take place on August 28, 2015, at 13.15 in Liivi 2-403.

The publication of this dissertation was financed by Institute of Computer Science, University of Tartu.

European Union
European Social Fund

Investing in your future

DoRa

For education and research in Estonia
ARCHIMEDES

ISSN 1024-4212
ISBN 978-9949-32-865-9 (print)
ISBN 978-9949-32-866-6 (pdf)

University of Tartu Press
www.tyk.ee

Abstract

Companies operating in multiple markets or market segments often need to manage multiple variants of the same business process. Such multiplicity of variants may stem from distinct products, different types of customers, different regulations across countries in which the company operates, or idiosyncratic choices made by multiple business units over time. During the ongoing management of these processes, analysts need to compare models of multiple process variants in order to identify opportunities for standardization or to understand relative performance differences across variants.

Existing approaches to process model comparison can be broadly classified into those based on structural similarity and those based on behavioral similarity. Approaches based on structural similarity can conveniently explain certain types of differences between pairs of process models, such as insertion, deletion or substitution of tasks, or simple re-arrangements of nodes (e.g. swapping of two tasks). However, two variants may be syntactically different and still be behaviorally equivalent. Conversely, they may be similar syntactically while very different behaviorally, as changes in a few gateways or edges may entail significant behavioral differences.

In this context, this thesis addresses the problem of diagnosing behavioral differences between pairs of business process models, based on a notion of equivalence that takes into account concurrency. Given two process models, the thesis proposes a method to determine if they are behaviorally equivalent, and if not, to describe their differences in terms of behavioral relations captured in one model but not in the other. The proposed solution is based on a translation from process models to event structures, specifically prime event structures, asymmetric event structures and flow event structures. A naïve version of this translation suffers from two limitations. First, this translation is not applicable to process models with cycles. Second, it produces redundant difference diagnostic statements because an event structure may contain unnecessary event duplication. To tackle the first limitation, the thesis proposes a notion of unfolding that captures all possible causes of each task, where the tasks that can occur more than once in a computation are distinguished from those that cannot. From this unfolding, an event structure is derived, thus enabling the diagnosis of behavioral differences in terms of repetition and behavioral relations that hold in one model but not in the other. For the second limitation, the thesis puts forward a technique to reduce event duplication in an (asymmetric and a flow) event structure while preserving canonicity by applying a set of behavior-preserving event folding rules. The proposed method has been implemented as a prototype that takes pairs of process models in the standard Business Process Model and Notation (BPMN) and produces difference diagnostics both in the form of statements in natural language and graphically overlaid on the process models.

Acknowledgements

This research was supported by the European Social Fund via the Doctoral Studies and Internationalisation Programme (DoRa), which is carried out by Foundation Archimedes, and by an institutional grant of the Estonian Research Council.

I would like to thank my supervisors Marlon Dumas and Luciano García-Bañuelos for their guidance, the fruitful discussions and continuous encouragement throughout these years. I am grateful for the constant opportunities for professional development they gave me. Special thanks to Paolo Baldan from University of Padova who took the role of my third supervisor during and after my research visit to University of Padova. His mentoring, availability for discussions and collaboration has made this thesis possible. Last but not least, I would also like to extend my sincere gratitude to the reviewers Eric Badouel and Josep Carmona for their insightful comments.

Contents

List of Abbreviations **11**

List of Symbols **12**

List of Figures **13**

List of Original Publications **17**

1 Introduction **19**

1.1 Problem Statement 21

1.2 Contributions 25

1.3 Outline 26

2 State of the art **29**

2.1 Process model comparison based on task labels 30

2.2 Process model comparison based on model structure 31

2.3 Process model comparison based on behavior 33

3 Background **36**

3.1 Petri nets 37

3.1.1 Petri net subclasses 45

3.1.2 Branching process of a Petri net system 51

3.1.3 Configurations and families of pomsets 54

3.2 Event structures 57

3.2.1 Prime event structures 58

3.2.2 Asymmetric event structures 59
3.2.3 Flow event structures 64
3.3 True concurrency semantic equivalences 68
3.3.1 Configuration equivalence 69
3.3.2 Completed visible-pomset equivalence 70
3.3.3 History preserving bisimilarity 71

4 Behavioral profiles for process model comparison 73
4.1 Behavioral profiles (BP) 74
4.2 FES as BP 77
4.3 An execution semantics for $BP|_w$ 80
4.4 Expressing differences using $BP|_w$ 85
4.5 BP and silent transitions 87
4.6 Discussion 90

5 Process model comparison based on event structures 92
5.1 Finite representation of cyclic process models 93
5.1.1 Multiplicity of activities 97
5.1.2 Multiplicity of activities in free-choice workflow nets 100
5.2 Comparison based on event structures 102
5.2.1 Partial synchronized product 106
5.2.2 Identifying differences 114
5.2.3 Verbalizing differences 117
5.3 Discussion 121

6 Reduction of event structures 123
6.1 Foldings 124
6.2 Reduction of AESs 124
6.3 Reduction of FESs 139
6.4 Deterministic foldings and canonicity 153
6.5 Discussion 156

7 Implementation and validation **159**

7.1 Evaluation 161

8 Conclusions **169**

8.1 Summary of contributions 169

8.2 Future work 172

References **175**

Appendix A Basic notions and notations **183**

A.1 Sets and numbers 183

A.2 Sequences 183

A.3 Relations 184

A.4 Functions 184

List of Abbreviations

Abbreviation	**Meaning**	**Page**
BPMN	Business Process Model and Notation	19
EPC	Event-driven Process Chain	19
UML	Unified Modeling Language	19
FSM	Finite State Machine	32
LTS	Labeled Transition System	33
BP	Behavioral Profiles	34
WF-net	Workflow net	47
ES	Event Structure	57
PES	Prime Event Structure	58
AES	Asymmetric Event Structure	59
FES	Flow Event Structure	64
WF-flow net	Workflow and flow net	76
CP	Complete Prefix	93

List of Symbols

Symbol	**Description**
$\mathbb{A}$	Asymmetric event structure.
$\mathbb{F}$	Flow event structure.
$\mathcal{N}$	Petri net system.
β	Branching process.
$\mathbb{P}$	Prime event structure.
$\approx_{conf}$	Configuration equivalence.
Λ	Tasks labels.
$\mathbb{N}_0$	Natural numbers including 0.
Δ	Executions of a net system.
ψ	Strong postconditions.
$\mathcal{P}$	Family of pomsets.
$\approx_{cp}$	Completed visible-pomset equivalence.
$\approx_{hp}$	History preserving bisimulation equivalence.
η	WF-flow nets.
$\mathscr{R}$	Behavioral relations.
$\mathfrak{N}$	Class of nets.
$\equiv_{iso}$	Isomorphism.
$\eta_{\overline{\lambda}}$	Unlabeled WF-flow nets.
Θ	Cutting context.
$\Theta_{McMillan}$	Cutting context defined in [McMi 95].
Θ_{ERV}	Cutting context defined in [Espa 02].
Θ_{Pred}	(new) Cutting context based on predecessors.
$\mathcal{R}$	Self-preceding transitions.
$\mathcal{K}$	Necessary transitions.
$\mathbb{E}$	Event structure, in Chapter 5 either $\mathbb{A}$ or $\mathbb{F}$.

List of Figures

1.1 Subset of core BPMN elements 20

1.2 Equivalent variants of business process models 22

1.3 M_3, process model variant of models in Figure 1.2 23

1.4 Mapping of tasks, events and gateways to Petri nets 24

1.5 Petri net of process model M_3 (Fig. 1.3) 25

3.1 Messaging system modeled as a Petri net N 37

3.2 Petri net system $\mathcal{N}$ of N with a marking M_0 39

3.3 Net systems exemplifying causal, conflict, and concurrent relations between events 40

3.4 Petri net system $\mathcal{N}$ with a terminal marking M_x 42

3.5 Example of causal nets 43

3.6 Non-free choice Petri net 46

3.7 Occurrence net N_1 47

3.8 (sound) Workflow net system 48

3.9 Non-flow and flow net example 49

3.10 Flow net system and its configurations ordered by set inclusion 51

3.11 Branching process, inductive rules 52

3.12 Petri net system and its unfolding 53

3.13 Family of pomsets ordered by inclusion, i.e., pomsets of flow net system in Fig. 3.10a. 56

3.14 Example of PES 58

3.15 Example of AES 60
3.16 Inheritance of conflict along causality in AESs. 62
3.17 Inheritance of $\nearrow$ 63
3.18 AES $\mathbb{A}$ and its set of configurations ordered by extension . . 65
3.19 Example of FES 65
3.20 A FES which is neither faithful nor full. 67
3.21 Example of configuration-equivalent families of pomsets. . . 69
3.22 Visible pomset equivalent flow net systems (a), (b) and their visible pomsets ordered by inclusion (c). 70
3.23 History preserving bisimilar AESs 72

4.1 Net system and its behavioral profile $BP|_w$ 74
4.2 WF-flow nets 75
4.3 Flow net system and its corresponding FES 77
4.4 $BP|_{fes}(\mathcal{N}_6)$ 79
4.5 Equivalent WF-flow nets without (4.5a) and with (4.5b) an implicit place and their corresponding FESs aside. 80
4.6 Net system $\mathcal{N}_7$ and its behavioral profiles $BP|_w$ and $BP|_{fes}$ 81
4.7 Net system $\mathcal{N}_8$ and its behavioral profiles $BP|_w$ and $BP|_{fes}$ 83
4.8 Net system $\mathcal{N}_9$ and its behavioral profile $BP|_w$ 86
4.9 Branching process of net system $\mathcal{N}_9$ (Fig. 4.8a) 86
4.10 WF-flow net system and its $BP|_{fes}$ 88
4.11 Net systems with isomorphic sets of $4C$ relations over labels 89
4.12 Generalization of the net systems in Fig. 4.11 90
4.13 Net systems with isomorphic sets of $4C$ relations over labels without concurrency 90

5.1 Petri net system and its complete unfolding prefix 93
5.2 Complete unfolding prefix β_2 94
5.3 Non-free choice "cyclic" net system and its unfolding 100

5.4 Non hp-bisimilar PESs, but completed visible-pomset equivalent . . . 103
5.5 PES and its restriction to observable behavior . . . 104
5.6 Example of graph-based PES comparison . . . 106
5.7 Partial matching operations given a partial match (C_1, ξ, C_2) 108
5.8 PESs and a pair of partial matches between their configurations 109
5.9 PESs and their partial synchronized product with the optimal partial matches . . . 112
5.10 (a) Matrix representations for (a) partial match $(\{a_1, b_1, c_1\}, \xi, \{a_2, b_2\})$ and (b) extended partial match $(\{a_1, b_1, c_1\}, \zeta = \xi[c_1 \mapsto c_2], \{a_2, b_2\})$ . . . 115
5.11 Synthetic matching operation . . . 117
5.12 PESs and their partial synchronized product with the optimal matches . . . 120
5.13 AES equivalent to $\mathbb{P}_7$ in Figure 5.12a . . . 121
5.14 Three history preserving bisimilar event structures . . . 122

6.1 AES $\mathbb{A}'$ and a folding $\mathbb{A}''$. . . 125
6.2 AESs such that $\mathbb{A}_0 \equiv_{hp} \mathbb{A}_1$ but $\mathbb{A}_0 \not\equiv_{hp} \mathbb{A}_2$. . . 125
6.3 Configurations of the AESs in Figure 6.2, ordered by extension 126
6.4 AES and its quotient . . . 129
6.5 Quotients with respect to a set $X = \{a_0, a_1\}$ of non-similar events . . . 133
6.6 The set $p(\{c_0, c_1\}) = \{a, b\}$, includes a which is neither in the history of c_0 nor of c_1 . . . 135
6.7 Foldings for the AES in Figure 6.2 . . . 138
6.8 FES $\mathbb{F}$ and a folding $\mathbb{F}'$ . . . 139
6.9 Sample FESs . . . 140
6.10 Example of direct conflict in FES, $a \#_\delta d$ and $\neg(d \#_\delta a)$ . . . 141
6.11 Example FESs to illustrate Condition 5 in Definition 6.12 . 143
6.12 FES and two minimal non-isomorphic quotients . . . 153

6.13 Equivalent AESs 153
6.14 Canonical labeling and folding 155
6.15 A PES $\mathbb{P}$ and a possible folding $\mathbb{P}'$ that cannot be obtained by composing elementary foldings 157
6.16 Non-completeness of the quotient technique for FESs 158

7.1 Process model example M_4 160
7.2 Representation of differences 161
7.3 Web interface of BP-Diff 162
7.4 Snippet of the process models SA 3 and WA 3 166
7.5 Difference 1 (J2, P2) between SA 3 and WA 3 using PES . . 167
7.6 Difference 2 (J2, P2) between SA 3 and WA 3 using PES . . 167
7.7 Difference 3 (J2, P2) between SA 3 and WA 3 using PES . . 167
7.8 Difference 4 (J2, P2) between SA 3 and WA 3 using PES . . 168
7.9 Difference 1 (J2, P2) between SA 3 and WA 3 using AES . . 168

List of Original Publications

I Abel Armas-Cervantes, Luciano García-Bañuelos and Marlon Dumas. Event Structures as a Foundation for Process Model Differencing, Part 1: Acyclic processes. In 9th International Workshop on Web Services and Formal Methods 2012, Tallinn, Estonia, September 6-7, 2012. Lecture Notes in Computer Science, vol. 7843, pages 69-86, Springer 2013.

- *The author contributed the selection of the examples, literature review and writing of some sections.*

II Abel Armas-Cervantes, Paolo Baldan and Luciano García-Bañuelos. Reduction of Event Structures under History Preserving Bisimulation. Submitted to Journal of Logical and Algebraic Methods in Programming, Special Issue: The 23rd Nordic Workshop on Programming Theory (NWPT 2013). 44 pages.

- *The author contributed part of the idea, part of writing, part of the formalizations and proofs and selection of examples.*

III Abel Armas-Cervantes, Paolo Baldan, Marlon Dumas and Luciano García-Bañuelos. Behavioral Comparison of Process Models Based on Canonically Reduced Event Structures. In Proceedings of the 12th International Conference on Business Process Management (BPM 2014), Eindhoven, The Netherlands, September 10, 2014. Lecture Notes in Computer Science, vol. 8659, pages 267-282, Springer 2014.

- *Lead author. The author contributed part of the idea, formalization, writing, proofs, selection of examples and implementation.*

IV Abel Armas-Cervantes, Marlon Dumas, Luciano García-Bañuelos and Artem Polyvyanyy. On the Suitability of Generalized Behavioral Profiles for Process Model Comparison. In 11th International Workshop on Web Services and Formal Methods: Formal Aspects of Service-Oriented and Cloud Computing 2014, Eindhoven, The Netherlands, September 11-12, 2014. LNCS, 15 pages, Springer 2015, (in press).

- *Lead author. The author contributed the idea, formalization, writing, proofs and selection of examples.*

V Abel Armas-Cervantes, Paolo Baldan, Marlon Dumas and Luciano García-Bañuelos. BP-Diff: A Tool for Behavioral Comparison of Business Process Models. In Proceedings of the BPM Demo Sessions 2014 Co-located with the 12th International Conference on Business Process Management (BPM) 2014, Eindhoven, The Netherlands, September 10, 2014. CEUR Workshop Proceedings, Vol. 1295, pp. 1-5.

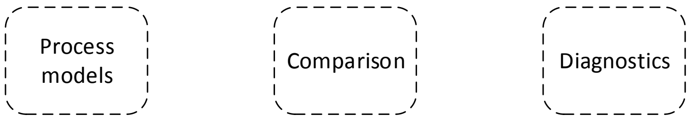
- *Lead author. The author contributed the idea, writing and implementation.*

VI Abel Armas-Cervantes, Paolo Baldan, Marlon Dumas and Luciano García-Bañuelos. Diagnosing Behavioral Differences Between Business Process Models: An Approach Based on Event Structures. Submitted to Journal of Information Systems, Special Issue of Business Process Management (BPM) 2014. 51 Pages.

- *Lead author. The author contributed the idea, implementation, writing, part of proofs and selection of examples.*

Chapter 1

Introduction



Business processes are the arteries of modern organizations. They determine how work is done in an organization. A business process is a "collection of related events, activities and decisions, that involve a number of actors and resources, and that collectively lead to an outcome that is of value to an organization or its customers." [Duma 13].

There are several modeling languages for business process modeling; BPMN, EPC and UML are cases in point. For the sake of exemplification, Figure 1.1 shows a subset of the core elements of BPMN. The *start* and *end* events represent the initiation and termination of an instance of a process, respectively. The *tasks* denote units of work to be performed. The *flow* represents the order among the events, gateways and tasks. Finally, *gateways* are control flow elements and they can represent either the splitting or merging of paths. In the case of *exclusive gateways*, a split has more than one outgoing flow, but only one of them can be activated (according to a defined condition); the counterpart, the join exclusive gateway, merges the incoming alternative flows. Conversely, the fork *parallel gateway* denotes the parallel activation of all the outgoing paths; whereas, the merging counterpart denotes the synchronization of the multiple incoming paths.

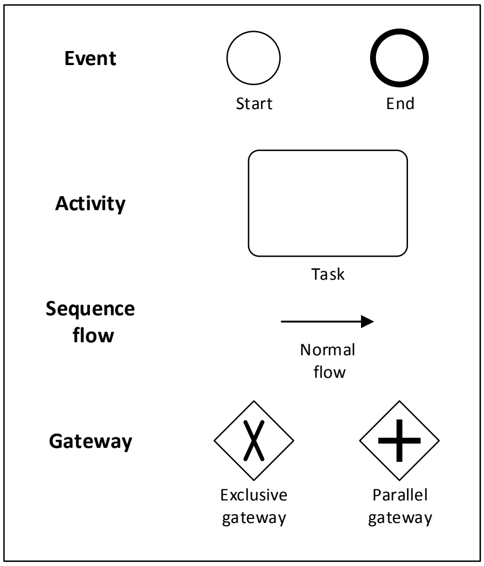


Figure 1.1: Subset of core BPMN elements

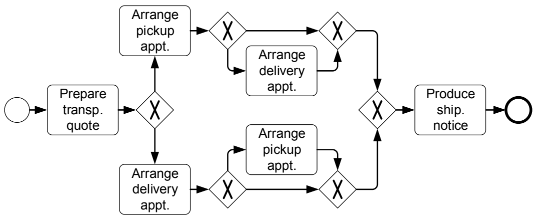
Process models offer a suitable representation for many analysis techniques, e.g., techniques to assess the performance of the overall process, and can cope with ambiguities that may emerge from, for instance, a textual description of processes. In business process management, process models are a pervasive element, insofar as the life cycle of a single process can encompass multiple versions of the same model. The *as-is* model represents an existing –or a new– process, from which the *to-be* model is constructed after a stage of analysis and redesign, and finally, a new improved model is created by correcting encountered issues, such as deviations or bottlenecks. The importance of process models as key information assets in modern organizations motivates the creation of effective techniques for their efficient management.

The comparison of process models is a basic operation when managing collections of business process models [Dijk 11]. For example, an organization with mature business process practices can gather large amounts of models. Oftentimes, they include multiple variants of the same process. Variants may stem, not only from the *as-is* and *to-be* process models, but from distinct products, different types of customers (e.g. corporate vs. private customers), different legislations across countries in which a company operates, or idiosyncratic choices made by multiple business units over time. In this setting, analysts need to compare models and accurately understand the differences between multiple variants in order to determine how to reconcile them.

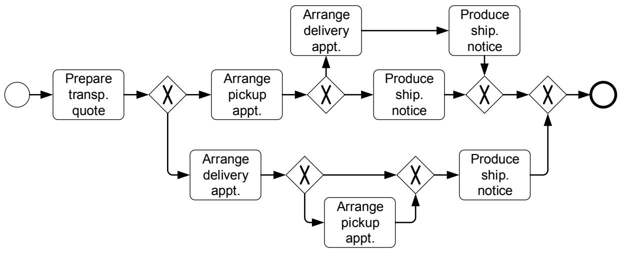
This thesis deals with the problem of comparing pairs of business process models, with an emphasis of comparing the behavior represented by the process models as opposed to comparing the process models lexically or syntactically. The specific process model comparison problem addressed in this thesis is formulated and scoped in Section 1.1. Next the contributions of the thesis are spelled out in Section 1.2. Finally, Section 1.3 provides the outline of the thesis.

1.1 Problem Statement

Existing techniques for process model comparison can be roughly divided into those based on structure and those based on behavior. In the former, the differences are explained as graph edit operations, such as remove, insert or replace tasks, that need to be applied in one model in order to obtain the other; whereas in the latter, the differences are explained in terms of the behavioral dependencies among tasks captured in one model and not in the other. In some cases, a structural comparison is sufficient to understand the differences between two variants. However, two variants may be structurally different, yet behaviorally equivalent; or they may be very similar structurally, but quite different behaviorally, as changes in a few gateways or edges may entail significant behavioral differences.



(a) M_1



(b) M_2

Figure 1.2: Equivalent variants of business process models

Figure 1.2 shows two process models M_1 and M_2 [1] represented using BPMN. They are structurally dissimilar to the extent that their numbers of tasks differ. Nevertheless, the executions of both process models are the same, and so they can be considered as behaviorally equivalent.

This thesis approaches the problem of diagnosing behavioral differences between pairs of business process models. Then given a pair of process models, this thesis proposes a method to determine if they are behaviorally equivalent – taking concurrency into account – and if not, the discrepancies are explained using simple and intuitive statements, the last in order to target the analysts as the end users of the technique.

The simplest way to explain the behavioral differences between a pair of processes is, possibly, as behavioral relations between pairs of tasks (or

[1]Based on an order fulfillment process presented in [Rosa 10].

occurrences of tasks) that hold in one model and not in the other. We specifically deal with three elementary types of behavioral relations that, together with repetition, have been postulated as basic control-flow workflow patterns [Aals 03], namely causal precedence (corresponding to “sequence” in a process model), conflict (exclusive branches in a process model), and concurrency (parallel branches in a process model).

Concurrency is an important construct in nowadays process modeling languages and analysis techniques. For instance, the cycle time of a process with the concurrent execution of a set of tasks differs from the cycle time of the same process with the arbitrary interleaved representation thereof. Therefore, we consider that while comparing process models, it is necessary to adopt an equivalence notion, together with behavioral abstractions, that preserve the concurrency modeled by the analysts.

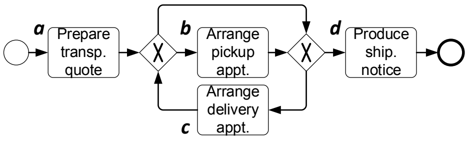


Figure 1.3: M_3, process model variant of models in Figure 1.2

As an example of the results that we aim at, consider the process model M_3 in Figure 1.3, which is a variant of the models in Figure 1.2. M_3 is structurally and behaviorally dissimilar to M_1 and M_2, then an intuitive way to explain the behavioral differences between M_3 and M_1 (similarly, between M_3 and M_2) to an analyst can be via statements of the form: *“In M_3, there is a state after* Prepare transportation quote *where* Arrange delivery appointment *can occur before* Produce shipment notice *or* Arrange delivery appointment *can be skipped, whereas in the matching state in M_1,* Arrange delivery appointment *has to occur before* Produce shipment notice*”*, and *“In M_3 activity* Arrange delivery appointment *occurs 0,1 or more times, whereas in M_1 it occurs at most once”*.

Petri nets [Petr 62] are a well-known modeling tool for concurrent processes [van 98] that has been widely used in the context of analysis of business processes. It has a formally defined semantics and there exist various available analysis techniques for it. Throughout the thesis we assume that the input process models are given as Petri nets. This design choice enables the application of the presented techniques to any process modeling language with a mapping to this formalism. For example, a transformation of a large subset of BPMN to Petri nets can be found in [Dijk 08b], and a subset of such mapping rules are shown in Figure 1.4.[1] For instance, the Petri net resulting from the application of the mapping rules to model M_3 (Fig. 1.3) is that of Figure 1.5. In addition to providing a language-neutral representation, the use of Petri nets allows us to reuse a large body of existing theoretical results, for example the theory of unfoldings [Enge 91, Niel 81].

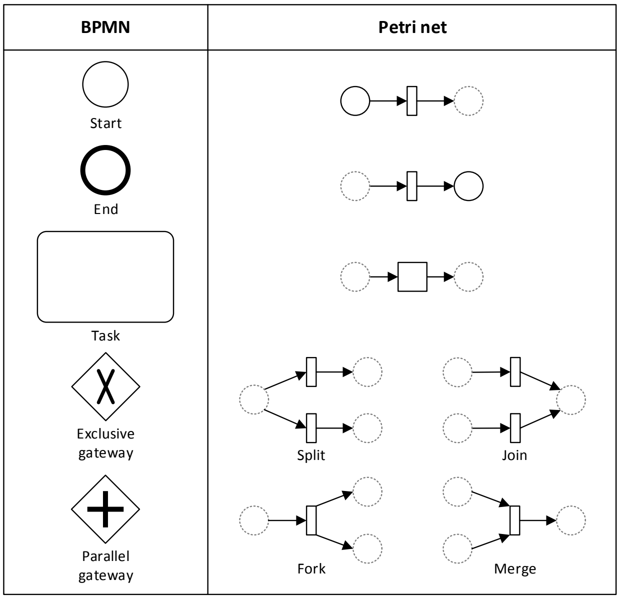


Figure 1.4: Mapping of tasks, events and gateways to Petri nets

[1]This transformation does not cover some BPMN constructs such as OR-joins, which cannot be straightforwardly translated into Petri nets [Favr 15].

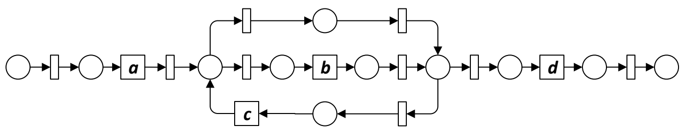


Figure 1.5: Petri net of process model M_3 (Fig. 1.3)

1.2 Contributions

The contributions of the thesis are the following.

- Unfolding technique of process models with cycles

  We propose an unfolding technique to compute a finite representation of a process model with cycles. The unfolding guarantees to capture all the causal dependencies between the tasks of a process model.

- Behavioral comparison of process models

  We propose a comparison technique based on event structures. Specifically, given a pair of process models, we compute their corresponding event structures, which describe the behavior in terms of events (occurrences of actions) and behavioral relations. Then another formalism, named as *partial synchronized product*, is used to determine the optimal matching between the behavior of the compared processes and, by the same token, to identify the behavioral differences. The adopted notion of equivalence is completed visible-pomset equivalence.

- Verbalization of differences as binary behavioral relations

  We present a method to detect and express behavioral differences as pairs of mismatching binary behavioral relations. The differences are detected in the partial synchronized product resulting from the comparison technique, whereas the verbalization uses the relations in the underlying event structures. The verbalization of the differences produces natural language statements using a set of predefined templates.

- (Deterministic) Reduction technique for asymmetric and flow event structures

  We propose behavior-preserving reduction techniques for asymmetric and flow event structures. The adopted equivalence notion is history preserving bisimulation. The reduced representation of an event structure can lead to more succinct diagnosis during the verbalization of the differences. Although, in general, there is no minimal representation of the behavior of a process using either asymmetric or flow event structures. Therefore, we define a deterministic order on the reduction operations to compute a canonical reduced representation.

- Implementation of the comparison technique, BP-Diff

  The proposed comparison technique has been implemented in a tool called BP-Diff. It is a web-based tool that takes pairs of process models in BPMN format and outputs the textual explanation and graphical representation of the differences.

1.3 Outline

State of the art techniques are discussed in Chapter 2. Specifically, we review comparison techniques based on three aspects of process models: tasks labels, structure and behavior.

The theoretical background of the thesis is presented in Chapter 3. The first part presents Petri nets and their semantics. Next, three variants of event structures are introduced: prime event structures, asymmetric event structures and flow event structures. Finally, we present the three behavioral equivalence notions used throughout the thesis.

Chapter 4 studies the expressive power of behavioral profiles, an existing formalism proposed for the behavioral representation of business process models. It is shown that while existing behavioral profiles can ensure a notion of equivalence for a restricted family of Petri nets, the interpretation of the relations can be vague and misleading in some scenarios. Finally, we present a set of counter examples where the notion of equivalence stops holding with existing behavioral profiles, i.e., when Petri nets contain silent transitions. The results of this chapter were published in [Arma 14e].

In Chapter 5, we present a behavioral comparison technique based on prime event structures. First, we propose an unfolding technique for process models with cycles. The unfolding constructs a finite representation of the behavior while capturing all possible causal dependencies between the tasks in the model. Using such representation the tasks that can occur more than once in a computation are distinguished from the ones that cannot. Second, in order to compute the similar and divergent behavior of a pair of processes, we introduce a, so called, partial synchronized product of (prime) event structures. Finally, we present a verbalization technique to produce natural language statements expressing encountered differences. However, we note that using other types of event structures, such as asymmetric and flow event structures, it is possible to provide smaller diagnosis. The results of this chapter were published in [Arma 14a, Arma 14c].

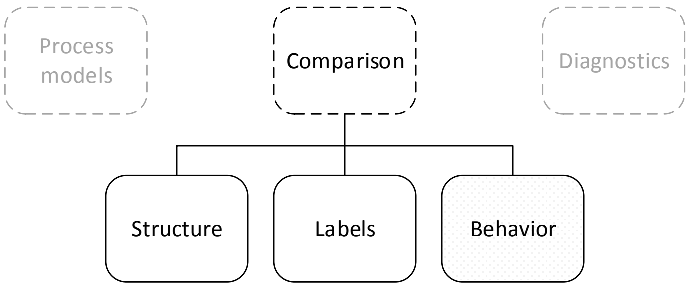
Chapter 6 presents reduction rules for asymmetric and flow event structures. The presented reduction techniques ensure the preservation of the behavior w.r.t. history preserving bisimulation. In general, there is not a single (minimal) representation for the behavior of a process, neither using asymmetric nor flow event structures. In the context of reduction of event structures, the order on which the reduction operations are applied can lead to non-isomorphic and non-reducible (equivalent) event structures with the same number of events. Thus, at the end of Chapter 6, we suggest a way to define a deterministic order on the reduction operations, such that the reduced version of an event structure is always the same. The results of this chapter were published in [Arma 14a, Arma 14d].

Chapter 7 provides an overview of BP-Diff, a prototype tool implementing the proposed technique. Additionally, it presents an evaluation of the tool using two libraries of real-life process models. The results of this chapter were published in [Arma 14b, Arma 14c].

Finally, Chapter 8 concludes this thesis and presents some future lines of research.

Chapter 2

State of the art



Existing comparison techniques for process models are based on three complementary aspects [Duma 09]: task labels, structure and behavior. The comparison of labels seeks an alignment among the tasks of a pair of process models. State-of-the-art techniques that approach this type of comparison are reviewed in Section 2.1. Next, we review structure-based comparison techniques in Section 2.2. Specifically, we review techniques that consider process models as labeled graphs and describe the differences as edit operations either over edges, nodes or both. Finally, Section 2.3 reviews behavior-based comparison techniques that focus at the comparison of the execution semantics of the process models.

2.1 Process model comparison based on task labels

The alignment between a pair of process models seeks for correspondences among their tasks [Duma 09]. Roughly, a task corresponds to another if both represent the same activity. An obvious way to define such alignment is to look at the labels attached to the tasks, such that if a pair of tasks have the same label then it is possible to define a 1:1 correspondence, aka elementary match, between them. Task correspondences can also be complex (non-elementary) 1:n matches where one task in one model can correspond to a set of n tasks in the other.

Tasks representing the same activity can have different, yet similar, labels attached, what makes more difficult the computation of a task alignment. In this regard, string similarity measures can mitigate the effect of having various ways to describe and name a single task. They can be defined with respect to the syntax or the semantics of a string. String edit distance [Leve 66] is an example of a syntactic similarity measure, and it counts the minimum string edit operations (delete, insert or substitute characters) necessary to transform one string into another. Conversely, a similarity measure based on semantics can be computed using a thesaurus, e.g, WordNet, that determines the semantic similarity between the words of a pair of strings. A thesaurus can help coping with the cases where a pair of labels are syntactically different, but they have a close meaning, e.g., "Send documents" and "Forward documents".

In the context of process model similarity, the majority of existing works have focused their attention on the computation of elementary matches between the tasks of the models. Different authors have used similarity measures between a pair of task labels based on syntax, semantics, or a combination of both, that is the case of the techniques presented in [van 08, Dijk 11, Duma 09, Ehri 07]; in such works the similarity of a pair of models is given by the combination of the similarities of their tasks labels.

A work addressing the problem of complex 1:n matches is [Weid 10]. The authors propose a framework comprising four different components: searchers, boosters, evaluators and selectors. In this framework, the similarity metrics to define matches among the tasks can use other aspects than string label similarity, e.g., they can also take into account the descriptions of the matched tasks, and structural or behavioral relations between them. The framework is then extended in [Leop 12, Klin 13] where probabilistic optimizations are integrated for the computation of the matches.

The alignment between a pair of process models can be seen as a preprocessing step for either a structural or behavioral comparison. In this work we acknowledge the importance of such alignment, but we assume that the correspondence between the tasks in the models is given. In the remaining of the thesis,we consider only elementary matches and the correspondence between tasks is denoted by the labels of the activities, i.e., if a pair of tasks represents the same activity then they have necessarily the same label.

2.2 Process model comparison based on model structure

Process models are annotated graphs where the flow relations are edges and the tasks, events, gateways, or any other element in the modeling language, are nodes. Then a structural comparison can rely on standard graph edit distance techniques to describe the differences between a pair of process models as edit operations [Mess 95], such as insert, delete and substitute nodes or edges. Such operations reflect the changes required in one graph in order to obtain the other.

The authors in [Ait 09] present a structure-based comparison technique for finite state machines (FSM). Roughly, given a pair of FSMs, the technique computes correspondences between their states and verify that the operations on one FSM are available on the other at every pair of matched states. Then the differences reflect the operations available in a FSM that are not available in the other, and they are expressed as addition, deletion and modification of operations (edges in the state machines).

Structural comparison techniques defined for business process graphs are presented in [Dijk 11, Dijk 09b, Yan 12, Dijk 09a]. Different from [Ait 09], these works define edit operations over both nodes and edges, thus the differences are expressed in terms of deletion, insertion and substitution of nodes, and deletion and insertion of edges. In [Dijk 09a, Yan 12], the authors present different heuristics that can cope with the complexity inherent to the graph edit distance technique (NP-complete [Mess 95]). Other related works include [Madh 04], where the authors propose a structural metric between process models based on similarity flooding [Meln 02].

The body of research in the field of label-based and structure-based comparison of process models has reached a certain level of maturity. A process model matching contest [Cayo 13] has been organized where a number of methods for process model comparison, based on both task labels and structure, have been pitched together. While a number of challenges remain, the current limitations of lexical and structural process model matching are understood.

As mentioned in Chapter 1, a fundamental limitation of structure-based comparison techniques is that a pair of process models can be structurally different, yet behaviorally equivalent. Conversely, a pair of process models can be very similar structurally, but they can entail completely dissimilar behavior. The next section reviews the techniques based on behavior.

2.3 Process model comparison based on behavior

A large amount of research has been devoted to the definition of equivalence notions for concurrent systems [Glab 89, Glab 90, Glab 01], ranging from trace equivalence to bisimulation equivalence, to finer equivalences in the true-concurrency (aka partial-order) semantics where the concurrent execution of tasks is taken into account. The adoption of one of those notions in the context of the comparison of behavior is crucial since it would establish the ground rules of the comparison. For example, the notions based on interleaving semantics deem as equivalent the concurrent and interleaved (sequential) execution of tasks; whereas, the notions based on true concurrency semantics would deem such case as inequivalent.

Perhaps one of the earliest works on diagnosing concurrent system differences is [Clea 91]. The author presents a technique to derive equations in a process algebra characterizing the differences between two *labeled transition systems* (LTSs). On the one hand, the use of a process algebra rather than a graphical language can make the feedback more difficult to grasp for end users (process analysts, in our context). On the other hand, the technique relies on interleaving bisimulation equivalence and does not take into account the concurrent structure of the process (a process model with concurrency and its interleaved version are equivalent). The authors in [Soko 06] present a method for assessing the dissimilarity of LTSs in terms of "edit" operations. This technique adopts a notion of equivalence that does not differentiate between the concurrent and the interleaved execution of tasks; whereas, the generated feedback does not tell the analyst what behavioral relations exist in one model that do not exist in the other. The same remarks apply to [Dijk 08a] that presents a method for diagnosing differences between pairs of process models using standard automata theory. In addition, this technique uses a fixed taxonomy of differences to identify the discrepancies between a pair of processes. As pointed out by the author, the taxonomy of differences is not guaranteed to be complete, and thus the technique might not report differences between inequivalent processes.

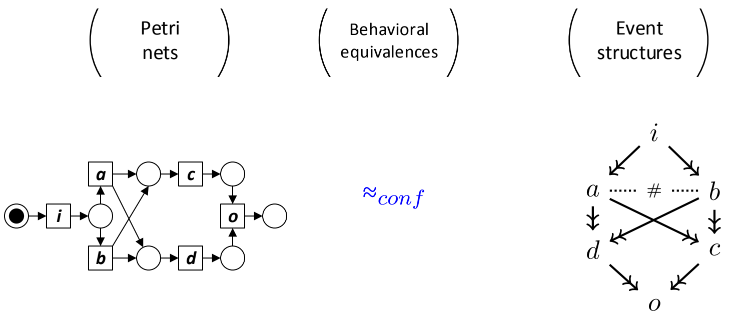
Behavioral profiles (BP) [Weid 11b] and *causal behavioral profiles* [Weid 11c] are two approaches that represent processes using binary relations. They abstract a process using a $n \times n$ matrix, where n is the number of tasks in the process. Each cell contains one out of three relations: *strict order*, *exclusive order* or *interleaving*; plus an additional *co-occurrence* relation in the case of causal behavioral profiles. Both techniques are incomplete as they mishandle several types of constructs, e.g., task skipping (silent transitions), duplicate tasks, and cycles. In this case, two processes can have identical BPs despite not being behaviorally equivalent in any standard sense (e.g., trace equivalent).

4C spectrum [Poly 14] is another family of binary relations that represents the behavior of a process in a $n \times n$ matrix. It offers a plethora of different relations and each cell in the matrix can contain more than one relation. However, 4C spectrum suffers from the same issues as the behavioral profiles because it does not guarantee any of the well-known notions of equivalence. Furthermore, the relations in this family can be difficult to interpret and may result counter intuitive to the analysts. In Chapter 4 we present a more extensive analysis, i.e., reach and limitations, of the behavioral comparison of process models using the relations in 4C spectrum and behavioral profiles.

Alpha relations [van 04] are another representation of processes using binary behavioral relations (direct causality, conflict and concurrency), proposed in the context of process mining. Direct causality however is not transitive (i.e., causality has a localized scope) and cannot capture so-called "short loops" and silent behavior [Bado 12]. *Relation sets* [Weid 12] are a generalization of alpha relations. Instead of one matrix, the authors use k matrices (with a variable k). In each matrix, causality is computed with a different look-ahead. It is shown that 1-look-ahead matrices induce trace equivalence for a restricted family of Petri nets. The authors claim that using k matrices improves accuracy. Nevertheless, it is unclear how human-readable diagnosis could be extracted from two sets of k matrices and to what notion of equivalence would this diagnostic correspond.

Chapter 3

Background



This chapter presents basic definitions and fixes the notation used in the remaining of the thesis. As a complement, a compilation of standard notions can be found in Appendix A.

Section 3.1 introduces fundamental notions on Petri nets, specifically, syntax, execution semantics and behavioral properties. By the same token, *families of pomsets* are presented as a generic model of concurrency applicable to different formalisms defining a notion of configuration. It is used later to formulate generic definitions, e.g., behavioral equivalence notions. Section 3.2 reviews basic definitions on event structures. We introduce three types of event structures: prime, asymmetric and flow event structures. Finally, different notions of equivalences in the true concurrency spectrum are presented in Section 3.3.

3.1 Petri nets

Petri nets [Petr 62] are a formal model for concurrent systems. It offers an intuitive graphical representation and a precise mathematical definition. Furthermore, additional key features of this model include its formally defined semantics and the availability of several analysis techniques. This section recalls the basics of Petri nets: graphical representation, notation and semantics.

Syntax of Petri nets

A Petri net is a directed graph with two types of *nodes*: transitions and places, and every arc in the net (aka flow relation) cannot connect two nodes of the same type. Intuitively, the transitions represent the tasks of a process, the places represent the states of the process and the order among the nodes is defined by means of flow relations. Typically, the transitions, places and flow relations in a Petri net are graphically represented with boxes, circles and directed arrows, respectively. For example, the Petri net in Figure 3.1 represents a process to file reports. The annotated boxes *Prepare report*, *Send report*, *Update documents*, *Update entry* and *Create new entry* are transitions, the annotated circles $p_1 - p_5$ are places, and the directed arrows are flow relations.

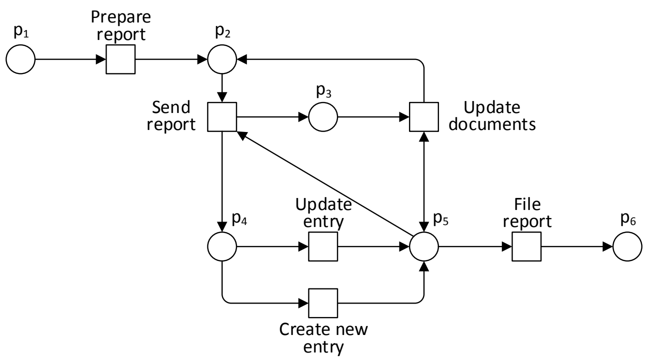


Figure 3.1: Messaging system modeled as a Petri net N

Formally, a Petri net is defined as follows.

Definition 3.1 (Petri net)**.** A *Petri net*, or a *net*, is a tuple $N = (P, T, F)$, where P and T are disjoint sets of *places* and *transitions*, respectively, and $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation; each element of F is an arc of the Petri net.

Throughout the thesis, we consider only finite Petri nets, i.e., nets with a finite number of places and transitions. The transitions of a Petri net can wear labels representing the activity that they represent. A transition wearing a label is called *observable*, otherwise *silent*. In the running example, Figure 3.1, all the transitions in the net are observable. A net with a labeling function from transitions to labels is called *labeled*. The formal definition of a net is extended below to consider the labeling function. Hereinafter, Λ stands for a fixed set of *tasks labels*.

Definition 3.2 (labeled Petri net)**.** A *labeled net* is the tuple (P, T, F, λ) where (P, T, F) is a net, and $\lambda : T \rightarrow \Lambda \cup \{\tau\}$ is a function that maps transitions to labels. Note that τ is a special label, $\tau \notin \Lambda$, and if $\lambda(t) = \tau$, where $t \in T$, then t is said to be *silent*; otherwise t is *observable*.

Oftentimes we will refer to the pre- (respectively, post-) set of a node. A node y is in the preset (respectively, postset) of another node x if there is a flow relation from y to x (respectively, from x to y). As an example, in Figure 3.1, p_4 is in the preset of *Create new entry*; whereas, p_3 and p_4 are in the postset of *Send report*. This intuition is formally captured in the next definition and extended to the pre- and postset of sets of nodes.

Definition 3.3 (pre- and postset of a (set of) node(s))**.** Let $N = (P, T, F)$ be a *Petri net* and $y \in P \cup T$ be a node in N. The *preset* of y is defined as ${}^\bullet y = \{x \in P \cup T \mid (x, y) \in F\}$; whereas the *postset* of y as $y^\bullet = \{z \in P \cup T \mid (y, z) \in F\}$.

Similarly, for a set of nodes $X \subseteq P \cup T$, ${}^\bullet X = \bigcup\{{}^\bullet x \mid x \in X\}$ and $X^\bullet = \bigcup\{x^\bullet \mid x \in X\}$.

The above concepts constitutes the syntax of the Petri nets. Different classes of nets have been defined by imposing restrictions at the syntactic level; further in this chapter we will present three classes of such nets, causal nets, occurrence nets and free choice nets. Now we turn our attention to the semantical aspect of Petri nets.

Semantics of Petri nets

Places are containers for *tokens* that stands for data, documents, messages or anything else that moves along the system. The cornerstone to formulate the behavior, or dynamics, of Petri nets is the firing rule, which establishes the circumstances under which a transition can occur and how the tokens are moved in the net. A token distribution of a net, aka *marking*, denotes a state during the execution of the system. Formally, a marking is a function from places to natural numbers including 0. E.g., the marking in Figure 3.2 associates p_1 and p_5 with 1, number of tokens (graphically represented as black filled circles) in those places, and p_2, p_3, p_4 and p_6 with 0.

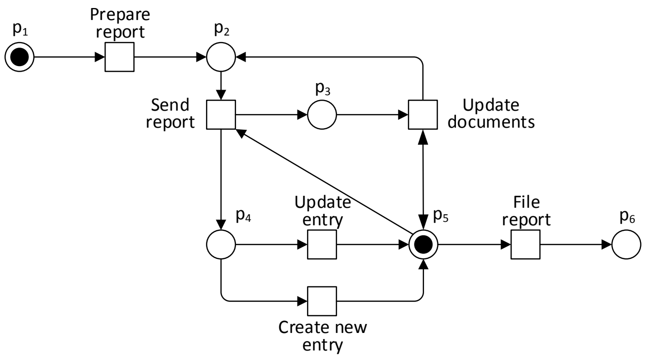


Figure 3.2: Petri net system $\mathcal{N}$ of N with a marking M_0

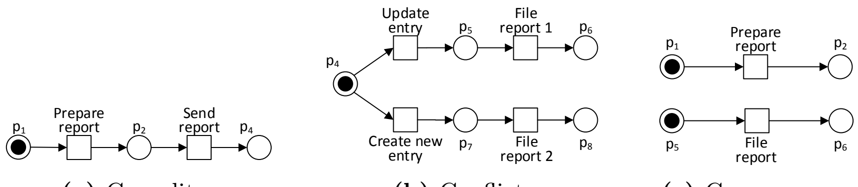
A Petri net equipped with a marking is called a Petri net system, e.g., Figure 3.2, and it is formalized in the next definition.

Definition 3.4 (marking, (labeled) Petri net system)**.** Given a Petri net $N = (P, T, F)$, a *marking* of N is a function $M : P \rightarrow \mathbb{N}_0$ that assigns a non-negative number to every place $p \in P$. A *Petri net system*, or simply a *net system*, $\mathcal{N} = (N, M)$ is the net $N = (P, T, F)$ with a *marking* M. The initial marking of a net is denoted as M_0. Finally, a net system (N, M) is *labeled* if N is labeled.

A transition is said to be *enabled* if every place in its preset has at least one token. In simple words, if a transition is enabled then it means that there exist the necessary resources to execute the corresponding activity, and thus it can occur. The *firing* of an enabled transition removes one token from every place in its preset, and sets one token to every place in its postset. The firing of a transition produces an *event* (an occurrence of an activity). In the running example, Figure 3.2, *Prepare report* and *File report* are the only enabled transitions, and in order to enable, for instance, *Send report* then it is necessary to fire *Prepare report* first.

Formally, a marking M of a net $N = (P, T, F)$ *enables* a transition $t \in T$, denoted as $M[t\rangle$, iff $\forall p \in {}^{\bullet}t : M(p) > 0$. Moreover, the occurrence (firing) of t leads to a new marking M', denoted as $M \xrightarrow{t} M'$, computed according to the following rules for every place $p \in P$:

$$M'(p) = \begin{cases} M(p) - 1 & \text{if } p \in {}^{\bullet}t \setminus t^{\bullet} \\ M(p) + 1 & \text{if } p \in t^{\bullet} \setminus {}^{\bullet}t \\ M(p) & \text{otherwise} \end{cases}$$



(a) Causality **(b)** Conflict **(c)** Concurrency

Figure 3.3: Net systems exemplifying causal, conflict, and concurrent relations between events

Implicitly a Petri net system represents different dependency relations between the events. For example, in Figure 3.3a, *Send report* can occur only after *Prepare report*, in which case the produced events are said to be in **causal** relation. In Figure 3.3b, *Update entry* and *Create new entry* are both enabled and the occurrence of one disables the occurrence of the other, thus the corresponding events are said to be in **conflict**. Finally, Figure 3.3c shows the case when *Prepare report* and *File report* can occur independently, i.e., the events are **concurrent**.

Using the firing rule of transition as a basis, we present techniques to describe the behavior of the Petri net models, namely sequential semantics and causal semantics.

Sequential semantics The sequential semantics describes the behavior of a Petri net by means of transition sequences that can be executed by the net. Such sequences are called *firing sequences*. A single transition can appear multiple times in a firing sequence, e.g., when the net has cycles, and thus several events corresponding to the same transition are produced. Firing sequences produce totally ordered sets of events and the concurrent execution of transitions is represented as arbitrary interleaving. The formal definition of a firing sequences is as follows.

Definition 3.5 (firing sequence)**.** Let $\mathcal{N} = (N, M_0)$, $N = (P, T, F)$, be a Petri net system. A sequence of transitions $\sigma = t_1 \dots t_n$ in T, where $n \in \mathbb{N}_0$, is a *firing sequence* in $\mathcal{N}$ iff σ is empty or it holds that $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \dots \xrightarrow{t_n} M_n$. In the latter case, we say that σ leads from M_0 to M_n and denote by $M_0[\sigma\rangle M_n$.

A marking M where there is no enabled transition is called *terminal*.

Definition 3.6 (terminal marking)**.** Let $\mathcal{N} = (N, M_0)$, $N = (P, T, F)$, be a Petri net system. A marking M of $\mathcal{N}$ is *terminal* iff there exist no transition enabled at M.

A firing sequence leading from an initial to a terminal marking is called an *execution*. For example, consider the net system with initial marking M_0

in Figure 3.2, and let σ be a firing sequence consisting of the transitions: *Prepare report*, *Send report*, *Create new entry* and *File report*. Then σ leads from M_0 to M_x, marking displayed in Figure 3.4. It is easy to check that M_x is terminal since there is no enabled transition, and therefore σ is an execution. This intuition is captured by the next definition.

Definition 3.7 (execution)**.** Let $\mathcal{N} = (N, M_0)$, $N = (P, T, F)$, be a Petri net system. A firing sequence σ that leads from M_0 to M, where M is terminal, is called an *execution*. By $\Delta(\mathcal{N})$, we denote the set of all executions of $\mathcal{N}$.

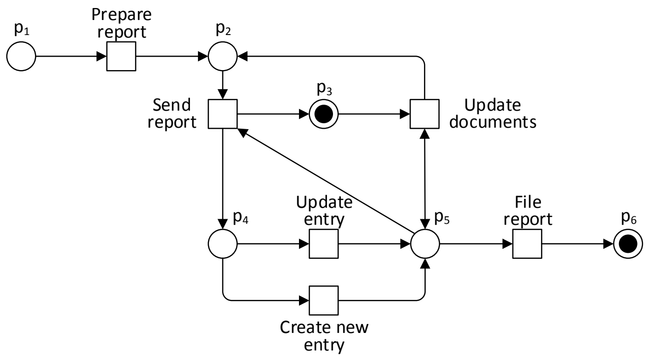


Figure 3.4: Petri net system $\mathcal{N}$ with a terminal marking M_x

Causal semantics Another technique to describe the behavior of a Petri net, besides sequential semantics, is via *causal nets*; in the literature they are also called *process nets*. This type of nets defines some syntactical restrictions, i.e., in a causal net the transitive closure of the flow relation is a partial order (there are no cycles) and the cardinality of the pre- and postset of every place is at most one. Then the causal semantics of a Petri net is composed by the causal nets representing the executions of the system.

Causal nets describe the partial order semantics of a net system. A causal net represents two relations between the nodes: causality and concurrency. Let us define first the behavioral relations before proceeding with the formal definition of causal nets.

Definition 3.8 (causality and concurrency relation)**.** Let $N = (P, T, F)$ be a Petri net and $x, y \in P \cup T$ two nodes in N. Then

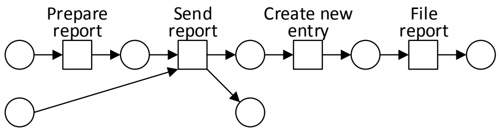
- x is a *cause* of y, denoted $x <^N y$, if $(x, y) \in F^+$. The *inverse causal* relation is denoted $>^N$. By $\leq^N$ we denote the *reflexive causal* relation.
- x and y are *concurrent*, written as $x \parallel^N y$, if $\neg(x <^N y)$ and $\neg(y <^N x)$.

Then the causal nets are formally defined as follows.

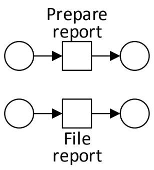
Definition 3.9 (causal net)**.** A causal net is a Petri net $N = (P, T, F)$ where:

1. $|{}^\bullet p| \leq 1$ for any $p \in P$,
2. $|p^\bullet| \leq 1$ for any $p \in P$, and
3. $\leq^N$ is a partial order.

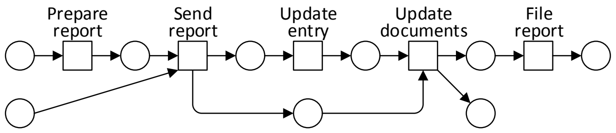
Figure 3.5 shows three different causal nets representing three executions of the net system in Figure 3.2. Note that *Prepare report* $<^{N'}$ *File report* in Figure 3.5a; whereas *Prepare report* $\parallel^{N''}$ *File report* in Figure 3.5b.



(a) N'



(b) N''



(c) N'''

Figure 3.5: Example of causal nets

Behavioral properties of Petri net systems

Petri net systems have a number of behavioral properties. We review the reachability and coverability of markings, and safeness, liveness, and boundedness of the net systems. A marking M is reachable if there is a firing sequence that leads from the initial marking M_0 to M.

Definition 3.10 (reachable marking)**.** Let $\mathcal{N} = (N, M_0)$, $N = (P, T, F)$, be a Petri net system. A marking M is *reachable* in $\mathcal{N}$ iff $M = M_0$ or there exist a firing sequence σ that leads from M_0 to M. The notation $M' \in [N, M\rangle$ represents that M' is reachable in (N, M).

A marking M is *coverable* if there is another reachable making associating every place with a larger (w.r.t. M) number of tokens.

Definition 3.11 (coverable marking)**.** Let $\mathcal{N} = (N, M_0)$, $N = (P, T, F)$, be a Petri net system. A marking $M \in [N, M_0\rangle$ is *coverable* if there exist another marking $M' \in [N, M_0\rangle$ such that $M'(p) \geq M(p)$ for every $p \in P$.

A desirable property for many analysis applications using Petri nets is the finiteness of the state space of makings. Such property is met if a Petri net can contain up to a fixed $n \in \mathbb{N}_0$ number of tokens at any reachable marking, in such case the net is said to be n-bounded.

Definition 3.12 (boundedness)**.** Let $\mathcal{N} = (N, M_0)$, $N = (P, T, F)$, be a Petri net system. A marking $M \in [N, M_0\rangle$ is *n-bounded* iff every place $p \in P$ contains up to $n \in \mathbb{N}_0$ tokens at M, i.e., $M(p) \leq n$. $\mathcal{N}$ is *n-bounded* iff all reachable markings are *n-bounded*.

If a net system is 1-bounded then it is called *safe*.

Definition 3.13 (safeness)**.** Let $\mathcal{N} = (N, M)$, $N = (P, T, F)$, be a Petri net system. A net system $\mathcal{N}$ is *safe* if it is 1-*bounded*.

Intuitively, a net system is *live* if every transition can always occur again from any reachable marking M.

Definition 3.14 (liveness)**.** Let $\mathcal{N} = (N, M_0)$, $N = (P, T, F)$, be a Petri net system. $\mathcal{N}$ is *live* iff for every reachable marking $M \in [N, M_0\rangle$ and for every transition $t \in T$ there exist a marking $M' \in [N, M\rangle$ such that $M'[t\rangle$.

Throughout the thesis, we restrict the discussions to safe net systems. A safe marking M of a net (P, T, F) is identified as the set of places $\{p \in P \mid M(p) = 1\}$.

3.1.1 Petri net subclasses

In the existing literature different classes of Petri nets have been put forward defining structural and/or semantical restrictions. Throughout the thesis we consider four different classes of Petri nets, namely *Free-choice* and *Occurrence* nets – which impose structural restrictions–, and (sound) *Workflow* and *Flow* nets – which impose semantical restrictions. Each of the considered classes of Petri nets is presented next. Some of the results presented in this thesis are formulated for specific classes of nets, nevertheless, if the class of nets is not specified, then it shall be assumed that the only imposed restriction is safeness.

Structural restrictions

Free-choice nets Free-choice Petri nets [Best 87, Dese 95] are a family of nets with specific structural restrictions. The characteristics of this kind of nets allow efficient verification techniques for several properties, which are hard to check for general Petri nets. In a free-choice net whenever two places share a transition in their postsets, then they share all transitions in their postsets. Thus, choices are free in this net, since they are not influenced by the rest of the system. For instance, the classic case of a non-free choice Petri net is depicted in Figure 3.6. In this example, one can observe that *Send report* and *File report* are not in conflict (*Send report* is not enabled), unless *Prepare report* is fired before *File report*.

Formally a free-choice Petri net is defined as follows.

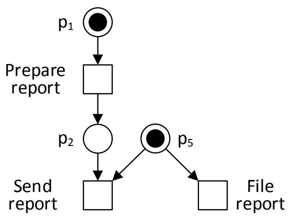


Figure 3.6: Non-free choice Petri net

Definition 3.15 (free-choice Petri net)**.** A Petri net N is free-choice if for any pair of places $p_1, p_2 \in P$ then either $p_1^\bullet \cap p_2^\bullet = \varnothing$ or $p_1^\bullet = p_2^\bullet$.

Occurrence nets Occurrence nets were introduced in [Niel 81], and they can be seen as a generalization of causal nets where a form of (forward) conflict is allowed. In this type of nets, places and transitions are often referred to as conditions and events, respectively. Among of the syntactical restrictions imposed by this type of nets is that every condition has up to one event in its preset, but it can have any number of events in its postset. Occurrence nets are acyclic and the transitive closure of the flow relation is a partial order. Different from causal nets, occurrence nets define three behavioral relations between the nodes: causality, concurrency and conflict. Let us first present the three behavioral relations and, even though, the definition for the causal relation is the same as the one in Definition 3.8, we include it below for completeness.

Definition 3.16 (causality, conflict and concurrency relation)**.** Let $N = (P, T, F)$ be a Petri net and $x, y \in P \cup T$ two nodes in N. Then

- x is a *cause* of y, denoted $x <^N y$, if $(x, y) \in F^+$. The *inverse causal* relation is denoted $>^N$. By $\leq^N$ we denote the *reflexive causal* relation.
- x and y are in *conflict*, denoted $x \#^N y$, if $x, y \in T \cup P$ are distinct nodes and $\exists t_1, t_2 \in T$, such that (1) $t_1 \neq t_2$, (2) ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \varnothing$, and (3) $t_1 \leq^N x$ and $t_2 \leq^N y$.
- x and y are *concurrent*, denoted as $x \parallel^N y$, if $\neg(x <^N y)$, $\neg(y <^N x)$ and $\neg(x \#^N y)$.

Then an *occurrence net* can be formally defined as follows.

Definition 3.17. A net $N = (P, T, F)$ is an *occurrence net* iff:

1. $|{}^\bullet p| \leq 1$ for every condition $p \in P$
2. N is acyclic, i.e., the causal relation $(\leq^N)$ is a partial order
3. The set $\{y \in P \cup T \mid y <^N x\}$ is finite for every $x \in P \cup T$
4. $\#^N$ is irreflexive, i.e., $\neg(x \#^N x)$ for any $x \in P \cup T$

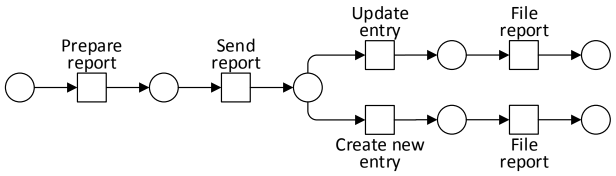


Figure 3.7: Occurrence net N_1

The net in Figure 3.7 is an occurrence net and, for example, it is easy to note that *Prepare report* $<^{N_1}$ *Send report*, and *Update entry* $\#^{N_1}$ *Create new entry*.

Behavioral restrictions

Workflow nets A class of Petri nets defining both semantical and structural restrictions is (sound) Workflow nets [van 97], shorthanded as WF-nets. The imposed syntactical restrictions include the definition of a dedicated source and sink place, and the property that every transition is on a path from the source to the sink place. Figure 3.8 shows a WF-net system version of the process to file reports. In this example, p_1 is the source place and p_6 is the sink place. Furthermore, it is easy to check that every transition is on a path from p_1 to p_6.

The formal definition of a WF-net (system) is given below.

Definition 3.18 (WF-net, WF-net system)**.** A Petri net $N = (P, T, F)$ is a *workflow net*, or a *WF-net*, iff N has a dedicated *source* place $i \in P$, with ${}^\bullet i = \varnothing$, N has a dedicated *sink* place $o \in P$, with $o^\bullet = \varnothing$, and the *short-circuit*

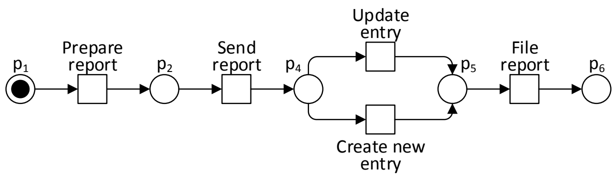


Figure 3.8: (sound) Workflow net system

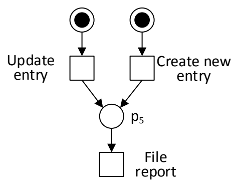
net $N^* = (P, T \cup \{t^*\}, F \cup \{(o, t^*), (t^*, i)\})$ of N is strongly connected, s.t., $t^* \notin T$. A *WF-net system* is a net system (N, M), where N is a WF-net with the source place i and $M = \{i\}$.

The commonly adopted criterion of correctness for WF-net systems is soundness [Aals 00]. It is a behavioral restriction guaranteeing that every execution of a WF-net system ends with one token in the sink place and no tokens elsewhere. Indeed, the WF-net system in Figure 3.8 is sound. Below we provide the formal definition of soundness for WF-nets.

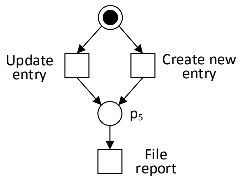
Definition 3.19 (soundness)**.** Let $\mathcal{N} = (N, M)$, $N = (P, T, F)$, be a WF-net system. $\mathcal{N}$ is *sound* iff the net system (N^*, M), where N^* is the short-circuit net of N, is live and bounded.

Flow nets Another behaviorally restricted class of Petri nets is Flow nets [Boud 90]. Transitions and places are referred to as events and conditions, respectively. Flow nets are semantically acyclic, meaning that in any firing sequence a token cannot return to a place that was previously used as a precondition. Thus, all the transitions in a firing sequence are distinct. Interestingly, any occurrence net is a flow net.

The places in flow nets define causal dependencies between transitions. Intuitively, we say that a transition t_j causally depends on another transition t_i if there is a place p between them, and whenever both transitions appear in a firing sequence then t_i is the only transition that puts a token in p; then p is said to be a *strong postcondition* of t_i. In general, flow nets



(a) Non-flow net system



(b) Flow net system

Figure 3.9: Non-flow and flow net example

are not required to be safe, but it is necessary that the causal dependencies between transitions are unambiguous.

Figure 3.9a shows an example of a Petri net that is not a flow net. Observe that *Update entry* and *Create new entry* can occur in the same firing sequence, and thus it is not possible to determine what event precedes an occurrence of *File report*. Specifically, when p_5 has two tokens, it is not possible to determine whose token will be consumed once *File report* is fired. Conversely, the Petri net displayed in Figure 3.9b is a flow net. In this flow net, either *Update entry* or *Create new entry* puts a token in the condition p_5 in any firing sequence. Therefore, it is clear what event precedes an occurrence of *File report*. Then p_5 is a strong postcondition of *Update entry* and *Create new entry*.

The formal definition of strong postcondition is presented next and complements the informal description provided above.

Definition 3.20 (string postcondition)**.** A place $p \in P$ of a net system $\mathcal{N} = (N, M)$, where $N = (P, T, F)$, is a *strong postcondition* of $t \in T$ if the following holds

1. $p \in t^\bullet$, and
2. for any firing sequence $\sigma = t_1 \dots t_n$, such that $t_i = t$, $n \in \mathbb{N}_0$ and $1 \leq i \leq n$, then $M(p) + |F'| = 1$, where $F' = \{(t_j, p) \in F\}$ for any $1 \leq j \leq n$.

Let $\psi(t, t')$ denote the set of strong postconditions between a pair of transitions $t, t' \in T$, i.e., $\psi(t, t') \subseteq t^\bullet \cap {}^\bullet t'$.

The formal definition of flow nets is given next.

Definition 3.21 (flow net, flow net system)**.** A net system $\mathcal{N} = (N, M)$, $N = (P, T, F)$, is a *flow net system* and N is a *flow net* iff for every firing sequence $\sigma = t_1 t_2 \ldots t_n$ in $\mathcal{N}$ and for every $i, j \in \mathbb{N}_0$, s.t. $1 \le i < j \le n$, it holds that:

- $p \in P$ cannot be used as a precondition more than once in a firing sequence σ, i.e., ${}^\bullet t_i \cap {}^\bullet t_j = \varnothing$, and
- if $t_i{}^\bullet \cap {}^\bullet t_j \ne \varnothing$ then $\exists\, p \in t_i{}^\bullet \cap {}^\bullet t_j \;:\; p \in \psi(t_i, t_j)$.

Flow nets are a class of nets defining a notion of configuration, which describe the partial order semantics of a net system. A configuration can be understood as sets of events that can occur in the same execution. In flow nets, firing sequences and configurations are in close relation due to the next definition.

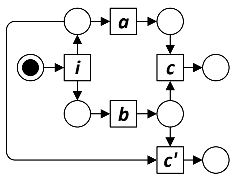
Definition 3.22 (configuration of flow net)**.** A configuration of a flow net system $\mathcal{N} = (N, M)$, $N = (P, T, F)$, is a subset $C \subseteq T$ of transitions in N, such that there exist a firing sequence σ in $\mathcal{N}$ that consists of the transitions in C, i.e.,

$$\sigma = t_1 t_2 \ldots t_n \quad \text{and} \quad C = \{t_1, t_2, \ldots, t_n\}$$

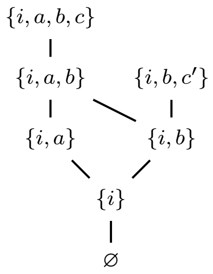
The set of all configurations of a flow net system $\mathcal{N}$ is denoted by $\mathit{Conf}(\mathcal{N})$.

Set inclusion ($\subseteq$) defines an order over the flow net configurations. We say that a configuration C *evolves into* a configuration C' if $C \subseteq C'$. Figure 3.10 shows a flow net system and its configurations ordered by inclusion.

Furthermore, using the notion of configuration, it is possible to define two relations between pairs of events in a flow net: *flow* and *conflict*. A pair of events is in flow relation if they can occur in a firing sequence and there is a strong postcondition between them. On the other hand, a pair of events is in conflict relation if they never occur in the same configuration. The formal definition of both relations, flow and conflict, is provided below.



(a) Flow net system $\mathcal{N}_1$



(b) $Conf(\mathcal{N}_1)$

Figure 3.10: Flow net system and its configurations ordered by set inclusion

Definition 3.23 (conflict and flow relation)**.** Let $\mathcal{N} = (N, M)$, $N = (P, T, F)$, be a *flow net system* and $x, y \in P \cup T$ be two events in N. Then

- x and y are in conflict, denoted as $x \#^N y$, if for all configurations $C \in Conf(\mathcal{N})$ then $\{x, y\} \nsubseteq C$.
- x is a potential cause of y, denoted as $x \prec^N y$ and referred to as *flow*, if $\neg(x \#^N y)$ and $\psi(x, y) \neq \varnothing$.

Note that the presented notion of configuration for flow nets is not applicable to any class of nets. In general, a configuration can contain several occurrences of a single transition (e.g., in the presence of cycles) thus it would not be a set, but a multi-set. Next we introduce the *branching process* of a net system. Intuitively, it represents the unfolding of a net system as an occurrence net that captures the partial order semantics of the system. Furthermore, in this structure, it is possible to have a notion of configurations as sets of events.

3.1.2 Branching process of a Petri net system

The causal semantics of a net can be described as runs or, more precisely, prefixes of runs, by means of *causal nets*. Alternatively, several (possibly all) runs can be accommodated in a single tree-like structure, called *branching process* [Enge 91, Niel 81]. A branching process is an occurrence net and so describes three behavioral relations between the nodes: causality, conflict and concurrency. We next provide a formal definition of branching process and unfolding of a net system.

Definition 3.24 (unfolding, branching process)**.** Let $\mathcal{N} = (N, M_0)$ be a net system and $N = (P, T, F)$ be a net. The *unfolding* of $\mathcal{N}$ is the tuple $\mathit{Unf}(\mathcal{N}) = (B, E, G, \rho)$, where (B, E, G) is an occurrence net generated by the inductive rules in Figure 3.11, and a homomorphism $\rho : B \cup E \to P \cup T$, such that for every $e_1, e_2 \in E$, if ${}^\bullet e_1 = {}^\bullet e_2$ and $\rho(e_1) = \rho(e_2)$ then $e_1 = e_2$.

We call *branching process* of $\mathcal{N}$ any prefix of the unfolding, i.e., any tuple $\beta = (B', E', G', \rho')$ such that $B' \subseteq B$, $E' \subseteq E$, for any $e \in E'$, $\lfloor e \rfloor \subseteq E'$ and ${}^\bullet e, e^\bullet \subseteq B'$, and G',ρ' are the obvious restriction of G and ρ.

$$\frac{p \in M_0}{b := \langle \varnothing, p \rangle \in B \qquad \rho(b) := p} \qquad\qquad \frac{t \in T \qquad X \subseteq B \qquad X^2 \subseteq \parallel \qquad \rho(X) = {}^\bullet t}{e := \langle X, t \rangle \in E \qquad {}^\bullet e := X \qquad \rho(e) := t}$$

$$\frac{e = \langle X, t \rangle \in E \qquad t^\bullet = \{p_1, \dots, p_n\}}{b_i := \langle e, p_i \rangle \in B \qquad e^\bullet := \{b_1, \dots, b_n\} \qquad \rho(b_i) := p_i}$$

Figure 3.11: Branching process, inductive rules

Intuitively the unfolding $\mathit{Unf}(\mathcal{N})$ represents any possible behavior of the net system, while a generic branching processes represents a subset of the possible computations.

As stated in the definition of occurrence nets, an unfolding and a branching process do not contain merging conditions. As a result, some nodes in a net system need to be represented more than once in the corresponding branching process. Figure 3.12 shows a net system and its unfolding. The shaded areas represent the relations in the function ρ and it is easy to note that p_5, *File report* and p_6 are related to multiple nodes in the unfolding.

In what follows we provide some formal definitions related to branching processes that will be used later.

Definition 3.25 (Basic notions on branching processes)**.** Let $\beta = (B, E, G, \rho)$ be a branching process.

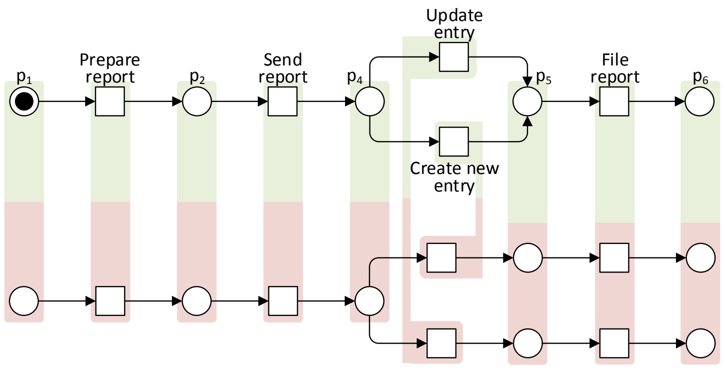


Figure 3.12: Petri net system and its unfolding

- A *configuration* C of β is a set of events, $C \subseteq E$, which is (i) causally closed, i.e., $\forall e' \in E, e \in C : e' \leq^{\beta} e \Rightarrow e' \in C$, and (ii) conflict free, i.e., $\forall e, e' \in C, \neg(e \#^{\beta} e')$. We denote by $Conf(\beta)$ the set of configurations of the branching process β and by $MaxConf(\beta)$ the subset of maximal configurations w.r.t. set inclusion.
- The *local configuration* of an event $e \in E$ is its set of causes $\lfloor e \rfloor = \{e' \mid e' \leq e\}$. The set of strict causes of an event $e \in E$ is $\lfloor e) = \lfloor e \rfloor \backslash \{e\}$.
- A *deterministic process* $\pi = (B_\pi, E_\pi, G_\pi, \rho)$ is the net induced by a configuration C, where $B_\pi = \bigcup\limits_{c \in C} ({}^{\bullet}c \cup c^{\bullet})$, $E_\pi = C$, and $G_\pi = G \cap (B_\pi \times E_\pi \cup E_\pi \times B_\pi)$.

For a condition $b \in B$ we will write $\lfloor b \rfloor$ as a shorthand for $\lfloor {}^{\bullet}b \rfloor$.

The set $Min(\beta)$ of minimal elements of $B \cup E$ with respect to causality corresponds to the set of places in the initial marking of $\mathcal{N}$, i.e., $\rho(Min(\beta)) = M_0$. A *co-set* is a set of conditions $B' \subseteq B$ such that for all $b, b' \in B'$ it holds $b \ \|^{\beta} \ b'$. A *cut* is a maximal co-set w.r.t. set inclusion.

The target cut for a configuration $C \in Conf(\beta)$ is defined as

$$Cut(C) = (Min(\beta) \cup \bigcup_{c \in C} c^{\bullet}) \backslash (\bigcup_{c \in C} {}^{\bullet}c).$$

The image of $Cut(C)$ in $\mathcal{N}$, $\rho(Cut(C))$, is a reachable marking in $\mathcal{N}$ denoted by $Mark(C)$. Let C and C' be configurations of β, such that

$C \subset C'$, and let π and π' be their corresponding deterministic branching processes. If $X = C' \setminus C$, then we write $\pi' = \pi \oplus X$ and we say that π' is an *extension* of π.

3.1.3 Configurations and families of pomsets

An alternative way to define the execution semantics of a net system is using a notion of configurations as in the case of flow nets and branching processes. Different from firing sequences, which describes the interleaving semantics of a net system, configurations describe the partial order semantics. A configuration C of a net system is a subset of events, occurrences of actions, that represents a state of the system, i.e., the state in which the events in C have occurred.

In order to have a more uniform presentation of the different formalisms used throughout the thesis, which define a notion of configuration, we introduce *families of pomsets* in the line of [Rens 92, Glab 96, Glab 95]; note that in [Arma 14d] we refer to this concept as an *abstract event structure*.

A *pomset* is a tuple $\langle X, \leq_X, \lambda_X \rangle$, where X is a set of events, $\leq_X$ is a partial order and λ_X is a labeling function. An *isomorphism* of pomsets X and Y is an isomorphism between the underlying sets, which respects labels and order, i.e., a bijection $f : X \to Y$ such that, $\lambda_X = \lambda_Y \circ f$, and $e <_X e' \Leftrightarrow f(e) <_Y f(e')$ for all $e, e' \in X$.

A configuration C can be seen as a pomset, where the elements in C are events and there is a partial order $\leq_C$ and a labeling function $\lambda_C : C \to \Lambda \cup \{\tau\}$. Then C will used interchangeably for both the configuration and its corresponding pomset. For a configuration C, we denote by $C^{\Lambda} = \{e \in C \mid \lambda(e) \neq \tau\}$ the subset of visible events in C or the corresponding pomset, which is called the *visible pomset* underlying C.

Definition 3.26 (family of pomsets)**.** A *family of pomsets* is a triple $\mathcal{P} = \langle E, \mathit{Conf}(\mathcal{P}), \lambda \rangle$ where E is a set of events, $\mathit{Conf}(\mathcal{P})$ is a set of configurations and $\lambda : E \to \Lambda \cup \{\tau\}$ is a labelling function. Each configuration consists of a set of events $C \subseteq E$, endowed with a partial order $\leq_C$ called the *local order* of C.

The relation $\leq_C$ associated with a configuration C intuitively represents the order in which the events in C can occur. A configuration will be often denoted simply by C, leaving the partial order $\leq_C$ implicit.

Definition 3.27 (extension order)**.** Let $\mathcal{P} = \langle E, \mathit{Conf}(\mathcal{P}), \lambda \rangle$ be a family of pomsets. The set of configurations $\mathit{Conf}(\mathcal{P})$ is endowed with the *extension order* defined as $C_1 \sqsubseteq C_2$ whenever $C_1 \subseteq C_2$, $\leq_{C_1} = \leq_{C_2} \cap (C_1 \times C_1)$ and for all $e_1 \in C_1$, $e_2 \in C_2$, if $e_2 \leq_{C_2} e_1$ then $e_2 \in C_1$.

Intuitively, $C_1 \sqsubseteq C_2$ means that the configuration C_1 can evolve into C_2 by executing the events in $C_2 \setminus C_1$. In fact, C_1 is required to be a subset of C_2, with events ordered exactly as in C_2 and the new events in $C_2 \setminus C_1$ cannot precede events already in C_1. Moreover, we denote by $\mathit{Conf}(\mathcal{P})^{\Lambda}$ the set of visible pomsets underlying a set of configurations of a family of pomsets, i.e., $\mathit{Conf}(\mathcal{P})^{\Lambda} = \{C^{\Lambda} : C \in \mathit{Conf}(\mathcal{P})\}$. Furthermore, $\mathit{MaxConf}(\mathcal{P})$ denotes the subset of maximal pomsets w.r.t. set inclusion, and $\mathit{MaxConf}(\mathcal{P})^{\Lambda}$ the underlying visible pomsets.

Flow nets can be seen as an instance of a family of pomsets. The order of each configuration $C \in \mathit{Conf}(\mathcal{N})$, of a flow net system $\mathcal{N} = (N, M)$, is given by $(<^{N}_{|C})^{*}$. As specified above, the extension order is simply subset-inclusion, so for $C_1, C_2 \in \mathit{Conf}(\mathcal{N})$, we have $C_1 \sqsubseteq C_2$, iff $C_1 \subseteq C_2$. Moreover, in the case of flow nets, if $e_i \in C_1$, $e_j \in C_2$ and $e_j \leq_{C_2} e_i$, then necessarily $e_j \in C_1$, this property is captured in the following proposition.

Proposition 3.28. *Let $\mathcal{N} = (N, M)$ be a flow net system and let $C_1, C_2 \in \mathit{Conf}(\mathcal{N})$ be a pair of configurations, such that $C_1 \subseteq C_2$. Then, for any pair of events $e_i \in C_1$, $e_j \in C_2$, if $e_j \leq_{C_2} e_i$ then necessarily $e_j \in C_1$.*

Proof. By the definition of configurations of flow nets (Def. 3.22), there is a firing sequence σ_2 consisting of the transitions of $C_2 = \{e_i, \ldots, e_n\}$, i.e., $\sigma_2 = e_i \ldots e_n$, where $n \in \mathbb{N}_0$, such that $1 \leq j < i \leq n$. Thus, by induction on the distance from e_j to e_i in σ_2, consider the base case when the length 0 and so $e_j \prec^N e_i = e_{j+1}$. By the definition of flow relation (Def. 3.23), $\exists p \in e_j{}^\bullet \cap {}^\bullet e_i$. First, note that ${}^\bullet e_i$ cannot be part of the initial marking M, i.e., ${}^\bullet e_i \nsubseteq M$, because the places in ${}^\bullet e_i$ would be marked twice in C_2, one during the initial marking and another time after the occurrence of e_j, and thus the places in ${}^\bullet e_i$ would be used as preconditions twice in a firing sequence. The last contradicts Definition 3.21. Then by the definition of flow nets (Def. 3.21), let p be a strong postcondition of e_j. Suppose that $e_j \notin C_1$, and since ${}^\bullet e_i \nsubseteq M$, $\exists e_3 \in C_1 : e_3{}^\bullet \subseteq {}^\bullet e_i \wedge p \in e_j{}^\bullet \cap e_3{}^\bullet$, but as $C_1 \subseteq C_2$ then $e_3 \in C_2$. Although, p would not be a strong postcondition of e_j and it contradicts the assumptions. The inductive step follows accordingly. $\square$

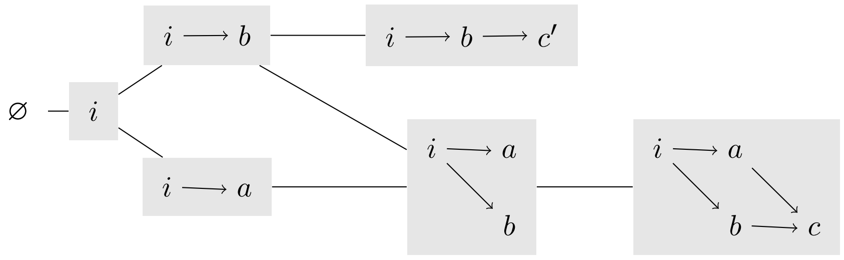


Figure 3.13: Family of pomsets ordered by inclusion, i.e., pomsets of flow net system in Fig. 3.10a.

Figure 3.13 shows an example of a family of pomsets ordered by inclusion, it corresponds to the flow net system in Fig. 3.10.

Furthermore, it should be clear that a branching process $\beta = (B, E, G, \rho)$ of a net system is also an instance of a family of pomsets, where each configuration $C \in \mathit{Conf}(\beta)$ is ordered by $\leq^{\beta}_{|C}$. The extension order is simply subset-inclusion. Furthermore, by Definition 3.25, the configurations of a branching process are causally closed, and thus given a pair of configurations $C_1, C_2 \in \mathit{Conf}(\beta) : C_1 \subseteq C_2$, if $e_1 \in C_1$, $e_2 \in C_2$ and $e_2 \leq^{\beta}_{|C_2} e_1$, then $e_2 \in C_1$.

3.2 Event structures

Event structures (ES) are another formalism for modeling concurrent processes. The seminal work [Wins 87, Niel 81] introduces event structures as intermediate representations that connect Petri nets and domains. In an event structure, computations underlying the execution of processes are represented by means of events and behavioral relations. Events represent occurrences of atomic actions; whereas behavioral relations, which differ in the various types of event structures, explain how events relate each other. Originally, two types of event structures were presented, *elementary event structures* and *prime event structures* (PES), since then many others have been proposed.

Elementary and prime event structures can be derived from causal and occurrence nets, respectively, where the relations of causality and conflict (in the case of occurrence nets) are defined. Either type of event structure is obtained from the corresponding nets by removing the conditions and keeping the events and the relations between them, i.e., causality in the case of causal nets, and causality and conflict in the case of occurrence nets. Intuitively, if a pair of events is in causal relation then the occurrence of one requires the prior execution of the other. On the other hand, when the occurrence of one event prevents the occurrence of another event, we say that they are in conflict relation.

In this thesis, we consider three types of event structures, prime event structures, *asymmetric event structures* (AESs) [Bald 01] and *flow event structures* (FESs) [Boud 89]. On the one hand, AESs provide an asymmetric version of conflict and, on the other hand, FESs provide a form of disjunctive causality.

3.2.1 Prime event structures

Prime event structures (PESs) [Wins 87, Niel 81] represent the computations of concurrent systems by means of events and two relations, causality and conflict. For example, Figure 3.14 represents a PES. The straight directed arrows represent causality. Since causality in PESs is a transitive relation, in pictures we only depict direct causal dependencies. The annotated dotted edges represent conflict. They are undirected since conflict in PES is symmetric. For instance, the presence of a straight directed arrow from a to b indicates that a is a cause of b, written $a \le b$, which means that *"in any computation where b occurs, event a must have occurred before"*. Instead, events d and b, connected by a dotted arrow labelled by #, are in conflict, written $b\#d$, which means that *"in any computation, either d or b does not occur"*.

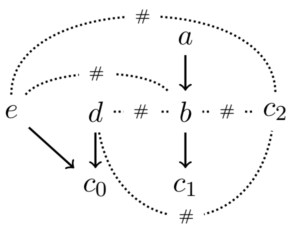


Figure 3.14: Example of PES

We recall the formal definition of *prime event structures* [Niel 81] that complements the informal description provided above.

Definition 3.29 (prime event structure)**.** A (labelled) *prime event structure* (PES) is a tuple $\mathbb{P} = \langle E, \le, \#, \lambda \rangle$, where E is a set of events, $\le$ and $\#$ are binary relations on E called *causality* and *conflict*, respectively, and $\lambda : E \to \Lambda \cup \{\tau\}$ is a labelling function, such that

- $\le$ is a partial order and $\lfloor e \rfloor = \{e' \in E \mid e' \le e\}$ is finite for all $e \in E$;
- $\#$ is irreflexive, symmetric and hereditary with respect to causality, i.e., for all $e, e', e'' \in E$, if $e\#e' \le e''$ then $e\#e''$.

Henceforth, we will write $e < e'$ for $e \leq e'$ and $e \neq e'$. In order to lighten the notation, events will be often named by the corresponding labels, possibly with subscripts (e.g., c_0, c_1 and c_2 are events labelled by c).

The computations in an event structure are usually described in terms of configurations, i.e., sets of events that are closed with respect to causality and conflict free.

Definition 3.30 (configuration of PES)**.** Let $\mathbb{P} = \langle E, \leq, \#, \lambda \rangle$ be a PES. A configuration of PES is a set of events $C \subseteq E$ such that

- for all $e \in C$, $\lfloor e \rfloor \subseteq C$ and
- for all $e, e' \in C$, $\neg(e \# e')$.

The set of all configurations of a PES $\mathbb{P}$ is denoted by $\mathit{Conf}(\mathbb{P})$.

PESs can be seen as instances of families of pomsets. More specifically, given a PES $\mathbb{P} = \langle E, \leq, \#, \lambda \rangle$ and its set of configurations $\mathit{Conf}(\mathbb{P})$, the local order of a configuration $C \in \mathit{Conf}(\mathbb{P})$ is $\leq_C = \leq_{|C}$, i.e., the restriction of the causality relation to C. The extension order turns out to be simply subset inclusion. In fact, given $C_1 \subseteq C_2$ clearly $\leq_{C_1} = \leq \cap (C_1 \times C_1)$ is the restriction to C_1 of $\leq_{C_2} = \leq \cap (C_2 \times C_2)$. Moreover, if $e_1 \in C_1$ and $e_2 \in C_2$, with $e_2 \leq_{C_1} e_1$, then necessarily $e_2 \in C_1$ since configurations are causally closed.

3.2.2 Asymmetric event structures

Asymmetric event structures (AESs) [Bald 01] are another flavor of event structures. In the case of AESs, there are two relations: causality, with the same interpretation as in PES, and asymmetric conflict, which is a non-symmetric version of the conflict in PES. Figure 3.15 depicts an AES. Causal dependencies are still represented using straight directed arrows, e.g., we have that $a \leq b$. Instead, asymmetric conflict is represented by a dotted directed arrow and the corresponding relation is denoted by $\nearrow$. For instance, we have that $b \nearrow c_{01}$ which means that *"the occurrence of event*

c_{01} *prevents b to occur afterwards"*. Hence b and c_{01} can occur in the same computation, but b has to precede c_{01} in such computations. Nevertheless, whenever two events, such as d and b, are related by asymmetric conflict in both directions, namely $d \nearrow b$ and $b \nearrow d$, then none can occur after the other, and thus *"either event d occurs or b occurs, but not both"*.

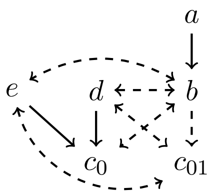


Figure 3.15: Example of AES

We start by briefly reviewing the basics of asymmetric event structures, which, as mentioned before, generalise PESs by allowing a conflict relation that is not required to be symmetric.

Definition 3.31 (asymmetric event structure)**.** A (labelled) *asymmetric event structure* (AES) is a tuple $\mathbb{A} = \langle E, \leq, \nearrow, \lambda \rangle$, where E is a set of events, $\leq$ and $\nearrow$ are binary relations on E called *causality* and *asymmetric conflict*, respectively, and $\lambda : E \to \Lambda \cup \{\tau\}$ is a labelling function, such that

- $\leq$ is a partial order and $\lfloor e \rfloor = \{e' \in E \mid e' \leq e\}$ is finite for all $e \in E$;
- $\nearrow$ satisfies, for all $e, e', e'' \in E$
  1. if $e < e'$ then $e \nearrow e'$,
  2. if $e \nearrow e'$ and $e' < e''$ then $e \nearrow e''$;
  3. $\nearrow_{|\lfloor e \rfloor}$ is acyclic;
  4. if $\nearrow_{|\lfloor e \rfloor \cup \lfloor e' \rfloor}$ is cyclic then $e \nearrow e'$.

Complementing the intuition provided above, we can say that asymmetric conflict has two possible interpretations, that is $e \nearrow e'$ can be understood as (i) the occurrence of e' *prevents* e to occur afterwards or (ii) the occurrence of e *precedes* the occurrence of e' in all computations where both appear. In the first view, $\nearrow$ can be seen as an asymmetric form of conflict, whence the name. Indeed, note that if e and e' are related by asymmetric conflict in both directions, i.e., $e \nearrow e'$ and $e' \nearrow e$, then none can occur after the other, and thus e and e' can never occur in the same computation as it happens for symmetric conflict in PESs. In the second view, $\nearrow$ can be seen as a weak form of causality since $e \nearrow e'$ imposes an order on the occurrences of e and e', but only when they appear in the same computation. Instead, causality $e < e'$ imposes a stricter requirement: in any computation in which e' occurs then e also occurs, and the latter must occur before.

Condition (1) of Definition 3.31 is motivated by the fact that, as observed in (ii), $\nearrow$ imposes weaker requirements than $<$, hence it is natural to ask that $\nearrow$ includes $<$. In the graphical representation of an AES, the asymmetric conflicts $e \nearrow e'$ between events that are also causally dependent $e < e'$ are not represented explicitly. Condition (2) expresses inheritance of asymmetric conflict along causality: if $e \nearrow e'$ and $e' < e''$ then e is necessarily executed before e'' when both appear in the same computation, hence $e \nearrow e''$ (see Fig. 3.17a). Conditions (3) and (4) can be understood by recalling that events forming a cycle of asymmetric conflict cannot appear in the same computation, since each event in the cycle should occur before itself. This leads to a notion of *conflict* over sets of events $\#X$, defined by the following rules

$$\frac{e_0 \nearrow e_1 \nearrow \dots \nearrow e_n \nearrow e_0}{\#\{e_0, \dots, e_n\}} \qquad\qquad \frac{\#(X \cup \{e\}) \;\; e \leq e'}{\#(X \cup \{e'\})}$$

The first rule captures the fact that events in a cycle of asymmetric conflict cannot occur in the same computation. The second rule expresses inheritance of conflict with respect to causality: if events in the set $X \cup \{e\}$ cannot occur in the same computation and $e \leq e'$, then also events in $X \cup \{e'\}$ cannot occur in the same computation. The reason is that the presence of e' requires the prior occurrence of e. Figure 3.16 shows an example where $\#\{e_1, e_2, e_3\}$ by the first rule of conflict over sets and, by the second rule, applied three times, we deduce $\#\{e_1', e_2', e_3'\}$. Note that the second rule is essential: in fact, by Definition 3.31(2) we have that $e_3 \nearrow e_1'$, $e_1 \nearrow e_2'$ and $e_2 \nearrow e_3'$ (as clarified later, inherited asymmetric conflicts are not represented in pictures), but events e_1', e_2', e_3' are not in a cycle of asymmetric conflict, hence the first rule would be insufficient to prove $\#\{e_1', e_2', e_3'\}$.

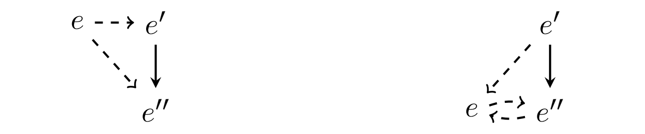
$$\begin{tikzcd}
e_1 \arrow[rr, dashed] \arrow[dd] & & e_2 \arrow[dd] \\
& e_3 \arrow[ul, dashed] \arrow[ur, dashed] \arrow[dd] & \\
e_1' & & e_2' \\
& e_3' &
\end{tikzcd}$$

Figure 3.16: Inheritance of conflict along causality in AESs.

In this view, condition (3) corresponds to irreflexiveness of conflict in PESs, and it ensures that any event is executable i.e., it appears in some computation. Concerning condition (4), notice that whenever the union of the causes of e and e' includes a cycle of asymmetric conflict, according to the rules for conflict above, we have that $\#\{e, e'\}$, i.e., e and e' are in binary symmetric conflict. In this case, condition (4) imposes that $e \nearrow e'$ and also $e' \nearrow e$, since union is symmetric and thus the role of e and e' is interchangeable. This means that symmetric conflict is represented by asymmetric conflict in both directions.

Conditions (2) and (4) impose a form of saturation for the asymmetric conflict relation. Whenever $e \nearrow e'' < e'$ or $\nearrow_{|\lfloor e \rfloor \cup \lfloor e' \rfloor}$ is cyclic, then it holds that e' cannot precede e in a computation. These conditions ask that this is also represented syntactically with an explicit asymmetric conflict. Apart from aesthetic motivations, the validity of these conditions will simplify the formulation of the folding technique described in Chapter 6.

As usual, a set of events X is called *consistent* if its causal closure does not include a subset of events in conflict, i.e., there is no $Y \subseteq \lfloor X \rfloor$ such that $\#Y$.



(a) $e \nearrow_\delta e'$ and $e \nearrow e'$ (b) $e' \nearrow_\delta e$ and $e \#_\delta e''$

Figure 3.17: Inheritance of $\nearrow$

We recall that PESs can be seen as special AESs where asymmetric conflict is a symmetric relation. Namely, the following holds (see [Bald 01]).

Lemma 3.32 (PESs are AESs). *If $\mathbb{P} = \langle E, \leq, \# \rangle$ is a PES then $\mathbb{A} = \langle E, \leq, \nearrow \rangle$, with $\# = \nearrow$ is an AES. If $\mathbb{A} = \langle E, \leq, \nearrow \rangle$ is an AESs with symmetric $\nearrow$, then $\mathbb{P} = \langle E, \leq, \# \rangle$ with $\# = \nearrow$ is a PES.*

In the following, direct relations, namely causality, asymmetric conflict and conflicts that are not inherited, will play a special role.

Definition 3.33 (direct relations). Let $\mathbb{A} = \langle E, \leq, \nearrow, \lambda \rangle$ be an AES and let $e, e' \in E$. We say that e is a *direct cause* of e', denoted $e <_\delta e'$, when $e < e'$ and there is no e'' such that $e < e'' < e'$. An asymmetric conflict $e \nearrow e'$ is called *direct*, written $e \nearrow_\delta e'$ when there is no e'' such that $e \nearrow e'' < e'$. A binary conflict $e \# e'$ is called *direct*, written $e \#_\delta e'$, when $e \nearrow_\delta e'$ and $e' \nearrow_\delta e$.

For instance, in Fig. 3.17a $e \nearrow_\delta e'$ while it is not the case that $e \nearrow_\delta e''$, since $e \nearrow e' < e''$. In Fig. 3.17b we have that $e'' \nearrow_\delta e$ and $e \nearrow_\delta e''$, hence $e \#_\delta e''$.

For the sake of readability and consistency with what we did for PESs, in pictures, often only direct relations will be represented.

Configurations in AESs are defined, as in PESs, as causally closed and conflict free sets of events.

Definition 3.34 (configuration of AES). A *configuration* of an AES $\mathbb{A} = \langle E, \leq, \nearrow, \lambda \rangle$ is a finite set of events $C \subseteq E$ such that i) for any $e \in C$, $\lfloor e \rfloor \subseteq C$ (causal closedness) and ii) $\nearrow_{|C}$ is acyclic (conflict freeness).

The set of all configurations of $\mathbb{A}$ is denoted by $\mathit{Conf}(\mathbb{A})$.

AESs can be seen as instances of families of pomsets by considering each configuration $C \in \mathit{Conf}(\mathbb{A})$ with local order $(\nearrow_{|C})^*$, i.e., the transitive closure of asymmetric conflict restricted to C. Differently from what happens for PESs, the extension order is not simply set-inclusion. It is easy to see that according to the definition showed above, for $C_1, C_2 \in \mathit{Conf}(\mathbb{A})$, we have $C_1 \sqsubseteq C_2$, iff $C_1 \subseteq C_2$ and for all $e \in C_1,\ e' \in C_2 \setminus C_1,\ \neg(e' \nearrow e)$. In words, configuration C_1 cannot be extended with events which should precede some of the events already present in C_1.

A fundamental notion is that of history of an event in a configuration.

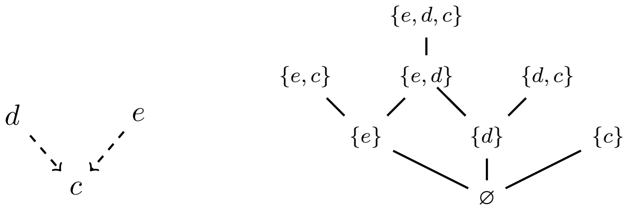
Definition 3.35 (possible histories)**.** Let $\mathbb{A} = \langle E, \leq, \nearrow \rangle$ be an AES and let $e \in E$ be an event in $\mathbb{A}$. Given a configuration $C \in \mathit{Conf}(\mathbb{A})$ such that $e \in C$, the *history* of $e \in C$ is defined as $C[\![e]\!] = \{e' \in C \mid e'(\nearrow_{|C})^* e\}$. The *set of possible histories* of e, denoted by $\mathit{hist}(e)$, is then defined as

$$\mathit{hist}(e) = \{C[\![e]\!] \mid C \in \mathit{Conf}(\mathbb{A}) \land e \in C\}$$

The history $C[\![e]\!]$ consists of the events which necessarily must occur before e in the configuration C or, in other words, it is the minimal sub-configuration of C, with respect to the extension order, which contains event e. For PESs, each event e has a uniquely determined history, which is the set $\lfloor e \rfloor$, independently of the configuration it occurs in. Instead, in the case of AESs, an event e may have several histories. For example, Figure 3.18 shows an AES $\mathbb{A}$ and its configurations ordered by extension. In this example, it is easy to check that the event c has four different histories, $\mathit{hist}(c) = \{\{c\}, \{d, c\}, \{e, c\}, \{d, e, c\}\}$.

3.2.3 Flow event structures

Flow event structures [Boud 89] is another type of event structures. This type of event structures has two relations, flow relation, which is represented with a double-headed straight arrow and denoted by $\prec$, and conflict, with



(a) $\mathbb{A}$ **(b)** $Conf(\mathbb{A})$

Figure 3.18: AES $\mathbb{A}$ and its set of configurations ordered by extension

the same interpretation and representation as in PES. The flow relation is not transitive and, intuitively, expresses the set of potential direct causes for a given event. Then, in order for an event to occur, a maximal, conflict free set of potential direct causes has to occur beforehand. Figure 3.19 shows an example of a FES, where $e \prec c_0$, $d \prec c_0$ and $b \prec c_0$. Hence, $\{e, d, b\}$ is the set of potential direct causes for c_0, whose execution must be preceded by either $\{e, d\}$ or $\{b\}$.

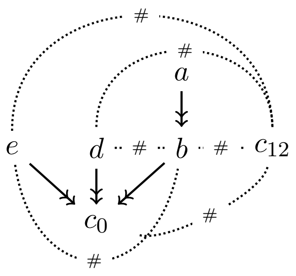


Figure 3.19: Example of FES

We start by recalling the formal definition of (labelled) flow event structures [Boud 89].

Definition 3.36 (flow event structure)**.** A (labelled) *flow event structure* (FES) is a tuple $\mathbb{F} = \langle E, \#, \prec, \lambda \rangle$ where E is a set of events, $\lambda : E \to \Lambda \cup \{\tau\}$ is a labelling function, and

- $\prec \subseteq E \times E$, the *flow* relation, is irreflexive.

- $\# \subseteq E \times E$, the *conflict* relation, is a symmetric relation,

The $\prec$-predecessors of an event $e \in E$, are defined as ${}^\bullet e = \{e' \mid e' \prec e\}$. Similarly, for a set of events X we write ${}^\bullet X = \bigcup_{x \in X} {}^\bullet x$.

Next, we present the formal definition of configuration of FES.

Definition 3.37 (configuration of FES)**.** Let $\mathbb{F} = \langle E, \#, \prec, \lambda \rangle$ be a FES. A *configuration* of $\mathbb{F}$ is a finite subset $C \subseteq E$ such that

1. $\neg(e \# e')$ for all $e, e' \in C$;
2. $\prec^*_{|C}$ is a partial order, i.e., $\leq_C = \prec^*_{|C}$;
3. for all $e \in C$ and $e' \notin C$ s.t. $e' \prec e$, there exists an $e'' \in C$ such that $e' \# e'' \prec e$.

We denote by $\mathit{Conf}(\mathbb{F})$ the set of configurations of $\mathbb{F}$.

A configuration is a conflict free subset of events, where $\prec^*$ is acyclic, conditions 1 and 2 in Definition 3.37. The third condition in Definition 3.37 requires that, given an event $e \in C$, for any $\prec$-predecessor $e' \prec e$, either $e' \in C$ or it is excluded by the presence of $e'' \in C$, where e'' is in conflict with e' and $e'' \prec e$. This means that for any $e \in C$, the configuration C must include a maximal consistent subset of $\prec$-predecessors of e.

An alternative formulation of configurations of flow event structures is done using proving sequences.

Definition 3.38 (proving sequence)**.** A *proving sequence* in a FES $\mathbb{F} = (E, \#, \prec)$ is a (finite or infinite) sequence $\sigma = e_1 \dots e_n \dots$ of distinct non-conflicting events, s.t. $\forall i \forall e \in E : e \prec e_i \Rightarrow (\exists\, j < i : (e = e_j \vee e \# e_j) \wedge e_j \prec e_i)$.

A subset of events $C \subseteq E$ is a configuration of a FES $\mathbb{F} = (E, \#, \prec)$, s.t. $C = \{e_1, \dots, e_n\}$, if and only C is conflict free and for every event $e_k \in C$, $k \leq n$ it holds that $e_1 \dots e_k$ is a proving sequence in $\mathbb{F}$, cf. [Boud 90].

FESs can be seen as another instance of families of pomsets by considering each configuration $C \in \mathit{Conf}(\mathbb{F})$, of a FES $\mathbb{F}$, ordered by $(\prec_{|C})^*$. As for PESs, the extension order is simply subset-inclusion, namely according to the definition above, for $C_1, C_2 \in Conf(\mathbb{F})$, we have $C_1 \sqsubseteq C_2$, iff $C_1 \subseteq C_2$. In particular, observe that if $e_1 \in C_1$, $e_2 \in C_2$ we have that $e_2 \leq_{C_2} e_1$, then $e_2 \in C_1$, this is proved by the following proposition.

Proposition 3.39. *Let $\mathbb{F} = (E, \#, \prec)$ be a FES and let $C_1, C_2 \in Conf(\mathbb{F})$ be a pair of configurations, such that $C_1 \subseteq C_2$. Then, for any $e_1 \in C_1$, $e_2 \in C_2$ if $e_2 \leq_{C_2} e_1$, then $e_2 \in C_1$.*

Proof. Assume by contradiction that $e_2 \notin C_1$. Since $\leq_{C_2} = \prec^*_{|C_2}$, the proof can proceed on induction on the length of the $\prec$-chain connecting e_2 to e_1. If the length is 0, namely $e_2 \prec e_1$, since $e_2 \notin C_1$, by definition of configuration, there must be $e_1' \in C_1$ such that $e_1' \prec e_1$ and $e_1' \# e_2$. Since $e_1' \in C_1 \subseteq C_2$ this means that C_1 include the conflictual events e_2, e_1', contradicting the assumption that it is a configuration. This concludes the base case. The inductive step is straightforward. □

In FESs, the flow relation is not transitive and the conflict relation is not inherited along causal chains as in PESs. Therefore, two events that are not in conflict syntactically, might not appear together in any configuration. For similar reasons, an event could be not executable at all. More precisely, let us define the semantic conflict relation $\#_s$ as $e \#_s e'$ when for all configuration $C \in Conf(\mathbb{F})$, it does not hold that $\{e, e'\} \subseteq C$. Then clearly $\# \subseteq \#_s$, but in general the inclusion is strict. Moreover, it could be that $e \#_s e$ for an event e (hence e is never executable).

$$\begin{tikzcd}
 & a \arrow[dl] \arrow[ddr] \arrow[rr, dash, dotted, "\#" description] & & b \arrow[dr] \arrow[ddl] & \\
c \arrow[drr] & & & & d \arrow[dll] \\
 & & e & &
\end{tikzcd}$$

Figure 3.20: A FES which is neither faithful nor full.

For example, the FES in Fig. 3.20 is not faithful, i.e., despite the fact that there is no conflict $b \# c$, it holds $b \#_s c$, namely b and c cannot appear in the same configuration. Moreover, since a is the only $\lessdot$-predecessor of c, for any configuration C, if $c \in C$ then also $a \in C$. Therefore, since $a\#b$, necessarily $b \notin C$. Similarly, $a\#_s d$ and $c\#_s d$. Additionally, observe that any configuration containing e, according to Definition 3.37, should include both c and d (since they are not in conflict with any other $\lessdot$-predecessor of e). Therefore, there is no such configuration, i.e., $e\#_s e$.

In line with the authors of [Boud 89], hereafter we restrict to the subclass of FES, where:

1. semantic conflict $\#_s$ coincides with conflict $\#$ (*faithfulness*),
2. conflict is irreflexive (*fullness*), hence all events are executable,
3. $\lessdot$ and $\#$ are disjoint.

Condition 3 is not in [Boud 89]. We assume it here since it is in line with conditions 1 and 2 and it allows us to simplify the presentation.

Note that that FESs generalise PESs. Specifically, every PES can be seen as a special FES where the flow relation is transitive and the $\lessdot$-predecessors of any event are conflict free.

3.3 True concurrency semantic equivalences

Several equivalence notions for concurrent systems have been presented in the literature (see [Glab 89, Glab 90, Glab 01] for a compilation of some of them). This thesis adopts equivalence notions on the true concurrency semantics. I.e., we consider equivalence notions that distinguish arbitrary interleaving from simultaneous (concurrent) execution of tasks. This section introduces configuration equivalence, completed visible-pomset equivalence and history preserving bisimulation equivalence. The following definitions are formulated in terms of families of pomsets, thus the notions are applicable to the different formalisms presented in this chapter.

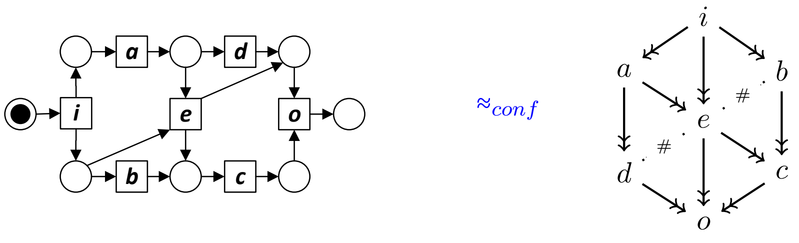
3.3.1 Configuration equivalence

The first equivalence notion is configuration equivalence [Glab 95]. A pair of families of pomsets are configuration equivalent if 1) there is a bijection between events, and 2) they represent, essentially, the same set of configurations over those events.

Observe that the bijection between the events is given and so the labels of the events, as well as the order between them are not (explicitly) taken into account.

Definition 3.40 (configuration equivalence $\approx_{conf}$)**.** Let $\mathcal{P} = \langle E, \mathit{Conf}(\mathcal{P}), \lambda\rangle$ and $\mathcal{P}' = \langle E', \mathit{Conf}(\mathcal{P}'), \lambda'\rangle$ be two families of pomsets, and let $\Gamma : E \to E'$ be a bijective function between the sets of events. Let $\mathcal{P} \mathrel{::\!\sim_{conf}} \mathcal{P}'$ denote that for any configuration C in $\mathit{Conf}(\mathcal{P})$ there is a corresponding configuration C' in $\mathcal{P}'$ consisting of the images of C. I.e., $\forall C \in \mathit{Conf}(\mathcal{P}) \exists C' \in \mathit{Conf}(\mathcal{P}') : C' = \{\Gamma(e) \mid e \in C\}$.

The families of pomsets $\mathcal{P}, \mathcal{P}'$ are *configuration equivalent*, denoted $\mathcal{P} \approx_{conf} \mathcal{P}'$, if $\mathcal{P} \mathrel{::\!\sim_{conf}} \mathcal{P}'$ and vice-versa.



(a) $\mathcal{N}_2$ **(b)** FES $\mathbb{F}$

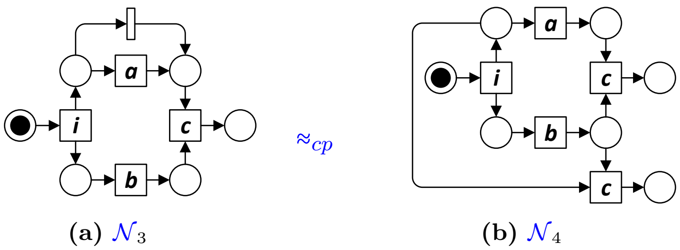
Figure 3.21: Example of configuration-equivalent families of pomsets.

Figure 3.21 shows an example of two families of pomsets, a flow net and a FES, which are configuration equivalent. The bijection between the events is given by the labels.

3.3.2 Completed visible-pomset equivalence

Completed visible-pomset equivalence [Glab 89, Golt 94] deems as equivalent a pair of families of pomsets iff they have isomorphic maximal visible-pomsets (namely computations that cannot be further extended, because they are either terminated or infinite). As mentioned previously, a pair of pomsets are isomorphic if there is a bijection between the events and it respects the order and labeling. This means that the concurrent structure of such computations (causal dependencies and parallelism between visible events) is exactly the same.

Definition 3.41 (completed pomset equivalence)**.** Let $\mathcal{P} = \langle E, \mathit{Conf}(\mathcal{P}), \lambda\rangle$ and $\mathcal{P}' = \langle E', \mathit{Conf}(\mathcal{P}'), \lambda'\rangle$ be two families of pomsets. We say that two net systems $\mathcal{P}$ and $\mathcal{P}'$ are *completed (visible) pomset equivalent*, denoted $\mathcal{P} \approx_{cp} \mathcal{P}'$, whenever $\mathit{MaxConf}(\mathcal{P})^{\Lambda} = \mathit{MaxConf}(\mathcal{P}')^{\Lambda}$.



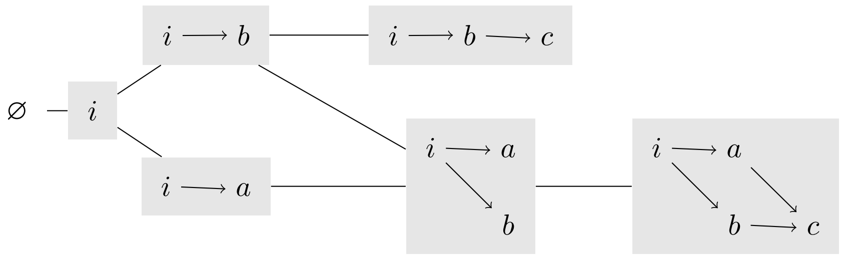


Figure 3.22: Visible pomset equivalent flow net systems (a), (b) and their visible pomsets ordered by inclusion (c).

Figure 3.22 depicts a pair of completed visible pomset-equivalent net systems along with their visible pomsets ordered by inclusion.

3.3.3 History preserving bisimilarity

History-preserving bisimilarity [Rabi 88, Glab 89, Best 91] is based on branching time partial order semantics. Thus, it does not only look at the behavior generated by the systems (e.g., pomsets or configurations), but it also considers the moments of choice between alternative branches of behavior.

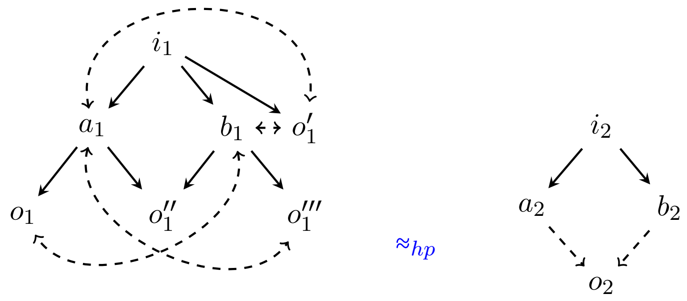
Definition 3.42 (history preserving bisimilarity $\approx_{hp}$)**.** Let $\mathcal{P} = \langle E, \mathit{Conf}(\mathcal{P}), \lambda \rangle$ and $\mathcal{P}' = \langle E', \mathit{Conf}(\mathcal{P}'), \lambda' \rangle$ be two families of pomsets. A *history preserving (hp-)bisimulation* is a set R of triples (C_1, f, C_2), where $C_1 \in \mathit{Conf}(\mathcal{P})$, $C_2 \in \mathit{Conf}(\mathcal{P}')$ and $f : C_1 \to C_2$ is an isomorphism of configurations, such that $(\varnothing, \varnothing, \varnothing) \in R$ and for all $(C_1, f, C_2) \in R$

a) if $C_1 \xrightarrow{e_1} C_1 \cup \{e_1\}$, for an event $e_1 \in E$, there exists $e_2 \in E'$ such that $C_2 \xrightarrow{e_2} C_2 \cup \{e_2\}$ and $(C_1 \cup \{e_1\}, f[e_1 \mapsto e_2], C_2 \cup \{e_2\}) \in R$;

b) if $C_2 \xrightarrow{e_2} C_2 \cup \{e_2\}$, for an event $e_2 \in E'$, there exists $e_1 \in E$ such that $C_1 \xrightarrow{e_1} C_1 \cup \{e_1\}$ and $(C_1 \cup \{e_1\}, f[e_1 \mapsto e_2], C_2 \cup \{e_2\}) \in R$;

When a history preserving bisimulation exists, $\mathcal{P}$, $\mathcal{P}'$ are called *history preserving bisimilar*, written $\mathcal{P} \approx_{hp} \mathcal{P}'$.

Observe that in the definition above, an event must be simulated by an event with the same label, as it follows from the fact that for triple $(C_1 \cup \{e_1\}, f[e_1 \mapsto e_2], C_2 \cup \{e_2\}) \in R$ the second component $f[e_1 \mapsto e_2]$ is an isomorphism of configurations (thus it preserves labels). An example of a pair of history-preserving bisimilar AESs is depicted in Figure 3.23.



(a) $\mathbb{A}_1$ **(b)** $\mathbb{A}_2$

Figure 3.23: History preserving bisimilar AESs

Chapter 4

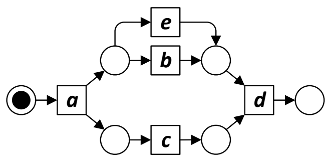
Behavioral profiles for process model comparison

Configuration equivalence
WF-flow nets
Unlabeled
Labeled
Behavioral profiles
Comparison
Diagnostics

This chapter analyses the expressive power of *Behavioral Profiles* (BP), which have been proposed as a behavioral abstraction of business process models. Section 4.1 introduces behavioral profiles. Section 4.2 shows that FESs can be behavioral profiles for a class of nets. An execution semantics for an existing type of behavioral profiles, $BP|_w$, is proposed in Section 4.3, and the interpretation of its relations is discussed in Section 4.4. Next, Section 4.5 shows that existing behavioral profiles cannot ensure a well-known notion of equivalence for nets with silent transitions. Final remarks and discussions are presented in Section 4.6.

4.1 Behavioral profiles (BP)

Behavioral profiles [Weid 11b] have been proposed as an abstract representation of process models' behavior. The behavioral profile of a process model can be seen as a complete graph over the set of tasks of the model, where edges are annotated by types of behavioral relations. Alternatively, a behavioral profile is a matrix where rows and columns represent tasks and each cell is labeled by a behavioral relation between a pair of tasks. In this section we assume that the tasks (transitions in the nets) have distinct labels and say that the size of the behavioral profile is $O(|\Lambda|^2)$.



(a) $\mathcal{N}_5$

	a	b	c	d	e
a	$+$	↦	↦	↦	↦
b	↤	$+$	⦀	↦	$+$
c	↤	⦀	$+$	↦	⦀
d	↤	↤	↤	$+$	↤
e	↤	$+$	⦀	↦	$+$

(b) $BP|_w$

Figure 4.1: Net system and its behavioral profile $BP|_w$

Figure 4.1 shows a net system and alongside its behavioral profile computed with the relations from [Weid 11b], referred to as *classic behavioral profile* and denoted as $BP|_w$. In the matrix representation, the strict order relation ($\mapsto$) denotes causal precedence between a pair of tasks in all the computations of the model. Exclusive order relation ($+$) denotes that a pair of tasks never occurs in the same computation. Finally, interleaving (⦀) represents the absence of order in the execution of a pair of tasks. Since the introduction of classic behavioral profiles many other families of relations have been proposed for creating $O(|\Lambda|^2)$ behavioral representations of process models; causal behavioral profiles [Weid 11c] and $4C$ spectrum [Poly 14] are cases in point.

Classic behavioral profiles count with several properties appealing for the behavioral comparison of process models. For instance, they can be efficiently computed [Weid 11a], they have been used to define a behavior similarity metric between process models [Kunz 11], and their comparison can generate feedback in the form of mismatching pairs of behavioral relations. By the same token, if the behavioral relations used in this formalism represent constructs in the modeling language – such as $\mapsto$, + and $\|$– then the generated feedback can be easily interpretable.

Since the introduction of the behavioral profiles [Weid 11b], the authors acknowledged that this representation does not correspond to any of the well-known notions of behavioral equivalence insofar as two behaviorally different models (e.g., by trace equivalence) may have the same matrix representation. Other families of relations that follow the same idea of behavioral profiles suffer from the same issue, e.g., causal behavioral profile and $4C$ spectrum. Furthermore, the lack of execution semantics of these representations hinders the analysis of their expressive power. Specifically, it is not clear what is the behavior captured or lost in a behavioral profile. Thus, it is still an open question how accurate behavioral profiles are, and if it is possible to characterize a substantial family of Petri nets for which a notion of equivalence can be ensured.

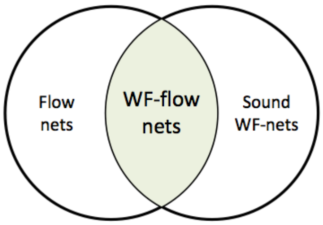


Figure 4.2: WF-flow nets

This chapter defines an execution semantics for behavioral profiles, concretely for $BP|_w$. The execution semantics is defined as a mapping from $BP|_w$ to FES. Then, it is shown that $BP|_w$ can ensure a well known-notion of equivalence in true concurrency semantics, i.e., configuration equivalence, for a family of nets without silent transitions. The discussions throughout this chapter consider Petri nets in the intersection of two families, sound WF-nets and flow nets, shorthanded as WF-flow nets and denoted by η, Figure 4.2.

WF-flow nets impose structural and behavioral restrictions, on the one hand, a WF-flow net is acyclic and has a dedicated source place ${}^\bullet i = \emptyset$ and a dedicated sink place $o^\bullet = \emptyset$. On the other hand, a WF-flow net system 1) is sound –and so every execution ends with one token in the sink place and no tokens elsewhere–; 2) has an initial marking $M = \{i\}$; and 3) for every firing sequence σ, a place can be in the preset of at most one transition of σ, and if two transitions in σ have a place between them then there is a strong postcondition between them. These restrictions are in line with the definitions of (sound) WF-net system and flow net system (Def. 3.21 and 3.18).

Generalized behavioral profiles

Behavioral profiles can be seen as a framework that is concretely defined according to a set of behavioral relations. Roughly speaking, a behavioral profile $BP|_{\mathscr{R}}$ of a process model is a complete graph over the set of tasks, which uses a set of relations $\mathscr{R}$ as edge labels. This general notion of behavioral profiles results useful for uniformly analyze the different formalisms considered in this section. We denote the behavioral profile over $\mathscr{R}$ of a net system $\mathcal{N}$ as $BP|_{\mathscr{R}}(\mathcal{N})$

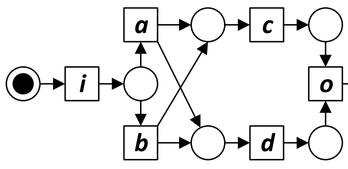
We say that a behavioral profile is *behavior preserving* for a class of nets $\mathfrak{N}$, if any pair of behavior-equivalent (under certain notion of equivalence) net systems with nets in $\mathfrak{N}$, have isomorphic behavioral profiles (denoted by $\equiv_{iso}$) and vice-versa. This intuition is captured by the following definition.

Definition 4.1 (behavior-preserving $BP|_{\mathscr{R}}$)**.** Let $\mathfrak{N}$ be a class of nets and $\approx$ be an equivalence relation on $\mathfrak{N}$. A behavioral profile $BP|_{\mathscr{R}}$ is behavior-preserving on $\mathfrak{N}$, if for any $N, N' \in \mathfrak{N}$ with net systems $\mathcal{N} = (N, M_0)$, $\mathcal{N}' = (N', M_0')$ and behavioral profiles $BP|_{\mathscr{R}}(\mathcal{N})$ and $BP|_{\mathscr{R}}(\mathcal{N}')$, respectively, the following holds:

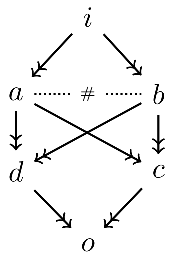
$$\mathcal{N} \approx \mathcal{N}' \Leftrightarrow BP|_{\mathscr{R}}(\mathcal{N}) \equiv_{iso} BP|_{\mathscr{R}}(\mathcal{N}').$$

4.2 FES as BP

Boudol shows that FES corresponds to the family of flow nets [Boud 90], i.e., it is always possible to compute a FES for a given flow net system, where the configurations of the FES are firing sequences in the system. Interestingly, it is possible to establish a bijection between the transitions and the events in the corresponding FES representation for a sound WF-flow net.[1] Figure 4.3 shows a flow net system and the corresponding FES aside. Note that for every transition in the net there is an event in the event structure with the same label, and the relations of flow and conflict are those of Definition 3.23. Intuitively, a pair of events are in flow relation if there is a strong postcondition between them in the net; whereas they are in conflict relation if they never occur in the same firing sequence.



(a) $\mathcal{N}_6$



(b) FES of $\mathcal{N}_6$

Figure 4.3: Flow net system and its corresponding FES

[1]Additional self-conflicting events can be required in a FES when, in the context of WF-nets, a net system does not meet the property of liveness.

The next definition suggests how to construct a FES from sound WF-flow nets and so self-conflicting events are omitted. Additionally, given that there is a bijection between the transitions in the net and the events in the FES, we use T to represent both, events and transitions, indistinctively.

Definition 4.2 (FES of a flow net)**.** Let $\mathcal{N} = (N, M)$, $N = (P, T, F) \in \eta$, be a WF-flow net system. The *FES of* $\mathcal{N}$ is the tuple $\mathbb{F} = \langle T, \#^N, \prec^N \rangle$, where $\#^N$ and $\prec^N$ are those defined in Definition 3.23.

The following proposition restates the results proved in [Boud 90] for flow nets.

Proposition 4.3 (Proposition 3.4 in [Boud 90])**.** *Let $\mathcal{N} = (N, M_0)$ be a WF-flow net system, with a net $N = (P, T, F) \in \eta$, and let $\mathbb{F}$ be its corresponding FES, then $Conf(\mathcal{N}) = Conf(\mathbb{F})$. More precisely, a sequence $t_1 \ldots t_n$ is firable in $\mathcal{N}$ if and only if it is a proving sequence in $\mathbb{F}$.*

A result from Proposition 4.3, captured in the following Corollary, is that a pair of configuration equivalent WF-flow nets have, similarly, configuration equivalent FESs.

Corollary 4.4. *Let N, N' be nets in η. Moreover, let $\mathbb{F}$ and $\mathbb{F}'$ be the FESs of $\mathcal{N}$ and $\mathcal{N}'$, respectively. Then, the following holds:*

$$\mathcal{N} \approx_{conf} \mathcal{N}' \Leftrightarrow \mathbb{F} \approx_{conf} \mathbb{F}'$$

FESs define a type of behavioral profiles, denoted by $BP|_{fes}$, where the events are the tasks of the behavioral profile, and the flow and conflict are the relations thereof. Additionally, the notions of configuration and extension of FES gives an execution semantics to this type of behavioral profiles, such that any conclusion (w.r.t. behavior) derived from $BP|_{fes}$ holds in the corresponding WF-flow nets system, and vice-versa. The $BP|_{fes}$ representing the FES in Figure 4.3b is shown in Figure 4.4 and, by completeness, it contains the inverse flow relations ($\prec^{-1}$).

	i	a	b	c	d	o
i		$\prec$	$\prec$			
a	$\prec^{-1}$		$\#$	$\prec$	$\prec$	
b	$\prec^{-1}$	$\#$		$\prec$	$\prec$	
c		$\prec^{-1}$	$\prec^{-1}$			$\prec$
d		$\prec^{-1}$	$\prec^{-1}$			$\prec$
o				$\prec^{-1}$	$\prec^{-1}$	

Figure 4.4: $BP|_{fes}(\mathcal{N}_6)$

In FESs, the flow relation is defined with respect to the strong postconditions between pairs of transitions. The following corollary shows that all the places, with exception of the source and sink places, in a WF-flow net are indeed strong postconditions. This technical result is used later to show that $BP|_{fes}$ are not behavior preserving.

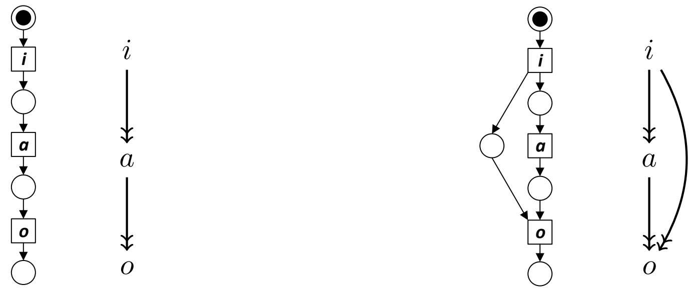
Corollary 4.5. *Let $\mathcal{N} = (N, M_0)$ be a net system, with a WF-flow net $N = (P, T, F) \in \eta$, and $t, t' \in T$ be a pair of transitions, such that $t^\bullet \cap {}^\bullet t' \neq \varnothing$. Then, any place $p \in t^\bullet \cap {}^\bullet t'$ is a strong postcondition, i.e., $p \in \psi(t, t')$.*

Proof. Let $\sigma = t_1\ t_2 \dots t_n \in \Delta(\mathcal{N})$, for a $n \in \mathbb{N}_0$, be an execution of $\mathcal{N}$, such that $t_i = t$ and $t_l = t'$ and $1 \leq i, l \leq n$. In this case, it is shown that the property holds for an execution σ, but then it also holds for any firing sequence, which elements are part of σ. Observe that p is neither the source nor the sink place since ${}^\bullet p \neq \varnothing \neq p^\bullet$ and that, by the properties of soundness of WF-nets, $M_0[\sigma\rangle M_n$ and $M_n = \{o\}$, where o is the sink place.

Suppose that p is not a strong postcondition of t, i.e., $p \notin \psi(t, t')$. The only chance p is not a strong postcondition is that $M_0(p) + |F'| > 1$, where $F' = \{(t_j, p) \in F\}$ for $1 \leq j \leq n$. $M_0(p)$ is clearly 0 since p is not the source place, but then there is a $t_k \neq t$ in σ, for a $1 \leq k \leq n$, such that $(t_k, p) \in F$. Then p was marked, at least, twice while firing σ, by t_k and t, but since the net is 1-safe, then a token was consumed before the other was set. By Definition 3.21, p is in the preset of at most one transition of σ and so t_l consumes a token from p, but then it remains with at least 1 token at M_n, i.e., $p \in M_n$. Nevertheless, it violates the property of sound WF-nets, since

$M_n = \{o\}$ and $p \neq o$. Thus $p \in t^\bullet \cap {}^\bullet t'$ is necessarily a strong postcondition of t, as required. □

As a result of the corollary above, the behavioral profiles $BP|_{fes}$ are not behavior preserving. I.e., a net system with implicit places would define "unnecessary" flow relations between the events in the corresponding FES. For instance, consider the WF-flow net systems and their FESs in Figure 4.5. The WF-flow net system in Figure 4.5b has an additional place that leads to the flow relation between the events i and o in the corresponding FES. Nevertheless, even though the resulting FESs are not isomorphic, they represent the same set of configurations, namely $\{\varnothing, \{i\}, \{i,a\}, \{i,a,o\}\}$.



(a) WF-flow net system and its corresponding FES

(b) WF-flow net system with an implicit place and the corresponding FES

Figure 4.5: Equivalent WF-flow nets without (4.5a) and with (4.5b) an implicit place and their corresponding FESs aside.

4.3 An execution semantics for $BP|_w$

As mentioned in Section 4.1, classic behavioral profiles [Weid 11b] lacks of an execution semantics, which hinders on a behavioral evaluation of this formalism. For example, it is not possible trace back the computations of the system that lead to a given $BP|_w$, or to identify the computations captured in this formalism. This section presents a transformation from $BP|_w$ to $BP|_{fes}$ and shows that it is behavior preserving for WF-flow nets without silent transitions. First, let us define the behavior relations used in $BP|_w$, along with its computation.

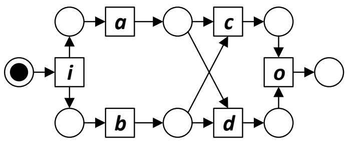
Definition 4.6 ($BP|_w$)**.** Let $\mathcal{N} = (N, M)$ be a net system, with $N = (P, T, F)$. A pair of transitions $t, t' \in T$ is in one of the following relations:

- *Strict order relation*, denoted by $t \mapsto t'$, if for every firing sequence $\sigma \in \Delta(\mathcal{N})$, with $\sigma = t_1\ t_2 \dots t_n$ such that $t_i = t$ and $t_j = t'$, it holds $1 \le i < j \le n$.
- *Exclusive order relation*, denoted by $t + t'$, if for every firing sequence $\sigma \in \Delta(\mathcal{N}) : \sigma = t_1\ t_2 \dots t_n$ there are no t_i, t_j, where $1 \le i, j \le n$, s.t. $i \ne j$, $t_i = t$ and $t_j = t'$.
- *Interleaving relation*, denoted by $t \mathbin{|||} t'$, if $\neg(t \mapsto t')$, $\neg(t' \mapsto t)$ and $\neg(t + t')$.

For technical reasons, we also define the *direct strict order*. Transitions t and t' are in direct strict order, denoted by $t \twoheadrightarrow t'$, iff

$$t_i \mapsto t_j \ \wedge\ t_i^{\bullet} \cap {}^{\bullet}t_j \ne \varnothing$$

The set $BP|_w(\mathcal{N}) = \{\mapsto, +, |||\}$ is the *classic behavioral profile* of $\mathcal{N}$.



(a) $\mathcal{N}_7$

	i	a	b	c	d	o
i	$+$	$\twoheadrightarrow$	$\twoheadrightarrow$	$\mapsto$	$\mapsto$	$\mapsto$
a	$\twoheadleftarrow$	$+$	$\|\|\|$	$\twoheadrightarrow$	$\twoheadrightarrow$	$\mapsto$
b	$\twoheadleftarrow$	$\|\|\|$	$+$	$\twoheadrightarrow$	$\twoheadrightarrow$	$\mapsto$
c	$\mapsfrom$	$\twoheadleftarrow$	$\twoheadleftarrow$	$+$	$+$	$\twoheadrightarrow$
d	$\mapsfrom$	$\twoheadleftarrow$	$\twoheadleftarrow$	$+$	$+$	$\twoheadrightarrow$
o	$\mapsfrom$	$\mapsfrom$	$\mapsfrom$	$\twoheadleftarrow$	$\twoheadleftarrow$	$+$

(b) $BP|_w$

	i	a	b	c	d	o
i		$\prec$	$\prec$			
a	$\prec^{-1}$			$\prec$	$\prec$	
b	$\prec^{-1}$			$\prec$	$\prec$	
c		$\prec^{-1}$	$\prec^{-1}$		$\#$	$\prec$
d		$\prec^{-1}$	$\prec^{-1}$	$\#$		$\prec$
o				$\prec^{-1}$	$\prec^{-1}$	

(c) $BP|_{fes}$

Figure 4.6: Net system $\mathcal{N}_7$ and its behavioral profiles $BP|_w$ and $BP|_{fes}$

For the sake of clarity, in the matrix representation of $BP|_w$ the direct strict order is explicitly represented, but it should be clear that $\twoheadrightarrow$ implies

$\mapsto$. Figure 4.6 shows a WF-flow net system, its $BP|_w$ and its $BP|_{fes}$. Note that with the definition of $\twoheadrightarrow$ in $BP|_w$, both behavioral profiles are very alike. In particular, direct strict order $\twoheadrightarrow$ and exclusive order $+$ (without the self-exclusive relations) in $BP|_w$ correspond to flow $\prec$ and conflict $\#$ in $BP|_{fes}$. Such correspondence defines the proposed transformation from $BP|_w$ to $BP|_{fes}$ and provides the former with the execution semantics of the latter. This transformation is a good basis to analyze the behavior represented (or lost) in a $BP|_w$.

The transformation from $BP|_w$ to $BP|_{fes}$ is formally defined as follows.

Definition 4.7 ($BP|_w^{fes}$)**.** Let $BP|_w(\mathcal{N}) = \{\mapsto, +, \|\|\}$ be the classic behavioral profile of a net system $\mathcal{N} = (N, M)$, with $N = (P, T, F) \in \eta$. Let $+' = +\backslash\{(t,t) \mid t \in T\}$ be the exclusive order relation without the self-exclusive relations. The $BP|_{fes}$ of $BP|_w(\mathcal{N})$ is defined as $BP|_w^{fes}(\mathcal{N}) = \{\twoheadrightarrow, +'\}$.

The reminder of this section considers unlabeled WF-flow nets denoted by $\eta_{\overline{\lambda}}$. The case for labeled nets is presented in Section 4.5. The following proposition shows that the $BP|_w^{fes}(\mathcal{N})$ derived from $BP|_w(\mathcal{N})$ is isomorphic to $BP|_{fes}(\mathcal{N})$, i.e., $BP|_w^{fes}(\mathcal{N})$ represents the same configurations as $BP|_{fes}(\mathcal{N})$.

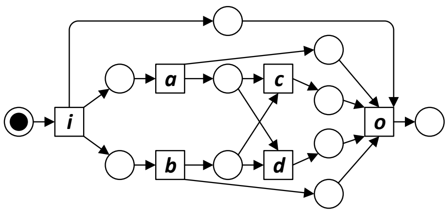
Proposition 4.8. *Let $\mathcal{N} = (N, M)$ be a net system, where $N = (P, T, F) \in \eta_{\overline{\lambda}}$ is an unlabeled WF-flow net. Then $BP|_w^{fes}(\mathcal{N}) = \{\twoheadrightarrow, +'\}$ is isomorphic to $BP|_{fes}(\mathcal{N}) = \{\prec, \#\}$. Specifically, for any two transitions $x, y \in T$ 1) $x +' y \Leftrightarrow x\#y$, and 2) $x \twoheadrightarrow y \Leftrightarrow x \prec y$.*

Proof. Given that the nets are unlabeled, then for any transition in T there is a task in $BP|_{fes}$ and in $BP|_w$ (and so in $BP|_w^{fes}$). Therefore, let us consider the same set of elements T throughout the different structures.

1. $x +' y \Leftrightarrow x\#y$. It is easy to check that the definition of exclusive ordering relation ($+'$) in $BP|_w$ and conflict ($\#$) in $BP|_{fes}$ is the same. Then the conflict in the FES coincides with the exclusive order relation in $BP|_w$.

2. $x \twoheadrightarrow y \;\Leftrightarrow\; x \prec y$. ($\Rightarrow$) Consider a pair of transitions $x, y \in T : x \twoheadrightarrow y$. By Definition 4.6, $\exists \sigma = t_1\, t_2 \ldots, t_n : x = t_i, y = t_j \wedge i < j$. Additionally, since the causal relation is direct, then there is at least a place $p \in x^\bullet \cap {}^\bullet y$. Thus, by Definition 3.23, $x \prec y$ in $BP|_{fes}(\mathcal{N})$.
($\Leftarrow$) Suppose $x \prec y$ in $BP|_{fes}(\mathcal{N})$, but $\neg(x \twoheadrightarrow y)$ in $BP|_w^{fes}(\mathcal{N})$. First, by Definition 3.23, since $x \prec y$ then $\exists p \in \psi(x, y)$ and $\neg(x \# y)$. Furthermore, since p is a strong postcondition of x, there is a firing sequence $\sigma = t_1\, t_2 \ldots, t_n$ such that $x = t_i, y = t_j$, where $1 \leq i < j \leq n$. The only case where $\neg(x \mapsto y)$ can hold is if there is another execution $\sigma' \in \Delta(\mathcal{N})$, s.t., $\sigma' = t'_1, t'_2, \ldots, t'_n$ and $t'_k = x, t'_l = y : 1 \leq l < k \leq n$, and thus $x \mathbin{|||} y$ in $BP|_w(\mathcal{N})$. Observe that since $p \in \psi(x, y)$ then $p \in {}^\bullet t'_l$. Although, the last would imply that $M(p) + |F'| > 1$, where $F' = \{(t_j, p) \in F\}$ for any $1 \leq j \leq n$, in σ'. Specifically, p has a token prior the firing of t'_l and after the firing of t'_k. The last contradicts the fact that p is a strong postcondition (Def. 3.20(2)). Thus, if $x \prec y$ then $x \twoheadrightarrow y$.

$\square$



(a) $\mathcal{N}_8$

	i	a	b	c	d	o
i	$+$	$\twoheadrightarrow$	$\twoheadrightarrow$	$\twoheadrightarrow$	$\twoheadrightarrow$	$\twoheadrightarrow$
a	$\twoheadleftarrow$	$+$	$\mathbin{|||}$	$\twoheadrightarrow$	$\twoheadrightarrow$	$\twoheadrightarrow$
b	$\twoheadleftarrow$	$\mathbin{|||}$	$+$	$\twoheadrightarrow$	$\twoheadrightarrow$	$\twoheadrightarrow$
c	$\twoheadleftarrow$	$\twoheadleftarrow$	$\twoheadleftarrow$	$+$	$+$	$\twoheadrightarrow$
d	$\twoheadleftarrow$	$\twoheadleftarrow$	$\twoheadleftarrow$	$+$	$+$	$\twoheadrightarrow$
o	$\twoheadleftarrow$	$\twoheadleftarrow$	$\twoheadleftarrow$	$\twoheadleftarrow$	$\twoheadleftarrow$	$+$

(b) $BP|_w$

	i	a	b	c	d	o
i		$\prec$	$\prec$	$\prec$	$\prec$	$\prec$
a	$\prec^{-1}$			$\prec$	$\prec$	$\prec$
b	$\prec^{-1}$			$\prec$	$\prec$	$\prec$
c	$\prec^{-1}$	$\prec^{-1}$	$\prec^{-1}$		$\#$	$\prec$
d	$\prec^{-1}$	$\prec^{-1}$	$\prec^{-1}$	$\#$		$\prec$
o	$\prec^{-1}$	$\prec^{-1}$	$\prec^{-1}$	$\prec^{-1}$	$\prec^{-1}$	

(c) $BP|_{fes}$

Figure 4.7: Net system $\mathcal{N}_8$ and its behavioral profiles $BP|_w$ and $BP|_{fes}$

Figure 4.7 shows another WF-flow net system and its corresponding behavioral profiles. The net systems $\mathcal{N}_8$ and $\mathcal{N}_7$ (Fig. 4.6) are configuration equivalent, but have non-isomorphic $BP|_{fes}$ due to the flow relations derived from the additional places in $\mathcal{N}_8$. Interestingly, the $BP|_w$ of both net systems are isomorphic, and the difference in the set of flow relations $\prec$ is blurred by the definition of the strict order $\mapsto$ in $BP|_w$.

Armed with the results above, the next proposition shows that $BP|_w$ is behavior preserving for the $\eta_{\overline{\lambda}}$.

Theorem 4.9. *Let $N = (P, T, F)$ and $N' = (P', T', F')$ be WF-flow nets in $\eta_{\overline{\lambda}}$, and let $\Gamma : T \to T'$ be a bijection between the transitions. Let $\mathcal{N} = (N, M_0)$ and $\mathcal{N}' = (N', M_0')$ be the net systems and M_0 and M_0' be the corresponding initial markings. Thus the following holds:*

$$BP|_w(\mathcal{N}) \equiv_{iso} BP|_w(\mathcal{N}') \Leftrightarrow \mathcal{N} \approx_{conf} \mathcal{N}'.$$

Proof. ($\Rightarrow$) Firstly, let us show that if $BP|_w(\mathcal{N}) \equiv_{iso} BP|_w(\mathcal{N}')$ then $\mathcal{N} \approx_{conf} \mathcal{N}'$.

Suppose that $BP|_w(\mathcal{N}) \equiv_{iso} BP|_w(\mathcal{N}')$, but $\neg(\mathcal{N} \approx_{conf} \mathcal{N}')$. By Corollary 4.4, we have $\neg(BP|_w^{fes}(\mathcal{N}) \approx_{conf} BP|_w^{fes}(\mathcal{N}'))$ since $\neg(\mathcal{N} \approx_{conf} \mathcal{N}')$.

Assume a configuration $C \subseteq T$ in $BP|_w^{fes}(\mathcal{N})$ and its mapping $C' = \{\Gamma(t') \mid t' \in C\}$ in $\mathcal{N}'$, such that C' is not a configuration in $BP|_w^{fes}(\mathcal{N}')$. By Definition 3.37, the configuration C (i) is conflict free, (ii) for all $e'' \in C$ and $e \notin C$, s.t., $e \prec e''$ there exist an $e' \in C$ s.t. $e\#e' \prec e''$, and (iii) has no flow cycles. Then consider the following cases:

(i) Conflict freeness. Since C is a configuration in $BP|_w^{fes}(\mathcal{N})$, then for any $e, e' \in C$ it holds $\neg(e\#e')$ and, in consequence, $\neg(e + e')$ in $BP|_w(\mathcal{N})$ by Proposition 4.8(1). Then, by the assumption on the isomorphism of the $BP|_w$'s, $\exists e_1, e_1' \in C' : \Gamma(e) = e_1 \ \wedge \ \Gamma(e') = e_1'$, such that $\neg(e_1 + e_1')$ and thus $\neg(e_1\#e_1')$. So, C' is also conflict free iff C is conflict free.

(ii) For any $e_1'' \in C'$ and $e_1 \notin C'$, s.t., $e_1 \prec e_1''$, there exist an $e_1' \in C'$: $e_1\#e_1' \prec e_1''$. Suppose that there is an event $e_1 \notin C'$, such that $\exists e_1'' \in C' : e_1 \prec e_1''$ and $\forall e_1' \in C' : \neg(e_1\#e_1')$. Given that $e_1 \prec e_1''$, then $e_1 \mapsto e_1''$

(more specifically, $e_1 \twoheadrightarrow e_1''$), and since $\neg(e_1 \# e_1')$ then $\neg(e_1 + e_1')$ for any $e_1' \in C'$, by Proposition 4.8. Hence, by the isomorphism of BP_w's, $\exists e \notin C, e'' \in C : \Gamma(e) = e_1 \;\wedge\; \Gamma(e'') = e_1'' \;\wedge\; e \mapsto e''$ and for any $e' \in C$ it holds $\neg(e + e')$. However, the last contradicts the fact that C is a configuration in $BP|_w^{fes}(\mathcal{N})$, because e would necessarily be in C and thus $e_1 \in C'$. Hence, condition 2 also holds for C'.

(iii) Free of flow cycles. The only case remaining, so that C' is not a configuration in $BP|_w^{fes}(\mathcal{N}')$, is when C' contains cycles, i.e., $\prec_{C'}^{*}$ is not a partial order. This case simply cannot happen because WF-flow nets are acyclic and any firing sequence contains at most one occurrence of each activity.

Therefore, if C is a configuration in $BP|_w^{fes}(\mathcal{N})$, then C' must also be a configuration in $BP|_w^{fes}(\mathcal{N}')$. The reverse case follows analogously.

($\Leftarrow$) The opposite case, $\mathcal{N} \approx_{conf} \mathcal{N}' \Rightarrow BP|_w(\mathcal{N}) \equiv_{iso} BP|_w(\mathcal{N}')$, holds directly from the construction of the $BP|_w$ (Def. 4.6). □

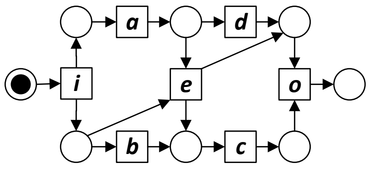
Finally, the next Corollary states the fact that $BP|_w$ is behavior-preserving for the class of nets $\eta_{\overline{\lambda}}$.

Corollary 4.10. *The behavioral profiles $BP|_w$ is behavior-preserving for the class $\eta_{\overline{\lambda}}$, w.r.t. configurations equivalence $\approx_{conf}$.*

The presented results also holds for the different extensions of the classic behavioral profiles that have strict and exclusive order relations, e.g., causal behavioral profile [Weid 11c] and behavioral profiles based on the relations of the $4C$ spectrum [Poly 14].

4.4 Expressing differences using $BP|_w$

A key consideration to use behavioral profiles for process model comparison is the clear interpretation of the relations. They can express patterns of behavior, such as strict order, exclusive order and interleaving in the case of $BP|_w$. For example, if a pair of tasks are in strict order relation in one model and exclusive order in another, then it is clear that in the first process the tasks can occur in the same computation and the occurrence of one follows the occurrence of the other; whereas they never occur together in the second process.



(a) $\mathcal{N}_9$

	a	b	c	d	e	i	o
a		$\vert\vert\vert$	$\vert\vert\vert$	$\twoheadrightarrow$	$\twoheadrightarrow$	$\twoheadleftarrow$	$\mapsto$
b	$\vert\vert\vert$		$\twoheadrightarrow$	$\vert\vert\vert$	$+$	$\twoheadleftarrow$	$\mapsto$
c	$\vert\vert\vert$	$\twoheadleftarrow$		$\vert\vert\vert$	$\twoheadleftarrow$	$\mapsfrom$	$\twoheadrightarrow$
d	$\twoheadleftarrow$	$\vert\vert\vert$	$\vert\vert\vert$		$+$	$\mapsfrom$	$\twoheadrightarrow$
e	$\twoheadleftarrow$	$+$	$\twoheadrightarrow$	$+$		$\twoheadleftarrow$	$\twoheadrightarrow$
i	$\twoheadrightarrow$	$\twoheadrightarrow$	$\mapsto$	$\mapsto$	$\twoheadrightarrow$		$\mapsto$
o	$\mapsfrom$	$\mapsfrom$	$\twoheadleftarrow$	$\twoheadleftarrow$	$\twoheadleftarrow$	$\mapsfrom$	

(b) $BP|_w$

Figure 4.8: Net system $\mathcal{N}_9$ and its behavioral profile $BP|_w$

Even though $BP|_w$ was shown to be behavior preserving for WF-flow nets without silent transitions, the interpretation of its relations can still be ambiguous and produce misleading diagnostics. Consider the WF-flow net system in Figure 4.8 and its behavioral profile $BP|_w$ aside. Let us draw you attention to transitions a and c, for which $BP|_w$ asserts an interleaving relation. However, in all the configurations where e occurs it is always the case that a precedes c. It is only in the set of configurations where e does not occur where a and c occur in any order. The fact is that these subtle kind of differences requires a diagnostic with contextual information[1].

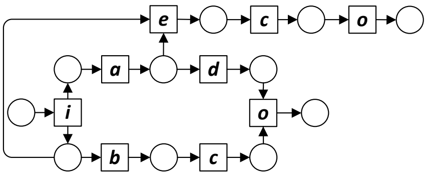


Figure 4.9: Branching process of net system $\mathcal{N}_9$ (Fig. 4.8a)

[1]It should be clear that it is possible to derive such sets of configurations from the $BP|_w^{fes}$.

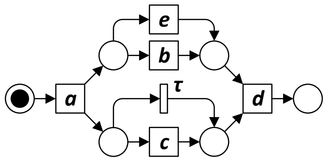
A solution to disambiguate the situation presented in Figure 4.8 is given by the branching processes, which reason not in terms of actions, but in terms of instances of actions. In a branching process, it is possible to define a single relation (causality, conflict or concurrency) between every pair of nodes. For instance, the branching process of the net system $\mathcal{N}_9$ (Fig. 4.8) is displayed in Figure 4.9, and it contains two instances of c, one which is preceded by a and one concurrent with a. The price to pay is that a branching process can contain several instances of a single activity, and the $O(|\Lambda|^2)$ size of the representation is no longer guaranteed.

Another approach to tackle the ambiguity of the $BP|_w$ is to use a larger set of behavioral relations. For instance, the $4C$ spectrum [Poly 14] defines a repertoire of eighteen basic behavioral relations that capture such behavioral phenomena as co-occurrence, conflict, causality, and concurrency. One can employ the relations of the $4C$ spectrum to construct a behavioral profile (guaranteeing a $O(|\Lambda|^2)$ size of the representation). Note that due to the nature of the $4C$ spectrum, a pair of tasks can be associated with several behavioral relations. Even though this approach solves the problem of the ambiguity for the family of unlabeled net systems, it falls short when trying to generalize the solution to the case of net systems with silent transitions (as discussed in the next section).

4.5 BP and silent transitions

This section extends the analysis of the behavioral profiles to labeled WF-flow nets, i.e., nets containing silent transitions. It is shown that for this class of nets neither the notion of classic behavioral profiles nor its extensions, including those based on the relations of the $4C$ spectrum, provide behavior-preserving representations of process models.

Proponents of classic behavioral profiles search for providing a representation that only considers the observable behavior. When it comes to representing labeled net systems, the common approach is to omit the columns and rows in the matrix that would be associated with silent transitions. This decision, however, comes with a loss of accuracy of the representation. E.g., consider the net system $\mathcal{N}_{10}$ in Fig. 4.10a. Its classic behavioral profile is isomorphic to the one of $\mathcal{N}_5$, cf. Fig. 4.1. However, $\mathcal{N}_{10}$ has two additional configurations: $\{a, b, d\}$ and $\{a, e, d\}$, w.r.t. $\mathcal{N}_5$.



(a) Net system $\mathcal{N}_{10}$

	a	b	c	d	τ	e
a		$\prec$	$\prec$		$\prec$	$\prec$
b	$\prec^{-1}$			$\prec$		#
c	$\prec^{-1}$			$\prec$	#	
d		$\prec^{-1}$	$\prec^{-1}$		$\prec^{-1}$	$\prec^{-1}$
τ	$\prec^{-1}$		#	$\prec$		
e	$\prec^{-1}$	#		$\prec$		

(b) $BP|_{fes}(\mathcal{N}_{10})$

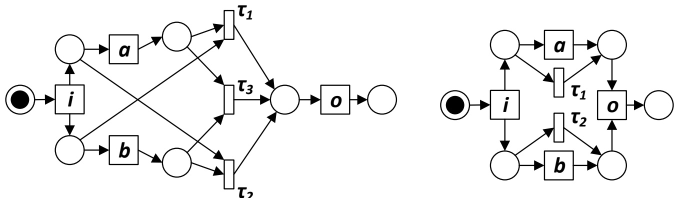
Figure 4.10: WF-flow net system and its $BP|_{fes}$

In order to preserve behavior, as for the case of unlabeled WF-flow nets, one possibility is to explicitly represent silent transitions in the matrix, as illustrated in Fig. 4.10b. It is easy to see that, using this approach, the behavior of $\mathcal{N}_5$ and $\mathcal{N}_{10}$ would be represented with two non-isomorphic matrices. However, this approach does not provide a complete solution since multiple net systems may exist with different numbers of silent transitions exhibiting the same observable behavior.

The use of a larger number of behavior relations can be seen as a way to tackle the above problem. For instance, both causal behavioral profiles and behavioral profiles based on the relations of $4C$ spectrum provide non-isomorphic representations for $\mathcal{N}_5$ and $\mathcal{N}_{10}$. However, none of them provides representations that distinguish the WF-flow net systems $\mathcal{N}_{11}$ and $\mathcal{N}_{12}$ in Fig. 4.11 w.r.t. configuration or trace equivalence.

$4C$ spectrum provides a vast family of behavioral relations. Although, the set of common configurations between $\mathcal{N}_{11}$ and $\mathcal{N}_{12}$, namely

$\{\{i,a,o\},\{i,b,o\},\{i,a,b,o\}\}$, gives rise to the same representation based on the relations of the $4C$ spectrum. Observe that there is only one configuration that distinguishes both systems: $\{i,o\}$. Then an interesting question is how accurate are the $4C$ spectrum relations and if it is worth to sacrifice some accuracy while preserving the $O(|\Lambda|^2)$ representation.

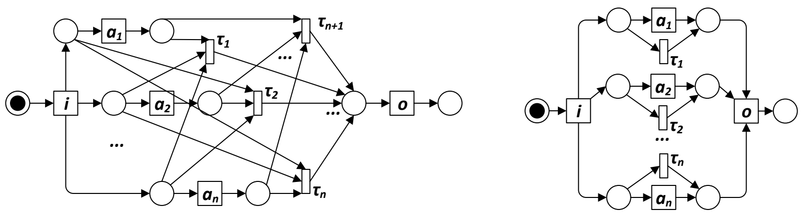


(a) $\mathcal{N}_{11}$ **(b)** $\mathcal{N}_{12}$

Figure 4.11: Net systems with isomorphic sets of $4C$ relations over labels

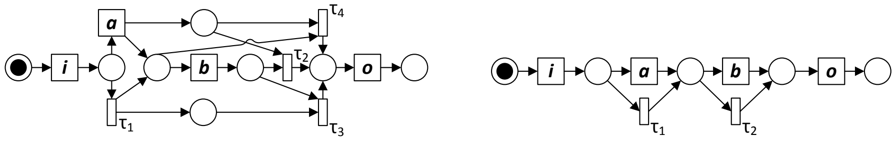
Figure 4.12 shows two constructions that generalize the net systems in Figure 4.11 with a variable number of transitions n. It turns out that, for any fixed value of $n \in \mathbb{N}_0$, the system $\mathcal{N}_{13}$ would comprise the set of configurations $\{\{i,a_1,a_2,\dots,a_n,o\}\} \cup \{\{i,a_m,o\} \mid m \in [1\,..\,n]\}$, however, it would have the same representation as the system $\mathcal{N}_{14}$ over the relations of the $4C$ spectrum. Note that system $\mathcal{N}_{13}$ encodes $n+1$ configurations, whereas system $\mathcal{N}_{14}$ describes 2^n configurations. Therefore, there exist two net systems for which there is an exponential number of configurations that are indistinguishable when using the representation based on the relations of the $4C$ spectrum. Specifically, $2^n - n - 1$ are indistinguishable configurations between the net systems in Figure 4.12. This fact is captured in the next proposition.

The counter example is not only for systems with concurrent behavior, the net systems $\mathcal{N}_{15}$ and $\mathcal{N}_{16}$ have the same $4C$ relations over labels. Although, the net system $\mathcal{N}_{16}$ describes an additional configuration $\{i,o\}$ which is not captured by the net system $\mathcal{N}_{15}$.



(a) Net system $\mathcal{N}_{13}$ **(b)** Net system $\mathcal{N}_{14}$

Figure 4.12: Generalization of the net systems in Fig. 4.11



(a) $\mathcal{N}_{15}$ **(b)** $\mathcal{N}_{16}$

Figure 4.13: Net systems with isomorphic sets of $4C$ relations over labels without concurrency

Proposition 4.11. *There exist two labeled WF-flow net systems that have the same $4C$ relations over labels, while the number of distinct configurations that the net systems describe differ in a value which is in the order of $O(2^n)$, where n is the number of distinct labels assigned to transitions of the net systems.*

The obtained results confirm that existing behavioral profiles are lossy behavioral representations of labeled net systems. So, if one relies on existing behavioral profiles for comparing process models, then one must tolerate inaccurate diagnosis.

4.6 Discussion

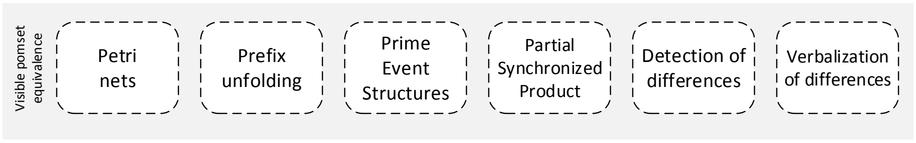
This section shows that, despite their efficient computation, behavioral profiles can be used to decide configuration equivalence only for a restricted family of acyclic and unlabeled net systems. Specifically, this result ceases to hold (for any currently known notion of behavioral profile) once transitions of net systems are allowed to ‘wear’ labels.

Event structures are a model of concurrency that represents processes by means of dependency relations and events. The events are occurrences of actions and the dependencies explain how the events relate each other. *Prime Event Structures* (PESs) [Niel 81, Wins 87] are one type of event structures where dependencies between events are reduced to causality and conflict (then a pair of event are concurrent if they are neither in causal nor in conflict relation). PESs solve the issues inherent to behavioral profiles. E.g., they have an execution semantics and every pair of tasks can be associated to a single relation that accurately describes the behavioral dependencies between them.

PES can be also represented as a matrix, but it can be considerably larger than $O(|\Lambda|^2)$, since a task may occur in many different computations. Indeed, the prime event structure of a process with cyclic behavior would contain an infinite amount of events. The following chapter proposes a technique to compute the PES of a process with cyclic behavior. To this end, we present an unfolding technique that captures all the causal dependencies between tasks. Then, we define a comparison technique to diagnose and explain behavioral differences using PESs.

Chapter 5

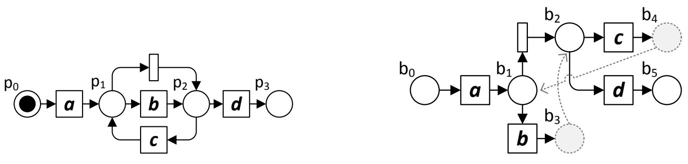
Process model comparison based on event structures


Visible pomset equivalence
Petri nets
Prefix unfolding
Prime Event Structures
Partial Synchronized Product
Detection of differences
Verbalization of differences

This chapter presents a comparison technique that takes pairs of (prime) event structures as input and produces natural language statements (using predefined templates) expressing encountered differences. A limitation of this technique is that it is not be applicable to process model with cycles, since the corresponding event structures have an infinite amount of events. Section 5.1 addresses this limitation and presents an unfolding technique to compute finite representations of cyclic process models. Section 5.2 introduces the partial synchronized product of a pair of (prime) event structures, which aims at finding similar and deviant behavior. By the same token, this section also shows how to verbalize encountered differences as natural language statements. Finally, Section 5.3 presents some discussions.

5.1 Finite representation of cyclic process models

A fundamental problem with cyclic process models is that their unfoldings are infinite. The seminal work in [McMi 95], later developed by many authors (see, e.g., [Espa 08] and citations therein) introduced sophisticated strategies for truncating the unfolding to a finite level, while keeping a representation of any reachable state, thus getting what is referred to as the *complete unfolding prefix (CP)*. In particular, the authors in [Khom 03] introduced a framework where a canonical unfolding prefix, complete with respect to a suitable property and not limited to reachability, can be constructed. Our own work relies on such a framework.



(a) $\mathcal{N}_{17}$ **(b)** β_1

Figure 5.1: Petri net system and its complete unfolding prefix

Consider the net system $\mathcal{N}_{17}$ and its complete unfolding prefix β_1 presented in Figure 5.1. Both conditions b_1 and b_4 correspond to the place p_1 in $\mathcal{N}_{17}$. To compute a complete unfolding prefix, we start applying the inductive rules in Figure 3.11 and stop the unfolding once we reach b_3 and b_4, roughly because any addition to the prefix would duplicate information already represented. Events b and c are called *cutoff events*. Even though this prefix includes a representation of all reachable markings and all executable transitions, in our setting, it does not include the information that we require to diagnose the behavioral differences of processes. E.g., the fact that c causally precedes b and d is not explicitly represented in this prefix.

We define a cutoff condition that is stronger than state reachability. Thus we obtain a larger prefix of the unfolding that makes explicit all the causal relations between tasks (visible transitions). In the case of the net system $\mathcal{N}_{17}$ in Figure 5.1a, the required unfolding prefix is β_2 in Figure 5.2. The *cutoff* conditions and their equivalent states are represented with the same color. This prefix describes all causal dependencies between tasks, thus it also captures the fact that an occurrence of b can be preceded by another occurrence of the same activity (e.g., events e_1 and e_7).

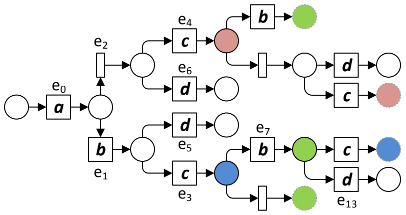


Figure 5.2: Complete unfolding prefix β_2

Formally, we resort to the notion of cutting context introduced in [Khom 03]. A cutting context is a tuple $\Theta = (\approx, \triangleleft, \mathcal{C})$ where $\approx$ is an equivalence relation over configurations, $\triangleleft$ is a total order over configurations, and $\mathcal{C}$ is the set of configurations used at the time of the computation of the unfolding prefix. E.g., the cutting context used in McMillan [McMi 95] is $\Theta_{McMillan} = (\approx_{mark}, \triangleleft_{size}, \mathcal{C}_{loc})$, where $\approx_{mark}$ equates two configurations when they produce the same marking, $\triangleleft_{size}$ is the total order induced by the size of configurations, and $\mathcal{C}_{loc} = \{\lfloor e \rfloor \mid e \in E\}$ is the set of local configurations. As already mentioned, the complete unfolding prefix β_1 is computed by using McMillan's cutting context. In fact, if we consider the local configurations $\lfloor c \rfloor = \{a, \tau, c\}$ and $\lfloor a \rfloor = \{a\}$, then one can easily check that $Mark(\lfloor a \rfloor) = Mark(\lfloor c \rfloor) = \{p_1\}$. Moreover, since $\|\lfloor a \rfloor\| < \|\lfloor c \rfloor\|$, then event c is a cutoff event. The cutting context in [Espa 02], denoted $\Theta_{ERV} = (\approx_{mark}, \triangleleft_{slf}, \mathcal{C}_{loc})$, differs from that in [McMi 95] for the definition of the partial order $\triangleleft_{slf}$, which is refined by considering action labels thus leading to more cut-offs and smaller prefixes (see [Espa 02] for details). For our purposes, consider a cutting context which is a modification of Θ_{ERV} with a refined equivalence relation over configurations taking into account also the tasks that produced the current marking. Roughly speaking, each token stores also the history of the events.

Definition 5.1 ($\approx_{Pred}$)**.** Let $\beta = (B, E, G, \rho)$ be a branching process of a labeled Petri net system with a net $N = (P, T, F, \lambda)$. A pair of configurations $C_1, C_2 \in Conf(\beta)$ are equivalent, represented as $C_1 \approx_{Pred} C_2$, iff $eMark(C_1) = eMark(C_2)$, where

$$eMark(C) = \{\langle \rho(b), \rho(\lfloor b \rfloor^{\Lambda}) \rangle \mid b \in Cut(C)\}.$$

We rely on the cutting context $\Theta_{Pred} = (\approx_{Pred}, \lhd_{slf}, \mathcal{C}_{loc})$. According to the theory in [Khom 03], once we have proved that the equivalence $\approx_{Pred}$ and the adequate order $\lhd_{slf}$ are preserved by finite configuration extensions, we immediately have an algorithm for constructing a canonical, finite prefix of the unfolding, complete with respect to the equivalence $\approx_{Pred}$. The latter means that for any configuration C in the full unfolding there will be a configuration C' in the finite prefix such that $C \approx_{Pred} C'$.

Since our cutting context is a slight variation of that in [Espa 02], we can rely on their work for the proof.

Proposition 5.2 (equivalence is preserved by extension)**.** *Let $\beta = (B, E, G, \rho)$ be the branching process of a net system $\mathcal{N}$, with a net $N = (P, T, F)$, and $C, C' \in Conf(\beta)$ be a pair of configurations, s.t. that $C \approx_{Pred} C'$. Therefore, for every suffix V of C, there exists a finite suffix V' of C' s.t.:*

$$C' \cup V' \approx_{Pred} C \cup V$$

Proof. Let C, C' be configurations such that $C \approx_{Pred} C'$ and let V be a suffix of C. We can assume that V consists of a single event, namely $V = \{e\}$. The general case easily follows by an inductive argument. This means that there is a transition t in N such that $\rho(e) = t$ and $Mark(C)[t\rangle$.

According to Definition 5.1, $eMark(C) = eMark(C')$, which in turn implies that $Mark(C) = Mark(C')$. Hence $Mark(C')[t\rangle$, which implies the existence of an extension $V' = \{e'\}$ of C', where $\rho(e') = t$.

Clearly $Mark(C \cup \{e\}) = Mark(C' \cup \{e'\})$. So, the fact that $eMark(C \cup \{e\}) = eMark(C' \cup \{e'\})$ is quite immediate. Take any condition $s' \in Cut(C' \cup \{e'\})$. There are two possibilities:

- $s' \in e'^{\bullet}$
  We have that $\lfloor s' \rfloor = \{e'\} \cup \bigcup_{s'' \in {}^{\bullet}e'} \lfloor s'' \rfloor$. Consider the only condition $s \in e^{\bullet}$ such that $\rho(s') = \rho(s)$. We have that

  $\rho(\lfloor s' \rfloor^{\Lambda}) = \{\rho(e') \mid \lambda(\rho(e')) \neq \tau\} \cup \bigcup_{s'' \in {}^{\bullet}e'} \rho(\lfloor s'' \rfloor^{\Lambda})$
  $= \{\rho(e) \mid \lambda(\rho(e)) \neq \tau\} \cup \bigcup_{s'' \in {}^{\bullet}e} \rho(\lfloor s'' \rfloor^{\Lambda})$ [since $\rho(e) = t = \rho(e')$ and $C \approx_{Pred} C'$]
  $= \rho(\lfloor s \rfloor^{\Lambda})$

  Therefore $\langle \rho(s), \rho(\lfloor s \rfloor^{\Lambda}) \rangle = \langle \rho(s'), \rho(\lfloor s' \rfloor^{\Lambda}) \rangle$.

- $s' \in Cut(C') \setminus {}^{\bullet}e'$
  In this case, if we take the only condition $s \in Cut(C) \setminus {}^{\bullet}e$ such that $\rho(s') = \rho(s)$, since $C \approx_{Pred} C'$, we immediately get that $\langle \rho(s), \rho(\lfloor s \rfloor^{\Lambda}) \rangle = \langle \rho(s'), \rho(\lfloor s' \rfloor^{\Lambda}) \rangle$.

Therefore we conclude that $eMark(C') \subseteq eMark(C)$. Since the argument is perfectly symmetric, we can deduce the converse inclusion, and thus equality. □

The following proposition shows that the canonical unfolding prefix constructed with Θ_{Pred} contains witnesses for all the causal relations that would be exhibited in the (possibly infinite) unfolding of a Petri net with cycles.

Proposition 5.3 (causal dependencies in the prefix). *Let $\mathcal{N}$ be a net system, let $\beta = (B, E, G, \rho)$ be its unfolding (i.e., $\beta = Unf(\mathcal{N})$) and let $\beta_\Theta = (B_\Theta, E_\Theta, G_\Theta, \rho_\Theta)$ be the CP based on the cutting context $\Theta_{Pred} = (\approx_{Pred}, \lhd_{slf}, \mathcal{C}_{loc})$. Then β_Θ is "complete with respect to causal dependencies", i.e., for any pair of events $e_1, e_2 \in E : e_1 <^{\beta} e_2$ then*

$$\exists e_1', e_2' \in E_\Theta : e_1' <^{\beta_\Theta} e_2', \text{ where } \rho(e_1) = \rho_\Theta(e_1') \text{ and } \rho(e_2) = \rho_\Theta(e_2').$$

Proof. Let $e_1, e_2 \in E^{\Lambda}$ be events of the unfolding such that $e_1 <^{\beta} e_2$. This means $e_1 \in \lfloor e_2)$. Consider the configuration $C = \lfloor e_2)$. By completeness there is a configuration C' in the prefix such that $eMark(C) = eMark(C')$. Certainly $Mark(C') = Mark(C)$ enables $\rho(e_2)$ hence C' admits an extension with event e_2' such that $\rho_{\Theta}(e_2') = \rho(e_2)$. Moreover, since $e_1 <^{\beta} e_2$ there is a condition $s \in {}^{\bullet}e_2 \cap Cut(C)$ such that $e_1 <^{\beta} s$ and thus $\rho(e_1) \in \rho(\lfloor s \rfloor^{\Lambda})$. If we take the only condition $s' \in Cut(C')$ such that $\rho(s) = \rho_{\Theta}(s')$, we have that $s' \in {}^{\bullet}e_2'$ and, since $eMark(C) = eMark(C')$, it holds that $\langle \rho_{\Theta}(s'), \rho_{\Theta}(\lfloor s' \rfloor^{\Lambda}) \rangle = \langle \rho(s), \rho(\lfloor s \rfloor^{\Lambda}) \rangle$. This means that there is $e_1' \in \lfloor s' \rfloor$ such that $\rho_{\Theta}(e_1') = \rho(e_1)$. Note that $e_1' \in \lfloor s' \rfloor$ means $e_1' <^{\beta_{\Theta}} s'$, whence $e_1' <^{\beta_{\Theta}} e_2'$, as desired. □

5.1.1 Multiplicity of activities

The unfolding prefix described above captures the fact that a task can occur multiple times in a single run and thus a notion of multiplicity can be attached to each task. Then if an event in the unfolding prefix, which is an occurrence of task a, can be preceded by another occurrence of a, then this activity can occur more than once in a computation. The multiplicity of tasks helps differentiating the behavior of isomorphic unfolding prefixes steaming from non-equivalent net systems. The last since the unfolding prefix of a cyclic net can be isomorphic to an acyclic one (i.e., an occurrence net) with duplicate labels.

We now show how to identify the multiplicity of each task (i.e., labeled transition in the original net) given the canonical unfolding prefix induced by Θ_{Pred}. To lighten the notation in the remaining of this subsection, we omit the super index of the relations of the branching process β, i.e., $<^{\beta}$ is simply represented as $<$.

Since we deal with safe nets, we observe that if a transition can occur twice in a configuration, the corresponding events must be causally related.

Proposition 5.4 (repetition). *Let β be a branching process of a net system $\mathcal{N}$, with a net $N = (P, T, F)$. Let $C \in Conf(\mathcal{N})$ be a configuration such that* *there exists* $e, e' \in C$, $e \neq e'$ *and* $\rho(e) = \rho(e') = t \in T$. *Then either* $e < e'$ *or* $e' < e$.

Proof. Observe that $e \# e'$ cannot hold, otherwise C would not be a configuration. If we had neither $e < e'$ nor $e' < e$, then e and e' would be concurrent. As a consequence also $^\bullet e \cup {}^\bullet e'$ would be concurrent. Therefore, the corresponding marking in $\mathcal{N}$ would be coverable and it would have two tokens in any place in $^\bullet t$, contradicting the assumption that $\mathcal{N}$ is safe. □

The above observation motivates the interest for the following definition in the study of repetitive behaviors.

Definition 5.5 (self-preceding transitions)**.** Let $\beta = (B, E, G, \rho)$ be the unfolding prefix induced by Θ_{Pred} for a net $N = (P, T, F, \lambda)$. The set of *self-preceding transitions* of N is defined as $\mathcal{R} = \{\rho(e_1) \mid \exists C \in Conf(\beta).\ e_1, e_2 \in C\ \wedge\ \rho(e_1) = \rho(e_2) \wedge e_1 < e_2\}$.

Note that the possibility of reducing the repetition to a causal dependency ensures that the finite prefix (which contains full information about causal dependencies) will be also sufficient to identify repeated events. As an example it can be checked that $C = \{e_0, e_2, e_6\}$, $C' = \{e_0, e_1, e_5\}$ and $C'' = \{e_0, e_1, e_3, e_7, e_{13}\}$ are configurations in the unfolding prefix β_2 from Figure 5.2. Activity b is part of repetitive behavior as C'' includes two (causal dependent) occurrences of b. This holds despite the fact that there are (maximal) configurations including only a single occurrence of b (like C) or none (like C').

Definition 5.5 tells us which transitions in the original net system may occur more than once. By the same token, we can determine which transitions occur at least once. The latter correspond to events that occur in the intersection of all completed configurations.

Definition 5.6 (necessary transitions)**.** Let $\beta = (B, E, G, \rho)$ be the unfolding prefix induced by Θ_{Pred} for a net system $\mathcal{N}$. The *necessary transitions* $\mathcal{K}$ of $\mathcal{N}$ is defined as $\mathcal{K} = \bigcap \varrho(MaxConf(\beta))$.

Based on the above definitions of $\mathcal{R}$ and $\mathcal{K}$, we classify transitions in a net system into three disjoint categories: those fired "0 or more times" (denoted as "$\ast$"); "1 or more times" (denoted as "+"); and at most once "0..1". Formally:

Definition 5.7 (multiplicity of a transition)**.** Let $\beta = (B, E, G, \rho)$ be the unfolding prefix induced by Θ_{Pred} for a net system $\mathcal{N}$, the multiplicity of a labeled transition is defined as:

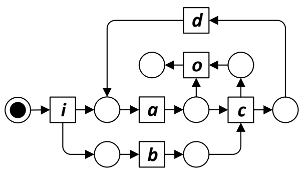
- $0..1 = \{e \in E \mid \rho(e) \notin \mathcal{R}\}$
- $+ = \{e \in E \mid \rho(e) \in \mathcal{R} \cap \mathcal{K}\}$
- $\ast = \{e \in E \mid \rho(e) \in \mathcal{R} \setminus \mathcal{K}\}$

Observe that if we are interested in the multiplicity of tasks, namely transition labels, rather than of transition themselves (this makes a difference if the labeling in the net is not injective), we need some adjustments to the definitions above. More precisely, the sets of labels corresponding to $\mathcal{R}$ and $\mathcal{K}$ above are

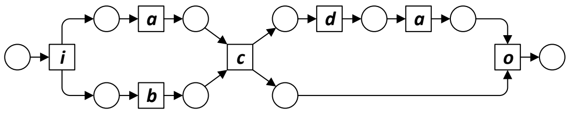
- $\Lambda\mathcal{R} = \{a \in \Lambda \mid \exists C \in \mathit{MaxConf}(\beta).\ |(\lambda \circ \rho)^{-1}(a) \cap C| \geq 2\}$,
  namely labels that can occur more than once in a computation are those that occur more than once in a maximal configuration of a branching process generated with Θ_{Pred} (this is, in general, a superset of $\lambda(\mathcal{R})$);

- $\Lambda\mathcal{K} = \bigcap \lambda(\varrho(\mathit{MaxConf}(\beta)))$
  namely labels that necessarily appear in a computation are those that occur in any maximal configuration of the branching process generated with Θ_{Pred}.

In the case a transition may be fired "one or more times" or "zero or more times" ("+" or "$\ast$"), the above definition does not tell us whether the transition can be repeated an unbounded or a bounded number of times. E.g., consider the net system and its unfolding in Figure 5.3, observe that



(a) $\mathcal{N}_{18}$



(b) β_3

Figure 5.3: Non-free choice "cyclic" net system and its unfolding

the multiplicity of activity a is "+", but it can occur at most twice in a computation.

Below, we refine the notion of multiplicity for a class of workflow nets, for which we show that when a task is classified as "+" or "$\star$", it means it can be fired any number of times.

5.1.2 Multiplicity of activities in free-choice workflow nets

Transitions which, according to Definition 5.5 are marked as repetitive, namely either "+" or "*" can surely occur more than once in a computation, but still they could occur at most a bounded number of times (e.g., Fig. 5.3). We next show that if we focus on the class of *sound free-choice workflow nets* [van 97], a transition which is marked as "+" or "*", may fire any number of times, namely it is part of a cyclic behavior.

We show that for the class of (safe) free-choice sound WF-nets, the self-preceding transitions captured by the proposed cutting context $\Theta_{Pred} = (\approx_{Pred}, \triangleleft_{slf}, \mathcal{C}_{loc})$, namely those transitions marked as "*" or "+" according to Definition 5.5 represent unbounded repetitive behavior.

We first need a preliminary technical result.

Lemma 5.8 (sequences of firings)**.** *Let $\mathcal{N}$ be a sound free-choice WF-net system. Let $t_0, \ldots, t_n$ be transitions such that $t_i^\bullet \cap {}^\bullet t_{i+1} \neq \varnothing$ for any $i \in \{0, \ldots, n-1\}$ and let M be a marking such that $M[t_0\rangle$. Then there are sequences of transitions $\sigma_i \in T^*$, $i \in \{0, \ldots, n-1\}$, such that $M[t_0\sigma_0 t_1 \sigma_1 \ldots \sigma_{n-1} t_n\rangle$.*

Proof. The proof is by induction on n. The base case $n = 0$ is trivial. Let us assume the result for n and prove it for $n + 1$. By inductive hypothesis there are $\sigma_0, \ldots, \sigma_n$ such that $M[t_0\sigma_0 t_1\sigma_1 \ldots \sigma_{n-1}t_n\rangle M_n$. Moreover, by hypothesis, there is at least one place $p \in t_n{}^\bullet \cap {}^\bullet t_{n+1}$ and we know that $p \in M_n$. Since $\mathcal{N}$ is a sound WF-net, from marking M_n there is a firing sequence which leads to a marking consisting of one token only in the sink place

$$M_n[\sigma\rangle\{o\}.$$

Since $p \in {}^\bullet t_{n+1}$, surely $p \neq o$. Hence the token in p is consumed by some transition in σ, namely $\sigma = \sigma' t \sigma''$ with $p \in {}^\bullet t$.

Since $\mathcal{N}$ is free-choice, and ${}^\bullet t \cap {}^\bullet t_{n+1} \supseteq \{p\} \neq \varnothing$ we deduce ${}^\bullet t = {}^\bullet t_{n+1}$. Therefore, since $M_n[\sigma' t\rangle$ we also have $M_n[\sigma' t_{n+1}\rangle$. Therefore

$$M[t_0\sigma_0 t_1\sigma_1 \ldots \sigma_{n-1}t_n\sigma' t_{n+1}\rangle$$

as desired. □

We can now easily conclude with the desired result.

Proposition 5.9. *Let $\mathcal{N}$ be a free-choice sound WF-net and let t be a transition marked as repetitive ("*" or "+"). Then there are firings sequences in which transition t fires any number of times.*

Proof. Let t be a transition marked as repetitive ("*" or "+"). This means that there are events e, e' in the prefix β_Θ, such that $\rho(e) = \rho(e') = t \wedge e < e'$. We show that for any marking M, such that $M[t\rangle$, there is a sequence $\sigma \in T^*$ such that $M[t\sigma t\rangle$. From this the result immediately follows.

Since $e < e'$, there must be a causal chain of $e = e_0 < e_1 < \ldots < e_n = e'$ such that $e_i{}^\bullet \cap {}^\bullet e_{i+1} \neq \varnothing$ for any $i \in \{0, \ldots, n-1\}$. Therefore, if we consider the image through ρ in $\mathcal{N}$, we get corresponding sequence of transitions $\rho(e_0) = t_0 = t$, $\rho(e_1) = t_1$, $\ldots$, $\rho(e_n) = t_n = t$, with $t_i{}^\bullet \cap {}^\bullet t_{i+1} \neq \varnothing$ for $i \in \{0, \ldots, n-1\}$.

Now, given any marking M such that $M[t\rangle$, we can simply apply Lemma 5.8, to deduce that there are $\sigma_i \in T^*$, $i \in \{0, \ldots, n-1\}$, such that

$$M[t_0\sigma_0 t_1\sigma_1 \ldots \sigma_{n-1}t_n\rangle.$$

recalling that $t = t_0 = t_n$ and denoting $\sigma = \sigma_0 t_1 \sigma_1 \ldots \sigma_{n-1}$, we get the desired result. □

Again, the theory can be adapted if the labeling of the net is not injective and we are interested in the repetition of labels (tasks) rather than transitions. In this case we can distinguish between labels that can occur more than one time in a computation (the class "*" defined as before) and the subclass of those which can occur an unbounded number of times in a computation, defined as $\lambda(\mathcal{R})$.

The insights we got from this section are used later in the verbalization of differences involving repeated tasks (Section 5.2.3).

5.2 Comparison based on event structures

This section presents a comparison technique that takes pairs of event structures and produces natural language statements expressing encountered differences. The computed differences reflect binary behavioral relations between events and repetition of tasks that hold in one model but not in the other. The comparison technique adopts completed visible-pomset as the notion of equivalence and is formulated in terms of families of pomsets, thus it is applicable to various flavors of event structures (e.g., prime, asymmetric and flow event structures).

As a starting point we consider the comparison of PESs, such that behavioral differences between a pair of process models are expressed in terms of causality, conflict and concurrency binary relations. Before presenting the comparison technique, we define the PES of a Petri net system and put forward two different approaches to handle silent events present in the event structures.

The PES of a Petri net system $\mathcal{N}$ is basically the unfolding $Unf(\mathcal{N})$ without the conditions and relations over conditions. However, in order to tackle the problem of infinite unfoldings (as explained in the previous section), we consider the branching process computed with the cutting context Θ_{Pred}. The formal definition of the PES of a net system is as follows.

Definition 5.10 (PES of a net system)**.** Let $\mathcal{N} = (N, M_0)$ be a net system, where $N = (P, T, F, \lambda)$, and $\beta_\Theta = (B, E, G, \rho)$ be its branching process computed with Θ_{Pred}. The labeled *Prime Event Structure* (PES) of β_Θ is defined as $\mathbb{P} = \langle E, \leq, \#, \lambda' \rangle$ where $\leq = \leq^{\beta}_{|E}$ and $\# = \#^{\beta}_{|E}$. Finally, $\lambda' = \lambda \circ \rho$ is a labeling function that associates each event $e \in E$ with the label of its corresponding transition $t \in T$, i.e., $\lambda'(e) = \lambda(\rho(e))$.

The differences between a pair of process models can include both observable and unobservable behavior. Although, reporting differences involving silent transitions or events can result irrelevant to the user. For this reason, the proposed technique considers only the observable events in the PESs. This approach precludes the adoption of an equivalence notion in branching semantics. For instance, Figure 5.4 shows a pair of non-history preserving bisimilar PESs, but completed visible-pomset equivalent, which differ only in the silent behavior (i.e., τ event).

$$\begin{tikzcd}
 & a \arrow[d] & \\
 & b \arrow[dl] \arrow[dr] & \\
c \arrow[r, dash, dotted] & \# \arrow[r, dash, dotted] & \tau \arrow[d] \\
 & & d
\end{tikzcd}$$

(a) $\mathbb{P}'$

$$\begin{tikzcd}
 & a \arrow[d] & \\
 & b \arrow[dl] \arrow[dr] & \\
c \arrow[r, dash, dotted] & \# \arrow[r, dash, dotted] & d
\end{tikzcd}$$

(b) $\mathbb{P}'_{|\Lambda}$

Figure 5.4: Non hp-bisimilar PESs, but completed visible-pomset equivalent

As a reference behavioral equivalence, we use a variation of *pomset equivalence*, which ignores invisible events and is sensible to termination [Glab 89, Golt 94]. Roughly speaking, it equates systems which can execute the same visible activities, with identical relations of causal dependency and concurrency.

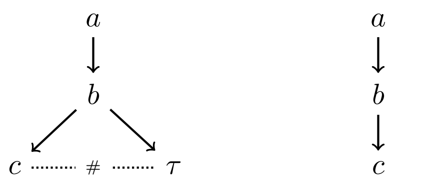
We can also take a more radical solution which consists in directly removing the silent events from the PES, keeping only the visible events and their dependencies. We adopt this latter approach.

The following definition is the restriction of a PES to its observable behavior.

Definition 5.11 (restriction of PES to Λ)**.** Let $\mathbb{P} = \langle E, \leq, \#, \lambda \rangle$ be a labeled PES, then the *restriction of PES* to observable behavior is defined as $\mathbb{P}^{\Lambda} = \langle E', \leq', \#', \lambda' \rangle$, where $E' = \{e \in E \mid \lambda(e) \neq \tau\}$, $\leq' = \leq_{|E'}$, $\#' = \#_{|E'}$ and $\lambda' = \lambda_{|E'}$.

In specific cases, the restriction of a PES to its observable behavior can lead to lose conflicts and maximal configurations. For example, the restriction of $\mathbb{P}''$ in Figure 5.5 "loses" the (visible) maximal configuration $\{a, b\}$ in $\mathbb{P}^{\Lambda}{}''$. The last is due to the fact that τ is the only event in conflict with c in $\mathbb{P}''$.

It is easy to see that the problem does not occur if we consider process models where silent events can never be maximal in a configuration, i.e., intuitively, where a silent event is never the last event of a computation.



(a) $\mathbb{P}''$ **(b)** $\mathbb{P}^{\Lambda}{}''$

Figure 5.5: PES and its restriction to observable behavior

Proposition 5.12 (restriction of PES to Λ preserves $\equiv_{cp}$)**.** *Let $\mathbb{P} = \langle E, \leq, \#, \lambda \rangle$ be a PES such that for any $C \in MaxConf(\mathbb{P})$, no event in $C \setminus C^{\Lambda}$ is $\leq$-maximal in C. Then $\mathbb{P} \approx_{cp} \mathbb{P}^{\Lambda}$.*

Proof. It is immediate to see that, under the hypothesis, for any $C \in \mathit{MaxConf}(\mathbb{P})$ we have $C = \bigcup_{e \in C^{\Lambda}} \lfloor e \rfloor_{\mathbb{P}} \in \mathit{Conf}(\mathbb{P})$ (where $\lfloor e' \rfloor_{\mathbb{P}}$ denotes the set of causes of e' in $\mathbb{P}$).

Now, if $C \in \mathit{MaxConf}(\mathbb{P})$ in order to conclude that $C^{\Lambda} \in \mathit{Conf}(\mathbb{P}^{\Lambda})$ is maximal, observe that if $C^{\Lambda} \subseteq C'$ for some $C' \in \mathit{Conf}(\mathbb{P}^{\Lambda})$ then, by the above, $C = \bigcup_{e \in C^{\Lambda}} \lfloor e \rfloor_{\mathbb{P}} \subseteq \bigcup_{e \in C'} \lfloor e \rfloor_{\mathbb{P}}$. Since the latter is a configuration in $\mathbb{P}$, by maximality of C we have $C = \bigcup_{e \in C'} \lfloor e \rfloor_{\mathbb{P}}$ hence $C^{\Lambda} = (\bigcup_{e \in C'} \lfloor e \rfloor_{\mathbb{P}})^{\Lambda} = C'$, as desired.

Vice versa, if $C^{\Lambda} \in \mathit{MaxConf}(\mathbb{P}^{\Lambda})$, in order to conclude that $C \in \mathit{Conf}(\mathbb{P})$ is maximal, observe that if $C \subseteq C_1$ for some $C_1 \in \mathit{Conf}(\mathbb{P})$ then $C^{\Lambda} \subseteq C_1^{\Lambda}$. By maximality of C^{Λ} this means that $C^{\Lambda} = C_1^{\Lambda}$. Therefore $C = \bigcup_{e \in C^{\Lambda}} \lfloor e \rfloor_{\mathbb{P}} = \bigcup_{e \in C_1^{\Lambda}} \lfloor e \rfloor_{\mathbb{P}} = C_1$, as desired. □

The comparison technique introduced in this chapter is compatible with both approaches, namely, after the translation of a process model to a PES one can either consider the underlying visible PES (if the transformation is known to preserve the behavior) or keep the silent events and ignore them in the later stage of the comparison.

The presence of different activities reduces, at the level of event structures, to the presence of events with different labels, which are easy to detect and describe. Instead, properly diagnosing and reporting differences in the way common activities (i.e., events carrying the same label in both process models) are related in the process is a more complex problem.

A PES can be seen as a labeled graph, where events are nodes and relations are edges. Thus, if two PES are diagnosed as isomorphic, it seems sensible to conclude that they are behaviorally equivalent. Moreover, if an error-correcting graph matching is used, the same algorithm would gather the information about the differences on event occurrences (process activities) and mismatching behavior relations. Unfortunately, a conventional approximate graph matching technique would not take into account the order induced by the behavioral relations.

$$\begin{tikzcd}
 & a \arrow[dl] \arrow[dr] & \\
b(1) \arrow[r, dotted, no head] & \# \arrow[r, dotted, no head] & c(2)
\end{tikzcd}$$

(a) $\mathbb{P}_1$

$$\begin{tikzcd}
a_1 \arrow[r, dotted, no head] \arrow[d] & \# \arrow[r, dotted, no head] & a_2 \arrow[d] \\
b(1) & & c(2)
\end{tikzcd}$$

(b) $\mathbb{P}_2$

Figure 5.6: Example of graph-based PES comparison

Figure 5.6 shows a pair of completed visible-pomset equivalent PESs and a (partial) mapping between the events, which is given by the numbers in the parenthesis. Note that there are two possible mappings for the event a in $\mathbb{P}_2$. Although, any mapping of a in $\mathbb{P}_1$, either with a_1 or a_2, will spot the fact that there is another instance of a in $\mathbb{P}_2$ that cannot not be mapped. Thus, relying on graph isomorphism techniques and reporting differences based on mappings and mismatches of nodes can lead to inaccurate diagnosis.

One additional concern is to provide a systematic approach to produce intuitive diagnostics describing the differences found while comparing a pair of PESs. Therefore, in the remainder of this section we present the elements of our approach to compare event structures: *matching behavior*, *identifying differences* and *verbalizing differences*.

5.2.1 Partial synchronized product

The first challenge is to determine the behavior similarity between a pair of event structures. We adopt completed visible-pomset equivalence as the reference notion for the comparison. So, if two event structures exhibit different behavior (due to differences in the set of events or in the underlying behavioral relations), it is clear that their corresponding visible-pomsets would differ as well. Hence, we are interested in finding the best (or at least a good) approximated behavioral matching between both event structures.

We start by introducing the concept of partial match between two configurations, which is intended to capture the idea of an approximated isomorphism between the corresponding visible-pomsets. Note that the definitions in this subsection are based on the notion of families of pomsets, thus they apply to the different flavors of event structures. Let us indicate that a partial function f is undefined on x as $f(x) = \perp$.

Definition 5.13 (partial match)**.** Let $\mathcal{P}_1 = \langle E_1, \mathit{Conf}(\mathcal{P}_1), \lambda_1 \rangle$ and $\mathcal{P}_2 = \langle E_2, \mathit{Conf}(\mathcal{P}_2), \lambda_2 \rangle$ be a pair of families of pomsets and let $C_i \in \mathit{Conf}(\mathcal{P}_i)$, for $i \in \{1, 2\}$ be configurations. A *partial match* between C_1 and C_2 is a partial injective function $\xi : C_1 \nrightarrow C_2$, such that for all $e_1, e_1' \in E_1$, with $\xi(e_1), \xi(e_1') \neq \perp$, the following holds:

1. $\lambda_2(\xi(e_1)) = \lambda_1(e_1)$
2. $e_1 \leq_1 e_1'$ iff $\xi(e_1) \leq_2 \xi(e_1')$

A partial match is a function ξ that establishes a correspondence between events of the two pomsets, respecting both labeling and order. Note that *partial match* is a partial and non surjective function, meaning that some events in C_1 may not have a mapping to any event in C_2, and vice versa.

A partial match between configurations can be thought as the result of applying two operations over "growing" pomsets

1. *matching* of events (both pomsets synchronously evolve a pair of events that have the same label), and
2. *hiding* of an event (only one pomset evolves with a single event while the other remains the same).

Matching and hiding operations can be expressed as inductive rules, as shown if Figure 5.7, that applied to a partial match ξ between C_1 and C_2 produce another partial match involving larger configurations. Since the same partial match can be associated with different pairs of configurations, we write (C_1, ξ, C_2) to refer to ξ seen as a partial match between C_1 and C_2. Finally, we write $C \xrightarrow{e}_{\lambda(e)} C \cup \{e\}$ to denote $C \cup \{e\} \in \mathit{Conf}(\mathcal{P})$, for a configuration $C \in \mathit{Conf}(\mathcal{P})$ and an event $e \notin C$. Note that, in case silent events have not been removed from the families of pomsets during their extraction from the process, the hiding operations are used also to ignore silent events.

$$\frac{C_1 \xrightarrow{e_1}_a C_1' \quad C_2 \xrightarrow{e_2}_a C_2' \quad \xi' = \xi[e_1 \mapsto e_2] \text{ partial match}}{(C_1, \xi, C_2) \xrightarrow{\mathit{match}\ e_1, e_2} (C_1', \xi', C_2')}\ \mathit{match}\ e_1, e_2$$

$$\frac{C_1 \xrightarrow{e_1}_a C_1'}{(C_1, \xi, C_2) \xrightarrow{\mathit{hide}\ e_1} (C_1', \xi, C_2)}\ \mathit{hide}\ e_1$$

$$\frac{C_2 \xrightarrow{e_2}_a C_2'}{(C_1, \xi, C_2) \xrightarrow{\mathit{hide}\ e_2} (C_1, \xi', C_2')}\ \mathit{hide}\ e_2$$

Figure 5.7: Partial matching operations given a partial match (C_1, ξ, C_2)

An example of partial matches between configurations is depicted in Figure 5.8. The first and second rows show a pair of PESs $\mathbb{P}_3$ and $\mathbb{P}_4$, and their corresponding visible-pomsets aside framed in a gray box. The second row shows a pair of partial matches between the configurations of both pomsets, the hidden events and their relations are highlighted in gray, whereas the matches are in black. Formally, the matchings in Figure 5.8 are denoted as $(\{a, b, c\}, [a \mapsto a, c \mapsto c], \{a, c\})$ and $(\{a, b, c\}, [b \mapsto b], \{b\})$.

Starting from the above concepts, we aim at defining a technique that allows to optimize the matching of pomsets or, equivalently, to minimize the number of hiding operations in a partial match. Clearly, whenever it is possible to establish a mapping between the pomsets of two families of pomsets using only *matching* operations, those families will be equivalent. Conversely, when the families of pomsets are not equivalent then the optimal match of their pomsets –the one with the minimum number of hidings– would capture both the largest approximate visible-pomset (or

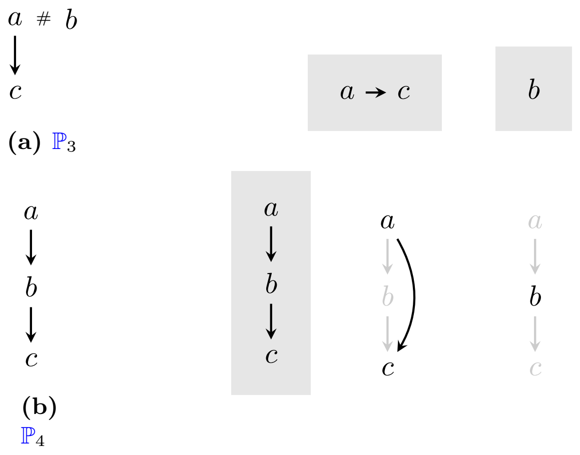


Figure 5.8: PESs and a pair of partial matches between their configurations

common behavior) and the corresponding differences, in the form of hiding operations.

Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be families of pomsets, and $C_1 \in \mathit{Conf}(\mathcal{P}_1)$ and $C_2 \in \mathit{Conf}(\mathcal{P}_2)$ be a pair of configurations. The "quality" of a partial match $s = (C_1, \xi, C_2)$ is captured by a value $g(s)$

$$g(s) = |C_1^{\Lambda}| + |C_2^{\Lambda}| - |\xi| \cdot 2. \tag{5.1}$$

The function $g(s)$ above is aimed at quantifying the "quality" of the matchings between a pair of pomsets. When $g(s) = 0$, then ξ is a visible-pomset isomorphism between pomsets C_1 and C_2. When $g(s) > 0$ the partial match ξ required one or more hiding operations. This case can be interpreted as an approximate (or non complete) visible-pomset isomorphism of pomsets C_1 and C_2. E.g., the quality of the partial matches in Figure 5.8 are $g(\{a,b,c\},[a \mapsto a, c \mapsto c],\{a,c\}) = 1$ and $g(\{a,b,c\},[b \mapsto b],\{b\}) = 2$.

Given two families of pomsets $\mathcal{P}_1$ and $\mathcal{P}_2$, for any two configurations $C_1 \in \mathit{Conf}(\mathcal{P}_1)$ and $C_2 \in \mathit{Conf}(\mathcal{P}_2)$ there is always a partial match. However, only a subset of the possible partial matches would have a minimum cost; those partial matches are said optimal.

Definition 5.14 (optimal match)**.** Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be families of pomsets and let $C_i \in \mathit{Conf}(\mathcal{P}_i)$, for $i \in \{1,2\}$ be configurations. A partial match $s = (C_1, \xi, C_2)$, where $\xi : C_1 \nrightarrow C_2$, is called *optimal* when

$$g(s) = \min\{g(s) \mid \xi' : C_1 \nrightarrow C_2\}$$

The partial matches between configurations of two families of pomsets can be collected in what we call a partial synchronized product, defined in the line of [Adri 13] but for pairs of families of pomsets.

Definition 5.15 (partial synchronized product)**.** Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be families of pomsets. The *partial synchronized product* is the graph $G = \langle S, T \rangle$ where:

- S is the set of triples (C_1, ξ, C_2), where $\xi : C_1 \nrightarrow C_2$ is a partial match;
- T is the set of transitions $(C_1, \xi, C_2) \xrightarrow{op} (C_1', \xi', C_2')$ defined by the rules in Figure 5.7.

It is immediate to see that the partial synchronized product is inductively built starting from an "initial" node $(\varnothing, \varnothing, \varnothing)$ corresponding to the unique partial matching for the empty configurations, and then expanding the graph by using the rules in Figure 5.7. Note that the hiding (*hide* e_1 and *hide* e_2) operations can only increase the cost g of a partial match by one (when the hidden event is visible) or leave it unchanged (when the hidden event is silent); whereas the *match* e_1, e_2 always leave the cost unchanged. Therefore, whenever $(C_1, \xi, C_2) \xrightarrow{op} (C_1', \xi', C_2')$, then $g((C_1, \xi, C_2)) \leq g((C_1', \xi', C_2'))$. This fact, when searching for optimal partial matches, allows for some pruning in the generation of the partial synchronized product.

The partial synchronized product obviously contains all optimal matches (as it contains all partial matches). However, the size of a partial synchronized product is exponential, making its full construction computationally unfeasible.

We adopt a branch an bound approach, more specifically an adaptation of the well-known A^* algorithm [Hart 68], in order to build an informative part of the partial synchronized product. As usual, the A^* algorithm requires two cost functions: one to evaluate the cost from the root of the state space to a given path, referred to as the function g or past-cost function, and a heuristic function to estimate the distance to the goal state, referred to as the function h or future-cost function.

Given a partial match $s = (C_1, \xi, C_2)$, the past-cost function $g(s)$ is that defined in Equation 5.1. Let $E'_i = \bigcup_{C_i \sqsubseteq C'} C' \setminus C_i$ be the set of all possible extensions of C_i, where $i \in \{1,2\}$ and C' is any configuration extending C_i. The future cost function $h(s)$ is shown in Equation 5.2.

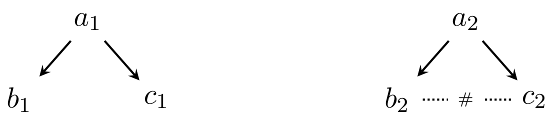
$$h(s) = |(\lambda(E'_1) \cup \lambda(E'_2)) \setminus (\lambda(E'_1) \cap \lambda(E'_2))| \tag{5.2}$$

Intuitively, h provides a measure of the number of events to be hidden in the future of C_1 and C_2. It optimistically assumes that events with the same label will indeed contribute to a one-to-one match between the two configurations. It can be seen that this estimate is admissible in the sense required in [Dech 85] for the use of the algorithm A^*. Specifically, given a partial match $s = (C_1, \xi, C_2)$, in order to match all the events in the extensions of C_1 and C_2, then it is necessary to hide at least the events with different labels (labels seen only in the extension of either C_1 or C_2). It does not necessarily mean that the partial matches obtained using the function h will be optimal, since a single partial match $s = (C_1, \xi, C_2)$ can be "in the path" to match several maximal configurations.

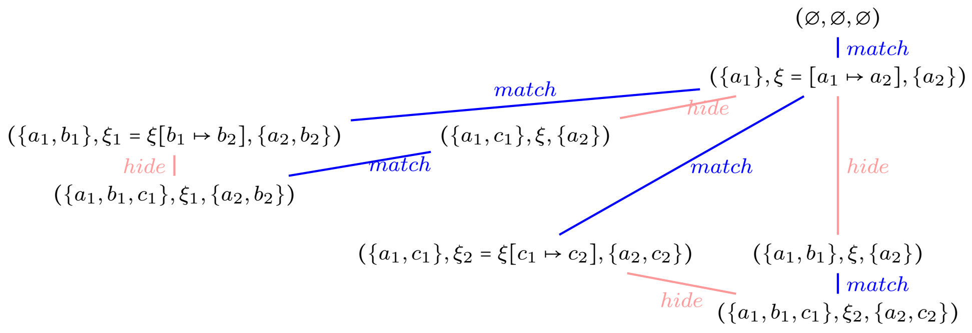
The function for the A^* algorithm is then $\kappa(s) = g(s) + h(s)$ for any partial match $s = (C_1, \xi, C_2)$. The pseudo-code for the search algorithm is presented in Algorithm 1 and uses function κ.

The presented version of the A^* algorithm is tightly coupled with the semantics of the underlying event structure, because the match and hide operations are based on the possible extensions of the configurations. In other words, the nodes expanded by the A^* algorithm from a partial match represent extensions in both configurations in the case of match, or extension in only one configuration in the case of hide.

Figure 5.9 shows two PESs and a part of their partial synchronized product, which contains the optimal matches for the maximal configurations. Observe that, in the partial synchronized product, the fact that a pair of operations can be applied independently is captured by diamonds-like shapes in the graph. E.g., in Figure 5.9, after $(\{a_1\}, \xi = [a_1 \mapsto a_2], \{a_2\})$, it is possible to match the events with label b $(\{a_1, b_1\}, \xi' = \xi[b_1 \mapsto b_2], \{a_2, b_2\})$ and hide the c-labeled event $(\{a_1, b_1, c_1\}, \xi', \{a_2, b_2\})$ in any order without affecting the final partial match.



(a) $\mathbb{P}_5$ **(b)** $\mathbb{P}_6$



$(\emptyset, \emptyset, \emptyset)$
match
$(\{a_1\}, \xi = [a_1 \mapsto a_2], \{a_2\})$
match
hide
$(\{a_1, b_1\}, \xi_1 = \xi[b_1 \mapsto b_2], \{a_2, b_2\})$
$(\{a_1, c_1\}, \xi, \{a_2\})$
hide
match
match
hide
$(\{a_1, b_1, c_1\}, \xi_1, \{a_2, b_2\})$
$(\{a_1, c_1\}, \xi_2 = \xi[c_1 \mapsto c_2], \{a_2, c_2\})$
$(\{a_1, b_1\}, \xi, \{a_2\})$
match
hide
$(\{a_1, b_1, c_1\}, \xi_2, \{a_2, c_2\})$

(c) partial synchronized product for $\mathbb{P}_5$ and $\mathbb{P}_6$

Figure 5.9: PESs and their partial synchronized product with the optimal partial matches

Algorithm 1 Computing partial matches

```
Algorithm
    Input: $\mathcal{P}_1 = \langle E_1, Conf(\mathcal{P}_1), \lambda_1\rangle$ and $\mathcal{P}_2 = \langle E_2, Conf(\mathcal{P}_2), \lambda_2\rangle$
    Output: Multiset of partial matches for the maximal configurations
    // Initialization
    foreach $C \in MaxConf(\mathcal{P}_1) \cup MaxConf(\mathcal{P}_2)$ do
        $GW(C) = \infty$
        $MATCHES[C] = \varnothing$
    end
    $s_0 = \langle\varnothing, \varnothing, \varnothing\rangle$
    OPEN $\leftarrow \{s_0\}$
    while $OPEN \neq \varnothing$ do
        Choose any $s = \langle C_1, \xi, C_2\rangle \in$ OPEN, with minimum $\kappa(s)$
            OPEN $\leftarrow$ OPEN $\setminus \{s\}$
            // Pruning
        if isCandidate($C_1, s, \mathcal{P}_1$) $\vee$ isCandidate($C_2, s, \mathcal{P}_2$) then
            // Best match
            if $C_1 \in MaxConf(\mathcal{P}_1) \wedge C_2 \in MaxConf(\mathcal{P}_2)$ then
                updateMatches($C_1, s$)
                updateMatches($C_2, s$)
            end
            foreach $C_1 \xrightarrow{e_1} C_1', C_2 \xrightarrow{e_2} C_2'$, s.t. $\lambda_1(e_1) = \lambda_2(e_2)$ do
                if $\xi[e_1 \mapsto e_2]$ is a partial match (Def. 5.13) then
                    OPEN $\leftarrow$ OPEN $\cup\{\langle C_1', \xi[e_1 \mapsto e_2], C_2'\rangle\}$    ▷ MATCH
                end
            end
            foreach $C_1 \xrightarrow{e_1} C_1'$ do
                OPEN $\leftarrow$ OPEN $\cup$ $\{\langle C_1', \xi, C_2\rangle\}$    ▷ HIDE $e_1$
            end
            foreach $C_2 \xrightarrow{e_2} C_2'$ do
                OPEN $\leftarrow$ OPEN $\cup$ $\{\langle C_1, \xi, C_2'\rangle\}$    ▷ HIDE $e_2$
            end
        end
    end
    return $MATCHES$
Procedure isCandidate($C, s, \mathcal{P}$)
    return $\exists M \in MaxConf(\mathcal{P}) : C \subset M \wedge \kappa(s) \leq GW(M)$
Procedure updateMatches($C, s$)
    if $\kappa(s) \leq GW(C)$ then
        $MATCHES[C] \leftarrow MATCHES[C] \cup \{s\}$
        $GW[C] \leftarrow \kappa(s)$
    end
```

5.2.2 Identifying differences

The partial synchronized product is a rich structure that represents the hide and match operations, which lead to some partial matches (possibly optimal or simply good, when determined with some heuristic approach).

In order to explain the behavioral differences, a possibility consists in simply verbalizing the hide operations. Note that differently from a purely syntactical approach this will captures how early a discrepancy can arise during the execution of the processes. In other words, the closer a hide operation is from the "initial" node $(\varnothing, \varnothing, \varnothing)$, the sooner the discrepancy can occur.

The partial synchronized product explicitly represents the state (partial match) where a discrepancy occurs, hereinafter called the *context.* Then, a hide operation can be expressed as an event that occurs in one model but not in the other. For instance, in Figure 5.9 there is an edge representing the hide operation $(\{a_1\}, \xi = [a_1 \mapsto a_2], \{a_2\}) \xrightarrow{hide\ c_1} (\{a_1, c_1\}, \xi, \{a_2\})$ and another representing the hide operation $(\{a_1, b_1\}, \xi' = \xi[b_1 \mapsto b_2], \{a_2, b_2\}) \xrightarrow{hide\ c_1} (\{a_1, b_1, c_1\}, \xi', \{a_2, b_2\})$. The behavioral difference represented by these two nodes is the same, namely: *"In model 1, there is a state where* c *occurs, whereas in the matching state in model 2, it cannot occur".* These two differences however differ in terms of the state where the difference is observed. In the first case, the state in question is the one reached immediately after we execute activity a, whereas in the second case, it is the state reached immediately after we execute activity b. We can therefore see that if we map each hide operation in the partial synchronized product into a difference diagnostic statement, the resulting statements can be largely redundant and difficult to interpret.

For this reason a more abstract explanation of the differences, e.g., in terms of behavioral relations that hold in one process and not in the other, can be more convenient and understandable for the user. Thus, we next present an approach for expressing the hide operations in the partial synchronized product as behavioral relations of an event structure.

In this approach, the behavioral difference between the PESs in Figure 5.9 can all be expressed using one single diagnostic statement, that is: *"In model 1,* b *and* c *are in parallel, whereas in model 2,* b *and* c *are mutually exclusive"*.

To implement this latter approach, we have to select the set of behavioral relations that best helps with the verbalization of the discrepancies captured by a given hide operation. To this end, we observe that a partial matching (C_1, ξ, C_2) can be seen as a partially filled matrix of behavioral relations, denoted as Ψ_ξ. In this alternative representation, the columns represent the matched events in ξ and the rows represent the hidden (unmatched) events. For instance, the matrix representation of the matching $(\{a_1, b_1, c_1\}, \xi, \{a_2, b_2\})$ (Figure 5.9) is displayed in Figure 5.10a.

	(a_1, a_2)	(b_1, b_2)
$(c_1, \quad)$	$(<, \quad)$	$(\Vert, \quad)$

(a) Ψ_ξ

	(a_1, a_2)	(b_1, b_2)
(c_1, c_2)	$(<, <)$	$(\Vert, \#)$

(b) Ψ_ζ

Figure 5.10: (a) Matrix representations for (a) partial match $(\{a_1, b_1, c_1\}, \xi, \{a_2, b_2\})$ and (b) extended partial match $(\{a_1, b_1, c_1\}, \zeta = \xi[c_1 \mapsto c_2], \{a_2, b_2\})$

The overall idea in order to diagnose the differences in terms of behavioral relations is the following. Given a partial match ξ, we aim at *extending* the mappings (even outside the configurations) for the unmatched events. The matching for an unmatched event shall have the same label, so that they can be seen as an instance of the same action, and their dependencies with the events in ξ shall be as similar as possible. The extension of a partial match renounces to the requirement that the match should respect the order in the pomsets, but still tries (following some heuristic) to match events which are alike. The formal definition of an extended partial match is presented below.

Definition 5.16 (extended partial match)**.** Let $\mathcal{P}_1 = \langle E_1, \mathit{Conf}(\mathcal{P}_1), \lambda_1 \rangle$ and $\mathcal{P}_2 = \langle E_2, \mathit{Conf}(\mathcal{P}_2), \lambda_2 \rangle$ be families of pomsets and let $\xi : C_1 \nrightarrow C_2$ be a partial match between configurations C_1 and C_2. An *extended partial match* for ξ is an injective partial function $\zeta : E_1 \nrightarrow E_2$ such that (i) $\xi \subseteq \zeta$, (ii) for any $e_1 \in C_1$ such that $\zeta(e_1) \neq \bot$ it holds $\lambda_2(\zeta(e_1)) = \lambda_1(e_1)$ and (iii) for any $e_1 \in E_1$ if $\zeta(e_1) \neq \bot$ then either $e_1 \in C_1$ or $\zeta(e_1) \in C_2$.

Intuitively, the extension of a partial match ξ is any label-preserving partial function extending ξ. Condition (iii) states that extensions are only allowed when they match previously unmatched events either in C_1 or C_2.

Let us introduce some measure of the "quality" of an extension. Roughly, we try to minimize the number of dependencies on which the matched events differ. The next definition uses a generic set of relations $\mathscr{R}$, which is concretely defined depending on the type of event structures used. I.e., in the case of PES $\mathscr{R} = \{\leq, \#\}$, in the case of AES $\mathscr{R} = \{\nearrow^*, <\}$ and in the case of FES $\mathscr{R} = \{\prec, \#\}$.

Definition 5.17 (cost of extensions)**.** Let $\zeta : E_1 \nrightarrow E_2$ be an extension of the partial match ξ between configurations C_1 and C_2, and let $\mathscr{R}$ be the set of relations in the event structure. The cost of ζ is defined as

$$\begin{aligned} K(\zeta) = |\{((e_1, e_2), rel, (e_1', e_2')) : \ & rel \in \mathscr{R} \ \wedge \ \zeta(e_1) = e_2 \ \wedge \ \zeta(e_1') = e_2' \ \wedge \\ & \neg(e_1 \ rel \ e_1' \iff e_2 \ rel \ e_2')\}| \end{aligned}$$

We are interested in maximal extensions ζ of a partial match (namely extensions where all pairs of events with the same labels have been matched), which minimize the cost $K(\zeta)$. If the explicit computation of a maximal extension with least cost is computationally too expensive, one can use a local search criteria, i.e., start from a partial match ξ and add a single pair of events each time (thus applying the rule in Figure 5.11, where either $e_1 \in C_1$ or $e_2 \in C_2$, minimizing the cost at each step).

Consider for example the optimal matching $(\{a_1, b_1, c_1\}, \ \xi, \{a_2, b_2\})$ (Figure 5.9). The corresponding optimal maximal extension is shown in

$$\frac{\zeta(e_1) = \perp = \zeta^{-1}(e_2) \quad \lambda_1(e_1) = \lambda_2(e_2)}{\zeta[e_1 \mapsto e_2]} \; \textit{synthetic match } e_1, e_2$$

Figure 5.11: Synthetic matching operation

Figure 5.10b, i.e., $(\{a_1, b_1, c_1\}, \zeta = \xi[c_1 \mapsto c_2], \{a_2, b_2\})$. This example is very simple because there is only one possibility to match the event c_1 in $\mathbb{P}_6$ (with event c_2).

The partial synchronized product may contain more than one optimal match for a maximal configuration (and also more than one extended partial match), each of which leads to the same number of differences. In the absence of any other intuitive criteria for distinguishing optimal matches, we select any such matching to generate a verbalization of differences. The following section describes the verbalization step.

5.2.3 Verbalizing differences

We propose to verbalize each discrepancy by means of a statement consisting of two parts: a description of the *context* where the discrepancy occurs and a description of the *difference* itself.

The context describes the state s in the partial synchronized product where a given discrepancy (hide operation) occurs. A full representation of the context consists of the set of events matched in s leading to the discrepancy. In the case of visual feedback, this can be visually represented by animating the process model in order to show to the user an execution path leading to the state in question. On the other hand, listing all the events leading to a given state is arguably less readable in textual form. Instead, when verbalizing a context in textual form, it may be more convenient to refer only to a partial description of the context, consisting only of the last event (i.e., last activity) executed before the hiding operation to be verbalized is reached. In the examples given below we opt for this latter (highly abbreviated) verbalization approach for the context. The problem of accurate abbreviation of execution paths leading to a given state (configuration) in a process model is further studied in [Lohm 14].

The difference itself is described by referring to either a behavioral relation in one model that is not present in the other, or by stating that the multiplicity of an activity in one model differs from the multiplicity of the same activity in the other model. Given that the comparison technique presented in this section is applicable to either AESs, PESs of FESs, below we provide the verbalization of the different possible behavioral relations between activities a and b:

- Causality ($<$): "a *has to occur before activity* b*"*.
- Asymmetric conflict ($\nearrow$): "a *can occur before* b *or* a *can be skipped"*.
- Flow ($\prec$): "a *can occur before activity* b*"*.
- Conflict ($\#$): "a *and* b *are mutually exclusive"*.
- Concurrency ($\|$): "a *and* b *are parallel"*.

The multiplicity of an activity is verbalized as follows:

- 0..1: *"occurs at most once"*,
- +: *"occurs at least once"*, and
- $\ast$: *"occurs 0,1 or more times"*.

Whereas, for safe and sound free-choice workflow nets, the multiplicity of an activity is verbalized as follows:

- +: *"occurs any number of times, but at least once"*, and
- $\ast$: *"occurs any number of times"*.

Based on the above verbalizations of context, behavioral relations and multiplicity, we use the following templates to verbalize a given discrepancy between two models *M1* and *M2*:

1. Case of unmatched event: *"In M1, there is a state after* < _context_ > *where* < _activity_ > *always occurs, whereas it cannot occur in the matching state in M2"*

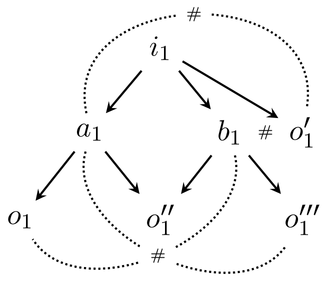
2. Case of mismatching relations. *"In M1, there is a state after* < _context_ > *where* < _verbalization of relation 1_ >*, whereas in the matching state in M2,* < _verbalization for relation 2_ >*"*

3. Case of mismatching multiplicity: *"In M1,* < _activity_ > < _verbalization of multiplicity in M1_ >*, whereas in M2, it* < _verbalization of activity multiplicity in M2_ >.

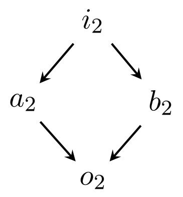
As an example, Figure 5.12 shows a pair of PES and their partial synchronized product. The differences, with the approximate context, can be expressed with the following mismatching relations:

1. $(b_1, b_2), (o'_1, o_2) = (\#, <)$: *"In M1, there is a state after* i *where* o *and* b *are mutually exclusive, whereas in the matching state in M2,* b *has to occur before activity* o*"*,
2. $(a_1, a_2), (o'_1, o_2) = (\#, <)$: *"In M1, there is a state after* i *where* o *and* a *are mutually exclusive, whereas in the matching state in M2,* a *has to occur before activity* o*"*
3. $(b_1, b_2)(o_1, o_2) = (\#, <)$: *"In M1, there is a state after* a *where* o *and* b *are mutually exclusive, whereas in the matching state in M2,* b *has to occur before activity* o*"*, and
4. $(a_1, a_2), (o'''_1, o_2) = (\#, <)$: *"In M1, there is a state after* b *where* o *and* a *are mutually exclusive, whereas in the matching state in M2,* a *has to occur before activity* o*"*.

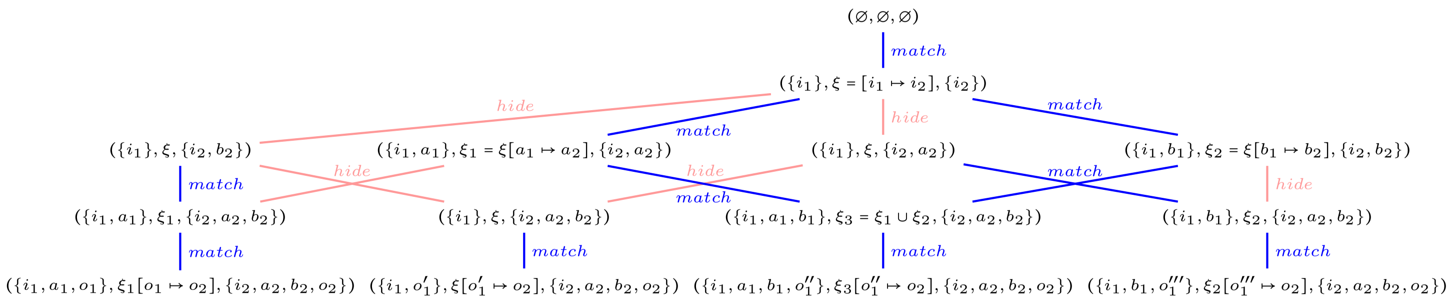
Observe that, even though there are six hiding operations in the partial synchronized product, there are only four sentences explaining the differences between both PES. It is due to the fact that $(\{i_1\}, \xi = [i_1 \mapsto i_2], \{i_2\}) \xrightarrow{hide\ b_2} (\{i_1\}, \xi, \{i_2, b_2\})$ and $(\{i_1\}, \xi, \{i_2, a_2\}) \xrightarrow{hide\ b_2} (\{i_1\}, \xi, \{i_2, a_2, b_2\})$ are representing the hiding of b with the same context ξ, similarly for *hide* a_2 in ξ.



(a) $\mathbb{P}_7$



(b) $\mathbb{P}_8$


$(\varnothing,\varnothing,\varnothing)$
match
$(\{i_1\},\xi=[i_1\mapsto i_2],\{i_2\})$
hide
match
hide
match
$(\{i_1\},\xi,\{i_2,b_2\})$
$(\{i_1,a_1\},\xi_1=\xi[a_1\mapsto a_2],\{i_2,a_2\})$
$(\{i_1\},\xi,\{i_2,a_2\})$
$(\{i_1,b_1\},\xi_2=\xi[b_1\mapsto b_2],\{i_2,b_2\})$
match
hide
hide
match
match
hide
$(\{i_1,a_1\},\xi_1,\{i_2,a_2,b_2\})$
$(\{i_1\},\xi,\{i_2,a_2,b_2\})$
$(\{i_1,a_1,b_1\},\xi_3=\xi_1\cup\xi_2,\{i_2,a_2,b_2\})$
$(\{i_1,b_1\},\xi_2,\{i_2,a_2,b_2\})$
match
match
match
match
$(\{i_1,a_1,o_1\},\xi_1[o_1\mapsto o_2],\{i_2,a_2,b_2,o_2\})$
$(\{i_1,o_1'\},\xi[o_1'\mapsto o_2],\{i_2,a_2,b_2,o_2\})$
$(\{i_1,a_1,b_1,o_1''\},\xi_3[o_1''\mapsto o_2],\{i_2,a_2,b_2,o_2\})$
$(\{i_1,b_1,o_1'''\},\xi_2[o_1'''\mapsto o_2],\{i_2,a_2,b_2,o_2\})$

(c) partial synchronized product for $\mathbb{P}_7$ and $\mathbb{P}_8$

Figure 5.12: PESs and their partial synchronized product with the optimal matches

5.3 Discussion

The use other types of event structures can lead to more compact representations and, by the same token, diagnosis. For instance, Figure 5.13 shows an AES that is equivalent to the PES in Figure 5.12a, i.e., history preserving bisimilar. In this example, the interpretation of the differences using AESs instead of PESs can be expressed only in two sentences:

1. $(b_1, b_2), (o_1, o_2) = (\nearrow, <)$: *"In M1, there is a state after* i *where* b *can occur before* o *or* b *can be skipped, whereas in the matching state in M2,* b *has to occur before activity* o*"*, and
2. $(a_1, a_2), (o_1, o_2) = (\nearrow, <)$: *"In M1, there is a state after* i *where* b *can occur before* o *or* b *can be skipped, whereas in the matching state in M2,* b *has to occur before activity* o*"*.

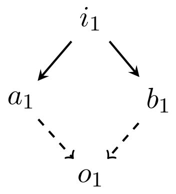
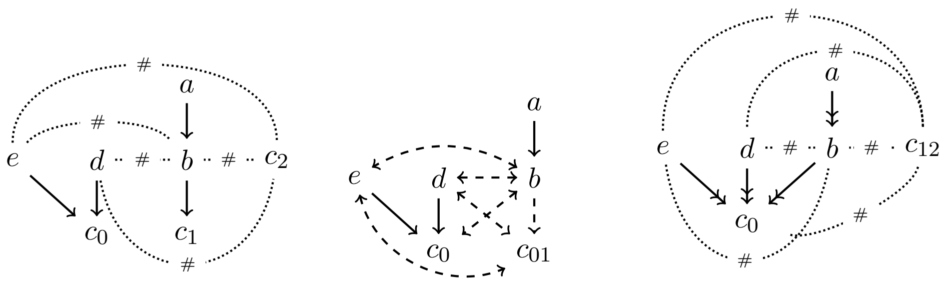


Figure 5.13: AES equivalent to $\mathbb{P}_7$ in Figure 5.12a

Many different kinds of event structures have been proposed, relying on more expressive dependency relations. In this work, we focus on two basic extensions of prime event structures, namely *asymmetric event structures* (AESs) [Bald 01], where conflict is allowed to be non-symmetric, and *flow event structures* (FESs) [Boud 89], which provide a form of disjunctive causality (the causes of an event can be chosen from a set of conflicting events).

Interestingly, it can be seen that the three event structures depicted in Figures 5.14(a)-(c) represent the same set of computations, but with different numbers of events. This happens because AESs and FESs can take advantage from their relations and semantics in order to avoid some



(a) PES **(b)** AES **(c)** FES

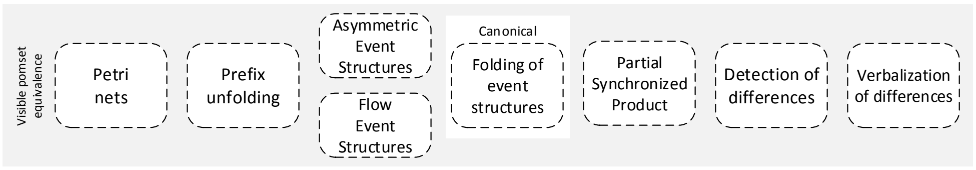
Figure 5.14: Three history preserving bisimilar event structures

duplication of events representing activity c. Also, it should be noted that PESs can be seen as special AESs, where asymmetric conflict is actually symmetric, and as special FESs, where the flow relation is transitive and potential causes do not contain conflicts.

The next chapter identifies suitable transformations which reduce the size of AESs and FESs, without altering the original behavior. The method is based on the identification of sets of events that intuitively represent occurrences of the same activity in different contexts and can be safely folded into a single event. As a reference notion of behavioral equivalence we consider history preserving bisimilarity [Rabi 88, Glab 89, Best 91], one of the classical equivalences in the true concurrent spectrum. Indeed, the three event structures in Figure 5.14 can be shown to be history-preserving bisimilar. The AES in Figure 5.14b can be seen as a reduction of the PES in Figure 5.14a that was obtained by “folding” the events c_0 and c_1 into a single event c_{01}. Similarly, the FES in Figure 5.14c can be seen as a reduction of the PES that was obtained by “folding” events c_1 and c_2 into c_{12}.

Chapter 6

Reduction of event structures


Visible pomset equivalence
Petri nets
Prefix unfolding
Asymmetric Event Structures
Flow Event Structures
Canonical
Folding of event structures
Partial Synchronized Product
Detection of differences
Verbalization of differences

Chapter 5 shows an example where smaller event structures can lead to more compact diagnostics during the comparison and verbalization of differences. Two types of event structures that can avoid some of the event duplication inherent to PESs are *Asymmetric Event Structures* (AESs) and *Flow Event Structures* (FESs). This chapter proposes behavior-preserving (w.r.t. hp-bisimulation) reduction techniques for AESs and FESs, which aim at finding sets of events that can be replaced with a single event while keeping the behavior unchanged. Section 6.1 presents an abstract notion of folding of event structures. Sections 6.2 and 6.3 presents the folding techniques for AESs and FESs, respectively. Section 6.4 defines a deterministic order on the reduction operations, such that the folded version of an event structure is always the same. Final discussions are presented in Section 6.5.

6.1 Foldings

We next introduce the notion of folding, which is intended to formalize the intuition of a behavior-preserving quotient for an event structure. In the next sections we will provide some concrete folding techniques for AESs and FESs. The following definition uses $\mathbb{E}$ to denote AES and FES indistinctively.

Definition 6.1 (folding)**.** Let $\mathbb{E}_1$ and $\mathbb{E}_2$ be event structures. A *folding morphism* is a surjective function $f : E_1 \to E_2$ such that the relation $R_f = \{(C_1, f_{|C_1}, f(C_1) \mid C_1 \in \mathit{Conf}(\mathbb{E}_1)\}$ is a hp-bisimulation. A folding is called *elementary* if there is a set $X_1 \subseteq E_1$ such that for all $e_1, e_1' \in E_1$, $e_1 \neq e_1'$ and $f(e_1) = f(e_1')$ iff $e_1, e_1' \in X_1$.

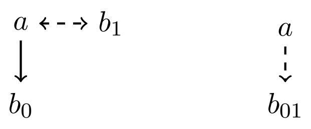
In words, a folding is a mapping that "merges" some sets of events of an event structure into single event keeping the behavior unaltered. It is elementary if it merges only a single set of events.

Sometimes, with abuse of terminology, we will refer to $\mathbb{E}_2$ as the folding of $\mathbb{E}_1$. It can be seen that under mild conditions, the target event structure is completely determined by the folding map, hence it can be seen as a sort of quotient along the map.

6.2 Reduction of AESs

The technique for behavior preserving reduction of AESs consists in iteratively identifying a set of events carrying the same label, i.e., intuitively referring to the same activity, and replacing all the events in the set with a single event. This quotient operation is shown to induce an elementary folding, i.e., it leaves the behavior unchanged with respect to hp-bisimilarity.

The prototypical example of folding in AESs, which exploits the expressiveness of asymmetric conflict, is provided in Figure 6.1. The right AES is obtained by merging the two conflicting b-labelled events b_0 and b_1 (the conflict $b_0 \# b_1$ is inherited from $a \# b_1$). Event a is in asymmetric conflict

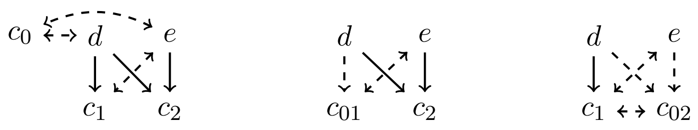


(a) $\mathbb{A}'$ **(b)** $\mathbb{A}''$

Figure 6.1: AES $\mathbb{A}'$ and a folding $\mathbb{A}''$.

with the event b_{01} resulting from the merge, thus $hist(b_{01})$ in $\mathbb{A}''$ includes $\{a, b_{01}\}$ and $\{b_{01}\}$, which corresponds exactly to the histories of b_0 and b_1, respectively, in the AES $\mathbb{A}'$. The function mapping a identically and b_0, b_1 to b_{01} can be easily shown to be a folding.

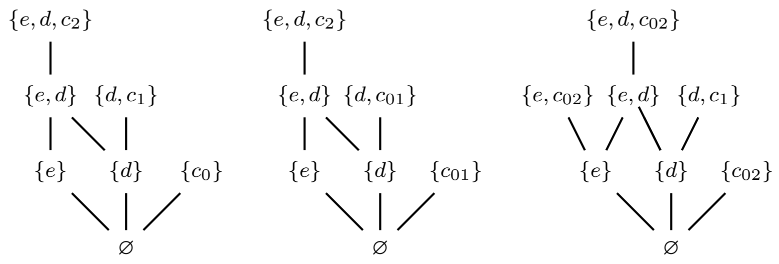
More generally, the rough idea is that a folding will merge events in conflict, with the same label and different sets of causes, into a single event having such sets of causes as possible histories. However, events to be merged have to be chosen carefully. Consider, for instance, the AESs in Figure 6.2. The AES $\mathbb{A}_1$ can be thought of as a quotient of $\mathbb{A}_0$ obtained by folding the two c-labelled events c_0 and c_1, the first in conflict with d and the second caused by d, into a single event c_{01}. The dependencies $d \# c_0$ and $d < c_1$ in $\mathbb{A}_0$ give rise to the asymmetric conflict $d \nearrow c_{01}$ in $\mathbb{A}_1$. Analogously, $\mathbb{A}_2$ is obtained from $\mathbb{A}_0$ by merging c_0 and c_2 into a single event c_{02}.



(a) $\mathbb{A}_0$ **(b)** $\mathbb{A}_1$ **(c)** $\mathbb{A}_2$

Figure 6.2: AESs such that $\mathbb{A}_0 \equiv_{hp} \mathbb{A}_1$ but $\mathbb{A}_0 \not\equiv_{hp} \mathbb{A}_2$.

Figure 6.3 shows the sets of configurations of the AESs in Figure 6.2, endowed with the extension order. Observe that the AESs $\mathbb{A}_0$ and $\mathbb{A}_1$ have isomorphic partially ordered sets of configurations. Instead, the poset



(a) $Conf(\mathbb{A}_0)$ **(b)** $Conf(\mathbb{A}_1)$ **(c)** $Conf(\mathbb{A}_2)$

Figure 6.3: Configurations of the AESs in Figure 6.2, ordered by extension

corresponding to $\mathbb{A}_2$ has an additional configuration $\{e, c_{02}\}$ that does not correspond to any configuration in $Conf(\mathbb{A}_0)$. Hence, even though $\mathbb{A}_1$ and $\mathbb{A}_2$ are obtained from $\mathbb{A}_0$ via an apparently similar procedure, the mapping into $\mathbb{A}_1$ is a folding, while the one into $\mathbb{A}_2$ is not.

Events that can be merged, intuitively, should represent occurrences of the same activity in different contexts (leading to different causal histories for the events). Hence they surely need to have the same label and be in conflict. Additionally they should relate to the remaining events, via asymmetric conflict, essentially in the same way. This is formalised by the notion of similar events.

Definition 6.2 (similar events). Let $\mathbb{A} = \langle E, \leq, \nearrow, \lambda \rangle$ be an AES. We say that $X \subseteq E$ is a set of *similar events* if for all $x, x' \in X$, $e \in E \setminus X$:

1. $\lambda(x) = \lambda(x')$ and $x \# x'$
2. if $x \nearrow e$ then $x' \nearrow e \ \vee \ e \nearrow x$;
3. $e \nearrow_\delta x \quad \Rightarrow \quad e \nearrow x'$.

Condition (1) requires that, as mentioned above, the events in X have the same label and are conflict. By condition (2), given two events $x, x' \in X$, if for an event $e \in E \setminus X$ we have $x \nearrow e$ then necessarily be also $x' \nearrow e$, unless $e \nearrow x$, and thus x and e are in conflict. This last clause captures the situation in which e is in the history of x' but not in that of x, and thus

x and e are in conflict. Finally, condition (3) requires that any direct $\nearrow$-predecessors of an event in X remains a $\nearrow$-predecessor for all other events in X.

We next define the AES which results from the merge of a set of similar events. For a relation r on events, we will denote by $r^{\forall}$ and $r^{\exists}$ the relations between events and sets of events defined in the expected way. For instance, given an event e and a set of events X, by $e\ r^{\forall}\ X$ we mean that $e\ r\ x$ holds for all $x \in X$, and by $X\ r^{\exists}\ e$ we mean that $x\ r\ e$ holds for some $x \in X$.

Definition 6.3 (quotient of an AES)**.** Let $\mathbb{A} = \langle E, \leq, \nearrow, \lambda\rangle$ be an AES and X be a set of similar events. The *quotient* of $\mathbb{A}$ with respect to X, denoted $\mathbb{A}_{/X}$, is the AES $\mathbb{A}_{/X} = \langle E_{/X},\ <_{/X}, \nearrow_{/X}, \lambda_{/X}\rangle$ defined as follows

$$\begin{array}{lcl} E_{/X} & = & (E \setminus X) \cup \{e_X\} \\ \leq_{/X} & = & \leq_{|(E\setminus X)} \cup \{(e, e_X) \mid e <^{\forall} X\} \cup \{(e_X, e) \mid X <^{\exists} e\} \\ \nearrow_X & = & \nearrow_{|(E\setminus X)} \cup \{(e, e_X) \mid e \nearrow^{\forall} X\} \cup \{(e_X, e) \mid X \nearrow^{\forall} e\} \\ \lambda_{/X} & = & \lambda[e_X \mapsto \lambda(x)] \text{ for an event } x \in X. \end{array}$$

The *quotient map* $f_X : \mathbb{A} \to \mathbb{A}_{/X}$ is defined by $f_X(x) = e_X$ for $x \in X$ and $f_X(e) = e$ for $e \in E \setminus X$.

In words, the quotient of $\mathbb{A}$ is obtained by replacing the set X of events with a single event e_X, with the same label as those in X. The causes of e_X are the common causes of the events in X. Any event originally caused by at least an event in X is now caused by e_X. This can be understood by recalling that the quotient map, in order to be a folding, must be in particular a simulation. Hence, on the one hand, in any computation, a common cause of all the events in X will surely occur before e_X and, on the other hand, e_X will occur before any causal consequence of an event in X. The asymmetric conflicts for e_X are exactly the common asymmetric conflicts of the events in X. This is explained by the fact that, in order to be a folding, the quotient map must preserve and reflect the local order of configurations which is given by (the transitive closure of) asymmetric conflict.

We can prove that, according to the intuition above, the quotient map is a simulation, in the sense that it preserves configurations and the extension relation on configurations. We start with a technical lemma, identifying some relevant properties of the quotient map. This will be used also to prove that $\mathbb{A}_{/X}$ is a well-defined AESs, a fact which has not be showed formally yet. It could be proved that the quotient map is an AES morphism in the sense of [Bald 01], but this has not a relevant use in this context.

Lemma 6.4 (properties of the quotient map)**.** *Let $\mathbb{A} = \langle E, \leq, \nearrow, \lambda \rangle$ be an AES and let $X \subseteq E$ be a set of similar events. Then for all $e \in E$, $z \in E_{/X}$*

1. *if $z <_{/X} f(e)$ then there exists $e' \in E$ such that $e' < e$ and $f(e') = z$;*
2. *if $f(e) \nearrow_{/X} f(e')$ then $e \nearrow e'$;*
3. *if $e \nearrow_{\delta} e'$ then $f(e) \nearrow_{/X} f(e')$ or $e \# e'$;*
4. *if $e < e'$ then $f_X(e) \nearrow_{/X} f_X(e')$.*

Proof. 1. Let $z \in E_{/X}$ and $e \in E$ be such that $z <_{/X} f(e)$. We distinguish various cases:

- If $z = e_X$ then, by Definition 6.3, there exists $x \in X$ such that $x < e$. Since $f(x) = e_X$ and $f(e) = e$, this is the desired conclusion.
- If $e \in X$ (and thus $f(e) = e_X$) then by Definition 6.3, $z = e' <^{\forall} X$, i.e., $e' < x$ for all $x \in X$. Therefore, in particular, $e' < e$, as desired.
- If none of the above apply, then $z = e' \in E$ and $f(e) = e$, hence the result trivially holds.

2. Let $e, e' \in E$ and assume $f(e) \nearrow_{/X} f(e')$. If $e \in X$ and thus $f(e) = e_X$ then, by Definition 6.3, $X \nearrow^{\forall} e'$ and thus, again, $e \nearrow e'$. If instead, $e' \in X$ and thus $f(e') = e_X$ then, by Definition 6.3, $e \nearrow^{\forall} X$. Thus in particular, $e \nearrow e'$ as desired. Finally, if $e, e' \notin X$ then f_X is the identity on e, e', and thus the result trivially holds.

3. Let $e, e' \in E$ and assume $e \nearrow_{\delta} e'$. We distinguish three cases:

- If $e \in X$ then, by Definition 6.2(2), either $e' \nearrow e$ and thus $e \# e'$ and we are done, or for all $x \in X$ we have $x \nearrow e'$, namely $X \nearrow^{\forall} e'$. In the last case, according to Definition 6.3, we thus have $f(e) = e_X \nearrow_{/X} e' = f(e')$, as desired.

- If $e' \in X$ then, by Definition 6.2(3), for all $x \in X$ we have $e \nearrow x$, namely $e \nearrow^{\forall} X$. Thus, by Definition 6.3, $f(e) = e \nearrow_{/X} e_X = f(e')$, as desired.

- Otherwise, neither e nor e' are in X and thus the thesis trivially follows.

4. Let $e, e' \in E$ and assume that $e < e'$. If $e, e' \notin X$ then the relations between the two events are left unchanged. Since $e < e'$ and thus $e \nearrow e'$ we have that $f_X(e) \nearrow_{/X} f_X(e')$. If $e \in X$ then by Definition 6.2(2) either $x' \nearrow e'$ for all $x' \in X$ or $e' \nearrow e$. The second possibility would lead to a contradiction, since we would have $e \# e'$ and $e < e'$. Hence the first possibility must hold and thus $X \nearrow^{\forall} e'$, thus $f_X(e) = e_X \nearrow_{/X} f_X(e')$. Finally, if $e' \in X$, from $e < e'$ we know that $e < e'' <_{\delta} e'$, for some e''. By Definition 6.2(3), since $e'' <_{\delta} e'$ and thus $e'' \nearrow_{\delta} e'$ we have that $e'' \nearrow x$, for all $x \in X$. Recalling that $e < e''$, we have $e \nearrow x$, for all $x \in X$, namely $e \nearrow^{\forall} X$. Therefore $f_X(e) \nearrow_X e_X = f_X(e')$, as desired. □

$$\begin{tikzcd}
a \arrow[d] & \\
b \arrow[d] \arrow[r, dashed, leftrightarrow] & c_1 \\
c_0 &
\end{tikzcd}$$

(a) $\mathbb{A}_3$

$$\begin{tikzcd}
a \arrow[d] \\
b \arrow[d, dashed] \\
c_{01}
\end{tikzcd}$$

(b) $\mathbb{A}_3'$

Figure 6.4: AES and its quotient

Note that the converse of (2) above, i.e., if $e \nearrow e'$ then $f_X(e) \nearrow_{/X} f_X(e')$, does not hold. For instance, consider the AES in Figure 6.4. If we merge c_0 and c_1, we get that $a \nearrow c_0$ but it is not true that $f_X(a) \nearrow_{/X} f_X(c_0) = c_{01}$. Moreover, note from (4) and the definition of $\leq$ in the quotient (Definition 6.3), it follows that the causes of some event in X which are not common to all events are turn into (proper) asymmetric conflicts.

Lemma 6.5 ($\mathbb{A}_{/X}$ is well-defined)**.** *Let $\mathbb{A} = \langle E, \leq, \nearrow, \lambda \rangle$ be an AES and let $X \subseteq E$ a set of similar events. Then $\mathbb{A}_{/X} = \langle E_{/X}, \leq_{/X}, \nearrow_{/X}, \lambda_{/X} \rangle$ is an AES.*

Proof. Let $\mathbb{A}_{/X} = \langle E_{/X}, \leq_{/X}, \nearrow_{/X}, \lambda_{/X} \rangle$ be defined as in Definition 6.3 and Let $f_X : \mathbb{A} \to \mathbb{A}_{/X}$ be the quotient map. We first note that $\leq$ is a partial order. Antisymmetry is obvious. Transitivity of $\leq_{/X}$ follows immediately by transitivity of $\leq$ in $\mathbb{A}$. Moreover, for any event $z \in E_{/X}$, we have that $\lfloor z \rfloor$ is finite. In fact, let e be any f_X-counterimage of z, i.e., $e \in E$ such that $f_X(e) = z$. For any $z' \in E_{/X}$, if $z' <_{/X} z$, by Lemma 6.4(1), there exists $e' < e$ such that $f_X(e') = z'$. This means that $\lfloor z \rfloor \subseteq f_X(\lfloor e \rfloor)$. Since $\lfloor e \rfloor$ is finite, we deduce that also $\lfloor z \rfloor$ is finite.

Concerning asymmetric conflict $\nearrow_{/X}$, conditions (1)-(4) in Definition 3.31 are easily inherited from those of $\nearrow$ in $\mathbb{A}$. More explicitly, let $z, z', z'' \in E_{/X}$. Then we have

1. If $z <_{/X} z'$ then $z \nearrow_{/X} z'$.

We distinguish various cases:

- if $z = e_X$ and $z' = f_X(e')$, for an event $e' \in E \setminus X$ then $X <^{\exists} e'$, namely, there exists $x \in X$ such that $x < e'$. This implies that $x \nearrow e'$ and thus, by the notion of similar events (Definition 6.2) either $x' \nearrow e'$ for all $x' \in X$ or $e' \nearrow x$. The latter possibility would lead to $x \# e'$, contradicting the fact that $x < e'$. Hence it must be $x' \nearrow e'$ for all $x' \in X$, namely $X <^{\forall} e'$, and thus $e_X \nearrow_{/X} f_X(e') = z'$.

- if $z = f_X(e)$, for an event $e \in E \setminus X$, and $z' = e_X$ then $e <^{\forall} X$. This implies that $e \nearrow^{\forall} X$ and thus $e \nearrow_{/X} e_X$.

- if both $e, e' \in E \setminus X$, the desired consequence is trivial since the relations between e and e' are not modified by the quotient operation.

2. if $z \nearrow_{/X} z' <_{/X} z''$ then $z \nearrow_{/X} z''$.
We distinguish various cases.

- If $z = e_X$ and thus $z' = f_X(e')$, $z'' = f_X(e'')$, for events $e', e'' \in E \setminus X$, then by Definition 6.3, we have $X \nearrow^{\forall} e'$ in $\mathbb{A}$, and thus $x \nearrow e' < e''$ for all $x \in X$. Therefore, $x \nearrow e''$ for all $x \in X$, namely $X \nearrow^{\forall} e''$ and thus $z = e_X \nearrow z'' = f_X(e'')$.

- If $z'' = e_X$ and thus $z = f_X(e)$, $z' = f_X(e')$, for events $e, e' \in E \setminus X$, then by Definition 6.3, we have $e <^{\forall} X$. Thus for all $x \in X$ it holds $e \nearrow e' < x$, hence $e \nearrow x$. This means that $e \nearrow^{\forall} X$ and thus $z = f_X(e) \nearrow_{/X} z'' = f_X(e_X)$, as desired.

- If $z' = e_X$ and thus $z = f_X(e)$, $z'' = f_X(e'')$, for events $e, e' \in E \setminus X$, then by Definition 6.3 there exists $e' \in X$ such that $e' < e''$. Moreover, $e \nearrow e'$ and thus $e \nearrow e''$ in $\mathbb{A}$. Since e, e'' are left unchanged by the quotient, $z = f_X(e) \nearrow f_X(e'') = z''$ in $\mathbb{A}_{/X}$.

- If none of $z, z', z'' \in X$ then the thesis trivially holds since the relations between such events are not modified by the quotient operation.

3. $\nearrow_{\lfloor x \rfloor_{\mathbb{A}_{/X}}}$ is acyclic for all $x \in E_{/X}$
Let $z \in E_{/X}$ be an event and suppose that $\lfloor z \rfloor$ contains a cycle $z_1 \nearrow_{/X} z_2 \nearrow_{/X} \dots \nearrow_{/X} z_1$. By surjectivity of f_X we can find $e \in E$ such that $z = f_X(e)$. By Lemma 6.4(1), there are events $e_1, \dots, e_n \in \lfloor e \rfloor$ such that $f_X(e_i) = z_i$ for any $i \in \{1, \dots, n\}$. By point (2) of the same lemma, $e_1 \nearrow e_2 \nearrow \dots \nearrow e_1$. This contradicts the property of $\nearrow_{\lfloor e \rfloor} \in \mathbb{A}$ being acyclic for any event $e \in \mathbb{A}$.

4. if $\nearrow_{/X \mid \lfloor z \rfloor \cup \lfloor z' \rfloor}$ is cyclic then $z \nearrow_{/X} z'$.
Let $e, e' \in E$ such that $f_X(e) = z$ and $f_X(e') = z'$. As observed in the proof of point (1), we have that $\lfloor z \rfloor = \lfloor f_X(e) \rfloor \subseteq f_X(\lfloor e \rfloor)$ and $\lfloor z' \rfloor = \lfloor f_X(e') \rfloor \subseteq f_X(\lfloor e' \rfloor)$. Therefore if $\nearrow_{/X}$ is cyclic over $\lfloor z \rfloor \cup \lfloor z' \rfloor$, it is cyclic also over $f_X(\lfloor e \rfloor) \cup f_X(\lfloor e' \rfloor) = f_X(\lfloor e \rfloor \cup \lfloor e' \rfloor)$. Since, by Lemma 6.4(2), f_X reflects asymmetric conflict, this implies that $\nearrow$ is cyclic on $\lfloor e \rfloor \cup \lfloor e' \rfloor$. Therefore $e \nearrow e'$. Since this holds for any e, e' such that $f_X(e) = z$ and $f_X(e') = z'$, a

case distinction similar to that in the previous points, allows us to conclude $z \nearrow z'$. $\square$

We can now show that quotient map preserves configurations and the extension order.

Lemma 6.6 (quotient preserves configurations)**.** *Let $\mathbb{A} = \langle E, \leq, \nearrow, \lambda \rangle$ be an AES, $X \subseteq E$ a set of similar events and let $f_X : \mathbb{A} \to \mathbb{A}_{/X}$ be the quotient map. Then for any configuration $C \in \mathit{Conf}(\mathbb{A})$ it holds that $f_X(C) \in \mathit{Conf}(\mathbb{A}_{/X})$ and $f_{X|C} : (C, \nearrow_C^*) \to (f_X(C), \nearrow_{f_X(C)}^*)$ is an isomorphism of configurations.*

Proof. Let $C \in \mathit{Conf}(\mathbb{A})$ be a configuration. We first observe that $f_X(C)$ is a configuration in $\mathit{Conf}(\mathbb{A}_{/X})$. For proving causal closedness, take $e \in C$ and consider the event $f_X(e) \in f_X(C)$. If $z <_{/X} f_X(e)$ by Lemma 6.4(1) there exists $e' \in E$ such that $e' < e$ and $f_X(e') = z$. Since C is a configuration, necessarily $e' \in C$ and thus $z = f_X(e') \in f_X(C)$.

Moreover, $\nearrow_{/X}$ is acyclic on $f_X(C)$. In fact, if there were a cycle in $f_X(C)$ it would be of the kind $f_X(e_1) \nearrow_{/X} f_X(e_2) \nearrow_{/X} \dots \nearrow_{/X} f_X(e_n) \nearrow_{/X} f_X(e_1)$, for $e_1 \dots, e_n \in C$. Then by Lemma 6.4(2), we would have $e_1 \nearrow e_2 \nearrow \dots \nearrow e_n \nearrow e_1$, contradicting the fact that C is a configuration.

In order to prove that $f_{X|C} : (C, \nearrow_C^*) \to (f_X(C), \nearrow_{f_X(C)}^*)$ is an isomorphism of configurations, it suffices to observe that for all $e, e' \in C$ we have that

1. if $f_X(e) \nearrow_\delta f_X(e')$ then $e \nearrow e'$;

2. if $e \nearrow_\delta e'$ then $f_X(e) \nearrow f_X(e')$.

Point (1) is a special case of Lemma 6.4(2). For point (2), let $e \nearrow_\delta e'$. Then by Lemma 6.4(3), either $f_X(e) \nearrow f_X(e')$ or $e \# e'$. Since the latter cannot hold, because $e, e' \in C$ which is a configuration, necessarily $f_X(e) \nearrow f_X(e')$, as desired. $\square$

As an immediate consequence of the above result, we can prove that the extension order is preserved and reflected by the quotient map.

$$\begin{tikzcd} a_0 \arrow[r, dashed, leftrightarrow] \arrow[d, dashed] & a_1 \\ b & \end{tikzcd}$$

(a) $\mathbb{A}_4$

$$\begin{tikzcd} b \arrow[d] & \\ a_0 \arrow[r, dashed, leftrightarrow] & a_1 \arrow[ul, dashed] \end{tikzcd}$$

(b) $\mathbb{A}'_4$

$$\begin{tikzcd} a_{01} \arrow[d, dashed] \\ b \end{tikzcd}$$

(c) $\mathbb{A}_{4/X}$

$$\begin{tikzcd} a_{01} & b \end{tikzcd}$$

(d) $\mathbb{A}'_{4/X}$

Figure 6.5: Quotients with respect to a set $X = \{a_0, a_1\}$ of non-similar events

Corollary 6.7. *Let $\mathbb{A} = \langle E, \leq, \nearrow, \lambda \rangle$ be an AES, $X \subseteq E$ a set of similar events and let $f_X : \mathbb{A} \to \mathbb{A}_{/X}$ be the quotient map. Then for all configuration $C, C' \in \mathit{Conf}(\mathbb{A})$ it holds that $C \sqsubseteq C'$ iff $f_X(C) \sqsubseteq f_X(C')$.*

Observe that conditions (2) and (3) in Definition 6.2 are necessary for the simulation result. For instance consider the AESs in Figures 6.5a and 6.5b, and their quotients $\mathbb{A}_{4/X}$ and $\mathbb{A}'_{4/X}$ with respect to the set $X = \{a_0, a_1\}$, in Figures 6.5c and 6.5d. In both cases, the quotients do not simulate the original AES.

More in detail, for the AES $\mathbb{A}_4$, we have $a_0 \nearrow b$ while neither $a_1 \nearrow b$ nor $b \nearrow a_0$, thus violating condition (2). Indeed $\mathbb{A}_4$ has the configuration $\{a_1, b\}$ with a_1 and b concurrent, which is not in the quotient. In the AES $\mathbb{A}'_4$ of Figure 6.5b, $b \nearrow_\delta a_0$ while it is not the case that $b \nearrow a_1$, thus violating condition (3). In this case $\mathbb{A}'_4$ has the configuration $\{b, a_0\}$ with $b < a_0$, which is not in the quotient.

However, quotienting an AES on a set of similar events X still can alter the behavior. Consider for instance the AESs $\mathbb{A}_0$ and $\mathbb{A}_2$ in Figure 6.2. We have that $\mathbb{A}_2 = \mathbb{A}_{0/\{c_0,c_2\}}$ and $\{c_0, c_2\}$ set of similar events. We already noted that $\mathbb{A}_0$ and $\mathbb{A}_2$ are not hp-bisimilar since $\mathbb{A}_2$ admits a configuration

$\{e, c_{02}\}$, which has no counterpart in $\mathbb{A}_0$: it represents a new history for a c-labelled event. The problem resides in the fact that the causes of some event $x \in X$, which are not causes for all events in X will become asymmetric conflicts in the quotient, hence they can either appear or not in the histories of e_X. The same applies to $\nearrow$-predecessors of such causes. The (consistent) combinations of these events will lead to different possible histories for the merged event e_X. Such histories must be already histories of some event in X in the original AES, otherwise they will represent newly generated behaviors.

In order to formalise this fact given an AES $\mathbb{A}$ and a set X of similar events $\mathbb{A}$ we introduce the set of possible events for X which intuitively are those events which, in the quotient, can either occur or be omitted in the histories of e_X.

Definition 6.8 (possible events)**.** Let $\mathbb{A} = \langle E, \leq, \nearrow, \lambda \rangle$ be an AES and let $X \subseteq E$ a set of similar events. The set of *possible events* for X is

$$p(X) = \{e \in E \mid \neg(X \nearrow^{\forall} e) \land \neg(e <^{\forall} X) \ \land \ e \nearrow^{\exists} X\}.$$

According the way in which $\nearrow_{/X}$ and $<_{/X}$ are introduced in Definition 6.3 the requirement $\neg(X \nearrow^{\forall} e)$ implies $\neg(e_X \nearrow_{/X} e)$ (and thus e_X and e are not in conflict) and the requirement $\neg(e <^{\forall} X)$ implies $\neg(e <_{/X} e_X)$. Finally, concerning the requirement $e \nearrow^{\exists} X$, namely $e \nearrow x$ for some $x \in X$, there are two possibilities. If $e \nearrow_{\delta} x$ then by Definition 6.2(3), $e \nearrow^{\forall} X$ and thus $e \nearrow_{/X} e_X$ in the quotient. Otherwise, $e \nearrow e' < x$ for some event e', and thus $e \nearrow_{/X} e' \nearrow_{/X} e_X$ in the quotient (since as observed above, causalities either remains unchanged or become asymmetric conflicts). In both cases, according to the informal explanation above, they can be either included or not in the history of e_X.

Marginally, we observe that the set $p(X)$ can include events that are not in the history of any event in X. This happens for the AES in Figure 6.6, taking $X = \{c_0, c_1\}$.

$$\begin{tikzcd} a \arrow[r, dashed] \arrow[d, dashed, leftrightarrow] & b \arrow[d, dashed, leftrightarrow] \arrow[dl] \\ c_0 \arrow[r, dashed, leftrightarrow] & c_1 \end{tikzcd}$$

Figure 6.6: The set $p(\{c_0, c_1\}) = \{a, b\}$, includes a which is neither in the history of c_0 nor of c_1

As mentioned above, in order not to modify the overall behavior, all consistent subsets of $p(X)$ should match some possible history of an event in X in the original AES. For instance, in Figure 6.2, in $\mathbb{A}_0$ we have that $p(\{c_0, c_1\}) = \{d\}$ while $p(\{c_1, c_2\}) = \{d, e\}$. While in the first case for any (consistent) subset of $p(\{c_0, c_1\})$ (namely $\varnothing$ and $\{d\}$) there are c-labelled events (namely c_0 and c_1) having these subsets as histories; in the second case the possible consistent subsets of $p(\{c_1, c_2\}) = \{d, e\}$ include $\{e\}$ which is not the history of any c-labelled event. Hence the first quotient $\mathbb{A}_1 = \mathbb{A}_{0/\{c_0,c_1\}}$ preserves the behavior, while the second $\mathbb{A}_2 = \mathbb{A}_{0/\{c_0,c_2\}}$ does not.

The above considerations lead to the notion of combinable set of events.

Definition 6.9 (combinable set of events)**.** Let $\mathbb{A} = \langle E, \leq, \nearrow, \lambda \rangle$ be an AES. A set of events $X \subseteq E$ of similar events is *combinable* if for all $Y \subseteq p(X)$, consistent and causally closed (namely if $e \in Y$ and $e' \in p(X)$, $e' < e$ then $e' \in Y$) there exists $e \in X$ and $H \in \mathit{hist}(e)$ such that $H \cap p(X) = Y$.

We finally now show that the quotient with respect to a combinable set of events is a folding, i.e., the corresponding quotient map can be seen as a hp-bisimilarity between $\mathbb{A}$ and $\mathbb{A}_{/X}$.

Theorem 6.10 (quotient map is a folding)**.** *Let $\mathbb{A} = \langle E, \leq, \nearrow, \lambda \rangle$ be an AES and let X be a combinable set of events. Then the quotient map $f_X : \mathbb{A} \to \mathbb{A}_{/X}$ is a folding.*

Proof. Let $\mathbb{A}$ be an AES, let X be a combinable set of events and let $f_X : \mathbb{A} \to \mathbb{A}_{/X}$ be the quotient map, where $\mathbb{A}_{/X} = \langle E_{/X}, \leq_{/X}, \nearrow_{/X}, \lambda_{/X} \rangle$

We prove that

$$R = \{(C_1, f_{|C_1}, f_X(C_1)) \mid C_1 \in \mathit{Conf}(\mathbb{A})\}$$

is a hp-bisimulation.

First of all notice that for any $C_1 \in \mathit{Conf}(\mathbb{A})$, if we let $C_2 = f_X(C_1)$, then by Lemma 6.4, $f_{|C_1} : (C_1, \nearrow^*) \to (C_2, \nearrow^*)$, is an isomorphism of pomsets.

In order to conclude, we next prove that

1. if there is $e \in E$ such that $C_1 \sqsubseteq C_1 \cup \{e\} \in \mathit{Conf}(\mathbb{A})$ then $C_2 \sqsubseteq C_2 \cup \{f_X(e)\} \in \mathit{Conf}(\mathbb{A}_{/X})$;
2. if there is $z \in E_{/X}$ such that $C_2 \sqsubseteq C_2 \cup \{z\} \in \mathit{Conf}(\mathbb{A}_{/X})$ then there is $e \in E$ such that $f_X(e) = z$ and $C_1 \sqsubseteq C_1 \cup \{e\} \in \mathit{Conf}(\mathbb{A}_{/X})$.

which corresponds to conditions (a) and (b) in Definition 3.42.

1. Note that $C_2 \cup \{f_X(e)\} = f_X(C_1 \cup \{e\})$ is a configuration by Lemma 6.6. Moreover $C_2 \sqsubseteq C_2 \cup \{f_X(e)\}$, namely there is no $e' \in C_1$ such that $f_X(e) \nearrow_{/X} f_X(e')$, otherwise by Lemma 6.4(2) we would have $e \nearrow e'$, contradicting $C_1 \sqsubseteq C_1 \cup \{e\}$.

2. Assume that $C_2 \sqsubseteq C_2 \cup \{z\} \in \mathit{Conf}(\mathbb{A}_{/X})$ for some $z \in E_{/X}$. We distinguish two cases.

2.a) $z \in E \setminus X$

Take the (unique) f_X-counterimage of e of z, namely $f_X(e) = z$. A key observation is that

$$\text{there is no } e' \in C_1 \text{ such that } e \nearrow e'. \qquad (\dagger)$$

In fact, we can show that given $e' \in C_1$ such that $e \nearrow e'$ then there exists $e'' \in C_1$ such that $z = f_X(e) \nearrow_{/X} f_X(e'')$, contradicting the fact that $C_2 \sqsubseteq C_2 \cup \{z\}$. In order to prove this, we distinguish two cases.

- First assume that $e \nearrow_\delta e'$. If $e' \notin X$ then clearly $f_X(e) \nearrow_{/X} f_X(e')$. If $e' \in X$ then by Definition 6.2(3) $e \nearrow x$ for all $x \in X$, namely $e \nearrow^\forall X$. Thus also in this case, by Definition 6.3, $f_X(e) = e \nearrow e_X = f_X(e')$. Hence the desired result holds taking $e'' = e'$.
- If instead the asymmetric conflict is not direct, then there exists e''' such that $e \nearrow_\delta e''' < e'$. Since $e' \in C_1$ by causal closure also $e''' \in C$, and thus the same argument of the previous case allows to conclude.

Now we can easily prove that $C_1 \cup \{e\} \in Conf(\mathbb{A})$. For thus, we need to show that $\lfloor e \rfloor \subseteq C_1$ Take any $e' < e$. Since $e \notin X$, by Definition 6.3, we have $f_X(e') <_{/X} f_X(e)$ and thus $f_X(e') \in f_X(C_1)$. Take $e'' \in C_1$ such that $f_X(e'') = f_X(e')$. We observe that it must necessarily be $e' = e''$. In fact, if $e' \neq e''$ it should be $e', e'' \in X$ and thus $e' \# e''$. By inheritance of conflict, this would lead to $e \# e''$ and hence $e \nearrow e''$ violating (†) above. Hence it must be $e' = e'' \in C_1$, as desired. The absence of cycles of asymmetric conflict in $C_1 \cup \{e\}$ follows immediately by the same property in C_1 and property (†) above.

Also the fact that $C_1 \sqsubseteq C_1 \cup \{e\}$ is an immediate consequence of (†) above.

2.b) $z = e_X$

Consider the set

$$Y = C_1 \cap p(X)$$

Clearly $Y \subseteq p(X)$. Moreover, it is consistent and causally closed. In fact, Y is consistent since it is a subset of C_1. It is also causally closed. In fact, if $e \in Y$ and $e' \in p(X)$, $e' < e$, since $e \in Y \subseteq C_1$ and configurations are causally closed, we deduce $e' \in C_1$ and thus $e' \in Y$.

Hence, by Definition 6.9, there exists $x \in X$ and $H \in hist(x)$ such that $H \cap p(X) = Y$.

As in the previous case we observe that

there is no $e \in C_1$ such that $x \nearrow e$. (†)

In fact, given $e \in C_1$ such that $x \nearrow e$ then, according to Definition 6.2(2), we have that either $x' \nearrow e$ for all $x' \in X$ or there exists $x' \in X$ such that $\neg(x' \nearrow e)$ and $x \# e$. In the first case, we would have $X \nearrow^{\forall} e$ and thus $z = e_X \nearrow_{/X} f_X(e) \in C_2$, contradicting the fact that $C_2 \sqsubseteq C_2 \cup \{z\}$. In the second case, from $x \# e$ we have $e \nearrow x$ and, additionally, there is $x' \in X$ such that $\neg(x' \nearrow e)$. Hence $e \nearrow^{\exists} X$ and $\neg(X \nearrow^{\forall} e)$. Moreover it cannot be $e < x$, since $e \# x$, thus $\neg(e <^{\forall} X)$. This means that $e \in p(X)$. Recalling $e \in C_1$, we deduce that $e \in Y$. Since by construction $Y \subseteq H$, in turn, we get $e \in H$ which leads to a contradiction since H is an history of x, and thus it cannot include events in conflict with x.

Now observe that $\lfloor x \rfloor \subseteq C_1$. In fact for any $e < x$ either $f_X(e) < f_X(x) = e_X$ or, by Lemma 6.4(4), $f_X(e) \nearrow f_X(x) = e_X$. In the first case, since $f_X(e) < e_X$ necessarily $f_X(e) \in C_2$ and thus, since f_X is the identity on e, we deduce $e \in C_1$. In the second case, by Definition 6.3, it must be $\neg(e <^\forall X)$. Additionally, since $e < x$ he have that $e \nearrow^\exists X$ and $\neg(X \nearrow^\forall e)$ (in particular, $\neg(x \nearrow e)$). Hence $e \in p(X)$ and, since $e < x$, necessarily $e \in H$. Thus $e \in Y = H \cap p(X)$ and therefore $e \in C_1$.

By above and (†) $C_1 \cup \{x\}$ is a configuration and $C_1 \sqsubseteq C_1 \cup \{x\}$. By Lemma 6.6, since $f(C_1 \cup \{x\}) = C_2 \cup \{e_X\}$, they are isomorphic. □

By iteratively applying the quotient to a given finite AES we can thus obtain an AES which is hp-bisimilar to the original one and not further reducible. Unfortunately, this does not provide a canonical minimal representative of the behavior. For instance, consider the AES in Figure 6.2(a). There exist two possible quotiented AESs, presented side-by-side in Figure 6.7, which are cannot be further reduced using the quotient operation.

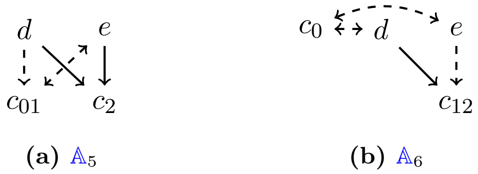


Figure 6.7: Foldings for the AES in Figure 6.2

Observe that this is not due to a limitation of our quotient technique, but rather it is intrinsic in the nature of AESs and their foldings. In fact, one can see that for these two AESs there are no non-trivial foldings (i.e., the only foldings are isomorphisms). This fact can be shown just by inspecting all the possible label preserving surjective mappings. In this regard, we address the problem of the non-canonical minimal representation in Section 6.4, where we present a way to define a deterministic order on the folding operations. Still, the question remains as to whether our quotient technique is in some sense complete, i.e., if it generates all the possible foldings. We will come back to this question in the discussions at the end of this chapter.

6.3 Reduction of FESs

The technique for behavior preserving reduction of FESs, as in the case of AESs, consists in iteratively identifying a set of conflicting events with the same label that, when replaced by a single event, induces an elementary folding. As observed in the introduction, the way in which FESs generalizes PESs is somehow orthogonal to that of AESs: the latter allow a non-symmetric form of conflict, while the former introduce a form of disjunctive causality. As a consequence, at a technical level the conditions defining the sets of events that can be merged are quite different.

$$\begin{tikzcd}
a \arrow[r, dotted, no head] \arrow[d] \arrow[drr, dotted, no head, "\#"] & \# \arrow[r, dotted, no head] & b \arrow[d] \arrow[dll, dotted, no head] \\
c_0 \arrow[r, dotted, no head] \arrow[d] & \# \arrow[r, dotted, no head] & c_1 \arrow[d] \\
d & & e
\end{tikzcd}$$

(a) $\mathbb{F}$

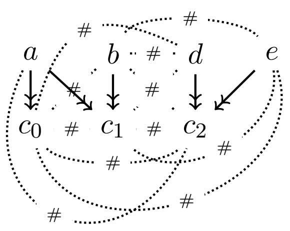
$$\begin{tikzcd}
a \arrow[r, dotted, no head] \arrow[dr] & \# \arrow[r, dotted, no head] & b \arrow[dl] \\
 & c_{01} \arrow[dl] \arrow[dr] & \\
d & & e
\end{tikzcd}$$

(b) $\mathbb{F}'$

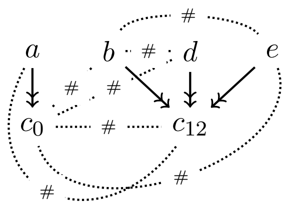
Figure 6.8: FES $\mathbb{F}$ and a folding $\mathbb{F}'$

A prototypical example of folding in FESs, which exploits the possibility of modelling disjunctive causality, is provided in Figure 6.8. The FES $\mathbb{F}'$ is obtained from $\mathbb{F}$ by merging the two conflicting c-labelled events c_0 and c_1. The resulting merged event c_{01} has a and b as $<$-predecessors, and d and e as $<$-successors. Since a and b are in conflict, exactly one of them will be in a configuration including c_{01}. The function mapping a, b, d, e identically, and c_0, c_1 to c_{01} can be easily shown to be a folding.

Now consider a more complex example in Figure 6.9a. First, if we take events c_0 and c_1 and try to merge them into a single event c_{01}, there would



(a) $\mathbb{F}_0$



(b) $\mathbb{F}_1$

Figure 6.9: Sample FESs

be no way of updating the dependency relations while keeping the behavior unchanged (since b excludes c_0 and precedes c_1, the resulting dependency between b and the merged event c_{01} would be an asymmetric conflict that cannot be represented in FESs). Instead, we can merge events c_1 and c_2 in $\mathbb{F}_1$ into a single event c_{12}, thus obtaining the FES in Figure 6.9b. In this case, the merge is possible because the original events c_1 and c_2 are enabled by $\{b\}$ and $\{d, e\}$, respectively, and since $b\#d$, $b\#e$, after the merge the same situation is properly represented as a disjunctive causality.

In order to define sets of events that can be safely merged we need some further notation. Given a set of events Z, we denote by $mc(Z)$ the set of maximal and consistent (i.e., conflict free) subsets of Z. Additionally, as in the case of AESs, we need to single out conflicts that are direct.

Definition 6.11 (direct conflict)**.** Let $\mathbb{F}$ be a FES and let $e, e' \in E$. We say that e is a *direct conflict* for e', denoted as $e \#_\delta e'$, if $e\#e'$ and $\exists Y \in mc({}^\bullet e)$ such that $Y \cup \{e'\}$ is consistent.

Intuitively, a conflict $e\#e'$ is direct when there is a way of reaching a configuration where e is enabled, without disabling e'. Note that direct conflict is not symmetric in FESs. For instance for the FES depicted in Figure 6.10, we have $a \#_\delta d$ while it is not the case that $d \#_\delta a$.

We use the extensions of relations $\#$ and $\prec$ to relations between sets and events, as already done for AESs. For instance, given $X \subseteq E$ and $e \in E$

$$\begin{tikzcd} a \arrow[r, dash, dotted, "\#"] \arrow[d] \arrow[dr, dash, dotted] & b \arrow[d] \arrow[dl, dash, dotted, "\#"] \\ c \arrow[r, dash, dotted, "\#"] & d \end{tikzcd}$$

Figure 6.10: Example of direct conflict in FES, $a \#_\delta d$ and $\neg(d \#_\delta a)$

we write $X\#^\forall e$ whenever for all $x \in X$, we have $x\#e$, or $X \prec^\exists e$ when there exists $x \in X$ such that $x \prec e$.

We can now define the notion of combinable set of events for FESs.

Definition 6.12 (combinable set of events)**.** Let $\mathbb{F}$ be a FES. A set of events $X \subseteq E$ is called *combinable* if for all $x, x' \in X$ and $e, e' \in E \setminus X$ the following holds

1. $\lambda(x) = \lambda(x')$ and $x\#x'$,
2. $x \#_\delta e \Rightarrow x'\#e$,
3. $x \prec e \Rightarrow x' \prec e \;\vee\; x'\#e$,
4. $e \prec x \Rightarrow {}^\bullet x' \neq \emptyset \;\wedge\; (e \prec x' \;\vee\; (\forall e' \prec x' \wedge e' \notin {}^\bullet x.\ e\#e'))$,
5. $x, e' \in {}^\bullet e \;\;\wedge\;\; x\#e' \wedge \;\; \neg(X \setminus \{x\}\#^\forall e')$
$$\Rightarrow \forall Y \in mc({}^\bullet e).\ \begin{array}{l} (x \in Y \Rightarrow \exists e'' \in Y \setminus \{x\}.\ e''\#e')\wedge \\ (X \cap Y = \emptyset \Rightarrow \exists e'' \in Y.\ X\#e'') \end{array}$$

Roughly speaking, condition (1) requires that the events in X are occurrences of the same activity (they have the same label and they are in conflict). Condition (2) requires that events in X have the essentially the same conflicts: for any $x \in X$, if x is in direct conflict with an event e (hence this conflict is not derivable from the $\prec$-predecessors) then all events in X must be in conflict with e. Conditions (3) and (4) state that predecessors and successors are preserved among events in X or they can become conflicts. The rough intuition is that events whose causes are in conflict can be possibly merged thus getting a single event having the conflicting causes as $\prec$-predecessors and the conflicting consequences as $\prec$-successors. More in detail, by condition (4), if an event $x \in X$ has a non-empty set of $\prec$-predecessors, then the same must be true for all events in X. Moreover, if e is a $\lessdot$-predecessors of some $x \in X$ then for any other $x' \in X$, either e is a $\lessdot$-predecessor of x' or it is in conflict with all the $\lessdot$-predecessors of x' not in common with x (namely with the events in ${}^{\bullet}x' \setminus {}^{\bullet}x$). This ensures that, whenever we merge the events in X thus joining their $\lessdot$-predecessors, the maximal consistent subsets of $\lessdot$-predecessors will remain unchanged (see Lemma 6.13, where the role of condition (4) emerges formally).

Finally, condition (5) takes into account the situation in which events $x \in X$ and $e' \in E \setminus X$ are conflicting $\lessdot$-predecessor of an event e, but not all events in X are in conflict with e'. This is problematic because, after the merging, the conflict between x and e' will be lost, thus possibly changing the maximal subsets of $\lessdot$-predecessors. The condition indeed says that merging is still allowed if the conflict $x\#e'$ is not essential when forming the maximal consistent sets of $\lessdot$-predecessors for e. In detail, it is required that for any $Y \in mc({}^{\bullet}e)$

- if $x \in Y$ then x is not the only event in Y which is in conflict with e', so that losing the conflict $x\#e'$ would not be problematic and Y would remain a maximal consistent set;

- if none of the events of X occur in Y then this is due to the presence in Y of an event e'' in conflict with all events in X (which, in particular, is not e' and thus this will remain a maximal set even if the conflict $x\#e'$ is lost).

For example, consider the FES $\mathbb{F}_2$ in Figure 6.11a. If we take $X = \{a_x, a_{x'}\}$ then condition (5) fails. Please note that events corresponding to those in condition (5) have a subscript which should suggest their role. We have ${}^{\bullet}c_e = \{a_x, a_{x'}, b_{e'}\}$ and thus $mc({}^{\bullet}c_e) = \{Y, Y'\}$ with $Y = \{a_x\}$ and $Y' = \{a_{x'}, b_{e'}\}$. Observe that $a_x \in Y$ but clearly there is no $e'' \in Y \setminus \{a_x\} = \varnothing$ satisfying $e''\#b_{e'}$. The quotient of $\mathbb{F}_2$ with respect to X (formally defined later in Definition 6.14) would lead to the FES $\mathbb{F}_3$ in Figure 6.11b, which is not behaviorally equivalent to $\mathbb{F}_2$. In particular, observe that c_e is no

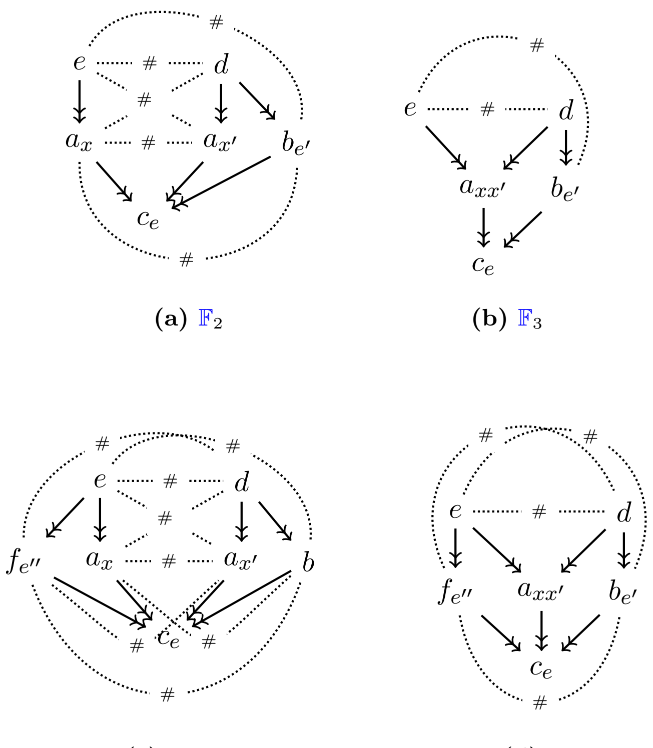


Figure 6.11: Example FESs to illustrate Condition 5 in Definition 6.12

longer executable after the occurrence of e since it would require the prior execution of $a_{xx'}$ and $b_{e'}$, which instead cannot be in the same computation since $b_{e'}\#e$. This means that $a_{xx'}\#_s b_{e'}$, i.e., the two events are in semantic conflict, although it is not the case that $a_{xx'}\#b_{e'}$ (hence the quotient FES is not faithful). Note that saturating the conflict would not solve the problem. In fact, if in the quotient FES $\mathbb{F}_3$ we enforced the conflict $a_{xx'}\#b_{e'}$, then a configuration corresponding to $\{d, a_x, b_{e'}\} \in \mathit{Conf}(\mathbb{F}_2)$ would be missing. A situation in which condition (5) is satisfied is instead illustrated by the FES $\mathbb{F}_4$ in Figure 6.11c. Again we take $X = \{a_x, a_{x'}\}$. We have ${}^\bullet c_e = \{f_{e''}, a_x, a_{x'}, b_{e'}\}$ and thus $mc({}^\bullet c_e) = \{Y, Y'\}$ with $Y = \{f_{e''}, a_x\}$ and $Y' = \{a_{x'}, b_{e'}\}$. Note that $a_x \in Y$ and there is indeed $f_{e''} \in Y$ such that $f_{e''}\#b_{e'}$.

The condition is satisfied also exchanging the roles of a_x and $a_{x'}$. Indeed, in the resulting quotient FES $\mathbb{F}_5$, depicted in Figure 6.11d, after the execution of e or d, there are still two maximal and consistent set of $<$-predecessors for the event c_e, namely $\{a_{xx'}, f_{e''}\}$ and $\{a_{xx'}, b_{e'}\}$.

We prove a technical lemma which shows that for a combinable set of events X, the maximal consistent sets of the $<$-predecessors of X and those of single events in X coincide. This clarifies the role of condition (4) in the definition of combinable set of events and will be useful later, for proving that the quotient does not alter the behavior.

Lemma 6.13 (preservation of consistent sets)**.** *Let $\mathbb{F} = \langle E, \#, <, \lambda \rangle$ be a FES and let $X \subseteq E$ be a combinable set of events. Then for any consistent set $Y \subseteq E$ it holds that $Y \subseteq {}^\bullet X$ iff there exists $x \in X$ such that $Y \subseteq {}^\bullet x$. Hence:*

$$Y \in mc({}^\bullet X) \text{ iff there exists } x \in X \text{ such that } Y \in mc({}^\bullet x).$$

Proof. Let $Y \subseteq E$ be consistent. Let us assume that $Y \subseteq {}^\bullet X = \bigcup_{x \in X} {}^\bullet x$ and prove that there exists $x \in X$ such that $Y \subseteq {}^\bullet x$. If $Y = \varnothing$ the assert is trivial. Otherwise, take $e' \in Y$. By the assumption $Y \subseteq {}^\bullet X$ there must be $x' \in X$ such that $e' \in {}^\bullet x'$. We show that $Y \subseteq {}^\bullet x'$. In fact, for any $e \in Y$ there must exists $x \in X$ such that $e \in {}^\bullet x$. Since $e < x$, by Definition 6.12(4), either $e < x'$ or we should have $e \# e'$. The latter possibility would contradict the consistency of Y. Hence it must be $e < x'$, namely $e \in {}^\bullet x'$. Therefore $Y \subseteq {}^\bullet x'$, as desired. The converse implication is trivial since ${}^\bullet X = \bigcup_{x \in X} {}^\bullet x$.

Now, the second part of the lemma, namely the fact that $Y \in mc({}^\bullet X)$ iff there exists $x \in X$ such that $Y \in mc({}^\bullet x)$ is an immediate consequence of the first. In fact, let $Y \in mc({}^\bullet X)$. Then, by the first part of the lemma we know that there is $x \in X$ such that $Y \subseteq {}^\bullet x$. Again by the first part of the lemma Y is maximal among the consistent subsets of ${}^\bullet x$, since these are also consistent subsets of ${}^\bullet X$. Hence $Y \in mc({}^\bullet x)$. Vice versa, let $Y \in mc({}^\bullet x)$. Clearly $Y \subseteq {}^\bullet X$. Moreover, Y is maximal among the consistent subsets of ${}^\bullet X$. To see this, take any $Y' \subseteq {}^\bullet X$ consistent and assume that $Y \subseteq Y'$. By the first part of the lemma, there is $x' \in X$ such that $Y \subseteq {}^\bullet x'$. Then necessarily $Y = Y'$, otherwise, by Definition 6.12(4), given $y' \in Y' \setminus Y$ we would have $y' \# y$ for any $y \in Y$, which is absurd since $Y \subseteq Y'$ and Y' consistent. $\square$

We next formally define the quotient of a FESs with respect to a combinable set of events.

Definition 6.14 (quotient of FESs). Let $\mathbb{F} = \langle E, \#, \prec, \lambda \rangle$ be a FES, X be a combinable set of events. The *quotient* of $\mathbb{F}$ with respect to X, denoted by $\mathbb{F}_{/X}$, is the FES $\mathbb{F}_{/X} = \langle E_{/X}, \#_{/X}, \prec_{/X}, \lambda_{/X} \rangle$ where

$$\begin{array}{lcl} E_{/X} & = & (E \setminus X) \cup \{e_X\} \\ \#_{/X} & = & \#_{|(E\setminus X)} \cup \{(e, e_X) \mid e \#^{\forall} X\} \\ \prec_{/X} & = & \prec_{|(E\setminus X)} \cup \{(e, e_X) \mid e \prec^{\exists} X\} \cup \{(e_X, e') \mid X \prec^{\exists} e'\} \\ \lambda_{/X} & = & \lambda_{/X}[e_X \mapsto \lambda(x)] \text{ for an event } x \in X. \end{array}$$

The *quotient map* $f_X : \mathbb{F} \to \mathbb{F}_{/X}$ is defined by $f_X(x) = e_X$ for $x \in X$ and $f_X(e) = e$ for $e \in E \setminus X$.

The rest of the section is dedicated to showing that the quotient operation on FESs induces a (elementary) folding, namely it preserves hp-bisimilarity.

The idea underlying the proof for AESs was that events that are merged are occurrences of the same activity with different histories. They could be merged if the histories were compatible and after merging, the possible histories remained the same. For FESs the intuition of the proof is similar, but now events can occur after a maximal consistent set of $\prec$-predecessors which roughly play the role of histories in AESs. By Lemma 6.13, after merging a set of combinable events this maximal subsets of consistent events remains unchanged. This will be a core ingredient in the proof that the quotient does not alter the behavior.

We start by showing some properties of the quotient map which will be used later for showing that it transforms configurations of the original FES into isomorphic configurations of the quotient FES. We do not rely on the notion of FES morphism from [Cast 97], which would be too strong for our needs (in particular, condition (iii) of [Cast 97, Definition 4] is not satisfied by our quotient map).

Lemma 6.15 (properties of the quotient map)**.** *Let $\mathbb{F} = \langle E, \#, <, \lambda \rangle$ be a FES, $X \subseteq E$ be a combinable set of events let $f_X : \mathbb{F} \to \mathbb{F}_{/X}$ be the quotient map. Then for all $e, e' \in E$:*

1. *if $f_X(e) \#_{/X} f_X(e')$ then $e \# e'$*
2. *if $e < e'$ then $f_X(e) <_{/X} f_X(e')$;*
3. *if $f_X(e) <_{/X} f_X(e')$ then $e < e' \ \vee \ e \# e'$*
4. *if $f_X(e) = f_X(e')$ then $e = e' \ \vee \ e \# e'$.*

Proof. 1. Let $e, e' \in E$ and assume $f_X(e) \#_{/X} f_X(e')$. Notice that at least one between e and e' is not in X, otherwise we would have $f_X(e) = f_X(e')$ that is a contradiction since, by construction, $\#_{/X}$ is irreflexive. We distinguish various cases. If $e \in X$ and thus $f_X(e) = e_X$, then by definition of conflict in the quotient FES (Definition 6.14), since $f_X(e) = e_X \#_{/X} f_X(e')$, it must be $X \#^{\forall} e'$, and thus in particular $e \# e'$, as desired. The case in which $e' \in X$ is analogous, since conflict is symmetric. Otherwise, if $e, e' \notin X$ the property trivially holds, since f_X is the identity on e, e' and their mutual relations are not changed by the quotient operation.

2. Let $e, e' \in E$ be such that $e < e'$. Note that it cannot be $e, e' \in X$, otherwise, we would have $e < e'$ and, by Definition 6.12(1), $e \# e'$, violating the disjointness of $<$ and $\#$. Hence we distinguish the following cases:

- If $e \in X$ and $e' \notin X$, by Definition 6.14, $e_X = f_X(e) <_{/X} f_X(e') = e'$ as desired.
- If $e' \in X$ and $e \notin X$, by construction, $e = f_X(e) <_{/X} f_X(e') = e_X$.
- If $e, e' \notin X$ then f_X is the identity on e, e' and the result trivially holds.

3. Let $e, e' \in E$ be such that $f_X(e) <_{/X} f_X(e')$. Note that it cannot be $e, e' \in X$, otherwise, we would have $f_X(e) = e_X <_{/X} e_X = f_X(e')$, while by construction $<_{/X}$ is irreflexive. Hence we distinguish the following cases:

- If $e \in X$ and $e' \notin X$, by construction, there exists $x' \in X$ such that $x' < e'$. Then, either $x' = e$ and thus $e < e'$, or, by Definition 6.12(3), $e' \# e$ as desired.

- If $e' \in X$ and $e \notin X$, by construction, there exists $x \in X$ such that $e < x$. Then, either $x = e'$ and thus $e < e'$, or, by Definition 6.12(4), $e' \# e$ as desired.

- Otherwise, if $e, e' \notin X$ then f_X is the identity on e, e' and hence $e < e'$.

4. Let $e, e' \in E$ such that $f_X(e) = f_X(e')$, with $e \neq e'$. This means that $e, e' \in X$ and thus, since the events in X are pairwise conflicting, we have that $e \# e'$. □

We can now show that the quotient map transforms any configuration of the original FES into an isomorphic configuration of the quotient.

Lemma 6.16 (quotient preserves configurations). *Let $\mathbb{F} = \langle E, \#, <, \lambda \rangle$ be a FES, $X \subseteq E$ be a combinable set of events and let $f_X : \mathbb{F} \to \mathbb{F}_{/X}$ be the quotient map. For any configuration $C \in Conf(\mathbb{F})$ then $f_X(C) \in Conf(\mathbb{F}_{/X})$ and, additionally, $f_{X|C} : (C, <^*_C) \to (f_X(C), <^*_{f_X(C)})$ is an isomorphism of configurations.*

Proof. We first prove that $f_X(C)$ is a configuration.

1. $f_X(C)$ is conflict free.

   This follows directly from Lemma 6.15(1). In fact, for $e, e' \in C$, if it were $f_X(e) \#_{/X} f_X(e')$ then we would deduce $e \# e'$, contradicting the fact that C is a configuration.

2. $f_X(C)$ has no $<$-cycles.

   Observe that, by Lemma 6.15(3), f_X reflects the flow relation over events of a configuration, namely for $e, e' \in C$, if $f_X(e) <_{/X} f_X(e')$ then $e < e'$ (since the case $e \# e'$ would contradict the fact that C is a configuration). As a consequence, a $<$-cycle in $f_X(C)$ would be reflected in C.

3. For all $z \in f_X(C)$ and $z' \notin f_X(C)$ s.t. $z' < z$, there exists $z'' \in f_X(C)$ such that $z' \# z'' < z$.

   Let $z \in f_X(C)$, $z' \notin f_X(C)$, such that $z' < z$. Therefore, there are $e \in C$ such that $z = f_X(e)$ and, by surjectivity of f_X, $e' \notin C$ such that $z' = f_X(e')$.

   By Lemma 6.15(3) either (i) $e' \# e$ or (ii) $e' < e$. Below we treat the two cases separately.

   (i) If $e' \# e$, the fact that $\neg(e' < e)$ while $f_X(e') <_{/X} f_X(e)$, the construction in Definition 6.14, implies that one of the following holds:

   - $e \in X$ and there exists $x \in X$ such that $e' < x$.
     Note that the conflict $e \# e'$ cannot be direct, otherwise, by Definition 6.12(2), one should have also $x \# e'$. Hence, since by definition of configuration, the set ${}^\bullet e \cap C \in mc({}^\bullet e)$, there must be $e'' \in {}^\bullet e \cap C$ such that $e' \# e''$. Hence $e'' \in C$ and $e'' < e$. Therefore by Lemma 6.15(2), $f_X(e'') <_{/X} f_X(e) = z$. Moreover, since $e', e'' \notin X$, we have $f_X(e'') \#_{/X} f_X(e') = z'$, as desired.
   - $e' \in X$ and there exists $x' \in X$ such that $x' < e$.
     In this case note that $f_X(x') = f_X(e') = e_X$ and thus we can take x' instead of e', and proceed as in case (ii).

   (ii) Let us focus on the other case, in which $e' < e$. Since C is a configuration, there exists $e'' \in C$ such that $e'' < e$ and $e'' \# e'$. By Lemma 3, $f_X(e'') < f_X(e) = z$. We distinguish various subcases:

   (a) $\{e', e''\} \subseteq X$. This simply cannot happen as it would imply $f_X(e') = f_X(e'') \in f_X(C)$, while we are assuming $f_X(e') \notin f_X(C)$.

   (b) $e' \in X, e'' \notin X$. Let $Y \in mc({}^\bullet e)$ be the set of maximal and consistent set of predecessors of e in C. Obviously, $e'' \in Y$ and, by Lemma 6.15(2), for all $e_1 \in Y$ we have $f_X(e_1) < f_X(e) = z$ and $f_X(e_1) \in f_X(C)$. Clearly, there is no $e_2 \in Y \cap X$ such that $e_2 \in C$, otherwise $f_X(e_2) = f_X(e') = z' \in f_X(C)$ and this would contradict the assumptions. Therefore, $Y \cap X = \emptyset$ and, by Definition 6.12(5), there exists $e_1'' \in Y$ such that $e_1'' \#^{\forall} X$. In this case, by construction, $f_X(e_1'') \#_{/X} f_X(e') = z' = e_X$ and, since $f_X(e_1'') \in f_X(C)$, this gives the desired result.

(c) $e' \notin X, e'' \in X$. By Definition 6.12(5), for all $Y \in mc({}^{\bullet}e)$, with $e'' \in Y$ there is $e_1 \in Y \setminus \{e''\}$ such that $e_1 \# e'$. Since neither e' nor e_1 are in X, this conflict is preserved by the quotient map and thus $f_X(e_1) \# f_X(e') = z'$. Since, $f_X(e_1) \in f_X(C)$ and, by Lemma 6.15(2), $f_X(e_1) \prec f_X(e) = z$, we get the desired results.

(d) $\{e', e''\} \nsubseteq X$. Since $\{e', e''\} \nsubseteq X$ and $e' \# e''$ then, by Lemma 6.15(1), $f_X(e'') \# f_X(e')$, as desired.

Concerning the last assertion, note that the fact that $f_{X|C} : (C, \prec_C^*) \to (f_X(C), \prec_{f_X(C)}^*)$ is an isomorphisms follows immediately by items (2) and (3) of Lemma 6.15. □

Recall that FESs are assumed to be faithful, full and disjoint. We next prove that the quotient preserves this properties.

Lemma 6.17 (quotient is full and faithful). *Let $\mathbb{F} = \langle E, \#, \prec, \lambda \rangle$ be a FES, X a combinable set of events and let $f_X : \mathbb{F} \to \mathbb{F}_{/X}$ be the quotient map. The FES $\mathbb{F}_{/X}$ is 1) faithful, 2) full and 3) disjoint.*

Proof. 1. Let $z, z' \in E_{/X}$ be events in $\mathbb{F}_{/X}$ such that $\neg(z \# z')$. We need to prove that there exists a configuration $C_1 \in Conf(\mathbb{F}_{/X})$ such that $\{z, z'\} \subseteq C_1$.

Take $e, e' \in E$ such that $f_X(e) = z$ and $f_X(e') = z'$ (they exist since f_X is surjective). If $\neg(e \# e')$ then, by faithfulness of $\mathbb{F}$, there exists $C_0 \in Conf(\mathbb{F})$ such that $\{e, e'\} \subseteq C_0$. By Lemma 6.16, $f_X(C_0) \in Conf(\mathbb{F}_{/X})$ is the desired configuration, since $\{z, z'\} = \{f_X(e), f_X(e')\} \subseteq f_X(C_0)$.

If instead $e \# e'$, it means that one of the two events is in X (otherwise their dependencies would not be changed by the quotient). Assume without loss of generality that $e \in X$, hence $z = e_X$, and $e' \notin X$. The fact that $\neg(f_X(e) \# f_X(e'))$ means that there is $e'' \in X$ such that $\neg(e'' \# e')$. Therefore, again by fullness there exists $C_0 \in Conf(\mathbb{F})$ such that

$\{e'', e'\} \subseteq C_0$ and we conclude as above. In fact, $f_X(e'') = f_X(e) = z$, hence $\{z, z'\} = \{f_X(e), f_X(e')\} \subseteq f_X(C_0)$, which is a configuration by Lemma 6.16.

2. By Lemma 6.15(1) and surjectivity of f_X, a self-conflicting (inconsistent) event in $\mathbb{F}_{/X}$ would be reflected in $\mathbb{F}$. More precisely, let $z \in \mathbb{F}_{/X}$ such that $z \# z$. Then take $e \in \mathbb{F}$ such that $f_X(e) = z$. We have $f_X(e) \# f_X(e)$ and thus, by Lemma 6.15(1), $e \# e$, contradicting the fullness of $\mathbb{F}$.

3. In order to show that $\#_{/X}$ and $\prec_{/X}$ are disjoint we proceed by contradiction. Assume that $z \prec_{/X} z'$ and $z \#_{/X} z'$. By Definition 6.14 there are $e, e' \in E$ such that $f_X(e) = z$, $f_X(e') = z'$ and $e \prec e'$. However, by Lemma 6.15(1), we have also $e \# e'$, contradicting the disjointness of $\mathbb{F}$. □

Building on the previous technical results, we can finally prove that the quotient map f_X is a folding, i.e., that it can be seen as a hp-bisimulation.

Theorem 6.18 (quotient map is a folding)**.** *Let $\mathbb{F} = \langle E, \#, \prec, \lambda \rangle$ be a FES and let $X \subseteq E$ be a combinable set of events. Then the quotient map $f_X : \mathbb{F} \to \mathbb{F}_{/X}$ is a folding.*

Proof. Let $\mathbb{F}$ be a FES, X be a combinable set of events and $f_X : \mathbb{F} \to \mathbb{F}_{/X}$ be the quotient map, where $\mathbb{F}_{/X} = \langle E_{/X}, \#_{/X}, \prec_{/X}, \lambda_{/X} \rangle$. We prove that

$$R = \{(C_1, f_{X|C_1}, f_X(C_1)) \mid C_1 \in \mathit{Conf}(\mathbb{F})\}$$

is a hp-bisimulation.

Given a configuration $C_1 \in \mathit{Conf}(\mathbb{F})$, define $C_2 = f_X(C_1)$. Recall from Lemma 6.16 that $f_{X|C_1} : (C_1, \prec^*) \to (C_2, \prec^*)$ is an isomorphisms of pomsets.

In order to show that R is a hp-bisimilarity it remains to prove that

1. if there is $e \in E$ such that $C_1 \cup \{e\} \in \mathit{Conf}(\mathbb{F})$ then $C_2 \cup \{f_X(e)\} \in \mathit{Conf}(\mathbb{F}_{/X})$.

2. if there is $z \in E_{/X}$ such that $C_2 \cup \{z\} \in \mathit{Conf}(\mathbb{F})$ then there is $e \in E$ such that $f_X(e) = z$ and $C_1 \cup \{e\} \in \mathit{Conf}(\mathbb{F})$.

In fact, since the extension order for FESs is subset inclusion, (1) and (2) above correspond to conditions (a) and (b) in Definition 3.42.

We prove the two points separately.

1. The fact that if $C_1 \cup \{e\} \in \mathit{Conf}(\mathbb{F})$ then $C_2 \cup \{f_X(e)\} \in \mathit{Conf}(\mathbb{F}_{/X})$ follows immediately by Lemma 6.16, since $C_2 \cup \{f_X(e)\} = f_X(C_1 \cup \{e\})$.

2. Let $z \in E_{/X}$ be such that $C_2 \cup \{z\} \in \mathit{Conf}(\mathbb{F}_{/X})$ and let us show that there is an event $e \in E$ such that $f_X(e) = z$, $C_1 \cup \{e\} \in \mathit{Conf}(\mathbb{F})$.

   Let $Y_2 = {}^\bullet z \cap C_2$ be the set of $\prec$-predecessors of z in C_2. By definition of configuration in FESs we know that $Y_2 \in mc({}^\bullet z)$.

   We distinguish two cases:

   (a) $z = e_X$.

   In this case events in ${}^\bullet z$ are left unchanged by the quotient and hence if we let $Y_1 = Y_2$ we have that $Y_1 \subseteq C_1$, $f_X(Y_1) = Y_2$ and Y_1 is consistent. By definition of the quotient (Definition 6.14) we have that $e \prec_{/X} e_X$ iff $e \prec^\exists X$ and hence $Y_1 \subseteq {}^\bullet X$ and, by Lemma 6.13, there is an event $e' \in X$, s.t. $Y_1 \in mc({}^\bullet e')$. Since $Y_1 \subseteq C_1$, we deduce that $C_1 \cup \{e'\} \in \mathit{Conf}(\mathbb{F})$, with $f_X(e') = e_X$, as desired.

   (b) $z \neq e_X$.

   In this case the event $z = e \in E \smallsetminus X$ is mapped identically by the quotient map f_X. In order to conclude, we just need to show that $C_1 \cup \{e\}$ is a configuration. Let $Y_1 = \{e' \in C_1 \mid f_X(e') \in Y_2\}$. We have that $Y_1 \subseteq {}^\bullet e$. In order to prove this fact, note that for any $e' \in Y_1$, since $f_X(e') \prec_{/X} f_X(e) = z$, by Lemma 6.15(3) we know that $e' \prec e$ or $e' \# e$. We show that the second case cannot happen. If $e' \notin X$ this is obvious. Otherwise, if $e' \in X$, from $\neg(f_X(e) \#_{/X} f_X(e'))$, by Definition 6.14, there is $x \in X$ such that $\neg e \# x$. Then by Definition 6.12(2), the conflict $e' \# e$

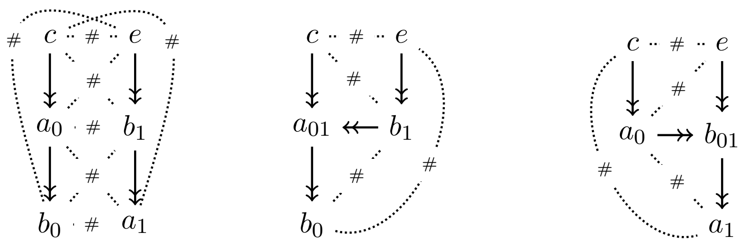
is not direct. Therefore, since ${}^{\bullet}e' \cap C_1 \in mc({}^{\bullet}e')$, by definition of direct conflict, there is $e'' \in {}^{\bullet}e' \cap C_1$ such that $e''\#e$. Since $e'' \notin X$, this conflict is preserved by the quotient map and we get that $f_X(e'')\#_{/X} f_X(e)$, which is absurd since $f_X(e), f_X(e'') \in f_X(C_1) \cup \{z\}$, and the latter is a configuration by hypothesis.

The set Y_1 is clearly consistent, since it is included in C_1. It is also maximal, i.e., $Y_1 \in mc({}^{\bullet}e)$. In fact if it were not maximal, there would be $e'' \in {}^{\bullet}e \setminus Y_1$ such that $Y_1 \cup \{e''\}$ is consistent. But then, since the quotient map preserves configurations and thus consistent sets, $f_X(Y_1 \cup \{e''\})$ would be consistent and strictly larger then Y_2.

Since $Y_1 \in mc({}^{\bullet}e)$, we conclude that $Y_1 \cup \{e\}$ is a configuration, as desired.

□

As in the case of AESs the iterative application of the quotient operation to a finite FES leads to a "minimal" FES hp-bisimilar to the original one. Different sequences of quotient operations can lead to non-isomorphic FESs, which are not further reducible. An example is provided in Figure 6.12. In the FES $\mathbb{F}_6$ there are two combinable sets of events, namely $\{a_0, a_1\}$ and $\{b_0, b_1\}$. In the quotient $\mathbb{F}_{6/\{a_0,a_1\}}$, depicted in Figure 6.12(b), the set $\{b_0, b_1\}$ is no longer combinable. In fact, condition (4) in Definition 6.12 is violated since $a \prec b_0$, but it does not hold that $a \prec b_1$ and there is event e such that $e \prec b_1$, $\neg(e \prec a)$ and $\neg(a\#e)$. Similarly, in the quotient $\mathbb{F}_{6/\{b_0,b_1\}}$, depicted in Figure 6.12(c), the set $\{a_0, a_1\}$ is no longer combinable. Hence we get two non-isomorphic FESs which are not further reducible. Also in this case, one can see that this is intrinsic in the nature of FESs and their foldings. In fact, by inspecting all the possible label preserving surjective mappings one realizes that the two quotients does not admit any non-trivial folding. The following section shows an approach to address this issue.

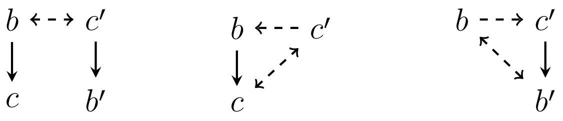


(a) $\mathbb{F}_6$ **(b)** $\mathbb{F}_{6/\{a_0,a_1\}}$ **(c)** $\mathbb{F}_{6/\{b_0,b_1\}}$

Figure 6.12: FES and two minimal non-isomorphic quotients

6.4 Deterministic foldings and canonicity

In order to exploit the reduced event structures for model comparison purposes, the result of the foldings should be uniquely determined from the original model. In other words, starting from two isomorphic AESs or FESs and repeatedly applying the behavior preserving folding operation, the resulting minimal AESs of FESs should be isomorphic.



(a) $\mathbb{A}_7$ **(b)** $\mathbb{A}_8$ **(c)** $\mathbb{A}_9$

Figure 6.13: Equivalent AESs

As showed in previous sections, different choices of the sets of events to be folded can lead to different minimal representations. For instance, the AESs $\mathbb{A}_8$ and $\mathbb{A}_9$ in Figure 6.13 can be obtained from $\mathbb{A}_7$ by folding events b, b' or c, c', respectively. They are not further reducible and thus they provide minimal representations of the same AES.

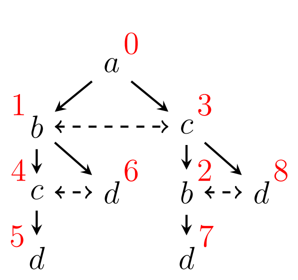
In order to address this problem, we leverage some concepts from graph theory. Specifically, we rely on the concept of canonical labeling of a graph [McKa 81], that originates as an approach to deciding graph isomorphism. Let $Canon(G)$ be a function that maps a graph G to a *canonical label* in the sense that, given graphs G and H, we have $Canon(G) = Canon(H)$ iff H and G are isomorphic. If we use the string representation of the adjacency matrix of a graph, then a canonical label for a graph G can be determined by computing all permutations of its adjacency matrix and selecting the largest (or the smallest) lexicographical exemplar among them. Clearly, this approach is computationally expensive, but state-of-the-art software implement several practical heuristics to compute canonical labels.

Formally, let $G = (V, A)$ be a graph, where V is the set of vertices and A the set of arcs. Moreover, let $M(G)$ be the adjacency matrix of G, in some fixed linear representation. For any order of the set of vertices, represented as a numbering $\gamma : V \to \{0, 1, ...|V|\}$, we get a corresponding string $M(G)^{\gamma}$. Then the canonical label of G is the string induced by an order $\hat{\gamma}$, s.t., $M(G)^{\gamma} \leq_{lex} M(G)^{\hat{\gamma}}$ holds for every possible order γ. The order $\hat{\gamma}$ is referred to as the canonical order.

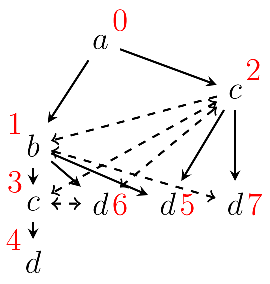
In our implementation, we use **nauty** (`http://pallini.di.uniroma1.it/`) for computing the graph canonical label and the corresponding order $\hat{\gamma}$ on the vertices which is mostly of interest for us. Nauty and other similar tools work on graphs with unlabeled edges, while an event structure can be naturally seen as graphs with labeled edges. The problem is easily overcome by using some isomorphism preserving transformation of edge-labeled into edge-unlabeled graphs (we used the one in [Kant 10]).

The canonical order on the vertices of the graph associated to an event structure can be easily used to establish a total order on the folding that yields a minimal and canonical AES or FES. For a combinable set of events X, we denote by $X^{\hat{\gamma}}$ the ordered string of numbers corresponding to the events in X.

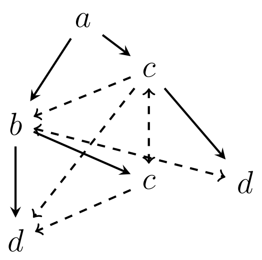
Definition 6.19 (deterministic folding)**.** Let $\mathbb{E}$ be an event structure and $\hat{\gamma} : E \to \mathbb{N}_0$ be the canonical order of events. Let $X, Y \subseteq E$ be combinable



(a) $\mathbb{A}_{10}$



(b) $f(\mathbb{A}_{10})_{/\{b,b\}}$



(c) $f^{+}(\mathbb{A}_{10})$

Figure 6.14: Canonical labeling and folding

sets of events. Then the precedence of X over Y in a deterministic folding is defined by the following conditions, listed in decreasing relevance:

(i) $\lambda(e) >_{lex} \lambda(e')$ where $e' \in Y$ and $e \in X$, or

(ii) $\lambda(e) =_{lex} \lambda(e') \wedge |X| > |Y|$, or

(iii) $\lambda(e) =_{lex} \lambda(e') \wedge |X| = |Y| \wedge X^{\hat{\gamma}} >_{lex} Y^{\hat{\gamma}}$.

Whenever, applying folding according to such order, we reach an event structure where no further folding steps are possible, this is denoted by $f^{+}(\mathbb{E})$ and referred to as *minimal canonical folding.*

Figure 6.14 illustrates the canonical folding of $\mathbb{A}_{10}$. The AES $\mathbb{A}_{10}$ shows the order $\hat{\gamma}$ assigned by nauty. The combinable sets of events in $\mathbb{A}_{10}$ are $\{\{b(1), b(2)\}, \{c(3), c(4)\}, \{d(5), d(6)\}, \{d(7), d(8)\}\}$, and from Definition 6.19 we know that $\{b(1), b(2)\}$ takes precedence over the others. The folding of $\{b(1), b(2)\}$ is depicted in Figure 6.14b. Note that a fresh event b is added, replacing the set $\{b(1), b(2)\}$, and the order $\hat{\gamma}$ is recalculated for the new AES. Finally, Figure 6.14c depicts the minimal and canonical AES. In this particular case, it was necessary to keep two events with label c and two with label d to preserve the behavior.

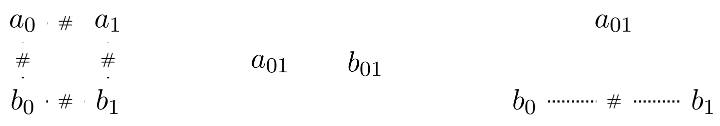
The fact that the order on folding steps given in Definition 6.19 is clearly total, and thus folding is essentially deterministic, ensures that the reduction of an event strucutre will produce a uniquely determined result.

Proposition 6.20 (canonical folding of an ES)**.** *Let $\mathbb{E}_1$ and $\mathbb{E}_2$ be isomorphic event structures. Then the deterministic folding of $\mathbb{E}_1$ and $\mathbb{E}_2$ produces a canonical event structure, such that $f^+(\mathbb{E}_1)$ is isomorphic to $f^+(\mathbb{E}_2)$.*

6.5 Discussion

This chapter presented (deterministic) reduction techniques for AESs and FESs. As mentioned previously, the use of more compact representations of the behavior, in the context of process model comparison, can lead to smaller and more succinct difference diagnostics. It is straightforward to see that canonical and minimal AESs and FESs can be used in the comparison technique presented in previous chapter, where the verbalization of their relations has also been presented.

A natural question arising from the quotient operations presented in this chapter is concerning the completeness of the technique. More precisely, is any folding induced by a sequence of quotient operations proposed for AES and FES? The answer is negative. In fact, consider the PES in Figure 6.15(a), which can either be seen as an AES or a FES. It admits the folding in Figure 6.15(b), where a_0, a_1 are merged into a_{01} and similarly b_0, b_1 are merged into b_{01}. It is not difficult to see that $\mathbb{P}'$ cannot be obtained by our quotient operations, neither seeing $\mathbb{P}$ as an AES nor as a FES.



(a) $\mathbb{P}$ **(b)** $\mathbb{P}'$ **(c)** $\mathbb{P}_{/\{a_0,a_1\}}$

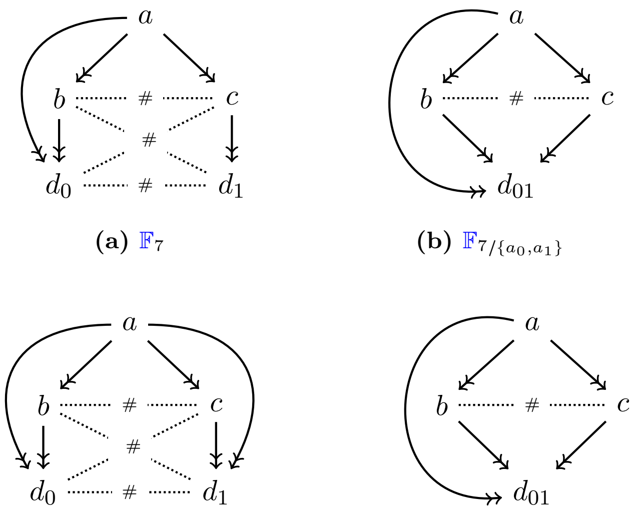
Figure 6.15: A PES $\mathbb{P}$ and a possible folding $\mathbb{P}'$ that cannot be obtained by composing elementary foldings

The limitation seems to reside in the fact that a quotient operations realize only elementary foldings (only a single set of events is merged each time). Indeed, the folding $\mathbb{P}'$ cannot be expressed as the composition of elementary foldings. For instance, notice that the quotient $\mathbb{P}_{/\{a_0,a_1\}}$ in Figure 6.15(c) is not a folding. In fact event a_0 should be simulated by $f_{\{a_0,a_1\}}(a_0) = a_{01}$. However, in $\mathbb{P}_{/\{a_0,a_1\}}$ after a_{01} we can execute b_0 while in $\mathbb{P}$ after a_0 event b_0 is ruled out.

Preliminary results lead us to conjecture that the quotient technique for AESs is complete for elementary foldings and a complete technique for general foldings can be defined at the price of reducing the efficiency (all sets to be merged have to be searched for at the same time). For FESs, instead, the intensional nature of the dependency relations seems to be an obstacle towards a completeness result already for elementary foldings.

Consider the FESs in Figure 6.16, $\mathbb{F}_7$ and $\mathbb{F}_8$, and corresponding quotients with respect to the set $X = \{d_0, d_1\}$. It is not difficult to see that the two FESs have exactly the same posets of configurations. Indeed, the only difference between $\mathbb{F}_7$ and $\mathbb{F}_8$ is the absence, in the former, of the flow $a \prec d_1$. Since a is the only $\prec$-predecessor of c, which in turn is the only $\prec$-predecessor of d_1, this flow is semantically enforced. Hence, its explicit presence does not alter, in any way, the behavior. However, this subtle difference is very important for the quotient operation. It is immediate to see that $\{d_0, d_1\}$ is combinable in $\mathbb{F}_8$; whereas, $\{d_0, d_1\}$ is not combinable in $\mathbb{F}_7$, because condition (4) of Definition 6.12 is violated. In fact, in $\mathbb{F}_7$, we have

$a < d_0$, but $\neg(a < d_1)$ and there is $c < d_1$ such that $\neg(c < d_0)$ and $\neg(c \# a)$. Still, the quotient operation applied to $\mathbb{F}_7$ and $\mathbb{F}_8$ produces the same result $\mathbb{F}_{7/\{d_0,d_1\}} = \mathbb{F}_{8/\{d_0,d_1\}}$. Thus, in both cases, the quotient preserves the behavior, namely it induces an elementary folding, but only the second is allowed by our technique. This means that, in the case of FESs, completeness fails also for elementary foldings. We conjecture that this problem can be faced by restricting to classes of FESs where the dependency relations are saturated (in the spirit of the faithfulness and fullness requirements).



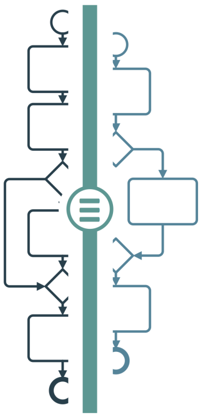
(a) $\mathbb{F}_7$ **(b)** $\mathbb{F}_{7/\{a_0,a_1\}}$

(c) $\mathbb{F}_8$ **(d)** $\mathbb{F}_{8/\{a_0,a_1\}}$

Figure 6.16: Non-completeness of the quotient technique for FESs

Chapter 7

Implementation and validation



We implemented the ideas presented in the previous Chapters in a research prototype, called BP-Diff [Arma 14b]. BP-Diff takes as input pairs of process models expressed in the standard BPMN notation and produces diagnostics of the differences found, both in visual and textual form. The textual feedback explains how a given pair of tasks is related in one model in contrast to the other model. Meanwhile, the visual feedback allows users to pinpoint the exact configuration where the discrepancy occurs. In its current version, BP-Diff uses the AES representation for the behavioral comparison, i.e., the internal representation of the behavior of a process is a canonically reduced AES and the differences are verbalized using the relations in AESs. The tool is publicly available as a Software-as-a-Service at `http://diffbp-BP-Diff.rhcloud.com/` and its source code can be found at `https://code.google.com/p/fdes/`.

Specifically, given a pair of process models, the BP-Diff tool: i) computes the canonically reduced AES of the behavior of each of the processes, ii) constructs the partial synchronized product, iii) identifies the best partial matchings between the maximal configurations of the AESs, and iv) outputs the pairs of mismatching relations (herein called *discrepancies*) in the corresponding AESs

In this setting, a discrepancy consists of: (i) a configuration where a behavioral difference is observed; and (ii) a description of the discrepancy, meaning what behavior is observed in said configuration in one of the models but not in the other. Note that designating the configuration where the discrepancy occurs is important as the same pair of events can appear in different behavioral relations depending on the configuration when they occur. For example, in the process model M_4 depicted in Figure 7.1 there is a run where the task *n form* precedes task *exe*; conversely, there is another run where *n form* does not occur together with *exe* (i.e., when *t form* occurs).

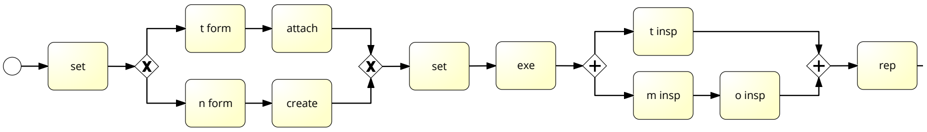


Figure 7.1: Process model example M_4

A configuration is characterized by the set of event occurrences leading to it. However, describing this set of events by means of a textual statement is impractical and would hinder on the understandability of the resulting statements. Accordingly, BP-Diff reports only the last event(s) that need to occur before the configuration where the discrepancy arises. To complement this incomplete textual designation of a configuration, BP-Diff additionally offers a graphical representation thereof by highlighting the events that lead to the configuration in question. For example, in the process model in Figure 7.2, the configurations where the difference occurs are highlighted in green. The discrepancy itself relates to the execution of the activities *m insp* and *o insp* (highlighted in red), which may occur in one model but not in the other in the configurations in question. The numbers attached to each of the highlighted elements represent the number of times the task was executed – this is relevant when there is repetitive behavior in the process. In the example of Figure 7.2, the numbers in gray boxes are attached to the tasks that lead to the execution of *o insp*; whereas, those in green boxes are attached to the tasks that lead to the execution of *m insp*.

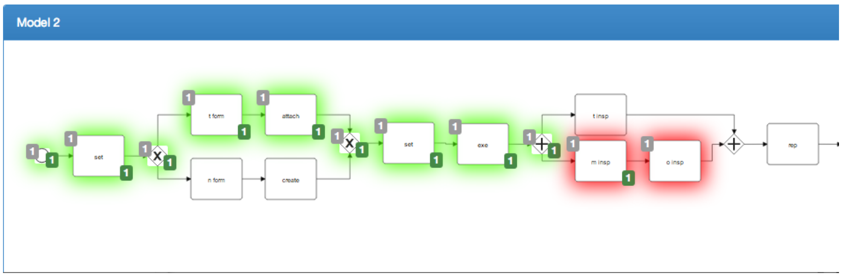


Figure 7.2: Representation of differences

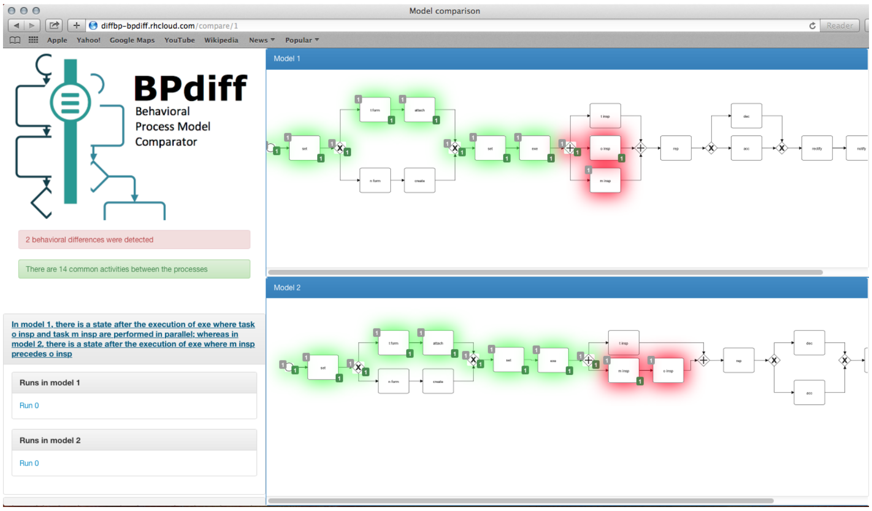
BP-Diff provides a simple Web interface as depicted in Figure 7.3. The textual descriptions of the encountered discrepancies are displayed on the left hand side of the screen. Finally, the models are rendered on the right-hand side of the window. The tool relies on a third-party libraries for the rendering BPMN process models, specifically Camunda's BPMN.io JavaScript library[1].

7.1 Evaluation

Performance

In order to assess the scalability of our method, we measured the performance of BP-Diff with respect to one of its more critical steps, namely the

[1]http://bpmn.io/



2 behavioral differences were detected

There are 14 common activities between the processes

In model 1, there is a state after the execution of exe where task o insp and task m insp are performed in parallel; whereas in model 2, there is a state after the execution of exe where m insp precedes o insp

Runs in model 1

Run 0

Runs in model 2

Run 0

Figure 7.3: Web interface of BP-Diff

computation of canonical reduced AESs. To this end, we used the BIT process library (release 2009), which is a collection of real-life process models from financial services, telecommunications and other domains [Fahl 09]. From this collection of models we selected the subset of sound models, since not all of the models exhibit this property. The final dataset consists of 352 models, with an average size of 12.4 elements. More details are given in Table 7.1.

Library	Number of models	Number of elements		
		Min	Max	Avg
A	152	3	33	12.3
B3	184	3	37	9.1
C	16	8	36	17.3

Table 7.1: BIT process library

We computed the canonical reduced AES for each model in the collection five times and averaged the execution time. The tests were run on a laptop computer running Mac OS X with a 2 GHz Intel Core i7 with 4 GB of main memory. As the tool is implemented in Java, we used an Oracle Java Virtual Machine 1.7 with 1 GB of maximum heap size.

Three models (two from Library A and one from Library C) were discarded because of the large overhead of the computation of the canonical labeling. It was due to the large size of the corresponding PESs, which consisted of 566, 779 and 1630 events, respectively. The computation time over such PESs was larger than 8 minutes. Table 7.2 summarizes the sizes of the resulting PESs and AESs for the remaining process models (150 in Library A, 184 in Library B and 15 in Library C). The greatest reduction of the AESs was observed in the process models from the library C, where the average size of the event structures was reduced from 57.93 to 36.27 from the PESs to AESs, accounting for a reduction of 37 %.

	#Events PES			#Events AES		
	Min	Max	Avg	Min	Max	Avg
A	3	203	18.27	3	203	16.77
B3	3	72	9.08	3	72	8.53
C	6	420	57.93	4	259	36.27

Table 7.2: Sizes of PESs and AESs for the BIT process library

The minimum, maximum and average execution times for this experiment are reported in Table 7.3. To better understand the source of overhead, the execution times are split for each of the major phases in the method, namely the computation of the PES (including the computation of the unfolding prefix of the net system), the computation of the canonical labeling of events in the PES, and the computation of the corresponding minimal canonical folded AES.

The largest execution time was observed on the computation of AESs for Library C. This is somewhat consistent with the fact that this Library also observed the largest reduction in the size of its AESs. Both overhead and reduction can be associated with a pair of process models that are particularly complex in their topology and that resulted in large PESs. The

	Computation time (sec)								
	PES			Canonical labeling			Minimal canonical folded AES		
	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg
A	0	2.39	0.06	0.01	0.56	0.02	0	5.93	0.07
B3	0	0.32	0.02	0.01	0.10	0.01	0	1.19	0
C	0.01	52.61	3.67	0.01	2.92	0.36	0	535.61	36.76

Table 7.3: Execution times for computing the minimal canonical AESs of the BIT process library

large size of those PESs induced also a high overhead in the computation of the folding of the AESs. In spite of the above, the average execution time remains reasonable in the order of seconds for most of the cases.

Comparison: Performance and size of the diagnostics

We conducted a second set of experiments to assess the size of the diagnostics reported to users. To this end, we selected a collection of process models taken from the process for handling land development applications used by two Australian states, namely South Australia (SA) and Western Australia (WA). The collection consists of 3 pairs of process models in BPMN notation, each pair corresponding to subprocesses of the whole land development application process from each state. The models use uniform naming conventions, in a way that we can consider that nodes with the same label as referring to the same concrete task. Table 7.4 presents the size of models in the final dataset.

Table 7.5 presents the size of the event structures associated to the models in the land development dataset. In this case, the size of the AES remained the same for the first two pairs of models. However, we observe a reduction for the AESs associated to the third case. The execution times for computing the canonical folded AESs are shown in Table 7.6.

Model	Number of BPMN elements
SA 1	37
SA 2	47
SA 3	36
WA 1	28
WA 2	50
WA 3	31

Table 7.4: Land development application process

	# Events	
	PES	**AES**
SA 1	13	13
SA 2	80	80
SA 3	52	30
WA 1	14	14
WA 2	80	80
WA 3	46	23

Table 7.5: Size of event structures of the land development application dataset

The execution times for comparing the introduced pairs of process models are presented in Table 7.7. The execution times includes the computation of the partial synchronized product for every pair of models.

The diagnostic of differences found when comparing the third pair of models comprised 104 statements when using PES, which was reduced to 80 statements when using AES. A more detailed analysis of the diagnostics showed that 14 statements distilled when comparing the PESs where summarized by 4 statements generated using AESs. Moreover, 10 statements generated from the comparison of PESs were not longer required because the corresponding events were folded in the AESs. All the remaining diagnostic statements were the same for both types of event structures.

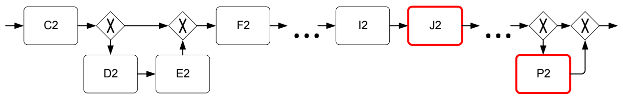
Consider the excerpts of the process models SA 3 and WA 3, which are presented in Figure 7.4. Concretely, we consider the differences involving tasks J2 and P2, which are rendered with red borders in the picture.

	Computation time (sec)		
	PES	Canonical numbering	Minimal canonical folded AES
SA 1	0.25	0.05	0.01
SA 2	1.34	0.14	0.44
SA 3	1.19	0.12	0.22
WA 1	0.09	0.03	0
WA 2	0.95	0.08	1.33
WA 3	0.32	0.05	0.54

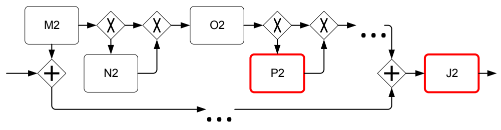
Table 7.6: Size of event structures, PESs and AESs, for the land development application process

		Avg. Time (sec)		**# Differences**	
Model 1	**Model 2**	PES	AES	PES	AES
SA 1	WA 1	0.16	0.29	23	23
SA 2	WA 2	2.79	5.17	6	6
SA 3	WA 3	98.56	145.52	104	80

Table 7.7: Comparison results. Average time and number of differences



(a) SA_3

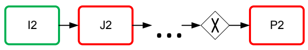


(b) WA_3

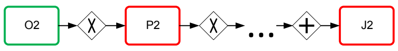
Figure 7.4: Snippet of the process models SA 3 and WA 3

The tool generates a total of 4 statements for explaining this difference, when using PESs. Such differences, both in textual and visual formats, are presented below.

1. *"In model 1, there is a state after the execution of* I2 *where* J2 *precedes* P2*; whereas in model 2, there is a state after the execution of* O2 *where* P2 *precedes* J2*"* (Fig. 7.5),



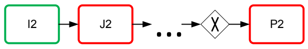
(a) $SA_3{}^1$



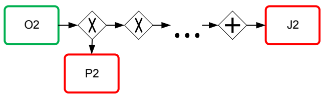
(b) $WA_3{}^2$

Figure 7.5: Difference 1 (J2, P2) between SA 3 and WA 3 using PES

2. *"In model 1, there is a state after the execution of* I2 *where* J2 *precedes* P2*; whereas in model 2, there is a state after the execution of* O2 *where* J2 *and* P2 *are mutually exclusive"* (Fig. 7.6),



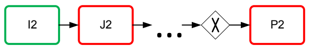
(a) $SA_3{}^1$



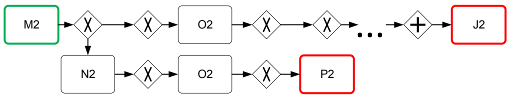
(b) $WA_3{}^3$

Figure 7.6: Difference 2 (J2, P2) between SA 3 and WA 3 using PES

3. *"In model 1, there is a state after the execution of* I2 *where* J2 *precedes* P2*; whereas in model 2, there is a state after the execution of* M2 *where* J2 *and* P2 *are mutually exclusive"* (Fig. 7.7), and



(a) $SA_3{}^1$

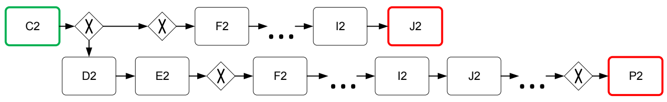


(b) $WA_3{}^4$

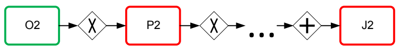
Figure 7.7: Difference 3 (J2, P2) between SA 3 and WA 3 using PES

4. *"In model 1, there is a state after the execution of C2 where* J2 *and* P2 *are mutually exclusive; whereas in model 2, there is a state after the execution of* O2 *where* P2 *precedes* J2*"* (Fig. 7.8).

Conversely, only one statement is produced when using AESs, which would replace the four statements presented above. The statement is:



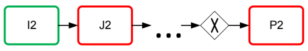
(a) $SA_3{}^2$



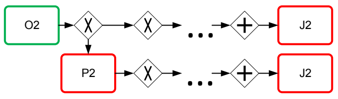
(b) $WA_3{}^2$

Figure 7.8: Difference 4 (J2, P2) between SA 3 and WA 3 using PES

"In model 1, there is a state after the execution of I2 *where* J2 *precedes* P2*; whereas in model 2, there is a state after the execution of* O2 *where* P2 *can occur before* J2*, or* P2 *can be skipped"* (Fig. 7.9).



(a) $SA_3{}^1$



(b) $WA_3{}^1$

Figure 7.9: Difference 1 (J2, P2) between SA 3 and WA 3 using AES

This gain in compactness clearly stems from the expressive power introduced by the presence of asymmetric conflict in AESs, which allows the technique to merge some duplicated tasks.

Chapter 8

Conclusions

8.1 Summary of contributions

Comparison of process models has become a basic operation when managing collections of process models. For example, analysts need compare models to identify opportunities for standardization or understand relative performance differences between pairs of variants of a process. Comparison techniques can be roughly divided into those based on structure and those based on behavior. Structure-based comparison techniques consider the process models as graphs and describes the differences between a pair of process models as edit operations (e.g., insert, remove or substitute) over nodes (tasks and gateways of the process models). Although a pair of structurally different process models can represent the equivalent behavior, for instance, they can produce the same set of traces.

This thesis approaches the behavioral comparison of process models, which abstracts away from the structural differences and, instead, focus on the differences of the behavior represented underneath. In this regard, we propose a comparison technique based on event structures and adopt a notion of equivalence in the true concurrency spectrum, i.e., *completed visible-pomset equivalence.*

Event structures represent concurrent processes by means of events, which are occurrences of actions (or tasks in a process model), and behavioral relations representing dependencies between the events, e.g., causal precedence or exclusiveness. Since the introduction of this formalism, different types of event structures have been proposed with a variety of behavioral relations. In this work, we use three different event structures: prime, asymmetric and flow event structures. Nevertheless, the event structure semantics of process models with cycles contains an infinite set of events, since every iteration in a cycle produces new events.

The first contribution of this thesis is an unfolding technique to compute a finite unfolding prefix of a process with cycles that captures all the causal dependencies between the tasks in the process. This finite representation has two main purposes, on the one hand, it is used to define a notion of multiplicity for each task in the process. I.e., it it is determined if a task can be executed more than one time in a computation, or at most once. On the other hand, the unfolding prefix is used to obtain the prime event structure describing the behavior in terms of causality, conflict and concurrency relations.

The second contribution of the thesis is the definition of a so called partial synchronized product. Given a pair of (any type of) event structures, a partial synchronized product is a graph where every node is a partial match between a pair of configuration pomsets (one configuration from each event structure) and every edge between a pair of nodes is an operation over source node to obtain the target node. We define two operations, matching events and hiding event. The former, matching events, aims at representing equivalent events (instances of the same action) that can be executed from the matched execution state in the two event structures; whereas the latter aims at finding the behavioral discrepancies, i.e., events that can be executed in one model and not in the other. This graph can then be used to find optimal behavioral matchings for the maximal configurations, i.e., those requiring the least amount of hide operations to match all the events.

The encountered differences (hiding operations) can then be verbalized as the events that can occur at a given point in one process but not in the other. Nevertheless, this can lead to a large, and potentially redundant, set of differences. Thus, a first approach, and third contribution of the thesis, is to interpret the differences as mismatching behavioral relations in the prime event structures. Nevertheless, more expressive formalisms than prime event structures, e.g., asymmetric and flow event structures, can provide more compact representations that lead to more concrete diagnosis.

The fourth contribution of the thesis is a reduction technique for event structures. I.e., we propose a set of rules to identify sets of events in an asymmetric and a flow event structure that can be replaced with a single event while preserving the behavior. In the case of the reduction techniques, the adopted equivalence notion is history preserving bisimulation. We note that an asymmetric and a flow event structure can be reduced to different minimal representations (i.e., no other reduction operation is possible), thus the canonicity is not ensured. We note that the non-canonicity of the minimal event structures is not due to the proposed reduction techniques, but it seems to be intrinsic to the nature of both types of event structures. In this regard, we put forward a method to compute a canonical folding of an event structure by leveraging canonical graph labeling techniques. Specifically, we define a deterministic order on the folding operations.

Finally, we present an overview, as well as an evaluation, of a tool implementing the proposed techniques, *BP-Diff*. It is a web-based tool that takes pairs of business process models in BPMN format and produces difference diagnostics in the form of textual statements and graphically overlaid on the process models.

8.2 Future work

The contributions of this thesis put into evidence the potential benefits of using event structures as a foundation for behavioral comparison of process models. At the same time, the contributions open up a number of directions for future research:

1. General theory of foldings.

   Following the discussions in Chapter 6, an interesting line of research is to develop a general theory of foldings. In this thesis we presented a folding technique based on elementary foldings, i.e., only a set of events is folded at a time. Although, in some scenarios, an AES or a FES can be a non-elementary folding of another event structure where, in order to keep the behavior unaltered, one needs to merge different sets of events at once. By the same token, a deeper research on the folding of FESs would ensure the completeness for elementary and non-elementary foldings.

2. Transformation from AES to FES.

   We noted that the conditions defining sets of combinable events are essentially orthogonal for AESs and FESs. In this respect, we envision a transformation from AESs to FESs which would allow further folding at the price of inserting unobservable events to simulate asymmetric conflict on a FES. We contend that such a transformation would open the possibility of taking advantage of the combined expressiveness of AES and FES, possibly leading to more compact representations.

3. Application of model comparison techniques in process mining.

   Another context where the comparison technique can be applied is that of compliance checking. In this regard, the comparison technique can detect and explain where the actual behavior (e.g., based on a mined process model) of a process differs from the one originally modeled.

4. (Semi-) Automatic consolidation of multiple process variants.

   The current comparison technique aims at providing textual and graphical description of the behavioral differences between a pair of process models. The analysts comparing the process models are then responsible to make the appropriate changes for the consolidation of a pair of variants. In this regard, a natural extension of the proposed comparison technique, is the automatic or semi-automatic consolidation of variants. For example, given a pair of process models, one can detect the behavioral differences and then the analyst can decide what is the behavior to keep in the consolidated process model.

5. Automatic propagation of changes across variants.

   Another extension is the automatic propagation of changes across variants. The idea is to compute the blueprint (common behavior) of a set of variants and monitor the changes performed over the corresponding models, such that every modification affecting the blueprint is propagated to all the variants.

6. Automated business process discovery based on event structures.

   In process mining, a widely studied concept is that of process discovery. Specifically, given a set of execution logs, it aims at constructing the processes represented in the logs. Then, a process discovery technique based on event structures can have several benefits, first, it would allow the explicit representation of behavioral relations, such as concurrency, causality and conflict in the case of PES. Second, the reduction techniques presented in this thesis would allow the construction of a more concrete representation of the discovered process.

It can ease the analysis of the represented behavior while keeping unaltered the discovered behavior [Bees 15].

7. Conformance checking based on event structures.

   Another path for future research in the context of process mining is conformance checking. Conformance checking aims at verifying if the behavior recorded in an execution log conforms to the corresponding process model, and vice-versa. The comparison technique presented in this thesis can be used to detect deviant and common behavior between an event structure (PES, AES or FES) extracted from the log, as considered in previous point, and the event structure obtained from the model.

References

[Aals 00] Wil van der Aalst. **Workflow Verification: Finding Control-Flow Errors Using Petri-Net-Based Techniques**. In: *BPM*, pp. 161–183, Springer, 2000. 48

[Aals 03] W. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. **Workflow Patterns**. In: *Distributed and Parallel Databases*, pp. 5–51, 2003. 23

[Adri 13] Arya Adriansyah, Boudewijn F van Dongen, and Wil MP van der Aalst. **Memory-efficient alignment of observed and modeled behavior**. *BPMcenter. org, Tech. Rep*, 2013. 110

[Ait 09] Ali Ait-Bachir, Marlon Dumas, and Marie-Christine Fauvet. **Detecting Behavioural Incompatibilities between Pairs of Services**. In: George Feuerlicht and Winfried Lamersdorf, editors, *Service-Oriented Computing – ICSOC 2008 Workshops*, pp. 79–90, Springer, 2009. 32

[Arma 14a] Abel Armas-Cervantes, Paolo Baldan, Marlon Dumas, and Luciano García-Bañuelos. **Behavioral Comparison of Process Models Based on Canonically Reduced Event Structures**. In: *Proc. of BPM*, pp. 267–282, Springer, 2014. 27, 28

[Arma 14b] Abel Armas-Cervantes, Paolo Baldan, Marlon Dumas, and Luciano García-Bañuelos. **BP-Diff: A Tool for Behavioral Comparison of Business Process Models**. In: *Proceedings of the BPM Demo Session 2014*, 2014. 28, 159

[Arma 14c] Abel Armas-Cervantes, Paolo Baldan, Marlon Dumas, and Luciano García-Bañuelos. **Diagnosing Behavioral Differ-**

ences Between Business Process Models: An Approach Based on Event Structures. *CoRR*, 2014. 27, 28

[Arma 14d] Abel Armas-Cervantes, Paolo Baldan, and Luciano García-Bañuelos. **Reduction of Event Structures under History Preserving Bisimulation**. *CoRR*, Vol. abs/1403.7181, 2014. 28, 54

[Arma 14e] Abel Armas-Cervantes, Marlon Dumas, Luciano Garcia-Banuelos, and Artem Polyvyanyy. **On the Suitability of Generalized Behavioral Profiles for Process Model Comparison**. 2014. 27

[Bado 12] Eric Badouel. **On the α-Reconstructibility of Workflow Nets**. In: Serge Haddad and Lucia Pomello, editors, *ATPN*, Springer, 2012. 34

[Bald 01] Paolo Baldan, Andrea Corradini, and Ugo Montanari. **Contextual Petri Nets, Asymmetric Event Structures, and Processes**. *Information and Computation 2001*, Vol. 171, pp. 1–49, 2001. 57, 59, 63, 121, 128

[Bees 15] N.R.T.P. van Beest, M. Dumas, L. García-Bañuelos, and M. La Rosa. **Log delta analysis: Interpretable differencing of Business process event logs**. Eprint No. 83018, Queensland University of Technology, 2015. 174

[Best 87] Eike Best. **Structure theory of Petri nets: the free choice hiatus**. In: *Petri Nets: Central Models and Their Properties*, pp. 168–205, Springer, 1987. 45

[Best 91] Eike Best, Raymond Devillers, Astrid Kiehn, and Lucia Pomello. **Concurrent bisimulations in Petri nets**. *Acta Informatica*, Vol. 28, pp. 231–264, 1991. 71, 122

[Boud 89] Gérard Boudol and Ilaria Castellani. **Permutation of transitions: An event structure semantics for CCS and SCCS**. In: *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, School/Workshop*, pp. 411–427, Springer, 1989. 57, 64, 65, 68, 121

[Boud 90] Gérard Boudol. **Flow event structures and flow nets**. In: *Semantics of Systems of Concurrent Processes*, pp. 62–95, Springer, 1990. 48, 66, 77, 78

[Cast 97] Ilaria Castellani and Guo qiang Zhang. **Parallel Product Of Event Structures**. *Theoretical Computer Science*, Vol. 179, pp. 203–215, 1997. 146

[Cayo 13] Ugur Cayoglu, Remco Dijkman, Marlon Dumas, Peter Fettke, Luciano Garcia-Banuelos, Philip Hake, Christopher Klinkmüller, Henrik Leopold, André Ludwig, Peter Loos, et al. **The process model matching contest 2013**. In: *4th International Workshop on Process Model Collections: Management and Reuse (PMC-MR'13)*, 2013. 32

[Clea 91] Rance Cleaveland. **On automatically explaining bisimulation inequivalence**. In: *CAV*, pp. 364–372, Springer, 1991. 33

[Dech 85] Rina Dechter and Judea Pearl. **Generalized Best-first Search Strategies and the Optimality of A***. *J. ACM*, Vol. 32, pp. 505–536, 1985. 111

[Dese 95] Jörg Desel and Javier Esparza. *Free Choice Petri Nets*. Cambridge University Press, New York, NY, USA, 1995. 45

[Dijk 08a] Remco Dijkman. **Diagnosing Differences between Business Process Models**. In: *BPM*, pp. 261–277, Springer, 2008. 33

[Dijk 08b] Remco Dijkman, Marlon Dumas, and Chun Ouyang. **Semantics and analysis of business process models in BPMN**. *Information and Software Technology*, Vol. 50, No. 12, pp. 1281–1294, 2008. 24

[Dijk 09a] Remco Dijkman, Marlon Dumas, and Luciano García-Bañuelos. **Graph Matching Algorithms for Business Process Model Similarity Search**. In: *BPM 2009*, pp. 48–63, Springer, 2009. 32

[Dijk 09b] Remco Dijkman, Marlon Dumas, Luciano García-Bañuelos, and Reina Kaarik. **Aligning business process models**. In: *Enterprise Distributed Object Computing Conference, 2009. EDOC'09. IEEE International*, pp. 45–53, IEEE, 2009. 32

[Dijk 11] Remco Dijkman, Marlon Dumas, Boudewijn van Dongen, Reina Käärik, and Jan Mendling. **Similarity of business process models: Metrics and evaluation**. *Inf. Sys.*, Vol. 36, No. 2, pp. 498–516, 2011. 21, 31, 32

[Duma 09] Marlon Dumas, Luciano García-Bañuelos, and Remco Dijkman. **Similarity Search of Business Process Models**. *IEEE Data Engineering Bulletin*, Vol. 32, No. 3, pp. 23–28, 2009. 29, 30, 31

[Duma 13] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. *Fundamentals of Business Process Management.* Springer, 2013. 19

[Ehri 07] Marc Ehrig, Agnes Koschmider, and Andreas Oberweis. **Measuring similarity between semantic business process models**. In: John F. Roddick and Annika Hinze, editors, *Proceedings of the fourth Asia-Pacific conference on Conceptual modelling*, pp. 71–80, Australian Computer Society, Inc., 2007. 31

[Enge 91] Joost Engelfriet. **Branching processes of Petri nets**. *Acta Informatica*, Vol. 28, pp. 575–591, 1991. 24, 51

[Espa 02] Javier Esparza, Stefan Römer, and Walter Vogler. **An Improvement of McMillan's Unfolding Algorithm**. *Formal Methods in System Design*, Vol. 30, No. 2, pp. 285–310, 2002. 12, 94, 95

[Espa 08] J. Esparza and K. Heljanko. *Unfoldings - A Partial order Approach to Model Checking. EACTS Monographs in Theoretical Computer Science*, Springer, 2008. 93

[Fahl 09] D. Fahland, C. Favre, B. Jobstmann, J. Koehler, N. Lohmann, H. Völzer, and K. Wolf. **Instantaneous soundness checking of industrial business process models**. In: *BPM*, pp. 278–293, Springer, 2009. 162

[Favr 15] Cédric Favre, Dirk Fahland, and Hagen Völzer. **The relationship between workflow graphs and free-choice workflow nets**. *Information Systems*, Vol. 47, pp. 197–219, 2015. 24

[Glab 01] Rob van Glabbeek and Ursula Goltz. **Refinement of actions and equivalence notions for concurrent systems**. *Acta Informatica*, Vol. 37, pp. 229–327, 2001. 33, 68

[Glab 89] Rob van Glabbeek and Ursula Goltz. **Equivalence Notions for Concurrent Systems and Refinement of Actions.** In: *Mathematical Foundations of Computer Science 1989*, pp. 237–248, Springer, 1989. 33, 68, 70, 71, 103, 122

[Glab 90] Rob J. van Glabbeek. *The linear time-branching time spectrum.* Springer, 1990. 33, 68

[Glab 95] R. J. van Glabbeek and Gordon D. Plotkin. **Configuration Structures**. In: *Proceedings of LICS'95*, pp. 199–209, IEEE Computer Society Press, 1995. 54, 69

[Glab 96] Rob van Glabbeek. **History preserving process graphs, draft**. *Draft available at: http://theory. stanford. edu/˜ rvg/abstracts. html# hppg*, 1996. 54

[Golt 94] U. Goltz and A. Rensink. **Finite Petri nets as models for recursive causal behaviour**. *Theoretical Computer Science*, Vol. 124, pp. 169–179, 1994. 70, 103

[Hart 68] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. **A formal basis for the heuristic determination of minimum cost paths**. *IEEE Transactions on Systems, Science, and Cybernetics*, Vol. SSC-4, No. 2, pp. 100–107, 1968. 111

[Kant 10] G. Kant. **Using Canonical Forms for Isomorphism Reduction in Graph-based Model Checking**. Technical Report, CTIT University of Twente, Enschede, July 2010. 154

[Khom 03] Victor Khomenko, Maciej Koutny, and Walter Vogler. **Canonical prefixes of Petri net unfoldings**. *Acta Informatica*, Vol. 40, No. 2, pp. 95–118, 2003. 93, 94, 95

[Klin 13] Christopher Klinkmüller, Ingo Weber, Jan Mendling, Henrik Leopold, and André Ludwig. **Increasing Recall of Process Model Matching by Improved Activity Label Matching**.

In: Florian Daniel, Jianmin Wang, and Barbara Weber, editors, *Business Process Management*, pp. 211–218, Springer Berlin Heidelberg, 2013. 31

[Kunz 11] Matthias Kunze, Matthias Weidlich, and Mathias Weske. **Behavioral Similarity - A proper metric**. In: Stefanie Rinderle-Ma, Farouk Toumani, and Karsten Wolf, editors, *Proc. of the International Conference on Business Process Management (BPM) 2011*, pp. 166–181, Springer, 2011. 75

[Leop 12] Henrik Leopold, Mathias Niepert, Matthias Weidlich, Jan Mendling, Remco Dijkman, and Heiner Stuckenschmidt. **Probabilistic Optimization of Semantic Process Model Matching**. In: Alistair Barros, Avigdor Gal, and Ekkart Kindler, editors, *Business Process Management*, pp. 319–334, Springer Berlin Heidelberg, 2012. 31

[Leve 66] Vladimir I Levenshtein. **Binary codes capable of correcting deletions, insertions, and reversals**. In: *Soviet physics doklady*, pp. 707–710, 1966. 30

[Lohm 14] Niels Lohmann and Dirk Fahland. **Where Did I Go Wrong? - Explaining Errors in Business Process Models.** In: *BPM*, pp. 283–300, 2014. 118

[Madh 04] Therani Madhusudan, J.Leon Zhao, and Byron Marshall. **A case-based reasoning framework for workflow model management**. *Data & Knowledge Engineering*, Vol. 50, No. 1, pp. 87–115, 2004. 32

[McKa 81] Brendan D McKay. *Practical graph isomorphism.* Department of Computer Science, Vanderbilt University, 1981. 153

[McMi 95] Kenneth L. McMillan and David K. Probst. **A Technique of State Space Search Based on Unfolding**. *Formal Methods in System Design*, Vol. 6, No. 1, pp. 45–65, 1995. 12, 93, 94

[Meln 02] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. **Similarity flooding: a versatile graph matching algorithm and its application to schema matching**. In: *Data Engineering, 2002. Proceedings. 18th International Conference on*, pp. 117–128, 2002. 32

[Mess 95] B. Messmer. **Efficient Graph Matching Algorithms for Preprocessed Model Graphs**. *PhD thesis, University of Bern, Switzerland*, 1995. 31, 32

[Niel 81] Mogens Nielsen, Gordon D. Plotkin, and Glynn Winskel. **Petri Nets, Event Structures and Domains, Part I**. *Theoretical Computer Science*, Vol. 13, pp. 85–108, 1981. 24, 46, 51, 57, 58, 91

[Petr 62] Carl Adam Petri. *Kommunikation mit Automaten.* PhD thesis, Darmstadt University of Technology, Germany,, 1962. 24, 37

[Poly 14] Artem Polyvyanyy, Matthias Weidlich, Raffaele Conforti, Marcello La Rosa, and Arthur H. M. ter Hofstede. **The 4C Spectrum of Fundamental Behavioral Relations for Concurrent Systems**. In: *Petri Nets*, pp. 210–232, 2014. 34, 74, 85, 87

[Rabi 88] Alexander M. Rabinovich and Boris A. Trakhtenbrot. **Behaviour structures and nets**. *Fundamenta Informatica*, Vol. 11, pp. 357–404, 1988. 71, 122

[Rens 92] A. Rensink. **Posets for Configurations!** In: W. R. Cleaveland, editor, *Proceedings of CONCUR'92*, pp. 269–285, Springer, 1992. 54

[Rosa 10] Marcello La Rosa, Stephan Clemens, Arthur H. M. ter Hofstede, and Nick Russell. **Appendix A. The Order Fulfillment Process Model**. In: *Modern Business Process Automation*, Springer, 2010. 22

[Soko 06] Oleg Sokolsky, Sampath Kannan, and Insup Lee. **Simulation-Based Graph Similarity**. In: *TACAS*, pp. 426–440, Springer, 2006. 33

[van 04] Wil M. P. van der Aalst, Ton Weijters, and Laura Maruster. **Workflow mining: discovering process models from event logs**. *IEEE TKDE*, Vol. 16, No. 9, pp. 1128–1142, 2004. 34

[van 08] Boudewijn van Dongen, Remco Dijkman, and Jan Mendling. **Measuring Similarity between Business Process Models**. In: *CAiSE*, pp. 450–464, Springer, 2008. 30

[van 97] Wil M. P. van der Aalst. **Verification of workflow nets**. In: *ICATPN*, pp. 407–426, Springer, 1997. 47, 100

[van 98] Wil M. P. van der Aalst. **The Application of Petri nets to Workflow Management**. *The Journal of Circuits, Systems and Computers*, Vol. 8, No. 1, pp. 21–66, 1998. 24

[Weid 10] Matthias Weidlich, Remco Dijkman, and Jan Mendling. **The ICoP Framework: Identification of Correspondences between Process Models**. In: *Advanced Information Systems Engineering*, pp. 483–498, Springer Berlin Heidelberg, 2010. 31

[Weid 11a] Matthias Weidlich, Felix Elliger, and Mathias Weske. **Generalised Computation of Behavioural Profiles Based on Petri-Net Unfoldings**. In: Mario Bravetti and Tevfik Bultan, editors, *Web Services and Formal Methods*, pp. 101–115, Springer, 2011. 75

[Weid 11b] Matthias Weidlich, Jan Mendling, and Mathias Weske. **Efficient Consistency Measurement Based on Behavioral Profiles of Process Models**. *IEEE TSE*, Vol. 37, No. 3, pp. 410–429, 2011. 34, 74, 75, 80

[Weid 11c] Matthias Weidlich, Artem Polyvyanyy, Jan Mendling, and Mathias Weske. **Causal Behavioural Profiles**. *Fundamenta Informaticae*, Vol. 113, No. 3-4, pp. 399–435, 2011. 34, 74, 85

[Weid 12] Matthias Weidlich and Jan van der Werf. **On Profiles and Footprints-Relational Semantics for Petri Nets**. In: *ATPN*, pp. 148–167, Springer, 2012. 34

[Wins 87] Glynn Winskel. **Event structures**. In: W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Applications and Relationships to Other Models of Concurrency*, pp. 325–392, Springer, 1987. 57, 58, 91

[Yan 12] Zhiqiang Yan, Remco Dijkman, and Paul Grefen. **Fast business process similarity search**. *Distributed and Parallel Databases*, Vol. 30, pp. 105–144, 2012. 32

Appendix A

Basic notions and notations

A.1 Sets and numbers

Let X and Y be sets.

$\lvert X\rvert$	cardinality of X
$X \setminus Y$	elements in X and not in Y
$X \subseteq Y$	X is subset of Y, including the case $X = Y$
$X \subset Y$	X is a proper subset of Y, i.e., $X \subseteq Y$ and $X \neq Y$

$\mathbb{N}$ denotes the set of natural numbers including 0.

A.2 Sequences

Let X be a set. A finite sequence is a mapping $\{1, \dots, n\} \to X$ or $0 \to X$.

$0 \to X$	empty sequence
$\sigma : \{1, \dots, n\} \to X$	sequence of n elements in X. σ is represented as a string $x_1\ x_2\ x_3 \dots x_n$, where $x_i = \sigma(i)$ for $1 \leq i \leq n$
length of σ (n or 0)	the length of a sequence σ is n if $\sigma = x_1\ x_2\ x_3 \dots x_n$, or 0 if σ is empty

A.3 Relations

Let X, Y be sets. Let $r \subseteq X \times Y$ be a binary relation.

r^+	irreflexive transitive closure of r
r^*	reflexive transitive closure of r
r is partial order	r is reflexive, antisymmetric and transitive
$r\vert_{(W \times Z)}$	restriction of r to $W \times Z$, where $W \subseteq X, Z \subseteq Y$. I.e., $r\vert_{W \times Z} = r \cap (W \times Z)$. By abuse of notation, if $W = Z$ and $X = Y$, the restriction $r\vert_{W^2}$ is denoted simply as $r\vert_W$.
r is *well-founded*	r has no infinite descending chain, i.e., a sequence $\langle e_i \rangle_{i \in \mathbb{N}}$, where $e_i \in X \cup Y$, such that $e_{i+1}\ r\ e_i$, $e_i \neq e_{i+1}$, for all $i \in \mathbb{N}$
r is *acyclic*	r has no "cycles", that is, $e_0\ r\ e_1\ r \ldots r\ e_n\ r\ e_0$ with $e_i \in X \cup Y$, does not hold. In particular, if r is well-founded, then it has no (non-trivial) cycles

A.4 Functions

Let X, Y, U, V be sets. A function $f : X \to Y$ is a relation $f \subseteq X \times Y$. Consider another function $g : U \to V$ and a subset of elements $X' \subseteq X$.

$f \circ g$	composition of functions f and g, i.e., for an element $x \in X$, then $(f \circ g)(x) = f(g(x))$
$f(X')$	mapping of the elements in X', i.e., $\{f(x) \mid x \in X'\}$
$f[x \mapsto y] : X \cup \{x\} \to Y \cup \{y\}$	function defined by $f[x \mapsto y](x) = y$ and $f[x \mapsto y](z) = f(z)$ for $z \in X \setminus \{x\}$. Similarly, it represents an update of f, when $x \in X$, or an extension of its domain, otherwise

Kokkuvõte (Summary in Estonian)

Äriprotsesside käitumuslike erinevuste diagnoosimine

Mitmel turul või turusegmendil tegutsevatel ettevõtetel tuleb tihti hallata sama äriprotsessi mitut varianti. Selline variantide paljusus võib olla tingitud eristuvatest toodetest, erinevatest klienditüüpidest, erinevatest eeskirjadest riikides, kus ettevõte tegutseb, või erinevatest aja jooksul tehtud valikutest mitmetes äriüksustes. Nende protsesside jätkuva haldamise käigus peavad analüütikud võrdlema mitme protsessivariandi mudeleid selleks, et tuvastada standardiseerimisevõimalusi või aru saada suhtelistest jõudluserinevustest eri variantide vahel.

Olemasolevaid lähenemisi protsessimudelite võrdlemiseks saab laias laastus jaotada struktuursel sarnasusel põhinevateks ja käitumuslikul sarnasusel põhinevateks. Struktuursel sarnasusel põhinevad lähenemised oskavad hästi seletada kindlaid protsessimudelite paaride vahelisi erinevusi nagu ülesannete lisamine, kustutamine või asendamine või lihtsad tippude ümberpaigutamised (nt. kahe ülesande vahetamine teineteisega). Siiski, kaks varianti võivad küll olla süntaktiliselt erinevad kuid käitumuslikult ekvivalentsed. Vastupidi, nad võivad olla süntaktiliselt sarnased kuid käitumuslikult väga erinevad kuna muutused üksikutes lüüsides või servades võivad kaasa tuua olulisi käitumuslikke erinevusi.

Selles kontekstis uurib see väitekiri äriprotsessimudelite paaride vaheliste käitumuslike erinevuste diagnoosimist võttes aluseks samaaegsust arvestava ekvivalentsi määratluse. Selles väitekirjas pakutakse välja meetod, mis kahe protsessimudeli puhul teeb kindlaks, kas nad on käitumuslikult ekvivalentsed või juhul kui nad seda pole, kirjeldab nendevahelisi erinevusi ühes mudelis leiduvate ja teises puuduvate käitumuslike seoste abil. Pakutud lahenduse aluseks on protsessimudelite teisendus sündmuste struktuuridesse, täpsemalt asümmeetrilistesse sündmuste ja voosündmuste struktuuridesse. Selle teisenduse naiivne versioon kannatab kahe piirangu all. Esiteks tekitab see üleliigseid erinevuste diagnostilisi lauseid kuna sündmuste struktuur võib sisaldada ebavajalikku sündmuste dubleerimist. Teiseks ei ole see teisendus kasutatav tsükleid sisaldavate protsessimudelite korral. Esimese piirangu ületamiseks pakub väitekiri võtte sündmuste struktuuris sündmuste duplitseerimise vähendamiseks säilitades kanoonilisuse rakendades hulka käitumist säilitavaid sündmuste voltimise reegleid. Teise piirangu tarvis pakub väitekiri iga tsüklis esineva sündmuse kõiki võimalikke põhjusi katva voltimise määratluse. Sellest voltimisest on võimalik tuletada sündmuste struktuur kus korduvad sündmused on eristatud mittekorduvatest, millega võimaldatakse käitumuslike erinevuste diagnoosimine ühes mudelis esinevate ja teises puuduvate korduvuste ja põhjuslike seoste kaudu.

Töös kirjeldatud meetod on teostatud prototüübina, mis võtab sisendiks äriprotsessimudelid Business Process Model and Notation (BPMN) kujul ja loob erinevuste kirjelduse loomulikus keeles. Erinevusi saab ka graafiliselt äriprotsesside kohal näidata.

Curriculum vitae

General

Name:	Abel Armas Cervantes
Date and place of Birth:	26.07.1986, Mexico
Citizenship:	Mexican

Education

2009 – 2011	University of Tartu, Faculty of Mathematics and Computer Science, Master of Science in Software Engineering
2004 – 2009	Universidad Autónoma de Tlaxcala, Facultad de Ciencias Básicas Ingeniería y Tecnología, Bachelor of Computer Engineering

Work experience

01/2010 – 08/2011	STACC, Junior researcher

Elulookirjeldus

Üldandmed

Nimi:	Abel Armas Cervantes
Sünniaeg ja koht:	26.07.1986, Mehhiko
Kodakondsus:	Mehhiko

Haridus

2009 – 2011	Tartu Ülikool, Matemaatika-informaatikateaduskond, magistriõpe, Eriala: tarkvaratehnika
2004 – 2009	Universidad Autónoma de Tlaxcala, Facultad de Ciencias Básicas Ingeniería y Tecnología, bakalaureuseõpe, Eriala: arvutiteadus

Teenistuskäik

01/2010 – 08/2011	STACC, nooremteadur

19. **Helger Lipmaa.** Secure and efficient time-stamping systems. Tartu, 1999, 56 p.
20. **Jüri Lember.** Consistency of empirical k-centres. Tartu, 1999, 148 p.
21. **Ella Puman.** Optimization of plastic conical shells. Tartu, 2000, 102 p.
22. **Kaili Müürisep.** Eesti keele arvutigrammatika: süntaks. Tartu, 2000, 107 lk.
23. **Varmo Vene.** Categorical programming with inductive and coinductive types. Tartu, 2000, 116 p.
24. **Olga Sokratova.** Ω-rings, their flat and projective acts with some applications. Tartu, 2000, 120 p.
25. **Maria Zeltser.** Investigation of double sequence spaces by soft and hard analitical methods. Tartu, 2001, 154 p.
26. **Ernst Tungel.** Optimization of plastic spherical shells. Tartu, 2001, 90 p.
27. **Tiina Puolakainen.** Eesti keele arvutigrammatika: morfoloogiline ühestamine. Tartu, 2001, 138 p.
28. **Rainis Haller.** *M(r,s)*-inequalities. Tartu, 2002, 78 p.
29. **Jan Villemson.** Size-efficient interval time stamps. Tartu, 2002, 82 p.
30. **Eno Tõnisson.** Solving of expession manipulation exercises in computer algebra systems. Tartu, 2002, 92 p.
31. **Mart Abel.** Structure of Gelfand-Mazur algebras. Tartu, 2003. 94 p.
32. **Vladimir Kuchmei.** Affine completeness of some ockham algebras. Tartu, 2003. 100 p.
33. **Olga Dunajeva.** Asymptotic matrix methods in statistical inference problems. Tartu 2003. 78 p.
34. **Mare Tarang.** Stability of the spline collocation method for volterra integro-differential equations. Tartu 2004. 90 p.
35. **Tatjana Nahtman.** Permutation invariance and reparameterizations in linear models. Tartu 2004. 91 p.
36. **Märt Möls.** Linear mixed models with equivalent predictors. Tartu 2004. 70 p.
37. **Kristiina Hakk.** Approximation methods for weakly singular integral equations with discontinuous coefficients. Tartu 2004, 137 p.
38. **Meelis Käärik.** Fitting sets to probability distributions. Tartu 2005, 90 p.
39. **Inga Parts.** Piecewise polynomial collocation methods for solving weakly singular integro-differential equations. Tartu 2005, 140 p.
40. **Natalia Saealle.** Convergence and summability with speed of functional series. Tartu 2005, 91 p.
41. **Tanel Kaart.** The reliability of linear mixed models in genetic studies. Tartu 2006, 124 p.
42. **Kadre Torn.** Shear and bending response of inelastic structures to dynamic load. Tartu 2006, 142 p.

43. **Kristel Mikkor.** Uniform factorisation for compact subsets of Banach spaces of operators. Tartu 2006, 72 p.
44. **Darja Saveljeva.** Quadratic and cubic spline collocation for Volterra integral equations. Tartu 2006, 117 p.
45. **Kristo Heero.** Path planning and learning strategies for mobile robots in dynamic partially unknown environments. Tartu 2006, 123 p.
46. **Annely Mürk.** Optimization of inelastic plates with cracks. Tartu 2006. 137 p.
47. **Annemai Raidjõe.** Sequence spaces defined by modulus functions and superposition operators. Tartu 2006, 97 p.
48. **Olga Panova.** Real Gelfand-Mazur algebras. Tartu 2006, 82 p.
49. **Härmel Nestra.** Iteratively defined transfinite trace semantics and program slicing with respect to them. Tartu 2006, 116 p.
50. **Margus Pihlak.** Approximation of multivariate distribution functions. Tartu 2007, 82 p.
51. **Ene Käärik.** Handling dropouts in repeated measurements using copulas. Tartu 2007, 99 p.
52. **Artur Sepp.** Affine models in mathematical finance: an analytical approach. Tartu 2007, 147 p.
53. **Marina Issakova.** Solving of linear equations, linear inequalities and systems of linear equations in interactive learning environment. Tartu 2007, 170 p.
54. **Kaja Sõstra.** Restriction estimator for domains. Tartu 2007, 104 p.
55. **Kaarel Kaljurand.** Attempto controlled English as a Semantic Web language. Tartu 2007, 162 p.
56. **Mart Anton.** Mechanical modeling of IPMC actuators at large deformations. Tartu 2008, 123 p.
57. **Evely Leetma.** Solution of smoothing problems with obstacles. Tartu 2009, 81 p.
58. **Ants Kaasik.** Estimating ruin probabilities in the Cramér-Lundberg model with heavy-tailed claims. Tartu 2009, 139 p.
59. **Reimo Palm.** Numerical Comparison of Regularization Algorithms for Solving Ill-Posed Problems. Tartu 2010, 105 p.
60. **Indrek Zolk.** The commuting bounded approximation property of Banach spaces. Tartu 2010, 107 p.
61. **Jüri Reimand.** Functional analysis of gene lists, networks and regulatory systems. Tartu 2010, 153 p.
62. **Ahti Peder.** Superpositional Graphs and Finding the Description of Structure by Counting Method. Tartu 2010, 87 p.
63. **Marek Kolk.** Piecewise Polynomial Collocation for Volterra Integral Equations with Singularities. Tartu 2010, 134 p.