Verification Based on Unfoldings of Petri Nets with Read Arcs

Thèse de doctorat

présentée par

César Rodríguez

à

l'École Normale Supérieure de Cachan

en vue de l'obtention du grade de

Docteur en Informatique

et soutenue à Cachan le 12 décembre 2013 devant le jury composé de :

Keijo Heljanko	Stefan Haar	Fabrice Kordon	Stefan Schwoon
Maciej Koutny	Jiri Srba		
Rapporteurs	Examinateurs	Président	Directeur de thèse

Laboratoire Spécification et Vérification (LSV) ; École Normale Supérieure de Cachan, CNRS & INRIA
61, avenue du Président Wilson ; 94235 CACHAN Cedex, France

ABSTRACT

Humans make mistakes, especially when faced to complex tasks, such as the construction of modern hardware or software. This thesis focuses on machine-assisted techniques to guarantee that computers behave correctly.

Modern computer systems are large and complex. Automated formal verification stands as an alternative to testing or simulation to ensuring their reliability. It essentially proposes to employ computers to exhaustively check the system behavior. Unfortunately, automated verification suffers from the state-space explosion problem: even relatively small systems can reach a huge number of states. Using the right representation for the system behavior seems to be a key step to tackle the inherent complexity of the problems that automated verification solves.

The verification of concurrent systems poses additional issues, as their analysis requires to evaluate, conceptually, all possible execution orders of their concurrent actions. Petri net unfoldings are a well-established verification technique for concurrent systems. They represent behavior by partial orders, which not only is natural but also efficient for automatic verification.

This dissertation focuses on the verification of concurrent systems, employing Petri nets to formalize them, and studies two prominent verification techniques: model checking and fault diagnosis.

We investigate the unfoldings of Petri nets extended with read arcs. The unfoldings of these so-called contextual nets seem to be a better representation for systems exhibiting concurrent read access to shared resources: they can be exponentially smaller than conventional unfoldings on these cases.

Theoretical and practical contributions are made. We first study the construction of contextual unfoldings, introducing algorithms and data structures that enable their efficient computation. We integrate contextual unfoldings with merged processes, another representation of concurrent behavior that alleviates the explosion caused by non-determinism. The resulting structure, called contextual merged processes, is often orders of magnitude smaller than unfoldings, as we experimentally demonstrate.

Next, we develop verification techniques based on unfoldings. We define SAT encodings for the reachability problem in contextual unfoldings, thus solving the problem of detecting cycles of asymmetric conflict. Also, an unfolding-based decision procedure for fault diagnosis under fairness constraints is presented, in this case only for conventional unfoldings.

Finally, we implement our verification algorithms, aiming at producing a competitive model checker intended to handle realistic benchmarks. We subsequently evaluate our methods over a standard set of benchmarks and compare them with existing unfolding-based techniques. The experiments demonstrate that reachability checking based on contextual unfoldings outperforms existing techniques on a wide number of cases.

This suggests that contextual unfoldings, and asymmetric event structures in general, have a rightful place in research on concurrency, also from an efficiency point of view.

RÉSUMÉ

L'être humain fait des erreurs, en particulier dans la réalisation de taches complexes comme la construction des systèmes informatiques modernes. Nous nous intéresserons dans cette thèse à la vérification assistée par ordinateur du bon fonctionnement des systèmes informatiques.

Les systèmes informatiques actuels sont de grande complexité. Afin de garantir leur fiabilité, la vérification automatique est une alternative au *testing* et à la simulation. Elle propose d'utiliser des ordinateurs pour explorer exhaustivement l'ensemble des états du système, ce qui est problématique: même des systèmes assez simples peuvent atteindre un grand nombre d'états. L'utilisation des bonnes représentations des espaces d'états est essentielle pour surmonter la complexité des problèmes posés en vérification automatique.

La vérification des systèmes concurrents amène des difficultés additionnelles, car l'analyse doit, en principe, examiner tous les ordres possibles d'exécution des actions concurrentes. Le dépliage des réseaux de Petri est une technique largement étudiée pour la vérification des systèmes concurrents. Il représentent l'espace d'états du système par un ordre partiel, ce qui se révèle aussi naturel qu'efficace pour la vérification automatique.

Nous nous intéressons à la vérification des systèmes concurrents modélisés par des réseaux de Petri, en étudiant deux techniques remarquables de vérification: le *model checking* et le diagnostic.

Nous étudions les dépliages des réseaux de Petri étendus avec des arcs de lecture. Ces dépliages, aussi appelés dépliages contextuels, semblent être une meilleure représentation des systèmes contenant des actions concurrentes qui lisent des ressources partagées : ils peuvent être exponentiellement plus compacts dans ces cas.

Ce travail contient des contributions théoriques et pratiques. Dans un premier temps, nous étudions la construction des dépliages contextuels, en proposant des algorithmes et des structures de données pour leur construction efficace. Nous combinons les dépliages contextuels avec les *merged process*, une autre représentation des systèmes concurrents qui contourne l'explosion d'états dérivée du non-déterminisme. Cette nouvelle structure, appelée *contextual merged process*, est souvent exponentiellement plus compacte, ce que nous montrons expérimentalement.

Ensuite, nous nous intéressons à la vérification à l'aide des dépliages contextuels. Nous traduisons vers SAT le problème d'atteignabilité des dépliages contextuels, en abordant les problèmes issus des cycles de conflit asymétrique. Nous introduisons également une méthode de diagnostic avec des hypothèses d'équité, cette fois pour des dépliages ordinaires.

Enfin, nous implémentons ces algorithmes dans le but de produire un outil de vérification compétitif et robuste. L'évaluation de nos méthodes sur un ensemble d'exemples standards, et leur comparaison avec des techniques issues des dépliages ordinaires, montrent que la vérification avec des dépliages contextuels est plus efficace que les techniques existantes dans de nombreux cas.

Ceci suggère que les dépliages contextuels, et les structures d'évènements asymétriques en général, méritent une place légitime dans la recherche en concurrence, également du point de vu de leur efficacité.

ACKNOWLEDGMENTS

I am indebted with Stefan Schwoon, who kindly accepted to be my advisor during both my Ph.D and Master thesis. For his generous time and dedication, for the many insights about unfoldings I learnt from him, and for the patience, support, enthusiasm, freedom, and pedagogy he applied in his guidance, I express to him my deepest gratitude. I extend this acknowledgement to Stefan Haar, who either as the *garant* of this thesis or as a co-author, played an essential rôle in making this research possible, proposing interesting questions and offering help whenever it was necessary.

I sincerely thank the reviewers and jury members: Keijo Heljanko and Maciej Koutny for accepting reviewing this manuscript and for their comments on it; and Fabrice Kordon and Jiri Srba for the time they have kindly employed for me.

ENS de Cachan, Fundación Caja Madrid, and INRIA generously provided the necessary financial support, and they are gratefully acknowledged.

I am particularly thankful to Paolo Baldan and Victor Khomenko, who always found time for replying mail and collaborated in many ways with me. Working with you was a pleasure. Victor hosted me during a visit to Newcastle University in February 2013, this was a very profitable time and I heartedly thank him. I am also thankful to Thomas Chatain for many discussions about research or teaching, and Javier Esparza for some inspiring conversations.

My interest in formal methods originates from previous work in the *Space Research Group* (SRG) at University of Alcalá (Spain). There, I had the luck of collaborating in the actual development of embedded software for the satellite Nanosat 1B, now in orbit. I warmly thank my advisor at SRG, Óscar R. Polo, for the freedom and support he offered me to pursue my own goals. I also thank Paul Gastin for having wisely guided my first steps in formal verification; Ocan Sankur and Aiswarya Cyriac were also important persons those days.

Many people from LSV (and LSV groupies) contributed to making these last three years a wonderful experience. I want to thank Hernán Ponce de León for helping to cope with the rainy weather of Paris… and for many discussions about unfoldings. Mahsa Shirmohammadi, Vincent Cheval, and Sinem Kavak always managed to propose a gathering around some eastern delights, they will be much missed. Guillaume Scerri kindly revised fragments of this document. I also thank Benoît Barbot for the many good moments we shared while teaching the course on network programming; Benedikt Bollig and Erwing Fang for all those coffee talks; Christoph Haase for his good advices when applying for my post-doc; and the administrative and systems team for making my life so much easier.

Outside LSV, I was very lucky to count on many friends to make these years enriching and worth remembering. They know who they are, and I thank all of them. Here it goes an incomplete list: Etienne Boisseau, Tong Bu, Normann Decker, Estíbaliz Fraca, Aina Frau, He Huang, Lue Huang, Jakub Kallas, Yusuke Kawamoto, Polyvios Kosmatos, Feifei Liang, Jiagui Liu, Sergey Lozenko, Sosuke Mizusaki, David Montoya, Elie Rached, Martín Rais, Zhongwei Tang, Mario Vega, Yao Wang, and Tibor Zavadil. I also thank CROUS for providing food and housing.

My warmest appreciation goes to my parents, Esteban and Esperanza, my sister Tamara, my uncles Evariste, Vitoria, Dori and Henri, and all my cousins for being a constant source of encouragement, inspiration, and love all across this period.

Finally, my love and gratitude goes to Xiaoju, because of all the uncountably many ways in which you supported me during these three years, and for remembering me every day how wonderful life is. 谢谢。

Oxford, November 25, 2013
César Rodríguez

CONTENTS

1 INTRODUCTION 1
- 1.1 Automated Verification of Concurrent Systems . . . 3
- 1.2 Petri Nets and Non-Sequential Semantics . . . 4
- 1.3 Unfoldings for Verification . . . 6
- 1.4 Contextual Net Unfoldings . . . 8
- 1.5 Contributions and Outline . . . 11
- 1.6 Publications . . . 13

2 PRELIMINARIES 15
- 2.1 Orders and Multisets . . . 15
- 2.2 Contextual Nets . . . 16
- 2.3 Encoding Contextual Nets into Ordinary Petri Nets . . . 20
  - 2.3.1 Plain Encoding . . . 20
  - 2.3.2 Place-Replication Encoding . . . 21
- 2.4 Unfoldings . . . 21
  - 2.4.1 Occurrence Nets . . . 22
  - 2.4.2 Branching Processes . . . 26

3 CONTEXTUAL UNFOLDING CONSTRUCTION 29
- 3.1 Introduction . . . 29
- 3.2 Pruning the Unfolding . . . 31
- 3.3 Adequate Strategies and Completeness . . . 36
- 3.4 Unfolding Procedure . . . 38
- 3.5 Possible Extensions . . . 40
- 3.6 A Concurrency Relation for Efficient Construction . . . 41
- 3.7 Characterizing Possible Extensions Concurrency . . . 43
  - 3.7.1 Lazy Approach . . . 44
  - 3.7.2 Eager Approach . . . 46
- 3.8 Updating the Concurrency Relation . . . 48
- 3.9 Unique Possible Extensions . . . 50
  - 3.9.1 Lazy Approach: Subsumption . . . 51
  - 3.9.2 Eager Approach: Asymmetric Concurrency . . . 54
- 3.10 Discussion: Lazy vs Eager Approach . . . 55
- 3.11 Memory Usage . . . 56
- 3.12 Conclusion . . . 57

4 REACHABILITY AND DEADLOCK CHECKING 59
- 4.1 Introduction . . . 59
- 4.2 Using c-net Unfoldings for Verification . . . 60
- 4.3 SAT-Encodings of c-net Unfoldings . . . 63
- 4.4 Asymmetric Conflict Loops . . . 64
- 4.5 Encoding Size . . . 67
- 4.6 Reduction of Stubborn Events for Deadlock Checking . . . 67
- 4.7 Additional Simplification . . . 70
- 4.8 Conclusions . . . 70

5 CONTEXTUAL MERGED PROCESSES 73
- 5.1 Introduction . . . 73
- 5.2 Contextual Merged Processes . . . 74
- 5.3 Interplay Between Read-Arcs and Choice . . . 78
- 5.4 Characterizing Reachability . . . 80
- 5.5 Encoding Reachability into SAT . . . 84
  - 5.5.1 Reachability via Arbitrary Runs . . . 84

5.5.2 Reachability via Mp-Configurations . . . 86
5.6 Constructing Complete CMPs . . . 87
5.7 Conclusions . . . 88
6 DIAGNOSIS 89
6.1 Introduction . . . 89
6.2 Basic Notions . . . 91
6.3 *Reveals* and Diagnosis . . . 94
6.3.1 *Reveals* Relations . . . 94
6.3.2 Diagnosis from Partial Observation . . . 95
6.4 A Solution Using Extended Reveals . . . 97
6.4.1 Succinct Explanations . . . 97
6.4.2 Characterizing Weakly Fair Configurations . . . 100
6.5 A Decision Procedure for diagnosis . . . 105
6.5.1 Preparation . . . 105
6.5.2 Constructing the Prefixes . . . 106
6.5.3 Encoding Diagnosis into SAT . . . 107
6.6 Conclusion . . . 108
7 TOOLS AND EXPERIMENTS 111
7.1 Prefix Construction: the Cunf Unfolder . . . 111
7.1.1 The History Graph . . . 112
7.1.2 Possible Extensions . . . 112
7.1.3 Concurrency Relation . . . 113
7.1.4 Splitting the Concurrency Relation . . . 114
7.1.5 Strategies . . . 115
7.2 Prefix Analysis: the Tool Cna . . . 116
7.2.1 Stubborn Event Elimination and Subset Reduction . . . 116
7.2.2 AMO Constraint . . . 117
7.2.3 Acyclicity Checking . . . 117
7.2.4 SAT-solver Settings . . . 118
7.3 Benchmarks . . . 119
7.4 Experiments with Cunf . . . 119
7.5 Experiments with Cna . . . 123
7.6 Experiments with Contextual Merged Processes . . . 125
7.7 A Case Study: Dijkstra's Mutual Exclusion Algorithm . . . 128
8 CONCLUSION AND PERSPECTIVES 131
A THE CUNF TOOL USER'S MANUAL, V1.6 135
A.1 Introduction . . . 135
A.2 Author and Contact . . . 136
A.3 Installation . . . 137
A.3.1 From Precompiled Binaries . . . 137
A.3.2 Compilation and Installation from the Source Code . . . 137
A.4 Getting Started . . . 138
A.4.1 Constructing Unfoldings with Cunf . . . 139
A.4.2 Deadlock and Coverability Analysis with Cna . . . 140
A.4.3 More on Dekker's Algorithm . . . 141
A.4.4 Producing Graphics of C-nets and Unfoldings . . . 141
A.4.5 Finding More Examples . . . 143
A.5 Command-line Syntax . . . 144
A.5.1 Cunf . . . 144
A.5.2 Cna . . . 145
A.6 File Formats . . . 146
A.6.1 The `ll_net` Format . . . 146
A.6.2 Unfolding Formats . . . 146

A.7 Producing Input for Cunf 147
A.7.1 Graphically 147
A.7.2 Programmatically 148

BIBLIOGRAPHY 151

1 INTRODUCTION

Since the advent of affordable digital computers, three decades ago, the complexity of hardware and software devices has grown without pause, and there is no reason to think that this will naturally stop in a near future. As systems grow in complexity, so does the difficulty to construct them, and consequently the likelihood that subtle errors remain undetected in the final product. Our modern society is increasingly dependent on computers, and this tendency is likely to hold for long time. We need, in summary, adequate methods for constructing complex and *reliable* computer-based systems.

Reliability is paramount in *safety-critical* systems, customarily defined as those whose malfunction may result in risks for human life, environmental damage, or substantial economic loss [Kni02]. Well known examples are avionics, nuclear-power plants, oil refineries, traffic control systems, spacecrafts, and medical or military equipment.

Reliability is also important for *non-critical* systems. Any unreliable system often has negative economic consequences for the manufacturer. For instance, a faulty smartphone may substantially decrease the sales and turn the market attention to the competing vendor.

Testing and simulation are two methods to improve reliability. They are very effective at least on the early stages of the development. However, testing or simulation explore *some* of the possible behaviors. They can only show the existence of errors, not the absence thereof. It is often claimed that testing can in the best case achieve systems whose failure rate oscillates around 10^{-4} or 10^{-5} failures per hour of operation [Bis13]. This is insufficient for certain applications. Failure rates for avionics, for instance, are often required to be under 10^{-9}, a figure thousands of times more stringent than what testing seems to offer.

In the last decades, *formal verification* has been suggested as an effective means of improving reliability. Unlike testing, formal verification examines *all* behaviors of the system and compares them to a *formal specification*, which distinguishes the good behaviors from the faulty ones. If the system is declared correct, formal verification has established a mathematically rigorous proof that the system is free from all faults described by the specification. Otherwise, depending on the particular technique used, one obtains some level of evidence that the system contains an error.

Observe that formal verification cannot guarantee *absolute correctness*, it only ensures the system to be as correct as the specification is. It can, however, effectively reveal very subtle errors that may otherwise go undetected if only testing or simulation was used.

Formal verification encompasses a number of methods for proving correctness. Two well-established ones are *theorem proving* [BC04] and *model checking* [CGP99], both of which are machine-assisted. In theorem proving, a number of proof obligations are generated from the specification and the implementation. These are formal statements whose validity entails the correctness of the system. Assisted by the theorem prover, the user constructs the proof of each obligation, either interactively or in a highly automated way, depending on the capabilities of the method used. A shortcoming of theorem proving is that it often requires substantial interaction of the user.

In contrast, model checking is a fully automatic technique. Here, a computer exhaustively explores all possible execution paths that the system could follow, and checks compliance with the specification for each of them. If the search terminates without finding any error, model checking has established a formal argument proving the system correct with respect to the specification. If not, an execution path that falsifies the specification is shown to the user, which often is highly valuable to fix the problem.

Asserting validity of the specification thus requires reasoning about the potentially intractably many possible ways in which the system reacts. Even very small systems contain a huge number of execution paths. The *state-space explosion (SSE)* problem refers to the computational difficulty of performing this analysis automatically, and is one of the main obstacles hindering the adoption of model checking in practice.

For certain applications, even formal verification is unable to provide the required reliability. This is the case, e.g., in ultra-critical systems [PNW12]. Recall that formal methods cannot guarantee absolute reliability. What if the formal specification was wrong? Also, formal verification often makes assumptions that may be hard to guarantee. Software verification, for instance, guarantee that a program is *logically* correct *provided that* the CPU or memory behave as expected, which in some harsh environments such as outer space one cannot really assume.

Complementary techniques have been proposed to mitigate the consequences of faults dynamically, while the system runs. Among them one finds runtime verification [LS09], fault tolerance [Avi67], or fault diagnosis [SSL+95]. The latter is specially useful when limited information of the system execution is available. For instance, the system could be a large telecommunication network with links and nodes that can fail at any moment, where only restricted or even outdated state information of them is known at every time. Fault diagnosis monitors a system in execution and infers which states the system could currently be in, and more specifically, whether a fault may have happened, or will inevitably happen.

Despite its clear benefits, formal verification is still far from being fully integrated into mainstream industrial practice, specially for the development of software. A number of causes have been identified [BBD+96; Par10; Was12], among which we cite:

1. Lack of robust, high-quality tool support [HB96; ADKT11; Was12].
2. Inherent complexity of the verification task hinders scalability to problems of industrial size [ADKT11].
3. Lack of representative benchmarks to drive tool quality and allow realistic evaluation of the proposed methods [HB96; Par10].
4. The growing gap between the concepts, needs, and views of research in formal methods and industrial practice [Gla96; Par10].

Thus foundational work aiming at more efficient algorithms for the automated verification of systems and the development of robust tools supporting these methods address at least two of the causes above.

This dissertation makes foundational and practical contributions to model checking and fault diagnosis. We present algorithms for reducing the internal representation of a system that model checkers or fault diagnosers need to construct during the analysis. Furthermore, we implement our model checking algorithms into the Cunf Toolset, a robust model checker intended to handle realistic examples.

We focus on the verification and diagnosis of *concurrent systems*.

1.1 AUTOMATED VERIFICATION OF CONCURRENT SYSTEMS

Two actions are said to be *concurrent* when they are executed simultaneously. A concurrent system is one which performs concurrent actions.

Verification techniques for concurrent systems need to deal with the additional complexity introduced by concurrency. If two concurrent actions a and b are present, the analysis needs to explore, in principle, the execution paths where a happens before b, or b before a, or even when a and b happen *at the same time*. If n concurrent actions are present, the number of execution paths grows exponentially with n. Concurrency is a well known source of SSE for automated verification techniques, including model checking and fault diagnosis.

In model checking of concurrent systems, existing approaches to mitigate the SSE problem can mainly be classified as *symbolic* or *partial-order* methods. Ordinarily, model checking explicitly generates and explores each possible state that the system under analysis could enter. In contrast, symbolic model checking explores and manipulates *sets* of reachable states represented *implicitly*, or symbolically, e.g., as the solutions of a Boolean formula. Symbolic model checking was introduced by Ken McMillan [BCM+92; McM93a], who proposed to use *binary decision diagrams* (BDDs) [Bry86] as a symbolic, more abstract, representation for sets of states. BDDs seem to exploit very well the regularity of synchronous digital circuits, making symbolic model checking with BDDs a very successful technique for the verification of hardware.

Partial-order methods encompasses two families of techniques of similar nature that are often presented quite differently. On the one hand we have *partial-order reduction* techniques, such as stubborn sets, persistent sets, and ample sets, see [Val98]. They work by classifying all execution paths of the system according to some equivalence relation such that all paths in each equivalence class either satisfy or falsify the property. Then, they explore at least one path on each equivalence class, thus potentially avoiding the exploration of many equivalent paths. In this sense, these techniques can also be described as model checking using representatives [Pel93].

On the other hand we have methods issued from *partial-order semantics* of the modelling language. Here, the system is transformed into a set of events partially ordered by a precedence relation. Concurrent actions are represented as unordered events, and execution paths correspond to precedence-closed sets of events. Importantly, every such set symbolically represents many execution paths, a fact that the model checker uses to avoid individually generating each of them. This manuscript develops model checking methods based on partial-order semantics, we come back to them in the next section.

Similarly, fault diagnosis faces the SSE problem due to concurrency. A prominent approach based on partial-order semantics has been proposed by Benveniste et al. [BFHJ03]. As before, the technique constructs a partial-order representation of the system upon which the reasoning is performed. This thesis contributes with a generalization and extension of this method.

Both model checking and fault diagnosis assume that a model of the system is given as input to the verification procedure. Well-established general models for concurrent systems are *process algebras* [Bae05] and *Petri nets* [Rei13]. Process algebras, or process calculi, are formal languages whose modelling primitives focus on expressing the interaction between the different agents, or processes, that compose the system. A process algebra provides algebraic laws making possible to analyze and transform system mod-

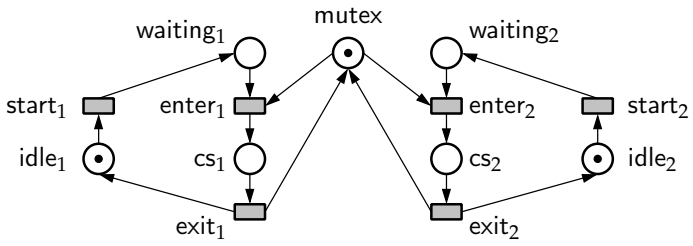


Figure 1: A Petri net modelling a mutual exclusion algorithm.

els using equational reasoning. The three most well-known process algebras are CCS, CSP, and ACP; many variants of them appear in the literature, see [Bae05]. Petri nets are both a graphical and formal notation for modelling systems, and they will be the specification language for the methods developed in this thesis.

1.2 PETRI NETS AND NON-SEQUENTIAL SEMANTICS

Petri nets [Rei13] are a formal notation to represent, manipulate, and reason about concurrent systems. Because they are general models of concurrency, their study is the study of the principles underlying diverse classes of systems. While Petri nets were invented by Carl Adam Petri at the age of 13 [PR08], the first semi-formal notations for them appeared on his thesis [Pet66].

Figure 1 shows the graphical representation of a Petri net. This net models a system with two components that operate in mutual exclusion, as it will be clear shortly. Petri nets are composed of three kinds of elements: *places*, depicted as circles, *transitions*, depicted as boxes, and *arcs*, which can only link places to transitions and vice versa, but never transitions to transitions or places to places. Certain places, idle_1 and idle_2 in our example, are depicted with a dot inside. Such dots are called *tokens*. The *preset* of a transition is the set of places from which there is an arc to the transition; the *postset* is the set of those reached by an arc originating at the transition.

Petri net can be given essentially two kinds of semantics, so called *sequential* and *non-sequential*. A third kind of semantics, so called *step semantics* can basically be derived from non-sequential semantics.

We recall now the sequential semantics of Petri nets. A state of the net, or *marking*, is an assignment of tokens to places. The *initial marking* is the one often depicted with the net: in Fig. 1, it is the marking that assigns one token to idle_1, idle_2, and mutex, and zero to the remaining places. From a marking, one may fire a transition and reach a new marking, provided that the transition is enabled by the first marking. A transition is *enabled* at a marking if the latter assigns at least one token to each place in the preset of the transition. For instance, the initial marking in Fig. 1 enables start_1 and start_2, but not enter_1, as there is no token in $\mathsf{waiting}_1$. Firing any enabled transition consumes one token in places of the preset and produces one token for the places in the postset. In the example, firing start_1 produces a new marking that assigns one token to $\mathsf{waiting}_1$, zero to idle_1, and leaves remaining places as they were on the initial marking. In this new marking, both enter_1, and start_2 are enabled and may be fired, again producing new markings. Any sequence of transitions that can be fired from the initial marking is called a *firing sequence* or *run*, and the last marking it produces is said to be *reachable*.

Finally, the *reachability graph* of the net is a directed, edge-labelled graph whose vertices are the reachable markings and such that there is an edge labelled by a transition t between a marking m and m' if and only if t is enabled at m and, after firing, produces m'. Clearly, the firing sequences of the net correspond to paths on the reachability graph starting from the initial marking. The reachability graph is often referred as the *state space* of the net.

It should now be clear that Fig. 1 represents a mutual exclusion algorithm for two parties, 1 and 2. Initially, any of the parties can execute start_i, reaching a marking where at most one can execute enter_i. This is because both enter_1 and enter_2 consume the token in mutex. So only after the party which manages to enter the critical section executes the corresponding exit transition, putting back the token in the mutex, will the other party be able to enter the critical section.

The literature of Petri nets contains many extensions to this basic formulation. Examples are time Petri nets, coloured nets, nets with inhibitor arcs, *nets with read arcs*, etc. The latter ones will be specially relevant in this work.

Sequential semantics, then, associate every net with a set of firing sequences, or alternatively, a directed graph that represent all reachable markings. A firing sequence imposes a total order on the transition occurrences and, as such, implicitly assumes the existence of a global time that marks the order of occurrence. Non-sequential semantics eliminate this assumption and order transition occurrences, also called *events*, not by their time of occurrence, but by their relative *causal* or *precedence relations*. For instance, any occurrence of the transition $\mathsf{writing}_1$ is always preceded by another one of start_1. In contrast, the two events corresponding to firing start_1 and start_2 after the initial marking are concurrent, causally unrelated, they can happen in any order without one affecting the other.

Replacing the total ordering of time by the relative precedences between transition occurrences induces a partial ordering of events. In it, concurrent events necessarily appear unordered.

Historically, the non-sequential semantics of Petri nets settled in two steps, see [Esp10]. Petri himself defined *non-sequential processes* [Pet77], the notion corresponding to sequential runs in the realm of partial-order semantics. A process represents a partially-ordered run, where precedence-unrelated events may happen in any order. Any linear extension of the precedence order yields a sequential run.

Figure 2 (a), (b), (c), and (d) show four different processes of the net in Fig. 1. Processes are actually represented as *acyclic Petri nets* instead of partial orders of events, but this is only a technical matter. For instance, (b) corresponds to the sequential run start_1, start_2, enter_1, but also to the same run if we invert the first two transitions. Conceptually, (b) represents a concurrent run where both parties intend to enter the critical section, but only the first enters, no matter in which order they interleave their actions.

The second step was to *fuse* the common parts of all processes of the net, into an object called *unfolding*. The unfolding is conceptually equivalent to the computation tree in sequential semantics, it binds together all processes and branches where executions diverge into different *futures*. This notion first appeared, with a different name, in [NPW81].

Figure 2 (e) shows the unfolding of Fig. 1. Actually, the unfolding of Fig. 1 is infinite: since an infinite sequential execution is possible in the net, the unfolding contains an infinite *branch*. So (e) only shows a *prefix* of the unfolding. Observe that all processes shown in Fig. 2 are embedded in this unfolding

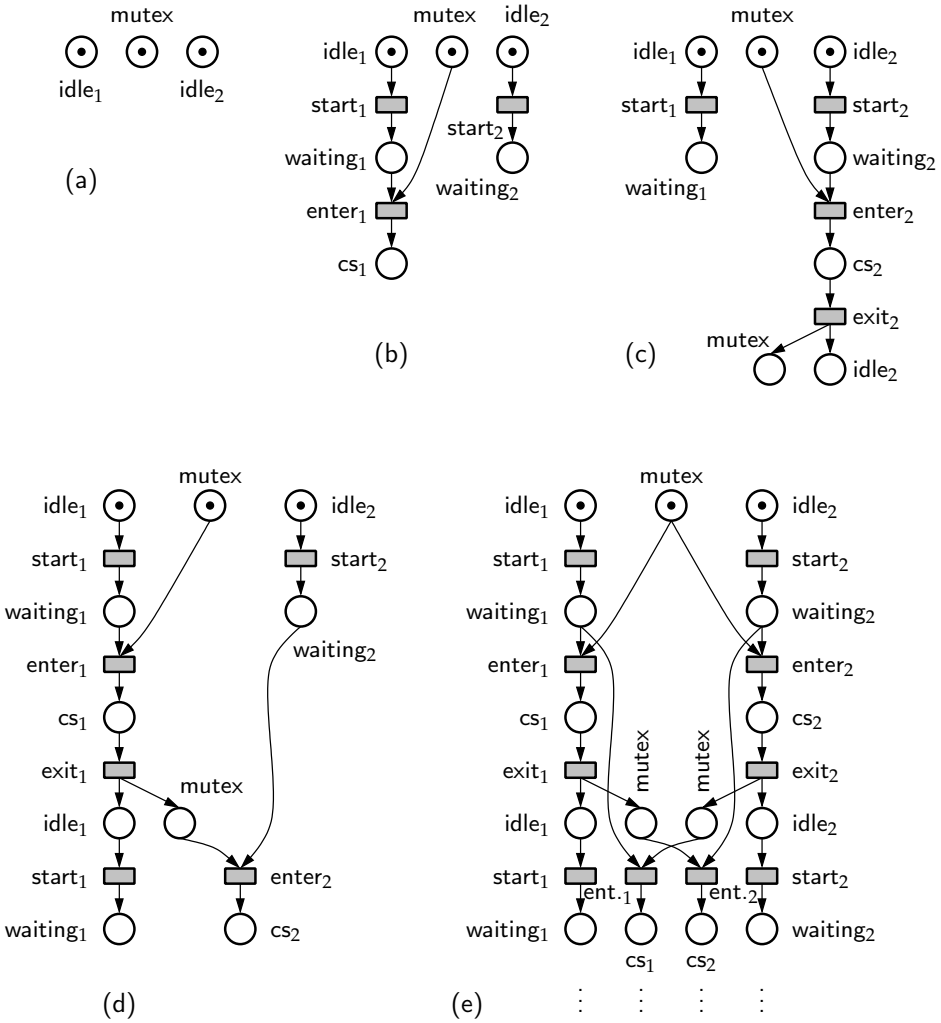


Figure 2: Non-sequential semantics of the Petri net in Fig. 1. In (a), (b), (c), and (d), four different processes are shown. (e) is a prefix of the infinite (thus the bottom dots) *unfolding*, all shown processes are embedded in it.

prefix, and that they share, roughly speaking, the events that are common to multiple processes.

Given a net, many natural questions can be posed, such as: (reachability) is a given marking reachable on the net? or, (deadlock-freeness) is it possible to reach a marking that enables no transition? Observe that the second question actually reduces to the first. Similarly, many verification questions reduced to the reachability problem. Answers to them can, theoretically, be found by looking into any of the two semantics presented here. But in practice, it will be more interesting to use non-sequential semantics, as we explain now.

1.3 UNFOLDINGS FOR VERIFICATION

Model checking was invented independently by two groups in the beginning of the 80s [CE82; QS82], see [Clao8]. It introduced a different regard for semantics. While semantics were, and still are, a mathematical object

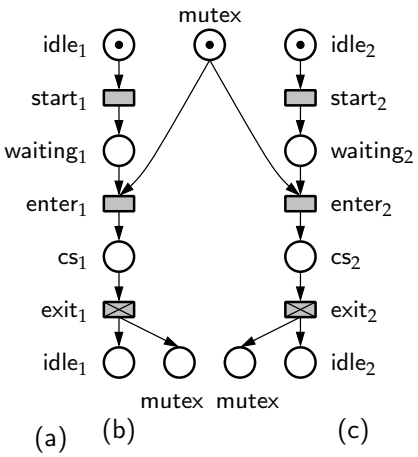


Figure 3: The finite, complete unfolding prefix constructed by McMillan's algorithm for Fig. 1.

of interest in capturing the precise meaning of programs, model checking suggested to *compute* them as a means to explore the reachable states of the system.

Model checking suffered from the SSE problem, one of the sources was concurrency, as explained above. The non-sequential semantics of Petri nets had been developed from the late 70s, providing not only an adequate representation of concurrency, but also a *concise* one, in the form of a partial order. The time was ripe and Ken McMillan made the key contribution [McM93b; McM95b]. He proposed an algorithm to construct and prune the infinite unfolding of a *bounded*[1] net into a *finite unfolding prefix* that still represented *complete* reachability information about the system.

Figure 3 shows the finite and complete unfolding prefix constructed by McMillan's algorithm from the net shown in Fig. 1. The events crossed out are the pruning points that the algorithm chooses, called *cutoffs* in the literature. Cutoffs are chosen in a way such that they prune all potentially infinite branches, but *late enough* to preserve in the prefix all reachable states of the system. Observe that this prefix is actually a prefix of Fig. 2 (e).

While complete unfolding prefixes are in general larger than the original net, they are usually much smaller than the reachability graph, effectively palliating the SSE problem derived from concurrency. Consider again Fig. 1. Exploring the states reached by all interleavings of the transitions start_1 and start_2 requires visiting four states. With n participants in the mutual exclusion, we would have 2^n states. However, the unfolding would just include n concurrent events, as shown in Fig. 3 for $n = 2$.

Although McMillan's algorithm usually builds very compact unfolding prefixes, it may also produce prefixes larger than the reachability graph. Subsequent work [ERV96; ERV02] addressed this, essentially improving the pruning algorithm so that the resulting prefix was never larger than the state space of the net.

From the perspective of complexity, unfoldings provide an interesting trade-off between space and time. While checking for deadlock-freeness or reachability on a bounded Petri net are PSPACE-complete problems on the net, they are only NP-complete when a complete unfolding prefix is on the input. Naturally, such prefix may be exponentially larger than the net, but

1 That is, one for which the reachability graph is finite.

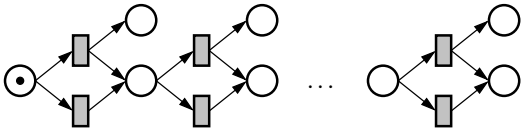


Figure 4: A Petri net with exponentially large unfolding prefix.

never larger (and often much smaller) than the reachability graph, where these problems can be solved in linear time.

Unfortunately, concurrency is not the only source of SSE in automatic verification. An important source are *sequences of choices*. For example, the smallest complete prefix of the Petri net in Fig. 4 is exponential in its size (as big as the state space) since no event can be declared a cutoff — intuitively, each reachable marking *remembers* its past, and so different runs cannot lead to the same marking.

Recently, a technique addressing this source of SSE emerged. In [KKKV06], a new condensed representation of Petri net behavior called *merged processes (MPs)* was proposed, which copes not only with concurrency, but also with sequences of choices. The main idea behind MPs is to fuse some nodes in the unfolding prefix, and use the result as the basis for verification. Moreover, MPs are sufficiently similar to unfoldings so that a large body of results developed for them can be re-used.

Another important source of SSE are concurrent read accesses, that is, multiple actions requiring non-exclusive access to a shared resource, as we explain in the next section.

1.4 CONTEXTUAL NET UNFOLDINGS

Assume that we want to model a system that has two concurrent actions, a and b, which need to test that some resource r is available for them to proceed. The resource could represent, for instance, a logical condition that a third concurrent component may independently enable or disable.

A Petri net modelling this system could be that shown in Fig. 5 (a). The only possibility for modelling the reading, or testing operation is by means of the depicted consume-produce loops. However, this is not entirely satisfactory. Actions a and b are supposed to be *truly* concurrent but they are not: the loops serialize the access to r, rather modelling a system where a and b acquire and return the right to access r. The problem lies in the fact that the modelling primitives of Petri nets can only express consuming and producing.

A solution is to incorporate *read arcs* into Petri nets, which results in the net shown in Fig. 5 (b). Read arcs are the undirected (red) lines between r and a or b. Petri nets with read arcs are called *contextual nets* [MR95] and, like ordinary Petri nets, have been given sequential, step, and non-sequential semantics [JK93; JK95; MR95; BP96; VSY98; GM98; BCM01]. Under sequential semantics, the nets in Fig. 5 (a) and (b) are completely equivalent. However, their *step semantics* authorize firing $\{a, b\}$ in (b) while such step can only be emulated by interleaving in (a).

While faithful modelling of concurrency is important, it is not the central subject of this dissertation.

Consider the unfolding semantics of Fig. 5 (a) and (b), shown in (d) and (e). Indeed, the consume-produce loops of (a) have been unfolded in (d), yielding an *explicit* representation of all the interleavings of a and b. The

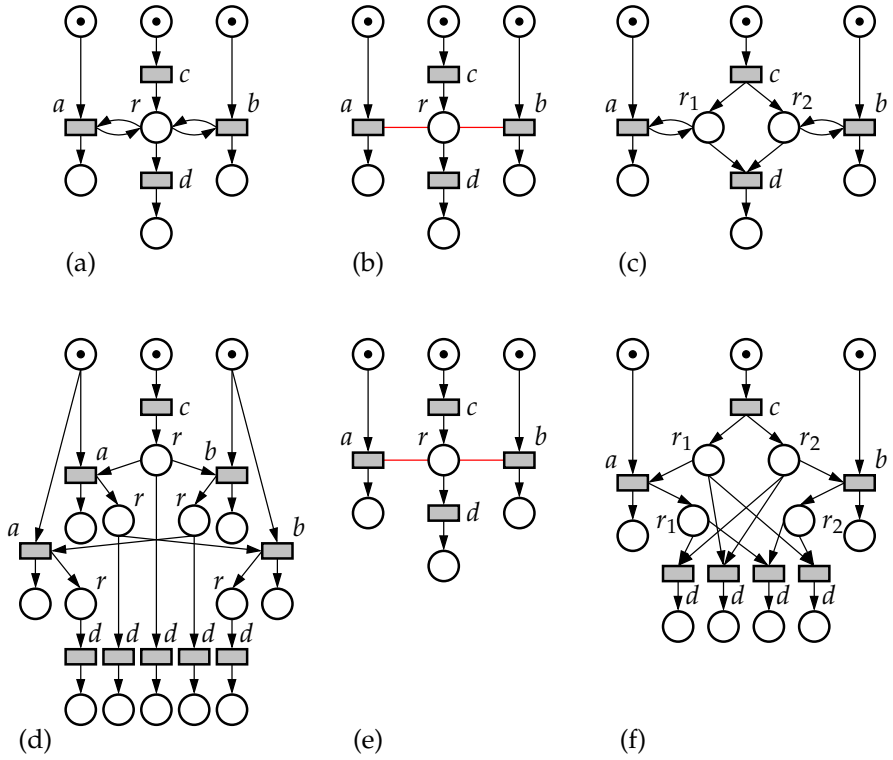


Figure 5: Three models (a), (b), and (c), of a system exhibiting concurrent read access, together with their respective unfoldings semantics (d), (e), and (f).

unfolding (e) of the already acyclic contextual net (b) is isomorphic to (b). If we had n reading actions in (a) instead of 2, (d) would be of size exponential in n, while (e) *would still be isomorphic to (b)*. The so called *place-replication (PR)* encoding of (a) [MR95; VSY98], shown in (c), partially mitigates the explosion affecting (d). The PR-encoding duplicates n times a resource with n readers, and each reader obtains a *private* copy which is accessed with a consume-produce loop. Although smaller than (d), the resulting unfolding (f) is still exponential in n.

Thus the unfolding of a contextual net can be exponentially more compact than that of an equivalent ordinary Petri nets, when the system contains concurrent read-access actions. One goal of this thesis is proving that automated verification based on contextual unfoldings improves existing methods based on ordinary unfoldings.

Concurrent read access naturally arise in many practical applications, and would benefit from verification methods based on contextual unfoldings. Contextual nets have been used, e.g., to model concurrent database access [Ris94], concurrent constraint programs [MR94], priorities [JK91], and asynchronous circuits [VSY98]. We highlight the following applications:

- *Distributed algorithms*. For instance, mutual exclusion protocols, such as those by Dijkstra and Dekker, use a number of state variables per process, which the other concurrent processes read in order to synchronize adequately. Verification of mutual exclusion protocols, and other distributed algorithms that need to *read state* could be an important application of c-net unfoldings. We provide a case study in Ch. 7.

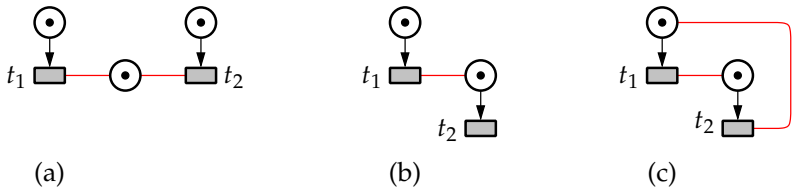


Figure 6: Three contextual nets where different step semantics found in the literature disagree on accepting the step $\{t_1, t_2\}$.

- *Asynchronous circuits*. Asynchronous digital circuits (ACs) are massively concurrent systems. Hazard checking in ACs [McM93b; VSY98] is another promising application. A network of asynchronous Boolean gates can be modelled by a c-net, where each gadget encoding a gate contains many read arcs. Hazards are unsafe behaviors of ACs, whose existence reduces to a coverability question on the c-net [McM93b]. In our experiments, we observed that signal changes in the circuit could propagate in many different orders, which were distinguished by Petri-net unfoldings but not by c-net unfoldings, reducing the unfolding size.
- *Concurrent Boolean programs*. Software verification often employs abstract representations of programs, such as Boolean programs [BR00]. A Boolean program is simply a program that only uses Boolean variables. A set of non-recursive Boolean programs communicating with shared memory can readily be encoded into a safe contextual net, where read arcs do reading or testing operations on variables. Any expression reading a global variable is a potential concurrent reading action. This application remains to be studied.

While the properties and construction of ordinary Petri net unfoldings are well-understood, research on how to construct and exploit contextual unfoldings for automated verification has been lacking so far. The rest of this section reviews the literature on contextual nets upon which this work stands.

Contextual nets have been proposed by Montanari and Rossi in [MR95], although the notion of read arc had already been discussed before [AF73; Pet81; JK91; JK93; CH93], in many cases under a different name, such as *test arc* or *activator arc*.

The step semantics of contextual nets can be grouped in two different schools, essentially distinguished by the steps that they assign to the nets in Fig. 6, see [Vog97]. While all considered semantics agree on allowing the step $\{t_1, t_2\}$ for (a), they disagree on (b) and (c). If, for instance, transition occurrences are viewed as activities taking time, then $\{t_1, t_2\}$ should be also a step of (b) and (c), as it can in fact be observed if t_1 and t_2 start at the same time. In [JK95], Janicki and Koutny develop step semantics under this notion of observation. In contrast, Montanari and Rossi [MR95] adopt a different view: only transitions that are independent from each other should be allowed in a step. Clearly, t_1 and t_2 are not independent because they cannot fire in any order: $t_1 t_2$ is a valid firing sequence but $t_2 t_1$ is not. Then $\{t_1, t_2\}$ should not be a step of (b) and (c). These semantics can be understood as viewing transition occurrences as instantaneous events. The same semantics are adopted in [CH93; BP96], as well as in this dissertation.

Process semantics for contextual nets can be found in [JK91; MR95; BP96; JK95; Vog97; Win98; GM98]. However, in this dissertation the focus will be

on the non-sequential unfolding semantics of contextual nets, independently introduced by Baldan, Corradini, and Montanari in [BCM98], and Vogler, Semenov, and Yakovlev in [VSY98]. In both works, the same unfolding semantics are defined, employing the view of [MR95] about the simultaneous occurrence of transitions.

A first unfolding procedure for constructing finite and complete contextual prefixes is found in [VSY98]. Their procedure is essentially a generalization of McMillan's algorithm, and works for only a restricted class of so-called *read-persistent* contextual nets. The authors find that interesting practical applications, such as hazard checking in asynchronous circuits, lie within this class. However, the class is quite restricted. It consist of all the contextual nets that have no reachable marking which enables two transitions such that one consumes a place read by the other. The c-nets in Fig. 6 (b) and (c), for instance, lie outside, despite their simple structure.

The problem with non read-persistent nets is that events have multiple *causal histories*. Roughly speaking, a history of an event e is a set of events that must precede e in a possible execution. This multiplicity negatively interacts with the cutoff criterion of McMillan's algorithm, preventing the latter from constructing complete prefixes.

A general procedure for the whole class of contextual nets was later proposed by Winkowsky [Win02], who lifted McMillan's cutoff criterion from local configurations [McM93b] to the aforementioned causal histories. The procedure always terminates with a complete unfolding prefix, but it is not constructive in general, as events may have infinitely many causal histories.

A constructive, general solution was finally given in [BCKS08], at the price of making the underlying theory notably more complicated. In particular, computing a complete prefix required to annotate every event e with a *subset* of its histories.

However, it remained unclear whether the approach of [BCKS08] could be implemented with reasonable efficiency, and how. For safe nets, the interest of computing a complete contextual prefix was not evident from a practical point of view: while the prefix can be exponentially smaller than the complete prefix of the corresponding PR-encoding, the intermediate structure used to produce it has asymptotically the same size. More precisely, the number of histories that the approach of [BCKS08] needs to consider to construct the contextual prefix matches the number of events in the prefix of the PR-encoding (for general bounded nets, this is not the case).

Additionally, only an abstract procedure was given in [BCKS08], mainly focusing on the technical difficulties of the cutoff criterion. It was left unresolved how to turn the procedure into a reasonably efficient algorithm, and no data structure for dealing with causal histories was proposed.

1.5 CONTRIBUTIONS AND OUTLINE

This dissertation makes foundational and practical contributions to model checking and fault diagnosis. We present verification algorithms for Petri nets that exploit the aforementioned compactness of contextual unfoldings, and extend an unfolding-based diagnosis method with fairness assumptions. We implement our model checking algorithms aiming at producing an efficient and robust model checker intended to handle realistic benchmarks.

We demonstrate that contextual unfolding construction and analysis is practical and outperforms existing techniques on a wide number of cases.

Contextual nets and their unfoldings therefore have a rightful place in research on concurrency, also from an efficiency point of view.

More specifically, the outline and contributions of the manuscript are:

- *Chapter 3*. We render effective the abstract procedure of [BCKS08] with a view to efficiency. We develop a concurrency relation on a notion of unfolding conditions enriched with histories, use it to characterize the possible extensions of the prefix, and provide an inductive characterization of this relation that lies at the heart of the efficient computation of prefix extensions. We include adequate orders in the framework of contextual unfoldings and compare our method to the one independently proposed in [BBC+10]. This chapter is based on [RSB11b; BBC+12].
- *Chapter 4*. We encode into SAT the reachability and deadlock-freeness problems based on contextual unfoldings. Unlike conventional unfoldings, contextual unfoldings may contain cycles of so-called asymmetric conflict. We propose and optimize encodings for avoiding them. A number of additional optimizations are also studied. This chapter is based on [RS12b].
- *Chapter 5*. We integrate two methods for representing the state space of Petri nets: merged processes and contextual unfoldings. The resulting technique, called *contextual merged processes* (CMPs), combines the advantages of the original techniques and copes with several important sources of state space explosion: concurrency, sequences of choices, and concurrent read accesses to shared resources. A SAT encoding of the reachability problem based on CMPs is presented. This chapter is based on [RSK13].
- *Chapter 6*. We present an unfolding-based procedure for deciding the weak fault diagnosis problem. The diagnosis approaches proposed in [BFHJ03] and [EK12], mostly focused on sequential observations. We generalize them to partially ordered observations, and extend the method adding fairness assumptions, which gives rise to weak diagnosis. Additionally, we present a decision procedure of the weak diagnosis problem using SAT. The results in this chapter hold for *ordinary Petri nets*, not contextual nets. This chapter is based on [HRS13].
- *Chapter 7*. We evaluate the techniques presented in Ch. 3 to 5, and compare them with with existing methods to construct or analyze unfoldings. We implemented the techniques of Ch. 3 and 4 into the Cunf Toolset, a competitive verification tool. The relevant data structures and algorithms of the implementation are described, and optimizations to the SAT encoding of Ch. 4 are evaluated. We demonstrate over a standard benchmark suite that contextual unfolding construction and analysis is practical and outperforms existing techniques on a wide number of cases. This chapter is based on [RSB11b; BBC+12; RS12b; RSK13; RS13b].

Chapter 2 fixes general definitions and assumptions for the rest of the manuscript, and contains a number of small technical results that will be used later. We conclude in Ch. 8. The manual of the Cunf Toolset is contained in App. A.

1.6 PUBLICATIONS

Most of the contributions in this manuscript have been already published in international conferences or journals. The following publications partially contain the results of this dissertation:

[BBC+12] Paolo Baldan, Alessandro Bruni, Andrea Corradini, Barbara König, César Rodríguez, and Stefan Schwoon. "Efficient unfolding of contextual Petri nets." In: *Theoretical Computer Science (TCS)* 449 (Aug. 2012), pp. 2–22.

[HRS13] Stefan Haar, César Rodríguez, and Stefan Schwoon. "Reveal Your Faults: It's Only Fair!" In: *2013 13th International Conference on Application of Concurrency to System Design (ACSD)*. 2013, pp. 120–129.

[RS12b] César Rodríguez and Stefan Schwoon. "Verification of Petri Nets with Read Arcs." In: *CONCUR 2012 – Concurrency Theory*. Ed. by Maciej Koutny and Irek Ulidowski. Vol. 7454. LNCS. Springer, Sept. 2012, pp. 471–485.

[RS13b] César Rodríguez and Stefan Schwoon. "Cunf: A Tool for Unfolding and Verifying Petri Nets with Read Arcs." In: *Automated Technology for Verification and Analysis (ATVA)*. Ed. by Dang Van Hung and Mizuhito Ogawa. Vol. 8172. LNCS. Springer, 2013, pp. 492–495.

[RSB11b] César Rodríguez, Stefan Schwoon, and Paolo Baldan. "Efficient contextual unfolding." In: *CONCUR 2011 – Concurrency Theory*. Ed. by Joost-Pieter Katoen and Barbara König. Vol. 6901. LNCS. Springer, Sept. 2011, pp. 342–357.

[RSK13] César Rodríguez, Stefan Schwoon, and Victor Khomenko. "Contextual Merged Processes." In: *Proc. International Conference on Application and Theory of Petri Nets and Concurrency (ICATPN)*. Vol. 7927. LNCS. June 2013, pp. 29–48.

During the last three years, the author has also collaborated in the development of three more publications, which have not been integrated into this manuscript to keep the presentation coherent. These are the following:

[KLB+13] Fabrice Kordon, Alban Linard, Marco Becutti, Didier Buchs, Lukasz Fronc, Francis Hulin-Hubard, Fabrice Legond-Aubry, Niels Lohmann, A. Marechal, Emmanuel Paviot-Adet, Franck Pommereau, César Rodríguez, Christian Rohr, Yann Thierry-Mieg, Haro Wimmel, and Karsten Wolf. *Web Report on the Model Checking Contest @ Petri Net 2013*. June 2013. URL: mcc.lip6.fr.

[RS13a] César Rodríguez and Stefan Schwoon. "An Improved Construction of Petri Net Unfoldings." In: *Proc. of the French-Singaporean Workshop on Formal Methods and Applications (FSFMA'13)*. Vol. 31. OASICS. Leibniz-Zentrum für Informatik, July 2013, pp. 47–52.

[SR11] Stefan Schwoon and César Rodríguez. "Construction and SAT-based verification of Contextual Unfoldings." In: *Proc. of the 13th International Workshop on Descriptional Complexity of Formal Systems (DCFS'11)*. Vol. 6808. LNCS. Extended abstract. Springer, July 2011, pp. 34–42.

2 PRELIMINARIES

This first technical chapter recalls and formally defines general background that will be necessary for the rest of the manuscript. New, often technical, results are spread over the presentation. We have decided to place them here not only because they are preliminary results, necessary for subsequent developments, but mainly because they contribute to forge deeper intuitions about the basic notions presented here, before they are used to construct, over the next chapters, the main contributions of the thesis.

The outline is as follows. We recall basic notions about orders and multisets in § 2.1. Petri nets with read arcs and related notions are formally defined in §§ 2.2 and 2.3, and two different ways to encode them into ordinary Petri nets are recalled. Their non-sequential unfolding semantics are presented in § 2.4.

2.1 ORDERS AND MULTISETS

Let S be a set. A relation $R \subseteq S \times S$ is

- *reflexive* if $(s,s) \in R$ for all $s \in S$;
- *irreflexive* if $(s,s) \notin R$ for all $s \in S$;
- *transitive* if $\{(s,t),(t,u)\} \subseteq R$ implies $(s,u) \in R$ for all $s,t,u \in S$;
- *asymmetric* if $(s,t) \in R$ implies $(t,s) \notin R$ for all $s,t \in S$; and
- *antisymmetric* if $\{(s,t),(t,s)\} \subseteq R$ implies $s = t$ for all $s,t \in S$.

A *(non-strict) partial order* is any reflexive, transitive, and antisymmetric relation. A *strict partial order* is any irreflexive and transitive relation. Any strict partial order is asymmetric and antisymmetric, which can easily be derived from the definition. Finally recall that if R is asymmetric, it is also irreflexive.

The reflexive closure of any strict partial order is a partial order. Conversely, the reflexive reduction of any partial order is a strict partial order.

Any partial order R, strict or not, is *total* if any two elements of S are ordered, i.e., ether $(s,t) \in R$ or $(t,s) \in R$ holds for all $s,t \in S$ with $s \neq t$. For any subset $S' \subseteq S$, we denote by $R_{S'}$ the relation $R \cap (S' \times S')$, also called the *restriction* of R to S'.

A *chain* of R is any subset $C \subseteq S$ that is totally ordered under R, i.e., R_C is total. A *linear extension* of R, or *topological ordering* of S, is any total order R' on S such that $R \subseteq R'$. Observe that a linear extension of R always exists because R is a partial order. In many cases we will also call R' an *interleaving* of R — or S if R is clear.

We identify any relation R on S with the directed graph whose edge relation is precisely R. We say that R is *acyclic* iff its associated digraph is. In other words, R is acyclic if its transitive closure is irreflexive, i.e., a strict partial order. Similarly, we will speak about the *Strongly Connected Components* (SCCs) of R to mean SCCs of the directed graph identified by R.

A *multiset* over a set S is a function $M \colon S \to \mathbb{N}$. The *support* of M is the set $\bar{M} := \{x \in S \colon M(x) > 0\}$ of elements in S occurring at least once in M.

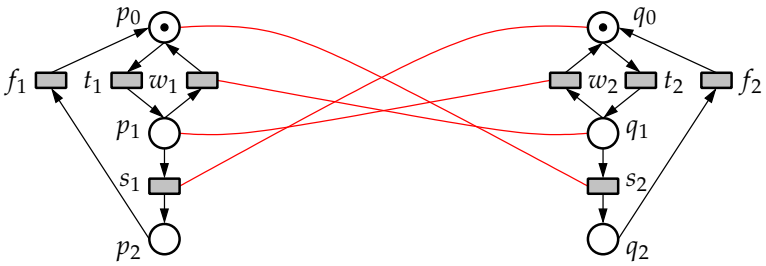


Figure 7: A c-net with four read arcs.

We write $x \in M$ if x is in the support of M; we say that M is finite iff its support is. Given multisets M and N over S, their sum and difference are

$$(M + N)(x) := M(x) + N(x),$$
$$(M - N)(x) := \max(0, M(x) - N(x)).$$

We write $M \leq N$ iff $M(x) \leq N(x)$ for all $x \in S$. We let $\mathbf{S}$ (observe the bold typography) denote the set of multisets over S. For any function $f\colon S \to T$ we lift f to multisets as the partial function $\mathbf{f}\colon \mathbf{S} \to \mathbf{T}$ (again observe the bold typography for $\mathbf{f}$) that maps $M \in \mathbf{S}$ to $\mathbf{f}(M) = N$ where $N(t) := \sum_{s \in f^{-1}(t)} M(s)$. Observe that N is well-defined iff for every $t \in T$ finitely many terms $M(s)$ are non-zero, with $s \in f^{-1}(t)$. This is always the case if, e.g., M has a finite support. Any set will be interpreted as a multiset in the natural way.

2.2 CONTEXTUAL NETS

A *contextual net* (c-net) is a tuple $N := \langle P, T, F, C, m_0 \rangle$, where

- P and T are disjoint sets of *places* and *transitions*,
- $F \subseteq (P \times T) \cup (T \times P)$ is the *flow relation*,
- $C \subseteq P \times T$ is the *context relation*, and
- the *initial marking* m_0 is a multiset over P.

A pair $(p, t) \in C$ is called *read arc*. A *Petri net* is a c-net without read arcs. N is called *finite* if P and T are finite sets. Places and transitions together are called *nodes*. The *size* $|N|$ of the net is the number of nodes and arcs of the net[1], i.e., $|P| + |T| + |F| + |C|$. For any node $x \in P \cup T$ we will sometimes write $x \in N$ to mean that x is a node of N. Throughout this manuscript, whenever N denotes a c-net, we will assume that $N := \langle P, T, F, C, m_0 \rangle$, unless otherwise stated. Subscripts, primes or other decorations of N will pass through to the components of the tuple.

Figure 7 shows a c-net. As usual, transitions are depicted as boxes and places as circles. Read arcs are depicted as undirected lines.[2] For instance, (s_1, q_0) is a read arc. Places initially marked will be drawn with a token inside. In this case, the initial marking puts one token on places p_0 and q_0.

Within this work, Petri nets will be referred or qualified as *ordinary* nets, as opposed to contextual nets.[3] By extension, any notion related to them will also be qualified as ordinary. For instance, we will speak of ordinary

1 Indeed, we do not include the initial marking. This is because in this manuscript we will assume that it is *safe*, see the general assumptions in p. 18.
2 In the color version of this document, read arcs are also colored in red for readability.
3 Observe that this use is not standard: in the literature of general Petri nets, an *ordinary Petri net* is one whose arc weights are 1's.

occurrence nets, or ordinary unfoldings when discussing the corresponding notions in *Petri net* theory.

For any node $x \in P \cup T$, the *preset*, *postset*, and *context* of x is defined as a function from x to, respectively, the set:

$$\begin{aligned} {}^\bullet x &:= \{y \in P \cup T \colon (y, x) \in F\}, \\ x^\bullet &:= \{y \in P \cup T \colon (x, y) \in F\}, \\ \underline{x} &:= \{y \in P \cup T \colon (y, x) \in C \cup C^{-1}\}. \end{aligned}$$

Such notions are naturally lifted to sets and multisets of nodes. In Fig. 7, we have ${}^\bullet s_1 = \{p_1\}$, $s_1^\bullet = \{p_2\}$, $\underline{s_1} = \{q_0\}$, $\underline{\{p_1, q_1\}} = \{w_1, w_2\}$.

A *marking* of N is a multiset $m \colon P \to \overline{\mathbb{N}}$ over P. We say that m is *k-bounded* if $m(p) \leq k$ for all $p \in P$. A set $A \subseteq T$ of transitions is *enabled* at marking m if for all $p \in P$,

$$m(p) \geq |p^\bullet \cap A| + \begin{cases} 1 & \text{if } \underline{p} \cap A \neq \emptyset \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Such A can *occur* or *be executed*, leading to a new marking m', where

$$m'(p) = m(p) - |p^\bullet \cap A| + |{}^\bullet p \cap A|$$

for all $p \in P$. We call $\langle m, A, m' \rangle$ a *step* of N. If A is a singleton with one transition $t \in T$, we also write $\langle m, t, m' \rangle$. Although the enabling condition could be defined for multisets A instead of sets, for the purposes of this work, sets will suffice.

Some examples follow. In Fig. 7, the initial marking enables the sets $\{t_1\}$ and $\{t_1, t_2\}$. In both cases, the marking covers the pre-places of all transitions; and the presets of t_1, t_2 do not overlap. A marking m such that $m(p_1) = 2$ and $m(q_0) = 2$ would enable the set $\{s_1, t_2\}$. Here, $\underline{s_1}$ overlaps ${}^\bullet t_2$, but the only place in common, q_0, contains two tokens. Coming back to the discussion about steps semantics in § 1.4, and considering Fig. 6 (b), the set $\{t_1, t_2\}$ is not enabled at the initial marking. In this case, the only place in $\underline{t_1} \cap {}^\bullet t_2$ contains only one token, but (1) demands at least two.

A finite sequence of transitions $\sigma = t_1 \dots t_n \in T^*$ is a *run* or *firing sequence* iff there exist markings $m_1, \dots, m_n$ such that $\langle m_{i-1}, t_i, m_i \rangle$ is an step for all for $1 \leq i \leq n$, and m_0 is the initial marking of N. If such a run exists, m_n is said to be *reachable*; we denote by $reach(N)$ the set of reachable markings of N. For instance, in Fig. 7, $t_1 s_1 t_2 f_1$ is a run. An infinite sequence $t_1 t_2 \dots \in T^\infty$ is a run if all its finite prefixes are. We call any run *repetition-free* if it fires every transition at most once.

A c-net N is said to be *k-bounded* if every reachable marking of N is k-bounded; we say N is *bounded* if it is k-bounded for some $k \in \mathbb{N}$ and *safe* if it is 1-bounded. For safe nets, we treat markings as sets of places. A marking m is *deadlocked* if it does not enable any transition, and *coverable* if there is some other reachable marking $m' \in reach(N)$ such that $m \geq m'$.

The *reachability graph* of N is the directed graph whose vertices are all reachable markings of N, i.e., $reach(N)$, and whose edges are all pairs (m, m') such that $\langle m, t, m' \rangle$ is an step for some $t \in T$.

The *immediate causality* relation on N is the binary relation $<_i$ on $P \cup T$ defined as $x <_i y$ iff

$$(x, y) \in F, \text{ or} \tag{2}$$

$$x, y \in T \text{ and } x^\bullet \cap \underline{y} \neq \emptyset \tag{3}$$

The second condition, (3), is new for contextual nets w.r.t. ordinary nets. In Fig. 8, for instance, it establishes that $t_1 <_i t_2$. The *causality* relation $<$ is the

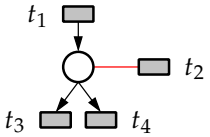


Figure 8: Illustrating causality, symmetric, and asymmetric conflict.

transitive closure of $<_i$, and we denote by $\leq$ the reflexive closure of $<$. For a node x, we define its set of *causes* as the set of *transitions* preceding it:

$$[x] := \{t \in T : t \leq x\}.$$

A set $X \subseteq T$ is *causally closed* if $[t] \subseteq X$ for all $t \in X$. The *conflict* relation $\# \subseteq (P \cup T)^2$ is the least symmetric relation satisfying

- $t \# t'$ if $t, t' \in T$ with $t \neq t'$ and ${}^\bullet t \cap {}^\bullet t' \neq \emptyset$; and
- $x \# z$ if there is $y \in P \cup T$ such that $x \# y$ and $y < z$.

In Fig. 8, we have $t_3 \# t_4$. Two transitions t and t' are in *asymmetric conflict*, written $t \nearrow t'$, iff either

$$t^\bullet \cap ({}^\bullet t' \cup \underline{t'}) \neq \emptyset, \text{ or} \tag{4}$$
$$\underline{t} \cap {}^\bullet t' \neq \emptyset, \text{ or} \tag{5}$$
$$t \# t'. \tag{6}$$

Again in Fig. 8, we have the following examples

$t_1 \nearrow t_2$, by (4) $\quad$ $t_1 \nearrow t_3$, by (4) $\quad$ $t_2 \nearrow t_3$, by (5)
$t_3 \nearrow t_4$, by (6) $\quad$ $t_4 \nearrow t_3$, by (6)

As explained in § 2.1, for any set $X \subseteq T$ of transitions, we will denote the relation $\nearrow \cap (X \times X)$ by $\nearrow_X$.

Although we define causality, symmetric, and asymmetric conflicts for general c-nets, the intuition behind them is only meaningful for occurrence nets, a subclass of c-nets defined in § 2.4. For this reason, it will be in § 2.4 where we explain more in detail these definitions. We will instantiate these notions for general c-nets on certain cases, so we need to define them for general c-nets rather than directly for occurrence nets.

Remark 1. *For any c-net,* $(<_i \cup <_i^2) \cap (T \times T) \subseteq \nearrow$.

Proof. If $t <_i t'$ for some t and t' in T, necessarily (3) holds, and then (4) also holds. If $t <_i^2 t'$ holds, there is some place p such that $t <_i p <_i t'$. Then (4) holds between t and t'. □

Remark 2. *For any Petri net,* $\nearrow = \# \cup (<_i^2 \cap (T \times T))$

GENERAL ASSUMPTIONS. Throughout this manuscript, we will often assume that a c-net N satisfies the following conditions:

1. $({}^\bullet t \cup t^\bullet) \cap \underline{t} = \emptyset$ for each transition $t \in T$; (7)
2. ${}^\bullet t \neq \emptyset$ for all $t \in T$; (8)
3. m_0 is safe; (9)

Observe that due to (7), relation $<_i$ is irreflexive. Assuming that the initial marking m_0 is safe is not restrictive[4] and will simplify the notation of some definitions, notably the branching processes of N, in § 2.4.2.

4 As it can always be ensured, introducing a new safe initial marking from which a set of transitions produce the unsafe previous initial marking.

Let $N' = \langle P', T', F', C', m'_0 \rangle$ be a c-net. A *homomorphism* [VSY98] from N to N' is a function $h\colon P \cup T \to P' \cup T'$ satisfying:

- $h(P) \subseteq P'$ and $h(T) \subseteq T'$; (10)
- $\mathbf{h}(m_0) = m'_0$; (11)
- for all $t \in T$, $\mathbf{h}({}^\bullet t) = {}^\bullet h(t)$, $\mathbf{h}(t^\bullet) = h(t)^\bullet$, and $\mathbf{h}(\underline{t}) = \underline{h(t)}$. (12)

Such a homomorphism is a specialisation of Definition 4.20 in [BCM01]. Notice that (11) is equivalent to asking that h restricted to m_0 is a bijection between m_0 and m'_0. Similarly, (12) requests that for every transition t of N, h restricted to ${}^\bullet t$ is a bijection to ${}^\bullet h(t)$, and similarly for $t^\bullet$ and $\underline{t}$. An *isomorphism* between N and N' is a bijection $f\colon P \cup T \to P' \cup T'$ such that f and f^{-1} are homomorphisms.

We finish this section showing that asymmetric conflict, causality, and steps are, among other notions, preserved by homomorphisms. Remark that the following result is restricted to singleton steps, which are enough for our purposes.

Lemma 1. *Let N and N' be c-nets, and h be a homomorphism from N to N'. If $\langle m, t, \widehat{m} \rangle$ is a step of N and $\mathbf{h}(m)$ is well-defined, then*

$$\langle \mathbf{h}(m), h(t), \mathbf{h}(\widehat{m}) \rangle \text{ is a step of } N'.$$

Furthermore, for any nodes x, y and transitions t, u of N,

$$x < y \text{ implies } h(x) < h(y)$$

and

$$t \nearrow u \text{ implies either } h(t) \nearrow h(u) \text{ or } h(t) = h(u).$$

Proof. That c-net homomorphisms preserve steps, the first part of the lemma, is already stated by Proposition 4.1 in [BCM01], since our definition of homomorphism is a specialisation of theirs — we enforce h to be total and $h(p)$ is a place instead of a multiset of places. At any rate, the second part of our lemma is new, and what follows is a proof of both parts.

Let P, T and P', T' be the places and transitions of, respectively, N and N'. Let $\langle m, t, \widehat{m} \rangle$ be step of N such that $m' := h(m)$ is a well-defined marking of N'. Let $t' := h(t)$.

We first show that t' is enabled at m'. Let $p' \in {}^\bullet t'$. Because h restricted to ${}^\bullet t$ is a bijection between ${}^\bullet t$ and ${}^\bullet t'$, there is a single $p \in {}^\bullet t$ such that $h(p) = p'$. Since t is enabled at m, $m(p) \geq 1$. Recall that $m'(p') = \sum_{\widehat{p} \in h^{-1}(p')} m(\widehat{p})$. Then $m'(p') \geq 1$ because $p \in h^{-1}(p')$. An analogous argument shows that $m'(p') \geq 1$ if $p' \in \underline{t'}$.

So m' enables t'. Let $\langle m', t', \widehat{m}' \rangle$ be a step of N'. We show that $h(\widehat{m}) = \widehat{m}'$, i.e., that for any $p' \in P'$, we have $h(\widehat{m})(p') = \widehat{m}'(p')$. We proceed as follows:

$$\begin{aligned}
h(\widehat{m})(p') &= \sum_{p \in h^{-1}(p')} \widehat{m}(p) \\
&= \sum_{p \in h^{-1}(p')} (m(p) - |\{p\} \cap {}^\bullet t| + |\{p\} \cap t^\bullet|) \\
&= \Big(\sum_{p \in h^{-1}(p')} m(p) \Big) - \Big(\sum_{p \in h^{-1}(p')} |\{p\} \cap {}^\bullet t| \Big) + \\
&\quad \Big(\sum_{p \in h^{-1}(p')} |\{p\} \cap t^\bullet| \Big) \\
&= \Big(\sum_{p \in h^{-1}(p')} m(p) \Big) - |h^{-1}(p') \cap {}^\bullet t| + |h^{-1}(p') \cap t^\bullet|
\end{aligned}$$

As for $\widehat{m}'(p')$, we have:

$$\begin{aligned}\widehat{m}'(p') &= h(m)(p') - |\{p'\} \cap {}^\bullet t'| + |\{p'\} \cap t'^\bullet| \\ &= \Big(\sum_{p \in h^{-1}(p')} m(p)\Big) - |\{p'\} \cap {}^\bullet t'| + |\{p'\} \cap t'^\bullet|\end{aligned}$$

Therefore, to show that $h(\widehat{m})(p') = \widehat{m}'(p')$ it suffices to show that

$$|\{p'\} \cap {}^\bullet t'| = |h^{-1}(p') \cap {}^\bullet t|$$

and that

$$|\{p'\} \cap t'^\bullet| = |h^{-1}(p') \cap t^\bullet|.$$

Either $p' \in {}^\bullet t'$ holds or not. If it holds, there is a single $p \in {}^\bullet t$ such that $h(p) = p'$, since otherwise h restricted to ${}^\bullet t$ would not be a bijection between ${}^\bullet t$ and ${}^\bullet t'$. Then $|h^{-1}(p') \cap {}^\bullet t| = 1$. If $p' \notin {}^\bullet t'$, then no $p \in {}^\bullet t$ is such that $h(p) = p'$, again because h is a homomorphism. Then $|h^{-1}(p') \cap {}^\bullet t| = 0$. This proves that $|\{p'\} \cap {}^\bullet t'| = |h^{-1}(p') \cap {}^\bullet t|$; an analogous argument shows that $|\{p'\} \cap t'^\bullet| = |h^{-1}(p') \cap t^\bullet|$.

We now show why $x <_i y$ implies $h(x) <_i h(y)$, for $x, y \in P \cup T$. That $x < y$ implies $h(x) < h(y)$ is basically a consequence of the previous fact. Three cases are possible:

- $x \in P$ and $x \in {}^\bullet y$. Since $y \in T$, we know that h restricted to ${}^\bullet y$ is a bijection to ${}^\bullet h(y)$. So $h(x) \in {}^\bullet h(y)$ and $h(x) <_i h(y)$.
- $x \in T$ and $x \in {}^\bullet y$. Analogous.
- $x, y \in T$ and $x^\bullet \cap \underline{y} \neq \emptyset$. Then let $p \in x^\bullet$ be such that $p \in \underline{y}$. Again, $h(p)$ is in $h(x)^\bullet$ and in $\underline{h(y)}$, and so $h(x) <_i h(y)$.

We now show that $t \nearrow u$ implies either $h(t) \nearrow h(u)$ or $h(t) = h(u)$. Assume the hypothesis and assume that $h(t) \neq h(u)$, we show that $h(t) \nearrow h(u)$. Three cases are possible:

- $t^\bullet \cap ({}^\bullet u \cup \underline{u})$ contains some place p. Then, by the properties of homomorphisms, $h(p)$ is contained in $h(t)^\bullet$, and in ${}^\bullet h(u)$ or in $\underline{h(u)}$. In any case, $h(t) \nearrow h(u)$.
- $\underline{t} \cap {}^\bullet u$ contains some place p. Analogous argument.
- ${}^\bullet t \cap {}^\bullet u$ contains some place p. Analogous argument. □

As usual, homomorphisms preserve runs and reachable markings: if σ is a run of N that reaches m, then $h(\sigma)$ is a run of N' that reaches $\mathbf{h}(m)$, because $\mathbf{h}(m_0) = m_0'$ is a well-defined marking and due to Lemma 1.

2.3 ENCODING CONTEXTUAL NETS INTO ORDINARY PETRI NETS

A c-net N can be encoded into a Petri net whose reachable markings are in one-to-one correspondence with those of N. We discuss two such encodings, and illustrate them by the c-net N in Fig. 9 (a). Place p has two transitions t_2, t_3 in its context, modelling a situation where, e.g., two processes are accessing in a read-only way a common resource represented by p. Note that the step $\{t_2, t_3\}$ can occur in N after executing t_1.

2.3.1 *Plain Encoding*

Given a c-net N, the *plain encoding* of N is the net N_p obtained by replacing every read arc (p, t) in the context relation by a consume-produce loop

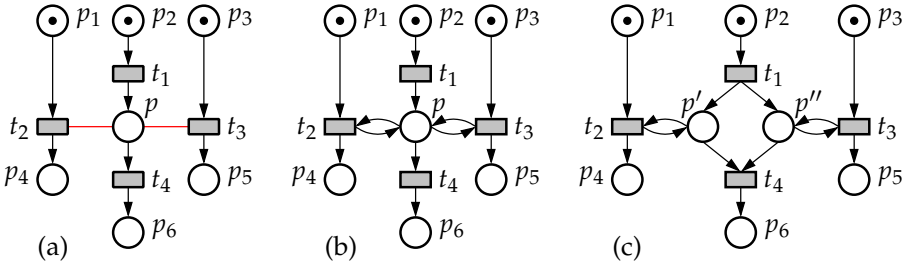


Figure 9: (a) A c-net; (b) its *plain encoding* N'; (c) and its *Place-Replication encoding* N''

$(p,t),(t,p)$ in the flow relation. The net N_p has the same reachable markings as N; it also has the same runs but not the same steps as N. The plain encoding of the net shown in Fig. 9 (a) is the net shown in (b). Observe that in (b) the firings of t_2 and t_3 are sequentialized: after executing t_1, the step $\{t_2, t_3\}$, which was possible in (a), can no longer occur.

2.3.2 *Place-Replication Encoding*

The *Place-Replication (PR-) encoding* [MR95; VSY98] of a c-net N is a Petri net N_r in which we substitute every place p in the context of $n \geq 1$ transitions $t_1, \ldots, t_n$ by places $p_1, \ldots, p_n$ and update the flow relation of N_r as follows. For all $i \in \{1, \ldots, n\}$,

1. transition t_i consumes and produces place p_i;
2. any transition t producing p in N produces p_i in N'';
3. any transition t consuming p in N consumes p_i in N''.

The PR-encoding of the net N in Fig. 9 (a) is the net N'' shown in Fig. 9 (c). Reachable markings, runs, and steps of N'' are in bijective correspondence to those of N.

More formally, let $N := \langle P, T, F, C, m_0 \rangle$ be a c-net. Assume that P is partitioned in two sets P_0 and P_1 that contain, respectively, all places with zero (P_0) and one or more (P_1) incident read arcs. The PR-encoding of N is the ordinary net $N_r := \langle P_r, T_r, F_r, m_{0,r} \rangle$, defined as follows.

- $P_r := P_0 \cup \{\langle p,t \rangle : p \in P_1, t \in T, (p,t) \in C\}$
- $T_r := T$
- F' contains all directed arcs (p,t) and (t,p) in F such that $p \in P_0$, plus
  - arcs $(\langle p,t \rangle, t')$ for all $\langle p,t \rangle \in P_r$ and all $(p,t') \in F$,
  - arcs $(t', \langle p,t \rangle)$ for all $\langle p,t \rangle \in P_r$ and all $(t',p) \in F$,
  - arcs $(t, \langle p,t \rangle)$ and $(\langle p,t \rangle, t)$ for all $\langle p,t \rangle \in P_r$.
- $m_{0,r}$ marks all $P_0 \cap P_r \subseteq m_0$ and all $\langle p,t \rangle \in P_r$ such that $p \in m_0$.

Remark 3. *A sequence $t_1 t_2 \ldots \in T^*$ is a run of N iff it is a run of N_r.*

Proof. Steps of N are in bijective correspondence with the steps of N_r. □

2.4 UNFOLDINGS

In this section we define the notion of unfolding of a c-net and recall properties of it. We mainly follow [BCKS08].

2.4.1 *Occurrence Nets*

In the context of this manuscript, an occurrence net is a c-net that represent the occurrences of transitions in one or several executions of the net. All five nets illustrated in Fig. 2 are occurrence nets.

Formally, an *occurrence net* (ON) is a c-net $O = \langle B, E, G, D, \tilde{m}_0 \rangle$ that satisfies the following:

- $|{}^\bullet c| \leq 1$ holds for any $c \in B$; (13)
- $[e]$ is finite for all $e \in E$; (14)
- $\nearrow_{[e]}$ is acyclic for all $e \in E$; and (15)
- $\tilde{m}_0 = \{c \in B : {}^\bullet c = \emptyset\}$. (16)

As before, when denoting an ON by O, unless otherwise stated, we will assume $O := \langle B, E, G, D, \tilde{m}_0 \rangle$. For the rest of this section, let O be an ON.

As per tradition, we call the elements of B *conditions*, and those of E *events*. The requirements (13), (14) and (16) are usual in this definition. The *general assumptions* of p. 18 also apply to ON; in particular, property (8) implies that all $<$-minimal nodes are conditions, and by (16) they are initially marked. Notice that $\tilde{m}_0$ is safe. As for (15), we shall see later in this section that it asks for $[e]$ to be, intuitively, a *firable* set of events.

Occurrence nets are defined above differently than in [BCKS08; BCM01; Win02]: we do not require O to be safe or $<$ to be a (strict) partial order.[5] These properties can rather be derived from the definition:

Remark 4. *For any occurrence net O:*

1. *$<$ is irreflexive, i.e., it is a strict partial order.*
2. *Every run of O is repetition-free.*
3. *O is safe.*

Proof.

1. For some node x, assume that $x < x$. If $x <_i x$ then necessarily $x \in E$ and by Rmk. 1 we have $x \nearrow x$, a contradiction to (15). On the other hand, if $<_i$ is irreflexive, then the pair (x, x) has been added to $<$ during the transitive closure and there are $n \geq 1$ nodes $x_1, \ldots, x_n$ of O that form a cycle of $<_i$ which involves x, i.e., such that
$$x <_i x_1 <_i \ldots <_i x_n <_i x.$$
Since either x or x_1 is an event, w.l.o.g. we can assume $x \in E$. From this cycle in the relation $<_i$ we can extract a cycle in the relation $<_e := <_i \cup <_i^2$ which only involves *events*. That is, there are $m \geq 0$ events $x_{i_1}, \ldots, x_{i_m}$ such that
$$x <_e x_{i_1} <_e \ldots <_e x_{i_m} <_e x,$$
and $i_j \in \{1, \ldots, n\}$ for $1 \leq j \leq m$, and each i_j is either $i_{j-1} + 1$ or $i_{j-1} + 2$ for $2 \leq j \leq m$. Notice that all x_{i_j} are in $[x]$. By Rmk. 1, such cycle of in relation $<_e$ is a cycle of $\nearrow_{[x]}$, a contradiction to (15).
2. By induction on the length n of any run $e_1, \ldots, e_n$ of O. The claim is obviously true for $n \leq 1$. If, for a proof by contradiction, $e_n = e_i$ for some $1 \leq i < n$, any condition $c \in {}^\bullet e_i$ is marked twice. By (16), $c \notin \tilde{m}_0$; by (13), the single event in ${}^\bullet c \subseteq \{e_1, \ldots, e_{i-1}\}$ fires twice in the run $e_1, \ldots, e_{n-1}$, a contradiction.
3. Marking a condition c twice needs to fire ${}^\bullet c$ twice. □

5 See def. 6 in [BCKS08], def. 5.24 in [BCM01], and def. 3.1 in [Win02]. In contrast, acyclicity of $<$ in [VSY98] is also derived from the definition of ONs. However, their definition of asymmetric conflict is slightly different from ours, and we prefer to have a new proof.

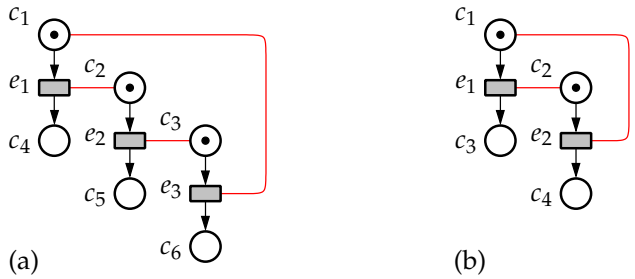


Figure 10: Two occurrence nets illustrating a circular asymmetric conflict of length 3 (a) and two (b).

In the light of Rmk. 4, let us present some intuition behind the relations $<$, #, and $\nearrow$ in ONs. Any two nodes x, y of O satisfy $x < y$ iff *any* run that fires or marks y first fires or marks x. Intuitively, this is a consequence of the fact that a condition is never marked again once it is consumed, which in turn follows after the second item in Rmk. 4 and (13).

If x, y are in symmetric conflict, then no run that fires or marks x, fires or marks y, for the same reasons as before. In *ordinary* ONs, the inverse statement is also true, i.e., if no run fires or marks x and y, then both are in symmetric conflict. This is however not valid in contextual ONs. Figure 10 (b) shows an ON where no run fires both events e_1 and e_2 but they are not in symmetric conflict. They actually form a cycle of *asymmetric conflict*, as we discuss now.

An asymmetric conflict can be thought of as a scheduling constraint: if two events e and e' *both occur* in a run σ of O and $e \nearrow e'$, then e must occur first — firing e' prevents e from firing. The definition of $\nearrow$, in p. 18, relates e and e' in three possible ways, let us see why this intuitive understanding of $\nearrow$ holds in all the three cases:

1. If (4) holds between e and e', then $e < e'$ and the intuition is trivially true.
2. If (5) holds, e' consumes some condition c read by e. Since c cannot be marked again once consumed, e needs to read before e' consumes. Notice that (5) does not force by itself to fire e in every run that fires e', in contrast to (4). Only *if both* events fire it forces an order of occurrence. For instance, in Fig. 10 (a), we have $e_1 \nearrow e_2$. If both e_1 and e_2 fire, then surely e_1 fires first. But firing e_2 on some run does not, itself, imply that e_1 fires, e_2 can fire alone.
3. As for (6), the intuition vacuously holds: e, e' cannot both occur.

If we think of asymmetric conflict as a scheduling constraint, it should be clear that any set of events containing a cycle of asymmetric conflict can never be entirely contained in a run — otherwise any event in the cycle would have to fire *before itself*. The simplest cycles we can find arise when $e \# e'$, where we have $e \nearrow e' \nearrow e$, a cycle of length two, due to (6). In ordinary ONs, every cycle of $\nearrow$ corresponds to a cycle of length two created by a symmetric conflict. Cycles of $\nearrow$ in (contextual) ONs may, however, have unbounded length: Fig. 10 shows two ONs with cycles of length two and three, that can be generalized to any length; even cycles of length two may be due to other causes than symmetric conflicts, as in Fig. 10 (b).

If three events e_1, e_2, e_3 satisfy $e_1 \nearrow e_2 \nearrow e_3$, then it is *not* true that every run that fires e_3 also fires e_1. It is not even true that, if both e_1 and e_3 fire, then e_1 fires before e_3. Both statements are, however, true if *all the three* fire

in the run. This illustrates a common pitfall about asymmetric conflicts. In contrast to the similarities between $<_i$ and $<$, not all intuitions behind $\nearrow$ are true for $\nearrow^*$.

Finally, notice that none of the intuitions given above about $<$, #, and $\nearrow$ is necessarily true for general c-nets. For instance, a run may mark twice one given place, and there may exist a run that fires two transitions in symmetric conflict.

A *configuration* of O is the set of events that a run of O fires. It can be defined formally as any set $\mathcal{C} \subseteq E$ satisfying

- $\mathcal{C}$ is causally closed; (17)
- $\nearrow_{\mathcal{C}}$ is acyclic; (18)
- for all $e \in \mathcal{C}$, the set $\{e' \in \mathcal{C} : e' \nearrow e\}$ is finite. (19)

Observe that any *finite* set of events automatically satisfies (19). For instance, $[e]$ is a configuration for all $e \in E$, which McMillan has called the *local configuration* [McM93b; McM95b]. We denote by $conf(O)$ the set of all configurations of O.

Let us check that this formalization meets the intuitive definition. Assume that $\sigma := e_1 e_2 \ldots$ is a finite or infinite run of O and let $\mathcal{C} := \{e_1, e_2, \ldots\}$ be the events in σ. By now, it should be clear that C is causally closed and induces no cycle in $\nearrow$. As all events e' in asymmetric conflict to e fire before e, and only finitely many events fire before e in σ, (19) also holds.

Conversely, if $\mathcal{C} := \{e_1, \ldots\}$ is a configuration, then any linear extension $\sigma := e_1, e_2, \ldots$ of $\nearrow_{\mathcal{C}}$, involving exactly the events in $\mathcal{C}$ is a run of O. This is because, first, $\nearrow$ is irreflexive in occurrence nets; second, $\nearrow^*_{\mathcal{C}}$ is a partial order, due to (18), so a linear extension σ of $\nearrow_{\mathcal{C}}$ always exists; third, σ is a run because any finite prefix $e_1, \ldots, e_{i-1}$ marks conditions enabling e_i, which can easily be shown using (17) to (19). In particular, (19) is necessary to ensure that only finitely many events in $\mathcal{C}$ need to be fired before e_i in σ. We call σ an *interleaving* of $\mathcal{C}$.

All interleavings of a *finite* configuration $\mathcal{C}$ mark the same marking of O. We call such marking the *cut* of $\mathcal{C}$, and define it as

$$cut(\mathcal{C}) := (\tilde{m}_0 \cup \mathcal{C}^\bullet) \setminus {}^\bullet\mathcal{C}.$$

Remark that the cut is defined only for finite configurations; similarly, the marking reached by a run is only defined for finite runs.

The *depth* of an event e, denoted by $depth(e)$, is the length of the longest $<$-chain of events in $[e]$, and can be inductively defined as

$$depth(e) := \begin{cases} 0 & \text{if } e \text{ enabled at } \tilde{m}_0 \\ 1 + \max_{e' < e} depth(e') & \text{otherwise} \end{cases} \tag{20}$$

The *depth of a configuration* $\mathcal{C}$ is the depth of the deepest event in the configuration, which

$$depth(\mathcal{C}) := \sup_{e \in \mathcal{C}} depth(e);$$

and similarly for the *depth of an occurrence net* O:

$$depth(O) := \sup_{e \in E} depth(e).$$

Given two configurations $\mathcal{C}$ and $\mathcal{C}'$, we say that $\mathcal{C}$ *evolves* to $\mathcal{C}'$, written $\mathcal{C} \sqsubseteq \mathcal{C}'$, iff

$$\mathcal{C} \subseteq \mathcal{C}' \text{ and } \neg(e' \nearrow e) \text{ for all } e \in \mathcal{C} \text{ and } e' \in \mathcal{C}' \setminus \mathcal{C}.$$

Note that $\sqsubseteq$ is not merely the subset relation between configurations (as it is in ordinary ONs). Intuitively, $\mathcal{C} \sqsubseteq \mathcal{C}'$ iff an interleaving of $\mathcal{C}$ can be

extended with all events in $\mathcal{C}' \setminus \mathcal{C}$ to form an interleaving of $\mathcal{C}'$. Note also that $\sqsubseteq$ is transitive. For instance, in Fig. 5 (e), which is an occurrence net, we have $\{c\} \sqsubseteq \{c,d\}$, and $\{c,a\} \sqsubseteq \{c,a,d\}$. But we also have that $\{c,d\} \not\sqsubseteq \{c,a,d\}$, an archetypical example of why $\sqsubseteq$ for c-nets is not just set inclusion.

An *extension* of a configuration $\mathcal{C}$ is any set $X \subseteq E$ of events such that

- $\mathcal{C} \cap X = \emptyset$, and
- $\mathcal{C} \cup X$ is a configuration.
- $\mathcal{C} \sqsubseteq \mathcal{C} \cup X$.

Configurations $\mathcal{C}$, $\mathcal{C}'$ are in *conflict*, written $\mathcal{C} \# \mathcal{C}'$, when there is no configuration $\mathcal{C}''$ satisfying $\mathcal{C} \sqsubseteq \mathcal{C}''$ and $\mathcal{C}' \sqsubseteq \mathcal{C}''$. Again in Fig. 5 (e) we have $\{c,d\} \# \{c,a,d\}$ and $\{c,a,d\} \# \{c,b,d\}$.

Remark 5. *The following statements are equivalent:*

1. $\mathcal{C} \# \mathcal{C}'$
2. $\mathcal{C} \not\sqsubseteq \mathcal{C} \cup \mathcal{C}'$ *or* $\mathcal{C}' \not\sqsubseteq \mathcal{C} \cup \mathcal{C}'$
3. *There is* $e \in \mathcal{C}$ *and* $e' \in \mathcal{C}' \setminus \mathcal{C}$ *such that* $e' \nearrow e$, *or the symmetric condition holds.*

Lemma 2. *Let* $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3 \in conf(O)$ *be configurations. We have:*

1. *If* $\neg \mathcal{C}_1 \# \mathcal{C}_2$, *then* $\mathcal{C}_1 \cup \mathcal{C}_2$ *is a configuration.*
2. *If* $\neg \mathcal{C}_1 \# \mathcal{C}_2$ *and* $\neg \mathcal{C}_2 \# \mathcal{C}_3$ *and* $\neg \mathcal{C}_1 \# \mathcal{C}_3$, *then* $\neg(\mathcal{C}_1 \# (\mathcal{C}_2 \cup \mathcal{C}_3))$.
3. *If* $\mathcal{C}_1 \# \mathcal{C}_2$ *and* $\mathcal{C}_1 \sqsubseteq \mathcal{C}_3$, *then* $\mathcal{C}_3 \# \mathcal{C}_2$.

Proof. We show each statement independently:

1. We check that (17) to (19) hold on $\mathcal{C}_1 \cup \mathcal{C}_2$. Clearly it is causally closed, as both $\mathcal{C}_1$ and $\mathcal{C}_2$ are. $\mathcal{C}_1 \cup \mathcal{C}_2$ satisfies (19) because for all $e \in \mathcal{C}_1 \cup \mathcal{C}_2$, all events $e' \in \mathcal{C}_1 \cup \mathcal{C}_2$ such that $e' \nearrow e$ need to be such that $e' < e$. Indeed, $e \# e'$ or $\underline{e'} \cap {}^\bullet e \neq \emptyset$ cannot hold because they entail $\mathcal{C}_1 \# \mathcal{C}_2$. As a result, (19) holds just because it holds independently for $\mathcal{C}_1$ and $\mathcal{C}_2$. Similarly, the existence of a cycle in $\nearrow_{\mathcal{C}_1 \cup \mathcal{C}_2}$ entails $\mathcal{C}_1 \# \mathcal{C}_2$, so (18) holds as well.
2. We know that $\mathcal{C}_2 \cup \mathcal{C}_3$ is a configuration. For a proof by contradiction, assume that $\mathcal{C}_1 \# (\mathcal{C}_2 \cup \mathcal{C}_3)$ holds. Two cases are possible:
   a) There exist $e_1 \in \mathcal{C}_1$ and $e_2 \in (\mathcal{C}_2 \cup \mathcal{C}_3) \setminus \mathcal{C}_1$ satisfying $e_2 \nearrow e_1$. Then either $\mathcal{C}_1 \# \mathcal{C}_2$ or $\mathcal{C}_1 \# \mathcal{C}_3$, a contradiction in any case.
   b) There exist $e_1 \in \mathcal{C}_2 \cup \mathcal{C}_3$ and $e_2 \in \mathcal{C}_1 \setminus (\mathcal{C}_2 \cup \mathcal{C}_3)$ satisfying $e_2 \nearrow e_1$. Analogous argument.
3. If $\mathcal{C}_1 \# \mathcal{C}_2$, then there is no configuration $\mathcal{C}$ such that $\mathcal{C}_1 \sqsubseteq \mathcal{C}$ and $\mathcal{C}_2 \sqsubseteq \mathcal{C}$. So if one assumes that $\neg \mathcal{C}_3 \# \mathcal{C}_2$, then $\mathcal{C}_3 \cup \mathcal{C}_2$ is a configuration. Then taking $\mathcal{C} := \mathcal{C}_3 \cup \mathcal{C}_2$ we have, using Rmk. 5 and transitivity of $\sqsubseteq$, that $\mathcal{C}_1 \sqsubseteq \mathcal{C}_3 \sqsubseteq \mathcal{C}$ and $\mathcal{C}_2 \sqsubseteq \mathcal{C}$, a contradiction to $\mathcal{C}_1 \# \mathcal{C}_2$. So necessarily $\mathcal{C}_3 \# \mathcal{C}_2$ holds. □

We finish this section with an important definition. Given occurrence nets O and $O' := \langle B', E', G', D', \tilde{m}'_0 \rangle$, we say that O is a *prefix* of O', and write $O \preceq O'$, iff O satisfies:

- $E \subseteq E'$ and E is causally closed in O';
- $B = \tilde{m}'_0 \cup E^\bullet$;
- G and D are the restrictions of G' and D' to $B \cup E$.

Note that O completely determined by its set of events E, which can be any causally closed subset of E'.

2.4.2 *Branching Processes*

The unfolding of a c-net N is an occurrence net O equipped with a homomorphism from O to N that is obtained by means of unrolling the *cycles* of N. Perhaps the most intuitive definition we can give is the following.

The *unfolding* of $N := \langle P, T, F, C, m_0 \rangle$, denoted by $\mathcal{U}_N$, is a labelled occurrence net $\langle O, h \rangle$, where $O := \langle \tilde{B}, \tilde{E}, \tilde{G}, \tilde{D}, \tilde{m}_0 \rangle$ is an occurrence net and $h \colon \tilde{B} \cup \tilde{E} \to P \cup T$ is a homomorphism from O to N, defined by the following inductive rules:

$$\frac{p \in m_0}{c := \langle \bot, p \rangle \in \tilde{B} \qquad h(c) := p \qquad c \in \tilde{m}_0} \textsc{Ini}$$

$$\frac{t \in T \qquad X, Y \subseteq \tilde{B} \qquad \mathbf{h}(X) = {}^\bullet t \qquad \mathbf{h}(Y) = \underline{t} \qquad X \cup Y \text{ is coverable}}{e := \langle X, Y, t \rangle \in \tilde{E} \qquad {}^\bullet e := X \qquad \underline{e} := Y \qquad h(e) := t} \textsc{Ev}$$

$$\frac{e \in \tilde{E} \qquad h(e) = t \qquad t^\bullet = \{p_1, \ldots, p_n\}}{c_i := \langle e, p_i \rangle \in \tilde{B} \qquad e^\bullet := \{c_1, \ldots, c_n\} \qquad h(c_i) := p_i} \textsc{Cond}$$

Although $\mathcal{U}_N$ is the pair $\langle O, h \rangle$, for simplicity we often identify $\mathcal{U}_N$ and O.

The reader familiar with the unfolding construction of *ordinary* nets will notice that the only difference between this construction and the one for ordinary nets (e.g., Definition. 3.5 in [EH08]) is that here, in the rule Ev, the set Y plays the role of context for an event, and we ask Y to be coverable. Remaining elements of the definition are standard.

In this definition, the nodes of $\mathcal{U}_N$ are assigned unique, canonical names. Every condition c has the form $\langle e, p \rangle$, where p is a place of N and e is the single event that produces c. If e is $\bot$, then c has been constructed by the rule Ini, p is initially marked in N, and so is c in $\mathcal{U}_N$. If e is an event in $\tilde{E}$, then c has been constructed by the rule Cond. The unique names guarantee that *exactly one* condition c satisfying $h(c) = p$ is added for each place $p \in h(e)^\bullet$, and also that no other event produces c.

Events are of the form $\langle X, Y, t \rangle$, where $t \in T$ is a transition of N and X, Y are sets of conditions h-labelled by the preset and context of t, respectively. All events are appended by the rule Ev, which guarantees that e is occurs in some run when asking that $X \cup Y$ is coverable. Although X, Y, ${}^\bullet t$, and $\underline{t}$ are sets, we remark that the premises $\mathbf{h}(X) = {}^\bullet t$ and $\mathbf{h}(Y) = \underline{t}$ interpret them as multisets. In other words, for every $p \in {}^\bullet t$, *exactly one* c with with $h(c) = p$ is to be present in X, and similarly for $\underline{t}$. Finally observe that different events have either different labels or different presets or different contexts.

Figure 11 shows a prefix of the unfolding for the net shown in Fig. 7. Events and conditions are given explicit names, and the labelling h is given in parenthesis. Condition c_1 would be constructed by the above rules as $\langle \bot, p_0 \rangle$; event e_5 we would be $\langle \{c_3\}, \{c_4\}, w_1 \rangle$. Now consider transition s_1 in Fig. 7, it consumes the place p_1 and reads q_0. There are two occurrences of s_1 in this unfolding prefix, e_3 and e_9, and both consume the same occurrence of p_1, namely c_3. Observe, for instance, that there is no occurrence of s_1 consuming c_{15} and reading c_9, and this is because such conditions are not coverable. Any run that marks c_{15} and c_9 needs to fire e_3 and e_4, but $[e_3] \cup [e_4]$ contains a cycle of asymmetric conflict: $e_3 \nearrow e_2 \nearrow e_4 \nearrow e_1 \nearrow e_3$.

We briefly argue that O and h are indeed an ON and a homomorphism. That O satisfies (13), (14) and (16) is immediate after the definition; that $\nearrow_{[e]}$ is acyclic, (15), can easily be shown by induction on the structure of O. As for h, it is immediate to show it satisfies (10) and (12). Condition (11) asks

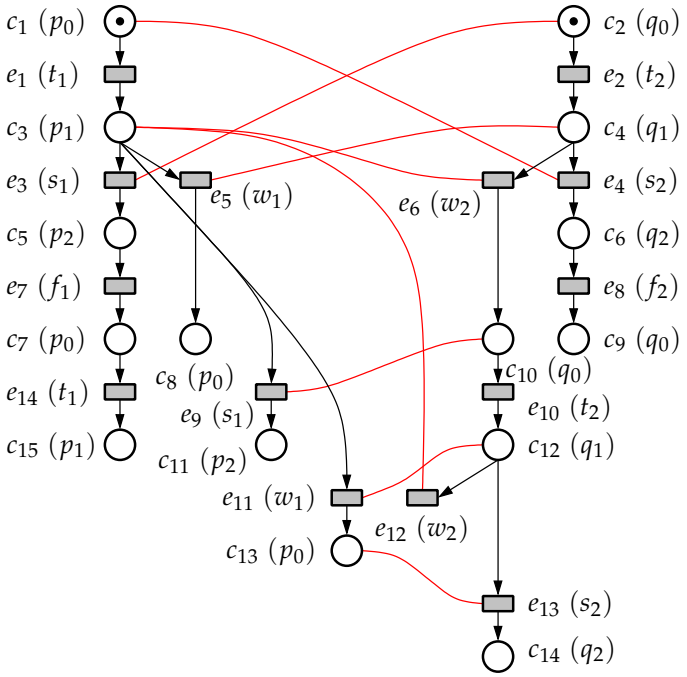


Figure 11: Unfolding prefix of the c-net in Fig. 7; the labelling is given in parenthesis.

that h acts as a bijection between $\tilde{m}_0$ and m_0 and holds because m_0 is safe, as we assumed in (9). Relaxing this is possible, at the cost of some impractical notation in the rule INI.

A *branching process* of N is a labelled occurrence net $\mathcal{P} := \langle O', h' \rangle$, where O' is an occurrence net and h' is a homomorphism from O' to N satisfying that for all events $e, e' \in O'$,

$$h'(e) = h'(e') \text{ and } {}^\bullet e = {}^\bullet e' \text{ and } \underline{e} = \underline{e'} \text{ imply } e = e' \tag{21}$$

This property intuitively means that $\mathcal{P}$ does not duplicate events: for any reachable marking m of $\mathcal{P}$ and any transition of N enabled at $h'(m)$, a single event e with $h'(e) = t$ will be found among those enabled by m.

The unfolding $\mathcal{U}_N$ is indeed a branching process, we have already argued why it satisfies the contrapositive of (21).

We lift the prefix order $\preceq$ on occurrence nets to branching processes. Given two branching processes $\mathcal{P}_1 := \langle O_1, h_1 \rangle$ and $\mathcal{P}_2 := \langle O_2, h_2 \rangle$, we say that $\mathcal{P}_1$ is a *prefix* of $\mathcal{P}_2$, written $\mathcal{P}_1 \preceq \mathcal{P}_2$, if $O_1 \preceq O_2$ and h_1 is h_2 restricted to O_1. Abusing of notation, we will often say that actually $\langle O_1, h_2 \rangle$ is a prefix of $\mathcal{P}_2$, which is safe due to h_1 being a restriction of h_2 to the nodes of O_1.

Remark 6. *$\langle O', h' \rangle$ is a branching process of N iff it is (isomorphic to) a prefix of $\mathcal{U}_N$.*

Proof. (Sketch) Obviously, every prefix of $\mathcal{U}_N$ is a branching process. The opposite direction can be shown by induction on the depth of O'. If $depth(O') = 0$, the branching process consist only of the initial marking, and by (11) it is (isomorphic to) the initial marking of $\mathcal{U}_N$. Otherwise O' have some maximal (w.r.t. causality) event e with $depth(e) \geq 1$. By the induction hypothesis $\mathcal{C} := [e] \setminus \{e\}$ is (isomorphic to) a configuration of $\mathcal{U}_N$, and $mark(\mathcal{C})$ is a reachable marking of N that enables $h(e)$. So e is (isomorphic to) an event of $\mathcal{U}_N$, and $\langle O', h' \rangle$ is (isomorphic to) a prefix of $\mathcal{U}_N$. □

In the light of Rmk. 6, branching processes will be also called *unfolding prefixes* or just *prefixes*. As we said, given any prefix $\langle O', h' \rangle$, we will also consider $\langle O', h \rangle$ a prefix of $\mathcal{U}_N$ (recall that h is the homomorphism in $\mathcal{U}_N$).

The unfolding $\mathcal{U}_N$ contains all behavioral information of the net. However, it is in general infinite and therefore of limited use for the purpose of automatic verification. Indeed, if N has at least one infinite run, $\mathcal{U}_N$ will contain at least one infinite configuration.

In general, we are interested in computing a *finite* prefix of the unfolding and using it to reason about the behavior of the net. If such a prefix is too small, it may not contain enough information. *Completeness criteria* encode which information is to be preserved in the prefix. Different criteria have been proposed in the literature to address different verification problems, see [KKV03] for a general discussion.

For bounded nets, however, there is always a finite unfolding prefix that contains sufficient information to characterize the reachable markings of the net (through the h-image of the prefix's reachable markings).

This is usually formalized as follows. First, for any finite configuration $\mathcal{C} \in \mathit{conf}(\mathcal{U}_N)$, one defines the *marking of* $\mathcal{C}$ *in* N as

$$\mathit{mark}(\mathcal{C}) := \mathbf{h}(\mathit{cut}(\mathcal{C})),$$

i.e., the marking of N that labels the marking $\mathit{cut}(\mathcal{C})$ reached by any interleaving of $\mathcal{C}$. Then, an unfolding prefix $\mathcal{P}$ is called *marking-complete* if for any marking m reachable in N there is a configuration $\mathcal{C} \in \mathit{conf}(\mathcal{P})$ with $\mathit{mark}(\mathcal{C}) = m$.

Marking-completeness often suffices for reachability-like problems. This will actually be the case for all applications investigated within this thesis, with the exception of the diagnosis problem in Ch. 6. In § 4.2, we discuss on the use of different criteria for checking deadlock-freeness.

Before finishing this section, let us remark some simple fact about the structure of $\mathcal{U}_N$. Consider two distinct, finite configurations $\mathcal{C}, \mathcal{C}'$ of $\mathcal{U}_N$ that reach the same marking $\mathit{mark}(\mathcal{C}) = \mathit{mark}(\mathcal{C}')$. Since they reach the same marking of N, the behavior that $\mathcal{U}_N$ displays after $\mathcal{C}$ and after $\mathcal{C}'$ must be *the same*. In other words, consider the two occurrence nets that one obtains after removing from $\mathcal{U}_N$ all nodes in $\mathcal{C}$ and $\tilde{m}_0 \cup \mathcal{C}^\bullet$, and similarly for $\mathcal{C}'$. Such occurrence nets, suitably labelled with h restricted to each of them, are isomorphic; let $f : \tilde{E} \to \tilde{E}$ be such isomorphism. Note that f also preserves the value of (the respective restrictions of) h. They are, in fact, isomorphic to the unfolding of N with the initial marking m_0 replaced by $\mathit{mark}(C)$.

As a result, for every *extension* I of C there is an isomorphic extension $I' := f(I)$ of C' reaching the same marking $\mathit{mark}(C \cup I) = \mathit{mark}(C' \cup I')$. For ordinary Petri nets, this is formalized, e.g., in Proposition 4.3 of [ERV02]. We will use extensively this simple fact in Ch. 6.

3 CONTEXTUAL UNFOLDING CONSTRUCTION

Theoretical foundations for constructing finite, complete unfolding prefixes of contextual nets have recently been proposed in [BCKS08]. While that work gave an abstract algorithm for the purpose, it left unresolved several important aspects, such as proposing an effective procedure for computing the possible extensions of the prefix. Moreover, it remained unclear whether the approach was useful in practice and which algorithms and data structures would be appropriate to implement it. In this chapter, we address these questions.

We provide one concrete method for computing contextual unfoldings, with a view to efficiency. We develop a concurrency relation on a notion of unfolding conditions enriched with histories, use it to characterize the possible extensions of the prefix, and provide an inductive characterization of this relation that lies at the heart of the efficient computation of prefix extensions. We include adequate orders in the framework of contextual unfoldings and compare our method the one independently proposed in [BBC+10].

This chapter includes and extends the results of [RSB11b; BBC+12].

3.1 INTRODUCTION

Not surprisingly, contextual unfoldings are constructed using procedures lifted from the theory of *ordinary* net unfoldings. We first recall how ordinary unfoldings are constructed, mostly for the reader unfamiliar to the concerned key literature [McM93b; McM95b; ERV02; ER99; KKV03; EH08].

Recall that our goal is producing a finite and marking-complete unfolding prefix for a given c-net.

It is often insightful to recall how this could be done if, instead of a Petri net, we are given a finite, directed graph G and one vertex v of it, and the goal is producing a labelled tree that represents all vertices of G reachable from v. Each node of the tree would be labelled by one vertex of the digraph. One would start with a tree that only contains the root, a node labelled by v. From there, one would proceed iteratively. For each unprocessed node n of the tree, labelled by u, one would extend the tree with a node labelled by u' if the digraph has an edge from u to u'. We call all such nodes *possible extensions*; after adding them all, n would be declared processed. Different orders to choose the next node to process are possible (depth-first, breath-first, etc.), we call them *strategies*.

If any vertex of G contained in a cycle is reachable from v, this construction would continue forever. But our goal is just visiting all reachable vertices, so we could declare *cutoff* any node of the tree whose label equals that of a node already present in the currently constructed part of the tree, and possible extensions would only be added for nodes that are not cutoffs. Since G is finite, this approach will always construct a finite tree that is *complete*, i.e., it contains a node labelled by u for every vertex u reachable from v.

This simple idea was first lifted to *ordinary* Petri nets by McMillan. His algorithm [McM93b; McM95b] constructs finite and complete unfolding prefixes for ordinary *bounded* Petri nets. As for digraphs, it starts with an un-

folding prefix containing only the initial marking, appropriately labelled, and iteratively computes and appends possible extensions to the prefix. The nodes of the tree before correspond here to markings of the prefix. Every marking of the prefix is labelled by a reachable marking on the net, and every transition enabled by it correspond to a possible extension. Thus finding possible extensions requires, conceptually, enumerating all reachable markings of the prefix and searching for the transitions of the net enabled by the labelling of those markings.

Again the construction would not stop if the net contains a reachable loop, and again, the algorithm designates some events as cutoff, preventing the prefix from being extended after them. An event is a cutoff if the marking of its local configuration (see § 2.4.1) is labelled by a marking that equals that of the local configuration of another event. Because the net is bounded, finitely many markings are reachable and the construction always stops with a finite prefix. As before, the algorithm relies on some strategy to chose the next event to extend the prefix. For digraphs, all strategies yield complete prefixes, but not for Petri nets, see [EKS07; EH08]. In [ERV02], a class of so called *adequate* strategies is presented that produce prefixes no larger than the reachability graph of the net.

The two key steps in McMillan's algorithm are thus (*i*) computing the possible extensions and (*ii*) deciding whether an event is a cutoff. Solutions for both were proposed by McMillan [McM93b; McM95b] and improved by others [ER99; ERV02; KKV03]. In particular, the use of *concurrency relations* to efficiently compute possible extension was proposed in [ER99].

Unfoldings for contextual nets can, conceptually, be constructed with McMillan's algorithm. For that, one first needs to appropriately lift the notion of cutoff and find an efficient algorithm for computing the extensions of the prefix.

A solution to the problem of choosing cutoffs in contextual unfoldings was given in [BCKS08], at the price of making the underlying theory notably more complicated. In particular, computing a complete prefix required to annotate every event e with a subset of its *histories*; roughly speaking, a history of e is a set of events that must precede e in a possible execution. How to effectively store these histories was left open.

Moreover, [BCKS08] did not address how to effectively compute possible extensions. It remained unclear whether contextual unfolding could be implemented with reasonable efficiency, and how. For safe nets, the interest of computing a complete contextual prefix was not evident from a practical point of view: while the prefix can be exponentially smaller than the complete prefix of the corresponding PR-encoding, the intermediate structure used to produce it has asymptotically the same size. More precisely, the number of histories in the contextual prefix of a safe c-net matches the number of events in the prefix of the PR-encoding (for general bounded nets, this is not the case).

In this chapter we address these open issues and propose an algorithmic solution for constructing finite, marking-complete contextual net unfoldings. Our solution is based on concurrency relations. In particular, we make the following contributions:

- We provide key elements to implement contextual-net unfoldings efficiently, including the necessary results to maintain a concurrency relation and exploit it for computing the possible extensions. We compare our approach to the one independently proposed in [BBC+10].

- We generalise the results in [BCKS08] in order to deal with a slight generalisation of the adequate orders from [ERV02]. Although not very surprising, this extension is quite relevant in practice as it can drastically reduce the size of the resulting prefixes.

We implemented both our approach and the one in [BBC+10], aiming for a competitive verification tool. The resulting implementation will be described in Ch. 7. Experiments with it suggest that efficient contextual unfolding is possible and performs better than the PR-encoding, even for safe nets.

The chapter is structured as follows. § 3.2 recalls the notion of history and cutoff, introducing a number of technical results that will be necessary later in the chapter. In § 3.3 we lift adequate orders from ordinary nets to c-nets, and show that they yield marking-complete prefixes. An abstract unfolding procedure is presented in § 3.4, and it is shown to construct marking-complete prefixes under given conditions. The rest of the chapter is devoted to presenting theoretical results enabling efficient algorithms for finding possible extensions of the prefix. Section 3.5 explains possible methods for this task; one of them is the development of concurrency relations, and we propose one in § 3.6. We next characterize possible extensions of the prefix with this relation, in § 3.7. The concurrency relation has to be maintained as the prefix grows, and § 3.8 provides theoretical results for this. Our characterization of possible extensions allows to construct the same possible extension multiple times. We refine this characterization in § 3.9. In §§ 3.10 and 3.11, we compare our approach to computing possible extensions and the one in [BBC+10]. We conclude in § 3.12.

3.2 PRUNING THE UNFOLDING

Finite, marking-complete unfolding prefixes are constructed iteratively. The two main tasks that the unfolder carries out during the construction are:

1. Identifying *possible extensions*, i.e., computing the events and conditions that shall be appended to the currently constructed prefix.
2. Deciding which events are the *cutoff* points, i.e., the events after which the construction does not need to continue because they do not contribute with new markings to the prefix.

In this section we focus on the second task. For ordinary nets, McMillan's algorithm identifies a set of so called *cutoff events*, and constructs the maximal prefix without cutoffs. Whether an event e is a cutoff depends on the marking $mark([e])$ reached by the local configuration $[e]$ and the events already present in the prefix. Every event is thus identified with a single marking of N.

However, for c-nets every event has, loosely speaking, different *local configurations* and it is not clear, a priori, which should be taken in order to associate a marking to the event. Such *local configurations* are formally called *histories*, they will be defined below.

Considering only the local configuration $[e]$ produces a finite prefix that unfortunately is not necessarily marking-complete, as observed in [VSY98], see Fig. 12. Instead, considering all histories [Win02] would yield a finite and marking-complete prefix, but an event may have infinitely many histories and the approach is not constructive.

A solution has been presented in [BCKS08], where the authors proposed to consider only a finite subset of useful histories, showing that this suffices

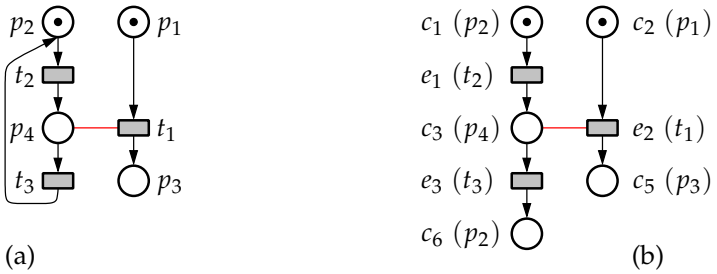


Figure 12: (a) A c-net; (b) a prefix of its unfolding. Applying McMillan's algorithm to (a) using only local configurations produces a prefix that is not marking-complete. In fact, it produces the prefix (b) without event e_3. Event e_3 would be considered a cutoff, as its local configuration reaches the initial marking. Then $\{e_1, e_2, e_3\}$ is *not* a configuration of the built prefix, and marking $\{p_2, p_3\}$ is unreachable on it.

to guarantee marking-completeness. This will be the approach followed in this chapter.

The notion of *history* can thus be seen as a suitable generalization of McMillan's local configurations [McM95b] to contextual unfoldings. Following [BCKS08], we now present the formal definition.

First, let us fix notation for the rest of the chapter. Unless otherwise stated we let $N := \langle P, T, F, C, m_0 \rangle$ be a finite, *bounded* c-net satisfying the general assumptions in p. 18. We let $\mathcal{U}_N := \langle \langle \tilde{B}, \tilde{E}, \tilde{G}, \tilde{D}, \tilde{m}_0 \rangle, h \rangle$ denote the full unfolding of N.

Definition 1 (history). *[BCKS08] Let $\mathcal{C}$ be a configuration of $\mathcal{U}_N$ and $e \in \mathcal{C}$ one of its events. We call the configuration $\mathcal{C}[\![e]\!]$, defined as*

$$\mathcal{C}[\![e]\!] := \{e' \in \mathcal{C} : e' \ (\nearrow_{\mathcal{C}})^* \ e\},$$

the history of e in $\mathcal{C}$. *Moreover, we denote by*

$$\mathit{Hist}(e) := \{\mathcal{C}[\![e]\!] : \mathcal{C} \in \mathit{conf}(\mathcal{U}_N) \wedge e \in \mathcal{C}\}$$

the set of histories of e.

The history of e in $\mathcal{C}$ is the set of events that fire before e in every possible firing sequence that involves all events in $\mathcal{C}$. Such history will necessarily contain $[e]$, but may contain other events of $\mathcal{C}$ as well. Notice that a history is always a configuration, as the definition claims: it is causally closed, free of cycles in $\nearrow$, and satisfies (19) because $\mathcal{C}$ already satisfies it.

For instance, let $\mathcal{C} := \{e_1, e_2, e_3\}$ be a configuration in Fig. 12 (b). The history of e_3 in $\mathcal{C}$ coincides with $\mathcal{C}$; the history of e_2 is $\{e_1, e_2\}$, which coincides with $[e_2]$. Consider now Fig. 11 and let $\mathcal{C} := \{e_2, e_4, e_8, e_1\}$ be a configuration there. The history of e_4 in $\mathcal{C}$ is the set $\{e_2, e_4\}$. Indeed, it does not hold that $e_1 \nearrow e_4$ or $e_8 \nearrow e_4$, so e_1 and e_8 must be excluded from the history. The history of e_1 in $\mathcal{C}$ is $\mathcal{C} \setminus \{e_8\}$, again because e_8 is not in asymmetric conflict to e_1.

The local configuration $[e]$ of any event e is always a history of e. While in Petri net unfoldings each event has exactly one history, a contextual unfolding may have multiple histories per event. For instance, in Fig. 12 (b), $\mathit{Hist}(e_3) = \{\{e_1, e_3\}, \{e_1, e_2, e_3\}\}$.

Events may even have infinitely many histories. Consider the unfolding of the net shown in Fig. 13 (a), a prefix of which is shown in Fig. 13 (b). The unfolding produces infinitely many copies of transition t_1 as events

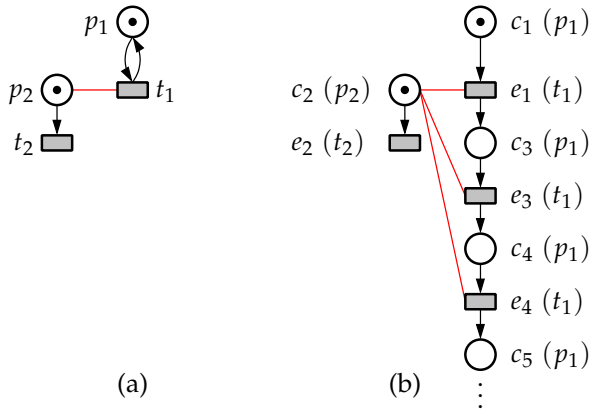


Figure 13: (a) a c-net; (b) a prefix of its unfolding. Event e_2 has infinitely many histories.

$e_1, e_3, e_4 \ldots$ All of them are in asymmetric conflict with e_2, whose histories are all infinitely many configurations that contain e_2.

Some technical results will be necessary for future developments in this chapter:

Remark 7. *Let $\mathcal{C}$ be a configuration of $\mathcal{U}_N$, and $e, e' \in \mathcal{C}$ events in $\mathcal{C}$. Then:*

1. $\mathcal{C}[\![e]\!] \sqsubseteq \mathcal{C}$; (22)
2. $\neg\,(\mathcal{C}[\![e]\!] \# \mathcal{C}[\![e']\!])$; (23)
3. *If* $e \nearrow_{\mathcal{C}}^{*} e'$, *then* $\mathcal{C}[\![e]\!] \sqsubseteq \mathcal{C}[\![e']\!]$ (24)
4. *For any* $A, B \subseteq \mathcal{C}$, *it holds that* $\neg\,(\bigcup_{\hat{e}\in A} \mathcal{C}[\![\hat{e}]\!]) \# (\bigcup_{\hat{e}\in B} \mathcal{C}[\![\hat{e}]\!])$ (25)

Proof. We show each statement independently:

1. This is a consequence of (25) and Rmk. 5 (2). If we take in (25) $A = \{e\}, B = \mathcal{C}$, we derive that $\neg\mathcal{C}[\![e]\!] \# \mathcal{C}$. This, together with Rmk. 5 (2) imply that $\mathcal{C}[\![e]\!] \sqsubseteq (\mathcal{C} \cup \mathcal{C}[\![e]\!])$.
2. Again, this follows after (25), taking $A = \{e\}$ and $B = \{e'\}$.
3. Let $H := \mathcal{C}[\![e]\!]$ and $H' := \mathcal{C}[\![e']\!]$. If $e \nearrow_{\mathcal{C}}^{*} e'$, then $e \in H'$. (24) is a consequence of (22) and the fact that $H = H'[\![e]\!]$, which in turn holds because for any $e'' \in \mathcal{C}$, $e'' \nearrow_{\mathcal{C}}^{*} e$ iff $e'' \nearrow_{H'}^{*} e$, which can be easily shown by induction.
4. Let $\mathcal{C}_1 := \bigcup_{\hat{e}\in A} \mathcal{C}[\![\hat{e}]\!]$ and $\mathcal{C}_2 := \bigcup_{\hat{e}\in B} \mathcal{C}[\![\hat{e}]\!]$. By contradiction, assume that $\mathcal{C}_1 \# \mathcal{C}_2$. Then by Rmk. 5 (3) we have two cases:
   a) There exist $e \in \mathcal{C}_1$ and $e' \in \mathcal{C}_2 \setminus \mathcal{C}_1$ such that $e' \nearrow e$. As $e \in \mathcal{C}_1$, there is some $\hat{e} \in A$ such that $e \nearrow_{\mathcal{C}}^{*} \hat{e}$; but then $e' \nearrow_{\mathcal{C}}^{*} \hat{e}$, which implies that $e' \in \mathcal{C}_1$ holds. This is a contradiction.
   b) There exist $e \in \mathcal{C}_2$ and $e' \in \mathcal{C}_1 \setminus \mathcal{C}_2$ satisfying $e' \nearrow e$. Analogous argument.

   In any case we reach a contradiction, so $\neg(\mathcal{C}_1 \# \mathcal{C}_2)$ holds. □

Given a configuration H that is also a history, the event e of which H is a history is uniquely determined. That event e is, by definition, the $\nearrow$-maximal event in H. However, it will be convenient to write the event together with the history, in the form of an enriched event.

Definition 2. *If $e \in \tilde{E}$ is an event and $H \in Hist(e)$ a history of e, a pair $\langle e, H\rangle$ is called* enriched event.

McMillan's algorithm labels every event with the marking of its local configuration, and we have already given the notion of history as a suitable generalization for c-net unfoldings. A second ingredient in the algorithm is an order on the configurations of the unfolding. It will be used to decide in which order events of $\mathcal{U}_N$ will be appended to the prefix.

Definition 3 (strategy). *A* strategy *on $\mathcal{U}_N$ is any strict partial order $\prec$ on the finite configurations of $\mathcal{U}_N$ verifying that for all finite $\mathcal{C}, \mathcal{C}' \in conf(\mathcal{U}_N)$,*

$$\text{if } \mathcal{C} \sqsubset \mathcal{C}' \text{ then } \mathcal{C} \prec \mathcal{C}'.$$

For the rest of this chapter we fix an arbitrary strategy $\prec$ on $\mathcal{U}_N$. Strategies not only fix the order in which the events of $\mathcal{U}_N$ are appended to the prefix. They parametrize the definition of the cutoff event set, i.e., different strategies identify different sets of cutoff events. For ordinary Petri nets, such identification is done through the local configuration of the event: to decide whether e is a cutoff, one needs to examine whether $\mathcal{C} \prec [e]$ holds for certain configurations $\mathcal{C}$ of the prefix. For c-nets, we need to compare not only $[e]$, but the possibly multiple histories of e to decide whether e should be included in the prefix. This amounts to lifting the notion of cutoff from events to histories or, in other words, to enriched events:

Definition 4 (feasible and cutoff enriched events). *An enriched event $\langle e, H\rangle$ is $\prec$-feasible if for all $e' \in H$ with $e' \neq e$, the enriched event $\langle e', H[\![e']\!]\rangle$ is not $\prec$-cutoff. A $\prec$-feasible enriched event $\langle e, H\rangle$ is $\prec$-cutoff if either:*

1. *$mark(H) = m_0$, or*
2. *there exists some $\prec$-feasible enriched event $\langle e', H'\rangle \in \mathcal{U}_N$, called* corresponding, *such that $H' \prec H$ and $mark(H) = mark(H')$.*

The idea of using enriched events as cutoffs is from [BCKS08], but our definition is more in line with [KKV03; EH08]. Intuitively, feasible events correspond to those explored by the unfolding algorithm. Cutoff events are those that will be discovered but not appended to the prefix, and feasible non-cutoff will be those appended to the unfolding prefix. We will come back to this in § 3.4.

Recall that we assumed the c-net N to be bounded. Due to this $reach(N)$ is a finite set, and we can show the following:

Lemma 3. *If $\langle e, H\rangle$ is $\prec$-feasible, then $depth(H) < |reach(N)|$.*

Proof. Assume that there is some $\prec$-feasible enriched event $\langle e, H\rangle$ that satisfies $depth(H) \geq |reach(N)|$. Let $e_1 <_i c_1 <_i e_2 <_i c_2 \ldots c_{n-1} <_i e_n = e$ be a chain of immediate causality in H, where $n \geq |reach(N)|$. Consider the sequence $m_0, mark(H[\![e_1]\!]), \ldots, mark(H[\![e_n]\!])$ of $n+1$ markings associated to those events, together N's initial marking m_0. By the pigeonhole principle, such a sequence repeats a marking at least twice. Let e_i be the event associated to the first repetition. Either $mark(H[\![e_i]\!]) = m_0$ and $\langle e_i, H[\![e_i]\!]\rangle$ is a $\prec$-cutoff, or there is some $j < i$ such that $mark(H[\![e_j]\!]) = mark(H[\![e_i]\!])$. Because $e_j < e_i$, by (24), we know that $H[\![e_j]\!] \sqsubseteq H[\![e_i]\!]$, which in turn implies that $H[\![e_j]\!] \prec H[\![e_i]\!]$. Then $\langle e_i, H[\![e_i]\!]\rangle$ is a $\prec$-cutoff as well, which is a contradiction to $\langle e, H\rangle$ being $\prec$-feasible. □

We can now show that the sets of $\prec$-feasible and $\prec$-cutoff events in Def. 4 are well defined and unique:

Lemma 4. *Definition 4 identifies exactly one finite set of $\prec$-feasible enriched events and one finite set of $\prec$-cutoff events, for any given strategy $\prec$.*

Event	Histories	Enriched event is	Corresponding
e_1	$\{e_1\}$	feasible, non-cutoff	
e_2	$\{e_1, e_2\}$	feasible, non-cutoff	
e_3	$\{e_1, e_3\}$	feasible, cutoff	$\tilde{m}_0$
	$\{e_1, e_2, e_3\}$	feasible, non-cutoff	

Table 1: Enriched events associated to the events of Fig. 12 (b).

Proof. There is finitely many $\prec$-feasible enriched events because their depth is bounded by $|reach(N)|$ and there are finitely many events in $\mathcal{U}_N$ of any given depth, and therefore finitely many histories using only those events.

Now we show that for any given strategy $\prec$, Def. 4 defines a unique set of $\prec$-feasible and $\prec$-cutoff enriched events. For a contradiction, let $\langle I, L\rangle$ be a collection of, respectively, feasible and cutoff events, and let $\langle I', L'\rangle$ be a different one. We show that $I = I'$, which in turn implies that $L = L'$. Because I and I are finite and different, the set $I \setminus I' \cup I' \setminus I$ is finite and non-empty. W.l.o.g., let $\langle e, H\rangle$ be an enriched event in $I \setminus I'$ such that H is $\prec$-minimal. Since $\langle e, H\rangle \notin I'$, there is some $e' \in H$ such that $\langle e', H[\![e']\!]\rangle$ is in L', also in I, but not in L; i.e., it is cutoff in the second collection, but feasible and non-cutoff in the first. Because it is cutoff in the second, there is some corresponding event in the second collection that is not feasible in the first — otherwise, $\langle e', H[\![e']\!]\rangle$ would be also cutoff in the first. That is, there exist some $\langle e'', H''\rangle \in I' \setminus I$ such that $H'' \prec H[\![e']\!]$ and $mark(H'') = mark(H[\![e']\!])$. But then $H'' \prec H[\![e']\!] \prec H$ and H is not the $\prec$-minimal history of an enriched event in the difference between I and I, as we assumed. □

Abusing the notation, we call an event $e \in \tilde{E}$ $\prec$-feasible if *some* enriched event associated to it is $\prec$-feasible, i.e. if there is some history $H \in Hist(e)$ such that $\langle e, H\rangle$ is $\prec$-feasible. Similarly, e is called $\prec$-cutoff if it is $\prec$-feasible and *all* its *feasible* histories correspond to $\prec$-cutoff enriched events, i.e., for all $H \in Hist(e)$, if $\langle e, H\rangle$ is $\prec$-feasible, then it is also $\prec$-cutoff.

The intuition behind this definition is that an event will be part of the constructed unfolding prefix only if one feasible, non-cutoff history for that event exists; and will be left outside in the opposite case, i.e., if all feasible histories are also cutoffs.

An important remark is that the set of $\prec$-feasible events is causally closed. To see this, recall that the local configuration $[e]$ of any event e is contained in any history $H \in Hist(e)$. Since any $\prec$-feasible event e has at least one feasible history H, by Def. 4 every causally related event $e' \in [e]$ has one feasible, *non-cutoff* history $H[\![e']\!]$. As a result, the $\prec$-feasible events identify a well-defined unfolding prefix:

Definition 5. *The* $\prec$-prefix *is the unique unfolding prefix* $\mathcal{P}_N^{\prec}$ *whose set of events is exactly the set of* $\prec$*-feasible events that are not* $\prec$*-cutoff.*

The $\prec$-prefix of an ordinary net was first defined in [KKV03], where it was called the *canonical prefix*.

We now present several examples. Consider the unfolding of Fig. 12 (a), a prefix of which is shown in Fig. 12 (b). Consider the *size* strategy $\prec_1$ defined as $\mathcal{C} \prec_1 \mathcal{C}'$ iff $|\mathcal{C}| < |\mathcal{C}'|$. Table 1 lists all enriched events associated to the events of the prefix in Fig. 12 (b), and specifies whether they are feasible or cutoffs w.r.t. $\prec_1$.

Event	Histories	Enriched event is	Corresponding
e_1	$\{e_1\}$	feasible, cutoff	$\tilde{m}_0$
e_3	$\{e_1, e_3\}$	unfeasible	
e_4	$\{e_1, e_3, e_4\}$	unfeasible	
e_2	$\{e_2\}$	feasible, non-cutoff	
	$\{e_1, e_2\}$	unfeasible	
	$\{e_1, e_3, e_2\}$	unfeasible	

Table 2: Some enriched events associated to the events of Fig. 13 (b).

According to the table, all the three events e_1, e_2, e_3 are feasible and not cutoff, as they have at least one feasible, non-cutoff history. They would therefore be part of the $\prec_1$-prefix. It is actually not difficult to see that the $\prec_1$-prefix only contains these three events. As for the unfolding of Fig. 13 (a), using the same strategy, we would have the enriched events listed in Table 2. In this case, all feasible enriched events associated to e_1 are cutoffs, so we declare e_1 cutoff; as for e_2, among its feasible histories, at least one is not a cutoff, so e_2 is feasible and not cutoff. The $\prec_1$-prefix in this case would only contain the event e_2.

3.3 ADEQUATE STRATEGIES AND COMPLETENESS

The shape and size of $\mathcal{P}_N^{\prec}$ depends on the particular choice of the strategy $\prec$. By Lemma 4 we know that $\mathcal{P}_N^{\prec}$ is finite for every strategy. In this section we show that for *adequate* strategies it is also marking-complete.

For contextual unfoldings, the only strategy that was known to generate marking-complete prefixes was the *size strategy*, defined in [BCKS08] as:

$$\mathcal{C} \prec_M \mathcal{C}' \text{ iff } |\mathcal{C}| < |\mathcal{C}'| \tag{26}$$

This condition was originally introduced by McMillan [McM93b] in his seminal paper on Petri net unfoldings. However, it is known that McMillan's order may create complete prefixes that are up to exponentially larger than the reachability graph [ERV02]. This is because $\prec_M$ is a partial order: multiple enriched events may lead to the same marking, but if they are incomparable (because their histories have the same size), then none of them is a cutoff.

Figure 14 illustrates a net for which this happens. Consider the two events labelled by t_1 and u_1. For both, the local configuration produces marking p_2 and has size 1. Then their local configurations are incomparable with $\prec_M$ and none of them is a cutoff. Similarly, any two events whose local configuration produces the marking p_i, for $2 \leq i \leq n+1$, has the same size and is incomparable with $\prec_M$. As a result, none of the exponentially many feasible events is a cutoff.

It is therefore preferable to replace McMillan's order by a suitable finer order, ideally a *total* order, in which case the resulting prefix will have at most as many events as there are reachable markings in the net (and usually far fewer).

For ordinary nets, this problem was addressed in [ERV02], where *adequate* strategies were introduced (called adequate orders in [ERV02]). Any adequate order will yield a complete prefix, and [ERV02] exhibits an adequate order that is total for safe nets. Below, in Def. 6, we present a slight

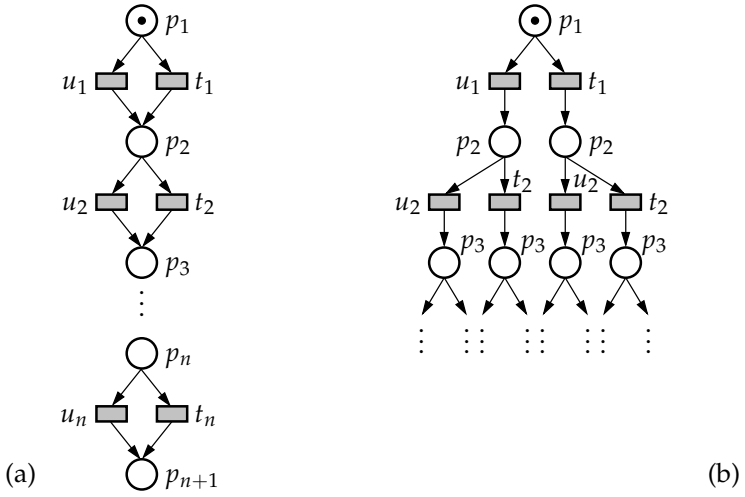


Figure 14: A c-net (a) whose unfolding (b) is exponentially larger than the set of reachable markings.

generalisation of the adequate orders from [ERV02] that is more suitable for c-nets. We then show that these orders also yield complete prefixes.

Definition 6 (adequate strategy)**.** *An strategy $\prec$ is* adequate *if it satisfies:*

1. *$\prec$ is well founded;*
2. *$\prec$ is preserved by finite extensions, i.e., if $\mathcal{C} \prec \mathcal{C}'$, and $mark(\mathcal{C}) = mark(\mathcal{C}')$, then for any finite extension X of $\mathcal{C}$ there is* some finite *extension X' of $\mathcal{C}'$ such that*
   a) *$\mathcal{C} \cup X \prec \mathcal{C}' \cup X'$, and*
   b) *$mark(\mathcal{C} \cup X) = mark(\mathcal{C} \cup X)$.*

The adequate strategies introduced in [ERV02] are slightly less general than in Def. 6. In particular, the second condition asked for the finite extension X' to be *isomorphic* to X. In [KKV03, Def. 2, item 3(b)], this condition was stated as in Def. 6, i.e., *any finite* extension suffices. Our notion of adequate strategy differs from the one in [KKV03] only on the fact that $\prec$ refines the evolution order $\sqsubseteq$ rather than subset inclusion $\subseteq$. Note that for Petri net unfoldings $\mathcal{C} \subset \mathcal{C}'$ implies $\mathcal{C} \sqsubset \mathcal{C}'$, hence the two notions coincide. However, for contextual unfoldings this is not the case; for instance, in Fig. 12 $\{e_1, e_3\} \not\sqsubset \{e_1, e_2, e_3\}$. For c-nets, Def. 6 is thus a slight generalisation of both [ERV02; KKV03].

Theorem 1. *$\mathcal{P}_N^{\prec}$ is marking-complete if $\prec$ is adequate.*

Proof. Let m be an arbitrary reachable marking of N. There exists a finite configuration $\mathcal{C}$ of $\mathcal{U}_N$ such that $mark(\mathcal{C}) = m$. Either $\mathcal{C}$ is a configuration of $\mathcal{P}_N^{\prec}$, and we are done, or $\mathcal{C}$ contains some $\prec$-cutoff event e. In the latter case, $\langle e, \mathcal{C}[\![e]\!]\rangle$ is $\prec$-cutoff and by Def. 4, its corresponding enriched event $\langle e', H'\rangle$ satisfies

$$H' \prec \mathcal{C}[\![e]\!], \text{ and}$$
$$mark(H') = mark(\mathcal{C}[\![e]\!]).$$

Since $\mathcal{C} \setminus \mathcal{C}[\![e]\!]$ is a finite extension of $\mathcal{C}[\![e]\!]$, by Def. 6 there is some finite extension X' of H' satisfying

$$H' \cup X' \prec \mathcal{C}, \text{ and}$$
$$mark(H' \cup X') = mark(\mathcal{C}).$$

We have found a finite configuration $C' := H' \cup X'$ of $\mathcal{U}_N$ that reaches the same marking m and is $\prec$-smaller. If C' is not yet a configuration of $\mathcal{P}_N^\prec$, it suffices to iterate this argument finitely many times until we get a configuration of $\mathcal{P}_N^\prec$ because $\prec$ is well-founded. □

Theorem 1 states that any strategy $\prec$ satisfying Def. 6 will yield a marking-complete $\prec$-prefix. It is easy to see that $\prec_M$, defined in (26) is adequate, but it may give rise to very large prefixes.

The essence of the problem is that $\prec_M$ is not a total order. Any total strategy orders any pair of $\prec$-feasible enriched events. Thus in any set of more than $|\mathit{reach}(N)|$ enriched events, the histories of two of them share the same marking and one is smaller than the other; therefore one of them is a $\prec$-cutoff. This shows that

Remark 8. *If $\prec$ is total, the number of $\prec$-feasible enriched events that are not $\prec$-cutoff is bounded by $|\mathit{reach}(N)|$.*

This remark entails that if $\prec$ is adequate and total, the $\prec$-prefix is marking-complete and not larger than the reachability graph of N. At this point, the natural question is whether we can find an adequate and total order. In [ERV02], the order $\prec_F$ is presented, and shown to be adequate and total for ordinary nets. Clearly, $\prec_F$ is adequate for contextual nets according to our definition, and showing that it is total is straight-forward.

3.4 UNFOLDING PROCEDURE

In previous sections we have defined $\mathcal{P}_N^\prec$ and shown that it is (*i*) finite, (*ii*) marking-complete if $\prec$ is adequate, and (*iii*) have no more events than N has reachable markings if $\prec$ is total. Following [BCKS08], in this section we present a procedure for constructing $\mathcal{P}_N^\prec$.

Histories had a prominent role in the definition of the $\prec$-prefix. Consequently, they are also key in its construction. To construct the $\prec$-prefix, one annotates events with a *subset* of their histories. This annotation is formalized by the function χ in our next definition:

Definition 7 (enriched prefix)**.** *[BCKS08] An* enriched prefix (EP) *is a tuple $\mathcal{E} := \langle O, h', \chi \rangle$ such that $\langle O, h' \rangle$ is an unfolding prefix, $O := \langle B, E, G, D, \tilde{m}_0 \rangle$, and $\chi \colon E \to 2^{2^E}$ satisfies that for all $e \in E$:*

- $\emptyset \neq \chi(e) \subseteq \mathit{Hist}(e)$, *and* (27)
- *for all $H \in \chi(e)$ and $e' \in H$, we have $H[\![e']\!] \in \chi(e')$.* (28)

Condition (27) simply asks that every event is labelled by a non-empty set of histories of that event. Remark that it does *not* require $\chi(e)$ to include all potentially infinitely many histories in $\mathit{Hist}(e)$.

The second condition (28) requires the labelling χ to be closed by the operation *'history of'*: if $\chi(e)$ contains one history H, then any event in H must be labelled with its history in H. This property will allow us in § 3.7 to construct EPs incrementally.

For the rest of this chapter, we fix an arbitrary enriched prefix $\mathcal{E} := \langle O, h, \chi \rangle$ where $O := \langle B, E, G, D, \tilde{m}_0 \rangle$. This is in addition to the notation already fixed: the net N, the unfolding $\mathcal{U}_N$, and the strategy $\prec$. We take the labelling h of $\mathcal{U}_N$ as the homomorphism for the underlying unfolding prefix $\langle O, h \rangle$ of $\mathcal{E}$, which is a safe abuse of notation, as we explained at p. 27.

Let us define some notation for EPs. An enriched event $\langle e, H \rangle$ *belongs* to $\mathcal{E}$, written $\langle e, H \rangle \in \mathcal{E}$, if $e \in E$ and $H \in \chi(e)$. A *configuration* of $\mathcal{E}$ is a

Algorithm 1 Contextual unfolding procedure

Input: A bounded c-net $N := \langle P, T, F, C, m_0 \rangle$ satisfying the general assumptions in p. 18 and a strategy $\prec$.
Output: An enriched prefix $\mathcal{E} = \langle O, h, \chi \rangle$, with $O = \langle B, E, G, D, \tilde{m}_0 \rangle$ and $\chi \colon E \to 2^{2^E}$.

```
Create n := |m_0| new conditions c_1, ..., c_n
Set m̃_0 := {c_1, ..., c_n}
Set h such that h(m̃_0) = m_0
χ := ∅
X := pe(E)
while X ≠ ∅ do
  Remove from X one enriched event ⟨e, H⟩ s.t. H is ≺-minimal
  if ⟨e, H⟩ is not ≺-cutoff then
    if e ∉ E then
      Append e to O
    end if
    Append H to χ(e)
    Create n := |h(e)•| new conditions c_1, ..., c_n
    Set e• := {c_1, ..., c_n}
    Set h such that h(e•) = h(e)•
    X = pe(E)
  end if
end while
```

any configuration $\mathcal{C}$ of O satisfying $\mathcal{C}[\![e]\!] \in \chi(e)$ for all $e \in \mathcal{C}$. We denote by $\mathcal{C} \in \mathcal{E}$ that $\mathcal{C}$ is a configuration of $\mathcal{E}$. Finally, we call $\mathcal{E}$ *marking-complete* iff for every reachable marking m of N there exists a configuration $\mathcal{C} \in \mathcal{E}$ verifying $mark(\mathcal{C}) = m$.

Unfolding prefixes of ordinary Petri nets are built incrementally. Similarly, in [BCKS08] a complete contextual prefix is constructed by a saturation procedure that adds one enriched event at a time until there remains no addition that would contribute new markings. Such appended enriched events are called possible extensions:

Definition 8 (possible extension)**.** *An enriched event $\langle e, H \rangle$ is a* possible extension (PE) *of $\mathcal{E}$ if $\langle e, H \rangle \notin \mathcal{E}$ and $\langle e', H[\![e']\!] \rangle \in \mathcal{E}$ for all $e' \in H \setminus \{e\}$.*

We will denote by $pe(\mathcal{E})$ the set of possible extensions of $\mathcal{E}$. We can now present an algorithm for constructing $\mathcal{P}_N^{\prec}$. Algorithm 1 takes as input a bounded c-net N and an arbitrary strategy $\prec$. It builds an enriched prefix $\mathcal{E} := \langle O, h, \chi \rangle$ whose size and shape depends on the choice of $\prec$. The algorithm can thus be seen as a family of algorithms, one for each strategy.

The construction proceeds iteratively. O is initially set as a copy of N's initial marking: an unfolding prefix with only conditions. X is initialized to the PEs of such a prefix, enriched events associated to events that consume from the initial marking. All such enriched events are necessarily $\prec$-feasible.

The loop iteratively extends the prefix with one enriched event whose history is $\prec$-minimal among all the PEs of the prefix at that time. Observe that, due to the choice of possible extensions in Def. 8, property (28) remains invariantly true on the successive versions of $\mathcal{E}_N$ that the loop produces: (28) is clearly satisfied initially, when there are no events; and every addition of a

PE maintains the property. Similarly, since $\prec$-cutoff extensions are skipped, the prefix only contains feasible (enriched) events.

An specialization of Algorithm 1, where the arbitrary strategy taken as input was instead the size strategy $\prec_M$, was first introduced in [BCKS08]. That specialization was there shown to produce finite and marking-complete prefixes. Here we show that it always terminates, producing the $\prec$-prefix, when invoked with arbitrary strategies:

Theorem 2. *Algorithm 1 always terminates and produces a prefix $\langle O, h, \chi \rangle$ such that $\langle O, h \rangle = \mathcal{P}_N^\prec$ when invoked on a bounded c-net N and some strategy $\prec$.*

Proof. We first show that the algorithm only appends $\prec$-feasible enriched events to the prefix. The proof is by induction on the size of the history. Let $\langle e, H \rangle$ be the last enriched event that Algorithm 1 appended to the prefix. If $|H| = 1$, then it is $\prec$-feasible by definition. Assume that $|H| \geq 2$. All $\langle e', H[\![e']\!] \rangle$, where $e' \in H$, are already in the prefix and, by induction hypothesis, we can assume they are $\prec$-feasible. The algorithm never appends a $\prec$-cutoff enriched event, and as a result $\langle e, H \rangle$ is necessarily $\prec$-feasible.

We now show that the algorithm always terminates. We start by observing that any finite enriched prefix always has finitely many PEs. This is a consequence of the fact that every finite configuration of $\mathcal{U}_N$ enables finitely many events, which in turn holds due to (21). Now, any time $pe(\cdot)$ is called within the loop, an enriched event is added to the prefix. Such enriched event is $\prec$-feasible and there are finitely many of them. Therefore, Algorithm 1 ever adds finitely many enriched events to the set X. As a result, the while loop makes finitely many iterations.

Finally, we show that every $\prec$-feasible, non $\prec$-cutoff enriched event is eventually added to the prefix. We do it by induction on the depth of the event. Let $\langle e, H \rangle$ be $\prec$-feasible and not $\prec$-cutoff. Assume that $depth(e) = 1$. Then $H = \{e\}$, and $\langle e, H \rangle$ is included in X on the first call to $pe(\cdot)$, just before the loop starts. Since finitely many events are ever added to X, $\langle e, H \rangle$ will eventually be appended to the prefix. Now assume that $depth(e) > 2$. By induction hypothesis, assume that all $\langle e', H[\![e']\!] \rangle$ with $e' \in H$ have already been added to the prefix. Then $\langle e, H \rangle$ is by definition a PE and will be included in X on the next call to $pe(\cdot)$. Again, once an extension is in X it will eventually be extracted. Since $\langle e, H \rangle$ is not $\prec$-cutoff, it will also be appended to the prefix. □

It follows that Algorithm 1 constructs complete prefixes when invoked with adequate strategies.

Corollary 1. *Algorithm 1 constructs a finite, marking-complete unfolding prefix $\mathcal{P}_N^\prec$ when invoked on $\langle N, \prec \rangle$ if $\prec$ is adequate.*

3.5 POSSIBLE EXTENSIONS

In the previous section, we deliberately left unspecified the algorithm to compute PEs. In this section, we discuss two existing approaches to compute PEs for ordinary unfoldings, explaining the issues that prevent lifting one of them to contextual unfoldings. This will serve as introduction to the developments in the next section, where we address these issues.

The main computational problem of Algorithm 1 is to identify the PEs in each iteration. For ordinary net unfolders, which do not deal with histories, this requires identifying sets M of conditions such that $conc(M)$ and $h(M) =$

${}^\bullet t$ for some $t \in T$, cf. the inductive rule Ev on p. 26. Two approaches have been proposed to this respect for ordinary unfoldings:

1. *Backwards Exploration*. A given a set $M \subseteq B$ of conditions from the prefix is coverable iff there is no two conditions from M in symmetric conflict. The flow relation in an unfolding prefix is acyclic, so exploring $[M]$ in backwards direction, from M to the initial marking, suffices to determine whether $conc(M)$ holds.
   This approach uses a working list, initialized to M. Then it iteratively extracts a node from the list and visits its causal immediate predecessors. All unvisited predecessors are appended to the working list; if an event is visited for the second time, it is not included in the working list. If a condition is visited twice, then a symmetric conflict has been found, and the algorithm stops. If the working list is eventually empty and no conflict has been found, then M is coverable. This algorithm runs in linear time on the prefix.
2. *Concurrency Relations*. [ER99] For ordinary unfoldings, it is known that $conc(M)$ holds iff $conc(\{c, c'\})$ holds for all pairs $c, c' \in M$. Deciding whether M is coverable thus reduces to decide whether $\{c, c'\}$ is coverable for all such pairs. One can therefore construct a *binary concurrency relation* on conditions and use it for finding PEs. Moreover, this binary relation can be computed efficiently and incrementally during prefix construction. This idea is exploited by existing tools such as Mole [Sch] or Punf [Kho]. The latter unfolder can also employ backwards exploration.

Turning now the attention to c-net unfoldings, the first method can be applied seamlessly to contextual prefix construction. The set M is, in this case, coverable iff the set of events in $[M]$ does not induce a cycle of asymmetric conflict. This can be easily detected by a depth-first search on $[M]$. One would explore the edges asymmetric conflict backwards, stopping as soon as a *back edge* is found, which indicates the existence of a cycle.

However, applying the second method to contextual unfolding entails some difficulties. Roughly speaking, these come from the fact that the above statement about $conc(\cdot)$ and *binary* concurrency relations is invalid for contextual unfoldings. Consider again the net shown in Fig. 10 (a) on p. 23, which is identical to its unfolding, and the set $M := \{c_4, c_5, c_6\}$. Clearly, M is not coverable, but all the elements of M are pairwise coverable. For instance c_4, c_5 may be covered by firing e_1, then e_2. In fact, $[\{c_4, c_5, c_6\}] = \{e_1, e_2, e_3\}$ cannot be fired in the same run because $e_1 \nearrow e_2 \nearrow e_3 \nearrow e_1$, i.e., it includes a cycle of asymmetric-conflict and it is not a configuration.

3.6 A CONCURRENCY RELATION FOR EFFICIENT CONSTRUCTION

In the following, we introduce a binary relation for c-net unfoldings in which pairwise concurrency does imply reachability of the whole set. This relation is defined on conditions enriched with histories.

Definition 9 (histories for conditions)**.** *Let $c \in \tilde{B}$ be a condition.*

- *A* generating history *of c is $\emptyset$ if $c \in \widehat{m}_0$, or $H \in Hist(e)$, where $\{e\} = {}^\bullet c$.*
- *A* reading history *of c is any $H \in Hist(e)$ such that $e \in \underline{c}$.*
- *A* compound history *of c is any configuration $H_1 \cup H_2$, where H_1 and H_2 are histories of c verifying $\neg(H_1 \# H_2)$.*

In general a history *of c is any of the three preceding kinds of histories.*

In words, for a condition c, not belonging to the initial marking, a generating history is any history of the unique event producing c. When c is in the initial marking it has only an empty generating history. A reading history is any history of the events reading c. Compound histories are conflict-free combinations of generating and (possibly multiple) reading histories.

If H is a history of c, we call $\langle c, H\rangle$ an *enriched condition (EC)*, referred to as generating, reading, or compound condition, according to H. Our notation for ECs is similar to that for enriched events. We say that $\langle c, H\rangle \in \mathcal{E}$ if for all $e \in ({}^\bullet c \cup \underline{c}) \cap H$ it holds that $H[\![e]\!] \in \chi(e)$, i.e., H is built from histories in χ. The mapping h is extended to enriched events and conditions by $h(\langle e, H\rangle) := h(e)$ and $h(\langle c, H\rangle) := h(c)$.

Definition 10 (concurrency for ECs)**.** *Two ECs $\langle c, H\rangle, \langle c', H'\rangle$ are said to be* concurrent*, written $\langle c, H\rangle \parallel \langle c', H'\rangle$, iff*

$$\neg(H \# H') \text{ and } c, c' \in cut(H \cup H'). \tag{29}$$

To illustrate the definition, we give some examples from Fig. 12.

- $\langle c_1, \emptyset\rangle \not\parallel \langle c_3, \{e_1\}\rangle$ because $c_1 \notin cut(\{e_1\})$;
- $\langle c_6, \{e_1, e_3\}\rangle \not\parallel \langle c_5, \{e_1, e_2\}\rangle$ because $\{e_1, e_3\} \# \{e_1, e_2\}$;
- $\langle c_6, \{e_1, e_3\}\rangle \not\parallel \langle c_6, \{e_1, e_2, e_3\}\rangle$ because $\{e_1, e_3\} \# \{e_1, e_2, e_3\}$;
- $\langle c_1, \emptyset\rangle \parallel \langle c_2, \emptyset\rangle$;
- $\langle c_3, \{e_1\}\rangle \parallel \langle c_3, \{e_1, e_2\}\rangle$;

Relation $\parallel$ on ECs allows to recover full information about coverability of conditions, as it was possible for ordinary unfoldings. This is easy to see formally. Let $M := \{c_1, \ldots, c_n\} \subseteq \tilde{B}$ be a set of n conditions. Using Lemma 2, one can show that $conc(M)$ holds iff there exists n histories $H_1, \ldots, H_n$, one for each condition in M, such that

$$\langle c_i, H_i\rangle \parallel \langle c_j, H_j\rangle \text{ holds for } 1 \leq i < j \leq n,$$

i.e., the associated ECs are pairwise concurrent. In other words, reachability of n ECs reduces to $\mathcal{O}(n^2)$ queries to a *binary*, as it was the case for ordinary nets. The price to pay now is that we need to keep track of histories for conditions.

Remark 9. *For ECs $\rho := \langle c, H\rangle$ and $\rho' := \langle c', H'\rangle$, the statement $\rho \parallel \rho'$ is equivalent to the conjunction of the next four statements:*

1. $\neg(\exists e_1 \in H, \exists e_2 \in H' \setminus H, e_2 \nearrow e_1)$
2. $\neg(\exists e_1 \in H', \exists e_2 \in H \setminus H', e_2 \nearrow e_1)$
3. $\neg(\exists e \in H, c' \in {}^\bullet e)$
4. $\neg(\exists e \in H', c \in {}^\bullet e)$

The following Lemma 5 will be useful in the subsequent proofs. It essentially states that all distinct histories of one event are in conflict, and that, as a consequence, no two distinct generating ECs are concurrent.

Lemma 5. *Let H, H' be two histories of the same event e and $c \in e^\bullet$ a condition in its postset. The following statements are equivalent:*

1. $\neg(H \# H')$.
2. *There is a configuration $\mathcal{C}$ such that $H, H' \sqsubseteq \mathcal{C}$.*
3. *Generating conditions $\rho := \langle c, H\rangle$ and $\rho' := \langle c, H'\rangle$ satisfy $\rho \parallel \rho'$.*
4. $H = H'$.

Proof. By definition of #, in p. 25, 1) and 2) are equivalent. Also by definition, 3) implies 1); and 1) implies 3) because there is no way in which either H or

H' can consume a condition in the preset of e, as e is $\nearrow$-maximal in any of its histories. Trivially 4) implies 1), so we only show that 1) implies 4).

Assume that 1) holds but 4) not, and that $H \setminus H'$ contains, w.l.o.g., some event e'. Since $e' \nearrow_H^* e$, it is easy to see that there exists some $e_1 \in H \setminus H'$ and $e_2 \in H'$ with $e_1 \nearrow e_2$, which by Rmk. 5 implies $H \# H'$, a contradiction. □

We are now ready to present how $\|$ can be used to compute PEs. The rest of the chapter is actually devoted to this point.

In § 3.7, we first characterize PEs in terms of $\|$ in a way suitable for constructing them algorithmically once $\|$ has been constructed. However, for each extension added to the prefix, new enriched conditions are appended, and one needs to update an stored copy of $\|$ with the newly appended conditions. We present in § 3.8 an inductive characterization of $\|$ that allows for this. Our characterization of PEs with $\|$ allows to construct the same PEs from different collections of enriched conditions. In § 3.9, we refine the characterization in order to obtain a unique decomposition for each PE. We then compare from a theoretical point of view our approach and another one independently developed in [BBC+10], discussing their benefits in § 3.10 and their memory usage in § 3.11.

Another approach for computing PEs, presented in [BBC+10], has been developed independently and at the same time as we worked on the method presented in following sections. Our approach was originally presented in [RSB11b], where we additionally discussed implementation details and gave an experimental evaluation. On a latter publication [BBC+12], the authors of both approaches made a coherent presentation of both solutions, comparing their differences and discussing their merits.

Following [BBC+12], in the remaining sections of this chapter we interleave the presentation of both approaches. Each approach proposes a different solution to each of the multiple subproblems that arise in producing an efficient algorithm for computing PEs. Our exposition alternates the presentation of both approaches for each such subproblem, which, we think, eases the comparison of both solutions.

Before we proceed, let us recall a notion that will be necessary in the sequel, that of an ancestor:

Definition 11 (ancestor). *[BBC+10] Let $\rho = \langle c, H\rangle$ be a reading history. The* ancestor *of ρ, denoted $\rho^\uparrow$, is the unique generating condition $\langle c, H'\rangle$ such that $H' \sqsubseteq H$.*

Note that indeed H' is uniquely determined due to Lemma 5.

3.7 CHARACTERIZING POSSIBLE EXTENSIONS CONCURRENCY

Our first step is to characterize PEs in terms of concurrent ECs; this is the goal of this section.

As we explained above, we discuss two ways of computing PEs. We call them, for reasons that we explain below, the *lazy* and *eager* approach. The first one was introduced in [BBC+10], while the second one is a contribution of us, originally presented in [RSB11b].

The lazy approach avoids constructing compound ECs, therefore reducing the number of ECs constructed by the unfolding algorithm. The eager approach does use compound conditions, which saves work when searching for PEs. In the light of these approaches, compound ECs can be seen as *pre-calculated coverability information* that the eager approach computes and

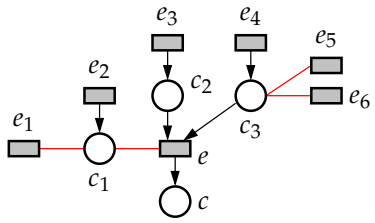


Figure 15: Predecessors w.r.t. asymmetric conflict of an event e.

stores before starting to search for PEs. The lazy approach computes equivalent information on the fly, every time that a PE is searched, and discards it once the search is done. As a result, the same information is possibly computed several times. Although this can incur computation overheads, it has the benefit of using the memory less eagerly. The eager approach computes ECs only once and avoids the overhead, but may store unnecessary compound ECs in memory.

3.7.1 *Lazy Approach*

The lazy approach [BBC+10] is based on the observation that the history associated with an event can be constructed by taking generating and reading histories for places in the pre-set, and generating histories for places in the context. The following Prop. 1 employs this facts to characterize enriched events in terms of concurrent collections of generating and reading histories. The proposition is a modified version of the Lemma 1 in [BBC+10].

Some remarks seem to be necessary at this point. Proposition 1 modifies the Lemma 1 of [BBC+10] in several aspects. First, we remove one technical condition of Lemma 1 to make Prop. 1 suitable to for fair comparison to our contribution, the eager approach, introduced in Prop. 2. Second, we provide a characterization while Lemma 1 in [BBC+10] only states half of this characterization: where Prop. 1 says '*iff*', Lemma 1 says '*only if*'. Third, although we gave an original and dedicated proof of Prop. 2 (published in [RSB11a], the long version of [RSB11b]), in [BBC+12] we rewrote it to reuse the proof of Prop. 1, and it will be this new, shorter proof the one we will give in § 3.7.2 for Prop. 2. For these three reasons, we present here a proof of Prop. 1.

Proposition 1 (possible extensions - lazy)**.** *A pair $\langle e, H\rangle$ is an enriched event iff there exist sets X_p, X_c of ECs such that*

- $h(X_p) = {}^\bullet t$ *and* $h(X_c) = \underline{t}$*, with* $h(e) = t$*;* (30)
- X_p *contains only generating or reading ECs;* (31)
- X_c *contains only generating ECs;* (32)
- $X_p \cup X_c$ *has exactly one generating EC for each* $c \in ({}^\bullet e \cup \underline{e})$*,* (33)
- $\rho \parallel \rho'$ *holds for all* $\rho, \rho' \in X_p \cup X_c$*;* (34)
- *finally,* $H = \{e\} \cup \bigcup_{\langle c, H'\rangle \in X_p \cup X_c} H'$*.* (35)

Proposition 1 allows to identify new possible extensions whenever a prefix is extended with new ECs. Compound conditions are avoided at the price of allowing X_p to contain, for every $c \in {}^\bullet e$, an arbitrary number of reading conditions.

To illustrate the meaning of X_p and X_c in Prop. 1, consider Fig. 15. To create a history for event e, X_p must contain generating histories for c_2 and c_3, i.e., histories of events e_3 and e_4. Optionally, X_p may contain one or more

reading histories of c_3, coming from e_5 or e_6 or both. As for X_c, it must contain a generating history of c_1, coming from e_2, but it cannot contain a reading history of c_1. In fact, note that e_1 is not an asymmetric-conflict predecessor of e, hence it is not included in any history of e.

Proof. We prove Prop. 1 in two steps. First assume that $\langle e, H\rangle$ is an enriched event. We construct sets X_p and X_c that satisfy (30) to (35). We start defining X_p. Property (31) requests X_p to contain generating or reading conditions. We define

$$X_p := X_1 \cup X_2,$$

where X_1 (resp. X_2) are sets of generating (resp. reading) conditions obtained from the generating (resp. reading) histories of ${}^\bullet e$ that extend to H:

$$X_1 := \{\langle c, \emptyset\rangle : c \in {}^\bullet e \cap \widehat{m}_0\} \cup \{\langle c, H[\![e']\!]\rangle : c \in {}^\bullet e \setminus \widehat{m}_0 \wedge \{e'\} = {}^\bullet c\}$$
$$X_2 := \{\langle c, H[\![e']\!]\rangle : c \in {}^\bullet e \wedge e' \in \underline{c} \cap H\}$$

For X_c, we can only take the generating histories that conditions in $\underline{e}$ induce in H, as (32) requests:

$$X_c := \{\langle c, \emptyset\rangle : c \in \underline{e} \cap \widehat{m}_0\} \cup \{\langle c, H[\![e']\!]\rangle : c \in \underline{e} \setminus \widehat{m}_0 \wedge \{e'\} = {}^\bullet c\}$$

This choice of X_p and X_c evidently satisfies properties (30) to (33). We now show that they also satisfy (34) and (35):

- *Property* (34). Let $\langle c_1, H_1\rangle, \langle c_2, H_2\rangle \in X_p \cup X_c$. By (23) we have that $\neg(H_1 \# H_2)$. Moreover, $c_1 \in \mathit{cut}(H_1)$ and $c_2 \in \mathit{cut}(H_2)$. Assume w.l.o.g. that $c_1 \notin \mathit{cut}(H_1 \cup H_2)$. Then there exists $e' \in H_2$ such that $c_1 \in {}^\bullet e'$. Since $c_1 \in {}^\bullet e \cup \underline{e}$, we have $e \nearrow e'$. Moreover, $e' \in H_2 \subseteq H$, where the configuration H is a history of e, therefore by definition $e' \nearrow_H^* e$. Then H contains an asymmetric conflict cycle, a contradiction.
- *Property* (35). Recall that any $e' \in H$ satisfies $e' \nearrow_H^* e$. So either $e' = e$ or there exists $e'' \in H$ such that $e' \nearrow_H^* e''$ and $e'' \nearrow e$. From all such e'', pick one that is maximal w.r.t. $<$. According to the definition of $\nearrow$ in (4) to (6), this leaves three cases: there is a condition c such that either
  - $c \in {}^\bullet e$ and $e'' \in {}^\bullet c$, or
  - $c \in {}^\bullet e$ and $e'' \in \underline{c}$, or
  - $c \in \underline{e}$ and $e'' \in {}^\bullet c$.

  Moreover, since $e'' \in H$, we use (22) to conclude that $H'' := H[\![e'']\!]$ is a history such that $H'' \sqsubseteq H$. Thus, in the first two cases, $\langle c, H''\rangle \in X_p$, and in the third, $\langle c, H''\rangle \in X_c$. Finally, $e' \in H''$ because $e' \in H$ and $e' \nearrow_H^* e''$.

This concludes one direction of the proof. As for the opposite direction, assume that there are sets X_p, X_c, and H fulfilling properties (30) to (35). We have to show that H is a history of e. To see this, it suffices to show that H is a configuration and that $H[\![e]\!]$ equals H.

- *H is causally closed.* Any H' such that $\langle e', H'\rangle \in X_p \cup X_c$ is causally closed. Moreover, the choice of X_p ensures that all causal predecessors of e are contained in H.
- *$\nearrow_H$ is acyclic.* By contradiction, assume that there exists a simple cycle

  $$e_1 \nearrow_H e_2 \nearrow_H \cdots \nearrow_H e_n \nearrow_H e_1,$$

  for some $n \geq 2$. Either e appears in the cycle or not. If it appears, then w.l.o.g. $e_1 = e$ and $e_2 \in H_2$, for some $\langle c_2, H_2\rangle \in X_p \cup X_c$. Now, $e \nearrow e_2$ implies, by (4) to (6), either

– $e < e_2$, or
– $\underline{e} \cap {}^\bullet e_2 \neq \emptyset$, or
– ${}^\bullet e \cap {}^\bullet e_2 \neq \emptyset$.

In all cases, H_2 consumes a token from some $c_1 \in {}^\bullet e \cup \underline{e}$. Due to (33), $X_p \cup X_c$ contains some tuple $\langle c_1, H_1 \rangle$, but $\langle c_1, H_1 \rangle \not\parallel \langle c_2, H_2 \rangle$ because H_2 consumes c_1, violating (34). Otherwise, if e does not appear in the cycle, then w.l.o.g. $e_2 \in H_2$ and $e_1 \in H_1 \setminus H_2$ for some $\langle c_1, H_1 \rangle, \langle c_2, H_2 \rangle \in X_p \cup X_c$, where $H_1 \neq H_2$. By Rmk. 5, we conclude that $H_1 \# H_2$, which violates (34).

- $H[\![e]\!] = H$. We need to show that $e' \nearrow_H^* e$ holds for each $e' \in H$. Recall that the elements $\langle c', H' \rangle \in X_p \cup X_c$ are chosen such that H' is either empty or a history of some e'' such that $e'' \nearrow e$. Thus, if $e' \neq e$, we have $e' \nearrow_H^* e'' \nearrow e$ for some suitable e'', and for $e' = e$ the condition holds trivially. □

3.7.2 *Eager Approach*

The eager approach, instead of attempting to combine generating and reading histories when computing a possible extension, explicitly produces all types of ECs, including compound ones. This means more ECs, but on the other hand less work when computing possible extensions.

Proposition 2 (possible extensions - eager)**.** *The pair $\langle e, H \rangle$ is an enriched event iff there exist sets X_p, X_c of ECs such that*

- $h(X_p) = {}^\bullet t$ *and* $h(X_c) = \underline{t}$, *with* $h(e) = t$; (36)
- X_p *contains arbitrary ECs;* (37)
- X_c *contains only generating ECs;* (38)
- $X_p \cup X_c$ *contains exactly one EC for each* $c \in ({}^\bullet e \cup \underline{e})$; (39)
- $\rho \parallel \rho'$ *holds for all* $\rho, \rho' \in X_p \cup X_c$; (40)
- *finally,* $H = \{e\} \cup \bigcup_{\langle c, H' \rangle \in X_p \cup X_c} H'$. (41)

Before we proceed with the proof, let us make some remarks. First, notice that $|X_p| = |{}^\bullet t|$ by (36) and (39), whereas no such bound exists in Prop. 1, where multiple reading ECs may be in X_p for the same condition. Like Prop. 1, Prop. 2 allows to identify new possible extensions upon addition of new ECs.

As an example, consider again Fig. 15. The set X_p must contain an arbitrary history for c_2 and c_3. Concretely, for c_2 we can take only a generating history, from e_3, while for c_3 we can use a generating history, from e_4, a reading history, from e_5, or a compound history, from the combination of reading histories of e_5 and e_6. Instead, X_c is still restricted to include generating histories only, in this case of c_1.

Second, one can establish a relation between possible extensions according to Prop. 1 and Prop. 2. That is, for each enriched event we can relate the set X_p identified by both propositions. Let X_p be a set satisfying the conditions in Prop. 1. We define

$$\mathsf{Eager}(X_p) := \{\langle c, \bigcup_{\langle c, H \rangle \in X_p} H \rangle : c \in {}^\bullet e\},$$

i.e., we merge all generating or reading histories associated to the same condition in ${}^\bullet e$, to produce generating, reading, or compound ECs. Remark that the elements of X_p are concurrent, due to (34), which is necessary to form compound histories, as Def. 9 requests.

On the other hand, if X_p is a set as in Prop. 2, let

$$\mathsf{Lazy}(X_p) := \{\langle c, H[\![e']\!]\rangle : \langle c, H\rangle \in X_p \wedge e' \in ({}^\bullet c \cup \underline{c}) \cap H\} \\ \cup \{\langle c, \emptyset\rangle : c \in {}^\bullet e \cap \widehat{m}_0\}.$$

In this case we perform the reverse operation, splitting reading or compound histories. Namely, from each reading EC $\rho \in X_p$ we get two ECs: ρ and $\rho^\uparrow$, i.e., a copy of the EC and its ancestor, a generating EC. Similarly, from each compound condition we get the elementary reading ECs forming the compound, together with their ancestors.

We are now ready to state the proof of Prop. 2. As explained at the beginning of § 3.7.1, recall that we prove here Prop. 2 via a reduction to the proof of Prop. 1. See [RSB11a], the long version of [RSB11b], for an original, dedicated proof of Prop. 2.

Proof. We shall prove that a collection of ECs satisfying the six conditions in Prop. 1 exists iff another collection exists, and satisfies the six properties in Prop. 2. We will define these collections, the reader may have guessed it, using $\mathsf{Eager}(\cdot)$ and $\mathsf{Lazy}(\cdot)$.

The proof has again two parts. From left to right, let X_p, X_c be a pair of sets of ECs as per Prop. 1. We define

$$X'_p := \mathsf{Eager}(X_p)$$

and prove that X'_p, X_c satisfy (36) to (41) in Prop. 2.

Conditions (36) to (38) and (41) are immediate. For property (39), observe that $\mathsf{Eager}(\cdot)$ produces a single history in X'_p out of all the histories X_p from the same condition, so the property holds.

As for property (40), it holds as a consequence of (34) and (24), in Rmk. 7. Let

$$\rho := \langle c, H_1 \cup \ldots \cup H_m\rangle,$$
$$\rho' := \langle c', H_{m+1}, \cup \ldots \cup H_n\rangle$$

be ECs in $X'_p \cup X_c$, such that

$$\langle c, H_i\rangle, \langle c', H_j\rangle \in X_p \cup X_c$$

for $1 \leq i \leq m < j \leq n$. By (34), the elements of $X_p \cup X_c$ are concurrent, so $c, c' \in cut(H_1 \cup \ldots \cup H_n)$. Moreover, defining suitable sets A, B in (24) it follows that

$$\neg(H_1 \cup \ldots \cup H_m) \# (H_{m+1} \cup \ldots \cup H_n),$$

which implies that $\rho \parallel \rho'$ holds.

This completes the left-to-right direction of the proof. For the opposite direction, let X_p, X_c be a pair of sets as per Prop. 2. Define

$$X'_p := \mathsf{Lazy}(X_p).$$

We show that X'_p, X_c satisfy all the six properties in Prop. 1.

The first four properties, (30) to (33) are immediate from the definition of X'_p, X_c. Property (34) follows from pairwise concurrency in X_p, X_c, Lemma 2 and Rmk. 7. As for (35), we need to show that every event $e' \in H$ is included in one of the elements of $X'_p \cup X_c$. For $e' = e$, this is immediate. Otherwise there exists a chain $e' \nearrow_H \ldots \nearrow_H e'' \nearrow_H e$ such that $e'' \in {}^\bullet({}^\bullet e) \cup {}^\bullet\underline{e} \cup {}^\bullet(\underline{e})$. The definition of X'_p and X_c implies that their union contains at least one tuple $\langle c, H[\![e'']\!]\rangle$, where $e' \in H[\![e'']\!]$. □

3.8 UPDATING THE CONCURRENCY RELATION

The previous section explains how to identify PEs: it suffices to find a set of concurrent ECs satisfying suitable technical conditions. We thus need a technique for efficiently computing the binary concurrency relation $\|$ on ECs.

This section discusses a method to do this incrementally, i.e., by extending $\|$ whenever the unfolding grows by the insertion of new ECs. Such extension will be possible thanks to the inductive characterization of $\|$ that Prop. 3 and Prop. 4 provide.

Again, it shall be useful to contrast our approach with that for ordinary Petri nets. Let e be the last event appended to the unfolding prefix of an ordinary Petri net; let $X := {}^\bullet e$ be its set of pre-conditions, and let c be a condition in the postset of e. One can show [ER99] that c is concurrent to

- any other condition in $e^\bullet$,
- any other condition c' from the prefix if c' is concurrent to *every* condition in X, and
- no other condition among those currently in the prefix.

This characterization is at the heart of efficient unfolding construction for ordinary nets. It gives rise to a dynamic programming algorithm to compute concurrency: whenever c is appended to the prefix and we want to compute whether it is concurrent to any other c', we *reuse* the concurrency information between c' and conditions in X to compute the answer.

This characterization is not correct for c-nets, even when lifted to ECs. Consider Fig. 12 (b) and let

$$\rho := \langle c_6, \{e_1, e_3\}\rangle$$
$$\rho' := \langle c_5, \{e_1, e_2\}\rangle.$$

be two ECs in that unfolding prefix. It should be clear that $\rho \not\| \rho'$, as $\{e_1, e_3\} \# \{e_1, e_2\}$, violating (29). Now, ρ is a generating EC, whose history $\{e_1, e_3\}$ was constructed, in terms of Prop. 2, using

$$\widehat{\rho} := \langle c_3, \{e_1\}\rangle.$$

However $\widehat{\rho} \| \rho'$, which contradicts the above characterization. Intuitively, the problem is that ρ' contains an event, e_2, that reads, without consuming, a condition in ${}^\bullet e_3$, namely c_3, and such event is not included in $\widehat{\rho}$. For computing the concurrency relation $\|$, we must therefore introduce an additional condition ensuring that such events are taken into account correctly.

The following two results show how to achieve this. Proposition 3 deals with generating and reading conditions, and Prop. 4 with compound conditions.

Proposition 3 (updating concurrency)**.** *Let $\langle e, H\rangle \in pe(\mathcal{E})$ be a PE of $\mathcal{E}$ and X_p, X_c its sets of ECs as per Prop. 1 or Prop. 2. Let*

$$Y_g := e^\bullet \times \{H\} \qquad \rho := \langle c, H\rangle \in Y_g \cup Y_r,$$
$$Y_r := \underline{e} \times \{H\} \qquad \rho' := \langle c', H'\rangle \in \mathcal{E}$$

be the sets of generating and reading ECs produced by $\langle e, H\rangle$ and two ECs, the second one, ρ', from $\mathcal{E}$. Then $\rho \| \rho'$ holds iff

$$\rho' \in Y_g \cup Y_r \ \vee \left(c' \notin {}^\bullet e \ \wedge\ {}^\bullet\underline{e} \cap H' \subseteq H \ \wedge\ \forall \widehat{\rho} \in X_p \cup X_c \colon (\widehat{\rho} \| \rho')\right).$$

Before we proceed with the proof, let us put forward the intuition. In the above statement $\langle e, H\rangle$ is a possible extension of the EP $\mathcal{E}$, which sooner

or later will be added to $\mathcal{E}$ by Algorithm 1. When that happens, all ECs in $Y_g \cup Y_r$ will be included in $\mathcal{E}$ as well, and one needs to compute whether they are concurrent to any other EC already present in $\mathcal{E}$. We thus characterize whether $\rho \parallel \rho'$ holds for any ρ in $Y_g \cup Y_r$ and any arbitrary ρ' in $\mathcal{E}$.

Proof. We prove both directions. From left to right, assume $\rho \parallel \rho'$ holds and $\rho' \notin Y_g \cup Y_r$ holds as well. Since $\neg(H \# H')$, there exists a configuration $\mathcal{C}$ such that $H \sqsubseteq \mathcal{C}$ and $H' \sqsubseteq \mathcal{C}$.

1. Clearly, $c' \notin {}^\bullet e$ is implied by $c' \in cut(H \cup H')$.
2. Let e'' be an event in $\underline{{}^\bullet e} \cap H'$. Then $e'' \nearrow e$. Since $\neg(H \# H')$, and due to Rmk. 5, e'' cannot be in $H' \setminus H$, so $e'' \in H$.
3. Let $\widehat{\rho} := \langle \widehat{c}, \widehat{H} \rangle \in X_p \cup X_c$ be an arbitrary EC used to construct $\langle e, H \rangle$. Then $\widehat{H} \sqsubseteq H \sqsubseteq \mathcal{C}$, therefore $\neg(\widehat{H} \# H')$. Moreover $c' \in cut(\widehat{H} \cup H')$, as otherwise there would exist $\widehat{e} \in \widehat{H}$ that consumes c', which would contradict $c' \in cut(H \cup H')$. It remains to show that $\widehat{c} \in cut(\widehat{H} \cup H')$. Let e' be the single event such that $H' \in Hist(e')$. For a proof by contradiction, assume that there is $e'' \in H'$ such that $\widehat{c} \in {}^\bullet e''$. We will show in the sequel that $e \notin H'$. This implies that $e \neq e'$, and also that either (5) or (6) holds between e and e'', so necessarily $e \nearrow e''$, with $e \in H \setminus H'$ and $e'' \in H'$. In turn, by Rmk. 5, this means that $H \# H'$, a contradiction. So we want to show that $e \notin H'$. We proceed by contradiction. Assume that $e \in H'$. Then either $H'[\![e]\!] = H$ or not. If $H'[\![e]\!] = H$, then by (22) $H \sqsubseteq H'$. But then because $\langle e', H' \rangle \in \mathcal{E}$ and using (28), this means that $\langle e, H \rangle \in \mathcal{E}$, a contradiction to the fact that $\langle e, H \rangle$ is a PE of $\mathcal{E}$. Otherwise, if $H'[\![e]\!] \neq H$, then Lemma 5 implies that $H'[\![e]\!] \# H$. Since $H'[\![e]\!] \sqsubseteq H'$, this implies that $H \# H'$, by Lemma 2. This is a contradiction to $\rho \parallel \rho'$.

This completes the left-to-right direction. For the opposite one, we show that if the right-hand side of Prop. 3 holds, then $\rho \parallel \rho'$ holds as well.

Suppose $\rho' \in Y_g \cup Y_r$. Then $H' = H$, so $\neg(H \# H')$. Moreover, since $c, c' \in \underline{e} \cup {}^\bullet e$, and H is a history for e, we have $c, c' \in cut(H)$. Therefore, $\rho \parallel \rho'$ as desired.

Let us now assume that the right-hand part of the disjunction holds and that $\rho \parallel \rho'$ does not hold. Then either $H \# H'$ or $c, c' \notin cut(H \cup H')$.

1. If $H \# H'$, then (i) either there exists $e_1 \in H$, $e_2 \in H' \setminus H$ with $e_2 \nearrow e_1$ or (ii) $e_1 \in H \setminus H'$, $e_2 \in H'$ with $e_1 \nearrow e_2$. In either case, $e_1 = e$ must hold, as if it were not the case, then e_1 would be in $H \setminus \{e\}$, and then then $\widehat{\rho} \not\parallel \rho'$ for some $\widehat{\rho} \in X_p \cup X_c$.
   (i) There are three cases for $e_2 \nearrow e$.
   - If $e_2 < e$, then $e_2 \in H$ because H is causally closed, which contradicts $e_2 \in H' \setminus H$.
   - If $\underline{e_2} \cap {}^\bullet e \neq \emptyset$, then again $e_2 \in H$ because $\underline{{}^\bullet e} \cap H' \subseteq H$.
   - If ${}^\bullet e_2 \cap {}^\bullet e \neq \emptyset$, then clearly $\rho' \not\parallel \widehat{\rho}$ for some $\widehat{\rho} \in X_p$.

   (ii) There are three cases for $e \nearrow e_2$.
   - If $e < e_2$, then H' consumes all tokens from ${}^\bullet e$, so ρ' is not concurrent with any element of X_p.
   - If $\underline{e} \cap {}^\bullet e_2 \neq \emptyset$, then ρ' is not concurrent with some element of X_c.
   - If ${}^\bullet e \cap {}^\bullet e_2 \neq \emptyset$, see (i).

2. If $c' \notin cut(H \cup H')$, then there exists $e_1 \in H$ with $c' \in {}^\bullet e_1$. If $e_1 \neq e$, we would get $\widehat{\rho} \not\parallel \rho'$ for some $\widehat{\rho} \in X_p \cup X_c$. But if $e_1 = e$, then $c' \in {}^\bullet e$, contradicting our assumption.
3. If $c \notin cut(H \cup H')$, then there exists $e_1 \in H'$ with $c \in {}^\bullet e_1$. If $\rho \in Y_g$, then H' consumes all of ${}^\bullet e$; if $\rho \in Y_r$, then H' consumes some element of $\underline{e}$. In either case, we get non-concurrency between ρ' and some element of $X_p \cup X_c$. □

As a complement to Prop. 3, the following result allows to compute the concurrency relation for compound conditions.

Proposition 4 (updating concurrency - compound)**.** *Let*

$$\rho_1 := \langle c, H_1 \rangle \qquad \rho_2 := \langle c, H_2 \rangle \qquad \rho := \langle c, H_1 \cup H_2 \rangle$$

be ECs verifying $\neg(H_1 \# H_2)$*, and* ρ *be a compound EC. For any EC* ρ' *it holds that*

$$\rho \parallel \rho' \;\; \textit{iff} \;\; \rho_1 \parallel \rho' \wedge \rho_2 \parallel \rho'$$

Proof. Let $H := H_1 \cup H_2$ be the history in ρ and let $\rho' := \langle c', H' \rangle$.

From left to right, assume, for a contradiction, that $\rho \parallel \rho'$ and w.l.o.g. $\rho_1 \not\parallel \rho'$. Then one of the four statements in Rmk. 9 must be false:

1. There is $e_1 \in H'$ and $e_2 \in H_1 \setminus H'$ verifying $e_2 \nearrow e_1$. As $e_2 \in H \setminus H'$, we have $H \# H'$, a contradiction to $\rho \parallel \rho'$.
2. There is $e_1 \in H_1$ and $e_2 \in H' \setminus H_1$ verifying $e_2 \nearrow e_1$. As $H_1 \subseteq H$, we have $e_1 \in H$. Regarding e_2, we have two cases: either $e_2 \notin H$ or $e_2 \in H$. Assuming the former immediately leads us to the contradiction $H \# H'$. Assuming $e_2 \in H = H_1 \cup H_2$ leads to $e_2 \in H_2 \setminus H_1$. In turn, this implies $H_1 \# H_2$, a contradiction to our hypothesis.
3. There is $e \in H_1$ such that $c' \in {}^\bullet e$. Then $e \in H$ and H also consumes c', a contradiction to $\rho \parallel \rho'$.
4. There is $e \in H'$ such that $c \in {}^\bullet e$. This is a contradiction to $\rho \parallel \rho'$.

From right to left, assume $\rho_1 \parallel \rho'$, $\rho_2 \parallel \rho'$, and by contradiction $\rho \not\parallel \rho'$. We consider the four cases of Rmk. 9:

1. There exist $e_1 \in H$ and $e_2 \in H' \setminus H$ verifying $e_2 \nearrow e_1$. Then either $e_1 \in H_1$, and $H_1 \# H'$ holds, or $e_1 \in H_2$ and $H_2 \# H'$ holds. In any case we reach a contradiction to our hypothesis.
2. There exist $e_1 \in H'$ and $e_2 \in H \setminus H'$ verifying $e_2 \nearrow e_1$. Same argument as before, regarding e_2 instead of e_1.
3. There exists $e \in H$ such that $c' \in {}^\bullet e$. Either $e \in H_1$ and $\neg(\rho_1 \parallel \rho')$ or $e \in H_2$ and $\neg(\rho_2 \parallel \rho')$. In any case we reach a contradiction to our hypothesis.
4. There exists $e \in H'$ such that $c \in {}^\bullet e$. This is a contradiction to $\rho_1 \parallel \rho'$. □

3.9 UNIQUE POSSIBLE EXTENSIONS

Section 3.7 shows how possible extensions are constructed, in both lazy and eager fashions. Essentially, a history H for an event is constructed by taking *generating* histories for the conditions in $\underline{e}$, while for the conditions in ${}^\bullet e$ one takes a generating history and optionally some *reading* histories. In the eager case, the latter are combined to one single *compound* history.

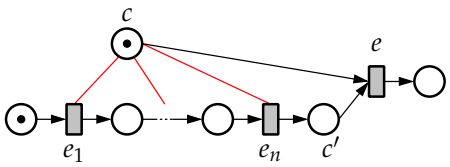


Figure 16: The characterization in § 3.7 allows multiple constructions of the enriched event $\langle e, \{e_1, \ldots, e_n, e\}\rangle$.

The optionality of the reading histories means that, in some cases, the same history H may be constructed in different ways, by combining different sets of ECs. Consider the unfolding in Fig. 16. Condition c has $n+1$ different reading histories: $H_0 := \emptyset$, $H_1 := \{e_1\}$, $\ldots$, $H_n := \{e_1, \ldots, e_n\}$, while c' has one single history H_n. Notice that we have $\langle c, H_i\rangle \parallel \langle c', H_n\rangle$ for all $i = 0, \ldots, n$. Thus, there exists a multitude of possibilities to construct the enriched event $\langle e, H_n \cup \{e\}\rangle$: there are 2^n collections satisfying Prop. 1 and $n+1$ collections for Prop. 2.

In the following, we discuss how to remove this ambiguity, i.e., how additional constraints can be inserted into Prop. 1 or Prop. 2 so that every tuple $\langle e, H\rangle$ can be obtained from a *unique* collection of ECs. Roughly, the idea is simple: if one element of $X_p \cup X_c$ contains an event e' that reads from $c \in {}^\bullet e$, then that event must be contained in a reading or compound condition for c included in X_p. In our example, this means we need to take *all* reading histories of c. That is, the unique collection associated to $\langle e, H_n \cup \{e\}\rangle$ in the lazy method would be the sets

$$X_c := \emptyset \tag{42}$$

$$X_p := \{\langle c', H_n\rangle, \langle c, \emptyset\rangle, \langle c, H_1\rangle, \ldots, \langle c, H_n\rangle\}. \tag{43}$$

Lazy would refuse any other pair of sets where X_p does not contain some $\langle c, H_i\rangle$, for $i = 1, \ldots, n$. Likewise, the eager method would construct a compound EC that includes all n reading histories of c, giving rise to the following associated sets:

$$X_c := \emptyset \tag{44}$$

$$X_p := \{\langle c', H_n\rangle, \langle c, H_1 \cup \ldots \cup H_n\rangle\}. \tag{45}$$

In both lazy and eager mode, this requires to compute an additional relationship between ECs. We recall how this is done in the lazy method and present a conceptually simpler solution for the eager method.

3.9.1 *Lazy Approach: Subsumption*

The lazy approach of [BBC+10] deals exclusively with generating and reading histories, and uses the notion of subsumption:

Definition 12 (subsumption). *[BBC+10] Let H' be a history of some e'. Let $\rho := \langle c, H\rangle$ be a generating or reading EC and $\rho' := \langle c', H'\rangle$ be a reading EC, with $c' \in \underline{e'}$. We say that ρ* subsumes *ρ', written $\rho \propto \rho'$ if $e \in H$, $c' \in cut(H)$, and $H' = H[\![e']\!]$.*

In other words, the subsuming condition ρ includes H' and reads but never consumes c'. For instance, in Fig. 16, we have that $\langle c', H_n\rangle \propto \langle c, H_i\rangle$, for all $i = 1, \ldots, n$.

A suitable closeness condition involving subsumption can help in modifying Prop. 1 to provide a unique characterization of PEs. Proposition 5

characterizes enriched events using all conditions (30) to (35) from Prop. 1, and adds one new condition, (46), ensuring that the pair of sets X_p, X_c is unique. When applied, e.g., to the enriched event $\langle e, H_n \cup \{e\}\rangle$ in Fig. 16, it only accepts X_p, X_c as defined in (42) and (43). Indeed, a set X_p which does not contain, e.g., $\langle c, H_1\rangle$ would be rejected by (46): $\langle c', H_n\rangle$ subsumes $\langle c, H_1\rangle$, and the ancestor $\langle c, H_1\rangle^\uparrow = \langle c, \emptyset\rangle$ is in X_p, due to (33). So $\langle c, H_1\rangle$ needs to be also in X_p. In [BBC+10], any pair of sets X_p, X_c satisfying (46) is said to be *subsumption-closed*.

This new condition, (46), is the technical condition that we removed from the Lemma 1 of [BBC+10] to make Prop. 1, see the beginning of § 3.7.1 for a detailed explanation. Thus Prop. 1 plus (46) (roughly) corresponds to Lemma 1 in [BBC+10]. The next proposition *additionally* states that the collections X_p, X_c are *unique* for each enriched event, which was not stated in [BBC+10], and provides a proof (no proof was given in [BBC+10] and we are not aware of any publication containing them, except [BBC+12]).

Proposition 5 (unique lazy extensions)**.** *The pair $\langle e, H\rangle$ is an enriched event iff there exist sets X_p, X_c of ECs satisfying (30) to (35) and additionally*

- *for any $\rho \in X_p \cup X_c$, if $\rho \propto \rho'$ and $\rho'^\uparrow \in X_p$, then $\rho' \in X_p$.* (46)

Moreover, for any enriched event $\langle e, H\rangle$ there exists exactly one pair of sets X_p, X_c satisfying (30) to (35) and (46).

Proof. The "iff" part follows almost directly from Prop. 1. We just need to observe that any collection that does not satisfy (46) can be made subsumption-closed by adding all subsumed ρ' according to (46). Observe that such additions never violate any condition in Prop. 1.

It remains to show the uniqueness of the pair X_p, X_c w.r.t. $\langle e, H\rangle$. Let X'_p, X'_c be another pair of sets of ECs satisfying (30) to (35) and (46) for $\langle e, H\rangle$. We show in the sequel that $X_p \subseteq X'_p$ and $X_c \subseteq X'_c$. Once done, by symmetry, both collections must coincide.

Let $\rho := \langle c, \widehat{H}\rangle \in X_p \cup X_c$ be an EC. We distinguish two cases. First, let ρ be a generating EC. By (33) there is another generating EC $\rho' = \langle c, \widehat{H}'\rangle \in X'_p \cup X'_c$. Because $\widehat{H}$ and $\widehat{H}'$ are either empty or histories of the same event in H, by (22), it holds that both evolve to H and so $\widehat{\rho} \parallel \widehat{\rho}'$. Then, by Lemma 5, $\rho = \rho' \in X'_p \cup X'_c$.

Second, let ρ be a reading EC, i.e., $c \in {}^\bullet e$ and $\widehat{H}$ is the history of some $\widehat{e} \in \underline{{}^\bullet e}$. Since $\widehat{e} \in H$, there is in $X'_p \cup X'_c$ some other $\rho' := \langle c', \widehat{H}'\rangle \in X'_p \cup X'_c$ that contains $\widehat{e}$, i.e., $\widehat{e} \in \widehat{H}'$. Of course, c and c' do not need to be the same condition. Now, clearly $c \in \mathit{cut}(\widehat{H}')$, as otherwise $\widehat{H}'$ would consume $c \in {}^\bullet e$. Therefore, ρ' subsumes the reading EC of c with the history of $\widehat{e}$ in $\widehat{H}'$. That is, $\rho' \propto \langle c, \widehat{H}'[\![\widehat{e}]\!]\rangle =: \rho''$. Moreover, it is easy to see that $\rho''^\uparrow \in X'_p$ and thus the subsumption closure property (46) of X'_p, X'_c leads us to conclude that $\rho'' \in X'_p$. To conclude, observe that $\widehat{H}'[\![\widehat{e}]\!] \sqsubseteq \widehat{H}' \sqsubseteq H$ and $\widehat{H} \sqsubseteq H$, and thus by Lemma 5, $\widehat{H}'[\![\widehat{e}]\!] = \widehat{H}$, since they are both histories of event $\widehat{e}$. Therefore $\rho = \rho'' \in X'_p \cup X'_c$. □

It also suggests to compute and maintain not only the concurrency relation $\parallel$, but also the subsumption relation $\propto$ during the construction of the enriched prefix. To do this, one solution is investigating an inductive characterization that could allow to dynamically update the relation when new ECs are included in the enriched prefix, in a similar fashion to Prop. 3. This is done in Prop. 6, where ρ plays the role of that new EC.

Proposition 6 (updating subsumption). *[BBC+10] Let $\langle e, H\rangle$, X_p, X_c, Y_g, Y_r, ρ, and ρ' be as in Prop. 3, but such that X_p, X_c and $\langle e, H\rangle$ satisfy Prop. 5. Then $\rho \propto \rho'$ iff*

$$\rho' \in Y_r \ \vee \ (\exists \rho'' \in X_p \cup X_c \colon \rho'' \propto \rho' \wedge \rho \parallel \rho')$$

We finish this section with an additional result. In Prop. 3 we gave an inductive characterization of $\parallel$. In [BBC+10], a different inductive characterization that uses the subsumption relation has been given too. Since they characterize the same relation, both are obviously equivalent. It it still natural to ask how the technical conditions of both relate.

We start recalling the characterization from [BBC+10]. Let $\langle e, H\rangle$, X_p, X_c, Y_g, Y_r, ρ, and ρ' be as in Prop. 6. Lemma 2 in [BBC+10] says that $\rho \parallel \rho'$ iff either $\rho' \in Y_g \cup Y_r$, or

- $c' \notin {}^\bullet e$, and
- for all $\widehat{\rho} \in X_p \cup X_c$ it holds that $\widehat{\rho} \parallel \rho'$, and
- if $\rho' \propto \rho''$ and $\rho''^{\uparrow} \in X_p$, then $\rho'' \in X_p$. (47)

The phrasing of Lemma 2 here is actually different than in [BBC+10], due to the fact that $\parallel$ is reflexive for us but defined as an irreflexive relation in [BBC+10]. The careful reader may quickly check that both statements are equivalent. Notation in [BBC+10] is also quite different.

The main difference between Prop. 3 and the above characterization is that the technical condition

$$\underline{{}^\bullet e} \cap H' \subseteq H \tag{48}$$

from Prop. 3 is substituted by (47). These two conditions should in principle be equivalent, and Prop. 7 confirms that this is the case.

Proposition 7 (concurrency vs. subsumption). *Let $\langle e, H\rangle \in pe(\mathcal{E})$ be a PE of $\mathcal{E}$ and X_p, X_c its sets of ECs as per Prop. 5. Let*

$$Y_g := e^\bullet \times \{H\} \qquad \rho' := \langle c', H'\rangle \in \mathcal{E}$$
$$Y_r := \underline{e} \times \{H\}$$

be the sets of generating and reading ECs produced by $\langle e, H\rangle$ and any EC ρ' from $\mathcal{E}$ satisfying

$$\rho' \notin Y_g \cup Y_r \ \text{ and } \ c' \notin {}^\bullet e \ \text{ and } \ \forall \widehat{\rho} \in X_p \cup X_c \colon (\widehat{\rho} \parallel \rho').$$

Then, the statements (47) and (48) are equivalent.

Proof. From (48) to (47), let e'' be in $\underline{{}^\bullet e} \cap H'$, i.e., $e'' \in H'$ and there is $c_1 \in {}^\bullet e \cap \underline{e''}$. Let $\rho'' := \langle c_1, H'[\![e'']\!]\rangle$ and note that ρ' does not consume c_1, so by definition $\rho' \propto \rho''$. Let $\rho_1 = \langle c_1, H_1\rangle \in X_p$ be the generating condition associated with c_1 in X_p. Denote $\rho''^{\uparrow} =: \langle c_1, H_1''\rangle$. Recall that $\rho_1 \parallel \rho'$ implies the existence of a configuration $\mathcal{C}$ such that $H_1 \sqsubseteq \mathcal{C}$ and $H_1'' \sqsubseteq H'[\![e'']\!] \sqsubseteq H' \sqsubseteq \mathcal{C}$. Therefore $\neg(H_1'' \# H_1)$, and thus, by Lemma 5, $H_1'' = H_1$. Therefore, $\rho_1 = \rho''^{\uparrow}$ and thus, by (47) we have $\rho'' \in X_p$ and hence $e'' \in H$.

From (47) to (48), suppose that $\rho' \propto \rho''$ where $\rho'' := \langle c_1, H_1''\rangle$, and $H_1'' := H'[\![e'']\!]$ for some $e'' \in H'$. Suppose also that $\rho_1 := \langle c_1, H_1\rangle := \rho''^{\uparrow}$ and $\rho_1 \in X_p$. Now $c_1 \in {}^\bullet e \cap \underline{e''}$, and by (7), one of the general assumptions in p. 18, we have that ${}^\bullet e \cap \underline{e} = \emptyset$, and so $e'' \neq e$. But $e'' \in \underline{{}^\bullet e} \cap H'$ and by (48) we get $e'' \in H$, so there must be some $\rho_2 = \langle c_2, H_2\rangle \in X_p \cup X_c$ with $e'' \in H_2$. By assumption, $\rho_2 \parallel \rho'$, so there exists some configuration $\mathcal{C}$ such that $H_2[\![e'']\!] \sqsubseteq H_2 \sqsubseteq \mathcal{C}$ and $H_1'' = H'[\![e'']\!] \sqsubseteq H' \sqsubseteq \mathcal{C}$. This implies $\neg(H_2[\![e'']\!] \# H_1'')$, hence by Lemma 5 they are equal, so by definition $\rho_2 \propto \rho''$. Since X_p, X_c are subsumption-closed, we have $\rho'' \in X_p$, as desired. □

3.9.2 *Eager Approach: Asymmetric Concurrency*

We now show how possible extensions can be made unique in the eager approach. In the lazy case, the concept of subsumption was used to achieve this. Recall the intuition behind it: if one element of $X_p \cup X_c$ contains an event e' that reads from some $c \in {}^\bullet e$, then that event must be contained in a reading condition for c included in X_p. For the eager case, this idea must be adapted to compound conditions, where the history of c may be a union of several of its readers. In this case, we demand that at least those readers of c contained elsewhere in $X_p \cup X_c$ are included in the (compound) history chosen for c in X_p.

We introduce a new relation between ECs that captures this intuition. It is a refinement of $\parallel$ that we call *asymmetric concurrency* ($/\!/$). It turns out that unique possible extensions can be characterised using only this relation.

Definition 13 (asymmetric concurrency)**.** *For any two ECs $\rho := \langle c, H\rangle$ and $\rho' := \langle c', H'\rangle$, we say that ρ is* asymmetrically concurrent *to ρ', written $\rho \mathbin{/\!/} \rho'$, iff*

$$\rho \parallel \rho' \text{ and } \underline{c} \cap H' \subseteq H. \tag{49}$$

Notice that $/\!/$ is an asymmetric relation. For instance, in Fig. 16 we have $\underline{c} \cap H_1 \subseteq H_2$ but $\underline{c} \cap H_2 = \{e_1, e_2\} \not\subseteq H_1$. Consequently $\langle c, H_2\rangle \mathbin{/\!/} \langle c, H_1\rangle$ holds but $\langle c, H_1\rangle \mathbin{/\!/} \langle c, H_2\rangle$ is false.

Observe how asymmetric concurrency can be used to satisfy the intuition stated at the beginning of this section. By asking that $\rho \mathbin{/\!/} \rho'$ for every $\rho \in X_p$ and every $\rho' \in X_p \cup X_c$, one is indeed asking that all events that read the preset of e and are included in some history ρ' in $X_p \cup X_c$ are also included in ρ as well.

We will use this fact in Prop. 9 to produce a unique characterization of PEs, but first we need to establish a relation between $\parallel$ and $/\!/$ for generating conditions. The following technical result will be necessary later.

Proposition 8 (concurrency vs asymmetric concurrency)**.** *For ECs ρ, ρ', if ρ is generating then $\rho \parallel \rho'$ iff $\rho \mathbin{/\!/} \rho'$ or $\rho' \mathbin{/\!/} \rho$.*

Proof. Let $\rho := \langle c, H\rangle$ and let $\rho' := \langle c', H'\rangle$. We only need to prove the direction from left to right, the other trivially follows from Def. 13. So suppose by contradiction that $\rho \parallel \rho'$ and neither $\rho \mathbin{/\!/} \rho'$ nor $\rho' \mathbin{/\!/} \rho$. Thus, there exist $e_1 \in (\underline{c} \cap H') \setminus H$ and $e_2 \in (\underline{c'} \cap H) \setminus H'$. So H is not empty, and it is in fact the history of some event $e \in {}^\bullet c$. Therefore $e <_i e_1$, so e must be in H'. Moreover, $e_2 \in H \setminus H'$, so $e_2 \neq e$ and $e_2 \nearrow^+ e$. This means $H \# H'$, contradicting $\rho \parallel \rho'$. □

We can now state a unique characterization of PEs suitable for the eager approach. This results shows that it is possible to entirely characterize PEs using only $/\!/$.

Proposition 9 (unique eager extensions)**.** *The pair $\langle e, H\rangle$ is an enriched event iff there exist sets X_p, X_c of ECs satisfying*

- $h(X_p) = {}^\bullet t$ *and* $h(X_c) = \underline{t}$, *with* $h(e) = t$; (50)
- X_p *contains arbitrary ECs;* (51)
- X_c *contains only generating ECs;* (52)
- $X_p \cup X_c$ *contains exactly one EC for each* $c \in ({}^\bullet e \cup \underline{e})$; (53)
- $\rho \mathbin{/\!/} \rho'$ *holds for all* $\rho \in X_p$ *and all* $\rho' \in X_p \cup X_c$; (54)
- *either* $\rho \mathbin{/\!/} \rho'$ *or* $\rho' \mathbin{/\!/} \rho$ *holds for all* $\rho, \rho' \in X_c$; (55)

- *finally,* $H = \{e\} \cup \bigcup_{\langle c,H'\rangle \in X_p \cup X_c} H'$. (56)

Moreover, for any enriched event $\langle e, H\rangle$ *there exists exactly one pair of sets* X_p, X_c *satisfying* (50) *to* (56).

Proof. The "iff" follows almost directly from Prop. 2 and Prop. 8. In particular, properties (54) and (55) imply that $\rho \parallel \rho'$ holds for all $\rho, \rho' \in X_p \cup X_c$, i.e., they imply (40). Moreover, let X_p, X_c be some collection satisfying the conditions of Prop. 2 but not Prop. 9. Then there are $\rho_1 = \langle c_1, H_1\rangle \in X_p$ and $\rho' = \langle c_2, H_2\rangle \in X_p \cup X_c$ with $\rho_1 \parallel \rho_2$ and some $e' \in (\underline{c_1} \cap H_2) \setminus H_1$. We have $H_2[\![e']\!] \sqsubseteq H$ and therefore $\neg(H_1 \# H_2[\![e']\!])$. Thus ρ_1 can be replaced with the compound condition $\rho_1' = \langle c_1, H_1 \cup H_2[\![e']\!]\rangle$ in X_p. This process can be repeated until (54) is satisfied.

It remains to show uniqueness. Suppose there exists another collection X_p', X_c' for the same enriched event $\langle e, H\rangle$. There must be some $\rho_1 = \langle c, H_1\rangle \in X_p \cup X_c$ and $\rho_2 = \langle c, H_2\rangle \in X_p' \cup X_c'$ with $H_1 \neq H_2$ and w.l.o.g. some $e_1 \in H_1 \setminus H_2$. Since $e_1 \in H$, there must be some $\rho_3 = \langle c', H_3\rangle \in X_p' \cup X_c'$ such that $e_1 \in H_3$.

If $c \in {}^\bullet e$ and $e_1 \in \underline{c}$, then $\rho_2 \in X_p$. But $\rho_2 \mathbin{//} \rho_3$ would be violated.

So let $c \in \underline{e} \cup {}^\bullet e$. If $e_1 \in {}^\bullet c$, then H_2 would not be a history of c. The final possibility is that $e_1 \nearrow^+ e'$ for some $e' \in {}^\bullet c \cup \underline{c})$ and $e' \in H_2$. But then $H_2 \# H_3$. □

Let us now briefly discuss the practical computation of //. First, observe that $\rho \mathbin{//} \rho'$ implies $\rho \parallel \rho'$, by definition. Therefore $\parallel$ is an overapproximation of //, which enables the use of the following idea. We use Prop. 3 and Prop. 4 to compute and store $\parallel$. Then we check whether $\underline{c} \cap H' \subseteq H$ and $\underline{c'} \cap H \subseteq H'$ hold, on a single exploration of the events in H. The data structure used to store $\parallel$ is slightly augmented with two bits for each pair in $\parallel$. Such bits store whether $\rho \mathbin{//} \rho'$ and $\rho' \mathbin{//} \rho$ holds. More details are discussed in § 7.1.3, on p. 113.

Because this method is available and seems to perform well in practice, we did not investigate an inductive characterization of // in the style of Prop. 3.

3.10 DISCUSSION: LAZY VS EAGER APPROACH

In order to discover possible extensions of the form $\langle e, H\rangle$, both approaches consider combinations of generating and reading histories for conditions $c \in {}^\bullet e$.

Consider Prop. 1. For every possible extension, the lazy approach takes one generating and possibly several reading histories for c, all of which must be concurrent. If the events in $\underline{c}$ have many different histories, or $\underline{c}$ is large, then many different combinations need to be checked for concurrency.

The eager characterization in Prop. 2 takes exactly one EC of arbitrary type, including compound, for c. A compound history is a set of concurrent reading histories, cf. Def. 9; thus a compound condition represents precomputed information needed to identify possible extensions. In this sense, the eager and the lazy approach can be thought of as different time/space tradeoffs.

We consider two examples in which eager beats lazy and vice versa. In Fig. 17 (a), condition c has a sequence of n readers and hence n reading or compound histories $\{e_1, \ldots, e_i\}$, for $i = 1, \ldots, n$ and one empty generating history. For each history H of c', eager simply combines H with the $n+1$

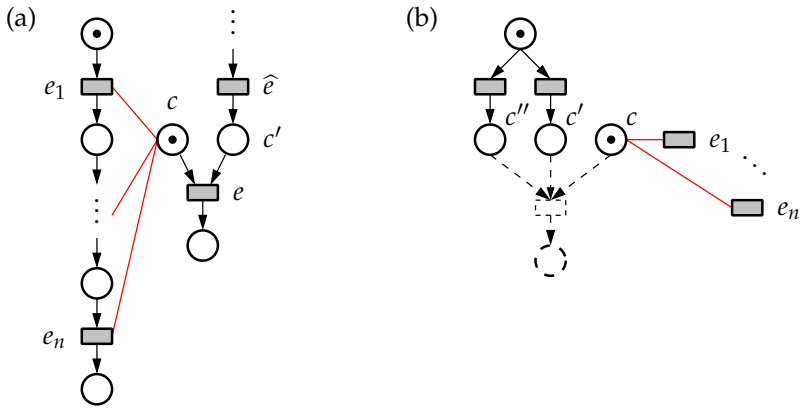


Figure 17: Good examples for the eager (a) and the lazy (b) approach.

histories for c, while lazy checks all 2^n subsets of $e_1, \ldots, e_n$ only to find these $n+1$ compound histories. This effect is actually increased if c' has many histories, eager thus becomes largely superior. Of course, an intelligent strategy may help lazy to avoid exploring all 2^n subsets one by one. However, even with a good strategy, lazy still has to enumerate at least the same combinations as eager; and since the problem of identifying the useful subsets is NP-complete [Hel99a], there will always be instances where lazy becomes inefficient, whatever strategy is employed.

On the other hand, consider Fig. 17 (b). Again, c has n concurrent readers, which this time yields 2^n histories for c. Suppose that $h(c)$ is an input place of some transition t. Now, if t also has $h(c')$ and $h(c'')$ in its preset, then no t-labelled event e that consumes c' and c'' will ever be generated in the unfolding, and all histories of c are effectively useless if $h(c)$ has no other transition than t in its postset. Since those compound conditions also appear in the computation of the concurrency relation, they become a liability in terms of both memory and execution time. The lazy approach does not suffer from this problem here.

Both approaches therefore have their merits, and we implemented them both. We shall report on experiments in Ch. 7.

3.11 MEMORY USAGE

We shall briefly discuss the memory usage arising from the methods proposed in this section. There are two major factors determining memory consumption:

- As some of our examples show, notably Fig. 9 (a) and Fig. 17 (b), a condition in the unfolding may have a number of histories exponential in the number of events reading from it. Thus, the memory usage for creating a finite unfolding prefix may be exponentially larger than the unfolding prefix that is eventually produced.
- Moreover, the memory needed to store the binary relations between ECs such as $\|$, $\propto$, and $/\!/$ is quadratic on the number of these enriched conditions, in the worst case.

One could ask whether these memory blowups are really necessary. Let us first discuss this question with regard to the second point. In a concrete implementation, the binary relations $\|$, $\propto$, and $/\!/$ could either be stored explicitly, at the cost of quadratic memory overhead, or decided individ-

ually for each pair whenever necessary. These two approaches have been discussed in § 3.5, and referred under the name of, respectively, *concurrency relations* and *backwards exploration*. The second approach requires, for each query to the relation, to explore the prefix using the respective definitions, thus incurring higher computation time. We initially implemented the second approach [Rod10]. The running times were such that only small unfoldings with a few hundred events could be produced in reasonable time. We therefore chose to store the binary relations explicitly.

In [RS13a], we have introduced an improved algorithm to perform the backwards exploration. Essentially, it tries to reduce both the fragment of the prefix to be explored, and the number of times it is explored. Although the approach only works for ordinary unfoldings, and the prototype implementation was certainly limited, its performance was quite promising when compared to that of stored concurrency relations.

Histories, on the other hand, have the twofold purpose of allowing to identify both cutoff events and new PEs, as in § 3.7. In principle, one could imagine an additional time/space trade-off in which not only the concurrency relation but even the histories themselves are constructed on demand whenever one needs to cutoff the prefix or compute PEs. The would lead to an algorithm which consumes only linear space on the size of the unfolding prefix. Due to the unsatisfactory results with the backwards exploration approach, we did not consider this idea.

In any case, the memory usage is asymptotically the same as for the PR-encoding. Section 7.1.1, on p. 112, contains some hints on how to store histories efficiently, and § 7.4, on p. 119, provides data on actual memory usage on several examples.

3.12 CONCLUSION

We have made theoretical contributions to the computation of contextual unfoldings, opening the way to their practical use for verification. Our implementation of the methods introduced in this chapter will be presented in Ch. 7, together with experimental results and evaluations. Those will show that these methods are practical and outperform existing techniques on a wide number of cases.

Our main contribution is the eager approach to compute PEs. Technically, we have proposed to associate histories to conditions of the prefix. We presented a concurrency relation on conditions enriched with such histories, showed how it can be used to characterize PEs (eager method), and gave an inductive characterization of this relation, thus enabling the use of efficient algorithms to maintain the relation as the construction of the prefix progresses. Next, we refined the characterization of PEs using asymmetric concurrency. This novel notion simplifies the theoretical characterization w.r.t. to existing results and has the practical benefit of not requiring an ad-hoc algorithm for computing it.

Additionally, we compared our eager method to the lazy method, another approach to computing PEs developed independently [BBC+10]. We compared both and discussed their merits on theoretical cases.

Unfolding-based methods need two ingredients: an efficient technique for constructing them, and efficient methods for analyzing the prefixes. We have provided the first ingredient in this quest. In the next chapter, we show that their succinctness helps in speeding up these analyses.

Moreover, it would also be interesting to investigate a mix between eager and lazy that tries to get the best of the two worlds. For instance, one could start with the eager approach and switch (selectively for some conditions) to lazy as soon the number of compound conditions exceeds a certain bound. This, and other ideas, remain to be tested.

4 REACHABILITY AND DEADLOCK CHECKING

Model checking based on unfoldings conceptually happens in two steps. First, one constructs an unfolding of the system; then one reduces the verification question to some analysis on the unfolding. A method to construct contextual unfoldings was presented in the previous chapter. Here we present a reduction of the coverability and deadlock-freeness problems into SAT. We study a number of optimizations of the encoding. In Ch. 7, we report on the impact of such optimizations and compare the performance of our encoding to other deadlock-checking tools.

This chapter presents results of [RS12b].

4.1 INTRODUCTION

McMillan introduced unfoldings as a method for verifying properties of *ordinary Petri nets*. He showed that for a bounded net N, one can use a finite unfolding prefix $\mathcal{P}$ to check certain properties of N such as reachability of markings or deadlock-freeness. The publication of his work [McM93b] triggered a large body of research on unfoldings, in particular on their use for verification [McM95a; MR97; Hel99c; KK00; EH01; ES01].

Recall that given a finite complete prefix $\mathcal{P}$ of a bounded net N, deciding deadlock-freeness, reachability, or coverability on N is NP-complete, see [McM93b]. Consequently, previous works on deadlock-checking of ordinary nets have proposed reductions to different NP-complete problems: McMillan [McM93b; McM95b] employed a branch-and-bound method, Heljanko [Hel99d; Hel99c] a stable-models encoding, and Melzer and Römer [MR97] used mixed integer linear programming, later improved by Khomenko and Koutny [KK00; KK07]. The technique used by Esparza and Schröter [ES01] is an ad-hoc algorithm based on additional information obtained while computing the unfolding.

The previous decade has seen the emergence of powerful SAT solving algorithms. Programs like MINISAT [ES03] employ advanced techniques such as clause learning, 2-watch propagation schemes, etc. It is natural to profit from these advances and reduce to SAT instead; all the more so because unfoldings are 1-safe nets, so the marking of a place naturally translates to a Boolean variable. Indeed, SAT solving has already been proposed for the similar problem of model-checking merged processes [KKKV06], and [EH08] gives an explicit SAT encoding for ordinary net unfoldings. However, we are not aware of a publicly available tool that uses this idea. In this chapter, we propose reachability and deadlock-freeness checking methods based on contextual unfoldings.

Our contributions are twofold: we investigate SAT-encodings of unfoldings, and we extend them to c-nets, proposing some optimizations. Building on advantages of contextual unfolding, i.e., faster construction and smaller size, we intend to produce faster verification methods. It is worth noting that the smaller size of c-net unfoldings does not automatically translate to an easier SAT problem, for the following reasons:

1. The presence of read arcs may cause so-called *cycles of asymmetric conflict*. Thus, a SAT encoding requires acyclicity constraints, which are not necessary for conventional unfoldings.
2. An event in a c-net unfolding can occur in multiple different execution contexts, called *histories*, and the constructions proposed in Ch. 3 require to annotate events with potentially many such histories. In contrast, every event of an ordinary net unfolding has only one history. Some verification algorithms for ordinary nets rely on this fact and do not easily adapt to c-nets.

In this chapter, we propose solutions for both problems. Our encoding does not refer to the histories at all, and the effect of the acyclicity constraints can be palliated by several strategies.

Our algorithms have been implemented in the tool CNA, integrated in the Cunf Toolset, and will be experimentally evaluated in Ch. 7. The solving times of the SAT instances generated by the tool are better than previous approaches when the tool is applied to ordinary net unfoldings, and even better when used on c-net unfoldings.

The chapter is structured as follows. In § 4.2 we discuss how unfoldings can be used to check for deadlock and reachability. In, § 4.3, the core of the reduction into SAT is presented. The detection of cycles of symmetric conflict plays an important role in the problem we solve, § 4.4 discusses the solution we propose. In §§ 4.5 to 4.7, we discuss the size of the encoding and present optimizations to it. Conclusions are drawn in § 4.8.

4.2 USING C-NET UNFOLDINGS FOR VERIFICATION

In this section, we illustrate why some existing verification approaches for Petri net unfoldings do not adapt well to c-net unfoldings. In particular, we show that existing deadlock-checking techniques based on ordinary unfoldings [McM95b; MR97; Hel99c; KK07] do not lift straight-forwardly to c-net unfoldings. This will justify the completeness criterion considered in Ch. 3 for constructing unfolding prefixes.

We fix now notation for the rest of the chapter. Unless otherwise stated we let $N := \langle P, T, F, C, m_0 \rangle$ be a finite, *bounded* c-net satisfying the general assumptions in p. 18. We let $\mathcal{U}_N := \langle \langle \tilde{B}, \tilde{E}, \tilde{G}, \tilde{D}, \tilde{m}_0 \rangle, h \rangle$ denote the full unfolding of N and denote by $\mathcal{E} := \langle O, h, \chi \rangle$ an arbitrary enriched prefix for N.

Recall that a marking m of a net N is deadlocked if it does not enable any transition of N. The deadlock-checking problem is to decide whether N has at least one such marking.

In Ch. 3, we presented an algorithm for the construction of marking-complete prefixes. Most of the existing deadlock-checking techniques based on ordinary unfoldings rely on prefixes that are complete w.r.t. a notion stronger than marking-completeness. We first lift this notion from ordinary unfoldings to contextual ones.

Consider the unfolding prefixes that Algorithm 1 would construct if it is modified to include all possible cutoff extensions in the prefix. Such extensions could be kept on a separate list and appended to the prefix after the main loop has terminated. Modified in this way, Algorithm 1 would produce an enriched prefix $\mathcal{E}$ that, besides being marking-complete, would satisfy the following: if a reachable marking m of N enables a transition t, then for any *cutoff-free* configuration $\mathcal{C}$ of $\mathcal{E}$, if $\mathcal{C}$ reaches m, then there is an event e *in the prefix* such that $h(e) = t$ and e is enabled by $cut(\mathcal{C})$. Let us formalize this notion:

Definition 14. *Let E_{cut} be a set of enriched events contained in $\mathcal{E}$. We call $\mathcal{E}$* enable-complete *w.r.t. E_{cut} if for every reachable marking m of N, there is a finite configuration $C \in conf(\mathcal{E})$ such that,*

1. *C is free of enriched events in E_{cut}, and*
2. *$m = mark(C)$, and*
3. *for every event e of $\mathcal{U}_N$, if $C' := C \cup \{e\}$ is a configuration, then $\mathcal{E}$ contains the enriched event $\langle e, C'[\![e]\!]\rangle$.*

Conditions 1 and 2 ask that $\mathcal{E}$ stripped of E_{cut} is marking-complete; 1 and 3 enforce the property we mentioned before.

A confusing point is that the prefix is enable-complete *w.r.t.* a set of enriched events and not *per se*. Unfolding prefixes are obtained by pruning the full unfolding $\mathcal{U}_N$ at arbitrary points. Such pruning introduces cuts $\tilde{m}$ that artificially enable fewer events than $h(\tilde{m})$, the associated marking in N. So any configuration with cutoffs may enable fewer events than its marking in N, and we need to restrict ourselves to configurations without cutoffs. The definition is therefore relative. When restricted to ordinary nets, enable-completeness coincides with the completeness criterion satisfied by canonical prefixes [KKV03].

Remark 10. *If $\mathcal{E}$ is enable-complete w.r.t. E_{cut}, then N has a reachable deadlocked marking iff $\mathcal{E}$ has a configuration C free of enriched events in E_{cut} such that $cut(C)$ deadlocks.*

Remark 10 is either directly [MR97; Hel99c; KK07] or indirectly [McM95b] employed in existing deadlock-checking techniques for *ordinary* Petri nets. Now, the problem with generalizing to c-nets any of these techniques is that Rmk. 10 requires to reason about cutoffs, and therefore about histories. Crucially, an unfolding prefix with $\mathcal{O}(n)$ events could have $\mathcal{O}(2^n)$ histories. Several approaches to overcome this are in principle possible, we discuss them now, keeping in mind that our goal is to encode the deadlock-checking problem into SAT. In the following, let $\mathcal{E}$ be an enable-complete enriched prefix.

REASONING ABOUT HISTORIES. We could produce a SAT encodings that explicitly handles the notion of history. Satisfying assignments of such formulas would encode configurations free of enriched events from the set

$$E_1 := \{\langle e, H\rangle \in \mathcal{E} : \langle e, H\rangle \text{ is cutoff}\}.$$

While the approach would be sound, even with a compact representation of histories such an encoding would probably be of exponential size on the number of events in $\mathcal{E}$. Clearly, this approach does not profit from the additional compactness of c-net unfoldings.

APPROXIMATING CUTOFFS BY EVENTS. A second approach could be based on the idea of transferring the notion of cutoff from enriched events to events, so that the SAT encoding reasons only about events. Let us show why this is a bad idea.

Events may have multiple histories, some of which are cutoffs while others are not. Each event e can be found in exactly one of the following classes:

1. All feasible histories in $Hist(e)$ are cutoffs.
2. Among the feasible histories in $Hist(e)$, there is at least one cutoff and one non-cutoff.
3. No feasible history in $Hist(e)$ is a cutoff.

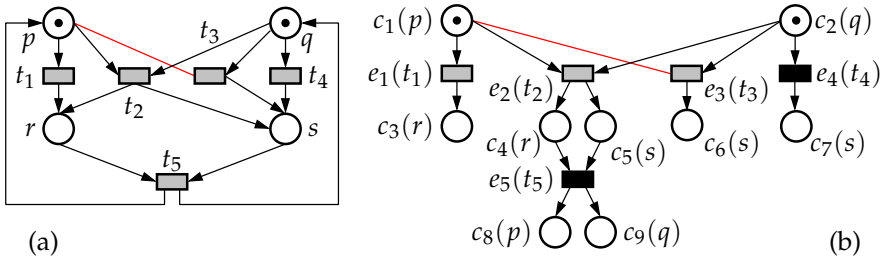


Figure 18: (a) A safe c-net N; and (b) an unfolding prefix $\mathcal{P}$ for N.

Observe that we restrict the attention to only the *feasible* histories in $Hist(e)$. Clearly the SAT encoding would forbid any event in the first class from being used in the satisfying configuration, and would allow all events in the third. As for those in the second, we now show that either including or forbidding them yields an unsound characterization of the deadlocked markings.

Assume that we allow events of the second class to be present in the satisfying configuration. This amounts to using the set

$$E_2 := \{\langle e, H\rangle \in \mathcal{E} : \ e \text{ is in the first class}\}$$

of enriched events as cutoffs in Def. 14, i.e. *not* forbidding events in the second class. Although encoding E_2 in SAT is easy, as it does not require information about histories, $\mathcal{E}$ is not enable-complete w.r.t. E_2, as we show below, and the resulting formula would incorrectly characterize the deadlocked markings of N.

Consider the c-net show in Fig. 18 (a), we will use it to show that $\mathcal{E}$ is not enable-complete w.r.t. E_2. The c-net is free of deadlocks. An unfolding prefix $\mathcal{P}$ is shown in Fig. 18 (b), the mapping h is given in parentheses. Event e_1 has two histories, $H_1 := \{e_1\}$ and $H_2 := \{e_3, e_1\}$. Now fix the size strategy $\prec_M$, defined in (26) as $\mathcal{C} \prec_M \mathcal{C}'$ iff $|\mathcal{C}| < |\mathcal{C}'|$. According to the framework of Ch. 3, both $\langle e_1, H_1\rangle$ and $\langle e_1, H_2\rangle$ are $\prec_M$-feasible, but $\langle e_1, H_2\rangle$ is $\prec_M$-cutoff while $\langle e_1, H_1\rangle$ is not. Indeed H_2 leads to the same marking $\{r, s\}$ as $\langle e_2, \{e_2\}\rangle$ and $\{e_2\} \prec_M H_2$. An event is shown in black if all its feasible histories are cutoffs, i.e., if it belongs to the first class above. The prefix in Fig. 18 (b) is marking-complete and also enable-complete w.r.t. E_1 but not w.r.t. E_2: consider the cut $\tilde{m} = \{c_3, c_6\}$, which is deadlocked in $\mathcal{P}$. The configuration leading to m' contains a cutoff, namely $\langle e_1, H_2\rangle$, but such enriched event is not included in E_2, because e_1 is in the second class. As a result, its marking is interpreted by Def. 14 as a faithful representative of a marking of N, which is wrong, as $h(\tilde{m}) = \{r, s\}$ enables transition t_5 in N.

Alternatively, one could forbid all events in the second class. This corresponds to using the set

$$E_3 := \{\langle e, H\rangle \in \mathcal{E} : \ e \text{ is in the first or second class}\}$$

of enriched events as cutoff points in Def. 14. Now the problem is that $\mathcal{E}$ stripped of E_3 is not necessarily marking-complete, as Def. 14 asks. There is thus no hope to achieving enable-completeness w.r.t. E_3, which intuitively forbids using *too many* enriched events.

ADDITIONAL LAYER OF CUTOFFS. A third option could be extending $\mathcal{E}$ with a further *layer* of infeasible enriched events. The resulting enriched prefix would be computed in two steps: (i) *saturating* the prefix with all histories arising from events already present on it; (ii) assuming that $\mathcal{E}'_N$ is

the prefix computed on the previous step, one would extend $\mathcal{E}'_N$ with the set

$$E_4 := pe(\mathcal{E}'_N)$$

The enriched prefix $\mathcal{E}''_N$ resulting after the two steps would indeed be enable-complete w.r.t. E_4. Observe that by definition all events of $\mathcal{E}''_N$ having histories in E_4 have been added in step (ii), and all those histories are infeasible: we now have a safe correspondence between cutoffs and events. As a result, the SAT encoding would just need to forbid all such events. Although this approach seems feasible, it has the disadvantage of requiring a potentially larger (enriched) prefix.

USING JUST MARKING-COMPLETENESS. Another, perhaps simpler approach is simply relying on a marking-complete unfolding prefix. One can just check if a configuration $\mathcal{C}$ represents a deadlocked marking by looking at the transitions enabled by $mark(\mathcal{C})$. Apart from being conceptually simpler, the main advantages are that we do not need histories at all. Moreover, according to Rmk. 8, the approach does not need a prefix with more events than reachable markings exist in N. In the next section, we provide a SAT encoding whose correctness relies on this idea.

Remark 11. *Let $\mathcal{P}$ be a marking-complete prefix for N. Then N contains a deadlock iff $\mathcal{P}$ has either a reachable marking $\tilde{m}$ such that $h(\tilde{m})$ is deadlocked.*

4.3 SAT-ENCODINGS OF C-NET UNFOLDINGS

In this section, we define two propositional formulas. The first one is satisfiable iff N has a reachable deadlocked marking, and the second iff it has a reachable marking that covers a given set of places. Our encodings relies on Rmk. 11 and requires a marking-complete unfolding prefix $\mathcal{P}$. When this encoding is restricted to ordinary nets, it essentially coincides with the one given in [EH08, § 5.1.2]. We also give a number of optimizations of the encoding. Most constraints that we give translate directly into CNF.

For the rest of the chapter, we fix a marking-complete unfolding prefix $\mathcal{P} := \langle\langle B, E, G, D, \tilde{m}_0\rangle, h\rangle$. Recall that additional notation has been fixed at the beginning of previous section. The SAT problem is as follows: given a formula ϕ of propositional logic, find whether there exists a satisfying assignment that makes ϕ true. Our goal is to construct from $\mathcal{P}$ and a given set $M \subseteq P$ of places in N, propositional formulas

- $\phi^{\text{dead}}_{\mathcal{P}}$, satisfiable iff N has a deadlocked reachable marking;
- $\phi^{\text{mark},M}_{\mathcal{P}}$, satisfiable iff N has a reachable marking m such that $m(p) \geq 1$ for all $p \in M$.

Both formulas reason about reachable markings of N. Their satisfying assignments encode configurations of $\mathcal{P}$ that reach the requested markings. Both formulas are defined over variables e for every event $e \in E$ and p for for every place $p \in P$, and have the form:

$$\phi^{\text{dead}}_{\mathcal{P}} := \phi^{\text{conf}}_{\mathcal{P}} \wedge \phi^{\text{dis}}_{\mathcal{P}}$$
$$\phi^{\text{mark},M}_{\mathcal{P}} := \phi^{\text{conf}}_{\mathcal{P}} \wedge \phi^{\text{cov},M}_{\mathcal{P}}$$

The first constraint $\phi^{\text{conf}}_{\mathcal{P}}$ is present on both formulas, and enforces that any satisfying assignment represents a finite configuration $\mathcal{C}$. Constraints $\phi^{\text{dis}}_{\mathcal{P}}$ and $\phi^{\text{cov},M}_{\mathcal{P}}$ compute the marking $mark(\mathcal{C})$ reached by the configuration and

require, respectively, that $mark(\mathcal{C})$ is deadlocked, or that it covers all places in M.

Recall that a finite configuration $\mathcal{C}$ is a set of events, causally closed and free of loops in the $\nearrow_{\mathcal{C}}$ relation, see (17) to (19) in p. 24. For efficiency reasons, we treat separately cycles in the symmetric and asymmetric conflict relation, and define:

$$\phi_{\mathcal{P}}^{\text{conf}} := \phi_{\mathcal{P}}^{\text{causal}} \wedge \phi_{\mathcal{P}}^{\text{sym}} \wedge \phi_{\mathcal{P}}^{\text{asym}}$$

Subformulae $\phi_{\mathcal{P}}^{\text{causal}}$ and $\phi_{\mathcal{P}}^{\text{sym}}$ request $\mathcal{C}$ to be a causally closed set of events that has no pair of events in symmetric conflict:

$$\phi_{\mathcal{P}}^{\text{causal}} := \bigwedge_{\substack{e \in E \\ e' \in {}^\bullet({}^\bullet e \cup e)}} (\mathsf{e} \to \mathsf{e}'), \qquad \phi_{\mathcal{P}}^{\text{sym}} := \bigwedge_{c \in C} \mathsf{AMO}(c^\bullet),$$

where $\mathsf{AMO}(\mathsf{x}_1, \ldots, \mathsf{x}_n)$ is satisfied iff *at most one* of $\mathsf{x}_1, \ldots, \mathsf{x}_n$ is satisfied[1] (see § 7.2.2). The subformula $\phi_{\mathcal{P}}^{\text{sym}}$ forbids symmetric conflicts, i.e., it ensures that any two events $e, e' \in \mathcal{C}$ satisfy $\neg(e \# e')$. Recall that due to (6), symmetric conflicts correspond to loops of length 2 in $\nearrow$. Constraint $\phi_{\mathcal{P}}^{\text{asym}}$ forbids all loops of asymmetric conflict not forbidden by $\phi_{\mathcal{P}}^{\text{sym}}$, as we detail in § 4.4.

The formula $\phi_{\mathcal{P}}^{\text{dis}}$ characterizes the marking reached by $\mathcal{C}$ and asks for it to be deadlocked. We define it as:

$$\phi_{\mathcal{P}}^{\text{dis}} := \Big(\bigwedge_{\substack{c \in B \\ p = f(c) \\ \{e\} = {}^\bullet c}} \Big((\mathsf{e} \wedge \bigwedge_{e' \in c^\bullet} \neg \mathsf{e}') \to \mathsf{p} \Big) \Big) \;\wedge\; \Big(\bigwedge_{t \in T} \bigvee_{p \in {}^\bullet t \cup \underline{t}} \neg \mathsf{p} \Big)$$

This constraint is a conjunction of two main subformulas. Recall that $\phi_{\mathcal{P}}^{\text{conf}}$ enforces any satisfying assignment to encode a configuration $\mathcal{C}$. The left-hand side of the main conjunction computes a (super)set of the *places* marked by $\mathcal{C}$. Intuitively, it says that any place p should be marked, i.e., variable p should be true, if it labels at least one condition c such that ${}^\bullet c$ is in $\mathcal{C}$ but no event in $c^\bullet$ is in $\mathcal{C}$. The right-hand side subformula just asks that no transition of the original net is enabled at the (superset of the) marking computed by the left-hand side subformula. Notice that a variable p may be true even if $p \notin mark(\mathcal{C})$. However, such an assignment can only serve to *hide* a deadlock, so this encoding is safe.

For coverability, we want to check whether N has a reachable marking m such that $M \subseteq m$, where $M \subseteq P$ is given. We define constraint $\phi_{\mathcal{P}}^{\text{cov},M}$ as the conjunction of two subformulas:

$$\phi_{\mathcal{P}}^{\text{cov},M} := \Big(\bigwedge_{p \in M} \big(\bigvee_{f(c) = p} \mathsf{c} \big) \Big) \;\wedge\; \Big(\bigwedge_{f(c) \in M} (\mathsf{c} \to (\bigwedge_{e \in {}^\bullet c} \mathsf{e} \wedge \bigwedge_{e \in c^\bullet} \neg \mathsf{e})) \Big)$$

The first subformula ensures that for every place p in M, at least one condition labelled with p is marked. For any such condition, the second subformula constraints which events fire. If a condition c is chosen, then necessarily ${}^\bullet c$ must be in $\mathcal{C}$ and none of the events in $c^\bullet$ can be in $\mathcal{C}$. Observe that the direction of the implication here and in $\phi_{\mathcal{P}}^{\text{dis}}$ is the opposite. This yields correct results in both cases.

4.4 ASYMMETRIC CONFLICT LOOPS

We now explain $\phi_{\mathcal{P}}^{\text{asym}}$, which ensures that $\nearrow_{\mathcal{C}}$ is acyclic. As explained in § 2.1, we equate a relation with a directed graph in the natural way. Sym-

1 Note that quite frequently, the actual implementation of the AMO constraint will introduce new auxiliary variables in $\phi_{\mathcal{P}}^{\text{dead}}$, as it is the case of the k-trees proposed in § 7.2.2.

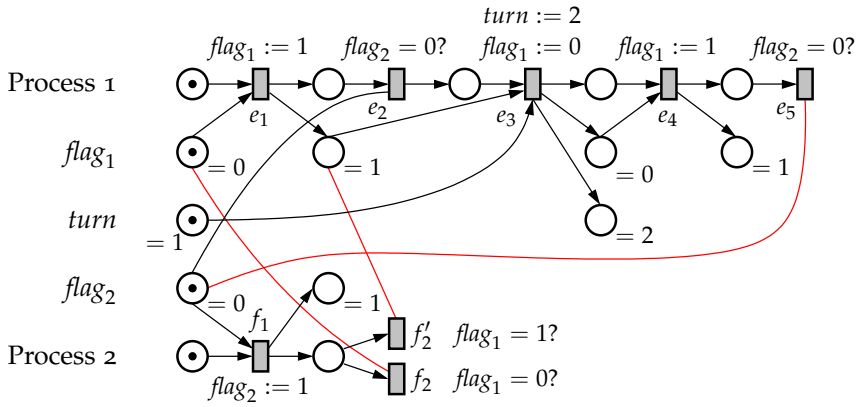


Figure 19: Partial unfolding of Dekker's algorithm with asymmetric cycles.

metric conflicts form cycles of length 2 in $\nearrow$ and are efficiently handled by the AMO constraints of $\phi_{\mathcal{P}}^{\text{sym}}$. In a Petri net, these are the only cycles that can occur. However, in a c-net there may also be other kind of cycles. Recall that the definition of $\nearrow$ considers three cases: (4), (5), and (6), the last one being symmetric conflict. For convenience, we explicitly repeat the first two cases. For events $e, e' \in E$, define

$$\begin{aligned} e \lessdot e' &\text{ iff } e^\bullet \cap ({}^\bullet e' \cap \underline{e'}) \neq \emptyset \\ e \nearrow\!\!\!\!\nearrow e' &\text{ iff } \underline{e} \cap {}^\bullet e' \neq \emptyset \end{aligned} \tag{57}$$

Indeed, observe that $e \lessdot e'$ holds (resp. $e \nearrow\!\!\!\!\nearrow e'$ holds) iff (4) holds (resp. (5) holds) between e and e'.

In a c-net unfolding, every cycle of $\nearrow$ not involving events in symmetric conflict is also a cycle in the relation

$$R := \lessdot \cup \nearrow\!\!\!\!\nearrow. \tag{58}$$

We show that these cycles occur naturally in well-known examples.

Consider Fig. 19, which shows the beginning of an unfolding of Dekker's mutual-exclusion algorithm [Ray86], where only some events of interest are shown. In the beginning, both processes indicate their interest to enter the critical section by raising their flag (events e_1, f_1). They then check whether the flag of the other process is low (events e_2, f_2) and if so, proceed (e_3) and possibly repeat (e_4, e_5). If both processes want to enter the critical section (f_2'), some arbitration happens (not displayed). Two conflict cycles in this example are

$$e_1 \lessdot e_2 \nearrow\!\!\!\!\nearrow f_1 \lessdot f_2 \nearrow\!\!\!\!\nearrow e_1$$

and

$$f_1 \lessdot f_2' \nearrow\!\!\!\!\nearrow e_3 \lessdot e_4 \lessdot e_5 \nearrow\!\!\!\!\nearrow f_1.$$

Several encodings have been proposed in the literature for acyclicity constraints, including transitive closure and ranks (see, e.g., [CGS09]). In the ranking method, one introduces for each event e additional Boolean variables that represent an integer, the so-called *rank* of e, up to r, where r is a large enough number. Then, for every pair $(e, f) \in R$, one introduces a clause of the form

$$(\mathsf{e} \wedge \mathsf{f}) \rightarrow \mathsf{s}_{e,f}$$

where $\mathsf{s}_{e,f}$ is a new variable that, if true forces the rank of e to be less than the rank of f. This is enforced by an additional set of clauses that involve

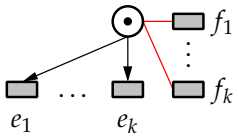


Figure 20: An occurrence net of size $\mathcal{O}(k)$ where $|\nRightarrow| = k^2$.

the ranks of e and f together with $\mathsf{s}_{e,f}$. Naturally, these clauses are only necessary if e and f are in the same SCC of R.

A lower bound for r is the length of the longest chain in $\nearrow$ that does not contain a cycle. However, finding whether any such chain of a given length exists is already a NP-complete problem. A simple upper bound for r is the size of the largest SCC of R.

If the ranks of events are represented in binary, the encoding involves $\mathcal{O}(|R| \log r)$ literals. Integers up to r can be encoded in binary by $\log r$ Boolean variables. The clauses that enforce two rankings to be sorted involve a number of literals linear on $\log r$, and every edge of R requires one such comparison. Alternatively, coding in unary the ranks yields an encoding of size $\mathcal{O}(|R|r)$. We refer the reader to § 7.2.3 for an experimental evaluation of the alternative encodings.

To further reduce the size of the encoding, observe the following. Let $n = \mathcal{O}(|\mathcal{P}|)$ be the size of the prefix, as defined in § 2.2, in p. 16. Observe that $|R| = \mathcal{O}(n^2)$, because $|\nRightarrow| = \mathcal{O}(n^2)$ even though $|\lessdot| = \mathcal{O}(n)$, as Fig. 20 illustrates. An straightforward application of the acyclicity encoding presented above to R produces a formula of size $\mathcal{O}(n^2 \log n)$, if ranks are coded in binary. One can however use a trick to reduce to this to $\mathcal{O}(n \log n)$, as we explain now.

The idea is to reduce the $\mathcal{O}(n^2)$-explosion in $\nRightarrow$ introducing additional nodes in the underlying graph. We explain it with an example, consider the occurrence net in Fig. 20. Here one would substitute the k^2 edges in (the underlying graph of) $\nRightarrow$ by one new node, which *represents* the single condition in the net, and $2k$ edges, k from the new node to all the events e_i and k reaching it from the events f_i.

Another idea for reduction is to exploit the fact that $\mathcal{C}$ is causally closed and that every cycle in R contains at least two edges stemming from $\nRightarrow$. Consider the relation

$$R' := \{(e, g) \colon \exists f, h \colon e \nRightarrow f \leq g \nRightarrow h\}. \tag{59}$$

One can easily see that *any causally closed* set of events contains a cycle in R iff it contains a cycle in R', so one can ask for acyclicity of R' instead of R.

On the other hand, R' may actually contain more pairs than R, and computing R' may take quadratic time in $|E|$. So instead, we reduce the size of R by a less drastic method that can run in linear time: An event e is eliminated from R by fusing its incoming and outgoing edges in R only if (i) e is not the source of a $\nRightarrow$-edge and (ii) fusing the edges and eliminating e will not increase the number of edges in R.

Figure 19 demonstrates another important point. The unfolding contains two different cycles, both of which contain f_1. Thus, all events in Fig. 19 belong to the same SCC in R. Indeed, we observe in our experiments that the SCCs of R tend to be large, often composed of thousands of events, but consist of many short, interlocking cycles. This suggests that an upper bound for r better than the size of R, even after reduction, may still be feasible. We therefore suggest another trick: first, check for deadlock or coverability

Subformula	Size	Size
$\phi_{\mathcal{P}}^{\text{causal}}$	$\lvert E\rvert q$	$\mathcal{O}(n)$
$\phi_{\mathcal{P}}^{\text{sym}}$	$\lvert B\rvert pk$	$\mathcal{O}(n)$
$\phi_{\mathcal{P}}^{\text{asym}}$	$(\lvert G\rvert + \lvert D\rvert)\log(\lvert E\rvert + \lvert B\rvert)$	$\mathcal{O}(n \log n)$
$\phi_{\mathcal{P}}^{\text{dis}}$	$\lvert B\rvert(1+p) + \lvert T\rvert q$	$\mathcal{O}(n)$
$\phi_{\mathcal{P}}^{\text{mark},M}$	$\lvert B\rvert + \lvert B\rvert(1+p)$	$\mathcal{O}(n)$

Table 3: Size of $\phi_{\mathcal{P}}^{\text{dead}}$ and $\phi_{\mathcal{P}}^{\text{cov},M}$. See text for interpretation.

while omitting $\phi_{\mathcal{P}}^{\text{asym}}$ from $\phi_{\mathcal{P}}^{\text{dead}}$ or $\phi_{\mathcal{P}}^{\text{cov},M}$ altogether. This may result in false positives, i.e., a set of events leading to the requested marking that is not actually reachable because it contains a cycle in $\nearrow$. If the SAT solver comes up with such a spurious solution, we repeat with $\phi_{\mathcal{P}}^{\text{asym}}$ properly included. The experiments concerning these points are discussed in § 7.2.3.

4.5 ENCODING SIZE

In this section we examine the size of the formulas $\phi_{\mathcal{P}}^{\text{dead}}$ and $\phi_{\mathcal{P}}^{\text{cov},M}$. Although not all the formulas given in § 4.3 are in CNF, they translate directly to CNF, with the exception of $\phi_{\mathcal{P}}^{\text{sym}}$ — for which we did not yet provide a definition. For instance, $\phi_{\mathcal{P}}^{\text{dis}}$ involves a conjunction of subformulas of the form $l_1 \wedge \ldots \wedge l_n \to l_0$ where all the l_i are literals. Each such subformula is obviously equivalent to a clause of the form $\neg l_1 \vee \ldots \vee \neg l_n \vee l_0$.

Table 3 presents the sizes of the different subformulas of our encodings, i.e., the number of literals in each subformula when it is translated to CNF. Sizes are given in terms of a constant k, which depends on the particular implementation of the AMO constraints in $\phi_{\mathcal{P}}^{\text{sym}}$, and the three following parameters depending on $\mathcal{P}$:

$$q := \max_{e \in E} |{}^\bullet e \cup \underline{e}| \qquad p := \max_{c \in B} |c^\bullet| \qquad n := |B| + |E| + |G| + |D|$$

It is almost immediate to see that all sub-constraints except $\phi_{\mathcal{P}}^{\text{asym}}$ are linear on the size n of the unfolding prefix $\mathcal{P}$. An AMO constraint for m variables can be implemented with a number of clauses and literals both linear on m, as we show in § 7.2.2.

The size $\mathcal{O}(n \log n)$ of $\phi_{\mathcal{P}}^{\text{asym}}$ often dominates the size of the encodings, although it can sometimes be completely omitted from it. Experiments concerning this will be presented in § 7.5.

4.6 REDUCTION OF STUBBORN EVENTS FOR DEADLOCK CHECKING

In SAT solving, the value of a variable that is either known or has been tentatively decided is *propagated* to simplify other clauses, see [SS96; ES03]. This process is referred as *propagation* or *unit propagation*. In this section, we discuss an optimization that palliates a propagation problem of SAT checkers when solving $\phi_{\mathcal{P}}^{\text{dead}}$. We also explain why such an optimization is not applicable to $\phi_{\mathcal{P}}^{\text{cov},M}$.

Consider the occurrence net shown in Fig. 21, where certain events have been explicitly named ($e_1, e_2, \ldots$) and only the label is given for conditions, in parenthesis. If e_1 fires, then nothing can prevent e_2, e_3, e_4, and e_5 from firing. Thus, any configuration leading to a deadlock must either contain all

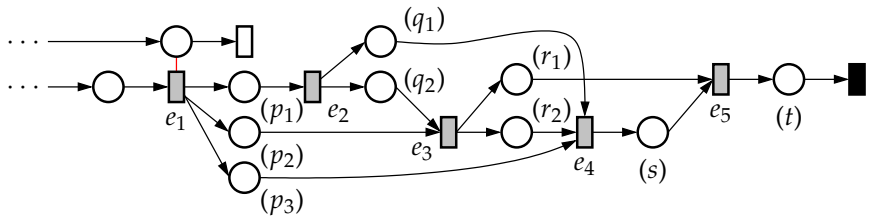


Figure 21: Stubborn events.

five events or none of them. However, e_1 is not guaranteed to fire due to the white event that consumes from its context.

The formula $\phi_{\mathcal{P}}^{\text{dead}}$ will consist of the following clauses for the named events and conditions in Fig. 21; we assume that the black event at the right is a cutoff (has no non-cutoff histories). For $\phi_{\mathcal{P}}^{\text{causal}}$, we have the clauses $\neg e_j \vee e_i$ for $1 \leq i < j \leq 4$ and $\neg e_5 \vee e_3$ and $\neg e_5 \vee e_4$. For $\phi_{\mathcal{P}}^{\text{sym}}$ and $\phi_{\mathcal{P}}^{\text{asym}}$, no clauses will be generated. The left-hand side of the main conjunction in $\phi_{\mathcal{P}}^{\text{dis}}$ will consist of the following clauses:

$$\begin{array}{lll} \neg e_1 \vee e_2 \vee p_1 & \neg e_2 \vee e_4 \vee q_1 & \neg e_3 \vee e_5 \vee r_1 \\ \neg e_1 \vee e_3 \vee p_2 & \neg e_2 \vee e_3 \vee q_2 & \neg e_3 \vee e_4 \vee r_2 \\ \neg e_1 \vee e_4 \vee p_3 & \neg e_4 \vee e_5 \vee s & \neg e_5 \vee t \end{array}$$

The clauses forming the right-hand side are:

$$\begin{array}{lll} \neg p_1 & \neg q_2 \vee \neg p_2 & \neg q_1 \vee \neg r_2 \vee \neg p_3 \\ \neg r_1 \vee \neg s & \neg t & \end{array}$$

When this formula is presented to a SAT solver like MINISAT, the unit clauses $\neg p_1$ and $\neg t$ lead to some immediate simplifications. This propagation is handled very efficiently by modern solvers, and there is no gain in emulating this behavior while generating the SAT encoding.

Suppose that the solver then tries to set e_1 to true and propagates this decision. The remaining simplified clauses are shown below. Note that e_2 and $\neg e_5$ are implied as consequence of e_1 and the other simplifications.

$$\begin{array}{llll} \neg e_4 \vee e_3 & e_3 \vee p_2 & e_4 \vee p_3 & e_4 \vee q_1 \\ e_3 \vee q_2 & \neg e_4 \vee s & \neg e_3 \vee r_1 & \neg e_3 \vee e_4 \vee r_2 \\ \neg q_2 \vee \neg p_2 & \neg q_1 \vee \neg r_2 \vee \neg p_3 & \neg r_1 \vee \neg s & \end{array}$$

In the remaining clauses, all variables appear both positively and negatively; the formula cannot be simplified any further. Notice, more importantly, that the solver is unable to find a conflict after tentatively setting e_1 to true when e_5 is set to false.

This happens because unit propagation in the proposed encoding is unable to detect that e_3, e_4, and e_5 are logical implications of e_1. Even when a solver tentatively sets e_1 to true, unit propagation only infers that e_2 must also be true, but not e_3 or e_4. It takes another decision, e.g., for e_3 or e_4, to derive a contradiction and, depending on the solver, possibly multiple steps to decide that e_1 must necessarily be false.

On the other hand, such information is easy to detect on the unfolding structure, and we shall modify $\phi_{\mathcal{P}}^{\text{dead}}$ in these cases.

Let us call *stubborn* any event e satisfying $({}^\bullet e \cup \underline{e})^\bullet = \{e\}$. Intuitively, once all events preceding e have fired, then firing e is unavoidable to find a deadlock. In Fig. 21, events e_2, e_3, e_4, e_5 are all stubborn.

Indeed, consider any deadlocked configuration $\mathcal{C}$ of $\mathcal{P}$, and let e be any stubborn event verifying ${}^\bullet({}^\bullet e \cup \underline{e}) \subseteq \mathcal{C}$. Then either e is in C or it is enabled at $cut(\mathcal{C})$, since $\mathcal{C}$ contains all events preceding e. But the latter is not possible because $\mathcal{C}$ is a deadlock, so e must be in C, which proves that $e \in C$ iff ${}^\bullet({}^\bullet e \cup \underline{e}) \subseteq \mathcal{C}$ (the other direction follows from the fact that $\mathcal{C}$ is causally closed).

This suggests that we could substitute every occurrence of e by a conjunction of the variables associated to the predecessors of e. We denote by E_s the set of stubborn events, and define inductively the set of *predecessors* of any event e as

$$pred(e) := {}^\bullet({}^\bullet e \cup \underline{e}) \setminus E_s \cup \bigcup_{e' \in {}^\bullet({}^\bullet e \cup \underline{e}) \cap E_s} pred(e')$$

The definition is well-given as a consequence of $<$ being well-founded.

Proposition 10. *If e is stubborn, then any deadlocked configuration $\mathcal{C}$ of $\mathcal{P}$ verifies that $e \in \mathcal{C}$ iff $pred(e) \subseteq \mathcal{C}$.*

Proof. Notice that $pred(e) \subseteq [e]$. If $e \in \mathcal{C}$, then obviously $pred(e) \subseteq \mathcal{C}$. We prove the opposite direction by induction on the size of $[e]$.

BASE. Let e be such that $|[e]| = 1$. Since $\mathcal{C}$ deadlocks, ${}^\bullet e \cup \underline{e} \not\subseteq cut(\mathcal{C})$ holds. Some event in $\mathcal{C}$ has thus consumed some condition of ${}^\bullet e \cup \underline{e} \subseteq \widehat{m}_0$, and it can only be e because it is stubborn.

STEP. Let $k \in \mathbb{N}$ be $k \geq 2$ and assume that the statement is true if $|[e]| < k$. Let now e be such that $|[e]| = k$. Again, since $\mathcal{C}$ deadlocks, ${}^\bullet e \cup \underline{e} \not\subseteq cut(\mathcal{C})$ holds. Since no event different than e can consume ${}^\bullet e \cup \underline{e}$, either $e \in \mathcal{C}$ or there exist $e' \in {}^\bullet({}^\bullet e \cup \underline{e})$ such that $e' \notin \mathcal{C}$. In the first case we are done. In the second we reach a contradiction, as we see now. First notice than e' has to be stubborn, since otherwise $e' \in pred(e)$ and $pred(e) \not\subseteq \mathcal{C}$, a contradiction. Because e' is stubborn and such that $|[e']| < k$ (since $e' < e$) and $pred(e') \subseteq pred(e) \subseteq \mathcal{C}$, the induction hypothesis applies, implying that $e' \in \mathcal{C}$. But this is a contradiction. □

Corollary 2. $\phi^{dead}_{\mathcal{P}} \equiv \phi^{dead}_{\mathcal{P}} \wedge \bigwedge_{e \in E_s} (\mathsf{e} \leftrightarrow \bigwedge_{e' \in pred(e)} \mathsf{e}')$

Corollary 2 can be exploited to modify $\phi^{\text{dead}}_{\mathcal{P}}$ in two ways: for every stubborn event e, (i) add a clause $\bigwedge_{e' \in {}^\bullet(\underline{e} \cup {}^\bullet e)} \mathsf{e}' \rightarrow \mathsf{e}$, or (ii) substitute e by $\bigwedge_{e' \in pred(e)} \mathsf{e}'$. Method (i), when applied to Fig. 21, will allow to derive a contradiction when e_1 is made true. On the other hand, when the solver sets e_4 to false, no information about the other events can be obtained through unit propagation. Method (ii) will eliminate the stubborn events from the encoding altogether. The resulting formula, after an initial unit propagation phase by the SAT solver, allows to immediately derive $\neg \mathsf{e}_1$.

We briefly explain the changes to $\phi^{\text{dead}}_{\mathcal{P}}$ motivated by method (ii): $\phi^{\text{sym}}_{\mathcal{P}}$ is not affected because no stubborn event appears in any symmetric conflict, and neither is $\phi^{\text{dis}}_{\mathcal{P}}$, which is only over variables for places. In $\phi^{\text{causal}}_{\mathcal{P}}$, however, clauses $\mathsf{e} \rightarrow \mathsf{e}'$ are discarded if e is stubborn or replaced by $\mathsf{e} \rightarrow \mathsf{e}''$ for every $e'' \in pred(e)$ if e is not stubborn. In a clause $\left(\mathsf{e} \wedge \bigwedge_{e' \in c^\bullet} \neg \mathsf{e}'\right) \rightarrow \mathsf{p}$ of $\phi^{\text{mark},M}_{\mathcal{P}}$, we need to replace e by a conjunction over $pred(e)$ if e is stubborn. In principle, the same needs to be done for e'. However, if $|c^\bullet| \geq 2$, then no event in $c^\bullet$ is stubborn, and nothing changes; but if $c^\bullet = \{e'\}$ is a singleton, and e' is stubborn, then the clause is split into $|pred(e')|$ different clauses. For $\phi^{\text{asym}}_{\mathcal{P}}$, in a clause of the form $\mathsf{e} \wedge \mathsf{f} \rightarrow [\![\mathsf{e} < \mathsf{f}]\!]$, both e and f are replaced

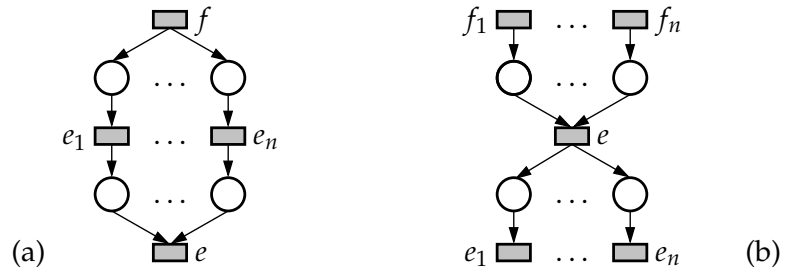


Figure 22: Good (a) and bad (b) cases for the stubborn event optimization

by conjunctions, if applicable; thus, the formula will still require ranks for e and f even if e or f are not present.

Method (ii) reduces the number of variables and either reduces (up to a linear factor) or increases (up to a quadratic factor) the number of clauses, as shown in Fig. 22. In (a), part of an occurrence net is shown. Without optimizations, n clauses of the form $\mathsf{e} \to \mathsf{e}_i$ and n of the form $\mathsf{e}_i \to \mathsf{f}$ are generated in $\phi_{\mathcal{P}}^{\text{causal}}$. Assuming that $e_1, \ldots, e_n$ are stubborn, method (ii) replaces these $2n$ clauses by one clause $\mathsf{e} \to \mathsf{f}$. Similar reductions occur in $\phi_{\mathcal{P}}^{\text{dis}}$ and $\phi_{\mathcal{P}}^{\text{mark},M}$. On the other hand, $\phi_{\mathcal{P}}^{\text{causal}}$ for (the fragment of) the occurrence net (b) has, when no optimization is used, n clauses of the form $\mathsf{e}_i \to \mathsf{e}$ and n of the form $\mathsf{e} \to \mathsf{f}_i$, but n^2 clauses of the form $\mathsf{e}_i \to \mathsf{f}_j$ when method (ii) is used. Nonetheless, in our experiments we used method (ii) due to the better behavior of unit propagation in the resulting encodings, as explained above.

We remark that stubborn events are also treated specially in the stable-models encoding of [Hel99c]. While stable models are similar to SAT, the treatment in [Hel99c] is simpler; its analogue in propositional logic would not eliminate stubborn events from the formula nor allow to directly conclude that e_1 cannot be fired.

It should be clear at this point that the optimization presented in this section only applies to the deadlock-checking encoding, not the coverability-checking encoding $\phi_{\mathcal{P}}^{\text{cov},M}$. Indeed, *only* in deadlocked configurations stubborn events can be equated to the conjunction of their predecessors.

4.7 ADDITIONAL SIMPLIFICATION

We briefly mention some possible simplifications of the formula. First, for a place p, if $p^\bullet \cup \underline{p} = \emptyset$, then p does not appear in $\phi_{\mathcal{P}}^{\text{dis}}$ and no condition c with $h(c) = p$ need to be considered on the left-hand side of the main conjunction in $\phi_{\mathcal{P}}^{\text{dis}}$. Similarly, during the construction of $\phi_{\mathcal{P}}^{\text{mark},M}$, only events e such that $h(e^\bullet)$ or $h({}^\bullet e)$ contains some place of M shall be included in $\phi_{\mathcal{P}}^{\text{conf}}$.

Secondly, a potentially more interesting simplification concerns subset checking. For two conditions c, d, if $c^\bullet \subseteq d^\bullet$, then $\mathsf{AMO}(c^\bullet)$ is implied by $\mathsf{AMO}(d^\bullet)$ and can be omitted from $\phi_{\mathcal{P}}^{\text{sym}}$. Similarly, for two transition t, u where ${}^\bullet t \subseteq {}^\bullet u$, disabledness of t implies disabledness of u, so u can be omitted from $\phi_{\mathcal{P}}^{\text{dis}}$. We return to this point in § 7.2.1.

4.8 CONCLUSIONS

We presented verification algorithms based on c-net unfoldings. Special attention was payed to the treatment of acyclicity in the asymmetric conflict relation and the optimization of the overall encoding. For acyclicity, we pro-

posed to deal separately with symmetric and asymmetric conflict cycles, and gave three optimizations to reduce the size of the asymmetric conflict relation. General optimizations of the overall encoding included the reduction of stubborn events and the elimination of subsumed disabled transitions. These optimizations operate at the level of the unfolding, in Ch. 7 we will discuss additional optimizations at the level of the SAT solver.

The encodings proposed here will be evaluated in Ch. 7. Experimental evaluation will show that solving times of these encodings beat the performance of existing unfolding-based deadlock-checking tools, a result that was not a foregone conclusion due to the richer structure of c-net unfoldings, in particular the presence of cycles and histories. Some of the optimizations proposed here will turn to be critical to achieve this.

5 CONTEXTUAL MERGED PROCESSES

Contextual unfoldings represent the state space of contextual nets. They cope with the state-space explosion due to concurrency and concurrent read access, as we have explained.

In this chapter, we integrate two compact representations: contextual unfolding prefixes and merged processes. The resulting representation, called *contextual merged processes (CMP),* combines the advantages of the original ones and copes with several important sources of state space explosion: concurrency, sequences of choices, and concurrent read accesses to shared resources. The chapter essentially presents theoretical results on the construction of CMPs and a reduction to SAT of the reachability problem based on CMPs. In Ch. 7 we demonstrate on a number of benchmarks that CMPs are more compact than either of the original representations.

This chapter is partially based on [RSK13].

5.1 INTRODUCTION

We explained in Ch. 1 that model checking is an important and practical way of ensuring the correctness of a system. However, it suffers from the *state-space explosion (SSE)* problem.

There are several common sources of SSE. One of them is concurrency, and the unfolding technique was primarily designed for efficient verification of highly concurrent systems. Indeed, a marking-complete prefix is often exponentially smaller than the corresponding reachability graph because it represents concurrency directly rather than by multidimensional *diamonds*, as it is done in reachability graphs. However, unfoldings do not cope well with some other important sources of SSE. In what follows, we consider two such sources.

One important source of SSE are sequences of choices. For example, the smallest complete prefix of the Petri net in Fig. 23 is exponential in its size since no event can be declared a cutoff — intuitively, each reachable marking *remembers* its past, and so different runs cannot lead to the same marking.

Recently, a technique has emerged that address this source of SSE, among others. In [KKKV06], a new condensed representation of Petri net behavior called *merged processes (MPs)* was proposed; it copes not only with concurrency, but also with sequences of choices. Moreover, this representation is sufficiently similar to the traditional unfoldings so that a large body of results developed for unfoldings can be re-used. The main idea behind MPs is to fuse some nodes in the unfolding prefix, and use the resulting net as the basis for verification. For example, the unfolding of the net shown in Fig. 23 will collapse back to the original net after the fusion. It turns out that

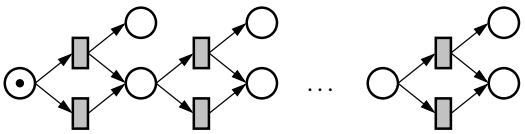


Figure 23: A Petri net with exponentially large unfolding prefix.

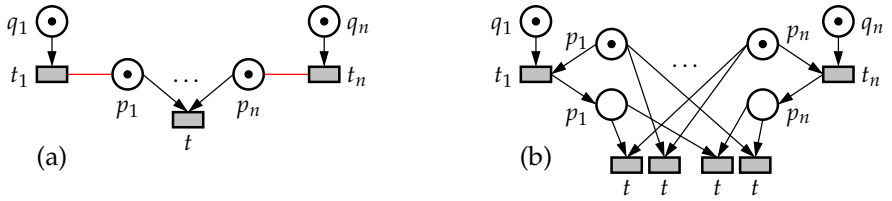


Figure 24: A c-net (a) whose contextual unfolding is isomorphic to the c-net itself, but whose plain encoding into a Petri net has exponentially large merged process, since no place instances in its unfolding (b) can be merged, and so there are 2^n mp-events corresponding to transition t. For this c-net the PR encoding coincides with the plain one, and so has the same unfolding and MP.

for a safe Petri net, model checking of a reachability-like property (i.e., the existence of a reachable state satisfying a predicate given by a Boolean expression) can be efficiently performed on its MP, and in [KKKV06] a polynomial reduction of this problem to SAT is presented. Furthermore, an efficient *unravelling algorithm* that builds a complete MP of a given safe ordinary net has been proposed in [KM11; KM13]. The experimental results in [KKKV06] indicate that this method is quite practical.

Unfortunately, MPs do not cope well with concurrent read access. As illustrated in Fig. 24, the contextual unfolding of a system may be exponentially smaller than its merged process. Similarly, contextual unfoldings suffer from SSE resulting from sequences of choices, e.g., they do not offer any improvement for the Petri net in Fig. 23, as it contains no read arcs.

We observe that in fact contextual unfoldings and merged process are techniques that address orthogonal aspects of the SSE problem when compressing an unfolding prefix into a more compact structure. They can thus be combined into one that copes with all the mentioned sources of SSE, viz.

1. concurrency,
2. sequences of choices, and
3. concurrent read accesses to a shared resource.

Moreover, there are striking similarities between the main complications that had to be overcome in the theories of MPs and c-net unfoldings: events have multiple local configurations, or histories (which causes difficulties in detection of cutoff events) and certain cycles (in the flow relation in case of MPs and in the asymmetric conflict relation in case of c-net unfoldings) have to be prohibited in valid configurations. Hence, the combination of the two techniques is not only possible, but also very natural.

The chapter is organised as follows. In § 5.2, the notion of a contextual merged process is presented. With the help of an example, in § 5.3 we deepen on the differences between the various ways to unfold and merge a c-net, clarifying the notion of CMP. In §§ 5.4 and 5.5, results to characterize the configurations of CMPs of safe c-nets and two encodings into SAT of the reachability problem using CMPs are presented. We use these results in § 5.6 to discuss the various ways to construct CMPS, and conclude in § 5.7.

5.2 CONTEXTUAL MERGED PROCESSES

In this section, we introduce the notion of *contextual merged processes* (CMPs) and discuss some of their properties. The results presented here generalize

those of [KKKV06]. In particular it turns out that the notion configuration of a contextual unfolding and mp-configuration of a merged process, defined in [KKKV06], both of which introduce acyclicity constraints, can be seamlessly integrated into a common framework.

Let us now fix notation for the rest of the chapter. Unless otherwise stated we let $N := \langle P, T, F, C, m_0 \rangle$ be a finite c-net satisfying the general assumptions in p. 18. Remark that N is not even necessarily bounded. We let $\mathcal{U}_N := \langle \langle \tilde{B}, \tilde{E}, \tilde{G}, \tilde{D}, \tilde{m}_0 \rangle, h \rangle$ denote the full unfolding of N and denote by $\mathcal{P} := \langle \langle B, E, G, D, \tilde{m}_0 \rangle, h \rangle$ an arbitrary unfolding prefix for N.

Recall that asymmetric conflict, causality, steps, runs, and reachable markings are, among other notions, preserved by homomorphisms, as a consequence of Lemma 1, on p. 19. The first step to define CMPs is the notion of occurrence depth.

Definition 15 (occurrence depth)**.** *Let x be a node of $\mathcal{P}$. The* occurrence depth *of x, denoted $\mathrm{od}(x)$, is the maximum number of $h(x)$-labelled nodes in any path in the directed graph*

$$(\tilde{m}_0 \cup [x] \cup [x]^\bullet, <_i)$$

starting at any initial condition and ending in x.

Recall that the cone $[x]$ is finite and that $<_i$ is a partial order, so there is only a finite number of paths to evaluate, and the definition is well-given.

As an example, in Fig. 25 (b), consider the condition c_9, and the associated digraph whose nodes are $\tilde{m}_0 \cup [c_9] \cup [c_9]^\bullet$ and whose edge relation is $<_i$. The occurrence depth of c_9 is 2 because the digraph contains the path $c_1 <_i e_2 <_i c_5 <_i e_4 <_i c_7 <_i e_6 <_i c_9$ and both c_5 and c_9 have the same label p_3.

A CMP is obtained from a branching process in two steps. First, all conditions that have the same label and occurrence depth are fused together (their initial markings are totalled); then all events that have the same label and environment (after fusing conditions) are merged. Conditions in the initial marking will have, by definition, occurrence depth 1. This is formalised as follows:

Definition 16 (Contextual Merged Process)**.** *The* Contextual Merged Process (CMP) *of $\mathcal{P}$ is the labelled c-net $\mathcal{Q} = \langle \langle \widehat{B}, \widehat{E}, \widehat{G}, \widehat{D}, \widehat{m}_0 \rangle, \widehat{h} \rangle$, with*

- $\widehat{B} \subseteq P \times \mathbb{N}$,
- $\widehat{E} \subseteq T \times 2^{\widehat{B}} \times 2^{\widehat{B}} \times 2^{\widehat{B}}$,
- $\widehat{h} \colon \widehat{B} \cup \widehat{E} \to P \cup T$ *is a homomorphism from $\mathcal{Q}$ to N,*

where $\mathcal{Q}$ is defined as follows. First, let $\hbar \colon B \cup E \to \widehat{B} \cup \widehat{E}$ be homomorphism from $\mathcal{P}$ to $\mathcal{Q}$ defined, for $b \in B$ and $e \in E$, by

$$\hbar(b) := \langle h(b), \mathrm{od}(b) \rangle$$
$$\hbar(e) := \langle h(e), \hbar({}^\bullet e), \hbar(\underline{e}), \hbar(e^\bullet) \rangle.$$

Then $\mathcal{Q}$ is defined by

1. $\widehat{B} := \hbar(B)$;
2. $\widehat{E} := \hbar(E)$;
3. $\widehat{G}, \widehat{D}$ *are such that for every $\widehat{e} = \langle t, X, Y, Z \rangle \in \widehat{E}$ we have ${}^\bullet\widehat{e} := X$ $\underline{\widehat{e}} := Y$ $\widehat{e}^\bullet := Z$;*
4. *for all $p \in P$, $\widehat{m}_0(\langle p, 1 \rangle) := 1$ and $\widehat{m}_0(\langle p, d \rangle) := 0$ if $d > 1$.*
5. $\widehat{h}$ *maps every node of $\mathcal{Q}$ to the first component of the tuple.*

The unravelling, *denoted $\mathcal{M}_N$, is the contextual merged process of $\mathcal{U}_N$.*

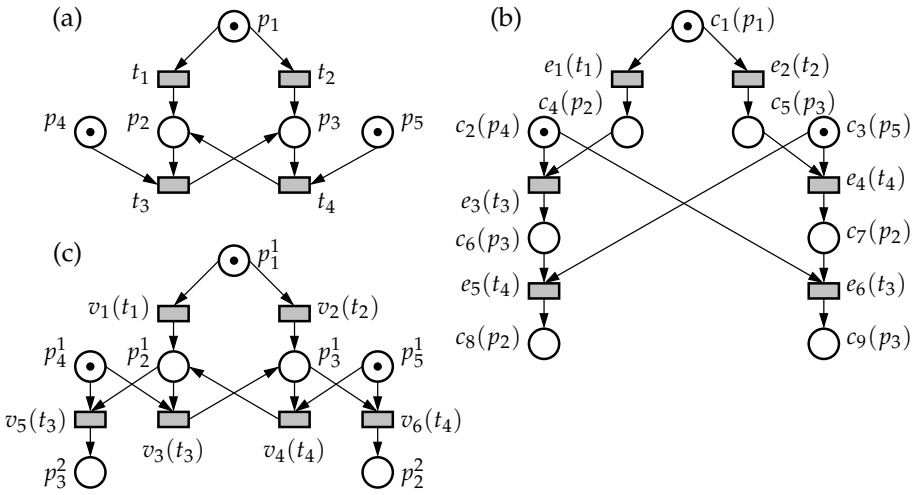


Figure 25: (a) A net; (b) its unfolding; (c) its unravelling.

Figure 25 shows a 1-safe net (taken from [KKKV06]), its unfolding, and its unravelling. For the rest of this chapter, unless otherwise stated, we let $\mathcal{Q} = \langle\langle \widehat{B}, \widehat{E}, \widehat{G}, \widehat{D}, \widehat{m}_0\rangle, \widehat{h}\rangle$ be the contextual merged processes of $\mathcal{P}$ and $\widehat{h}$ the associated homomorphism from $\mathcal{Q}$ to N. We additionally let $\hbar$ denote the homomorphisms from $\mathcal{P}$ to $\mathcal{Q}$.

The places of $\mathcal{Q}$ are called *mp-conditions* and its transitions *mp-events*. We shall write p^d for an mp-condition $\langle p, d\rangle$. Note that $\widehat{m}_0(p^d)$ equals $m_0(p)$ if $d = 1$ and is 0 otherwise. An mp-event $\widehat{e}$ is an *mp-cutoff* if all events in $\hbar^{-1}(\widehat{e})$ are cutoffs. Recall from Ch. 3 that speaking about cutoffs requires to first fix an strategy on $\mathcal{U}_N$. Indeed, here we are assuming that one strategy $\prec$ has been fixed, and that $\mathcal{P}$ is in fact $\mathcal{P}_N^{\prec}$, which allows to speak about the cutoff events of $\mathcal{P}$. We denote the set of mp-cutoffs by $\widehat{E}_{\text{cut}}$.

Remark 12. *The following properties hold for CMPs or c-net unfoldings:*

1. *In general, $\mathcal{M}_N$ is not acyclic; see Fig. 25 (c).*
2. *There can be mp-events consuming conditions in the postset of an mp-cutoff.*
3. *There is at most one mp-condition p^k resulting from fusing occurrences of place p at depth $k \geq 1$.*
4. *For two mp-conditions p^k and p^{k+1}, there is a directed path in the $<_i$ relation from the former to the latter.*
5. *Two different conditions c_1 and c_2 having the same label and occurrence depth are not causally related. Hence, if the original c-net is safe, then $\nearrow_{[c_1]\cup[c_2]}$ contains a cycle.*
6. *$h = \widehat{h} \circ \hbar$.*
7. *$\hbar$ and $\widehat{h}$ are homomorphisms.*
8. *A sequence of transitions σ is a run of N iff there exists a run $\widehat{\sigma}$ of $\mathcal{M}_N$ such that $\sigma = \widehat{h}(\widehat{\sigma})$.*

Additionally, if N is safe, we have:

9. *$\hbar$ is injective when restricted to the events of a configuration.*
10. *Property 8 is true if we additionally require $\widehat{\sigma}$ to be repetition-free.*

Proof. Properties 1 and 2 are already true for merged processes of ordinary Petri nets; 3 and 4 are immediate after the definition.

5 If the original c-net is safe, and c_1, c_2 have the same label and occurrence depth, then $[c_1] \cup [c_2]$ is not a configuration, since otherwise

they would be concurrent, producing two occurrences of the associated place (they cannot be causally related). Since $[c_1] \cup [c_2]$ is finite, it contains a cycle in $\nearrow$.

6 Any mp-event inherits the label associated to the events that were merged to produce it, so for any $e \in E$, we have $h(e) = \widehat{h}(\hbar(e))$.

7 By construction $\hbar(E) \subseteq \widehat{E}$, $\hbar(B) \subseteq \widehat{B}$, $\widehat{h}(E) \subseteq T$, and $\widehat{h}(B) \subseteq P$; so (10) holds for $\hbar$ and $\widehat{h}$. Also trivially (11) holds: $\mathbf{\hbar}(\tilde{m}_0) = \widehat{m}_0$ because mp-conditions with depth 1 map, through $\hbar$, to only conditions in $\tilde{m}_0$, and because $\tilde{m}_0$ is 1-safe due to the general assumptions in p. 18. Similarly $\widehat{\mathbf{h}}(\widehat{m}_0) = m_0$ because

$$m_0 = \mathbf{h}(\tilde{m}_0) = \mathbf{h}(\mathbf{\hbar}(\widehat{m}_0)) = \widehat{\mathbf{h}}(\widehat{m}_0).$$

For the rest of the proof, let $\widehat{e} \in \widehat{E}$ be an mp-event, $e \in E$ any event such that $\hbar(e) = \widehat{e}$, and $t = h(e) = \widehat{h}(\widehat{e})$.
We now show that $\hbar$ restricted to ${}^\bullet e$ is a bijection. Recall that ${}^\bullet\widehat{e}$ is defined as $\hbar({}^\bullet e)$, so we only need to show that $\hbar$ restricted to ${}^\bullet e$ is injective. Let $c, c' \in {}^\bullet e$. If $c \neq c'$ then $h(c) \neq h(c')$, because h is a homomorphism. Then c and c' cannot be merged and $\hbar(c) \neq \hbar(c')$. Analogous arguments prove that $\hbar$ restricted to $e^\bullet$ or $\underline{e}$ is also bijective. Finally, we show that $\widehat{h}$ restricted to ${}^\bullet\widehat{e}$ is bijective. The proof for $\widehat{e}^\bullet$ or $\underline{\widehat{e}}$ is similar. Let $\widehat{c}, \widehat{c}' \in {}^\bullet\widehat{e}$, and let $c, c' \in {}^\bullet e$ such that $\hbar(c) = \widehat{c}$ and $\hbar(c') = \widehat{c}'$. Because h is a homomorphism, $h(c) \neq h(c')$. Then $h(c) = \widehat{h}(\widehat{c}) \neq h(c') = \widehat{h}(\widehat{c}')$. Hence $\widehat{h}$ restricted to ${}^\bullet\widehat{e}$ is injective. To see why it is surjective, let $p \in {}^\bullet t$. Again, because h is a homomorphism, there is a single $c \in {}^\bullet e$ such that $h(c) = p$. Let $\widehat{c} = \hbar(c)$. Recall that $h(c) = \widehat{h}(\widehat{c}) = p$. Because $\hbar$ is a homomorphism, $\widehat{c} \in {}^\bullet\widehat{e}$, and so $\widehat{h}$ restricted to ${}^\bullet\widehat{e}$ is surjective.

8 For any run $\widehat{\sigma}$ of $\mathcal{M}_N$, $\widehat{h}(\widehat{\sigma})$ is a run of N, by Lemma 1. For any run σ of N, there is, by the properties of $\mathcal{U}_N$, a run $\tilde{\sigma}$ in $\mathcal{U}_N$ such that $h(\tilde{\sigma}) = \sigma$. Then $\widehat{\sigma} := \hbar(\tilde{\sigma})$ is a run of $\mathcal{M}_N$ that satisfies $\widehat{h}(\widehat{\sigma}) = h(\tilde{\sigma}) = \sigma$.

9 Let $\mathcal{C}$ be a configuration of $\mathcal{P}_N$. We prove that $e \neq e'$ implies $\hbar(e) \neq \hbar(e')$ for all $e, e' \in \mathcal{C}$. For an argument by contradiction, assume $\hbar(e) = \hbar(e')$. Events e and e' have been merged into the same mp-event, so $h(e) = h(e')$. Since $e \neq e'$, by (21) either ${}^\bullet e \neq {}^\bullet e'$ or $\underline{e} \neq \underline{e'}$, which implies that there exists some $c \in {}^\bullet e \cup \underline{e}$ and $c' \in {}^\bullet e' \cup \underline{e'}$ such that $c \neq c'$ but both c and c' are labelled by p and have occurrence depth k. By property 5, $[c] \cup [c']$ contains a cycle in relation $\nearrow$, but $[c] \cup [c'] \subseteq \mathcal{C}$. This is a contradiction.

10 Any repetition-free run $\widehat{\sigma}$ of $\mathcal{M}_N$ is, by property 8, such that $\widehat{h}(\widehat{\sigma})$ is a run of N. Now, let σ be a run of N. We know (cf. proof of property 8) that there is a run $\tilde{\sigma}$ of $\mathcal{U}_N$ with $h(\tilde{\sigma}) = \sigma$ and that there is a run $\widehat{\sigma}$ of $\mathcal{M}_N$ verifying $\hbar(\tilde{\sigma}) = \widehat{\sigma}$ and $\widehat{h}(\widehat{\sigma}) = \sigma$. We now prove that $\widehat{\sigma}$ is repetition-free. Recall that the set $\{e \in E \mid e \text{ fires in } \tilde{\sigma}\}$ is a configuration of $\mathcal{U}_N$, and that $\tilde{\sigma}$ is repetition-free. If $\widehat{\sigma}$ was not repetition-free, some mp-event $\widehat{e}$ would fire two times, implying that there are two different events e, e' that fire in $\tilde{\sigma}$ such that $\hbar(e) = \hbar(e')$. This contradicts property 9. □

Note that Property 9 is still true when $\hbar$ is restricted to the elements of $\tilde{m}_0 \cup \mathcal{C} \cup \mathcal{C}^\bullet$. Indeed, $\hbar$ is bijective when restricted to $\tilde{m}_0$, because $\widehat{m}_0$ is safe, and two conditions $c, c' \in \mathcal{C}^\bullet$ cannot be merged because $\nearrow_{[c]\cup[c']}$ would have cycles and $[c] \cup [c'] \subseteq \mathcal{C}$.

Definition 17 (mp-configuration). *A multiset of mp-events $\widehat{\mathcal{C}}$ is an* mp-configuration *of $\mathcal{Q}$ if there exists a configuration $\mathcal{C}$ of $\mathcal{U}_N$ verifying $\hbar(\mathcal{C}) = \widehat{\mathcal{C}}$.*

As it is the case for configurations of branching processes, any mp-configuration of a merged process represents a (concurrent) run of its mp-events, i.e., there exists at least one linear ordering of the mp-events of $\widehat{\mathcal{C}}$ that is a run of the merged process. This is because the same is true for configurations of the associated branching process and because $\hbar$ is a homomorphism.

Every finite firing sequence of $\mathcal{U}_N$ consists of a set of events that form a configuration $\mathcal{C}$, which, due to Def. 17, corresponds to an mp-configuration $\widehat{\mathcal{C}}$ of $\mathcal{M}_N$. However, the inverse statement is not true: a firing sequence of $\mathcal{M}_N$ may consist of a multiset of events X that is not an mp-configuration since no $\mathcal{C} \in \mathit{conf}(\mathcal{U}_N)$ satisfies $\hbar(\mathcal{C}) = X$. This already holds for ordinary nets, as the example in Fig. 25 shows: v_1v_5 is a valid firing sequence of $\mathcal{M}_N$ corresponding to events e_1 and e_6 of $\mathcal{U}_N$ (i.e. $\hbar(e_1) = v_1$ and $\hbar(e_6) = v_5$) which do not form a configuration. However, $\widehat{h}$ applied to v_1v_5 still gives a valid firing sequence t_1t_3 of N thanks to Rmk. 12 (8).

We shall regard a CMP $\mathcal{Q}$ as marking-complete when all the markings of N are represented, not by arbitrary runs of $\mathcal{Q}$, but by those associated to mp-configurations of $\mathcal{Q}$.

Definition 18 (marking-complete CMP). *Let X be a finite multiset of mp-events. The* cut *and* marking *of X are respectively defined as the multisets*

$$\mathit{cut}(X) := (\widehat{m}_0 + X^\bullet) - {}^\bullet X$$
$$\mathit{mark}(X) := \widehat{\mathbf{h}}(\mathit{cut}(X)).$$

We call $\mathcal{Q}$ marking-complete *if for each reachable marking m of N there exists a cutoff-free mp-configuration $\widehat{\mathcal{C}}$ in $\mathcal{Q}$ satisfying $\mathit{mark}(\widehat{\mathcal{C}}) = m$.*

The intuition behind these definitions is as follows. If X is the multiset of mp-events associated to a finite run (i.e., the multiset M such that $M(\widehat{e}) = n$ if $\widehat{e}$ fires n times) then $\mathit{cut}(X)$ is the marking reached by this run in the CMP, and $\mathit{mark}(X)$ is the $\widehat{h}$-image of $\mathit{cut}(X)$, i.e., the corresponding marking of N.

Observe that in the definition of a marking-complete CMP, one could ask for a finite run (rather than a configuration) that reaches a marking m. The resulting definition would be equivalent, but we preferred the current variant because it (i) mimics the analogous definition for unfoldings and (ii) avoids some unpleasant properties of runs: e.g., finite CMPs can have infinite runs and therefore infinitely many finite runs, which is impossible for configurations.

5.3 INTERPLAY BETWEEN READ-ARCS AND CHOICE

Let us now illustrate the benefits of using CMPs on a concrete family of c-nets. We compare the various approaches to unfold and merge such c-nets, and explain why CMPs, on this family, achieve such gains.

Our family of c-net examples is called n-GEN, and the instance for $n = 2$ is shown in Fig. 26. The net represents n processes that concurrently generate resources r_i. Once all resources r_i are produced, an action t consumes them all. Resource r_i can be produced if one of two conditions is fulfilled, symbolised by transitions t_i or t'_i. Thus, t_i, t'_i share context with transitions t_j and t'_j, respectively, whenever $j \neq i$.

For some $n \geq 1$, let N_c be the c-net n-GEN, N_p its plain encoding, and N_r its PR encoding. The unfoldings of the three nets and the MPs of N_p and N_r

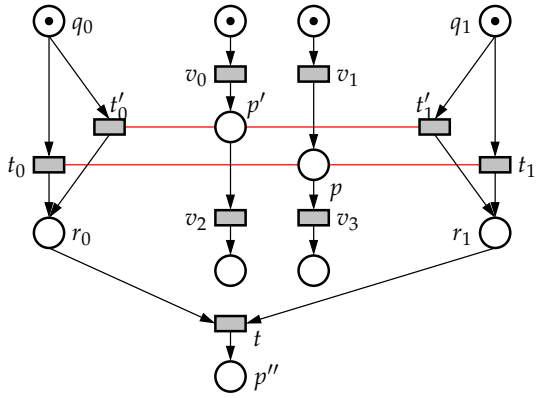


Figure 26: The c-net 2-GEN.

blow up due to at least one of the following reasons, which we explain in the sequel: (a) choices between t_i and t'_i or (b) sequentialized read access to p and p'.

For (a), notice that process i can produce r_i in two different ways. At least two occurrences of each r_i are thus present in the unfolding of any of the three nets. Hence there are at least 2^n ways of choosing t's preset, i.e., at least 2^n occurrences of t and p'' in any of the three unfoldings.

Roughly speaking, (b) refers to the same phenomena that were demonstrated in Fig. 5 and discussed in § 1.4, on p. 8. While all t_i are concurrent in N_c, they are sequentialized in N_p: they all consume and produce the same p. This creates conflicts between them, and as a result all their exponentially many interleavings are explicitly present in $\mathcal{U}_{N_p}$. Importantly, any occurrence of t_i that consumes an occurrence of p at depth d, produces an occurrence of p at depth $d+1$.

In N_r, even if all t_i are still concurrent to each other, their occurrences produce two conditions with occurrence depths 1 and 2, each labelled by their respective private copy of p. For $\mathcal{U}_{N_r}$, this again has the consequence of producing 2^n ways of choosing v_3's preset, and 2^n events labelled by v_3. More importantly, the private copies of t_i cannot be merged with those of t_j and they remain in $\mathcal{Q}_{N_r}$. As a result, all 2^n occurrences of v_3 are also present in the MP of N_r. This suggests that MPs of PR unfoldings may not yield, in general, much gain.

While the size of the contextual unfolding of N_c explodes due to (a), it is unaffected by (b). On the other hand, the MP of N_p effectively deals with (a), but only partially with (b). We now see why. Notice that there are $\mathcal{O}(2^n)$ conditions labelled by p in $\mathcal{U}_{N_p}$, all with occurrence depths between 1 and $n+1$. In the MP, they are merged into the $n+1$ mp-conditions $p^1, \ldots, p^{n+1}$. Since all instances of q_i and r_i have occurrence-depth 1, all the exponentially many events labelled by t_i are merged into n mp-events, each consuming some p^j and producing p^{j+1}, for $1 \leq j \leq n$. This yields an MP of size $\mathcal{O}(n^2)$.

Finally, the CMP of N_c deals effectively with both (a) and (b); it is, in fact, isomorphic to N. Roughly speaking, this is because the unfolding of N_c already deals with (b), as we said, and the '*merging*' solves (a). Thus, the CMP is polynomially more compact than the MP of N_p and exponentially more than the MP of N_r, or the unfoldings of N_c, N_p, or N_r. See Table 4 for a summary.

Table 4: Growth of contextual, plain, and PR unfoldings and MPs for the collection of c-nets n-GEN.

Merged Processes			Unfoldings		
Ctx	Plain	PR	Ctx	Plain	PR
$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(2^n)$	$\mathcal{O}(2^n)$	$\mathcal{O}(2^n)$	$\mathcal{O}(2^n)$

While this example in itself is artificial, the underlying structures are quite simple and commonly occur in more complex c-nets, which explains some of the experimental results below.

5.4 CHARACTERIZING REACHABILITY

In this section we present two results that pave the way to a practical reachability algorithm based on CMPs. We give, for safe nets, characterizations of sets of mp-events that correspond to reachable markings of N (Prop. 11) and to configurations of $\mathcal{Q}$ (Prop. 12). These characterizations will be used in the next section to produce SAT encodings of the reachability problem based on CMPs. In particular, they will serve to aid CMP-based model-checking, as well as the direct construction of CMPs.

We note that the problem of generalizing these approaches to bounded, but not safe, nets is still open even for merged processes without read arcs [KKKV06].

Therefore we focus on the practically relevant class of safe c-nets. Here, the mapping $\hbar$ lifted to configurations of $\mathcal{U}_N$ establishes an injective correspondence between the configurations of the unfolding and the mp-configurations of the unravelling. For each mp-configuration $\widehat{\mathcal{C}}$ there exists a *unique* configuration $\mathcal{C}$ of $\mathcal{U}_N$ such that $\widehat{\mathcal{C}} = \hbar(\mathcal{C})$.

Our first step is to characterize a *subset* of repetition-free runs of $\mathcal{Q}$ that is large enough to include all interleavings of all mp-configurations. It thus contains sufficient firing sequences to characterize the set of reachable markings of N, as discussed further below.

Lemma 6. *Assume N is safe, and let X be any set of mp-events of $\mathcal{Q}$ satisfying:*

1. $^\bullet X \cup \underline{X} \subseteq \widehat{m}_0 \cup X^\bullet$, *and* (60)
2. $\nearrow_X$ *is acyclic.* (61)

Then any linear extension of $\nearrow_X$ is a repetition-free firing sequence of $\mathcal{Q}$. Moreover, any mp-configuration of $\mathcal{Q}$ satisfies (60) and (61).

Proof. Let $\widehat{e}_1, \ldots, \widehat{e}_n$ be a total order on X that is compatible with $\nearrow$, i.e., such that $\widehat{e}_i \nearrow \widehat{e}_j$ implies $i < j$ for all $1 \leq i, j \leq n$. We prove, by induction on n, that such sequence is a run of $\mathcal{Q}$.

We show that $\widehat{e}_1$ is enabled at the initial marking. Condition (60) says that all $\widehat{c} \in {}^\bullet\widehat{e}_1 \cup \underline{\widehat{e}_1}$ are initially marked or generated by some other mp-event in X. Since $\widehat{e}_1$ is $\nearrow$-minimal, no event $\widehat{e}_j$ satisfies $\widehat{e}_j < \widehat{e}_1$, so $\widehat{c}$ is initially marked.

We now show that, if $\widehat{e}_1, \ldots, \widehat{e}_k$ is a run, then $\widehat{e}_1, \ldots, \widehat{e}_k, \widehat{e}_{k+1}$ is a run too, for $1 \leq k < n$. Assume the hypothesis. Let $\widehat{C}_k := \{\widehat{e}_1, \ldots, \widehat{e}_k\}$ and let $\widehat{m} := \mathit{cut}(\widehat{C}_k)$ be the cut reached after firing $\widehat{e}_1, \ldots, \widehat{e}_k$. We prove that any $\widehat{c} \in {}^\bullet\widehat{e}_{k+1} \cup \underline{\widehat{e}_{k+1}}$ verifies $\widehat{c} \in \widehat{m}$. By condition 1, we know that $\widehat{c} \in \widehat{m}_0$ or $\widehat{c} \in X^\bullet$. Since any mp-event $\widehat{e}_j$ in the preset of $\widehat{c}$ is such that $\widehat{e}_j \nearrow \widehat{e}_i$, we know that

$\widehat{e}_j \in \widehat{C}_k$, and $\widehat{c} \in \widehat{C}_k^\bullet$. So $\widehat{c} \in \widehat{C}_k^\bullet \cup \widehat{m}_0$. It remains to prove that $\widehat{c} \notin {}^\bullet\widehat{C}_k$. If this was not the case, then some $\widehat{e}_l \in \widehat{C}_k$ would be such that $\widehat{c} \in {}^\bullet\widehat{e}_l$, and also such that $\widehat{e}_i \nearrow \widehat{e}_l$, since either ${}^\bullet\widehat{e}_i \cap {}^\bullet\widehat{e}_l \neq \emptyset$ or $\underline{\widehat{e}_i} \cap {}^\bullet\widehat{e}_l \neq \emptyset$. But then $i < l$, and $\widehat{e}_l \notin \widehat{C}_k$, a contradiction. So $\widehat{c} \in (\widehat{C}_k^\bullet \cup \widehat{m}_0) \setminus {}^\bullet\widehat{C}_k = cut(\widehat{C}_k)$, and $\widehat{e}_1, \ldots, \widehat{e}_k, \widehat{e}_{k+1}$ is a run.

As for the second statement, let $\widehat{C}$ be an mp-configuration of $\mathcal{Q}$ and C a configuration of $\mathcal{P}$ such that $\hbar(C) = \widehat{C}$. We show that $\widehat{C}$ satisfies (60) and (61). Note that $\widehat{m}_0$, $\widehat{C}$, and $\widehat{C}^\bullet$ are sets rather than general multisets, essentially because N is safe and $\hbar$ restricted to C is injective.

- Since $\widehat{C}$ is an mp-configuration, there is a linear order $\widehat{e_1}, \ldots, \widehat{e_n}$ on the mp-events of $\widehat{C}$ that is a run of $\mathbb{N}$. Let $\widehat{e_i}$ be any mp-event of $\widehat{C}$, and let $\widehat{c} \in {}^\bullet\widehat{e_i} \cup \underline{\widehat{e_i}}$ be any mp-condition on its preset or context. We prove that either $\widehat{c} \in \widehat{m_0}$ or ${}^\bullet\widehat{c} \cap \widehat{C} \neq \emptyset$. This is trivially true for $\widehat{e_1}$, since it is enabled at the initial marking, so assume that $i \geq 2$. The sequence $\widehat{e_1}, \ldots, \widehat{e_{i-1}}$ is a run, let $\widehat{m}$ be the marking it reaches. Then $\widehat{e_i}$ is enabled at $\widehat{m}$, so $\widehat{c} \in \widehat{m}$. So either $\widehat{c}$ is initially marked or there is some mp-event $\widehat{e_j} \in \widehat{C}$, $1 \leq j < i$, such that $\widehat{e_j} \in {}^\bullet\widehat{c}$, which shows (60).
- Since $\hbar$ restricted to C is injective, $\hbar^{-1}$ restricted to $\widehat{C}$ is an injective function; furthermore, it is a homomorphism. The absence of cycles in $\nearrow_{\widehat{C}}$ follows from the properties of homomorphisms. Specifically, if, by contradiction, it was possible to find a cycle

  $$\widehat{e_1} \nearrow \ldots \nearrow \widehat{e_n} \nearrow \widehat{e_1}$$

  in $\widehat{C}$, then, using Lemma 1, we could also find the cycle

  $$\hbar^{-1}(\widehat{e_1}) \nearrow \ldots \nearrow \hbar^{-1}(\widehat{e_n}) \nearrow \hbar^{-1}(\widehat{e_1})$$

  in C, which would contradict (18). □

Lemma 6 identifies a subset of repetition-free runs of $\mathcal{Q}$. Observe that not every repetition-free run satisfies the two conditions: $v_1v_3v_4$ is a repetition-free run of Fig. 25 but $\{v_1, v_3, v_4\}$ violates (61). This means that Lemma 6 characterizes a *strict subset* of repetition-free runs, which is, however, large enough to contain all interleavings of all mp-configurations, and therefore enough runs for representing *all* reachable markings of N. This fact is exploited in Prop. 11.

A key detail in both results is that acyclicity of $\nearrow$ prohibits, at the same time, asymmetric conflicts inherent to c-net unfoldings, as those in Fig. 10, on p. 23, and cycles in the flow relation introduced by merging, as shown in Fig. 25 (c).

Using Lemma 6, we can now characterize the reachable markings of N in terms of a marking-complete CMP $\mathcal{Q}$ of N. In essence, the following result states that $reach(N)$ is the $\widehat{h}$-image of the markings reached by a *subset* of $\mathcal{Q}$'s repetition-free runs.

Proposition 11. *If N is safe and $\mathcal{Q}$ marking-complete, a marking m is reachable in N iff there is a cutoff-free set X of mp-events of $\mathcal{Q}$ satisfying:*

1. *properties (60) and (61), and*
2. $m = mark(X)$. (62)

Proof. Let m be a reachable marking of N. Then there is a mp-configuration X of $\mathcal{Q}$, free of cutoffs and satisfying $mark(X) = m$. As Lemma 6 shows, X also satisfies (60) and (61).

For the opposite direction, if X satisfies all the three properties, it clearly identifies a marking m reachable in N. By Lemma 6, any linear extension of

$\nearrow_X$ is a run of $\mathcal{Q}$, that reaches a cut $cut(X)$ of $\mathcal{Q}$. Such reachable marking of $\mathcal{Q}$ maps, through $\widehat{h}$, to a reachable marking of N, thanks to Lemma 1. □

We now extend Lemma 6 with two additional constraints to fully characterize the mp-configurations of $\mathcal{Q}$. The first condition asks that all mp-conditions $p^1, \ldots, p^{k-1}$ are present in the candidate set if p^k is also present, and the second ensures that they are *visited* in the right order.

Proposition 12. *If N is safe, a set of mp-events $\widehat{\mathcal{C}}$ is an mp-configuration of $\mathcal{Q}$ iff it satisfies the following conditions, for any $p^{k+1} \in \widehat{\mathcal{C}}^\bullet$ with $k \geq 1$:*

1. *(60) and (61) hold for $\widehat{\mathcal{C}}$, and*
2. *$p^k \in \widehat{m}_0 \cup \widehat{\mathcal{C}}^\bullet$, and* (63)
3. *the digraph $(\widehat{m}_0 \cup \widehat{\mathcal{C}} \cup \widehat{\mathcal{C}}^\bullet, <_i)$ has a path from p^k to p^{k+1}.* (64)

Proof. Assume that $\widehat{\mathcal{C}}$ is an mp-configuration. By Lemma 6 it satisfies (60) and (61). We show it also satisfies (63) and (64), essentially due to the properties of $\hbar$. Let $\mathcal{C}$ be the such that $\hbar(\mathcal{C}) = \widehat{\mathcal{C}}$. Recall that $\widehat{m}_0$, $\widehat{\mathcal{C}}$, and $\widehat{\mathcal{C}}^\bullet$ are sets rather than general multisets.

- Assume that $p^{k+1} \in \widehat{\mathcal{C}}^\bullet$, and let $c \in B$ be the single condition in $\tilde{m}_0 \cup \mathcal{C}^\bullet$ such that $\hbar(c) = p^{k+1}$. Condition c exists and is unique by property 9 in Rmk. 12. Since c has occurrence depth $k+1$, there exists c' with occurrence depth k and label p, such that $c' < c$, and therefore $c' \in \tilde{m}_0 \cup \mathcal{C}^\bullet$. Then $\hbar(c') = p^k$, and $p^k \in \widehat{m}_0 \cup \widehat{\mathcal{C}}^\bullet$. This shows (63).
- Let c and c' be as in the previous paragraph. Let $e_1, \ldots, e_l$ be events verifying

  $$c' <_i e_1, \text{ and } e_l <_i c, \text{ and } e_i^\bullet \cap ({}^\bullet e_{i+1} \cup \underline{e_{i+1}}) \neq \emptyset$$

  for $1 \leq i < l$. They exists because $c' < c$, and they identify a path from c' to c in $(\tilde{m}_0 \cup \mathcal{C} \cup \mathcal{C}^\bullet, <_i)$, that $\hbar$ maps to a path identified by $\hbar(e_1), \ldots, \hbar(e_l)$ from p^k and p^{k+1} in the digraph $(\widehat{m}_0 \cup \widehat{\mathcal{C}} \cup \widehat{\mathcal{C}}^\bullet, <_i)$. This shows (64).

We now prove, by induction, the opposite direction. We show that any set $\widehat{\mathcal{C}}$ of mp-events satisfying the four properties above is an mp-configuration.

For the base case, assume that $\widehat{\mathcal{C}} = \emptyset$. Clearly, $\emptyset$ is a configuration of $\mathcal{U}_N$, and $\hbar(\emptyset) = \emptyset$. For the inductive step, assume that the statement is true if $\widehat{\mathcal{C}}$ has at most $n \geq 0$ mp-events, and let $\widehat{\mathcal{C}}$ be a set of $n+1$ mp-events. Let $\widehat{e}$ be any $\nearrow$-maximal mp-event in $\widehat{\mathcal{C}}$, which exists by (61), and let $\widehat{\mathcal{C}}' := \widehat{\mathcal{C}} \setminus \{\widehat{e}\}$. Recall that no mp-event $\widehat{e}' \in \widehat{\mathcal{C}}'$ satisfies $\widehat{e} < \widehat{e}'$ because $\widehat{e}$ is $\nearrow$-maximal. We prove that $\widehat{\mathcal{C}}'$ satisfies all the four conditions above.

- Property (60). Assume that $\widehat{\mathcal{C}}'$ violates (60). Then there is some $\widehat{e}' \in \widehat{\mathcal{C}}'$ such that $\widehat{e}^\bullet \cap ({}^\bullet \widehat{e}' \cup \underline{\widehat{e}'}) \neq \emptyset$. But this is not possible because $\widehat{e} < \widehat{e}'$.
- Property (61) is trivially true for $\widehat{\mathcal{C}}'$ as it was already for $\widehat{\mathcal{C}}$.
- Property (63). For some $k \geq 1$ and place $p \in P$, assume that $p^{k+1} \in \widehat{\mathcal{C}}'^\bullet$ but $p^k \notin \widehat{m}_0 \cup \widehat{\mathcal{C}}'^\bullet$. Then $p^k \in \widehat{e}^\bullet$ because $p^k \in \widehat{m}_0 \cup \widehat{\mathcal{C}}^\bullet$. Since $\widehat{\mathcal{C}}$ satisfies (64), there is a path in $(\widehat{m}_0 \cup \widehat{\mathcal{C}} \cup \widehat{\mathcal{C}}^\bullet, <_i)$ from p^k to p^{k+1}, so there exists some $\widehat{e}' \in \widehat{\mathcal{C}}$ with $p^k < \widehat{e}' < p^{k+1}$. But then, $\widehat{e} < \widehat{e}'$, a contradiction, so $p^k \in \widehat{\mathcal{C}}'^\bullet$.
- Property (64). To show that there is a path in $(\widehat{m}_0 \cup \widehat{\mathcal{C}}' \cup \widehat{\mathcal{C}}'^\bullet, <_i)$ from p^k to p^{k+1}, let $\widehat{e}_1, \ldots, \widehat{e}_l \in \widehat{\mathcal{C}}$ be mp-events satisfying

  $$p^k <_i \widehat{e}_1 < \ldots < \widehat{e}_l <_i p^{k+1}, \text{ and } \widehat{e}_i^\bullet \cap ({}^\bullet \widehat{e}_{i+1} \cap \underline{\widehat{e}_{i+1}}) \neq \emptyset$$

  for $1 = 1, \ldots, l$ and for some $l \geq 1$. We prove that $\widehat{e} \neq \widehat{e}_i$, for any $i = 1, \ldots, l$, and therefore $\widehat{e}_i \in \widehat{\mathcal{C}}'$. Obviously, for $i = 1, \ldots, l-1$ we have

that $\widehat{e} \neq \widehat{e}_i$, since otherwise $\widehat{e}_{i+1}$ would satisfy $\widehat{e} < \widehat{e}_{i+1}$, contradicting that $\widehat{e}$ is $\nearrow$-maximal in $\widehat{\mathcal{C}}$. So we only have to check that $\widehat{e} \neq \widehat{e}_l$. Assume, for a proof by contradiction, that $e = \widehat{e}_l$. Then, $p^{k+1} \in \widehat{e}^\bullet$. Since $p^{k+1} \in \widehat{\mathcal{C}}'^\bullet$ and because $\widehat{e} \notin \widehat{\mathcal{C}}'$, there is some other $\widehat{e}' \in \widehat{\mathcal{C}}'$ such that $p^{k+1} \in \widehat{e}'^\bullet$. So p^{k+1} has at least two different mp-events in its preset. Now, by Lemma 6, all events in $\widehat{\mathcal{C}}$ can be ordered to form a run of $\mathcal{Q}$. Then, since N is safe, there exists some mp-event $\widehat{e}'' \in \widehat{\mathcal{C}}$ that consumes p^{k+1}. But $\widehat{e}$ is $\nearrow$-maximal in $\widehat{\mathcal{C}}$, so necessarily $\widehat{e} = \widehat{e}''$. So $\widehat{e}$ consumes and produces p^{k+1}. Any event in $\hbar^{-1}(\widehat{e})$ obviously consumes one condition labelled by p and produces a *different* one, also labelled by p; and both conditions have been merged into p^{k+1}. This is a contradiction, both conditions have different occurrence depth.

Since $\widehat{\mathcal{C}}'$ satisfies all the three conditions, by induction hypothesis, there exists some configuration $\mathcal{C}'$ of $\mathcal{U}_N$ such that $\hbar(\mathcal{C}') = \widehat{\mathcal{C}}'$. In the sequel, we prove that there exists some event e enabled at $cut(\mathcal{C}')$ such that $\hbar(e) = \widehat{e}$, and therefore $\hbar(\mathcal{C}' \cup \{e\}) = \widehat{\mathcal{C}}' \cup \{\widehat{e}\} = \widehat{\mathcal{C}}$. Then $\widehat{\mathcal{C}}$ is an mp-configuration because $\mathcal{C}' \cup \{e\}$ is a configuration.

Let $m := cut(\mathcal{C}')$ be the cut of $\mathcal{C}'$, and $\widehat{m} := cut(\widehat{\mathcal{C}}')$ be the cut of $\widehat{\mathcal{C}}'$. Recall that $\hbar(m) = \widehat{m}$, because $\hbar$ is a homomorphism. The mp-event $\widehat{e}$ is necessarily enabled at $\widehat{m}$. Then some transition $\widehat{h}(\widehat{e})$ is enabled at the marking $\widehat{h}(\widehat{m}) = h(m)$ of N, and therefore, some event e exists in $\mathcal{U}_N$ such that $h(e) = \widehat{h}(\widehat{e})$. Note that e is unique, due to (21). We now show that $\hbar({}^\bullet e) = {}^\bullet\widehat{e}$, and $\hbar(\underline{e}) = \underline{\widehat{e}}$, and $\hbar(e^\bullet) = \widehat{e}^\bullet$, which implies that $\hbar(e) = \widehat{e}$.

- We prove that $\hbar({}^\bullet e) = {}^\bullet\widehat{e}$, the case for $\hbar(\underline{e}) = \underline{\widehat{e}}$ is analogous. Let $c \in {}^\bullet e$ be a condition in the preset of e. We show that $\hbar(c) \in {}^\bullet\widehat{e}$. Since e is enabled at m, then $c \in m$, and $\hbar(c) := p^k \in \widehat{m}$. Since $h(e) = \widehat{h}(\widehat{e})$, there exists some mp-condition $p^{k'} \in {}^\bullet\widehat{e}$. Also, $p^{k'} \in \widehat{m}$, because $\widehat{e}$ is enabled at $\widehat{m}$. But $p^k = p^{k'}$, since otherwise $\widehat{h}(\widehat{m})$ would not be safe. This proves that $\hbar({}^\bullet e) \subseteq {}^\bullet\widehat{e}$. Recall that $|{}^\bullet e| = |{}^\bullet\widehat{e}|$, so necessarily $\hbar({}^\bullet e) = {}^\bullet\widehat{e}$.
- We prove that $\hbar(e^\bullet) = \widehat{e}^\bullet$. Since $|e^\bullet| = |\widehat{e}^\bullet|$, because $h(e) = \widehat{h}(\widehat{e})$, and since $\hbar$ is injective when restricted to $e^\bullet$, we only need to prove that $\hbar(e^\bullet) \subseteq \widehat{e}^\bullet$. Let $c \in e^\bullet$ be such that $\hbar(c) = p^k$. We know that some mp-condition $p^{k'}$ is present in $\widehat{e}^\bullet$. We prove that (*i*) $k \geq k'$ and (*ii*) $k' \geq k$, and therefore $k = k'$.

  i Because $\widehat{\mathcal{C}}$ satisfies (64), there is a path $\widehat{\sigma}$ in the directed graph $(\widehat{m}_0 \cup \widehat{\mathcal{C}} \cup \widehat{\mathcal{C}}^\bullet, <_i)$ from some initial condition to $p^{k'}$ that visits $p^1, p^2, \ldots, p^{k'}$ in that order. The last two elements of $\widehat{\sigma}$ are $\widehat{e}$ and $p^{k'}$, and when removed from $\widehat{\sigma}$, the resulting path $\widehat{\sigma}'$ is also a path in the graph $(\widehat{m}_0 \cup \widehat{\mathcal{C}}' \cup \widehat{\mathcal{C}}'^\bullet, <_i)$. Since $\mathcal{C}'$ is a configuration, $\hbar$ restricted to it is a bijection from $\mathcal{C}'$ to $\widehat{\mathcal{C}}'$, and therefore $\sigma' := \hbar^{-1}(\widehat{\sigma}')$, where $\hbar^{-1}$ denotes now the inverse of such restriction, is a path in the graph $(\tilde{m}_0 \cup \mathcal{C}' \cup \mathcal{C}'^\bullet, <_i)$ from some initial condition of m_0 to some element of ${}^\bullet(e \cup \underline{e})$. But this means that σ' followed by e, c is a path in $(\tilde{m}_0 \cup [c] \cup [c]^\bullet, <_i)$ from the initial conditions to c where at least k' occurrences of p happen, so the occurrence depth of c is at least k', and so $k \geq k'$.

  ii For an argument by contradiction, assume that $k > k'$, and that therefore some condition $c' < e$ is such that $\hbar(c') = p^{k'}$. By Lemma 1, we have $p^{k'} < \widehat{e}$; moreover, there is an immediate causality path in $\widehat{m}_0 \cup \widehat{\mathcal{C}} \cup \widehat{\mathcal{C}}^\bullet$ from $p^{k'}$ to $\widehat{e}$. Recall that, by def-

inition we have $\widehat{e} < p^{k'}$. So we have a causality cycle in $\widehat{\mathcal{C}}$, a contradiction to (61). □

5.5 ENCODING REACHABILITY INTO SAT

In the previous section, two results, Prop. 11 and Prop. 12, helped to characterize reachable markings of N and configurations of $\mathcal{Q}$ in terms of suitably constrained sets of mp-events. Here, we turn these results into SAT formulas whose solutions encode runs of N or mp-configurations of $\mathcal{Q}$.

Note that [KKKV06] discusses the corresponding problems for ordinary MPs and Ch. 4 for contextual unfoldings. Remarkably, both problems require to encode acyclicity for different purposes, which are united into a single acyclicity constraint in our case.

5.5.1 *Reachability via Arbitrary Runs*

Assume we want to verify the coverability of a set $m \subseteq P$ of places in N, and we have a marking-complete CMP $\mathcal{Q}$ of N. Proposition 11 gives necessary and sufficient conditions for m to be reachable; using this result, we define a propositional formula $\gamma_{\mathcal{Q}}^{\mathrm{mark},M}$ that is satisfiable iff m is *coverable*, although our formula can be very easily modified to check that m is reachable. The formula is over variables e for every mp-event $e \in \widehat{E}$ and $\mathsf{c}_{\mathrm{gen}}, \mathsf{c}_{\mathrm{con}}$ for every mp-condition $c \in \widehat{B}$. Actually, it will involve other variables, but this will be discussed later. The intuition behind the encoding is as follows. Any satisfying assignment of $\gamma_{\mathcal{Q}}^{\mathrm{mark},M}$ corresponds to a set X of mp-events that satisfy (60), (61), and a variant of (62). Specifically, if V is the set of variables in $\gamma_{\mathcal{Q}}^{\mathrm{mark},M}$ and $v\colon V \to \{0,1\}$ is a satisfying assignment, then the set $X := \{e \in \widehat{E}\colon v(\mathsf{e}) = 1\}$ satisfies all the three properties. The formula is a conjunction of four subformulas,

$$\gamma_{\mathcal{Q}}^{\mathrm{mark},M} := \gamma_{\mathcal{Q}}^{\mathrm{flow}} \wedge \gamma_{\mathcal{Q}}^{\mathrm{asym}} \wedge \gamma_{\mathcal{Q}}^{\mathrm{cov},M} \wedge \gamma_{\mathcal{Q}}^{\mathrm{aux}},$$

where the first three of these are defined as follows:

$$\gamma_{\mathcal{Q}}^{\mathrm{flow}} := \bigwedge_{e \in \widehat{E},\ c \in {}^\bullet e \cup \underline{e}} \mathsf{e} \to \mathsf{c}_{\mathrm{gen}}$$

$$\gamma_{\mathcal{Q}}^{\mathrm{asym}} := \mathsf{ACY}(\Delta_{\mathcal{Q}}) \wedge \bigwedge_{c \in \widehat{B}} \mathsf{AMO}(c^\bullet)$$

$$\gamma_{\mathcal{Q}}^{\mathrm{cov},M} := \bigwedge_{p \in m} \Big(\bigvee_{c \in \widehat{h}^{-1}(p)} \mathsf{c}_{\mathrm{cut}} \Big)$$

The first formula, $\gamma_{\mathcal{Q}}^{\mathrm{flow}}$, enforces X to satisfy (60). Indeed, (60) asks that every mp-event has its preset and context covered by either conditions generated by X or the initial marking. Now, $\gamma_{\mathcal{Q}}^{\mathrm{flow}}$ requests any such mp-condition c to be *generated* by X. Below we will see that $\mathsf{c}_{\mathrm{gen}}$ holds iff c is initially marked or at least one mp-event in the preset of c is in X.

The constraint $\gamma_{\mathcal{Q}}^{\mathrm{asym}}$ enforces (61) to hold for X. As in Ch. 4, we separate the handling of symmetric and purely asymmetric conflict loops. The second part of the constraint forbids symmetric loops, its implementation is the same as in § 4.3. The first part forbids satisfying assignments that include cycles of $\nearrow$. We come back to it later in this section.

Finally, $\gamma_{\mathcal{Q}}^{\mathrm{cov},M}$ constrains the satisfying assignment so that at least one mp-condition among those whose label is p, for every $p \in m$, is in the cut

of X. Observe that this encodes coverability of m, in contrast to (62), which asks for reachability of m. The variable c_{cut} holds iff c is generated by some mp-event in the satisfying assignment and it is not consumed. It is defined in $\gamma^{\text{aux}}_{\mathcal{Q}}$, which is the conjunction of the following auxiliary subformulas:

$$\bigwedge_{c\in\widehat{B}}(\mathsf{c}_{\text{con}} \leftrightarrow \bigvee_{e\in c^\bullet} \mathsf{e}) \qquad \bigwedge_{c\in\widehat{B}\setminus\widehat{m}_0}(\mathsf{c}_{\text{gen}} \leftrightarrow \bigvee_{e\in {}^\bullet c} \mathsf{e})$$

$$\bigwedge_{c\in\widehat{B}} \mathsf{c}_{\text{cut}} \leftrightarrow (\mathsf{c}_{\text{gen}} \wedge \neg\mathsf{c}_{\text{con}}) \qquad \bigwedge_{c\in\widehat{m}_0} \mathsf{c}_{\text{gen}}$$

For every mp-conditions c in $\mathcal{Q}$, the variable c_{gen} holds iff c is in $\widehat{m}_0 \cup X^\bullet$; similarly c_{con} holds if c is in ${}^\bullet X$.

Although this SAT encoding naturally bears certain resemblance with the one presented in Ch. 4, this one requires a more careful treatment. For instance, any condition of an unfolding has a preset with zero or one events. However, mp-conditions may have presets with more than one mp-event. This requires to use dedicated variables for mp-conditions, like c_{gen} or c_{con}, to encode a property such as (60), which could be seen, roughly speaking, as the equivalent for CMPs of causal closure for unfoldings.

We now discuss $\gamma^{\text{asym}}_{\mathcal{Q}}$ in detail. This constraint implements (61), it requires satisfying assignments of $\gamma^{\text{mark},M}_{\mathcal{Q}}$ to represent sets X such that $\nearrow_X$ is acyclic. As we did in Ch. 4, we handle separately symmetric and asymmetric conflict cycles. The $\mathsf{AMO}(\cdot)$ constraint forbids symmetric cycles. The constraint $\mathsf{ACY}(\cdot)$ forbids all cycles of asymmetric conflict which do not contain a cycle of symmetric conflict, as we explain now.

Given a digraph G, the constraint $\mathsf{ACY}(G)$ contains one Boolean variable for each vertex of G. An assignment of truth to these variables satisfies $\mathsf{ACY}(G)$ iff the variables assigned *true* correspond to a set of vertices that induces an acyclic subgraph of G.

In $\gamma^{\text{asym}}_{\mathcal{Q}}$, we apply $\mathsf{ACY}(\cdot)$ to the digraph $\Delta_{\mathcal{Q}}$, generated out of $\mathcal{Q}$. This digraph contains vertices for each mp-event and mp-condition of $\mathcal{Q}$. The intuition behind is that any set $Y \subseteq \widehat{E}$ of mp-events in $\mathcal{Q}$ is $\nearrow$-acyclic iff the set of vertices in $\Delta_{\mathcal{Q}}$ associated to $Y \cup {}^\bullet Y \cup Y^\bullet$ induces an acyclic subgraph. This is quite similar to what we did in Ch. 4, were we also needed to encode acyclicity of the asymmetric conflict relation. There, we had digraph whose nodes were *only* the events of the prefix and whose edges were, roughly speaking, the transitive reduction of causality on events. That graph was of linear size w.r.t. the prefix. In contrast, $\Delta_{\mathcal{Q}}$ needs to include variables for mp-conditions in order to achieve linear size w.r.t. to $\mathcal{Q}$.

We do not formally define $\Delta_{\mathcal{Q}}$, but a formal definition is easy to derive from the example in Fig. 27, on which we now elaborate. In Fig. 27 (a) we see a CMP, say $\mathcal{Q}$; in (b) the associated digraph $\Delta_{\mathcal{Q}}$. The digraph contains one vertex for every mp-event of $\mathcal{Q}$, and between one and two vertices for every mp-condition. Two vertices are necessary for ensuring the right order of occurrence for mp-events that have read arcs, as it will be clear shortly. Recall the definition of $\nearrow$, in p. 18: mp-events e, e' satisfy $e \nearrow e'$ iff

$e^\bullet \cap ({}^\bullet e' \cup \underline{e}') \neq \emptyset$, or
$\underline{e} \cap {}^\bullet e' \neq \emptyset$, or
$e \# e'$.

So given an mp-condition such as c_6 in Fig. 27 (a), the digraph needs to enforce that e_2 fires before, e.g., e_4 and e_6, and that e_6 fires before, e.g., e_5. We do this by introducing in $\Delta_{\mathcal{Q}}$ two vertices $\mathsf{c}_{6\mathsf{b}}$ and $\mathsf{c}_{6\mathsf{a}}$ ('b' for *before* and 'a' for *after*), and edges as follows: for any mp-event producing c_6, an edge

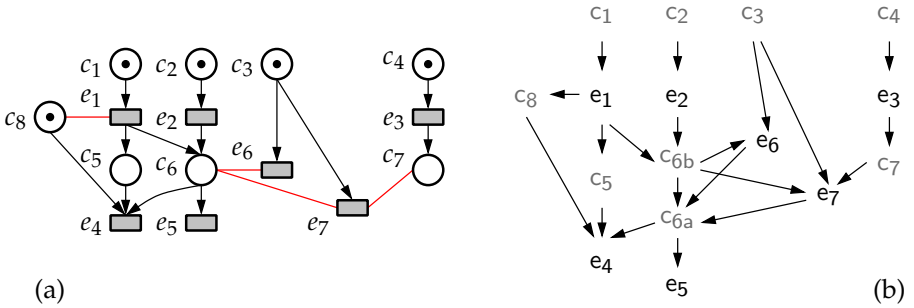


Figure 27: (a) A CMP $\mathcal{Q}$ together with (b) its associated digraph $\Delta_{\mathcal{Q}}$.

from it to c_{6b}; for any mp-event consuming c_6, an edge from c_{6a} to the mp-event; read arcs from c_6 are are translated as pairs of edges, leaving c_{6b} and coming back to c_{6a}; one edge is put from c_{6b} to c_{6a}. This construction allows any mp-event to optionally read c_6 without imposing an order among the possibly multiple readers, but enforcing that such readers happen after any mp-event in the preset of c_6 and before any event in the postset of c_6.

Some mp-conditions, such as c_8, are only read and consumed, but not produced. For those, we only need one vertex in $\Delta_{\mathcal{Q}}$, and read arcs are translated as edges from the readers to the mp-condition. A similar situation happens in c_7, where there is no consumer. In this case the read arc is translated by an edge in the opposite direction, see Fig. 27 (b).

Several optimizations are obvious to this encoding, such as removing nodes associated to mp-conditions with one incoming edge, but we do not discuss them here. Independently of whether they are used, the size of $\Delta_{\mathcal{Q}}$ is linear on $\mathcal{Q}$.

The constraint $\mathsf{ACY}(\cdot)$ is implemented as in Ch. 4. It encodes the idea that if two nodes of the graph are *selected*, their ranks must be sorted. We note that mp-events are selected if the associated variable is true, and mp-conditions are selected if $\mathsf{c}_{\mathrm{gen}}$ is true.

5.5.2 *Reachability via Mp-Configurations*

The encoding in the previous section does not require the selected set of events to be an mp-configuration; it only demands it to represent a repetition-free firing sequence of the CMP.

In this section we produce a propositional formula whose models encode all and only the mp-configurations of $\mathcal{Q}$ that cover a given marking of N. The encoding is based on Prop. 12 and reuses all the constraints that compose $\gamma_{\mathcal{Q}}^{\mathrm{mark},M}$, presented in the previous section.

Specifically, given a set of places $M \subseteq P$, we define a propositional formula $\gamma_{\mathcal{Q}}^{\mathrm{con},M}$ whose satisfying assignments encode sets X of mp-events that satisfy (60), (61), (63), (64), and a variant of (62). The formula is a conjunction of two constraints:

$$\gamma_{\mathcal{Q}}^{\mathrm{con},M} := \gamma_{\mathcal{Q}}^{\mathrm{mark},M} \wedge \gamma_{\mathcal{Q}}^{\text{no-gap}}.$$

The subformula $\gamma_{\mathcal{Q}}^{\mathrm{mark},M}$ requests M to be coverable by means of a repetition-free run, and enforces (60), (61), and (62), as we have already seen; $\gamma_{\mathcal{Q}}^{\text{no-gap}}$ enforces (63) in a fairly obvious way:

$$\gamma_{\mathcal{Q}}^{\text{no-gap}} := \bigwedge_{c:=p^k \in \widehat{B},\ c':=p^{k-1} \in \widehat{B},\ k \geq 1} \mathsf{c}_{\mathrm{gen}} \rightarrow \mathsf{c}'_{\mathrm{gen}}.$$

We now discuss how (64) is encoded. This condition requests that X contains a path of immediate causality that visits all mp-conditions $p^1, p^2, \ldots, p^n$ labelled with the same place p, by increasing order of depth. As in [KKKV06], we enforce this property augmenting the digraph $\Delta_{\mathcal{Q}}$ with additional edges. We first observe that asking for the presence of a path from p^k to p^{k+1} is equivalent to forbidding a path from p^{k+1} to p^k:

Lemma 7. *If N is safe, for any set $\widehat{\mathcal{C}}$ of mp-events satisfying (60), (61), and (63), the following statement is equivalent to (64): for $k \geq 1$,*

- *the digraph $(\widehat{m}_0 \cup \widehat{\mathcal{C}} \cup \widehat{\mathcal{C}}^{\bullet}, <_i)$ has* no *path from p^{k+1} to p^k.* (65)

Proof. First, remark the difference between both properties: (64) requests a path from p^k to p^{k+1}, and (65) forbids a path from p^{k+1} to p^k. We denote in the sequel the aforementioned digraph by G.

Trivially if $\widehat{\mathcal{C}}$ satisfies (61), (63), and (64), it also verifies (65). Otherwise, there would exist some p^k and p^{k+1} such that the $\widehat{\mathcal{C}}$ contains a cycle of causality $p^k < p^{k+1} < p^k$, a contradiction to (61).

For the other direction, assume (65) holds but (64) is violated. We have mp-conditions p^k and p^{k+1} in $\widehat{m}_0 \cup \widehat{\mathcal{C}}$ that are mutually unreachable in G. We show this is a contradiction to N being safe. Let $\widehat{e}_1$ be any mp-event in ${}^{\bullet}p^{k+1} \cap \widehat{\mathcal{C}}$, which exists because $k+1 \geq 2$. Consider the set

$$\widehat{\mathcal{C}}' := \widehat{\mathcal{C}} \setminus \{\widehat{e} \in \widehat{\mathcal{C}} \colon p^{k+1} \lhd^* \widehat{e}\},$$

where $\lhd$ denotes the relation $<_i \cap (\widehat{\mathcal{C}} \cup \widehat{\mathcal{C}}^{\bullet})^2$. That is, we remove from $\widehat{\mathcal{C}}$ all mp-events that can be reached from p^k in G, observe that $\lhd$ is the edge relation of G. Obviously $\widehat{\mathcal{C}}'$ contains $\widehat{e}_1$, otherwise $\widehat{\mathcal{C}}$ would violate (61). Also, $\widehat{m}_0 \cup \widehat{\mathcal{C}}'^{\bullet}$ contains p^k as a consequence of p^k not being reachable from p^{k+1}. It is also immediate to show that $\widehat{\mathcal{C}}'$ satisfies (60) and (61). As a result, any linear extension of $\nearrow_{\widehat{\mathcal{C}}'}$ is a run of $\mathcal{Q}$ that marks p^{k+1}. Now consider the set

$$\widehat{\mathcal{C}}'' := \widehat{\mathcal{C}}' \setminus \{\widehat{e} \in \widehat{\mathcal{C}}' \colon p^k \lhd^* \widehat{e}\},$$

i.e., we remove from $\widehat{\mathcal{C}}'$ all mp-events reachable from p^k in G. Similarly, $\widehat{e}_1$ is in $\widehat{\mathcal{C}}''$ because $\widehat{e}_1$ is unreachable from p^k, and $\widehat{m}_0 \cup \widehat{\mathcal{C}}''^{\bullet}$ contains p^k, due to $\widehat{\mathcal{C}}'$ satisfying (61). So $p^k, p^{k+1} \in \mathit{cut}(\widehat{\mathcal{C}}'')$. As before, $\widehat{\mathcal{C}}''$ satisfies (60) and (61), so there is a run of $\mathcal{Q}$ that marks p^k and p^{k+1}, a contradiction to N being safe. □

With this result, our task is simple: given two mp-conditions $\widehat{c}, \widehat{c}'$, forbidding a path from $\widehat{c}$ to $\widehat{c}'$ amounts to including in $\Delta_{\mathcal{Q}}$ an edge from $\widehat{c}$ to $\widehat{c}'$; if such a path is selected, this edge will close a cycle that $\gamma_{\mathcal{Q}}^{\text{asym}}$ will reject.

So we enforce (64) by including one edge from p^k to p^{k+1}, for every $p^k, p^{k+1} \in \widehat{B}$, in the digraph $\Delta_{\mathcal{Q}}$.

5.6 CONSTRUCTING COMPLETE CMPS

In this section, we discuss various algorithmic aspects of CMPs, in particular, how to construct a marking-complete CMP from a given *safe* c-net N.

Recall that a CMP is marking-complete if every reachable marking m of N is the $\widehat{h}$-image of the cut of some cutoff-free mp-configuration. We wish to construct such a CMP in order to analyze properties of N such as reachability or deadlock.

It follows from Def. 16 that one can achieve this goal by (*i*) constructing a marking-complete unfolding prefix $\mathcal{P}$ and (*ii*) applying the construction from Def. 16 to $\mathcal{P}$. This does not yield any gain for practical verification, as

the unfolding prefix can be much larger than the CMP. It will be, however, the method we employ in § 7.6 to construct marking-complete CMPs in order to compare their sizes with unfoldings.

Another option is to construct a CMP directly from the c-net N. A similar approach for nets without read arcs was presented in [KM11; KM13]. No such implementation currently exists for CMPs; in the following we describe some key elements that are required for extending [KM13] to CMPs.

A procedure for directly constructing the CMP would start with a CMP containing only the mp-conditions that represent the initial marking of N and extend it one mp-event at a time. To know whether the current CMP $\mathcal{Q}$ can be extended by an mp-event $\widehat{e}$, one has to identify an mp-configuration $\widehat{\mathcal{C}}$ of $\mathcal{Q}$ and check (i) whether $\widehat{\mathcal{C}} \cup \{\widehat{e}\}$ is an mp-configuration of $\mathcal{M}_N$ and (ii) whether $\widehat{e}$ constitutes a cutoff.

Problem (i) is already solved, we can use the SAT encoding presented in § 5.5.2 to find mp-configurations $\widehat{\mathcal{C}}$ that enable a new event $\widehat{e}$. For (ii), observe that an mp-configuration $\widehat{H}$ corresponds to some history H of an event e of $\mathcal{U}_N$ with $\hbar(e) = \widehat{e}$ iff $\widehat{e}$ is the maximal element of the relation $\nearrow_{\widehat{H}}$. The problem then corresponds to asking whether for all such $\widehat{H}$ there exists another mp-configuration $\widehat{\mathcal{C}}$ such that $\mathit{mark}(\widehat{\mathcal{C}}) = \mathit{mark}(\widehat{H})$ and $\widehat{\mathcal{C}} \prec \widehat{H}$. For $\widehat{\mathcal{C}}, \widehat{H}$ in $\mathcal{Q}$, this problem can be encoded in 2QBF, which is more complicated than SAT but less so than QBF in general, and for which specialised solutions exist [RTM04].

As $\mathcal{Q}$ grows, the number of possible candidates for $\widehat{H}$ may increase. As a result, in general $\widehat{e}$ cannot be designated a cutoff until the construction has been terminated. So $\widehat{e}$ may become a cutoff as the construction progresses, and the algorithm has to re-checked periodically.

To summarize, the basic structure of the algorithm from [KM13] would remain unchanged, however one needs to use the encoding of § 5.5.2 to find mp-configurations rather than the non-contextual one explained in [KM13].

5.7 CONCLUSIONS

We have developed a new condensed representation of the state space of a contextual Petri net, called contextual merged processes. This representation combines the advantages of merged processes and contextual unfoldings, and copes with several important sources of state space explosion: concurrency, sequences of choices, and concurrent read accesses to shared resources. Experimental results in Ch. 7 will demonstrate that this representation is significantly more compact than either merged processes or contextual unfoldings.

We also proved a number of results which lay the foundation for model checking of reachability-like properties of safe c-nets based on CMPs. In particular, given a CMP, they allow one to reduce (in polynomial time) such a model checking problem to SAT. Furthermore, since the algorithm for direct construction of merged processes of safe Petri nets proposed in [KM13] is based on model checking, it can be transferred to the contextual case, which would complete the verification flow based on CMPs.

Future work includes implementing the proposed model checking algorithm and porting the algorithm for direct construction of MPs proposed in [KM13] to the contextual case. While the high-level structure of the latter algorithm remains the same, moving from Petri nets to c-nets entails several low-level changes in the nets representation, which pervade the whole code; thus, this porting requires significant implementation effort.

6 DIAGNOSIS

Like Ch. 4, this chapter focuses on analysis methods for concurrent systems, in this case, fault diagnosis. We present a methodology for the diagnosis of faults in concurrent, partially-observable systems.

The chapter extends and generalizes the unfolding-based diagnosis approaches by Benveniste et al. [BFHJ03] as well as Esparza and Kern [EK12]. The latter work focused only on the use of sequential observations. We remove this assumption and employ partially-ordered observations. Diagnosis infers information from the system execution in order to detect faults. A second contribution is extending the method to exploit the additional information provided by the assumption of fair behavior. Theoretical foundations and a decision procedure are presented. We define three unfolding prefixes such that the diagnosis problem reduces to the existence of certain configurations in them. Next we show that the existence of these configurations can easily be encoded in SAT.

The results contained this chapter hold for *ordinary Petri nets*, not contextual nets. We find this work already interesting for ordinary nets. Furthermore, the theory of the *reveals* relation, employed in characterizing weak diagnosis, is not yet ready for contextual nets.

This chapter presents results of [HRS13].

6.1 INTRODUCTION

Diagnosis under partial observation is a classical problem in automatic control in general, and has received considerable attention in discrete event system (DES) theory, among other fields.

In formal verification, the objective is removing errors from a system under development. In contrast, fault diagnosis only tries to detect and report faults during execution. For certain systems, such as large telecommunication networks, with links and routers that can fail at any moment, it is just not possible to avoid faults during operation. Diagnosis plays here, as we explained in Ch. 1, a complementary role to verification.

Diagnosis thus monitor the system and tries to detect faults. In certain cases, however, the amount of information available to the monitor is not enough to determine in which stat the system currently is. For instance, on a large network, distant nodes often cannot send full log information due to limited available bandwidth. Also, system-on-chip devices are often constructed with only very few pins to transmit debugging information. So limited visibility of what the system is doing happens often in practice.

Fault diagnosis observes the *visible* part of the system execution and infers which states the system could currently be in, and more specifically, whether a fault has happened. Like verification, it assumes that a model of the system is available.

The classical setting [SSL+95; CL08] assumes that the observed system is an automaton with a transition set T. The automaton is equipped with a labeling $\lambda\colon T \to \mathbb{X} \cup \{\varepsilon\}$ which models the information observed by each execution: each transition is labelled by either some *alarm* or *observable label* from the set $\mathbb{X}$, or by the empty string ε. The former transitions are called

observable transitions, the latter *unobservable*, and the mapping is not necessarily assumed to be injective. When the system executes a sequence $w \in T^*$ of transitions, the monitor only sees $\lambda(w)$. Some particular unobservable transition ϕ is a *fault*. Diagnosis is then the task of deciding, given an observation $\widehat{w} \in \mathbb{X}^*$, whether or not all possible behaviors that *explain* the observation, i.e., in $\lambda^{-1}(\widehat{w})$, contain an occurrence of ϕ.[1] Diagnosis, in the strict sense, reasons exclusively about the *past* of the system.

Classical diagnosis thus assumes that observations are streams of alarms. However, different types of diagnosis emerge from the distinct architectures of both the system and its supervision, see [BFHJ03].

In centralized, non-sequential, or *asynchronous diagnosis*, there are several sensors, each of which observes (a fragment of) a concurrent or distributed system. Each sensor produces a sequence of observations, as above. However, the different streams from the multiple sensors reach the (centralizing) supervisor asynchronously; no assumption is made about the communication architecture or speed. One assumes that the architecture respects causality (if occurrence of a causally precedes that of a', the supervisor sees a before a'), and that the ordering of observations from the *same* sensor is respected. By contrast, for any two alarms a and a' recorded by *distinct* sensors, any interleaving of a and a' must be considered. *Decentralized diagnosis* involves several supervisors that cooperate to reach a global verdict on whether or not a fault has occurred. The supervisors emit local verdicts (e.g. yes/no) that are synthesized into a global one. In *distributed diagnosis* [FBHJ05; BHK06], several supervisors compute *explanations* in a distributed unfolding procedure.

In this chapter we focus on asynchronous diagnosis of concurrent systems. These are difficult to supervise using the classical approach because SSE appears in multiple places as a consequence of using automata models. First, modelling a concurrent system by an automaton provokes SSE due to concurrency. But also, explanations are not in general streams of alarms, but rather sets of streams of alarms (one per sensor), or more in general *labelled partial orders*. Partially ordered observations must be first interleaved before using them under the classical approach, yielding again a combinatorial explosion.

Partially ordered observations and Petri net models of the system are employed in this work. The use of models that reflect the local and distributed nature of the observed system, such as Petri nets, is not only helpful in terms of computational efficiency, but also conceptually [FB07]. In [BFHJ03] diagnosis has been extended to asynchronous models using Petri net unfoldings and partially ordered observations. Roughly speaking, from a given observation, a prefix of the unfolding is identified that contains all explanations. The diagnosis method in [BFHJ03] assumes, however, that the model of the system has no unobservable loops, i.e., every infinite execution produces infinitely many alarms. This assumption is removed in [EK12], but a new one, not present in [BFHJ03], is introduced. The method from [EK12] assumes that observations are sequences of alarms instead of labelled partial orders.

Here, we extend [EK12] to remove the assumption of sequential observations. Our approach extends that of [BFHJ03; EK12] or [HF13] and is also based on unfoldings. As second aspect our extension, not considered in any of the aforementioned works, is the introduction of weak fairness.

1 The problem is also stated assuming that the automaton has a *set* of faults, and the goal is to see if any explanation at least contains one, but this does not alter the problem much.

In concurrent systems, certain events and properties can be implied, or *revealed* by others if some notion of fairness is assumed. Naturally, every event that causally precedes another, is revealed by the latter; but fairness allows to infer more. For instance, consider the occurrence net shown in Fig. 29. Under the assumption that only the $\subseteq$-maximal configurations correspond to the runs of the system that we may observe, one may clearly conclude that any run containing a also contains c, as a has disabled b and c has to fire, by assumption.

The additional assumption that we need is that of *weakly fair behavior*: on weakly fair runs, a transition t that becomes enabled at some point cannot stay enabled forever; eventually, either t or another conflicting transition t' must fire. Call t', or t itself, a *spoiler* of t; for the run to be weakly fair, some spoiler of t must fire. Weak fairness is also often called progress. The notion of spoiler will play an important role in this chapter.

Diagnosis under the assumption of weakly fair behaviour is called *weak diagnosis*. We say that an observation $\widehat{w}$ weakly diagnoses fault ϕ iff all weakly fair runs that explain $\widehat{w}$ contain ϕ. Thus we aim at extending the frameworks of [BFHJ03; EK12] to weak diagnosis, and providing a decision procedure for the weak diagnosis problem.

Both [BFHJ03] and [EK12] use Petri net unfoldings under certain restrictions: [BFHJ03] accepts partial-order observations, but refuses models with unobservable loops; [EK12] accepts the latter, thanks to dedicated *cutoff criteria*, but refuses the former. Our work uses both features, additionally accounting for weak fairness in the diagnosis procedure. We generalize the cutoff criteria of [EK12] to the partially ordered observations from [BFHJ03], and extend the generalization to include fairness.

The chapter is organized as follows. In § 6.2 we recall basic notions and explain how our notation for contextual nets specializes for the case of ordinary Petri nets. In § 6.3, we present the diagnosis framework and formally define the problem that we solve. In § 6.4, theoretical foundations are presented, supporting the definition of a decision procedure, in § 6.5, for the weak diagnosis problem. This procedure is based on SAT solving. In § 6.6, we conclude and discuss future work.

6.2 BASIC NOTIONS

In this section, we establish notations and recall how definitions introduced in Ch. 2 specialize for *ordinary* Petri nets. We also discuss the notions of weak fairness and labelled partial orders, and prove some statements about them that will be used in the rest of the chapter.

PETRI NETS A *Petri net* is a c-net with an empty context relation. In this chapter we will denote a Petri net by a tuple $N := \langle P, T, F, m_0 \rangle$, where we have indeed omitted the empty context relation. The immediate causality relation $<_i$ for Petri nets, defined for c-nets in (2) and (3), on p. 17, coincides now with the flow relation F, and the causality relation is, as before, the transitive closure of $<_i$. The asymmetric conflict relation, similarly, reduces to the relation $\# \cup (<_i^2 \cap (T \times T))$, as Rmk. 2 states. At any rate, we will not use asymmetric conflict in this chapter, as symmetric conflict suffices to express the conflicts that may relate events.

The axioms for occurrence nets, (13) to (16) on p. 22, are defined as before, with the particularity that (15) reduces to asking that every event e in the net is such that $[e]$ is conflict-free, i.e., it contains no two events in symmetric

conflict. Let $O := \langle B, E, G, \tilde{m}_0 \rangle$ be an occurrence net. The configurations of O are all causally-closed, conflict-free set of events. Indeed, the axiom (18), which forbids cycles of asymmetric conflict for configurations of a contextual ON, reduces to forbidding the presence of conflicting events. By $conf(O)$ we denote the set of all such configurations.

We define specific notation for this chapter. For an event $e \in E$ and configuration $\mathcal{C}$ of O, we say that $\mathcal{C}$ *enables* e, written $\mathcal{C} \overset{e}{\rightsquigarrow}$, iff $e \notin \mathcal{C}$ and $(\mathcal{C} \cup \{e\}) \in conf(O)$.

Two nodes x, y of O are *concurrent*, written $x \parallel y$, if neither $x \leq y$, nor $y \leq x$, nor $x \# y$ holds. Observe that we reuse here the symbol $\parallel$ from Ch. 3 to denote a different relation, here between nodes of O, while in Ch. 3 it denoted the concurrency relation between enriched conditions of an enriched prefix.

WEAK FAIRNESS Runs represent a sequential view of the executions of a net, whereas configurations represent the concurrent point of view. We give and relate the definitions of weak fairness for both.

Let $N = \langle P, T, F, m_0 \rangle$ be a finite Petri net, and $\mathcal{U}_N = \langle \langle \tilde{B}, \tilde{E}, \tilde{G}, \tilde{m}_0 \rangle, h \rangle$ its unfolding. A configuration of $\mathcal{U}_N$ is *weakly fair* if it is $\subseteq$-maximal in $conf(\mathcal{U}_N)$. Recall that the evolution order $\sqsubseteq$ between configurations of contextual unfoldings reduces to set inclusion for ordinary unfoldings. We denote by $\Omega(\mathcal{U}_N)$ the set of weakly fair configurations, or Ω if no confusion can arise.

Lemma 8. *A configuration ω is weakly fair iff it does not enable any event.*

Proof. If $\omega \in \Omega$ enables e, then $\omega \cup \{e\}$ is a configuration, and ω is not maximal, a contradiction. If ω is not weakly fair, then there exists a configuration $\mathcal{C}$ that is a proper superset of ω. Pick some $<$-minimal event e from $\mathcal{C} \setminus \omega$. By assumption, all causal predecessors of e are in ω, so all conditions in ${}^\bullet e$ are either initial or receive a token from some event in ω. Since $\mathcal{C}$ is a configuration, it is conflict-free. So no event in ω will remove a token from ${}^\bullet e$, and ω enables e. □

Here we assume, for the sake of simplicity, that all weakly fair configurations of $\mathcal{U}_N$ are infinite. This entails no loss of generality, finite weakly fair configurations correspond to deadlocks that can be detected and processed separately — for instance, by adding a looping transition.

A *spoiler* of a transition t of N is any $t' \in T$ such that ${}^\bullet t \cap {}^\bullet t' \neq \emptyset$ (including t itself). We write $spoilers(t)$ for the set of such transitions. Following Vogler [Vog95], who adapts the concept of *weakly fair termination* [Fra86] to a Petri net setting, we say that an infinite run $\sigma = t_1 t_2 \ldots \in T^\omega$ of N is *weakly fair* if its marking sequence $m_0, m_1, \ldots$ satisfies that for all $i \in \mathbb{N}$ and all $t \in T$, if m_i enables t, then there exists $j > i$ such that $t_j \in spoilers(t)$. In other words, any t enabled at some point along σ either fires eventually, or some other transition that consumes from its preset is fired. For runs of an occurrence net O, such as $\mathcal{U}_N$, we can make the following, stronger statement.

Lemma 9. *Let σ be a weakly fair run of O and $m_0, m_1 \ldots$ its marking sequence. For all $i \in \mathbb{N}$ and all $e \in E$, if m_i enables e, then*

$$\exists k > i \; \forall j \geq k \colon m_j \text{ does not enable } e.$$

Proof. The statement follows from the definition of weakly fair runs and the fact that $<$ is acyclic for O; once a token from ${}^\bullet e$ is consumed, it cannot be replaced, and e remains disabled forever. □

Finally, we observe that weakly fair runs and weakly fair configurations are related in the following way:

Lemma 10. *Every weakly fair run of $\mathcal{U}_N$ is an interleaving of some $\omega \in \Omega$. Conversely, all interleavings of every $\omega \in \Omega$ are weakly fair.*

Proof. Let $\sigma = e_1 e_2 \ldots$ be an (infinite) weakly fair run and $\omega := \{ e_i : i \geq 1 \}$. Since no e_i can fire without its causal predecessors putting tokens into its preset, ω is causally closed. Due to the acyclic structure of $\mathcal{U}_N$, no condition can receive a token twice, no event will be repeated in σ, so no two events can consume from the same place, therefore ω is conflict-free and hence a configuration. Now suppose ω is not weakly fair, then by Lemma 8 it enables some event e. Arguing like in the proof of Lemma 8, we can conclude that all conditions in ${}^\bullet e$ are either initial or will receive tokens from events in ω, and that no event in ω consumes from ${}^\bullet e$. Thus, some marking m_i in the marking sequence for σ enables e, and then e is never disabled, which contradicts the weak fairness property for σ.

For the converse, let σ be an interleaving of a weakly fair configuration ω. Suppose that σ is not weakly fair; then some event e eventually becomes enabled during σ but neither e nor any conflicting event is in ω. Now, e can only become enabled if all its causal predecessors are in ω and put tokens into its preset, so $\omega \cup \{e\}$ is also causally closed. Thus, $\omega \cup \{e\}$ is a configuration, which contradicts maximality of ω. □

LABELLED PARTIAL ORDERS An *alphabet* is a finite set $\mathbb{X}$, whose elements are called *letters*. A *labelled partial order* (LPO) over $\mathbb{X}$ is a tuple $\alpha = \langle S, < , \lambda \rangle$ where $< \subseteq S \times S$ is an irreflexive and transitive (hence antisymmetric) relation on S, and $\lambda : S \to \mathbb{X}$ a labelling map. The size $|\alpha|$ of α is $|S|$. Let $\alpha' = \langle S', <', \lambda' \rangle$ be an LPO over $\mathbb{X}$. A *homomorphism* from α to α' is a function $h : S \to S'$ verifying

- $\lambda(a) = \lambda'(h(a))$, and (66)
- $a < b$ implies $h(a) <' h(b)$ for all $a, b \in S$. (67)

An *isomorphism* between α and α' is a bijective homomorphism h from α to α' where h^{-1} is a homomorphism from α' to α. We say that α is *compatible* with α' if there exists a bijective function $f : S \to S'$ such that

- $\lambda(a) = \lambda'(f(a))$, and (68)
- $a < b$ implies $\neg(f(b) <' f(a))$ for all $a, b \in S$. (69)

Observe that (67) and (69) are not equivalent, and f must be bijective. Denote by $\mathit{compat}(\alpha)$ the set of LPOs compatible with α.

Lemma 11. *Given LPOs α, α', if there is a bijective homomorphism h from α' to α, then $\mathit{compat}(\alpha) \subseteq \mathit{compat}(\alpha')$.*

Proof. Let $A := \langle S_A, <_A, \lambda_A \rangle \in \mathit{compat}(\alpha)$, and let $f_1 : S_A \to S$ be the associated bijection. Let $f_2 := h^{-1}$ be the (bijective) inverse of h. We show that $f := f_2 \circ f_1$ satisfies (68) and (69) from A to α'. For $a, b \in S_A$, we prove that:

- f is bijective, as it is the composition of two bijections.
- $\lambda_A(a) = \lambda'(f(a))$. This is because f_1 (by definition) and f_2 (by construction from h) preserve labels.
- $a <_A b$ implies $\neg(f(b) <' f(a))$. Assume that $a <_A b$ and $f(b) <' f(a)$. Let $a_\alpha := f_1(a)$, and $b_\alpha := f_1(b)$. We know that $\neg(b_\alpha < a_\alpha)$ holds by definition of f_1. But by definition of h also $b_\alpha = h(f(b)) < h(f(a)) = a_\alpha$ holds, a contradiction. □

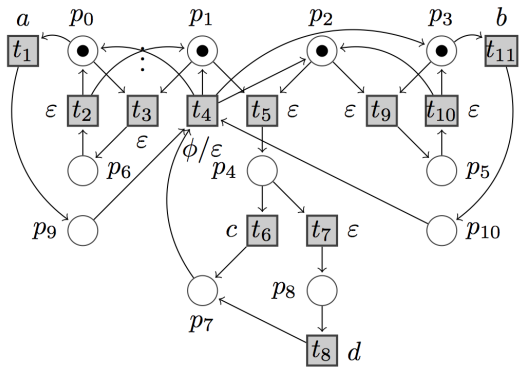


Figure 28: A safe Petri net with partial observation. The inscription of a transition indicates its name; the label next to it is either a Latin letter or empty (ε), in which case the transition is invisible. Transition t_4 is the invisible fault transition, which is called ϕ in the text and whose label is ε.

Lemma 12. *LPOs α and α' are isomorphic iff $compat(\alpha) = compat(\alpha')$.*

Proof sketch. One direction is trivial, the other is reasoning by cases on the possible orderings in the LPOs, after establishing that S and S' have the same size. □

6.3 *reveals* AND DIAGNOSIS

All diagnosis strives to detect *'hidden'* events, but we aim at diagnosing also *latent* but *inevitable* events, possibly in the future of the system evolution. That is, we wish to diagnose exactly whether every *weakly fair run* that is compatible with the observations so far, contains a fault occurrence. By the above, we thus need to consider all weakly fair configurations in Ω that contain an explanation of the current observation as a prefix. Let us formalize these notions.

6.3.1 *Reveals Relations*

In occurrence nets, given two events e, e', we say [Haa07; BCH11; HKS13]) that e *reveals* e', written $e \triangleright e'$, iff

$$e \in \omega \Rightarrow e' \in \omega \text{ for all } \omega \in \Omega,$$

that is, the occurrence of e entails that e' *inevitably* will occur, or has already occurred. For instance, in Fig. 29, which shows an unfolding prefix of Fig. 28, we have $e_5 \triangleright e_1$, and $e_4 \triangleright e_6$, and $e_3 \triangleright e_2$. After the occurrence of e_5, the occurrence of e_3 has become impossible; in a weakly fair execution, e_1 must thus necessarily occur. Similarly, when e_4 occurs, e_6 must already have occurred previously ($\triangleright^{-1}$ includes and extends *causal precedence*). Also, all weakly fair configurations containing e_3 must contain e_2. Note that $\triangleright$ is *not* a causal or temporal relation. Any revealed event can be, with respect to the event that reveals it, in the past (as for $e_4 \triangleright e_6$), be concurrent (as in $e_5 \triangleright e_1$), or in the future (as in $e_3 \triangleright e_2$).

It is shown in [Haa10] that one can characterize $\triangleright$ by:

$$x \triangleright y \quad \text{iff} \quad \#[y] \subseteq \#[x],$$

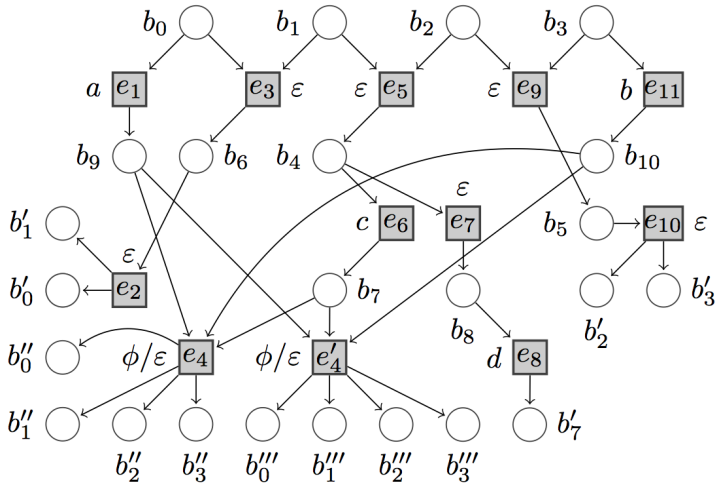


Figure 29: A prefix of the unfolding for the net in Fig. 28. Events are named according to the corresponding transition (via the mapping h): e_i, e'_i, e''_i etc. for the occurrences of t_i. Observation labels are as in Fig. 28.

where $\#[u] := \{z \in E : z \# u\}$. That is, $x \triangleright y$ iff every event that prevents y, and these are contained in $\#[y]$, is also prevented by x, i.e., lies in $\#[x]$. This binary relation $\triangleright$ can be computed, for 1-safe Petri nets, on a finite prefix whose height is bounded [HKS13].

The *binary* reveals relation helps in detecting invisible events; however, for diagnosis purposes, it is not strong enough. We shall need more general relation that relates sets to sets, namely the *extended-reveals* relation $\twoheadrightarrow$ introduced by Balaguer et al. [BCH11]. Assume that in Fig. 29, a and b are observable labels, and that we actually do observe their occurrence. From inspection of Fig. 29, we know that (*i*) e_5 is bound to occur, and (*ii*) one of the two instances of ϕ, either e_4 or e'_4, are inevitable. However, the binary relation $\triangleright$ does not permit to deduce (*i*) and (*ii*): while the *conjunction* of e_1 and e_{11} makes e_5 inevitable, we have neither $e_1 \triangleright e_5$ nor $e_{11} \triangleright e_5$ individually; also, the *disjunction* of e_4 or e'_4 is certain once e_5 is assured, but neither e_4 or e'_4 are revealed individually. To account for such situations, following [BCH11] we say, for sets of events A, B, that A *extended-reveals* B, written $A \twoheadrightarrow B$, iff every weakly fair configuration that contains A also '*hits*' B, i.e.,

$$A \twoheadrightarrow B \quad \text{iff} \quad \forall \omega \in \Omega : (A \subseteq \omega \Rightarrow B \cap \omega \neq \emptyset).$$

6.3.2 *Diagnosis from Partial Observation*

We fix now, for the rest of the chapter, the following framework. Let $N := \langle P, T, F, m_0 \rangle$ be a finite, safe Petri net satisfying the general assumptions on p. 18, and let $\mathcal{U}_N := \langle \langle \tilde{B}, \tilde{E}, \tilde{G}, \tilde{m}_0 \rangle, h \rangle$ be its unfolding. We denote by Ω the set of weakly fair configurations of $\mathcal{U}_N$ and assume that all of them are infinite. Let ε denote a unique '*empty*' symbol, and let $\mathbb{X}$ denote a non-empty *observation* (or *alarm*) *alphabet* where $\varepsilon \notin \mathbb{X}$. Let $\lambda : T \to \mathbb{X} \cup \{\varepsilon\}$ be the mapping associating transitions of N with observations or ε, and $\phi \in T$ the unique *fault* transition. We let

$$T_{obs} := T \setminus T_{ubs} \qquad T_{ubs} := \lambda^{-1}(\varepsilon)$$

be the sets of *observable* and *unobservable transitions* of N, and

$$E_{obs} := h^{-1}(T_{obs}) \qquad E_{ubs} := h^{-1}(T_{ubs})$$

be the sets of *observable* and *unobservable events* of $\mathcal{U}_N$. Naturally, we assume that ϕ is unobservable, i.e., $\phi \in T_{ubs}$, and define the set of *fault events* as

$$E_\phi := h^{-1}(\phi).$$

Observe that $E_\phi \subseteq E_{ubs}$. We extend λ to $\tilde{E}$ and, by abuse of notation, define $\lambda(e) := \lambda(h(e))$ for all $e \in \tilde{E}$.

We now define the notion of *observation pattern*, or simply *observation*. Observations are LPOs over an observation alphabet. LPOs allow to capture linearly ordered observations, produced by a single sensor that observes the *interleaving* of the system; or sets of linear orders, produced by a number of sensors that locally observe each concurrent process of a distributed system; or yet others. This allows to work in an *asynchronous* setting as in [BFHJ03], but without the need to enumerate the *interleavings* of observation patterns, as opposed to the approach of [EK12]. For the rest of the chapter, fix a *finite* observation pattern $\alpha := \langle S_\alpha, <_\alpha, \lambda_\alpha \rangle$ over the observation alphabet $\mathbb{X}$.

Given N and α, where α is the observation of some execution of N, our goal is to determine whether that execution contains a fault, assuming that runs of N are weakly fair. So we need to consider all those weakly fair configurations of $\mathcal{U}_N$ that are *compatible* with α. We formalize this in two steps. First, we associate every configuration $\mathcal{C}$ with an LPO $lpo(\mathcal{C}) := \langle S, <', \lambda' \rangle$, where $S := \mathcal{C} \cap E_{obs}$ are the observable events in $\mathcal{C}$, $<'$ is the restriction to S of the causal order $<$ on $\mathcal{U}_N$, and $\lambda' \colon S \to \mathbb{X}$ is the restriction of λ to S. Since $<'$ and λ' are restrictions of $<$ and λ, it is safe to confuse them here, and so we will.

Second, we define the *observations* of $\mathcal{C}$ as the set

$$obs(\mathcal{C}) := compat(lpo(\mathcal{C})),$$

i.e., the set of all (LPOs modelling) observations compatible to *the* LPO of $\mathcal{C}$. Conversely, we say that $\mathcal{C}$ *explains* observation α if $\alpha \in obs(\mathcal{C})$, and define

$$expl(\alpha) := \{\mathcal{C} \in conf(\mathcal{U}_N) \colon \alpha \in obs(\mathcal{C})\}.$$

As a consequence of Lemma 12, for configurations $\mathcal{C}$, $\mathcal{C}'$, we have $obs(\mathcal{C}) = obs(\mathcal{C}')$ iff $lpo(\mathcal{C})$ is isomorphic to $lpo(\mathcal{C}')$.

Definition 19. *An observation pattern* α weakly diagnoses ϕ *iff*

$$\text{for all } \mathcal{C} \in expl(\alpha),\ \mathcal{C} \twoheadrightarrow E_\phi. \tag{70}$$

Since weak diagnosis is the only form of diagnosis we consider, we henceforth simply speak of *diagnosis*.

In the context of Fig. 28, any observation containing a label c or d clearly diagnoses ϕ. This is because once any transition labelled by them (t_6 or t_8) has fired, p_7 is marked. By fairness, necessarily t_1 and t_{11} need to fire, marking p_9 and p_{10}, which together with p_7 enable t_4, the faulty transition.

What is more, and may serve to see the power of weak diagnosis here, is that observing a *and* b also weakly diagnoses ϕ. In fact, every weakly fair configuration that allows to observe a and b must contain an occurrence of t_5, which marks p_4 and forces to fire either t_6 or t_7. By the same argument as above, the fault is bounded to happen. On the other hand, observing only a or even a sequence a^k is not sufficient for diagnosing ϕ (consider a weakly fair run composed only of occurrences of t_1, t_9 and t_{10}).

The *diagnosis problem* is to decide, given N and an observation α, whether or not α weakly diagnoses ϕ. One of the keys to solve it is deciding relation (70) for a given configuration.

6.4 A SOLUTION USING EXTENDED REVEALS

By Def. 19 and definition of $\twoheadrightarrow$, α diagnoses ϕ iff

$$\forall \mathcal{C} \in expl(\alpha)\ \forall \omega \in \Omega\colon (\mathcal{C} \subseteq \omega \ \Rightarrow\ E_\phi \cap \omega \neq \emptyset) \tag{71}$$

Swapping the two $\forall$, this can be equivalently rephrased as:

$$\forall \omega \in \Omega\colon\ (\exists \mathcal{C} \in expl(\alpha)\colon \mathcal{C} \subseteq \omega) \ \Rightarrow\ E_\phi \cap \omega \neq \emptyset \tag{72}$$

In this section we derive an algorithm for deciding the *negation* of (72), i.e., given α, the algorithm decides whether or not there is some $\omega \in \Omega$ that contains an explanation $\mathcal{C} \in expl(\alpha)$ and such that $\omega \cap E_\phi = \emptyset$. Deriving this algorithm needs to overcome two obstacles:

1. $expl(\alpha)$ may be infinite due to *unobservable loops*. In Fig. 29, a is explained by any configuration in which t_9 and t_{10} fire any number of times, followed by e_1.
2. Ω is an infinite set in general, which we need to finitely represent while still being able to check for set inclusion or whether $E_\phi \cap \omega \neq \emptyset$ for each weakly fair ω.

The main ideas behind our solution can be summarized as follows. Following [EK12], we fix the first problem by showing that it is *sufficient* for deciding (72) to search for $\mathcal{C}$ within a *finite subset* of $expl(\alpha)$, instead of the entire, potentially infinite set of explanations. Because this subset is finite, there exists an unfolding prefix that contains it entirely (P_α), and we will see how to construct it. Once such a configuration $\mathcal{C}$ has been found, the algorithm needs to decide if it can be extended into a fault-free, weakly fair $\omega \in \Omega$. We show (Lemma 15) that this is the case iff two configurations $\mathcal{C}_1 \subset \mathcal{C}_2$ exist such that both of them reach the same marking, both are free of faults, $\mathcal{C}_2$ disables every event enabled by $\mathcal{C}_1$, and $\mathcal{C} \subseteq \mathcal{C}_1$. This result does not quite yet give an algorithm, as, e.g., $\mathcal{C}_2$ could be unboundedly large. To fix this, we define two *finite* unfolding prefixes $\mathcal{P}_N^1$ and $\mathcal{P}_N^2$ verifying that (*i*), $\mathcal{P}_N^1$ is contained in $\mathcal{P}_N^2$ and, (*ii*), the aforementioned $\mathcal{C}_1, \mathcal{C}_2$ exist iff configurations $\tilde{\mathcal{C}}_1 \in conf(\mathcal{P}_N^1)$ and $\tilde{\mathcal{C}}_2 \in conf(\mathcal{P}_N^2)$ exist and satisfy (again) that $\tilde{\mathcal{C}}_1, \tilde{\mathcal{C}}_2$ reach the same marking, both are free of faults, $\tilde{\mathcal{C}}_2$ disables all events enabled by $\tilde{\mathcal{C}}_1$, and some other technical condition asking (modulo details) that $\mathcal{C} \subseteq \tilde{\mathcal{C}}_1$. This 'iff' is shown in Lemma 17. These two prefixes can be seen as fixing the second problem mentioned above. Our main result, Theorem 3, formalizes these ideas; § 6.5 derives a decision procedure from them.

6.4.1 *Succinct Explanations*

In this section, we define, following [EK12], a finite subclass of explanations of α and show, in Lemma 14, that this class is *sufficient* for deciding (72), i.e., that the existential quantification of $\mathcal{C}$ in (72) can be restricted to this finite class.

Definition 20. *Configuration $\mathcal{C} \in conf(\mathcal{U}_N)$ is* verbose *if it contains two events $e, e' \in \mathcal{C}$ satisfying*

- $e' < e$, *and* (73)

- $mark([e']) = mark([e])$, *and* (74)
- $obs([e']) = obs([e])$. (75)

If $\mathcal{C}$ is not verbose, it is succinct.

Intuitively, $\mathcal{C}$ is verbose if it contains the occurrence of some loop of N consisting entirely of unobservable transitions. In Fig. 28, t_2, t_3 is an unobservable loop, that produces the verbose configuration $\{e_1, e_3', e_2'\}$ in Fig. 29 (set e, e' of Def. 20 as $e := e_2'$ and $e' := e_1$).

Observe that $lpo([e'])$ is isomorphic to $lpo([e])$, by (75) and Lemma 12, and that $[e'] \subseteq [e]$, by (73). This means that all events in $[e] \setminus [e']$ are unobservable, see Fig. 30 (a). It also means that $lpo([e]) = lpo([e'])$, i.e., the LPO isomorphism is the identity function on $\mathcal{C}$ restricted to observable events. Finally, observe that (74) does *not* imply (73), even for 1-safe nets.[2]

Def. 20 is different from the equivalent definition in [EK12] only in that $obs(\mathcal{C})$ is defined differently: here, it is a set of LPOs while in [EK12] it is the set of sequences $\lambda(\sigma)$ where σ is an interleaving of $\mathcal{C}$.

We now introduce the notion of *peeling* a configuration, which will be necessary in subsequent proofs. Intuitively, if $\mathcal{C}$ is a verbose explanation of some α, peeling corresponds to finding a shorter explanation $\mathcal{C}'$ where unnecessary (unobservable) fragments of $\mathcal{C}$ have been removed. Peeling $\mathcal{C}$ one time yields a shorter explanation of α that reaches the same marking as $\mathcal{C}$. The new configuration may not yet be succinct, but since the operation reduces the size of the explanation, finite explanations need to be peeled only finite number of times until getting a succinct explanation.

Let us formalise this idea. Let $\mathcal{C}$ be verbose and finite, and let $e, e' \in \mathcal{C}$ be events satisfying (73)–(75). Let $I := \mathcal{C} \setminus [e]$. We define $peel_{e,e'}(\mathcal{C})$ as the configuration

$$\mathcal{C}' := [e'] \cup I',$$

where I' is the isomorphic copy of I that *continues* $\mathcal{U}_N$ just after $cut([e'])$. Recall that I' is well defined, as we have explained in § 2.4.2. Since $|[e']| < |[e]|$, we have $|\mathcal{C}'| < |\mathcal{C}|$. So if $\mathcal{C}'$ is still verbose, we only need to peel again finitely many times before obtaining a succinct configuration. We denote by $peelmax(\mathcal{C})$ the set of *succinct* configurations resulting from peeling $\mathcal{C}$ as many times as necessary (choosing any e, e' that satisfy (73)–(75) every time we peel).[3] Lemma 13 implies that $peelmax(\mathcal{C})$ are explanations of α if $\mathcal{C}$ is.

Lemma 13. *For any verbose configuration $\mathcal{C}$ with $\mathcal{C}' := peel_{e,e'}(\mathcal{C})$, it holds that:*

- $mark(\mathcal{C}) = mark(\mathcal{C}')$ (76)
- $obs(\mathcal{C}) \subseteq obs(\mathcal{C}')$ (77)
- $\mathcal{C}' \cap E_\phi \neq \emptyset \;\Rightarrow\; \mathcal{C} \cap E_\phi \neq \emptyset$ (78)

Proof. Let $e, e' \in \mathcal{C}$ be events satisfying (73)-(75). Recall that $\mathcal{C}$ has the form $[e] \uplus I$, and $\mathcal{C}'$ the form $[e'] \uplus I'$.

(76) is a consequence of (74) and the fact that I, I' are isomorphic. Showing (77) is more laborious. Let $lpo(\mathcal{C}) := \langle S, <, \lambda \rangle$ and $lpo(\mathcal{C}') := \langle S', <, \lambda \rangle$. In the sequel we define a mapping $g \colon S' \to S$ and prove that g is a bijective homomorphism from $lpo(\mathcal{C}')$ to $lpo(\mathcal{C})$. (77) then follows by Lemma 11.

Let f_1 be the LPO isomorphism between $lpo([e])$ and $lpo([e'])$. Recall that f_1 is actually the identity function. Let $f_2 \colon I' \to I$ be the isomorphism between I' and I. Define $g := f_1 \cup f_2'$ where f_2' is the restriction of f_2 to S', i.e., the observable events of $\mathcal{C}'$.

2 More precisely, there is a 1-safe net whose unfolding contains a configuration that has two events, e and e', such that $mark([e']) = mark([e])$ but $\neg(e' < e)$.

3 We conjecture that $peelmax(\mathcal{C})$ is a singleton, but do not rely on it in the sequel.

Observe that g is bijective because it is the union of two bijections whose domains and codomains are disjoint; it satisfies (66) because so do f_1 and f_2. Finally, for $e_1, e_2 \in S'$ with $e_1 < e_2$, we show that (67) holds. There are three cases:

- $e_1, e_2 \in [e']$. Then $g(e_1) = e_1 < e_2 = g(e_2)$.
- $e_1, e_2 \in I'$. Since the isomorphism f_2 preserves causality, we have $g(e_1) = f_2(e_1) < f_2(e_2) = g(e_2)$.
- $e_1 \in [e']$ and $e_2 \in I'$. Then there is some $c \in cut([e'])$ such that $e_1 < c < e_2$. Since N is safe and by (74), there is a single condition $c' \in cut([e])$ such that $h(c) = h(c')$, where h is $\mathcal{U}_N$'s labelling. It does not hold that $c \# c'$ or $c \parallel c'$, because $[c] \cup [c'] \subseteq \mathcal{C}$ and N is safe. So $c \leq c'$ holds, since $c' < c$ contradicts $e' < e$. Since $[e_2]$ consumes c, necessarily $[f_2(e_2)]$ consumes c', as I' and I are isomorphic. We therefore have:

$$g(e_1) = f_1(e_1) = e_1 < c \leq c' < f_2(e_2) = g(e_2)$$

As for (78), let $\tilde{e} \in \mathcal{C}' \cap E_\phi$ be some fault. If $\tilde{e} \in [e'] \subseteq \mathcal{C}$, then $\tilde{e} \in \mathcal{C}$. If $\tilde{e} \in I'$, then $f_2(\tilde{e}) \in I \subseteq \mathcal{C}$ is also fault, because f_2 preserves transition labels. In both cases $\mathcal{C} \cap E_\phi \neq \emptyset$. □

We define the set of succinct explanations of α as

$$succexpl(\alpha) := \{\mathcal{C} \in expl(\alpha) \colon \mathcal{C} \text{ is succinct}\}$$

Proposition 13. *$succexpl(\alpha)$ is finite.*

Proof. Since N is finite, there are finitely many events in $\mathcal{U}_N$ of depth less or equal to any given $n \in \mathbb{N}$, and thus finitely many configurations made up of such events. Assume now there are infinitely many succinct explanations of α. Because of the above, there must be a succinct explanation $\mathcal{C}$ that contains $e \in \mathcal{C}$ such that $depth(e) = k(|\alpha| + 1)$, where k is the number of reachable markings in the net. Let

$$e_1 < \ldots < e_{|\alpha|+1} = e$$

be the events of some causality chain from the initial marking $\tilde{m}_0$ to e, such that $mark(e_1) = \ldots = mark(e_{|\alpha|+1})$, which exist by the pigeonhole principle. Since only $|\alpha|$ events in $\mathcal{C}$ are observable, $[e_{i+1}] \setminus [e_i] \subseteq E_{ubs}$ holds for some i. So $\mathcal{C}$ contains two causally related events, whose local configurations have the same LPO, thus the same observation (Lemma 12), and reach the same marking. Then $\mathcal{C}$ is verbose, a contradiction. □

The previous proof works even if (73) is removed from Def. 20. However, (73) is required for the following statement.

Proposition 14. *The shortest explanations of α (in number of events) are succinct.*

Proof. If $\mathcal{C}$ is an explanation of α and it is verbose, then $peelmax(\mathcal{C})$ are shorter explanations of α, so no shortest explanation can be found among the verbose ones. □

Prop. 14 would still work if (73) is replaced by the more general condition $|[e']| < |[e]|$, but then (77) would become false.

The main lemma of this subsection shows that (72) can be rephrased into (79), eliminating the need to examine all potentially infinitely many explanations of α.

Lemma 14. *Observation pattern α diagnoses ϕ iff*

$$\forall \omega \in \Omega \colon \ (\exists \mathcal{C} \in succexpl(\alpha) \colon \mathcal{C} \subseteq \omega) \Rightarrow E_\phi \cap \omega \neq \emptyset \tag{79}$$

Proof. (72) and (79) differ only in that $\mathit{succexpl}(\alpha)$ has replaced $\mathit{expl}(\alpha)$. Trivially, (72) implies (79). Assume that (79) holds and let $\omega \in \Omega, \mathcal{C} \in \mathit{expl}(\alpha)$ be such that $\mathcal{C} \subseteq \omega$. We show that $\omega \cap E_\phi \neq \emptyset$. If $\mathcal{C}$ is succinct, we are done, so assume $\mathcal{C}$ is verbose and let $\mathcal{C}' \in \mathit{peelmax}(\mathcal{C})$. By (76), we can append an isomorphic copy I of $\omega \setminus \mathcal{C}$ to $\mathcal{C}'$, yielding a weakly fair configuration $\omega' := \mathcal{C}' \cup I$. By (77), $\mathcal{C}'$ is a succinct explanation of α. Because $\mathcal{C}' \subseteq \omega'$ and (79), it holds that $\omega' \cap E_\phi \neq \emptyset$. If $\mathcal{C}' \cap E_\phi \neq \emptyset$, by (78), we have $\mathcal{C} \cap E_\phi \neq \emptyset$. If $I \cap E_\phi \neq \emptyset$, then $\omega \setminus \mathcal{C}$ must contain a fault because it is isomorphic to I. In any case, $\omega \cap E_\phi \neq \emptyset$. □

6.4.2 *Characterizing Weakly Fair Configurations*

We now investigate a finite characterization of the weakly fair configurations in Ω that allows to reason about (*i*) inclusion of (succinct) explanations and (*ii*) absence of faults.

According to (79), α does *not* diagnose ϕ iff we can find a fault-free, weakly fair configuration ω that contains a succinct explanation. The next lemma establishes a characterization of such weakly fair configurations, where the spoilers of an event play an important role.

The main idea is simple: given a finite configuration $\mathcal{C}$ that plays the role of the explanation, a fault-free, weakly fair configuration that extends $\mathcal{C}$ exists iff one can find configurations $\mathcal{C}_1, \mathcal{C}_2$ with $\mathcal{C}_1 \subseteq \mathcal{C}_2$, both free of faults, extending $\mathcal{C}$, reaching the same marking, and such that $\mathcal{C}_2 \setminus \mathcal{C}_1$ disables all events enabled by $\mathcal{C}_1$. Since both reach the same marking, the fragment $\mathcal{C}_2 \setminus \mathcal{C}_1$ can then be iterated infinitely often without ever leaving any event enabled:

Lemma 15. *Let $\mathcal{C}$ be a finite configuration. There exists a weakly fair $\omega \in \Omega$ such that $\mathcal{C} \subseteq \omega$ and $\omega \cap E_\phi = \emptyset$ iff there are $\mathcal{C}_1, \mathcal{C}_2 \in \mathit{conf}(\mathcal{U}_N)$ satisfying*

- $\mathcal{C} \subseteq \mathcal{C}_1 \subseteq \mathcal{C}_2$, *and* (80)
- $\mathit{mark}(\mathcal{C}_1) = \mathit{mark}(\mathcal{C}_2)$, *and* (81)
- $\forall e \in E : \mathcal{C}_1 \overset{e}{\rightsquigarrow} \Rightarrow \mathit{spoilers}(e) \cap \mathcal{C}_2 \neq \emptyset$, *and* (82)
- $\mathcal{C}_2 \cap E_\phi = \emptyset$. (83)

Proof. Let $\omega \in \Omega$ be weakly fair, such that $\mathcal{C} \subseteq \omega$ and $\omega \cap E_\phi = \emptyset$. Let $\sigma = e_1 e_2 \ldots$ be any weakly fair interleaving of ω, and e_n the last event of $\mathcal{C}$ in σ. By the pigeonhole principle there are infinitely many $n \leq n_1 < n_2 < n_3 < \ldots \in \mathbb{N}$ such that

$$\mathit{mark}(\sigma_{n_1}) = \mathit{mark}(\sigma_{n_2}) = \mathit{mark}(\sigma_{n_3}) = \ldots$$

where σ_i denotes the run $e_1 e_2 \ldots e_i$. Let $\mathcal{C}_1$ be the restriction of ω to σ_{n_1}. Because σ is weakly fair, there is some $i \in \mathbb{N}$ such that σ_{n_i} contains one spoiler for every event enabled by $\mathcal{C}_1$ (Lemma 9). Let $\mathcal{C}_2$ be the restriction of ω to σ_{n_i}. Then $\mathcal{C}_1, \mathcal{C}_2$ satisfy (80)-(83).

For the opposite direction, let $\mathcal{C}_1, \mathcal{C}_2$ be configurations satisfying (80)-(83). We construct a fault-free, weakly fair $\omega \in \Omega$. For convenience, we write $\mathit{ploop}(\mathcal{C}_1, \mathcal{C}_2)$ for all pairs $\mathcal{C}_1, \mathcal{C}_2 \in \mathit{conf}(\mathcal{U}_N)$ satisfying (80) and (81). Since $\mathit{mark}(\mathcal{C}_1) = \mathit{mark}(\mathcal{C}_2)$, we can append an isomorphic copy of $\mathcal{C}_2 \setminus \mathcal{C}_1$ to $\mathcal{C}_2$, yielding $\mathcal{C}_3$, such that $\mathit{ploop}(\mathcal{C}_2, \mathcal{C}_3)$ and $\mathcal{C}_3 \cap E_\phi = \emptyset$ hold. Iterating this construction, one can obtain a family $(\mathcal{C}_n)_{n \in \mathbb{N}}$ of configurations satisfying $\mathit{ploop}(\mathcal{C}_n, \mathcal{C}_{n+1})$ and $\mathcal{C}_n \cap E_\phi = \emptyset$ for all $n \in \mathbb{N}$. We now let

$$\omega := \bigcup_{n \in \mathbb{N}} \mathcal{C}_n$$

be the configuration resulting from their union. We prove that ω does not enable any event, and thus $\omega \in \Omega$ by Lemma 8. By contradiction, let e be any event enabled by ω. It is therefore enabled by $\mathcal{C}_i$ for some $i \in \mathbb{N}$. Since the unfolding after $cut(\mathcal{C}_i)$ is isomorphic to the unfolding after $cut(\mathcal{C}_1)$, there is some e' isomorphic to e that is enabled by $\mathcal{C}_1$. By construction, $spoilers(e') \cap \mathcal{C}_2 \neq \emptyset$, so there is some $\widehat{e'}$ in $\mathcal{C}_2 \setminus \mathcal{C}_1$ that disables e'. Because $\mathcal{C}_{i+1} \setminus \mathcal{C}_i$ is isomorphic to $\mathcal{C}_2 \setminus \mathcal{C}_1$, there is some spoiler of e in $\mathcal{C}_{i+1} \setminus \mathcal{C}_i$, and ω does not enable e, a contradiction. □

As an example, consider the weakly fair run $(t_1, t_9, t_{10})^\omega$ in Fig. 28. It is represented in Fig. 29 by the fault-free, weakly fair configuration $\omega := \{e_1, e_9, e_{10}, e_1', \ldots\} \in \Omega$. Setting $\mathcal{C} := \emptyset$, Lemma 15 implies the existence of $\mathcal{C}_1 = \emptyset$ and $\mathcal{C}_2 = \{e_1, e_9, e_{10}\}$.

While Lemma 15 identifies a method for finding fault-free, weakly fair configurations that extend a given configuration, there are still infinitely many configurations $\mathcal{C}_1, \mathcal{C}_2$ to consider. We would thus like to define a *finite* unfolding prefix of $\mathcal{U}_N$ such that, $\mathcal{C}_1, \mathcal{C}_2$ exist and verify (80) to (83) iff there are *small copies* of $\mathcal{C}_1, \mathcal{C}_2$ among the configurations of such a prefix and they still satisfy (80) to (83).

A first approach would be considering a finite prefix that is large enough to contain such small copies. Observe that $\mathcal{C}_1$ is a superset of $\mathcal{C}$. This means that $\mathcal{C}$ should be included in the prefix, and $\mathcal{C}$ could be any succinct explanation, so the prefix should include all succinct explanations. Conceptually, it should also *unfold enough* to see all markings that can be reached from any explanation (to find $\mathcal{C}_1$), and all markings that can be reached from those ones (to find $\mathcal{C}_2$). Such a prefix would probably be very large. Fortunately, the same information that this prefix carries can be represented in, not one, but two unfolding prefixes.

Assume that we already have a prefix $\mathcal{P}_\alpha$ that contains all succinct explanations. In order to find configurations $\mathcal{C}_1, \mathcal{C}_2$ as in Lemma 15, we first need to search for any configuration $\mathcal{C}_1$ such that $\mathcal{C} \subseteq \mathcal{C}_1$. The key observation is that such $\mathcal{C}_1$ *exists* iff $mark(\mathcal{C}_1)$ is reachable from $mark(\mathcal{C})$ without firing faulty events, as (83) requests. So we do not need to extend $\mathcal{P}_\alpha$ into a larger prefix where to find $\mathcal{C}_1$. Essentially, all that we need is a prefix that has the following property: marking m' is reachable from marking m in N iff the prefix contains configurations $\mathcal{C}_m \subseteq \mathcal{C}_{m'}$ such that $mark(\mathcal{C}_m) = m$ and $mark(\mathcal{C}_{m'}) = m'$. Such prefix can actually be understood as two overlapping prefixes, an *small* prefix where $\mathcal{C}_m$ is to be found and a *large* prefix, which extends the small one, and where $\mathcal{C}_{m'}$ is to be found. This idea is used by [HKS13] for a similar purpose.

Next, we need to find a suitable $\mathcal{C}_2$, which by Lemma 15 must include $\mathcal{C}_1$ and fire at least one spoiler for each event enabled by $\mathcal{C}_1$. This requires to reason not only about the possibility of reaching one marking from another one, but also about which events have been disabled on the way, or which spoilers have been fired. We show in the sequel how to construct the aforementioned *small* and *large* to convey complete information with regard to this stronger requirement.

So we need to define two prefixes. For the *small* one, any finite, marking-complete prefix will be enough. Intuitively, the first prefix just needs to represent all possible reachable markings. We explained in Ch. 3 how to construct marking-complete prefixes for contextual nets, and methods and

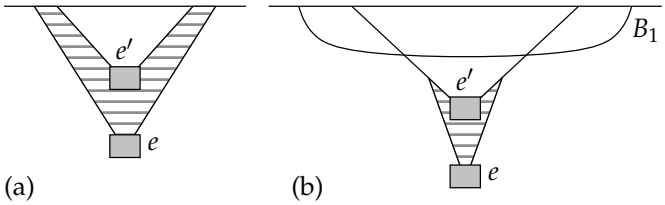


Figure 30: The lined area in (a) represents unobservable, ε-labelled, events of a verbose explanation. In (b), the lined area depicts $D := [e] \setminus [e']$ in Def. 21, which does not consume any condition in B_1.

tools to construct them for ordinary nets have already been described [EH08; ERV02; McM93b], so we will not emphasize this point. We denote by

$$\mathcal{P}_N^1 := \langle B_1, E_1, G_1, \tilde{m}_0 \rangle$$

some finite, marking-complete prefix of $\mathcal{U}_N$. For any configuration $\mathcal{C}$ of $\mathcal{U}_N$, we denote by $\mathcal{C}_{E_1}$ the set $\mathcal{C} \cap E_1$ of events in $\mathcal{C}$ that are also in $\mathcal{P}_N^1$.

Then we need to define a prefix $\mathcal{P}_N^2$ that includes $\mathcal{P}_N^1$ and preserves not only reachability of markings but also the capability of a configuration to *spoil* previously enabled events, i.e., those events that can take the role of $\mathcal{C}_2$ in (82). The prefix $\mathcal{P}_N^2$ will be defined using the following notion of cutoff, which is relative to $\mathcal{P}_N^1$:

Definition 21. *Event $e \in E$ is an* sp-cutoff *w.r.t. $\mathcal{P}_N^1$ if there is $e' \in E$ such that, setting $D := [e] \setminus [e']$, we have $e' < e$, and*

- $h({}^\bullet D \setminus D^\bullet) = h(D^\bullet \setminus {}^\bullet D)$, *and* (84)
- $B_1 \cap {}^\bullet D = \emptyset$. (85)

The main intuition behind this cutoff criterion is that an event e should be considered cutoff if it spoils the same events, among all those enabled by any $\mathcal{C}_1 \in \mathit{conf}(\mathcal{P}_N^1)$, that some other event e' which reaches the same marking and is a causal predecessor of e.

Intuitively, this is achieved by Def. 21 in the following way, which we illustrate in Fig. 30 (b). The set D in Def. 21 is the *difference* between $[e]$ and $[e']$, and (85) ensures that e and e' consume exactly the same conditions generated by events of $\mathcal{P}_N^1$. As a result, both events are in conflict with exactly the same events enabled by any configuration of $\mathcal{P}_N^1$. On the other hand, that both events reach the same marking is a consequence of (84), which implies that $\mathit{mark}([e']) = \mathit{mark}([e])$, as Lemma 16 shows:

Lemma 16. *Let e be an sp-cutoff, and e' as in Def. 21. For all $\mathcal{C} \in \mathit{conf}(\mathcal{P}_N^1)$, if $\mathcal{C} \cup [e]$ is a configuration, then*

$$\mathit{mark}(\mathcal{C} \cup [e]) = \mathit{mark}(\mathcal{C} \cup [e']).$$

Proof. (sketch) Show by cases, using (84)-(85), that

$$f((\tilde{m}_0 \cup \mathcal{C}^\bullet \cup [e]^\bullet) \setminus ({}^\bullet\mathcal{C} \cup {}^\bullet[e])) = f((\tilde{m}_0 \cup \mathcal{C}^\bullet \cup [e']^\bullet) \setminus ({}^\bullet\mathcal{C} \cup {}^\bullet[e'])). \qquad \square$$

An important remark is that no event in $\mathcal{P}_N^1$ can be an sp-cutoff, as otherwise the difference set D would consume at least one condition in B_1. We can now define

$$\mathcal{P}_N^2 := \langle B_2, E_2, G_2, \tilde{m}_0 \rangle$$

as the largest unfolding prefix that contains no sp-cutoff events, i.e., E_2 is the $\subseteq$-maximal set of events that is causally closed and contains no sp-cutoff. It is immediate to see that $\mathcal{P}_N^2$ is well-defined, in particular that it is unique. Because no sp-cutoff is contained in $\mathcal{P}_N^1$, we get that $\mathcal{P}_N^1$ is a prefix of $\mathcal{P}_N^2$.

Proposition 15. *$\mathcal{P}_N^2$ is finite.*

Proof. Assume E_2 is infinite. As in the proof of Prop. 13, we can find infinitely many events $e_1 < e_2 < \ldots$ in E_2. Because T and the number of reachable markings in N are finite, we can furthermore assume that $h(e_1) = h(e_2) = \ldots$ and $mark([e_1]) = mark([e_2]) = \ldots$ Define the sequence of *difference* sets $D_i = [e_i] \setminus [e_{i+1}]$, for $i \geq 1$. Since $D_i \neq \emptyset$ for all $i \geq 1$, we have ${}^\bullet D_i \neq \emptyset$. For $i < j$, ${}^\bullet D_i \cap {}^\bullet D_j = \emptyset$, otherwise $[e_j]$ would have conflicts. Since B_1 is finite, the number of D_is consuming from B_1 must be finite. So there is some $k \geq 1$ such that $B_1 \cap {}^\bullet D_i = \emptyset$ holds for all $i \geq k$. Then e_{k+1} is an sp-cutoff, a contradiction. □

We have now defined two finite unfoldings prefixes, $\mathcal{P}_N^1$ and $\mathcal{P}_N^2$, where $\mathcal{P}_N^1$ is a prefix of $\mathcal{P}_N^2$. Our next goal is showing that, roughly speaking, configurations $\mathcal{C}_1, \mathcal{C}_2$ that satisfy Lemma 15 exist iff it is possible to find representatives of them in $\mathcal{P}_N^1, \mathcal{P}_N^2$. Specifically, we want to show that for any given $\mathcal{C}_1 \in conf(\mathcal{P}_N^1)$, there exists some $\mathcal{C}_2 \in conf(\mathcal{U}_N)$ that satisfies (80) to (83) iff there is some $\mathcal{C}_2' \in conf(\mathcal{P}_N^2)$ such that $\mathcal{C}_1, \mathcal{C}_2'$ satisfy (80) to (83). To define $\mathcal{C}_2'$ we introduce the notion of trimming a configuration, and prove some properties of this transformation.

Any configuration $\mathcal{C}$ of $\mathcal{U}_N$ that contains some sp-cutoff e can be *trimmed* in a way analogous to the way verbose configurations can be peeled into succinct configurations, cf. Lemma 13. Trimming $\mathcal{C}$ corresponds to finding some smaller configuration $\mathcal{C}'$ that preserves the aforementioned spoiling capabilities.

Formally, let $\mathcal{C}$ be any configuration that contains an sp-cutoff e and event $e' \in \mathcal{C}$ as in Def. 21. Consider the configuration $\mathcal{C}_{E_1} \cup [e] \subseteq \mathcal{C}$. Since $\mathcal{C}_{E_1}$ is a configuration of $\mathcal{P}_N^1$, by Lemma 16, we have that

$$mark(\mathcal{C}_{E_1} \cup [e]) = mark(\mathcal{C}_{E_1} \cup [e']).$$

So we can partition $\mathcal{C}$ as $(\mathcal{C}_{E_1} \cup [e])$ and $I := \mathcal{C} \setminus (\mathcal{C}_{E_1} \cup [e])$, and define $trim_{e,e'}(\mathcal{C})$ as the configuration

$$\mathcal{C}' := (\mathcal{C}_{E_1} \cup [e']) \cup I',$$

where I' is the isomorphic copy of I after $cut(\mathcal{C}_{E_1} \cup [e'])$.

Lemma 17. *Let $\mathcal{C}' := trim_{e,e'}(\mathcal{C})$ for any configuration $\mathcal{C}$ of $\mathcal{U}_N$. We have:*

- $mark(\mathcal{C}) = mark(\mathcal{C}')$ (86)
- $\mathcal{C} \cap E_1 \subseteq \mathcal{C}' \cap E_1$ (87)
- $\mathcal{C} \cap E_\phi = \emptyset \Rightarrow \mathcal{C}' \cap E_\phi = \emptyset$ (88)
- $\forall e \in B_1^\bullet$: $spoilers(e) \cap \mathcal{C} \neq \emptyset \Rightarrow spoilers(e) \cap \mathcal{C}' \neq \emptyset$ (89)
- $|\mathcal{C}'| < |\mathcal{C}|$ (90)

Proof. (87), (88), and (90) hold by construction of $\mathcal{C}'$. (86) is a consequence of the fact that $\mathcal{U}_N$ stripped of $\mathcal{C}_{E_1} \cup [e]$ is isomorphic to $\mathcal{U}_N$ stripped of $\mathcal{C}_{E_1} \cup [e']$. So isomorphic sets of events I and I' yield the same marking of N. As for (89), let $e, e' \in \mathcal{C}$ be as in Def. 21, let $\hat{e} \in B_1^\bullet$, and let $\hat{e}^\dagger \in spoilers(\hat{e}) \cap \mathcal{C}$. Three cases are possible:

- $\hat{e}^\dagger \in (\mathcal{C} \cap E_1) \cup [e']$. Then by construction, $\hat{e}^\dagger \in \mathcal{C}'$.
- $\hat{e}^\dagger \in [e] \setminus [e']$. Not possible, entails contradiction to (85).

- $\hat{e}^\dagger \in I$. Let $c \in {}^\bullet\hat{e}^\dagger \cap B_1$ be any condition in B_1 consumed by $\hat{e}^\dagger$. We show that $c \in cut(\mathcal{C}_{E_1} \cup [e'])$. This is because $c \in \tilde{m}_0 \cup (\mathcal{C} \cap E_1)^\bullet$ (since $\mathcal{C}$ is causally closed), and also $c \notin {}^\bullet(\mathcal{C}_{E_1} \cup [e'])$ (since the only event $\hat{e}^\dagger$ in $\mathcal{C}$ that consumes c is in I). With analogous reasoning, one shows that $c \in cut(\mathcal{C}_{E_1} \cup [e])$. Now, because I, starting from $cut(\mathcal{C}_{E_1} \cup [e])$, is isomorphic to I', starting from $cut(\mathcal{C}_{E_1} \cup [e'])$, and c belongs to both cuts, if $\hat{e}^\dagger$ consumes from c, its isomorphic event in I' also consumes from c and hence spoils $\hat{e}$. □

Trimming decrements the number of events (90), so if $trim_{e,e'}(\mathcal{C})$ still has a sp-cutoff we can trim again finitely many times until getting a configuration free of sp-cutoff events, choosing any e, e' as in Def. 21 every time we trim. Let $trimmax(\mathcal{C})$ denote the set of such configurations. Again, we conjecture that $trimmax(\mathcal{C})$ is a singleton, but this is not important for our purposes. Since it has no sp-cutoff, $trimmax(\mathcal{C})$ are always configurations of $\mathcal{P}_N^2$.

We can now define the configuration $\mathcal{C}_2'$ that we have mentioned, above, when motivating the investigation of trimming configurations. We said that $\mathcal{C}_2'$ needs to be in $conf(\mathcal{P}_N^2)$ and that $\mathcal{C}_1, \mathcal{C}_2'$ needs to satisfy (80) to (83). Taking $\mathcal{C}_2' := trimmax(\mathcal{C}_2)$ we get the desired result, which can be proved using Lemma 17.

We can now state the main result of the chapter:

Theorem 3. *Observation pattern α does* not *diagnose ϕ iff there exist configurations*

$$\mathcal{C}, \mathcal{C}_1' \in conf(\mathcal{U}_N), \qquad \mathcal{C}_1 \in conf(\mathcal{P}_N^1), \qquad \mathcal{C}_2 \in conf(\mathcal{P}_N^2)$$

satisfying the following properties:

- $\mathcal{C}$ *is a succinct explanation of* α, *and* (91)
- $\mathcal{C} \subseteq \mathcal{C}_1'$, *and* (92)
- $\mathcal{C}_1 \subseteq \mathcal{C}_2$, *and* (93)
- $mark(\mathcal{C}_1') = mark(\mathcal{C}_1) = mark(\mathcal{C}_2)$, *and* (94)
- $\forall e \in E : \mathcal{C}_1 \overset{e}{\rightsquigarrow} \Rightarrow spoilers(e) \cap \mathcal{C}_2 \neq \emptyset$, *and* (95)
- *there is no fault event in either* $\mathcal{C}_1'$ *or* $\mathcal{C}_2$. (96)

Proof. By (79), if α does not diagnose ϕ, there is a fault-free, weakly fair configuration $\omega \in \Omega$ and some succinct explanation $\mathcal{C} \in succexpl(\alpha)$ with $\mathcal{C} \subseteq \omega$. By Lemma 15, there are configurations $\tilde{\mathcal{C}}_1, \tilde{\mathcal{C}}_2$ that satisfy (80)-(83). Define $\mathcal{C}_1 \in conf(\mathcal{P}_N^1)$ as any configuration in $\mathcal{P}_N^1$ that reaches $mark(\tilde{\mathcal{C}}_1)$. Now let $\mathcal{C}_2'$ denote $\mathcal{C}_1 \cup I$ where I is an isomorphic copy of $\tilde{\mathcal{C}}_2 \setminus \tilde{\mathcal{C}}_1$ starting at $cut(\mathcal{C}_1)$. Define $\mathcal{C}_2$ as either $\mathcal{C}_2'$ if $\mathcal{C}_2'$ contains no sp-cutoff or as any configuration in $trimmax(\mathcal{C}_2')$ otherwise. In both cases, $\mathcal{C}_2 \in conf(\mathcal{P}_N^2)$. Define $\mathcal{C}_1' \subseteq \omega$ as any configuration satisfying (92) whose marking is $mark(\tilde{\mathcal{C}}_1)$, which exists because ω repeats $mark(\tilde{\mathcal{C}}_1)$ infinitely often.

(91) and (92) holds by definition of $\mathcal{C}, \mathcal{C}_1'$. By construction, $\mathcal{C}_1 \subseteq \mathcal{C}_2'$. If $\mathcal{C}_2'$ has sp-cutoffs and $\mathcal{C}_2$ is taken from $trimmax(\mathcal{C}_2')$, by (87) and the fact that $\mathcal{C}_1 \subseteq E_1$, we have $\mathcal{C}_1 \subseteq \mathcal{C}_2$. So (93) holds in any case. (94) holds by construction of $\mathcal{C}_1', \mathcal{C}_1$, (81), and (86). Because ω is fault-free, $\mathcal{C}_1'$ is as well. By (83), $\tilde{\mathcal{C}}_1, \tilde{\mathcal{C}}_2$ are fault free, and so is $\mathcal{C}_2'$ (by isomorphism). Then, by (88), $\mathcal{C}_2$ is fault-free. This shows (96). As for (95), we observe the following. $\tilde{\mathcal{C}}_1, \tilde{\mathcal{C}}_2$ satisfy (82). Since $\tilde{\mathcal{C}}_1$ cannot contain any spoiler of the events it enables, all such spoilers are in $\tilde{\mathcal{C}}_2 \setminus \tilde{\mathcal{C}}_1$. Then, $\mathcal{C}_2'$ disables all events enabled by $\mathcal{C}_1$ because $\mathcal{C}_2' \setminus \mathcal{C}_1$ is isomorphic to $\tilde{\mathcal{C}}_2 \setminus \tilde{\mathcal{C}}_1$. Now, because $\mathcal{C}_1 \subseteq E_1$, all such events are in $B_1^\bullet$. Then by (89), $\mathcal{C}_2$ disables all them, and (95) holds.

If $\mathcal{C}, \mathcal{C}_1', \mathcal{C}_1, \mathcal{C}_2$ exist and verify (91)-(96), by Lemma 15 some fault-free, weakly fair configuration $\omega \in \Omega$ exists and repeats infinitely often the

marking $mark(\mathcal{C}_1) = mark(\mathcal{C}'_1)$. Construct another weakly fair configuration $\omega' := \mathcal{C}'_1 \cup I \in \Omega$ where I is an isomorphic copy of $\omega \setminus \mathcal{C}_1$ starting at $cut(\mathcal{C}'_1)$. Now, ω' contains a succinct explanation and is fault-free because so was $\mathcal{C}'_1$, and by isomorphism between $\omega \setminus \mathcal{C}_1$ and I. □

In other words, Theorem 3 states that α does not diagnose ϕ iff one can find suitable configurations $\mathcal{C}_1, \mathcal{C}_2$ in certain *finite* unfolding prefixes and a succinct explanation $\mathcal{C}$ of α such that $mark(\mathcal{C}_1)$ can be reached from $mark(\mathcal{C})$ without executing fault events. There are only finitely many succinct explanations of α, and we can decide whether one marking is reachable from another without executing faults using the next proposition. So Theorem 3 suggests a decision algorithm for the diagnosis problem that we shall investigate in § 6.5.

Proposition 16. *There exist fault-free configurations $\mathcal{C}, \mathcal{C}'$ in $\mathcal{U}_N$ such that $\mathcal{C} \subseteq \mathcal{C}'$ iff there are fault-free configurations $\hat{\mathcal{C}} \subseteq \hat{\mathcal{C}}'$ of, respectively, $\mathcal{P}^1_N, \mathcal{P}^2_N$, that satisfy*

- $mark(\mathcal{C}) = mark(\hat{\mathcal{C}})$, *and*
- $mark(\mathcal{C}') = mark(\hat{\mathcal{C}}')$.

Proof sketch. $\mathcal{P}^1_N$ is marking-complete and we can find the requested fault-free $\hat{\mathcal{C}}$ in $conf(\mathcal{P}^1_N)$, see the proof of Proposition 4.9 (a) in [ERV02]. Let $\hat{\mathcal{C}}'' := \hat{\mathcal{C}} \cup I$ where I is an isomorphic copy of $\mathcal{C}' \setminus \mathcal{C}$ starting at $cut(\hat{\mathcal{C}})$. Let $\hat{\mathcal{C}}' := \hat{\mathcal{C}}''$ if $\hat{\mathcal{C}}'' \subseteq E_2$, or $\hat{\mathcal{C}}' \in trimmax(\hat{\mathcal{C}}'')$ otherwise. Then $\hat{\mathcal{C}}, \hat{\mathcal{C}}'$ satisfy the proposition by (86)-(88). □

6.5 A DECISION PROCEDURE FOR DIAGNOSIS

Theorem 3 states a set of necessary and sufficient conditions that characterize whether or not a given observation α diagnoses ϕ. In this section, we present a method for deciding if these conditions hold. We discuss which information is needed in order to decide them (§ 6.5.1), and how to obtain that information (§ 6.5.2). Based on this, we present an encoding of the diagnosis problem into SAT (§ 6.5.3).

6.5.1 *Preparation*

Given the observation α, we need to decide whether all conditions in Theorem 3 hold. Our goal is to minimize the work that is sensitive to changes in α, in particular when α is extended by additional observations. However, (91) and (92) seem to require constructing not only the prefix containing all succinct explanations, but also a large section of the unfolding beyond each of these explanations. Fortunately, this can be avoided thanks to Prop. 16 and the fact that $\mathcal{C}'_1$ and $\mathcal{C}_1$ in Theorem 3 do not depend on $\mathcal{C}$ being an explanation of α, merely on the fact that $\mathcal{C}'_1$ is a fault-free extension of $\mathcal{C}$. So, we only require that $mark(\mathcal{C}'_1)$ is reachable from $mark(\mathcal{C})$ without executing faults, and replace (92) by:

$$\exists \mathcal{C}' \colon\ mark(\mathcal{C}) = mark(\mathcal{C}') \wedge \mathcal{C}' \subseteq \mathcal{C}'_1 \tag{97}$$

Hence it suffices to construct only two unfolding prefixes:

- One prefix $\mathcal{P}_\alpha$ containing all succinct explanations of α, used to search for $\mathcal{C}$.
- Prefix $\mathcal{P}^2_N \succeq \mathcal{P}^1_N$, to check for the existence of a weakly fair configuration starting from a given marking (see Lemma 17), and whether one

marking is reachable from another (see Prop. 16). We shall search for $\mathcal{C}', \mathcal{C}_1$ in $\mathcal{P}^1_N$ and for $\mathcal{C}'_1, \mathcal{C}_2$ in $\mathcal{P}^2_N$.

Observe that restricting the construction of these prefixes to their fault-free parts automatically satisfies (96). Also, notice that $\mathcal{P}_\alpha$ depends only on the observation, whereas $\mathcal{P}^2_N$ depends only on N. So $\mathcal{P}^2_N$ can be constructed off-line, before α is acquired.

6.5.2 *Constructing the Prefixes*

We now explain how to compute the prefixes $\mathcal{P}^2_N$ and $\mathcal{P}_\alpha$. There exist well-known algorithms [ERV02] and efficient tools [Sch; Kho] for constructing Petri net unfoldings. For our constructions, the iterative structure of these algorithms can be maintained, it suffices to replace the criteria for cutoffs.

6.5.2.1 *Constructing $\mathcal{P}_\alpha$*

We need to restrict the unfolding construction as follows:

1. exclude fault events to ensure (96);
2. restrict to explanations of α; and
3. preserve all succinct explanations and eliminate all verbose ones.

For the first and second points, we synchronize N with a net representing $\alpha = \langle S, <, \lambda \rangle$. Let $S_{\min}$ (resp. $S_{\max}$) be the elements without predecessor (resp. successor) in S. We re-translate α into an occurrence net $O_\alpha = \langle P_\alpha, S, F_\alpha, m_\alpha \rangle$, whose events are S and whose causal relation is $<$. The definition of O_α is quite standard, we only remark that $P_\alpha := P_{min} \uplus P_{mid} \uplus P_{max}$ is partitioned in three sets, where P_{max} (resp. P_{min}) is the postset (resp. preset) conditions of $S_{\max}$ (resp. $S_{\min}$).

We then compose $N = \langle P, T, F, m_0 \rangle$ and O_α into a Petri net $N_\alpha = \langle P', T_o \cup T_u, G, m'_0 \rangle$, where:

- $P' = P \cup P_\alpha$;
- $T_o = \{ \langle t, s \rangle : t \in T^{obs}, s \in S, \lambda(t) = \lambda(s) \}$;
- $T_u = T^{ubs} \setminus \{\phi\}$;
- for $\langle t, s \rangle \in T_o$, ${}^\bullet\langle t, s \rangle = {}^\bullet t \cup {}^\bullet s$ and $\langle t, s \rangle^\bullet = t^\bullet \cup s^\bullet$
- for $t \in T_u$, ${}^\bullet t$ and $t^\bullet$ remain as in N;
- $m'_0 = m_0 \cup m_\alpha$.

Intuitively, N_α adds the places of O_α to N in order to record which parts of α have been seen during an execution. The observable transitions of N and O_α are synchronized to ensure that no run contradict α or add further observable events, and faults are excluded. Consider the unfolding $\mathcal{U}_{N_\alpha}$. Projecting each event labelled with a tuple $\langle t, s \rangle$ to t instead, then each configuration $\mathcal{C}$ of $\mathcal{U}_{N_\alpha}$ is also a configuration of $\mathcal{U}_N$; moreover $\mathcal{C}$ explains α iff $mark(\mathcal{C})$ contains P_{max}.

It remains to ensure the third point. Thanks to Def. 20 it suffices to cut the construction of $\mathcal{U}_{N_\alpha}$ at any event e such that there is another event $e' < e$ with $mark([e']) = mark([e])$. Indeed, this ensures both (74) and (75) as no observable event has occurred after e'. By the pigeon-hole principle on the finitely many reachable marking in N, this cutoff criterion is guaranteed to yield a finite prefix $\mathcal{P}_\alpha$.

6.5.2.2 *Constructing $\mathcal{P}^2_N$*

We construct $\mathcal{P}^2_N$ in two phases. First, $\mathcal{P}^1_N$ is obtained by the usual unfolding methods for marking-complete prefixes (e.g., [ERV02]). Then, we extend $\mathcal{P}^1_N$ by additional events, using Def. 21 as a cutoff criterion. Observe that deciding whether $e \in E$ is an sp-cutoff entirely depends information contained in $[e]$. In particular any strategy for extending the prefix during construction can be used.

6.5.3 *Encoding Diagnosis into SAT*

We propose an encoding of the diagnosis problem into SAT. Given prefixes $\mathcal{P}^1_N, \mathcal{P}^2_N, \mathcal{P}_\alpha$, computed as per § 6.5.2, we construct a formula φ that is satisfiable iff α does not diagnose ϕ. This approach immediately gives a decision procedure via efficient SAT solving. Not surprisingly, one can show (via reduction from the reachability problem for unfolding prefixes) that finding the configurations $\mathcal{C}, \mathcal{C}', \mathcal{C}_1, \mathcal{C}'_1, \mathcal{C}_2$ discussed in § 6.5.1 in these prefixes is NP-hard.

A SAT-based decision procedure for deadlock and coverability has been proposed in Ch. 4. There, a satisfying assignment represents one configuration with suitable properties. While we re-use this idea, the specificities of diagnosis require to encode multiple configurations and relate them according to Theorem 3.

For an unfolding prefix $\mathcal{P} := \langle\langle B, E, G, \tilde{m}_0\rangle, h\rangle$ of N or N_α, and a *label* l, we define the following collections of Boolean variables:

$$v(l) := \{\mathsf{v}^l_x \colon x \in B \cup E\},$$
$$m(l) := \{\mathsf{m}^l_p \colon p \in P\}.$$

Intuitively, all variables in $v(l)$ will encode a configuration that we will identify by l, and those in $m(l)$ a marking referred by l. For labels l, l', we define the following predicates. First we have the predicate

$$\begin{aligned}\mathsf{config}(l, \mathcal{P}) := &\Big(\bigwedge_{e\in E}\bigwedge_{e'\in {}^\bullet\!\bullet e}(\mathsf{v}^l_e \Rightarrow \mathsf{v}^l_{e'})\Big)\wedge\\ &\Big(\bigwedge_{c\in B, \{e_1,\dots,e_n\}={}^\bullet c}\mathsf{amo}(\mathsf{v}^l_{e_1},\dots,\mathsf{v}^l_{e_n})\Big)\wedge\\ &\Big(\bigwedge_{c\in B}\mathsf{v}^l_c \Leftrightarrow \big(\bigwedge_{e\in {}^\bullet c}\mathsf{v}^l_e \wedge \bigwedge_{e\in c^\bullet}\neg\mathsf{v}^l_e\big)\Big),\end{aligned}$$

which demands $v(l)$ to represent a configuration of $\mathcal{P}$ and its cut. The first conjunct requests the configuration to be causally closed; the second one asks for absence of symmetric conflicts and the third one defines the configuration's cut. Observe that these formulas are quite close to those presented in § 4.3, on p. 63. Next we have

$$\mathsf{subset}(l, l', \mathcal{P}) := \bigwedge_{e\in E}(\mathsf{v}^l_e \Rightarrow \mathsf{v}^{l'}_e),$$

which asks that l-labelled events are a subset of l'-labelled events. The predicate

$$\mathsf{mark}(l, l', \mathcal{P}) := \bigwedge_{p\in P}\Big(\mathsf{m}^l_p \Leftrightarrow \big(\bigvee_{c\in f^{-1}(p)}\mathsf{v}^{l'}_c\big)\Big)$$

computes the marking of a configuration represented by $v(l)$. It asks that $m(l)$ reflects the marking associated with the cut of $v(l)$, assuming that $v(l)$ encodes a configuration of $\mathcal{P}$. Similarly,

$$\mathsf{enables}(l, l', \mathcal{P}) := \bigwedge_{e \in E} (\mathsf{v}_e^{l'} \Leftrightarrow (\bigwedge_{c \in {}^\bullet e} \mathsf{v}_c^l))$$

assumes that $\mathsf{config}(l, \mathcal{P})$ holds, and forces the variable $\mathsf{v}_e^{l'}$ associated to an event e to be true iff e is enabled in the configuration represented by $v(l)$. The predicate

$$\mathsf{spoils}(l, l', \mathcal{P}) := \bigwedge_{e \in E} (\mathsf{v}_e^{l'} \Rightarrow (\bigvee_{e' \in ({}^\bullet e)^\bullet} \mathsf{v}_{e'}^l))$$

holds iff $v(l)$ has one spoiler for each event true in $v(l')$ and

$$\mathsf{explains}(l) := \bigwedge_{p \in P_{max}} \bigvee_{f(c)=p} \mathsf{v}_c^l$$

requests that the configuration referred by l is a succinct explanation of α.

All these predicates are linear except for $\mathsf{spoils}(\cdot)$, which can easily be made linear by introducing new variables for conditions. Also, remark that the amo() constraint in config() can be implemented in linear size, as we will explain in § 7.2.2. Also, not all of them are directly given in CNF, but a linear translation is always possible.

We can now turn to the encoding of Theorem 3, where (92) is replaced by (97), as discussed in § 6.5.1. Fix labels $\mathtt{m}, \mathtt{m}', \mathtt{C}, \mathtt{C}', \mathtt{C}'_1, \mathtt{C}_1, \mathtt{C}_2, \mathtt{D}$. Each of these labels identifies (the collection of Boolean variables representing) a configuration or a marking, except for $\mathtt{D}$, which represents a set of events. For instance $\mathtt{C}$ represents the configuration $\mathcal{C}$, note the different typography. Our formula φ is the conjunction of the following constraints:

1. $\mathsf{config}(\mathtt{C}, \mathcal{P}_\alpha) \wedge \mathsf{mark}(\mathtt{m}, \mathtt{C}, \mathcal{P}_\alpha) \wedge \mathsf{explains}(\mathtt{C})$
2. $\mathsf{config}(\mathtt{C}', \mathcal{P}_N^1) \wedge \mathsf{mark}(\mathtt{m}, \mathtt{C}', \mathcal{P}_N^1)$
3. $\mathsf{config}(\mathtt{C}'_1, \mathcal{P}_N^2) \wedge \mathsf{mark}(\mathtt{m}', \mathtt{C}'_1, \mathcal{P}_N^1)$
4. $\mathsf{config}(\mathtt{C}_1, \mathcal{P}_N^1) \wedge \mathsf{mark}(\mathtt{m}', \mathtt{C}_1, \mathcal{P}_N^2)$
5. $\mathsf{config}(\mathtt{C}_2, \mathcal{P}_N^2) \wedge \mathsf{mark}(\mathtt{m}', \mathtt{C}_2, \mathcal{P}_N^2)$
6. $\mathsf{subset}(\mathtt{C}', \mathtt{C}'_1, \mathcal{P}_N^1) \wedge \mathsf{subset}(\mathtt{C}_1, \mathtt{C}_2, \mathcal{P}_N^2)$
7. $\mathsf{enables}(\mathtt{C}_1, \mathtt{D}, \mathcal{P}_N^2) \wedge \mathsf{spoils}(\mathtt{C}_2, \mathtt{D}, \mathcal{P}_N^2)$

Only 1) actually depends on α, whereas remaining constraints can be built before α is known. 7) corresponds to (95), the others to (93), (94), and (97). Conditions (91) and (96) of Theorem 3 are guaranteed by construction.

6.6 CONCLUSION

We presented an unfolding-based decision procedure for solving the problem of *weak diagnosis* in partially observable safe Petri nets.

Weak diagnosis exploits indirect dependencies, captured by the *reveals* relations, to determine the inevitability of a fault. We stress that despite its name, 'weak' diagnosis is actually *stronger* than conventional diagnosis as in [BFHJ03]. Whereas in [BFHJ03] an observation can only be used to detect faults having occurred *in the past*, weak diagnosis captures also faults that are concurrent or in the future of the observation, under weak fairness. The requirement of [BFHJ03] that no unobservable cycle is present in the system

is also dropped, thanks to a characterization of the succinct explanations, that bound the unfolding prefix needed to perform diagnosis. The results here contain those of [EK12] and strengthen the existing approaches to the more powerful capability of *weak diagnosis*. We have shown how diagnosis can be performed using an algorithmic construction, and given an encoding into SAT.

While the chapter provides many of the ingredients necessary for an implementation, a practical obstacle to overcome could be the sizes of the prefixes required. As for prefix $\mathcal{P}_N^2$, note that it contains all system behaviors, but can be constructed off-line once and for all. The size of $\mathcal{P}_N^2$ can be exponential in the size of the net; however, it is known that unfoldings tend to be much smaller than this for systems that exhibit a high degree of concurrency. In general, the weak-diagnosis problem for Petri nets is PSPACE-complete (hardness follows by reduction from the reachability problem, membership by the fact that a fault-free weakly fair run matching the observation pattern can be nondeterministically simulated in linear space).

The prefix containing all the succinct explanations must be created online, but contains only the behaviors compatible with the observation. Notice that it can alternatively be obtained by producing a marking-complete unfolding of the net N_α from § 6.5.2, which should result in a reduction of its size. In this work, we omitted this possibility for the sake of a simpler presentation. Also, representing both prefixes as merged processes should result in a dramatic reduction of their size.

Future work also includes verification of *weak diagnosability* [HBFJ03; Haa07; Haa09; Haa10; AMH12] based on the results here. Further, local projections of observations, as exploited in [EK12], are interesting, especially in the context of *distributed* diagnosis.

7 TOOLS AND EXPERIMENTS

In Ch. 3 to 5 we presented methods for representing and analyzing the state space of a contextual net. Implementations supporting these methods have been made. In this chapter we report on these implementations and evaluate them on a standard benchmark suite.

Specifically, we present implementation details of the algorithm introduced in Ch. 3; we evaluate the optimizations proposed for the SAT encoding of Ch. 4; and we evaluate how compact are the contextual merged processes of Ch. 5 with respect to the various ways to unfold and merge a contextual net. This chapter is based on [RSB11b; BBC+12; RS12b; RSK13; RS13b].

All implementations discussed here are distributed with the Cunf Toolset, a collective name for a set of programs to carry out verification based c-net unfoldings. The toolset includes one program for constructing unfoldings, the unfolder Cunf, one program for analyzing coverability and deadlock, called Cna, and a number of programs for handling, e.g., contextual merged processes, format conversions, etc. All tools have been developed entirely within the frame of this thesis and are publicly available under

`http://code.google.com/p/cunf/`

The manual of the Cunf Toolset is included in this manuscript on App. A.

The outline of the chapter is as follows. In §§ 7.1 and 7.2 the implementations of Cunf and Cna are described. All benchmarks considered in this work are discussed in § 7.3, and the experimental evaluations of the different tools are presented in §§ 7.4 to 7.6. In § 7.7 we study the performance of merged processes and unfoldings on a model of Disktra's mutual exclusion algorithm [Dij65].

7.1 PREFIX CONSTRUCTION: THE CUNF UNFOLDER

In Ch. 3 we presented two approaches for constructing complete prefixes for contextual nets. We implemented both of them, in this section we mainly describe the eager approach to computing possible extensions. Specifically, based on the results of Ch. 3 we present concrete data structures to represent histories and algorithms for computing possible extensions and updating the concurrency relation.

The resulting tool, called Cunf, and included in the Cunf Toolset, expects as input a *1-safe* c-net and produces as output a complete unfolding prefix. Recall that the theory developed in Ch. 3 stands for bounded nets, not necessarily 1-safe. The tool, as many other unfolders, is restricted to 1-safe nets, as our examples of interest are in this domain. Moreover, this choice simplifies the implementation of certain data structures and algorithms.

Notice that there exists efficient tools for the unfolding of ordinary Petri nets, such as Mole [Sch] or Punf [Kho] — which as Cunf, can only deal with 1-safe nets. While much experience was gained from experimenting with the mature code of Mole, Cunf is not an extension of it. The issues of asymmetric conflict and histories permeate every aspect of the construction so that we went for a completely new implementation in C language,

comprising around 6.600 lines of code in version 1.6. CUNF matches the performance of dedicated Petri net unfolders like MOLE [Sch] or PUNF [Kho] on ordinary Petri nets, and additionally handles contextual unfoldings.

In this section, we review some features such as data structures and implementation details relevant to handling the complications imposed by contextual unfoldings. These helped to produce an efficient tool. Experiments with CUNF are reported in § 7.4.

7.1.1 *The History Graph*

Recall from Ch. 3 that the construction of a marking-complete enriched prefix $\mathcal{E}$ requires to deal with enriched events and enriched conditions. Recall that those are tuples $\langle e, H\rangle$ and $\langle c, H\rangle$ where H is a history of e or c.

CUNF needs to store them, and it does so in a graph structure. The construction algorithm iteratively extends the enriched prefix with new enriched events and conditions, so our data structure needs to be easy to extend when $\mathcal{E}$ is extended.

Formally, the *history graph* associated with $\mathcal{E}$ is a labelled directed graph $\mathcal{H}_\mathcal{E}$ such that

1. the nodes are the enriched events of $\mathcal{E}$;
2. there is an edge $\langle e, H\rangle \to \langle e', H'\rangle$ iff $e' \in H$ and $H' = H[\![e']\!]$ and either
   a) $(e'^\bullet \cup \underline{e'}) \cap {}^\bullet e \neq \emptyset$, or
   b) $e'^\bullet \cap \underline{e} \neq \emptyset$;
3. each node $\langle e, H\rangle$ is labelled by e.

The second condition states simply that $\mathcal{H}_\mathcal{E}$ has an edge from enriched event $\langle e, H\rangle$ to $\langle e', H'\rangle$ iff some enriched condition $\langle c, H'\rangle$ was used to construct $\langle e, H\rangle$ in the sense of the eager or lazy methods of Prop. 9 and Prop. 5 in § 3.7.

This structure allows CUNF to perform many operations efficiently: every enriched event appended to $\mathcal{H}_\mathcal{E}$ enlarges the graph by just one node plus some edges; common parts of histories are shared. We can easily enumerate the events of $H \in \chi(e)$ by following backwards the edges from node $\langle e, H\rangle$. The graph also represents, implicitly, the relation $\sqsubset$.

The unfolder also stores for every event e the set of histories $\chi(e)$ currently present in $\mathcal{E}$. This is done by means of a list of pointers to the corresponding nodes of $\mathcal{H}_\mathcal{E}$ that are labelled by e. This list is stored in the data structure that represent events. Given a condition c, we can enumerate its generating and reading histories similarly.

Compound conditions are stored in a shared-tree-like structure, where leaves represent reading histories and internal nodes compound histories. An internal node has two children, one of which is a leaf, the other either internal or a leaf. One easily sees that a compound history of c corresponds, w.l.o.g., to a union $H_1 \cup \ldots \cup H_n$ of reading histories. Every internal node represents such a union, and the structure allows sharing if one compound history contains another.

7.1.2 *Possible Extensions*

Computing the possible extensions (PE) of the prefix is the central task of any unfolder. This section explains which algorithms CUNF uses for this purpose.

CUNF behaves similar to MOLE or other unfolders in its flow of logic, but its actions are on enriched events and enriched conditions. The tool implements Algorithm 1. There, we denoted by $pe(\mathcal{E})$ the set of PEs of $\mathcal{E}$, formally defined in Def. 8, on p. 39. According to the eager approach to characterize PEs, presented in § 3.9.2, on p. 54, this set can be computed using the following steps, whose correctness relies on the Prop. 9. For any given transition t of N, do the following:

1. Let $\{p_1, \ldots, p_n\} := {}^\bullet t$ and $\{p'_1, \ldots, p'_m\} := \underline{t}$ be all places in the preset and context of t.
2. Search for all sets of ECs in $\mathcal{E}$ of the form $X_p := \{\rho_1, \ldots, \rho_n\}$ and $X_c := \{\rho'_1, \ldots, \rho'_m\}$ that satisfy all the following requirements:
   a) X_c contains only generating ECs;
   b) $h(\rho_i) = p_i$ and $h(\rho'_j) = p'_j$ for all $i = 1, \ldots, n$ and $j = 1, \ldots, m$.
   c) $\rho \mathbin{//} \rho'$ holds for all ρ in X_p and all ρ' in $X_p \cup X_c$;
   d) $\rho \parallel \rho'$ holds for all ρ, ρ' in X_c;
3. For any pair of sets X_p, X_c found in the previous step, the pair $\langle e, H\rangle$ is a PE of $\mathcal{E}$, where e is a (possibly new) event whose preset and context are all conditions in, respectively, X_p and X_c, and H is defined as $H := \{e\} \cup \bigcup_{\langle c, H'\rangle \in X_p \cup X_c} H'$.

These steps compute all extensions $\langle e, H\rangle$ such that $h(e) = t$. In principle, the function $pe(\mathcal{E})$ just needs to repeat the above steps for every transition t in N, and return all found PEs.

CUNF computes PEs using an optimized version of these steps, explained now. This optimization has already been described for ordinary unfoldings [KK01]. Observe that Algorithm 1 can be easily modified so that $pe(\mathcal{E})$ does not need to return *all* possible extensions of $\mathcal{E}$, but just those which are possible due to the last EC appended to $\mathcal{E}$. Indeed, after including a new EC ρ in the prefix, the only new PEs that $pe(\cdot)$ will return w.r.t. the last time it was called are those for which ρ is in X_p or X_c. So step 2 above can restrict the search in this way, and $pe(\cdot)$ can be modified to return PEs constructed using ρ, which now has to be passed to the function. Naturally, in Algorithm 1 we have to update the set X of possible extensions differently. Before, Algorithm 1 used the statement

$$X = pe(\mathcal{E}),$$

which now needs to be updated to

$$X = X \cup pe(\mathcal{E}, \rho).$$

Another consequence of this optimization is that $pe(\cdot)$ needs to be called after each EC ρ included in the prefix, and not just once after each enriched event is added. Extending the prefix with one enriched event $\langle e, H\rangle$ entails adding as many ECs as conditions there are in the postset of e, as we explain in § 7.1.3.

7.1.3 *Concurrency Relation*

We have seen how CUNF computes PEs and stores them in the history graph. Upon their discovery, PEs are kept in a *heap*, which allows to efficiently find the $\prec$-minimal history among those in PE, as Algorithm 1 requires.

Extending the prefix with a PE $\langle e, H\rangle$ gives rise to various types of ECs for whom CUNF needs to compute the concurrency relation. Namely, for every condition $c \in e^\bullet$, we have a new generating EC $\langle c, H\rangle$ and for every $c' \in \underline{e}$,

a new reading EC $\langle c', H\rangle$, and perhaps other compound ECs derived from $\langle c', H\rangle$.

We saw in § 7.1.2 that CUNF needs to compute the relations $\|$ and $/\!/$ in order to find possible extensions. The key results for their efficient computation are Prop. 3 and Prop. 4, which give an inductive characterization of the relation $\|$. Based on them, both $\|$ and $/\!/$ can be computed, as we demonstrate below.

Several alternatives for handling $\|$ are available. Roughly speaking, one could transform Prop. 3 and Prop. 4 into a recursive function that, given two ECs, returns whether they are concurrent, recursively calling itself as necessary. However, following the approach of MOLE and PUNF[1], CUNF incrementally computes and *stores in memory* both $\|$ and $/\!/$, instead of resorting to a recursive computation. This requires updating the stored version of the relation as the prefix grows. We detail now how Prop. 3 and Prop. 4 are used to efficiently compute this update. We comment on the recursive computation of $\|$ in Ch. 8.

Let $\mathsf{c}(\rho)$ denote the set of enriched conditions ρ' verifying $\rho \parallel \rho'$. The relation $\|$ is generally sparse and CUNF stores $\mathsf{c}(\rho)$ as a list. However, for the purpose of the following, $\mathsf{c}(\rho)$ could also be a row in a matrix representing the relation $\|$. Once computed, CUNF can check whether $\rho \parallel \rho'$ by either searching for ρ' in $\mathsf{c}(\rho)$ or for ρ in $\mathsf{c}(\rho')$.

For reading and generating enriched conditions ρ, CUNF computes $\mathsf{c}(\rho)$ using Prop. 3 as follows. It initially sets $\mathsf{c}(\rho)$ to $Y_p \cup Y_c$. Next, it computes the intersection of $\mathsf{c}(\rho')$ for all $\rho' \in X_p \cup X_c$, and filters out those $\langle c', H'\rangle$ for which $\underline{{}^\bullet e} \cap H' \not\subseteq H$ holds. In order to compute this condition without actually traversing H and H', we use the sets $r(H)$ and $s(H)$ computed earlier (see above). These are defined as

$$r(H) := \{e' \in H : \underline{e'} \cap \mathit{cut}(H) \neq \emptyset\}$$
$$s(H) := \{e' \in H : e' \in \underline{{}^\bullet e}\}.$$

Then $\underline{{}^\bullet e} \cap H' \not\subseteq H$ holds iff $\underline{{}^\bullet e} \setminus s(H) \cap r(H') \neq \emptyset$, which can be computed traversing $\underline{{}^\bullet e}$ and $s(H)$ one time, and checking $r(H')$ for every ρ'. Note that, while the other steps have their counterparts in Petri net unfoldings, this step is new and specific to c-nets. However, we find that this implementation keeps the overhead very small.

In summary, this allows to compute $\mathsf{c}(\rho)$ for every ρ appended to the prefix. CUNF uses the vector $\mathsf{c}(\rho)$ to additionally store $/\!/$. This is done using the lowest two bits of each pointer in $\mathsf{c}(\rho)$, which are are "abused" to store whether $\rho /\!/ \rho'$ or $\rho' /\!/ \rho$ holds. Computing these two bits requires testing logical conditions of the form $\underline{c} \cap H' \subseteq H$ for any $\rho' \in \mathsf{c}(\rho)$. This test is done similarly to the aforementioned test, it also uses the lists $s(H)$ and $r(H)$.

As for compound conditions ρ built using ρ_1 and ρ_2, CUNF computes $\mathsf{c}(\rho)$ as the intersection of $\mathsf{c}(\rho_1)$ and $\mathsf{c}(\rho_2)$, which relies on Prop. 4.

Certain enriched conditions $\rho = \langle c, H\rangle$ need not to be included in the concurrency relation. It is safe, for instance, to leave $\mathsf{c}(\rho)$ empty if ρ is generating and $f(c)^\bullet \cup \underline{f(c)} = \emptyset$, or if H is a cutoff. We can also avoid computing $\mathsf{c}(\rho)$ if ρ is reading or compound and $f(c)^\bullet = \emptyset$, even if $\underline{f(c)} \neq \emptyset$.

7.1.4 *Splitting the Concurrency Relation*

Let $\rho = \langle c, H\rangle$ be an enriched condition. As mentioned in § 7.1.3, CUNF constructs the list $\mathsf{c}(\rho)$ of enriched conditions ρ' such that $\rho \parallel \rho'$. We found

1 PUNF can actually also use the backwards exploration algorithm explained in § 3.5.

that the performance of the tool benefits greatly in some cases by splitting $\mathsf{c}(\rho)$ into two sets:

$$\mathsf{c}_1(\rho) := \{\langle c, H'\rangle : \rho \parallel \langle c, H'\rangle\}$$
$$\mathsf{c}_2(\rho) := \mathsf{c}(\rho) \setminus \mathsf{c}_1(\rho).$$

In other words, $\mathsf{c}_1(\rho)$ contains the concurrent pairs for the same condition c and $\mathsf{c}_2(\rho)$ the others. This simple split helps in several places. Suppose, for instance, that ρ is a new enriched condition that we have just added to the prefix.

- If ρ is reading or generating (where H is a history for an event e), CUNF uses the algorithm of § 7.1.3 to compute $\mathsf{c}(\rho)$. There, for any ρ' in X_p, any $\langle c', H'\rangle \in \mathsf{c}_1(\rho')$ is such that $c' \in {}^\bullet e$, so all ECs in $\mathsf{c}_1(\rho')$ can be excluded from consideration in the intersection of vectors the algorithm performs.
- Next, in the eager approach, we may use ρ to generate compound conditions. For this, we now simply take all $\rho' = \langle c, H'\rangle$ from $\mathsf{c}_1(\rho)$ and create a new compound condition $\rho'' = \langle c, H \cup H'\rangle$. Moreover, we have that $\mathsf{c}_1(\rho'') = \mathsf{c}_1(\rho) \cap \mathsf{c}_1(\rho')$ and $\mathsf{c}_2(\rho'') = \mathsf{c}_2(\rho) \cap \mathsf{c}_2(\rho')$.
- Finally, CUNF uses new ECs ρ to search for PEs by means of the algorithm in § 7.1.2. In order to find the sets X_p, X_c, CUNF can restrict the search of ECs to $\mathsf{c}_2(\rho)$ rather than $\mathsf{c}(\rho)$ in certain cases.

7.1.5 *Strategies*

Algorithm 1 is parametrized by an strategy, i.e., an order on the finite configurations of the unfolding. Although this order is defined for finite configurations, observe that Algorithm 1, and consequently CUNF, only uses it on histories. CUNF implements three different strategies, and uses by default the ERV strategy. See § A.3.2 to learn how to instruct CUNF to use one particular strategy.

1. *Size strategy*. The order $\prec_M$, defined in (26), was introduced by McMillan [McM93b]. CUNF stores the size of every history in the history graph $\mathcal{H}_\mathcal{E}$, so the implementation of this strategy is obvious.
2. *Parikh strategy*. [EH08, p. 64] CUNF first compares the sizes of both histories, declaring them sorted if they are different. If not, it compares lexicographically the Parikh vectors [EH08] of both configurations. Such vector have also been computed and stored on each history, so at this point CUNF only needs to perform the lexicographic comparison. Recall that this strategy is adequate but not total.
3. *ERV strategy*. [ERV02] This strategy is the order $<_F$ defined in [ERV02] which, as explained in Ch. 3, it is adequate and total for contextual unfoldings. It first compares the configurations with the Parikh strategy. If this suffices to sort them, $<_F$ returns the result. If not, CUNF computes and compares the *Foata normal forms* [ERV02] of both configurations. Unlike the size or Parikh vectors of histories, Foata normal forms are computed every time they are needed. Code profiling over numerous examples shows that the computational cost of doing this is quite low. We note that CUNF implements the order $<_F$ as defined in [ERV02] as well as the slight variation implemented in MOLE, which is also adequate and total, but different from $<_F$.

The size and Parikh vectors of any history H is computed as soon as the enriched event $\langle e, H\rangle$ in which it takes part is discovered as a PE. The

marking $mark(H)$ of the history is also computed at this stage, it will be relevant for deciding whether the PE is a cutoff. The lists $r(H), s(H)$ that we mentioned in § 7.1.3 are also computed at this point during two linear traversals of H.

7.2 PREFIX ANALYSIS: THE TOOL CNA

In Chapter 4, a SAT encoding of the coverability and deadlock-freeness problem has been presented. We implemented this encoding into the tool CNA (Contextual Net Analyzer), distributed within the Cunf Toolset. The manual of this tool is included in App. A.

CNA inputs unfoldings generated with CUNF and searches for reachable markings of the original c-net that enable no transition (deadlocks) or mark a set of given places (coverability). The tool constructs the propositional formulas $\phi_{\mathcal{P}}^{\text{mark},M}$ or $\phi_{\mathcal{P}}^{\text{dead}}$ presented in § 4.3.

We briefly recall the structure of $\phi_{\mathcal{P}}^{\text{mark},M}$ and $\phi_{\mathcal{P}}^{\text{dead}}$. Both formulas request that satisfying assignments represent configurations of the unfolding, which each of them additionally constraints to be those who cover the places in M, for $\phi_{\mathcal{P}}^{\text{mark},M}$, or those reaching a deadlocked marking, for $\phi_{\mathcal{P}}^{\text{dead}}$. In encoding configurations, multiple *at-most-one* constraints are used to forbid symmetric conflicts, and one acyclicity constraint is employed to forbid cycles of asymmetric conflict.

Satisfying assignments to these formulas encode the (offending) firing sequences searched by the tool. The tool relies on MINISAT [ES03] to solve the formula, and displays a firing sequence of the original c-net if one is found. Notice that once the unfolding is built, it can serve to answer multiple queries.

A number of optimizations and variants of the SAT encodings were proposed in § 4.3. In this section, we empirically evaluate their impact on the solving time. We have employed as benchmarks the set of safe nets referred in § 7.3.

7.2.1 *Stubborn Event Elimination and Subset Reduction*

Over the set of ordinary nets considered in § 7.5, we found that removing stubborn events reduces the accumulated SAT solving time by 27%. When applied together with the subset optimization from § 4.7, this grows to 30%. For c-nets, we measured a 14% reduction when stubborn events are removed from the encoding *without* acyclicity constraints but only a 6% reduction if additionally the subset optimizations are applied. Experiments over the encoding *with* acyclicity were similar.

These experiments suggest that removal of stubborn events has a positive impact on performance, while subset optimization has very limited, even negative impact. For the following experiments, we applied only the stubborn event optimization.

7.2.2 *AMO Constraint*

We briefly discuss the *At-Most-One* (AMO) constraint, used in $\phi^{\text{sym}}_{\mathcal{P}}$, in Ch. 4 but also in $\gamma^{\text{asym}}_{\mathcal{Q}}$, in Ch. 5. The constraint $\text{AMO}(x_1, \ldots, x_n)$ can be trivially implemented by

$$\bigwedge_{1 \le i < j \le n} (\neg x_i \vee \neg x_j).$$

However, this *pairwise encoding* is quadratic, and the SAT performance suffered for examples with large conflict sets when this was used.

A survey of better encodings can be found in [Che10]. CNA employs a *k-tree* encoding, that introduces $\mathcal{O}(n)$ additional variables and adds $\mathcal{O}(n)$ clauses. The k-tree encoding on variables $x_1, \ldots, x_n$ is an AMO constraint denoted by $T^k_y(x_1, \ldots, x_n)$ that we specify here for $k = 2$:

$$T^2_y(x_1, \ldots, x_n) := \begin{cases} x_1 \to y & \text{if } n = 1 \\ T^2_{y_1}(x_1, \ldots, x_{\lceil \frac{n}{2} \rceil}) \wedge T^2_{y_2}(x_{\lceil \frac{n}{2} \rceil + 1}, \ldots, x_n) \wedge & \text{otherwise} \\ (\neg y_1 \vee \neg y_2) \wedge (y_1 \to y) \wedge (y_2 \to y) & \end{cases}$$

Here, y, y_1, and y_2 are new variables. The k-tree constraint $T^k_y(x_1, \ldots, x_n)$ is satisfied by assignments that either satisfy no x_i or satisfy one x_i and also y, for $1 \le i \le n$. It

1. partitions the n variables in k groups,
2. constrains recursively each group,
3. ensures that at most one group has one satisfied variable (by means of the associated new y_i variables, for $1 \le i \le k$), and
4. sets y if any group has one satisfied variable.

Constraining the k new variables y_i can be done by means of the pairwise encoding or by a new k'-tree encoding (with $k' < k$). For $k = 2$, this encoding can be optimized to yield $3n - 5$ clauses and $n - 1$ new variables.

The base case, when $n = 1$, produces one clause with two literals; the inductive case produces three with two literals each. Because the encoding produces $\mathcal{O}(n)$ clauses, with two literals each, the number of literals generated by the encoding is also $\mathcal{O}(n)$.

We observed an overall improvement when replacing the pairwise with the k-tree encoding. The accumulated SAT solving time on our benchmarks under values of $k = 2, \ldots, 8$ was minimal for $k = 4$. Experiments over c-nets on the encoding suggested $k = 4$ as a good candidate, as well. We therefore used 4-tree encodings in $\phi^{\text{sym}}_{\mathcal{P}}$ for the following experiments.

7.2.3 *Acyclicity Checking*

Section 4.4 explained that $\phi^{\text{asym}}_{\mathcal{P}}$ encodes cycle-freeness of configuration $\mathcal{C}$ w.r.t. the relation $R = <_i \cup \nearrow$. We investigated three encodings suggested in [CGS09]: transitive closure, unary ranks, and binary ranks. The latter clearly outperformed the others. In the binary rank encoding, every event is associated with a rank, i.e., an integer up to some bound r, that is represented by $\lceil \log_2 r \rceil$ Boolean variables. A number of constraints ensure that the rank of event e is less than the rank of event f if $(e, f) \in R$. If n is the number of events in $\mathcal{P}$, the resulting SAT encoding is of size $\mathcal{O}(n \log n)$ if ranks are encoded in binary.

Moreover, § 4.4 proposed a method to reduce the size of R. Table 5 shows the size of the direct asymmetric conflict relation before and after this reduc-

Table 5: Reduction of the asymmetric-conflict relation.

	Before		After		Ratio after/before	
Net	Nodes	Edges	Nodes	Edges	Nodes	Edges
bds_1.sync	192	271	27	52	0.14	0.19
bds_1.fsa	66	89	9	16	0.14	0.18
byzagr4_1b	3197	64501	2348	61088	0.73	0.95
dme7	2541	9856	1571	8557	0.62	0.87
dme8	3648	15232	2179	13314	0.60	0.87
dme9	5031	22572	2936	19908	0.58	0.88
dme10	6720	32320	3836	28726	0.57	0.89
dme11	8745	44968	4918	40301	0.56	0.90
key_3.sync	3	3	2	2	0.67	0.67
q_1.sync	189	4095	126	4032	0.67	0.98
rw_12.sync	3	3	2	2	0.67	0.67
rw_2w1r	1766	8877	915	7447	0.52	0.84

tion for six c-nets unfoldings with at least one cycle in R. More precisely, we show the size of the largest SCC, in most examples there is in fact only one non-trivial SCC.

In average, the proposed method eliminates 66% of the nodes and 26% of the edges, seeming thus to be more effective at reducing the number of nodes rather than the number of edges, which in turn becomes a reduction in the number of variables rather than the number of clauses of the encoding.

However, in some examples, the remaining SCCs are still rather large, on the order of tens of thousands of events, and in these cases $\phi_{\mathcal{P}}^{\text{asym}}$ negatively impacts the solving time of the formula $\phi_{\mathcal{P}}^{\text{mark},M}$ or $\phi_{\mathcal{P}}^{\text{dead}}$ where it is included. We therefore implemented a two-stage approach. First we generate and solve all subformulas of $\phi_{\mathcal{P}}^{\text{mark},M}$ or $\phi_{\mathcal{P}}^{\text{dead}}$ except $\phi_{\mathcal{P}}^{\text{asym}}$, which is entirely omitted. This may of course produce a satisfying assignment that is not a configuration, but if the formula is already unsatisfiable, then $\phi_{\mathcal{P}}^{\text{mark},M}$ or $\phi_{\mathcal{P}}^{\text{dead}}$ would also be unsatisfiable if $\phi_{\mathcal{P}}^{\text{asym}}$ had been included. Only when this first stage yields a false positive, a second stage with $\phi_{\mathcal{P}}^{\text{asym}}$ is used to obtain a definitive result. This approach was very successful: in over 100 different nets from various sources that we tried, only 2 (small) nets yielded a false positive. The experiments presented in the following use this two-stage approach.

7.2.4 *SAT-solver Settings*

MINISAT allows the user to tweak certain aspects of the SAT-solving algorithm. It is tempting to do so in order to exploit knowledge about the problem domain.

In $\phi_{\mathcal{P}}^{\text{dead}}$ for instance, the chosen configuration $\mathcal{C}$ determines the marking m. The subformula $\phi_{\mathcal{P}}^{\text{dis}}$ introduces one variable for each place of the net. MINISAT can be instructed to search a solution by actively choosing values only for a subset of variables, the so-called *decision variables*. By default, all variables are decision variables. In $\phi_{\mathcal{P}}^{\text{dead}}$ it is safe to exclude variables associated to places from being decision variables, as their value is safely

propagated from the values to variables for events. We tried removing these variables from being decision variables, however the effect on the running time was negative overall.

Similarly, we tried to exploit the causal structure of $\mathcal{P}$ by instructing MINISAT to prefer deciding the values of events with few (resp. many) causal predecessors first. This, too, tended to impact the solving time negatively.

Thirdly, we measured the impact of choosing the polarity of the variables (i.e., whether MINISAT first tries setting them to true or false). While this had a positive impact on certain examples, other examples were very negatively affected. Overall, the default settings of MINISAT proved to be very good and did not benefit from our adjustments.

7.3 BENCHMARKS

A number of experiments are performed on the next three sections, all of them over a set of benchmarks that are discussed now.

We have employed a set of 1-safe nets that have previously served as benchmarks in the literature on Petri net unfoldings, e.g. [McM93b; MR97; Hel99c; Kho03; Sch06]. Most of the examples were originally collected by J. C. Corbett [Cor96], but the set also includes nets contributed by K. McMillan [McM93b; McM95b], S. Melzer and S. Römer [MR97], S. Merkel [Mer97], and M. Heiner, P. Deussen, K. Schmidt, C. Schröter (see [EH01]). In all preceding bibliographic references, and specially in [Cor96; MR97; Kho03], detailed explanations about the models are offered, so we will not reproduce them here again. See also [Bes96] for additional information.

These nets are not specifically geared towards using the contextual unfolding approach, though read arcs occur naturally here, as we explain below. They were already favourable for unfoldings, and have various characteristics that allowed to test many aspects of our implementations, thus ensuring their robustness.

For each net N in the set, we first obtained the c-net N' by substituting pairs of arcs (p, t) and (t, p) in N by read arcs. Evidently, the plain encoding of N' is N. Secondly, we obtained the PR-encoding N'' of N'.

Some benchmarks have also been contributed by the author, although not all of them are used for experiments reported in this manuscript. Most of the nets are in fact parametric collections of nets, generated by an script included in the distribution of the Cunf Toolset, see App. A. The parametric collections include *Dijkstra's mutual exclusion algorithm* [Dij65], described in detail in § 7.7, *Asynchronous Conway's game of life* [Gar70], on a square grid of parametric dimension, and *Asynchronous Random Boolean Networks*, a generalization of random Boolean networks [Ger04]. The last two collections have not been used in this manuscript.

7.4 EXPERIMENTS WITH CUNF

In order to experimentally evaluate the tool CUNF, we performed a series of experiments with the benchmarks of § 7.3. Those put to test many features of the implementation, thus improving the robustness of the tool. We were interested in the following questions:

- Is the contextual unfolding procedure efficient?
- What is the size of the unfoldings, compared to Petri net unfoldings?

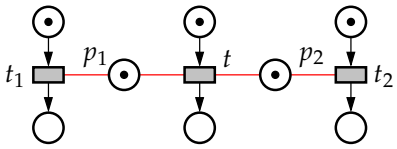


Figure 31: Pairs of independent readers

- How do the various approaches (lazy, eager, PR, plain encoding) compare?

Concerning the second and third point, it is worth noting that we could contrive examples to show arbitrarily large differences between various approaches. As far as the size of the final unfolding is concerned, Fig. 5, on p. 9 already shows that contextual unfoldings may be up to exponentially more succinct than Petri net unfoldings. As far as running time is concerned, § 3.10 contains examples that would distinguish the eager and the lazy approach in both senses.

To see how the running time of the contextual approaches can be superior to the plain encoding, consider the net in Fig. 31, where transition t reads from two places p_1 and p_2. Both places have an additional reading transition, so they each have one (empty) generating history, two reading histories, and one compound history. The contextual unfolding is isomorphic to the net itself. If one expands the context of transition t to k places like p_1 and p_2, then the contextual approaches produce the prefix in time linear to k. The plain encoding, on the other hand, will create an exponential number of events for t, each corresponding to some set of transitions that have previously read from $\underline{t}$.

In order to abstract from such artefacts and get numbers from more realistic examples, we considered the benchmarks referred in § 7.3. Recall that these nets are not specifically geared towards using contextual approaches, although they had various characteristics that allowed to test many aspects of the implementation.

Let N be any c-net in the benchmark, N_p its plain encoding and N_r its place-replication encoding. We first compared Mole [Sch] and Cunf on the nets N_p and N_r, which are ordinary nets, without read arcs. The object of this exercise was to establish whether Cunf was working reasonably efficient on known examples. Indeed, its running times were always within 70% and 140% of those of Mole, the differences due to minor implementation choices. To abstract from these details, we used Cunf for all further comparisons.

We then used Cunf to produce marking-complete prefixes of N, N_p, and N_r, using both lazy and eager methods and the order $\prec_F$ from [ERV02]. Table 6 summarizes the results.[2]

The columns in the table are subdivided into three parts, corresponding to the contextual net, its PR-encoding, and its plain encoding. For contextual nets, we first give the number of events and conditions contained in the marking-complete prefix (columns $|E|$ and $|B|$, in thousands). These prefixes include cutoff events, which means that the number for $|E|$ is actually somewhat larger than what is strictly required by Rmk. 8.

The column marked $|E_{cut}|$ provides the percentage of such events; this percentage could be subtracted from $|E|$ to obtain the prefix that is strictly necessary to characterize, e.g., coverability as in Ch. 4. Cunf does need to

2 Experiments performed using Cunf v.1.4, compiled with gcc 4.4.5. Our machine has twelve 64bit Intel Xeon CPUs, running at 2.67GHz, 50GB RAM and executes Linux 2.6.32-5.

compute those cutoff events anyway, so their inclusion in the output prefix hardly affects the running time.

The column $|{}^\bullet e|$ gives the average preset size of an event. The four columns mentioned so far are identical for the eager and the lazy approach. For the eager approach, we then list the number of compound conditions (causing additional memory overhead of the eager approach w.r.t. lazy) and the running time[3] in seconds (t_E) as well as (maximum virtual) memory consumption in megabytes (m_E). For lazy, we list running times (t_L) and memory consumption (m_L) *relative to the eager approach*, i.e., a factor less than 1 means a faster/less memory-consuming computation, and a factor larger than 1 a slower/more memory-consuming one.

For the PR-encoding and the plain encoding, the data for number of events and conditions, running times, memory consumption, and average preset size (only PR) is also given *relative to the eager approach*. We additionally provide the percentage of cutoff events ($|E_{cut}|$). Notice that the number of *enriched* events in lazy and eager equals the number of events in PR, cf. the discussion on § 1.4. The ratio between number of events in contextual and number of events in PR is thus the average number of histories per event in the contextual approach. We make the following observations:

- We first look at the comparison between lazy and eager. It turns out that in this set of benchmarks, many examples did not exhibit any compound conditions (despite the presence of many read arcs), e.g., because reading actions took place sequentially, or multiple potential readers happened to be in conflict with one another. In those examples, the differences between the two versions are due to the different implementations of the possible extensions (see § 3.7) and the various relations that must be maintained (see § 3.9), sometimes slightly favouring one approach, sometimes the other.
  Significant differences arise where (like in `key_4`) there are many compound conditions; here lazy has some memory savings but performs very badly. An effect to the contrary like in Fig. 17 (b), while in principle possible, did not manifest itself in our benchmarks.
- Compared with PR, the eager approach is consistently more efficient. In several cases (such as `elevator_4` or `rw_2w1r`), PR is orders of magnitudes slower. This clear tendency is slightly surprising given that the enriched contextual prefix has essentially the same size as the prefix of the PR-encoding. We experimentally traced the difference to the enlarged presets of certain transitions in the PR-encoding (see Fig. 9 in p. 21), causing combinatorial overhead and increasing the number of conditions in the concurrency relation.[4] Indeed, high running times for PR seem to coincide with high numbers in the $|{}^\bullet e|$ column for PR (recall that this number is relative to the one for contextual).
- Both the eager approach and the plain unfolding handle all examples gracefully. The factors of the running times are between 0.7 and 4.2, meaning eager is between 40% slower and 4 times faster w.r.t. plain. The prefixes produced by the contextual unfolding methods

3 Actually, the *CPU time*, as reported by the Unix `time` command.

4 Another interesting point here concerns the theoretical compexity of computing possible extensions. Deciding whether a given contextual prefix $\mathcal{P}$ can be extended with an occurrence of transition t is NP-complete. The algorithm employed in CUNF, see § 7.1.2, is exponential in $|{}^\bullet t \cup \underline{t}|$ and polynomial in $|\mathcal{P}|$. Thus larger presets should be accompained by exponentially slower PE computations. However, we did not observe this theoretical explosion, and so, we conclude that it must have remained small in comparison to the overhead of computing a larger concurrency relation.

Table 6: Experimental results. Data for the eager approach are absolute numbers, whereas for lazy, place-replication, and plain unfolding the ratio w.r.t. eager is given; see the text for more information.

Benchmark	Contextual unfolding									PR unfolding						Plain unfolding				
					Eager			Lazy												
Name	$\|E\|$	$\|B\|$	$\|E_{\text{cut}}\|$	$\|{}^\bullet e\|$	comp.	t_E	m_E	t_L	m_L	$\|E\|$	$\|B\|$	$\|E_{\text{cut}}\|$	$\|{}^\bullet e\|$	t_R	m_R	$\|E\|$	$\|B\|$	$\|E_{\text{cut}}\|$	t_P	m_P
bds_1.sync	1.8k	2.8k	40%	1.5	0	0.11	17	1.5	0.9	2.3	4.8	40%	2.3	2.1	1.9	7.0	13.5	67%	4.2	2.5
byzagr4_1b	8.0k	17.6k	4%	2.9	72	2.60	163	1.4	1.0	1.0	1.4	4%	1.9	3.4	1.5	1.8	2.4	5%	1.4	1.5
dpd_7.sync	10.4k	21.4k	25%	2.1	0	0.98	59	1.2	1.0	1.0	1.4	25%	1.4	1.2	1.4	1.0	1.4	25%	1.0	1.2
elevator_4	16.9k	28.6k	44%	1.7	0	1.24	71	1.1	0.9	1.0	13.7	44%	22.4	358	66.3	1.0	1.7	44%	1.8	1.1
ftp_1.sync	50.9k	96.6k	26%	1.9	0	24.81	296	0.6	1.0	1.0	1.6	26%	1.6	2.5	6.3	1.8	2.8	37%	2.0	1.9
furnace_4	94.4k	147.4k	68%	1.6	0	19.04	266	0.7	0.9	1.1	1.9	70%	1.8	1.7	2.1	1.2	1.8	69%	0.9	1.4
key_4	4.8k	7.9k	16%	1.7	23.3k	1.58	110	711	0.4	4.6	5.5	19%	13.4	6.4	0.7	14.6	17.7	46%	0.8	1.1
mmgt_4.fsa	46.9k	92.1k	45%	2.0	0	0.97	70	1.1	1.0	1.0	1.0	45%	1.0	1.0	1.0	1.0	1.0	45%	0.9	1.0
q_1.sync	10.7k	20.6k	13%	1.9	0	1.36	77	1.0	0.9	1.0	1.5	13%	1.5	2.2	2.1	1.0	1.5	13%	1.0	1.1
rw_12.sync	98.4k	196.8k	92%	2.0	0	3.35	171	1.2	1.0	1.0	1.5	92%	1.5	3.4	2.5	1.0	1.5	92%	1.5	1.0
rw_1w3r	14.5k	24.2k	32%	1.7	191	0.27	36	1.6	1.0	1.0	1.7	34%	1.7	2.0	1.6	1.1	1.2	34%	0.8	0.9
dme11	9.2k	16.7k	1%	1.8	0	7.89	516	0.9	1.0	1.0	1.9	1%	1.9	1.0	0.9	1.0	1.9	1%	0.8	0.9
rw_2w1r	9.4k	15.3k	15%	1.6	0	0.23	31	1.0	1.0	1.0	4.2	15%	4.7	61.9	6.4	1.0	1.2	15%	0.7	0.9

are smaller than in the plain approach in half the cases. Interestingly, these are not always the same as those on which they run faster: for `elevator_4` and `rw_12.sync`, the same number of events is produced more quickly. Here, the read arcs are arranged in such a way that each event still has only one history; the time saving comes from the fact that the contextual approach produces fewer *conditions* and hence a smaller concurrency relation. For `key_4` and `rw_1w3r`, the contextual methods produce smaller unfoldings but take longer to run, due some overhead in the computation of the // relation.

To summarize, this set of benchmarks contained examples where lazy and PR performed badly, whereas eager and plain handled all cases gracefully. The eager approach was the fastest overall, and for all examples its running time was within factor 2 of the fastest approach for that example. The prefixes produced by the contextual methods can be significantly smaller than for their Petri net encodings, which make them suitable candidates for subsequent analysis methods, such as those in Ch. 4.

7.5 EXPERIMENTS WITH CNA

The tool CNA, presented in § 7.2, implements the techniques for deadlock and coverability checking of Ch. 4. This section experimentally compares the efficiency of the tool with that of other well-established deadlock-checking methods based on unfoldings. The effects of using ordinary nets and c-net are evaluated over the benchmark discussed in § 7.3.

In [KK07], Khomenko and Koutny compared three versions of their deadlock checking method, implemented in the tool CLP, against the methods by McMillan [McM93b], Melzer and Römer [MR97], and Heljanko [Hel99c]. In their benchmarks, the first version of their algorithm[5] outperformed the other methods on almost all examples. We experimentally confirmed this conclusion. Moreover, we learnt of an unpublished SAT-based tool by V. Khomenko which is said to be slower than CLP.[6] We therefore compare our technique with the first method of CLP.[7]

Table 7 presents the results on the aforementioned standard suite. We used MOLE [Sch] to produce finite complete prefixes of the Petri nets and CUNF to do the same for c-nets.[8] The running times for MOLE and CUNF are given in the respective columns, together with the number of events $|E|$ and conditions $|B|$ of the generated prefix. In ordinary Petri nets, the column labelled CLP shows the running times of CLP on the unfolding prefix constructed with MOLE. For both ordinary and contextual nets, the column labelled SAT provides the running times of MINISAT on the formulas produced by CNA, using the settings discussed in § 7.2. Times are given in seconds, and represent averages over 10 runs; this was essential to obtain running times that reasonably repeatable.

We do not provide the translation times to generate linear equation systems (for CLP) or SAT formulas (for MINISAT). Those times would not be very representative since both translators are potentially suboptimal. The constraints generated by CNA are of linear size w.r.t. the unfolding prefix,

5 Column *std* in Tables 1 and 2 in [KK07].

6 According to the author, V. Khomenko.

7 All experiments have been performed using CUNF and CNA v.1.4, MOLE v.1.0.6, both compiled with gcc 4.4.5, version 301 of CLP, and MINISAT v.2.2.0. Our machine has twelve 64bit Intel Xeon CPUs, running at 2.67GHz, 50GB RAM and executes Linux 2.6.32-5.

8 The running times of MOLE and CUNF are comparable on Petri nets, but MOLE produces prefixes in a format suitable for CLP.

Table 7: Comparison of deadlock-checking methods; the Res(ult) is L(ive) or D(ead)

Benchmark		Petri net unfoldings					C-net unfoldings			
		Mole			Clp	SAT	Cunf			SAT
Name	Result	Time	$\|E\|$	$\|B\|$	Time	Time	Time	$\|E\|$	$\|B\|$	Time
bds_1.sync	L	0.58	12900	37306	0.04	0.01	0.14	1830	2771	<0.01
byzagr4_1b	L	3.71	14724	42276	0.53	0.26	3.25	8044	17603	0.19
dme11	L	6.56	9185	31186	0.60	0.28	10.86	9185	16710	0.25
dpd_7.sync	L	1.21	10354	29939	0.10	0.18	1.09	10354	21359	0.02
ftp_1.sync	L	45.37	91730	275099	1.13	0.38	26.85	50928	96617	0.05
furnace_4	L	37.44	114477	264823	1.29	0.19	19.11	94413	147438	0.12
rw_12.sync	L	3.95	98361	295152	0.08	0.02	3.96	98361	196796	0.02
rw_1w3r	L	0.30	15432	28207	0.11	0.22	0.36	14521	24174	0.40
rw_2w1r	L	0.22	9363	18575	0.04	0.34	0.32	9363	15304	0.58
elevator_4	D	2.58	16856	47743	0.24	0.03	1.51	16856	28593	0.06
key_4	D	1.68	69600	139206	0.07	0.08	2.07	4754	7862	<0.01
mmgt_4.fsa	D	1.16	46902	92940	0.02	0.04	1.17	46902	92076	0.05
q_1.sync	D	1.76	10716	30087	<0.01	0.02	1.54	10716	20567	0.01
$\sum$		106.52			4.25	2.05	72.23			1.75

except for the acyclicity constraint, which is of size $\mathcal{O}(n \log n)$. Optimized times for CNA should arguably be of fractions of a second.

Compared to CLP, SAT checking performs well over Petri nets, solving the problems twice as fast on aggregate. Concerning the comparison of SAT checking between Petri nets and c-nets, another advantage of 13% for deadlock verification is obtained. More significantly, the time for generating c-net unfoldings is 30% less than for Petri nets. This advantage is not huge, but recall that these benchmarks are already favourable examples for Petri net unfoldings and were not specifically designed to exploit the advantages of c-nets. Contextual unfolding construction and verification performs very well, for instance, on `ftp_1.sync` and `furnace_4.sync`, where unfoldings run twice faster and deadlock checking one order of magnitude faster.

The two-stage approach was essential for performance: while the acyclic constraints had a big impact only on a few examples (notably `byzagr4_1b`, `dme`, and `rw*`), that effect would have nullified the advantage of smaller unfoldings.

7.6 EXPERIMENTS WITH CONTEXTUAL MERGED PROCESSES

This section reports on experimental results comparing the sizes of CMPs, merged processes (MPs), and complete unfolding prefixes for a number of benchmarks from § 7.3. All these methods construct a representation of the state space of the net. The goal here is comparing the relative compression factors of these various representations. We do not yet have an implementation of the direct construction algorithm for CMPs, so we cannot compare at this stage the times to construct them.

To elaborate the comparison, unfoldings and merged processes were constructed.[9] For every net in the benchmark, we considered the equivalent c-net and the PR-encoding of the c-net, constructed as explained in § 7.3. We then constructed the unfoldings of the three nets, and merged them into the corresponding merged processes. The following consistent setup was used to produce both unfoldings and MPs.

- The total adequate strategy proposed in [KM11] was used to construct unfoldings. This strategy is not implemented in CUNF, so other tools were used to construct contextual unfoldings, as explained below.
- All configurations were allowed as cutoff correspondents, not only the local ones. When constructing ordinary unfoldings, usually only the local configurations are authorized to be the corresponding configurations. Similarly, in § 3.2, the $\prec$-cutoff enriched events in Def. 4 on p. 34 were defined using only histories as corresponding configurations. The idea of using general corresponding configurations originates from [Hel99b]. We are aware of only one unfolder, PUNF [Kho], that implements the approach[10], which requires to integrate a SAT solver in the unfolder. However, in existing direct construction methods for MPs, the SAT solver is already necessary for other purposes, cf. Ch. 5, so using general configurations is rather natural and entails no performance penalty. To make the comparison fair to merged processes, we thus employ non-local corresponding configurations.
- The cutoff (mp-)events and post-cutoff (mp-)conditions has not been counted.

9 All the benchmarks and tools referenced in this section are publicly available from `http://www.lsv.ens-cachan.fr/~rodriguez/experiments/pn2013/`.

10 Together with the implementation of [Hel99b], which works after unfolding.

This setup has been used to facilitate the comparison with future experiments on the direct construction algorithm of CMPs, and to avoid similar problems in comparing experimental results as those explained in [KM11].

The marking-complete unfolding prefixes of the plain and PR encodings have been constructed using PUNF [Kho]. The plain and PR MPs have been merged from the corresponding unfolding prefixes with MCI2MP, a tool developed by V. Khomenko. Similarly, CMPs were merged from the corresponding contextual unfolding prefixes using CMERGE, a tool developed by the author that essentially implements the merging operation of Def. 16 on events and conditions. The direct construction algorithms for MPs and CMPs would yield the same results.

Contextual prefixes could not be computed with CUNF, so we constructed them by compressing the PR ones with Stefan Schwoon's tool PRCOMPRESS. The tool a marking-complete contextual prefix $\mathcal{P}$ of a c-net N out of a marking-complete prefix $\mathcal{P}'$ of the PR encoding N_r. It applies a *folding* operation to $\mathcal{P}'$, defined by the repeated iteration of the following steps:

1. All conditions that were created due to a consume-produce loop are merged and their flow arcs replaced by a read arc;
2. The PR encoding N_r contains multiple replicas of certain places in N. The tool next merges all conditions in the postset of any event if all them are labelled with those replicas. More specifically, any set of conditions is merged if, (*i*) it is contained in the postset of an arbitrary event, and (*ii*) the conditions in the set are labelled by all place replicas of any place in N. The resulting new conditions is labelled by the place of N.
3. All events with the same label and the same preset are merged, and so are their postsets.

The resulting c-net prefix $\mathcal{P}$ has the same reachable markings as $\mathcal{P}'$ and is therefore marking-complete. Indeed, applying this operation to the prefix shown in Fig. 5 (f), which is the unfolding of Fig. 5 (c), would yield the c-net unfolding Fig. 5 (e).

Recall the following theoretical guarantees:

- The c-net unfolding prefix is never larger than the PR prefix.
- The plain/PR/contextual MP is never larger than the corresponding unfolding prefix.

Table 8 compares the sizes of plain, PR and contextual unfolding prefixes and MPs. The 4^{th} and 5^{th} columns from the left are, respectively, the number of read arcs in the net and place replicas in its PR encoding.[11] The numbers of conditions and events for the plain and PR unfoldings are normalised w.r.t. that of the contextual unfolding. Similarly, mp-conditions and mp-events of the plain and PR MPs are normalised w.r.t. those of the CMPs. The last three columns show the compression gains of CMPs w.r.t. plain and contextual unfolding prefixes, and the gain of plain MPs w.r.t. plain unfolding prefixes.

One can see that CMPs are the most compact of all the considered representations.[12] Comparing the three columns on MPs, on certain examples, such as DME and RW, the reduction in the number of mp-conditions is substantial, while the number of mp-events remains the same. Furthermore, on some benchmarks, notably KEY(4), CMPs have significant advantages over

11 More precisely, $\sum_{p \in P, |\underline{p}|>1}(|\underline{p}|-1)$.

12 Though the PR MP of RW(1,2) has four mp-events fewer, it has many more mp-conditions.

Table 8: Comparing unfoldings, MPs, and CMPs.

Benchmark					Unfolding						Merged process						Gains		
Name	Net $\lvert P\rvert$	$\lvert T\rvert$	Stats. $\lvert C\rvert$	Repl.	Plain $\lvert B\rvert$	$\lvert E\rvert^{(a)}$	PR $\lvert B\rvert$	$\lvert E\rvert$	Ctx $\lvert B\rvert$	$\lvert E\rvert^{(b)}$	Plain $\lvert\widehat{B}\rvert$	$\lvert\widehat{E}\rvert^{(c)}$	PR $\lvert\widehat{B}\rvert$	$\lvert\widehat{E}\rvert$	Ctx $\lvert\widehat{B}\rvert$	$\lvert\widehat{E}\rvert^{(d)}$	a/d	b/d	a/c
Bds	53	59	24	15	4.50	3.79	2.60	2.14	424	252	1.11	1.14	1.36	1.07	70	44	21.73	5.73	19.12
Brujin	86	165	158	142	2.84	1.97	7.00	1.70	286	208	1.33	1.44	3.83	1.31	115	127	3.22	1.64	2.23
Byz	504	409	376	284	2.38	1.80	1.41	1.00	17019	7748	1.23	1.03	1.85	1.22	529	303	46.11	25.57	44.78
Eisenbahn	44	44	6	3	2.19	2.15	2.22	2.04	99	53	1.13	1.30	1.22	1.19	69	43	2.65	1.23	2.04
Ftp	176	529	39	33	1.06	1.04	1.02	1.00	73516	37540	1.02	1.05	1.19	1.00	254	455	85.74	82.51	81.61
Knuth	78	137	114	98	2.62	1.81	5.42	1.62	247	178	1.32	1.31	3.25	1.27	102	112	2.88	1.59	2.20
Mutual	49	41	12	4	1.41	1.23	1.51	1.23	187	121	1.12	1.26	1.23	1.27	92	73	2.04	1.66	1.62
Dme(2)	135	98	132	0	1.61	1.00	1.61	1.00	293	118	1.55	1.04	1.55	1.04	195	90	1.31	1.31	1.26
Dme(4)	269	196	264	0	1.73	1.00	1.73	1.00	1337	636	1.56	1.04	1.56	1.04	389	180	3.53	3.53	3.38
Dme(6)	403	294	396	0	1.79	1.00	1.79	1.00	3517	1794	1.56	1.04	1.56	1.04	583	270	6.64	6.64	6.36
Dme(8)	537	392	528	0	1.83	1.00	1.83	1.00	7217	3832	1.56	1.04	1.56	1.04	777	360	10.64	10.64	10.19
Dme(10)	671	490	660	0	1.86	1.00	1.86	1.00	12821	6990	1.56	1.04	1.56	1.04	971	450	15.53	15.53	14.87
Elev(1)	63	99	18	11	1.02	1.00	1.34	1.00	152	85	1.02	1.00	1.38	1.00	56	46	1.85	1.85	1.85
Elev(2)	146	299	59	47	1.03	1.00	1.85	1.00	858	477	1.01	1.00	2.02	1.00	125	135	3.53	3.53	3.53
Elev(3)	327	783	160	141	1.03	1.00	2.66	1.00	4050	2241	1.00	1.00	2.90	1.00	263	346	6.48	6.48	6.48
Elev(4)	736	1939	405	375	1.04	1.00	4.08	1.00	17360	9567	1.00	1.00	3.98	1.00	556	841	11.38	11.38	11.38
Furn(1)	27	37	9	5	1.12	1.04	1.18	1.00	130	72	1.09	1.03	1.23	1.00	43	33	2.27	2.18	2.21
Furn(2)	40	65	11	6	1.14	1.06	1.14	1.00	582	324	1.07	1.05	1.19	1.02	75	94	3.66	3.45	3.47
Furn(3)	53	99	13	7	1.14	1.08	1.12	1.00	2265	1250	1.07	1.02	1.17	1.01	106	221	6.09	5.66	5.95
Key(2)	94	92	32	31	2.74	2.16	1.72	1.27	302	191	1.22	2.50	1.59	1.17	112	105	3.92	1.82	1.57
Key(3)	129	133	48	47	5.81	4.60	2.81	2.19	1276	806	1.21	4.13	1.64	2.34	151	186	19.93	4.33	4.83
Key(4)	164	174	64	63	11.37	9.08	5.56	4.43	5806	3637	1.21	5.26	1.67	9.92	190	290	113.82	12.54	21.63
Mmgt(1)	50	58	1	0	1.00	1.00	1.00	1.00	79	38	1.00	1.00	1.00	1.00	57	38	1.00	1.00	1.00
Mmgt(2)	86	114	2	0	1.00	1.00	1.00	1.00	502	250	1.00	1.00	1.00	1.00	99	155	1.61	1.61	1.61
Mmgt(3)	122	172	3	0	1.00	1.00	1.00	1.00	2849	1424	1.00	1.00	1.00	1.00	141	355	4.01	4.01	4.01
Mmgt(4)	158	232	4	0	1.00	1.00	1.00	1.00	14900	7450	1.00	1.00	1.00	1.00	183	638	11.68	11.68	11.68
RW(1,1)	84	208	123	75	1.23	1.00	1.78	1.00	142	94	1.26	1.00	2.03	1.00	77	65	1.45	1.45	1.45
RW(2,1)	72	88	27	16	1.19	1.01	1.30	1.00	845	554	1.26	1.01	1.45	1.00	113	165	3.39	3.36	3.37
RW(3,1)	106	270	129	81	1.19	1.04	1.60	1.00	5100	3376	1.24	1.02	1.93	1.00	160	368	9.54	9.17	9.39
RW(1,2)	209	1482	1132	717	1.21	1.00	3.36	1.00	2836	1838	1.35	1.01	4.76	0.99	159	371	4.95	4.95	4.90
Sentest(25)	104	55	8	5	1.32	1.29	1.35	1.29	188	104	1.06	1.11	1.10	1.11	107	55	2.44	1.89	2.20
Sentest(50)	179	80	8	5	1.23	1.23	1.25	1.23	263	129	1.03	1.08	1.06	1.08	182	80	1.99	1.61	1.85
Sentest(75)	254	105	8	5	1.18	1.19	1.19	1.19	338	154	1.02	1.06	1.04	1.06	257	105	1.75	1.47	1.66
Sentest(100)	329	130	8	5	1.15	1.17	1.16	1.17	413	179	1.02	1.05	1.03	1.05	332	130	1.61	1.38	1.54

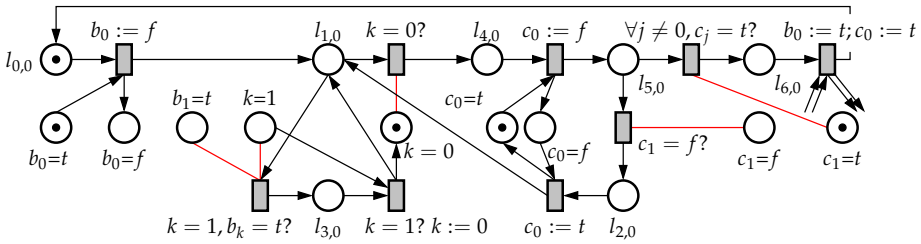


Figure 32: The fragment of 2-Dijkstra that encodes thread 0. Note that arrows from transition $b_0 := t; c_0 := t$ are only partially depicted.

both plain and PR MPs. Interestingly, in this case the PR MP is significantly larger than even the plain MP, which seems to be due to place replication making the subsequent merging much less efficient. As CMPs do not suffer from this problem, they come as a clear winner in such cases.

7.7 A CASE STUDY: DIJKSTRA'S MUTUAL EXCLUSION ALGORITHM

In this section we analyze the performance of merged processes and unfoldings of contextual nets on a well-known concurrent algorithm for mutual exclusion due to Dijkstra [Dij65]. We start with a condensed technical explanation of the algorithm, see [Dij65] for more details.

Dijkstra's algorithm allows n threads to ensure that no two of them are simultaneously in a critical section. Two Boolean arrays b and c of size n, and one integer variable k, satisfying $1 \leq k \leq n$, are employed. All the entries of both arrays are initialised to *true*, and k's initial value is irrelevant. All threads use the same algorithm, which runs in two phases. During the first, thread i sets $b[i] := \mathit{false}$, and repeatedly checks the value of $b[k]$, setting $k := i$ if $b[k]$ is true, until $k = i$ holds. At this point, thread i starts phase 2, where it sets $c[i] := \mathit{false}$, and enters the critical section if $c[j]$ holds for all $j \neq i$. If the check fails, it sets $c[i] := \mathit{true}$ and restarts in phase 1. After the critical section, $b[i]$ and $c[i]$ are set to *true*. Note that more than one thread could pass phase 1, and phase 2 is thus necessary.

We encoded Dijkstra's algorithm into a c-net as follows. We denote by n-Dijkstra the c-net that encodes Dijkstra's algorithm running on n processes. The entries of arrays b, c are represented by two places, e.g., $b_i{=}t$ and $b_i{=}f$. Variable k is encoded by n places of the form $k{=}0$, $k{=}1$, ..., $k{=}n{-}1$. Places $l_{0,i}, \ldots, l_{6,i}$ encode the instruction pointer of thread i. Figure 32 shows the fragment of 2-Dijkstra that encodes thread 0. Roughly, each transition represent one instruction of the original algorithm [Dij65], updating the instruction pointer and the variables affected by the instruction. Transitions encoding conditional instructions, like $k = 0?$, or $\forall j \neq 0, c_j = t?$ employ read arcs to the places coding the variables involved in the predicate.

MPs of n-Dijkstra, and in particular CMPs, exhibit a very good growth with respect to n. Table 9 shows the figures, obtained under the same setting as in § 7.6. While all unfoldings are exponential in n and $|T|$, all the MPs are of polynomial size. The sizes of the plain and PR unfoldings seem to increase by a factor of 5 for each process added. The contextual unfolding reduces this factor down to 3. The plain and PR MPs seems to fit a polynomial curve of degree close to 3. The CMP seems to grow linearly with n^2, i.e. linear with $|T|$, the number of transitions in the net. As it was the case

Table 9: Unfolding and MP sizes of n-DIJKSTRA, its plain, and PR encodings. Last row obtained through regression analysis, see the text.

Net		Merged Processes			Unfoldings		
n	$\lvert T\rvert$	Ctx	Plain	PR	Ctx	Plain	PR
2	18	31	42	40	35	54	54
3	36	64	113	121	131	371	364
4	60	105	220	278	406	2080	1998
5	90	155	375	582	1139	10463	9822
6	126	214	589	1198	3000	49331	44993
	$\propto n^2$	$\propto n^2$	$\propto n^3$	$\propto n^3$	$\propto 3^n$	$\propto 5^n$	$\propto 5^n$

for the family of c-nets n-GEN, discussed in § 5.3, PR MPs seem to be less efficient than plain MPs on n-DIJKSTRA.

We note that this example exhibits some of the features explained in § 5.3. For instance, process 0 can transition from $l_{5,0}$ to $l_{2,0}$ if there exists another process i with $c_i = f$. Thus, for $n \geq 3$ there would be a choice between multiple (i.e., $n-1$) transitions in parallel to implement the check, a structure also found in the n-GEN example. We note that such structures would also naturally ensue from other mutual exclusion algorithms that typically involve checking for the presence of some other event with a certain property.

8 CONCLUSION AND PERSPECTIVES

The study of asymmetric event structures has, not surprisingly, focused so far on foundational aspects. The publication of [BCKS08] was a turning point, making in principle possible the use of these structures in practical verification. The question then was whether this is profitable.

This dissertation demonstrates that asymmetric event structures can be rendered practical and outperform existing techniques on non-trivial classes of systems. They have, we believe, a rightful place in research on concurrency, also from an efficiency point of view.

The manuscript makes theoretical and practical contributions to model checking and fault diagnosis, focused on improving their scalability. In particular, the methods and tools proposed here open the way to using contextual unfoldings for practical verification.

Unfolding-based verification conceptually takes place in two steps. First, an unfolding of the system is *constructed*, carrying all information relevant to deciding the verification question. Then the unfolding is *analyzed*, to actually answer the verification question.

UNFOLDING CONSTRUCTION. For this step, we provide in Ch. 3 one concrete method for computing contextual unfoldings, and develop contextual merged processes in Ch. 5. Both techniques aim at representing compactly the state space of contextual Petri nets, achieving different degrees of compression, and both were studied with a view to efficiency. Experimental evaluation of these methods has been carried out, the results were reported in Ch. 7.

For contextual unfolding construction, the main contribution presented here is the eager approach to compute possible extensions. We proposed to associate histories to conditions of the prefix and presented a concurrency relation on conditions enriched with such histories. We showed how to characterize and compute possible extensions with this relation and gave key results for computing the relation itself. This notion of concurrency was then refined into asymmetric concurrency, providing not only a more uniform and simple characterization of possible extensions, but also a faster algorithm for their computation.

We implemented the eager algorithm into the Cunf Toolset, a set of competitive verification tools entirely developed during this thesis, striving to make the implementation fast and robust. These tools have been tested and profiled over a large set of benchmarks. Important implementation details were reported in Ch. 7. Experiments show that not only are contextual unfoldings more compact than ordinary ones, but they can be computed with the same or better efficiency, in particular with respect to alternative approaches based on encoding contextual nets into ordinary nets.

Formal verification of concurrent systems faces diverse sources of state-space explosion. Contextual unfoldings mitigate the combinatorial explosion due to concurrency and concurrent read access. We have proposed contextual merged processes, which additionally cope with the explosion due to sequences of choices. We proved a number of results which lay the foundation for their construction and their use in model checking of reachability-

like properties. Experiments showed that they can be orders of magnitude more compact than contextual unfoldings.

UNFOLDING ANALYSIS. Once a representation of the system is available, either in the form of an unfolding or a merged process, formal verification reduces to a particular analysis of the representation. Methods for answering reachability and deadlock-freeness queries using both contextual unfoldings and contextual merged processes were presented in Ch. 4 and Ch. 5, and partially evaluated in Ch. 7. In both cases, the problem in question was reduced to SAT. Additionally, a method was presented in Ch. 6 to perform weak diagnosis of faults under fairness constraints using *ordinary* unfoldings.

Our SAT encodings for reachability and deadlock are quite concise, due to the aforementioned compactness of contextual unfoldings and merged processes. The performance of modern SAT solvers is however only loosely related to the encoding size. Optimizations were necessary to make the presented deadlock-checking method competitive compared to existing methods. Among all proposed ones, particular attention was put on the acyclicity constraint, as it often dominates the size of the encoding. Once implemented in the Cunf Toolset, the approach was compared with other unfolding-based techniques, showing that the method is practical and outperforms existing techniques on a wide number of cases.

Weak diagnosis explores the fair executions of the system that are compatible with a given observation and determines whether an unobservable fault is inevitable. We stress that despite its name, *weak* diagnosis is actually stronger than conventional diagnosis. Our results in Ch. 6 allow to decide whether a given observation weakly diagnoses a fault. They require to build two unfolding prefixes, one to represent all behaviors compatible with the observation and another one to represent all fair behaviors. We then reduced the weak diagnosis problem to a SAT instance generated out of these prefixes.

PERSPECTIVES. Along the manuscript, we have already mentioned ideas for short-term future work extending of the results presented. We now present additional ideas for extension and some more broader perspectives of future work. The following are ideas for future work that follow up the developments in this manuscript:

- The algorithm for direct construction of ordinary merged processes proposed in [KM13] is based on a procedure for reachability checking. The results in § 5.5 lay the foundation for reachability checking with CMPs. The major ingredient for transferring [KM13] to the contextual case is thus ready. An important problem yet to be solved is finding a total strategy that has good encoding into SAT, as the one presented in [KM13].
- While the SAT-based analysis techniques for unfoldings and merged processes presented here exploit the compactness of these representations, they include a new acyclicity constraint which is not necessary in ordinary nets. This constraint dominates the size of the encoding and often reduces the performance of the solver. Improvements on it may significantly speed up solving times.
- The step semantics considered in this work follow those proposed in [MR95]. Other semantics have been proposed [JK91; JK95]. Study-

ing the algorithmics of the resulting structures remains a possible line of future work.

- Most of the time spent for the construction of an unfolding is employed in computing the concurrency relation. More efficient algorithms for constructing concurrency relations thus have direct impact on the unfolding time. The concurrency relation itself is used for two purposes: (1) to find, ultimately, possible extensions, and (2) to update the relation itself. However, there are pairs in the relation which never contribute, even indirectly, to any of these purposes, and their computation could be avoided — for the advanced reader, think of a net with two isolated components which have large unfoldings. Algorithms and data structures to compute only the necessary fragment of the relation are likely to speed up unfolding construction.
- We intend to produce an implementation of the presented diagnosis approach. While many of its ingredients are available by minor modifications of existing tools and verification infrastructure, e.g., [Sch], a practical obstacle to overcome could be the sizes of the prefixes required in order to perform diagnosis.

Existing literature on unfoldings, including the works behind this dissertation, have focused on finite nets with finite state space [EH08]. Moreover, the algorithmics of unfoldings, and tools implementing them, have almost exclusively focused on 1-safe nets [ERV02; ER99; EH08; KM13], as non-safeness is a well known source of explosion for unfoldings [KKKV06]. An exception to this is [AIN04], where the coverability problem of an unbounded Petri net is solved using backwards unfoldings.

Leveraging on the efficiency of unfolding algorithms for safe nets, an interesting perspective is the investigation of unfoldings for 1-safe, *infinite* Petri nets. Since they are safe, their algorithmics are likely to be efficient. Because they are infinite, certain verification questions could be reduced to them. For instance, straightforward semantics could be given to non-recursive Boolean programs with unbounded thread creation [CKS07], reducing safety properties to coverability queries on the net.

The coverability problem for these infinite nets is of course undecidable. However, this should not prevent us from making an abstract interpretation of them. Under reasonable assumptions, such as finite-enabling[1], such nets can readily be given unfolding semantics. Next, the coverability problem could be overapproximated by a finite prefix constructed with a suitable cutoff criteria. In the definition these criteria, the theory of canonical prefixes [KKV03] will arguably provide a supporting framework.

Although read arcs seem not to play a central role in such extension, they would arguably contribute to more compact unfoldings.

A second perspective would be related to partial-order reduction techniques, also referred as model checking using representatives [Pel93]. These techniques and unfoldings are often presented as two classes of approaches to cope with the state explosion problem derived from concurrency. Although they are applied to the similar problems, we are not aware of any work comparing them. Understanding better the connections between the two would not only be of interest for practitioners trying to decide which to use for a particular problem, but arguably also for theoreticians seeking for better representations of concurrent state spaces. We intend to carry out an study in this direction.

1 That is, only finitely many transitions are enabled at any reachable marking.

A

THE CUNF TOOL USER'S MANUAL, V1.6

This is both a user guide and a tutorial of the Cunf Toolset. The Cunf Toolset is a toolset for carrying out unfolding-based verification of Petri nets extended with read arcs, also called *contextual nets* (c-nets). Unfoldings fully represent the state-space (reachable markings) of a c-net by a partial order rather than by a set of interleavings; they are often exponentially smaller than the reachablity graph, and never larger than it. Additionally, c-net unfoldings can be exponentially more compact than those of corresponding Petri nets. The toolset specifically contains one unfolding-construction tool and one reachablity and deadlock-checking tool.

A.1 INTRODUCTION

The *Cunf Toolset* is a *set* of programs for carrying out unfolding-based verification of Petri nets extended with read arcs, a.k.a. contextual nets, or c-nets. The package specifically contains the following tools:

1. CUNF: constructs the unfolding of a c-net;
2. CNA: performs reachability and deadlock analysis using unfoldings constructed by CUNF.
3. Scripts such as `pep2dot` or `grml2pep` to do format conversion between various Petri net formats, unfolding formats, etc.

Petri nets are a modelling language for concurrent systems. The reader unfamiliar to the topic could perhaps start with [Wik13] or [Mur89].

Contextual nets are Petri nets where, in addition to the ordinary *arrows* between places and transitions, one may find *read arcs*. These allow transitions to verify that tokens exist in a place before firing, but don't consume them when firing. Transitions can then be thought of reading a *context* required to fire, hence the name. See Section 2 of [BBC+12] for a brief formalization and [MR95] for more details.

Observe that every Petri net is a contextual net (without read arcs). Also notice that for every c-net we obtain an equivalent Petri net after substituting read arcs for pairs of *consume-produce* loops. We call this Petri net the *plain encoding* of the c-net. An example of this encoding is shown in Fig. 33.

The unfolding of a c-net is another well-defined c-net of acyclic structure that fully represents the *behavior* (reachable markings) of the first, see Fig. 33 (c) for an example. A c-net unfolding is at most as big as the reachablity graph of the c-net.[1] Because unfoldings represent behavior by partial orders rathen than by interleavings, for highly concurrent c-nets, unfoldings are often much (exponentially) smaller, which makes for natural interest in them for the verification of concurrent systems.

C-net unfoldings bring additional advantages w.r.t. ordinary Petri net unfoldings. The unfolding of a c-net can be exponentially smaller than the

1 The careful reader may notice that there are more events in Fig. 33 (c) than reachable markings in Fig. 33 (b). This is a matter of presentation. (c) is actually the *full unfolding* [BBC+12] of (b), while the term *unfolding* in this document refers to the finite marking-complete unfolding prefix that one can build using a total adequate order [BBC+12]. In other words, (c) contains some cut-off events (actually only 1) that, when removed, would yield a marking-complete prefix not larger than the state-space of (b).

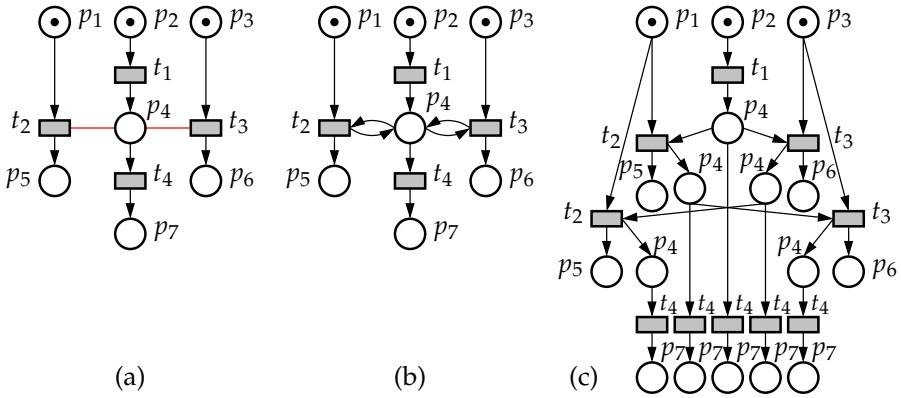


Figure 33: (a) a c-net; (b) its encoding into a Petri net; (c) unfolding of (b)

unfolding of its plain encoding. For instance, Fig. 33 (a) is a c-net and (b) its plain encoding. The unfolding of (b) is (c), but the unfolding of (a) is a c-net isomorphic to (a). If (a) was generalized to *n reading transitions* (copies of t_2), we would still have an isomorphic contextual unfolding, but (c) would blow up. See [BBC+12] for more details.

An unfolding is suitable for checking certain properties of the net giving rise to it, such as reachability or deadlock-freeness. Checking these directly on the net is computationally difficult (PSPACE-complete). Building the unfolding is (for highly concurrent systems) efficient, and checking these properties using the unfolding is also *easy* — NP-complete. Together, unfolding construction and unfolding analysis is almost always faster than verification based on the reachability graph. Also notice that we can check many properties once the unfolding is built.

CUNF implements the c-net unfolding procedure proposed by Baldan et al. in [BCKS08]. The algorithms and data structures actually implemented have been partially described in [RSB11b; BBC+12]. While the theoretical results of [BCKS08], [RSB11b; BBC+12] allow for unfolding bounded c-nets in general, CUNF can only unfold 1-safe c-nets (i.e., no reachable marking puts more than one token on every place), and for the time being the tool will blindly assume the input is 1-safe — the unfolding could be wrong if this is not fulfilled. In [Rod10], and old and inefficient version of the tool is described.

CNA, whose name stands for *Contextual Net Analyzer*, checks for place coverability or deadlock-freedom of a c-net by examining its unfolding. The tool reduces these problems to the satisfiability of a propositional formula that it generates out of the unfolding, and uses MINISAT [ES03] as a back-end to solve the formula. The algorithms used by CNA has been described in [RS12b].

A.2 AUTHOR AND CONTACT

The Cunf Toolset is developed and maintained by

César Rodríguez

LSV, ENS de Cachan
61, avenue du Président Wilson
94235 Cachan Cedex
France

e-mail: cesar.rodriguez@lsv.ens-cachan.fr
Website : http://www.lsv.ens-cachan.fr/~rodriguez/

If you experience any difficulty using the tool, or found a bug, or just wish to let me know that you are using it, send me an e-mail.

A.3 INSTALLATION

The Cunf Toolset has only been tested in Linux and Mac OS X; it should run well in other Unix systems. You can choose to install precompiled binaries or compile and install from the source code.

CNA requires the MINISAT solver to be installed in your machine, and available in the `$PATH`. For your convenience, MINISAT v.2.2.0 is distributed with the precompiled binaries and can be found in the `minisat/` folder of the repository.

A.3.1 *From Precompiled Binaries*

From the following address, choose the latest version (1.6) of the bundle with precompiled binaries that suits your machine, download, and unpack it:

https://code.google.com/p/cunf/downloads/list

Then follow the next steps:

1. Copy all files in the `bin/` folder to any folder in your computer pointed by your `$PATH`.[2]
2. The folder `lib/ptnet/` contains a Python module that should be correctly installed in your system for CNA and other Python scripts to work. Copy it to, **either**,
   - any folder pointed by your installation-dependent Python's default *Module Search Path*,[3]
   - **or** any folder pointed by (one of the paths in) the environment variable `$PYTHONPATH`,
   - **or** the same folder where you copied CNA and the other Python scripts, as Python will first search for modules in that folder.

A.3.2 *Compilation and Installation from the Source Code*

First, get the source code. You have two options:

1. Download and unpack the boundle containing the source code of version 1.6 of the tool, from

   https://code.google.com/p/cunf/downloads/list

2. *Clone* the `Git` repository where the development takes place, running the command:

   ```
   git clone https://code.google.com/p/cunf/
   ```

2 In particular, the MINISAT tool has to be somewhere where the *system(3)* C-library function will find it.

3 To discover the actual search path that Python is using run the command: `echo 'import sys; print sys.path' | python`.

CUNF is written in C, the source code is the `src/` and `include/` folders of the tool repository. CNA is written in Python, and uses the `ptnet` Python module, both are located in the `tools/` folder.

The compilation is handled by `make`, the following targets are available:

`ALL` Compiles CUNF and MINISAT, leaving the respective binaries in the files `src/main` and `minisat/core/minisat`.

`DIST` Builds all necessary files and creates the folder `dist/`, containing the binaries and libraries ready to be installed — the same files that are distribued from the boundle in § A.3.1.

Some compilation-time options can be tweaked in `include/config.h`, here you have a description:

`CONFIG_DEBUG` Define this macro to compile debugging code. CUNF will perform extensive assertions aiming at finding bugs in the code and corrupt data structures. It will additionally dump verbose debugging information during the unfolding computation. Switching on the option will considerably increase CUNF's running times.

`CONFIG_MCMILLAN` Define this macro to use McMillan's adequate order [McM95b] when CUNF picks possible extensions to extend the unfolding prefix. One and only one of the macros `CONFIG_MCMILLAN`, `CONFIG_PARIKH`, `CONFIG_ERV` or `CONFIG_ERV_MOLE` can be active. Recall that this is not a total order.

`CONFIG_ERV` Selects the $\prec_F$ total adequate order of [ERV02].

`CONFIG_PARIKH` Selects the *Parikh* order of [EH08, definition 4.46]. Although this this is an adequate order, it is *not* total, and may produce prefixes larger than any of the other available total orders. However, it always produces prefixes smaller than when using `CONFIG_MCMILLAN`.

`CONFIG_ERV_MOLE` Selects the adequate order used in the Mole unfolder [Sch]. This is a minor, non-documented modification of the order $\prec_F$. It makes possible to compare the unfolding prefixes computed by CUNF and MOLE. Refer to the function `h_cmp`, in `src/h.c`, for more details.

`CONFIG_NODELIST_STEP` Defines the number of items to be allocated together whenever certain dynamic linked-lists need to grow. Among other uses, these lists are used to store markings.

To generate the binaries, execute

```
$ make dist
```

and follow the steps in § A.3.1 with the folder `dist`. Observe that your installed copy of CNA will not work before you install the `ptnet` module, as indicated in § A.3.1. You can of course run CNA from the `tools/` folder, since Python first searches for modules in the same directory where the executable is located.

A.4 GETTING STARTED

In this section we explain the usage of the Cunf Toolset with a simple, well-known example.

Figure 34 shows a c-net representing a variant of Dekker's mutual exclusion algorithm for two processes, represented by numbers 0 and 1 in the figure. A process may fire `try` and set a `flag` to signal intention to enter the critical section. It can then `enter` the critical section `p3` if the other process is

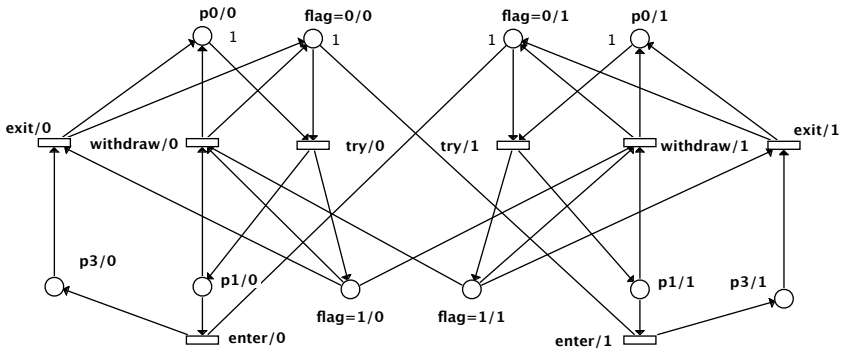


Figure 34: A variant of Dekker's mutual exclusion algorithm for two processes

not trying. Otherwise it may `withdraw` its intention and clear the flag. Read arcs allow `enter` and `withdraw` to read the flag of the other process.

This c-net is distributed in the `examples/` folder, included in the boundle of § A.3.1 or the `dist/` folder generated after compilation, see § A.3.2. The file in question is

```
examples/dekker/dek02.ll_net.
```

For the purposes of this presentation, let us just give it a shorter name:

```
$ cp examples/dekker/dek02.ll_net dek02.ll_net
```

This file, as any other c-net in the `examples/` folder, is formated in a simple modification of the PEP's *low-level* Petri net format, which will be described in § A.6. For historical reasons, this is the only input format of the Cunf Toolset. The toolset comes, however, with scripts to translate the output format of some graphical editors, like Coloane [LIP] or PIPE2 v2.5 [BLPK07], see § A.7 for more details.

A.4.1 *Constructing Unfoldings with Cunf*

Assume we wish to check if `dek02.ll_net` is deadlock-free, or whether mutual exclusion is guaranteed. We first build its unfolding using Cunf, and store it in the file `u.cuf`:

```
$ cunf -o u.cuf dek02.ll_net
```

By default Cunf outputs in `CUF03` format (documented in § A.6). A number of statistics about the computation of the unfolding and the size and shape of the output are printed just before the tool terminates. We can distinguish the following lines:

```
[...]
hist    12
events  8
cond    18
[...]
co(r)   5.25
rco(r)  0.50
[...]
pre(e)  1.75
ctx(e)  0.50
```

```
pst(e)  1.75
cutoffs 6
[...]
```

The first line above is the number of *histories* generated by CUNF (you may see [BBC+12] for a definition), and gives a rough idea of the size of the internal object CUNF had to build in order to produce the unfolding. The unfolding itself has the number of *events* (transitions) and *conditions* (places) reported in the next two lines.

CUNF's unfolding algorithm builds and maintains a *concurrency relation* that the tool internally needs for constructing the unfolding. Explaining the purpose of this relation is out of the scope of this manual (see [BBC+12]). It is, however, important to say that computation of this relation often takes most of the time consumed by the tool. In the line

```
co(r)   5.25
```

CUNF reports that every element over which the relation is defined was related to an average of 5.25 other elements. In general, the higher this number is, the slower CUNF's unfolding algorithm proceeds. More details in [BBC+12, section 7].

Next in the list is the average number of conditions in the preset, context, and postset of every event. The last line reports the number of cut-off (enriched) events present in the internal object constructed by CUNF.

A.4.2 *Deadlock and Coverability Analysis with Cna*

After the construction of the unfolding, we can proceed to its analysis. We use now CNA to ask, for instance, about the presence of deadlocks:

```
$ cna -d u.cuf
answer          : NO , the net is deadlock-free
clauses         : 52
event variables : 4
reductions      : bin 4-tree
sccs            : [(4, 4, 4, 4)]
variables       : 30
```

Option `-d` instructs CNA to look for deadlocks. By default, it will build and solve a propositional formula associated to the unfolding, and will internally invoke MINISAT to solve it. If called with `-n`, CNA will instead dump the formula and exit.

Option `-c` checks for coverability, in this case, of places `p3/1` and `p3/0`:

```
$ cna u.cuf -c 'p3/1' 'p3/0'
answer       : NO , no rechable marking covers 'p3/0' 'p3/1'
[...]
```

Observe that `p3/i` is marked iff process `i` is in the critical section. We may also ask if it is possible to find one process trying to enter the critical section while the other is already in, which is naturally possible:

```
$ cna u.cuf -c 'p1/0' 'p3/1'
answer          : YES, places 'p1/0' 'p3/1' are coverable
clauses         : 39
event variables : 4
```

```
reductions      : bin 4-tree
sccs            : [(4, 4, 4, 4)]
trace           : 'try/1:e0' 'enter/1:e3' 'try/0:e1'
variables       : 22
```

The `trace` indicates the firing sequence of the c-net (or the unfolding) that makes possible to cover the requested places. The lines labelled by `clauses` and `variables` inform about the size of the SAT formula fed to MINISAT.

Notice that in this example, the construction of the `CUF03` file is necessary only once, while CNA can be applied many times. For most problems, the former step is the bottleneck whereas CNA works very fast thanks to efficient SAT solving techniques. For cases where only one coverability query is needed, the option `-t` of CUNF checks whether a given transition is firable, and stops computing the unfolding if the answer is yes. This can be used to check if the set of places in the preset of a (intentionally inserted) transition is coverable.

Notice the line starting with `reductions`. CNA applies several optimizations to the SAT encoding it produces. The word `bin` means that it is using *binary rankings* to encode certain acyclicity constraint; `4-tree` says that CNA used *4-trees* to encode the absence of symmetric conflicts (technical details in [RS12b], [RS12a]). Run the tool without arguments to obtain a full list of all the optimizations available, and information about them. This information is available in § A.5.2. The effect of almost all them is described in [RS12a].

A.4.3 *More on Dekker's Algorithm*

How the unfolding grows as we add more processes to our model of the Dekker's algorithm?

First, observe that when the 2-process example in Fig. 34 is generalized to n processes, $\mathcal{O}(2^n)$ markings are reachable in these c-nets — for a net of size $\mathcal{O}(n^2)$. Several instances of the protocol are distributed in the folder `examples/dekker/`. We can easily run CUNF on some of them and retain the unfolding size:

```
$ for i in 10 20 30 40 50; do
   cunf examples/dekker/dek$i.ll_net | grep events
done

events  120
events  440
events  960
events  1680
events  2600
```

We see that the numbers roughly follow an square progression on the number of processes involved, i.e., linear on the number of transitions of the c-net.

A.4.4 *Producing Graphics of C-nets and Unfoldings*

The Cunf Toolset is distributed with a number of scripts capable of producing images depicting a c-net or an unfolding. These scripts actually rely on the tool `dot` from the Graphviz project [Gra] to actually produce the image.

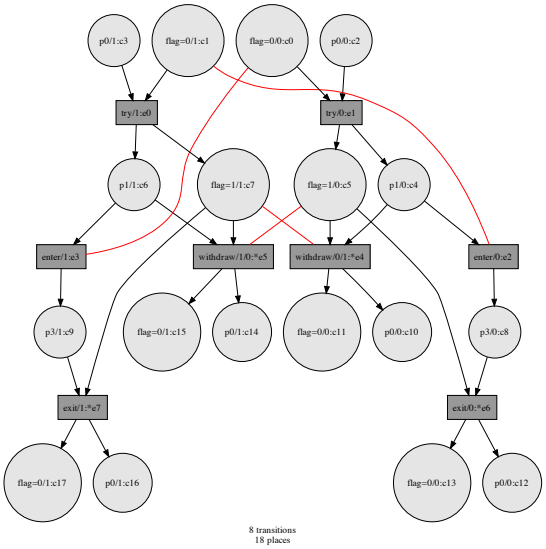


Figure 35: The unfolding of Fig. 34.

Say that we want to *see* the c-net `dek02.ll_net` which we unfolded in the previous sections. We first generate a *dot* script out of the net, using the tool `pep2dot`, included in the `src/` directory of the source code (and also in the precompiled distribution):

```
$ src/pep2dot dek02.ll_net > dek02.dot
```

Then use `dot` tool, like this:

```
$ dot -T pdf < dek02.dot > dek02.pdf
```

The file `dek02.pdf` depicts the c-net. Similarly, if we wish to see the unfolding, we could type:

```
$ tools/cuf2pep.py < u.cuf > u.ll_net
$ src/pep2dot u.ll_net > u.dot
$ dot -T pdf < u.dot > u.pdf
```

The first command has converted the `cuf` file in to a `ll_net` file, and subsequent commands are like before. The `cuf2pep.py` script is also included in the precompiled distribution. The resulting PDF is shown in Fig. 35. Regular arrows are in black, and read arcs are depicted in red. The initially marked conditions are the four top ones, and cut-off events have an asterisk in the name.

Actually, the `make` machinery included in the source code already knows how to produce PDF or even JPEG images of the c-nets or unfoldings — pretty much in the same way it knows how produce an object file out of a C source file. You just need to type `make` followed from the desired file. For instance, we can obtain a JPEG image of the c-net with:

```
$ make dek02.jpg
P2D dek02.ll_net
DOT dek02.dot
rm dek02.dot
```

The tool `make` reports that it first converts the PEP `ll_net` file into a `dot` script (`P2D`), and then converts the `dot` script into PDF (`DOT`). This will produce the file `dek02.jpg`.

The same is true for building unfoldings. To build the unfolding of a net `abc.ll_net` you just need to *make* the file `abc.unf.cuf`:

```
$ make dist
$ make dist/examples/dijkstra/dij03.unf.cuf
UNF dist/examples/dijkstra/dij03.ll_net
time    0.013
mem     2
[...]
```

Of course, more complex transformations can be achieved. For instance, we could generate a PDF depicting the unfolding of a c-net without explicitly generating the `cuf` file:

```
saiph:cunf$ make dist/examples/dijkstra/dij02.unf.pdf
UNF dist/examples/dijkstra/dij02.ll_net
time    0.001
mem     0
[...]
C2P dist/examples/dijkstra/dij02.unf.cuf
P2D dist/examples/dijkstra/dij02.unf.ll_net
DOT dist/examples/dijkstra/dij02.unf.dot
rm dist/examples/dijkstra/dij02.unf.ll_net
   dist/examples/dijkstra/dij02.unf.dot
   dist/examples/dijkstra/dij02.unf.cuf
```

Observe that `make` make is removing, in the last line, all intermediate files it has generated — because you did not ask for any of them. The file generated by the previous command is shown in Fig. 36.

If you wish to know other transformation rules `make` knows, have a look to the end of the file `defs.mk`, located at the root of the repository.

A.4.5 *Finding More Examples*

More examples are distributed in the `examples/` folder in the bundle of § A.3.1 or the (generated) `dist/examples/` folder in the source code:

`EXAMPLES/DIJKSTRA/` This folder contains c-nets that model Dijkstra's mutual exclusion algorithm [Dij65] for a number of processes varying between 2 and 6. These nets have been generated using the script `tools/mkdijkstra.py`, included in the source code (but not the bundle of precompiled binaries).

`EXAMPLES/CORBETT/` This is a superset of the of the popular benchmarks compiled by Corbett [Cor96]. An short description of them can be found in [Kho03, pp. 29–31]
The nets are distributed in four sub-folders. The folder `cont/` contains contextual nets, and `plain/` contains the plain encodings of the nets in `cont/`. Similarly, the folder `pr/` contains the *Place-Replication encoding* [VSY98] of the same c-nets. The folder `other/` contains ordinary Petri nets.

The folder `tools/` of the source code contains a number of scripts (all them have names starting by `mk`) to generate several families of nets, other Dijkstra's or Dekker's mutual exclusion models. This folder is *not* distributed with the bundle of precompiled binaries.

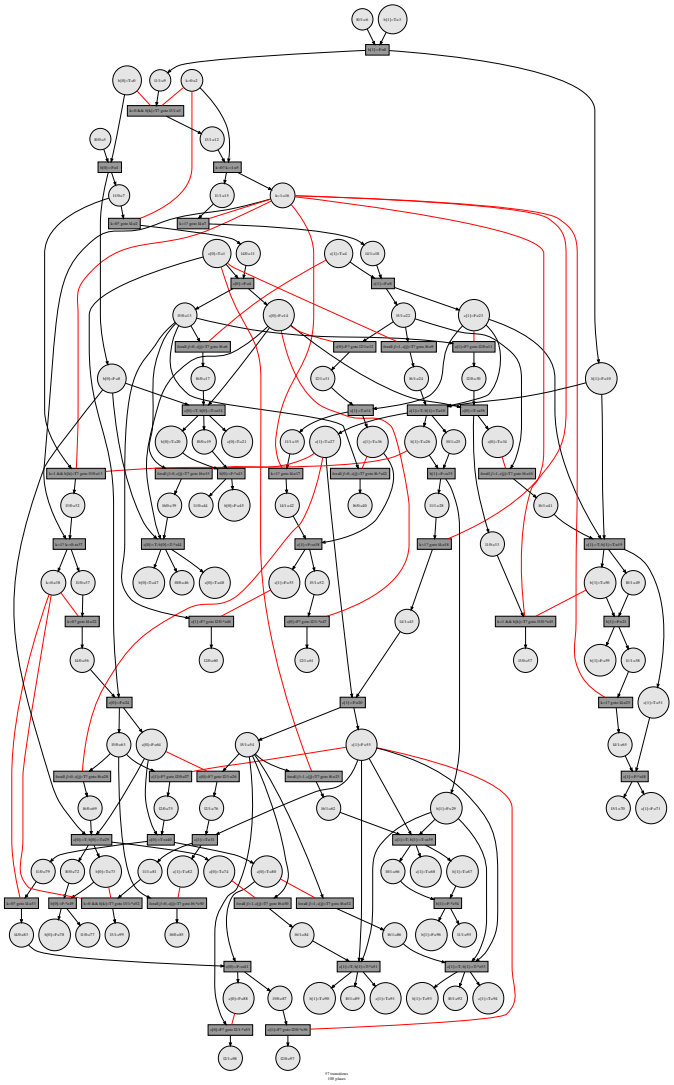


Figure 36: The unfolding of `examples/dijkstra/dij02.ll_net`.

A.5 COMMAND-LINE SYNTAX

A.5.1 *Cunf*

CUNF expects an optional list of command-line options followed by the name of the input file to be provided as command-line arguments:

`cunf [OPTIONS] NETFILE`

Run the tool without arguments to obtain the list of available options. They are the next:

- `-t NAME` Optional. Stop the unfolding construction as soon as the transition `NAME` is unfolded for the first time, and output the unfolding prefix currently built. The prefix will include exactly one occurrence of the transition, if it has been found, or zero occurrences — and it will be a *marking-complete*, cf. [BBC+12].
- `-d DEPTH` Optional. `DEPTH` is a natural number. Don't include in the unfolding prefix enriched events whose history has a depth greater or

equal than DEPTH. The depth of a history H is the maximum natural n for which there exist events $e_1, \ldots, e_n$ in H such that $e_1 \nearrow e_2 \nearrow \ldots \nearrow e_n$, where $\nearrow$ is the *asymmetric conflict relation*, as defined in [BBC+12].

`-O FILE` Optional. Write the unfolding prefix to the file FILE. If not provided, CUNF will obtain the path to write the unfolding from the input file. It will strip the string `ll_net` from the INPUT file and will concatenate the string `unf.dot`.

`-F FORMAT` Optional. FORMAT shall be one of `cuf`, `dot`, or `fancy`. See § A.6.2.

A.5.2 *Cna*

Run CNA without arguments to obtain help about its command-line syntax. The tool expects a mandatory CUF file and an optional OUTFILE where it will write its textual output (standard output if not given):

```
cna [CUF] [OUTFILE] [OPTIONS]
```

Here is the list of OPTIONS it accepts:

`-H, -HELP` Show a help message and exit.

`-D, -DEADLOCK` Optional. Default: no. Tell whether or not a deadlocked marking is reachable. At least one of `-d` or `-c` must be given.

`-C PLACE [PLACE ...], -COVER PLACE [PLACE ...]` Optional. Default: no. Tell whether or not the list of PLACEs are coverable. You may consider quoting place names if they contain spaces. At least one of `-d` or `-c` must be given.

`-R REDUC [REDUC ...], -REDUCE REDUC [REDUC ...]` The default is: `4-tree bin`. Use REDUCtions of the propositional formula generated by the tool that improve running time of the SAT solver. The following options are available:

- `K-TREE` (where $1 < k < 10$) Use k-trees to implement the at-most-one constraints in the encoding of symmetric conflicts. If none of the options '`seq`', '`log`', or '`pair`' is present, `4-tree` will be used.
- `SEQ` Use the sequential encoding of [Sin05] for symmetric conflicts.
- `LOG` Use the logaritmic encoding of [FPDN05] for symmetric conflicts.
- `PAIR` Use pairwise encoding for symmetric conflicts.
- `STB` Use elimination of stubborn events, see [RS12a] for a detailed, technical description of the optimization.
- `SUB` Reduce the *symmetric* and *disabled* constraints to certain maximal sets.
- `NOCY` Do not produce constraints to check for cycles in the asymmetric conflict relation.
- `BIN` Generate acyclicity constraints using ranks with binary encoding, see [CGS09]. If none of the options '`nocy`', '`trans`', or '`unary`' is present, '`bin`' will be used.
- `UNARY` Generate acyclicity constraints using ranks with unary encoding.
- `TRANS` Generate acyclicity constraints encoding transitive closure.
- `SCCRED` Reduce the SCCs before generating the acyclicity constraints.

`-N FILE, -DONT-SOLVE FILE` Dump the propositional formula to FILE and exit, instead of running the solver and displaying the result.

A.6 FILE FORMATS

A.6.1 *The `ll_net` Format*

CUNF accepts a c-net as input, represented in a slightly modified version of the PEP's low-level format. We describe here this modification. We assume the reader is familiar with the PEP's low-level format [PEP; BG].

The input file should be formated in the low-level net syntax extended with one additional section, which specifies the context relation. The new section contains a list of read arcs, each one specified in the same way as either the place-transition arcs in section `PT` or the transition-place arcs in section `TP`. The new section is headed by the keyword `RA`, standing for Read Arcs.

Sections in the low-level net format must be placed in certain order. The new `RA` section must appear after the section `TP` (transition to place arcs) and before the (optional) section `PTP` (phantom transition to place arcs). Figure 37 shows an example, together with the graphical representation of the c-net.

```
PEP
PetriBox
FORMAT_N2
PL
"P0"M1
"P1"M1
"P2"
"P3"
"P4"
"P5"
TR
"T0"
"T1"
"T2"
"T3"
TP
1<3
2<4
3<5
4<6
PT
1>1
2>2
3>3
3>4
4>4
RA
2<1
```

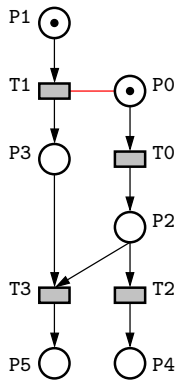


Figure 37: Example of the `ll_net` format with read arcs, and graphical representation of the encoded c-net.

A.6.2 *Unfolding Formats*

CUNF can write its output, the unfolding of the input c-net, in several formats. By default, it will write a `cuf` file, but other formats can be selected with the option `-f`:

DOT For historical reasons, CUNF is able to write `dot` scripts, suitable for the `dot` tool. Scripts to produce `dot` input out any `cuf` or file are included in the source and precompiled distributions.

FANCY This also produces a `dot` file, but every event in the resulting graphical representation of the unfolding is annotated by the histories that CUNF had to construct. This is right now the only way of extracting information about the history-enriched unfolding prefix that CUNF constructs — apart from compiling the tool with `CONFIG_DEBUG` and parsing the debugging output.

CUF The default output format, called `CUF03`, is binary and similar to the MCI format of PEP. C and Python code for dealing with `CUF03` files is available in the repository.
The standard reference defining the `CUF03` format is the comment in the the line 263 of the file

`src/output.c`,

just before the function `write_cuf`. All technical details of the file format are explained there.

A.7 PRODUCING INPUT FOR CUNF

Most of the c-nets on which the Cunf Toolset has been used has been generated programmatically. It is the case, for instance, of the `dekNN.ll_net` nets in § A.4.

Sometimes it is interesting, however, to draw a c-net and perform some analysis on it. The Cunf Toolset has currently no graphical user interface, and there is no plan for it to be available in the future. The Cunf Toolset is however integrated in the COSYVERIF [Cos] tool, which includes a graphical interface and which will internally invoke the Cunf Toolset in several mouse clicks. There is also plans to include the Cunf Toolset as a verification engine of TAPAAL [DJJ+12].

A.7.1 *Graphically*

A simple way of producing c-nets for CUNF is using the COLOANE graphical editor [LIP]. This is the same editor used in the Cosyverify tool, but can be installed as an stand-alone program. Once you have edited your c-net in COLOANE, see Fig. 38 (observe that you can also introduce read arcs), right-click on the model name in the *Projet Explorer* window, and then click on *Export*. Export the c-net in GRML format.[4]

Once your file, say `net.grml` (the extension is important), is in GRML format, use the script `grml2pep.py` to convert the file into `ll_net` format. If you read § A.4.4, you will be unsurprised to know that the `make` machinery of the Cunf Toolset sources also knows about GRML files. You can, alternatively, type:

```
$ make net.ll_net
```

or even directly

```
$ make net.unf.cuf
```

4 The dialog could incorrectly display GML, instead of GRML.

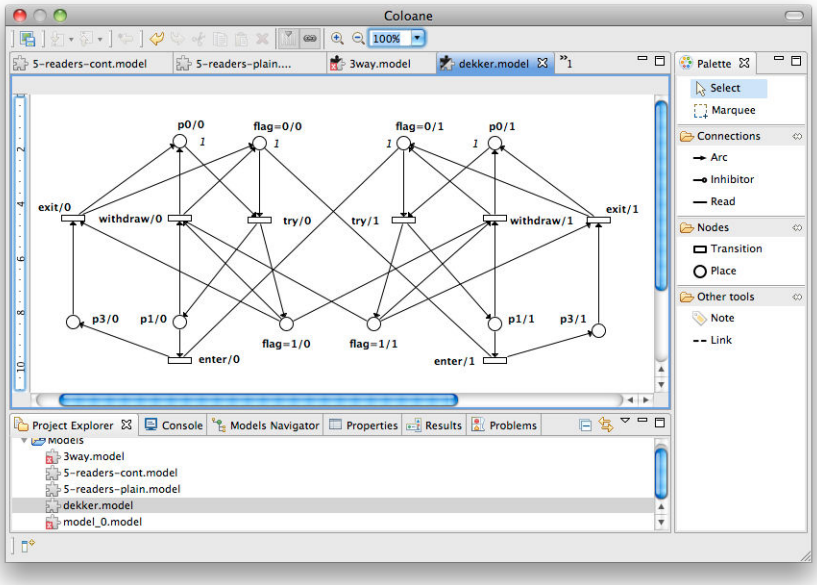


Figure 38: The COLOANE graphical editor can export c-nets for the Cunf Toolset.

A.7.2 *Programmatically*

A Python module for reading and writing a number of c-net and Petri net formats is distributed with the Cunf Toolset. CNA actually uses it to read `cuf` files and many scripts in the `tools/` folder of the sources use it to carry out various operations. All `tools/mk*` scripts use this module, cf. § A.4.5.

We briefly illustrate its use with a simple example. The following code produces the c-net in Fig. 39

```
import sys
import ptnet

# creates the net object
n = ptnet.net.Net ()

# creates three places, two initially marked
p1 = n.place_add ('p1', 1)
p2 = n.place_add ('p2', 1)
p3 = n.place_add ('p3')

# and two transitions
t1 = n.trans_add ('t1')
t2 = n.trans_add ('t2')

# sets arrows and read arcs
t1.cont_add (p2)
t1.pre_add (p1)
t2.pre_add (p1)
t1.post_add (p3)
t2.post_add (p3)
```

```
# writes the c-net in 'pep' format,
# see source code to know about other formats
n.write (sys.stdout, 'pep')
```

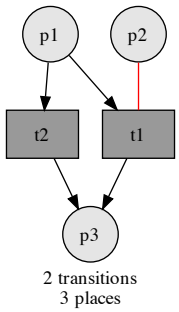


Figure 39: A c-net programmatically produced with the `ptnet` Python module.

BIBLIOGRAPHY

[ADKT11] Jade Alglave, Alastair F. Donaldson, Daniel Kroening, and Michael Tautschnig. "Making Software Verification Tools Really Work." In: *Proc. ATVA*. Ed. by Tevfik Bultan and Pao-Ann Hsiung. Vol. 6996. LNCS. Springer, Jan. 2011, pp. 28–42.

[AF73] Tilak Agerwala and Mike Flynn. "Comments on capabilities, limitations and correctness of Petri nets." In: *Proc. of the Annual Symposium on Computer Architecture*. ISCA '73. New York, NY, USA: ACM, 1973, pp. 81–86.

[AIN04] Parosh Aziz Abdulla, S. Purushothaman Iyer, and Aletta Nylén. "SAT-Solving the Coverability Problem for Petri Nets." In: *Formal Methods in System Design* 24.1 (Jan. 2004), pp. 25–43.

[AMH12] Anoopam Agarwal, Agnes Madalinski, and Stefan Haar. "Effective Verification of Weak Diagnosability." In: *Proc. IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes*. Mexico City, Mexico: IFAC, 2012.

[Avi67] Algirdas Avižienis. "Design of fault-tolerant computers." In: *Proc. AFIPS'67 (Fall)*. New York, NY, USA: ACM, 1967, pp. 733–743.

[Bae05] J.C.M. Baeten. "A brief history of process algebra." In: *Theoretical Computer Science* 335.2–3 (May 2005), pp. 131–146.

[BBC+10] Paolo Baldan, Alessandro Bruni, Andrea Corradini, Barbara König, and Stefan Schwoon. "On the Computation of McMillan's Prefix for Contextual Nets and Graph Grammars." In: *Proc. ICGT*. Vol. 6372. LNCS. 2010, pp. 91–106.

[BBC+12] Paolo Baldan, Alessandro Bruni, Andrea Corradini, Barbara König, César Rodríguez, and Stefan Schwoon. "Efficient unfolding of contextual Petri nets." In: *Theoretical Computer Science (TCS)* 449 (Aug. 2012), pp. 2–22.

[BBD+96] J.P. Bowen, R.W. Butler, D.L. Dill, R.L. Glass, D. Gries, and A. Hall. "An Invitation to Formal Methods." In: *Computer* 29.4 (Apr. 1996), p. 16.

[BC04] Yves Bertot and P Castéran. *Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions*. Berlin; New York: Springer, 2004.

[BCH11] Sandie Balaguer, Thomas Chatain, and Stefan Haar. "Building Tight Occurrence Nets from Reveals Relations." In: *Proc. ACSD*. IEEE, 2011, pp. 44–53.

[BCKS08] Paolo Baldan, Andrea Corradini, Barbara König, and Stefan Schwoon. "McMillan's complete prefix for contextual nets." In: *ToPNoC*. LNCS 5100 (2008), pp. 199–220.

[BCM01] Paolo Baldan, Andrea Corradini, and Ugo Montanari. "Contextual Petri nets, asymmetric event structures, and processes." In: *Inf. Comput.* 171.1 (2001), pp. 1–49.

[BCM+92] J.R. Burch, Edmund M. Clarke, K. L. McMillan, D.L. Dill, and L.J. Hwang. "Symbolic model checking: 10^{20} states and beyond." In: *Information and Computation* 98.2 (June 1992), pp. 142–170.

[BCM98] P. Baldan, A. Corradini, and U. Montanari. "An Event Structure Semantics for P/T Contextual Nets: Asymmetric Event Structures." In: *Proc. FoSSaCS*. Vol. 1378. LNCS. 1998, pp. 63–80.

[Bes96] Eike Best. "Partial Order Verification with PEP." In: *Proc. of Workshop on Partial Order Methods in Verification (POMIV)*. Ed. by G. Holzmann, D. Peled, and V. Pratt. Princeton: American Mathematical Society, 1996, pp. 305–328.

[BFHJ03] Albert Benveniste, Éric Fabre, Stefan Haar, and Claude Jard. "Diagnosis of Asynchronous Discrete Event Systems: A Net Unfolding Approach." In: *IEEE Transactions on Automatic Control* 48.5 (May 2003), pp. 714–727.

[BG] Eike Best and Bernd Grahlmann. *Format descriptions*. Appendix of: PEP Documentation and User Guide 1.8. 1998. [Online, accessed April-2013]: http://parsys.informatik.uni-oldenburg.de/~pep/Paper/formats_all.ps.gz.

[BHK06] Paolo Baldan, Stefan Haar, and Barbara König. "Distributed Unfolding of Petri Nets." In: *Proc. FoSSaCS*. Vol. 3921. LNCS. Springer, Mar. 2006, pp. 126–141.

[Bis13] Peter Bishop. "Does Software Have to Be Ultra Reliable in Safety Critical Systems?" In: *Computer Safety, Reliability, and Security*. Ed. by Friedemann Bitsch, Jérémie Guiochet, and Mohamed Kaâniche. Vol. 8153. LNCS. Springer, Jan. 2013, pp. 118–129.

[BLPK07] P. Bonet, C.M. Llado, R. Puijaner, and W.J. Knottenbelt. "PIPE v2.5: A Petri Net Tool for Performance Modelling." In: *Proc. CLEI*. 2007, pp. 91–106.

[BP96] Nadia Busi and G. Michle Pinna. "Non sequential semantics for contextual P/T nets." In: *Proc. ICATPN*. Ed. by Jonathan Billington and Wolfgang Reisig. Vol. 1091. LNCS. Springer, Jan. 1996, pp. 113–132.

[BR00] Thomas Ball and Sriram K. Rajamani. "Bebop: A Symbolic Model Checker for Boolean Programs." In: *Proc. SPIN*. Ed. by Klaus Havelund, John Penix, and Willem Visser. Vol. 1885. LNCS. Springer, 2000, pp. 113–130.

[Bry86] Bryant. "Graph-Based Algorithms for Boolean Function Manipulation." In: *IEEE Transactions on Computers* C-35.8 (Aug. 1986), pp. 677–691.

[CE82] Edmund M. Clarke and E. Allen Emerson. "Design and synthesis of synchronization skeletons using branching time temporal logic." In: *Proc. of Logics of Programs*. Ed. by Dexter Kozen. Vol. 131. LNCS. Springer, 1982, pp. 52–71.

[CGP99] E. M. Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, 1999.

[CGS09] Michael Codish, Samir Genaim, and Peter J. Stuckey. "A declarative encoding of telecommunications feature subscription in SAT." In: *Proc. PPDP*. ACM, 2009, pp. 255–266.

[CH93] Søren Christensen and Niels Damgaard Hansen. "Coloured Petri nets extended with place capacities, test arcs and inhibitor arcs." In: *Proc. ICATPN*. Ed. by Gerhard Goos, Juris Hartmanis, and Marco Ajmone Marsan. Vol. 691. LNCS. Springer, 1993, pp. 186–205.

[Che10] Jingchao Chen. "A new SAT encoding of the at-most-one constraint." In: *Proc. Constraint Modelling and Reformulation*. 2010.

[CKS07] Byron Cook, Daniel Kroening, and Natasha Sharygina. "Verification of Boolean programs with unbounded thread creation." In: *Theoretical Computer Science* 388.1–3 (Dec. 2007), pp. 227–242.

[CL08] Christos G. Cassandras and Stephane Lafortune. *Introduction to Discrete Event Systems*. Springer, 2008.

[Cla08] Edmund M. Clarke. "The Birth of Model Checking." In: *25 Years of Model Checking*. Ed. by Orna Grumberg and Helmut Veith. Vol. 5000. LNCS. Berlin, Heidelberg: Springer, 2008, pp. 1–26.

[Cor96] James C. Corbett. "Evaluating Deadlock Detection Methods for Concurrent Software." In: *IEEE Transactions on Software Engineering* 22 (1996), pp. 161–180.

[Cos] Cosyverif Project. COSYVERIF. http://www.cosyverif.org.

[Dij65] E. W. Dijkstra. "Solution of a problem in concurrent programming control." In: *Commun. ACM* 8.9 (Sept. 1965), 569ff.

[DJJ+12] Alexandre David, Lasse Jacobsen, Morten Jacobsen, Kenneth Yrke Jørgensen, Mikael H. Møller, and Jiri Srba. "TAPAAL 2.0: Integrated Development Environment for Timed-Arc Petri Nets." In: *Proc. TACAS*. 2012, pp. 492–497.

[EH01] Javier Esparza and Keijo Heljanko. "Implementing LTL Model Checking with Net Unfoldings." In: *Proc. SPIN*. Vol. 2057. LNCS. 2001, pp. 37–56.

[EH08] Javier Esparza and Keijo Heljanko. *Unfoldings - A Partial-Order Approach to Model Checking*. EATCS Monographs in Theoretical Computer Science. Springer, 2008.

[EK12] Javier Esparza and Christian Kern. "Reactive and Proactive Diagnosis of Distributed Systems Using Net Unfoldings." In: *Proc. ACSD*. 2012, pp. 154–163.

[EKS07] Javier Esparza, Pradeep Kanade, and Stefan Schwoon. "A negative result on depth-first net unfoldings." In: *International Journal on Software Tools for Technology Transfer* 10.2 (Feb. 2007), pp. 161–166.

[ER99] Javier Esparza and Stefan Römer. "An Unfolding Algorithm for Synchronous Products of Transition Systems." In: *Proc. CONCUR*. Vol. 1664. LNCS. Springer, 1999, pp. 2–20.

[ERV02] Javier Esparza, Stefan Römer, and Walter Vogler. "An Improvement of McMillan's Unfolding Algorithm." In: *Formal Methods in System Design* 20 (2002), pp. 285–310.

[ERV96] Javier Esparza, Stefan Römer, and Walter Vogler. "An improvement of McMillan's unfolding algorithm." In: *Proc. TACAS*. Ed. by Tiziana Margaria and Bernhard Steffen. Vol. 1055. LNCS. Springer, Jan. 1996, pp. 87–106.

[ES01] Javier Esparza and Claus Schröter. "Unfolding Based Algorithms for the Reachability Problem." In: *Fund. Inf.* 47.3-4 (2001), pp. 231–245.

[ES03] Niklas Eén and Niklas Sörensson. "An Extensible SAT-solver." In: *Proc. SAT*. LNCS 2919. 2003, pp. 502–518.

[Esp10] Javier Esparza. "A False History of True Concurrency: From Petri to Tools." In: *Proc. SPIN*. Ed. by Jaco van de Pol and Michael Weber. Vol. 6349. LNCS. Springer, 2010, pp. 180–186.

[FB07] Eric Fabre and Albert Benveniste. "Partial Order Techniques for Distributed Discrete Event Systems: Why You Cannot Avoid Using Them." In: *Discrete Event Dynamic Systems* 17.3 (Sept. 2007), pp. 355–403.

[FBHJ05] Eric Fabre, Albert Benveniste, Stefan Haar, and Claude Jard. "Distributed Monitoring of Concurrent and Asynchronous Systems." In: *Discrete Event Dynamic Systems* 15.1 (Mar. 2005), pp. 33–84.

[FPDN05] Alan M. Frisch, Timothy J. Peugniez, Anthony J. Doggett, and Peter Nightingale. "Solving Non-Boolean Satisfiability Problems with Stochastic Local Search: A Comparison of Encodings." In: *Journal of Automated Reasoning* 35.1-3 (2005), pp. 143–179.

[Fra86] Nissim Francez. *Fairness*. New York, NY, USA: Springer, 1986.

[Gar70] Martin Gardner. "Mathematical games: The fantastic combinations of John Conway's new solitaire game 'Life'." In: *Scientific American* 223.4 (1970), pp. 120–123.

[Ger04] Carlos Gershenson. "Introduction to Random Boolean Networks." In: *Proc. International Conference on the Simulation and Synthesis of Living Systems*. Ed. by M. Bedau, P. Husbands, T. Hutton, S. Kumar, and H. Suzuki. Boston, MA, USA, 2004, pp. 160–173.

[Gla96] Robert L. Glass. "Formal Methods are a Surrogate for a More Serious Software Concern." In: *Computer* 29.4 (Apr. 1996), p. 16.

[GM98] F. Gadducci and U. Montanari. "Axioms for contextual net processes." In: *Automata, Languages and Programming*. Ed. by Kim G. Larsen, Sven Skyum, and Glynn Winskel. Vol. 1443. LNCS. Springer, 1998, pp. 296–308.

[Gra] Graphviz Project. *Homepage*. http://www.graphviz.org/.

[Haa07] Stephan Haar. "Unfold and cover: Qualitative diagnosability for Petri Nets." In: *Proc. Conference on Decision and Control (CDC)*. 2007, pp. 1886–1891.

[Haa09] Stephan Haar. "Qualitative diagnosability of labeled petri nets revisited." In: *Proc. Conference on Decision and Control (CDC)*. 2009, pp. 1248–1253.

[Haa10] Stefan Haar. "Types of Asynchronous Diagnosability and the Reveals-Relation in Occurrence Nets." In: *IEEE Transactions on Automatic Control* 55.10 (2010), pp. 2310–2320.

[HB96] Michael Holloway and Ricky W. Butler. "Impediments to Industrial Use of Formal Methods." In: *Computer* 29.4 (Apr. 1996), p. 16.

[HBFJ03] Stephan Haar, Albert Benveniste, Eric Fabre, and Claude Jard. "Partial order diagnosability of discrete event systems using petri net unfoldings." In: *Proc. Conference on Decision and Control (CDC)*. Vol. 4. 2003, pp. 3748–3753.

[Hel99a] Keijo Heljanko. "Deadlock and reachability checking with finite complete prefixes." Licentiate's thesis. Helsinki University of Technology, 1999.

[Hel99b] Keijo Heljanko. "Minimizing Finite Complete Prefixes." In: *Proc. of the Workshop Concurrency, Specification & Programming 1999*. Ed. by H.-D. Burkhard, L. Czaja, H.-S. Nguyen, and P. Starke. Warsaw, Poland: Warsaw University, Sept. 1999, pp. 83–95.

[Hel99c] Keijo Heljanko. "Using Logic Programs with Stable Model Semantics to Solve Deadlock and Reachability Problems for 1-Safe Petri Nets." In: *Fund. Inf.* 37.3 (1999), pp. 247–268.

[Hel99d] Keijo Heljanko. "Using Logic Programs with Stable Model Semantics to Solve Deadlock and Reachability Problems for 1-Safe Petri Nets." In: *Proc. TACAS*. Ed. by W. Rance Cleaveland. Vol. 1579. LNCS. Springer, 1999, pp. 240–254.

[HF13] Stefan Haar and Éric Fabre. "Diagnosis with Petri Net Unfoldings." In: *Control of Discrete-Event Systems*. Ed. by Carla Seatzu, Manuel Silva, and Jan H. van Schuppen. Vol. 433. LNCIS. Springer, 2013, pp. 301–318.

[HKS13] Stefan Haar, Christian Kern, and Stefan Schwoon. "Computing the reveals relation in occurrence nets." In: *Theoretical Computer Science* 493 (July 2013), pp. 66–79.

[HRS13] Stefan Haar, César Rodríguez, and Stefan Schwoon. "Reveal Your Faults: It's Only Fair!" In: *2013 13th International Conference on Application of Concurrency to System Design (ACSD)*. 2013, pp. 120–129.

[JK91] Ryszard Janicki and Maciej Koutny. "Invariant Semantics of Nets with Inhibitor Arcs." In: *Proc. CONCUR*. Ed. by Jos C. M. Baeten and Jan Frisco Groote. Vol. 527. LNCS. 1991, pp. 317–331.

[JK93] Ryszard Janicki and Maciej Koutny. "Structure of concurrency." In: *Theoretical Computer Science* 112.1 (Apr. 1993), pp. 5–52.

[JK95] R. Janicki and M. Koutny. "Semantics of Inhibitor Nets." In: *Information and Computation* 123.1 (Nov. 1995), pp. 1–16.

[Kho] Victor Khomenko. PUNF. http://homepages.cs.ncl.ac.uk/victor.khomenko/tools/punf/.

[Kho03] Victor Khomenko. "Model Checking Based on Prefixes of Petri Net Unfoldings." PhD thesis. School of Computing Science, Newcastle University, 2003.

[KK00] Victor Khomenko and Maciej Koutny. "LP Deadlock Checking Using Partial Order Dependencies." In: *Proc. CONCUR*. LNCS 1877. 2000, pp. 410–425.

[KK01] Victor Khomenko and Maciej Koutny. "Towards an Efficient Algorithm for Unfolding Petri Nets." In: *Proc. CONCUR*. LNCS 2154. 2001, pp. 366–380.

[KK07] Victor Khomenko and Maciej Koutny. "Verification of bounded Petri nets using integer programming." In: *Formal Methods in System Design* 30.2 (2007), pp. 143–176.

[KKKV06] Victor Khomenko, Alex Kondratyev, Maciej Koutny, and Walter Vogler. "Merged Processes – a New Condensed Representation of Petri Net Behaviour." In: *Acta Inf.* 43.5 (2006), pp. 307–330.

[KKV03] Victor Khomenko, Maciej Koutny, and Walter Vogler. "Canonical prefixes of Petri net unfoldings." In: *Acta Informatica* 40.2 (2003), pp. 95–118.

[KLB+13] Fabrice Kordon, Alban Linard, Marco Becutti, Didier Buchs, Lukasz Fronc, Francis Hulin-Hubard, Fabrice Legond-Aubry, Niels Lohmann, A. Marechal, Emmanuel Paviot-Adet, Franck Pommereau, César Rodríguez, Christian Rohr, Yann Thierry-Mieg, Haro Wimmel, and Karsten Wolf. *Web Report on the Model Checking Contest @ Petri Net 2013*. June 2013. URL: mcc.lip6.fr.

[KM11] V. Khomenko and A. Mokhov. "An Algorithm for Direct Construction of Complete Merged Processes." In: *Proc. ICATPN*. LNCS 6709. 2011, pp. 89–108.

[KM13] Victor Khomenko and Andrey Mokhov. "Direct Construction of Complete Merged Processes." In: *The Computer Journal* (Feb. 2013).

[Kni02] John C. Knight. "Safety critical systems: challenges and directions." In: *Proc. ICSE*. ACM Press, 2002, p. 547.

[LIP] LIP6/MoVe Team. COLOANE. http://coloane.lip6.fr/.

[LS09] Martin Leucker and Christian Schallhart. "A brief account of runtime verification." In: *The Journal of Logic and Algebraic Programming* 78.5 (May 2009), pp. 293–303.

[McM93a] Kenneth L. McMillan. *Symbolic model checking*. Kluwer, 1993.

[McM93b] Kenneth L. McMillan. "Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits." In: *Proc. CAV'92*. Ed. by Gregor von Bochmann and David Karl Probst. Vol. 663. LNCS. Springer, 1993, pp. 164–177.

[McM95a] Kenneth L. McMillan. "Trace theoretic verification of asynchronous circuits using unfoldings." In: *Proc. CAV*. Ed. by Pierre Wolper. Vol. 939. LNCS. Springer, 1995, pp. 180–195.

[McM95b] K. L. McMillan. "A technique of state space search based on unfolding." In: *Form. Methods Syst. Des.* 6.1 (1995), pp. 45–65.

[Mer97] Stephan Merkel. *Verification of Fault Tolerant Algorithms Using PEP*. Tech. rep. 1997.

[MR94] Ugo Montanari and Francesca Rossi. "Contextual occurrence nets and concurrent constraint programming." In: *Proc. ICGT*. Vol. 776. LNCS. 1994, pp. 280–295.

[MR95] Ugo Montanari and Francesca Rossi. "Contextual nets." In: *Acta Informatica* 32.6 (June 1995), pp. 545–596.

[MR97] Stefan Melzer and Stefan Römer. "Deadlock Checking Using Net Unfoldings." In: *Proc CAV*. Vol. 1254. LNCS. 1997, pp. 352–363.

[Mur89] Tadao Murata. "Petri nets: Properties, analysis and applications." In: *Proc. of the IEEE* 77.4 (Apr. 1989), pp. 541–580.

[NPW81] Mogens Nielsen, Gordon Plotkin, and Glynn Winskel. "Petri nets, event structures and domains, part I." In: *Theoretical Computer Science* 13.1 (1981), pp. 85–108.

[Par10] David Lorge Parnas. "Really Rethinking 'Formal Methods'." In: *Computer* 43.1 (Jan. 2010), pp. 28–34.

[Pel93] Doron Peled. "All from one, one for all: on model checking using representatives." In: *Proc. CAV*. Ed. by G. Goos, J. Hartmanis, and Costas Courcoubetis. Vol. 697. Springer, 1993, pp. 409–423.

[PEP] PEP Project. *PEP homepage*. http://theoretica.informatik.uni-oldenburg.de/~pep/.

[Pet66] Carl Adam Petri. "Communication with automata." PhD Thesis. Universität Hamburg, 1966.

[Pet77] Carl Adam Petri. *Non-sequential Processes*. Research Report 77-05. Bonn: GMD-ISF, 1977.

[Pet81] J.L. Peterson. *Petri Nets and the Modeling of Systems.* Englewood Cliffs, New Jersey: Prentice-Hall, 1981.

[PNW12] Lee Pike, Sebastian Niller, and Nis Wegmann. "Runtime Verification for Ultra-Critical Systems." In: *Proc. Runtime Verification*. Ed. by Sarfraz Khurshid and Koushik Sen. Vol. 7186. LNCS. Springer, Jan. 2012, pp. 310–324.

[PR08] Carl Adam Petri and Wolfgang Reisig. "Petri net." In: *Scholarpedia* 3.4 (2008), p. 6477.

[QS82] J. P. Queille and J. Sifakis. "Specification and verification of concurrent systems in CESAR." In: *International Symposium on Programming*. Ed. by Mariangiola Dezani-Ciancaglini and Ugo Montanari. Vol. 137. LNCS. Springer, 1982, pp. 337–351.

[Ray86] M. Raynal. *Algorithms for Mutual Exclusion*. MIT Press, 1986.

[Rei13] Wolfgang Reisig. *Understanding petri nets modeling techniques, analysis methods, case studies*. Berlin: Springer, 2013.

[Ris94] Gioia Ristori. "Modelling Systems with Shared Resources via Petri Nets." PhD thesis. Department of Computer Science, University of Pisa, 1994.

[Rod10] César Rodríguez. "Implementation of a complete prefix unfolder for contextual nets." Master thesis. MPRI, Paris, Sept. 2010.

[RS12a] César Rodríguez and Stefan Schwoon. *Verification of Petri Nets with Read Arcs*. Tech. rep. LSV-12-12. LSV, ENS de Cachan, France, 2012.

[RS12b] César Rodríguez and Stefan Schwoon. "Verification of Petri Nets with Read Arcs." In: *CONCUR 2012 – Concurrency Theory*. Ed. by Maciej Koutny and Irek Ulidowski. Vol. 7454. LNCS. Springer, Sept. 2012, pp. 471–485.

[RS13a] César Rodríguez and Stefan Schwoon. "An Improved Construction of Petri Net Unfoldings." In: *Proc. of the French-Singaporean Workshop on Formal Methods and Applications (FSFMA'13)*. Vol. 31. OASICS. Leibniz-Zentrum für Informatik, July 2013, pp. 47–52.

[RS13b] César Rodríguez and Stefan Schwoon. "Cunf: A Tool for Unfolding and Verifying Petri Nets with Read Arcs." In: *Automated Technology for Verification and Analysis (ATVA)*. Ed. by Dang Van Hung and Mizuhito Ogawa. Vol. 8172. LNCS. Springer, 2013, pp. 492–495.

[RSB11a] César Rodríguez, Stefan Schwoon, and Paolo Baldan. *Efficient contextual unfolding*. Tech. rep. LSV-11-14. This is a full version of [RSB11b]. LSV, ENS de Cachan, France, 2011.

[RSB11b] César Rodríguez, Stefan Schwoon, and Paolo Baldan. "Efficient contextual unfolding." In: *CONCUR 2011 – Concurrency Theory*. Ed. by Joost-Pieter Katoen and Barbara König. Vol. 6901. LNCS. Springer, Sept. 2011, pp. 342–357.

[RSK13] César Rodríguez, Stefan Schwoon, and Victor Khomenko. "Contextual Merged Processes." In: *Proc. International Conference on Application and Theory of Petri Nets and Concurrency (ICATPN)*. Vol. 7927. LNCS. June 2013, pp. 29–48.

[RTM04] Darsh P. Ranjan, Daijue Tang, and Sharad Malik. "A Comparative Study of 2QBF Algorithms." In: *Proc. SAT*. 2004.

[Sch] Stefan Schwoon. MOLE *Homepage*. http://www.lsv.ens-cachan.fr/~schwoon/tools/mole/.

[Sch06] Claus Schröter. "Halbordnungs- und Reduktionstechniken für die automatische Verifikation von verteilten Systemen." PhD thesis. Universität Stuttgart, 2006.

[Sin05] Carsten Sinz. "Towards an Optimal CNF Encoding of Boolean Cardinality Constraints." In: *Proc. CP*. 2005, pp. 827–831.

[SR11] Stefan Schwoon and César Rodríguez. "Construction and SAT-based verification of Contextual Unfoldings." In: *Proc. of the 13th International Workshop on Descriptional Complexity of Formal Systems (DCFS'11)*. Vol. 6808. LNCS. Extended abstract. Springer, July 2011, pp. 34–42.

[SS96] João P. Marques Silva and Karem A. Sakallah. "GRASP - a new search algorithm for satisfiability." In: *Proc. ICCAD*. 1996, pp. 220–227.

[SSL+95] Meera Sampath, Raja Sengupata, Stéphane Lafortune, Kasim Sinnamohideen, and Demosthenis Teneketzis. "Diagnosability of discrete-event systems." In: *IEEE Transactions on Automatic Control* 40.9 (1995), pp. 1555–1575.

[Val98] Antti Valmari. "The State Explosion Problem." In: *Lectures on Petri Nets I: Basic Models*. Vol. 1491. LNCS. Springer, 1998, pp. 429–528.

[Vog95] Walter Vogler. "Fairness and Partial Order Semantics." In: *Inf. Process. Lett.* 55.1 (1995), pp. 33–39.

[Vog97] Walter Vogler. "Partial order semantics and read arcs." In: *Proc. MFCS*. Ed. by Igor Prívara and Peter Ružička. Vol. 1295. LNCS. Springer, 1997, pp. 508–517.

[VSY98] Walter Vogler, Alexei L. Semenov, and Alexandre Yakovlev. "Unfolding and Finite Prefix for Nets with Read Arcs." In: *Proc. CONCUR*. Vol. 1466. LNCS. 1998, pp. 501–516.

[Was12] Alan Wassyng. "Who Are We, and What Are We Doing Here?" In: *Proc. FM*. Ed. by Dimitra Giannakopoulou and Dominique Méry. Vol. 7436. LNCS. Springer, Jan. 2012, pp. 7–9.

[Wik13] Wikipedia. *Petri net — Wikipedia, The Free Encyclopedia*. [Online; accessed 18-April-2013] http://en.wikipedia.org/w/index.php?title=Petri_net&oldid=550780771. 2013.

[Win02] Józef Winkowski. "Reachability in Contextual Nets." In: *Fund. Inf.* 51.1–2 (2002), pp. 235–250.

[Win98] Józef Winkowski. "Processes of Contextual Nets and their Characteristics." In: *Fundamenta Informaticae* 36.1 (1998), pp. 71–101.

LIST OF FIGURES

Figure 1 A Petri net modelling a mutual exclusion algorithm. 4
Figure 2 Non-sequential semantics of the Petri net in Fig. 1. . 6
Figure 3 The finite, complete unfolding prefix constructed by McMillan's algorithm for Fig. 1. 7
Figure 4 A Petri net with exponentially large unfolding prefix. 8
Figure 5 Three models of a system exhibiting concurrent read access, and their unfoldings. 9
Figure 6 Contextual nets where steps semantics disagree. . . . 10
Figure 7 A c-net with four read arcs. 16
Figure 8 Illustrating causality, symmetric, and asymmetric conflict. 18
Figure 9 (a) A c-net; (b) its *plain encoding* N'; (c) and its *Place-Replication encoding* N'' 21
Figure 10 Two occurrence nets illustrating a circular asymmetric conflict of length 3 (a) and two (b). 23
Figure 11 Unfolding prefix of the c-net in Fig. 7; the labelling is given in parenthesis. 27
Figure 12 McMillan's algorithm fails in contextual nets. 32
Figure 13 (a) a c-net; (b) a prefix of its unfolding. Event e_2 has infinitely many histories. 33
Figure 14 A c-net (a) whose unfolding (b) is exponentially larger than the set of reachable markings. 37
Figure 15 Predecessors w.r.t. asymmetric conflict of an event e. 44
Figure 16 The characterization in § 3.7 allows multiple constructions of the enriched event $\langle e, \{e_1, \ldots, e_n, e\}\rangle$. 51
Figure 17 Good examples for the eager (a) and the lazy (b) approach. 56
Figure 18 Safe c-net and unfolding prefix. 62
Figure 19 Partial unfolding of Dekker's algorithm with asymmetric cycles. 65
Figure 20 An occurrence net of size $\mathcal{O}(k)$ where $|\nearrow\!\!\!\nearrow| = k^2$. . . . 66
Figure 21 Stubborn events. 68
Figure 22 Good (a) and bad (b) cases for the stubborn event optimization 70
Figure 23 A Petri net with exponentially large unfolding prefix. 73
Figure 24 Contextual Merged Processes are compact. 74
Figure 25 (a) A net; (b) its unfolding; (c) its unravelling. 76
Figure 26 The c-net 2-Gen. 79
Figure 27 Illustrating the digraph Δ_Q. 86
Figure 28 Running example of Ch. 6 — the Petri net. 94
Figure 29 Running example of Ch. 6 — the unfolding. 95
Figure 30 Unobservable events in verbose configurations. . . . 102
Figure 31 Pairs of independent readers 120
Figure 32 The net 2-Dijkstra. 128
Figure 33 (a) a c-net; (b) its encoding into a Petri net; (c) unfolding of (b) 136
Figure 34 A variant of Dekker's mutual exclusion algorithm for two processes 139

Figure 35 The unfolding of Fig. 34. 142
Figure 36 The unfolding of `examples/dijkstra/dij02.ll_net`. 144
Figure 37 Example of the `ll_net` format with read arcs, and graphical representation of the encoded c-net. 146
Figure 38 The Coloane graphical editor can export c-nets for the Cunf Toolset. 148
Figure 39 A c-net programmatically produced with the `ptnet` Python module. 149

LIST OF TABLES

Table 1 Enriched events of Fig. 12. 35
Table 2 Some enriched events from Fig. 13. 36
Table 3 Size of $\phi_{\mathcal{P}}^{\text{dead}}$ and $\phi_{\mathcal{P}}^{\text{cov},M}$. See text for interpretation. . 67
Table 4 Growth of contextual, plain, and PR unfoldings and MPs for the collection of c-nets n-Gen. 80
Table 5 Reduction of the asymmetric-conflict relation. 118
Table 6 Experimental results of Cunf. 122
Table 7 Experimental results obtained with Cna. 124
Table 8 Comparing unfoldings, MPs, and CMPs. 127
Table 9 Unfolding and merging sizes of plain and PR encodings of n-Dijkstra. 129