On Conflicts in Concurrent Systems

A thesis

submitted in partial fullfilment

of the requirements for the degree

of

Doctor of Philosophy

at

The University of Waikato

by

SIMON WARE

THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

2013

Abstract

This dissertation studies conflicts. A conflict is a bug in concurrent systems where one or more components of the system may potentially be blocked from completing their task. This dissertation investigates how nonconflicting completions may be used to characterise the situations in which individual components of a system may be in conflict with other components. The first major contributions of this dissertation are new methods of abstracting systems with respect to conflicts, and showing how these methods may be used to check whether a large system is conflict-free. The second contribution is a method of comparing whether one system is less susceptible to conflict than another. The last major contribution is a method of expressing all conflicts in a system in a finite and canonical way. The methods developed have applications for model checking, refinement, and the development of contracts for concurrent systems.

Contents

Chapter 1

Introduction

As the world becomes more and more dependant on electronic systems, there is an increasing need for strong and robust mathematical models, to describe, to understand, and to ensure the correctness of these systems. Such models have existed for years for most engineering tasks, helping to ensure outcomes which are, correct, and safe. The world of computing however has been slow to catch up to engineering in this regard, to a large extent relying upon testing to ensure correctness rather than stringent proofs.

One of the key difficulties in describing software is that in general its expected behaviour changes based upon the context it is in, often requiring different outcomes dependant upon different inputs. This can often make it hard to describe what a software or hardware system is supposed to do in formal terms. One method of describing software and hardware systems and how their behaviour evolves over time is discrete event systems (DES). Here the behaviour of a system is described using finite state automata (FSA). DESs are also used to describe concurrency, where seperate processes of a system are represented as seperate FSAs which syncronise on shared events.

DESs can then be examined for common design faults. One such fault is the problem of blocking and conflicts. A system is considered to be blocking, if it is possible for the system to reach a state where it is no longer capable of performing any further productive activity. This is analagous to asking whether the model is free of possible deadlock and/or livelock situations. In contrast, two or more components of a system are considered to be in conflict, if when run concurrently the components are blocking.

The naive way of determining whether a model is nonblocking is by manually exploring the automaton constructed by composing all the automata in the model together. This is called the monolithic method. Unfortunately in the general case, the size of this automaton grows exponentially with the number of FSAs in the model, thus making its construction intractable for larger models. In fact the problem of verifying whether or not a DES is nonblocking is NP-Hard [14].

Large models which would be impossible to check using the monolithic approach in many cases can be checked using a compositional conflict checker [11]. A compositional model checker will iteratively compose a subset of a model, apply abstraction techniques to simplify the resulting automaton, then repeat this process until the entire model has been composed. Because at each step the model is simplified in most cases it is never necessary for a compositional checker to compose an automaton which is the same size as the monolithic approach. In order for a compositional checker to give the correct result it must use abstraction techniques which ensures that the simplified automaton is equivalent to the original automaton with respect to the behaviour which is being tested. Conflict equivalence is the best equivalence relation for compositional nonblocking verification. Two automata are considered conflict equivalent if they reach a state of conflict under exactly the same conditions. Conflict equivalence was first introduced in [25].

This thesis studies the conflict equivalence relation and how it can be used to abstract automata, as well as to gain a greater understanding of what makes an automaton conflict with other automata. [11] develops several methods of simplifying automata which preserved conflict equivalence in order to verify the nonblocking property. Before the work carried out in this thesis it was not understood how two processes could be compared to one another in order to determine whether they were conflict equivalent, nor was it understood how to derive a unique automaton in order to represent a given conflict equivalence class.

Conflict equivalence is similiar to many other equivalence relations which can be used to simplify finite state machines. The most commonly used equivalence relation is language equivalence. Language equivalence determines whether two automata are equivalent to each other based solely upon whether they are capable of performing exactly the same sequence of events, i.e. if they both have the same language. It is currently well known how to abstract and compare automata with respect to language equivalence. [18] outlines how any finite state automaton can be converted into a language equivalent automaton which is both minimal and unique using subset construction and minimization. Abstraction with respect to language equivalence was used in [34] in order to verify safety properties. Unfortunately language equivalence does not preserve all the information necessary to determine whether two automata will be conflicting with one another.

Bisimulation considers two automata equivalent to one another if they both have equal nondeterministic branching behaviour [26]. Bisimulation is one of the finest known behavioural equivalences. Bisimulation preserves all temporal logic properties including nonblocking. This means that bisimulation is stronger than conflict equivalence, that is to say any two automata which are considered equivalent with respect

to bisimulation are equivalent with respect to conflict equivalence, but not vice versa. Bisimulation is a well-understood equivalence relation which has fast algorithms capable of simplifying automata with respect to it.

Another equivalence relation which is used to compare automata is that of failures equivalence [26]. Two automata are failures equivalent if they will reach a deadlock situation, or fail, under exactly the same situations. Failures equivalence is also a well-understood equivalence relation and it has been known for a long time how to find a minimal automaton representation of any particular failures equivalence class. Unfortunately failures equivalence is not adequate to reflect all the conflict information contained within an automaton, as it does not preserve livelocks. The ideas used to represent failures equivalence can however be extended to reflect conflicts.

Fair testing [2] is the closest equivalence relation to conflict equivalence. Fair testing differs from conflict equivalence in the respect that only the test automaton can determine when the test has passed. This makes fair testing a stronger eqivalence relation. It is currently understood how to compare two automata with respect to fair testing, however the algorithm has never been implemented to our knowledge, and it is not understood how to simplify with respect to fair testing in a general way.

1.1 Contributions

The most important contributions of this thesis are

- New methods of abstracting automata with respect to conflict equivalence.
- A finite canonical characterization of an automaton's generalised nonblocking equivalence class.
- An algorithm to calculate whether an automaton is less conflicting to another.
- A finite canonical characterization of an automaton's conflict equivalence class.

1.2 Outline

This thesis is divided into chapters. Chapter 2 describes the notation used throughout this thesis as well as key concepts. It introduces finite state automata, parallel composition, conflict equivalence, nonconflicting completions, and compositional verification.

In Chapter 3 a new method of abstracting automata with respect to conflict equivalence is introduced. This method converts the automaton by marking states with annotations, which are similar to the failures sets used in failures equivalence. The annotated

automaton is then simplified using abstraction rules which make use of the annotations. In addition to this we show how the method can be used in a conflict checker in order to verify whether discrete event models are nonblocking, and give experimental results of its use. This chapter is based upon work published in [35, 38].

Chapter 4 investigates the related problem of generalised nonblocking equivalence. Generalised nonblocking adds to standard nonblocking the ability to restrict the set of states from which blocking is checked. This improves the expressive power of nonblocking but makes it so that less information can be abstracted from an automaton. Because of this it turns out that generalised nonblocking equivalence is in fact easier to charaterise than conflict equivalence. This shows how to compare automata with respect to generalised nonblocking and proposes a normal form which can be used to represent automata with respect to generalised nonblocking. In addition to this it shows experimental results derived from using this normal form to verify models with respect to generalised nonblocking. This chapter is based upon work published in [36].

Chapter 5 builds upon the understanding of conflicts developed in the previous two chapters to demonstrate how two automata can be compared to one another with respect to conflict equivalence. It also shows experimental results from using the algorithm developed to compare automata. In addition it is shown that the algorithm for comparing automata with respect to conflict equivalence can be used to compare automata with respect to fair testing, and that the algorithm has lower time complexity. This chapter is based upon work published in [37].

Chapter 6 describes the conflict normal form which is a canonical representation of a given conflict equivalence class. As the conflict normal form represents a unique representation of any given conflict equivalence class it can be considered to be a form which keeps only that information which is relevant to conflicts. Because of this it has the potential to be used as a powerful abstraction for verifying nonblocking. In addition to this however it can be used to be able to understand better what exactly makes a process conflict with other processes. The work covered in this chapter has yet to be published.

Chapter 2

Preliminaries

This chapter introduces the notations used throughout this thesis. Discrete event systems are modelled using automata, with the possibility of nondeterminism, which naturally arises from abstraction and hiding [16, 32]. System behaviour is described using languages, with notations taken from the background of discrete event systems and automata theory [18, 30]. In section 2.1 it is shown what an event is and how they can be concatenated to form traces and languages. Section 2.2 describes how a finite state automaton is defined and how they relate to languages and traces in this thesis. Furthermore section 2.3 describes several operations which are commonly used upon finite state automata throughout this thesis. Next section 2.4 describes the conflict equivalence relation which is used throughout this thesis. The concept of certain conflicts is described in section 2.5. In addition section 2.6 describes how conflict equivalence can be encapsulated using nonconflicting completions. Finally section 2.7 describes how abstracting an automaton while preserving conflict equivalence can be used to verify whether a large model is nonblocking.

2.1 Events and Traces

Event sequences and languages are a simple means to describe discrete system behaviours. Their basic building blocks are *events*, which are taken from a finite *alphabet* Σ. Two special events are used, the *silent event* τ and the *termination event* ω. These are never included in an alphabet Σ unless mentioned explicitly. For this, $\Sigma_\tau = \Sigma \cup \{\tau\}$, $\Sigma_\omega = \Sigma \cup \{\omega\}$, and $\Sigma_{\tau,\omega} = \Sigma \cup \{\tau, \omega\}$ are used. The silent event τ represents behaviour which is local to the automaton in which it can occur, as such other automata in the model can neither block or observe τ events. The termination event ω represents termination, when an ω event occurs that means that the model has successfully terminated.

Σ^* denotes the set of all finite *traces* of the form $\sigma_1\sigma_2\cdots\sigma_n$ of events from Σ, including the *empty trace* ε. The *concatenation* of two traces $s,t \in \Sigma^*$ is written as st. A subset $\mathscr{L} \subseteq \Sigma^*$ is called a *language*. The trace s is a *prefix* of t if if there exists a trace u such that $su = t$. This thesis uses the notation $s \sqsubseteq t$ to signify that s is a prefix of t. Traces and languages can also be catenated, for example $sL = \{\, st \in \Sigma^* \mid t \in L \,\}$. *Natural projection* $P_\tau \colon \Sigma_\tau^* \to \Sigma^*$ is the operation that deletes all silent (τ) events from traces. *Prefix-closure* is the operation which saturates a language L such that for every trace $s \in L$ if the trace t is a prefix of s then t is also in the prefix-closure of L. Thus $\overline{L} = \{t \mid \forall s \in L \text{ such that } t \sqsubseteq s\}$.

Language *derivation* [4] is the operation which describes the behaviour of a language after a given trace similiar to language derivation. The language L derived by the trace s consists of all the traces which L can perform after the trace s. Thus $L/s = \{t \mid st \in L\}$.

2.2 Nondeterministic Automata

In this thesis, process behaviour is modelled using nondeterministic *labelled transition systems* or *automata* $A = \langle \Sigma, Q, \to, Q^\circ \rangle$, where Σ is a finite alphabet of *events*, Q is a set of *states*, $\to \subseteq Q \times \Sigma_{\tau,\omega} \times Q$ is the *state transition relation*, and $Q^\circ \subseteq Q$ is the set of *initial states*. A is called *finite-state* if its state set Q is finite.

The transition relation is written in infix notation $x \xrightarrow{\sigma} y$, and extended to traces by letting $x \xrightarrow{\varepsilon} x$ for all $x \in Q$, and $x \xrightarrow{s\sigma} y$ if $x \xrightarrow{s} z \xrightarrow{\sigma} y$ for some $z \in Q$. The transition relation must satisfy the additional requirement that, whenever $x \xrightarrow{\omega} y$, there does not exist any outgoing transition from y. The automaton A is *deterministic* if $|Q^\circ| \leq 1$ and the transition relation contains no transitions labelled τ, and if $x \xrightarrow{\sigma} y_1$ and $x \xrightarrow{\sigma} y_2$ always implies $y_1 = y_2$.

A state $x \in Q$ is considered accepting if $x \xrightarrow{\omega} y$ for some state $y \in Q$. This is slightly different from standard automata which have the additional state set Q^m which contains all accepting states. The definition allows many equivalence relations such as bisimulation and observation equivalence to be expressed more concisely and otherwise is equivalent to the standard mark state definition of automata.

To support silent transitions, $x \stackrel{s}{\Rightarrow} y$, with $s \in {\Sigma_\tau}^*$, denotes the existence of a trace $t \in {\Sigma_{\tau,\omega}}^*$ such that $x \xrightarrow{t} y$, and s is obtained from t by deleting all τ events. For a state set $X \subseteq Q$ and a state $y \in Q$, the expression $X \stackrel{s}{\Rightarrow} y$ denotes the existence of $x \in X$ such that $x \stackrel{s}{\Rightarrow} y$, and $A \stackrel{s}{\Rightarrow} y$ means that $Q^\circ \stackrel{s}{\Rightarrow} y$. Furthermore, $x \Rightarrow y$ denotes the existence of a trace s such that $x \stackrel{s}{\Rightarrow} y$, and $x \stackrel{s}{\Rightarrow}$ denotes the existence of a state $y \in Q$ such that $x \stackrel{s}{\Rightarrow} y$.

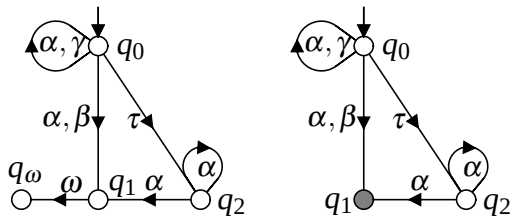


Figure 2.1: Graphical representations of two equivalent automata

For a state or state set x, the *continuation language* is defined as

$$\mathscr{L}(x) \;=\; \{\, s \in \Sigma^* \mid x \overset{s}{\Rightarrow} \} \;, \tag{2.1}$$

and likewise the *ω-marked language* is

$$\mathscr{L}^{\omega}(x) = \{\, s\omega \in \Sigma^*\omega \mid x \overset{s\omega}{\Rightarrow} \} \;. \tag{2.2}$$

The language and the ω-marked language of the automaton G are $\mathscr{L}(G) = \mathscr{L}(Q^{\circ})$ and $\mathscr{L}^{\omega} = \mathscr{L}^{\omega}(Q^{\circ})$. This is similar to the notion of the language recognized by an automaton [16] with the addition of an ω event. Lastly we define the *eligible event set* of a state. This is the set of events which a state allows to occur.

$$Elig(x) = \{\sigma | x \overset{\sigma}{\Rightarrow} y \text{ where } \sigma \in \Sigma_{\omega}\} \tag{2.3}$$

States are represented as circles and transitions as arrows between a source and target state. The names of states and the events associated with transitions are represented using floating text located next to their associated state or transition. If there are two or more transitions with the same source and target state the same arrow is used to represent all such transitions with all the events of those transitions listed next to the arrow. Unless otherwise stated the alphabet of such an automaton is assumed to be the union of all events associated with transitions in the graph with the exception of the termination event ω. In other words it is assumed that the alphabet does not contain any events which can never be executed by the automaton, unless otherwise stated. Initial states are identified as states which have an arrow with no source entering them. Finally sometimes automata are represented using marked states instead of ω transitions in order to make the graphical representation more concise, in this case grayed out circles represent a marked state.

Example 2.1 Figure 2.2 shows two representations of the same automaton. One explicitly shows an ω-transition to represent termination, the other uses a colored in ac-

cepting state. Both have the same meaning.

2.3 Operations

The process-algebraic operations of synchronous composition and hiding are used in this thesis to compose automata. *Synchronous composition* models the parallel execution of two or more automata, and is done using lock-step synchronisation in the style of [16].

Definition 2.1 Let $G = \langle \Sigma, Q_G, \rightarrow_G, Q_G^\circ \rangle$ and $H = \langle \Sigma, Q_H, \rightarrow_H, Q_H^\circ \rangle$ be automata. The *synchronous product* of G and H is

$$G \,\|\, H = \langle \Sigma, Q_G \times Q_H, \rightarrow, Q_G^\circ \times Q_H^\circ \rangle \tag{2.4}$$

where

$$\begin{array}{ll} (x_G, x_H) \xrightarrow{\sigma} (y_G, y_H) & \text{if } \sigma \in \Sigma,\ x_G \xrightarrow{\sigma}_G y_G, \text{ and} \\ & x_H \xrightarrow{\sigma}_H y_H; \\ (x_G, x_H) \xrightarrow{\tau} (y_G, x_H) & \text{if } x_G \xrightarrow{\tau}_G y_G; \\ (x_G, x_H) \xrightarrow{\tau} (x_G, y_H) & \text{if } x_H \xrightarrow{\tau}_H y_H; \end{array}$$

In synchronous composition, shared events (including ω) must be executed by both automata synchronously, while other events (including τ) are executed independently. In the notation of this thesis,

$$G_1 \,\|\, G_2 \stackrel{s}{\Rightarrow} (x_1, x_2) \qquad \text{if and only if} \qquad G_i \stackrel{P_i(s)}{\Longrightarrow} x_i \quad \text{for } i = 1, 2\ , \tag{2.5}$$

where $P_i \colon \Sigma \to \Sigma_i$ denotes the natural projection.

Automata with different alphabets can also be composed by lifting them to a common alphabets first: when an event σ is added to the alphabet Σ, selfloop transitions $x \xrightarrow{\sigma} x$ are added for all states $x \in Q$. Other than chapter 3 it is assumed that automata are always lifted to a common alphabet before composition.

It is easily confirmed that synchronous composition is a commutative and associative operation.

Hiding is the process-algebraic operation that generalises natural projection of languages when nondeterministic automata are considered [2]. Events that are not of interest are replaced by silent (τ) transitions or *ε-moves* [18].

Definition 2.2 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an automaton, and let $\Upsilon \subseteq \Sigma$. The result of *hiding* Υ in G is

$$G \setminus \Upsilon \;=\; \langle \Sigma \setminus \Upsilon, Q, \rightarrow \setminus \Upsilon, Q^\circ \rangle\ , \tag{2.6}$$

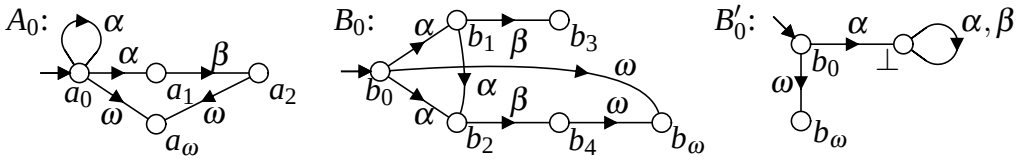


Figure 2.2: Examples of blocking and nonblocking automata.

where $\rightarrow \setminus \Upsilon$ is obtained from $\rightarrow$by replacing all events in Υ with the silent event τ.

Automata *derivation* is an operation which describes the behaviour of an automaton after a given trace similiar to language derivation. The automaton G derived by the trace s is identical to the automaton G except that its initial state set consists of the set states G can reach after the state s.

Definition 2.3 For $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ and $s \in \Sigma^*$, define $G/s = \langle \Sigma, Q, \rightarrow, Q_s^\circ \rangle$ where $Q_s^\circ = \{ x \in Q \mid G \stackrel{s}{\Rightarrow} x \}$.

The state set Q_s° can be calculated using subset construction.

2.4 Conflict Equivalence

The key liveness property in supervisory control theory [30] is the *nonblocking* property. Given an automaton A, it is desirable that every trace in $\mathscr{L}(A)$ can be completed to a trace in $\mathscr{L}^\omega(A)$, otherwise A may become unable to terminate. A process that may become unable to terminate is called *blocking*. This concept becomes more interesting when multiple processes are running in parallel—in this case the term *conflicting* is used instead. In this thesis we use a modified version of the nonblocking property presented in [30] which can be applied to nondeterministic automata [**?**].

Definition 2.4 An automaton $A = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ is *nonblocking* if for every state $x \in Q$ and every trace $s \in \Sigma^*$ such that $Q^\circ \stackrel{s}{\Rightarrow} x$ it holds that $\mathscr{L}^\omega(x) \neq \emptyset$. Otherwise A is *blocking*. A state x such that $\mathscr{L}^\omega(x) = \emptyset$ is a *blocking state*. Two automata A and B are *nonconflicting* if $A \parallel B$ is nonblocking, otherwise they are *conflicting*.

Example 2.2 Automaton A_0 in figure 2.2 is nonblocking, as for every state $x \in Q$ which is reachable using a trace $s \in \Sigma^*$ it is always possible to reach the state a_2 and terminate. As a_ω can only be reached after the ω event which is not in Σ the fact that a_ω is blocking does not make A_0 blocking. Automaton B_0 on the other hand is blocking, because it can enter state b_3 after executing $\alpha\beta$, from which is no longer possible to reach a state where the termination event ω is enabled.

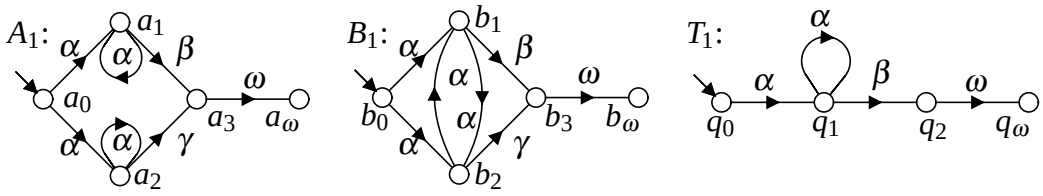


Figure 2.3: Example of automata that are not conflict equivalent.

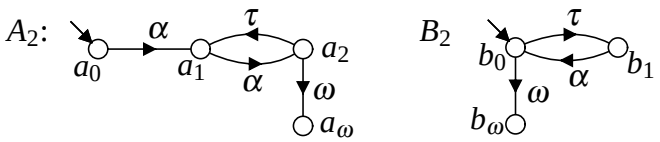


Figure 2.4: Two automata that are conflict equivalent.

For an automaton to be nonblocking, it is enough that a terminal state *can* be reached from *every* reachable state. There is no requirement for termination to be guaranteed. For example, automaton A_0 in figure 2.2 is nonblocking despite the presence of a possibly infinite loop of α-transitions in state a_0. Nonblocking is also different from "may"-testing [31], which only requires the possibility of termination from the initial state. The testing semantics most similar to nonblocking is "should"-testing, which is also known as *fair testing* [31].

A blocking states is equivalent to either a *deadlock*, where the automaton is no longer capable of doing anything, or a *livelock* where the automaton can still execute events but it can never terminate.

To reason about nonblocking in a compositional way, the notion of *conflict equivalence* is developed in [25]. According to process-algebraic testing theory, two automata are considered as equivalent if they both respond in the same way to all tests of a certain type [6]. For conflict equivalence, a *test* is an arbitrary automaton, and the *response* is the observation whether or not the test is conflicting with the automaton in question.

Definition 2.5 Let A and B be two automata. A is *less conflicting* than B, written $A \lesssim_{\text{conf}} B$, if, for every automaton T, if $B \parallel T$ is nonblocking then $A \parallel T$ also is nonblocking. A and B are *conflict equivalent*, $A \simeq_{\text{conf}} B$, if $A \lesssim_{\text{conf}} B$ and $B \lesssim_{\text{conf}} A$.

Example 2.3 Automaton A_1 in figure 2.3 is *not* less conflicting than B_1, since $A_1 \parallel T_1$ is blocking while $B_1 \parallel T_1$ is nonblocking. This is because $A_1 \parallel T_1$ can enter the blocking state (a_2, q_1) after executing α. This state is blocking because the event β can never be executed after entering the state a_2. In the case of B_1 however after executing α, it eventually becomes possible to continue using a β-transition regardless of whether the state b_1 or b_2 is entered. It can also be shown that $B_1 \lesssim_{\text{conf}} A_1$ does not hold.

Example 2.4 Automata A_2 and B_2 in figure 2.4 are conflict equivalent. For example let $T = \langle \Sigma, Q_T, \rightarrow_T, Q_T^\circ \rangle$ be an automaton such that $B_2 \parallel T$ is nonblocking. It can be inferred for every state $x \in Q_T^\circ$ that $x \stackrel{\alpha}{\Rightarrow} y$ for some $y \in Q_T$. This is because $b_0 \stackrel{\tau}{\rightarrow} b_1$, thus $(b_0, x) \stackrel{\tau}{\rightarrow} (b_1, x)$ and α is the only event which can be performed in b_1. Furthermore as $(b_1, x) \stackrel{\alpha}{\rightarrow} (b_0, y) \stackrel{\tau}{\rightarrow} (b_1, y)$ thus it can be inferred that y must also be able to perform an α event and by induction that the state x must capable of performing an infinite number α events followed by an ω, this is also what A_2 requires to be nonblocking.

The properties of the conflict preorder $\lesssim_{\text{conf}}$ and of conflict equivalence and their relationship to other process-algebraic relations are studied in [25]. It is enough to consider deterministic tests in definition 2.5, and conflict equivalence is is the coarsest possible congruence with respect to synchronous composition that respects blocking, making it an ideal equivalence for use in compositional verification [12, 35].

2.5 The Set of Certain Conflicts

Every automaton can be associated with a language of *certain conflicts*.

Definition 2.6 For an automaton $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$, write

$$\begin{aligned} \textsc{Conf}(G) = \{\, s \in \Sigma^* \mid & \text{ for every automaton } T \text{ such that } T \stackrel{s}{\Rightarrow}, \text{ it holds that} \\ & G \parallel T \text{ is blocking} \,\}\,, \end{aligned} \tag{2.7}$$

$$\begin{aligned} \textsc{NConf}(G) = \{\, s \in \Sigma^* \mid & \text{ there exists an automaton } T \text{ such that } T \stackrel{s}{\Rightarrow} \text{ and} \\ & G \parallel T \text{ is nonblocking} \,\}\,. \end{aligned} \tag{2.8}$$

$\textsc{Conf}(A)$ is the set of *certain conflicts* of A. It contains all traces that, when possible in the test, necessarily cause blocking. Its complement $\textsc{NConf}(A)$ is the most general behaviour of processes that are to be nonconflicting with A. If A is nonblocking, then $\textsc{Conf}(A) = \emptyset$ and $\textsc{NConf}(A) = \Sigma^*$, because in this case $A \parallel U$ is nonblocking, where U is a deterministic automaton such that $\mathscr{L}^\omega(U) = \Sigma^*\omega$. The set of certain conflicts becomes more interesting for blocking automata.

Example 2.5 Consider again automaton B_0 in figure 2.2. Obviously, $\alpha\beta \in \textsc{Conf}(B_0)$ as B_0 can enter the blocking state b_3 by executing $\alpha\beta$, and therefore every test T that can execute $\alpha\beta$ is conflicting with B_0. But also $\alpha \in \textsc{Conf}(B_0)$, because B_0 can enter state b_2 by executing α, from where the only possibility to terminate is by executing $\beta\omega$. So any test that can execute α also needs to be able to execute $\alpha\beta$ if it is to be nonconflicting with B_0; but such a test is conflicting with B_0 as explained above. It can be shown that $\textsc{Conf}(B_0) = \alpha\Sigma^*$.

The set of certain conflicts is introduced in [21], and its properties and its relationship to conflict equivalence are studied in [25]. Even if an automaton is nondeterministic, its set of certain conflicts is a *language*, but as shown in example 2.5, it is not necessarily a subset of the language $\mathscr{L}(A)$ of its automaton. An algorithm to compute the set of certain conflicts for a given finite-state automaton is presented in [22].

It can further be shown that an automaton's nonconflicting language is always prefix-closed.

Lemma 2.1 Let G be an automaton, it holds that $\mathrm{NConf}(G) = \overline{\mathrm{NConf}(G)}$.

Proof. $\mathrm{NConf}(G) \subseteq \overline{\mathrm{NConf}(G)}$ is trivially proven. Let $s \in \overline{\mathrm{NConf}(G)}$ be a trace. As $s \in \overline{\mathrm{NConf}(G)}$ there exists a trace $t \in \mathrm{NConf}(G)$ such that $s \sqsubseteq t$. From definition 2.6 as $t \in \mathrm{NConf}(G)$, there exists an automaton T such that $G \parallel T$ is nonblocking and $T \overset{t}{\Rightarrow}$. As $s \sqsubseteq t$ it must be the case that $T \overset{s}{\Rightarrow}$. Therefore $s \in \mathrm{NConf}(G)$. As s was chosen arbitrarily $\overline{\mathrm{NConf}(G)} \subseteq \mathrm{NConf}(G)$. □

Lastly it can be shown that for any given trace $s \in \mathrm{NConf}(G)$ every state which is reachable by s can terminate using a trace in $\mathrm{NConf}(G)$.

Lemma 2.2 Let G be an automaton, x be a state and $s \in \mathrm{NConf}(G)$ be a trace such that $G \overset{s}{\Rightarrow} x$. Then there exists $t \in \Sigma^*$ such that $x \overset{t\omega}{\Rightarrow}$ and $st \in \mathrm{NConf}(G)$.

Proof. Let $s \in \mathrm{NConf}(G)$ such that $G \overset{s}{\Rightarrow} G_s$. Since $s \in \mathrm{NConf}(G)$, according to definition 2.6, there exists a test automaton T such that $G \parallel T$ is nonblocking and $T \overset{s}{\Rightarrow} x_T$ for some state x_T. Then $G \parallel T \overset{s}{\Rightarrow} (x, x_T)$, so there exists $t \in \Sigma^*$ such that $G \parallel T \overset{s}{\Rightarrow} (x, x_T) \overset{t\omega}{\Rightarrow}$. Clearly $x \overset{t\omega}{\Rightarrow}$, and furthermore $T \overset{st}{\Rightarrow}$, which implies $st \in \mathrm{NConf}(G)$ as $G \parallel T$ is nonblocking. □

2.6 Nonconflicting Completions

Automata can be further associated to a set of nonconflicting completions. Nonconflicting completions are a pair of trace and language. If the pair (c, C) is a nonconflicting completion of the automaton G, then for every test automaton T which is capable of performing the trace c it must be the case that either T can continue with at least one trace $t \in C$ or $G \parallel T$ is blocking.

Definition 2.7 For an automaton G, define

$$\mathrm{CC}(G) = \{\, (c,C) \in \Sigma^* \times 2^{\Sigma^+_\omega} \mid \text{for every test automaton } T \text{ and state } x_T\colon \text{ if } G \parallel T \text{ is nonblocking and } T \overset{c}{\Rightarrow} x_T \text{ then } \exists t \in C\colon\ x_T \overset{t}{\Rightarrow} \,\} \;; \tag{2.9}$$

$$\mathrm{CC}^\omega(G) = \{\, (c,C) \in \Sigma^* \times 2^{\Sigma^*\omega} \mid \text{for every test automaton } T \text{ and state } x_T\colon \text{ if } G \parallel T \text{ is nonblocking and } T \overset{c}{\Rightarrow} x_T \text{ then } \exists t \in C\colon\ x_T \overset{t}{\Rightarrow} \,\} \;. \tag{2.10}$$

The only difference between $\mathrm{CC}(G)$ and $\mathrm{CC}^\omega(G)$ is that $\mathrm{CC}^\omega(G)$ only contains complete traces that end with ω. $\mathrm{CC}(G)$ is called the *nonconflicting continuation semantics*, and $\mathrm{CC}^\omega(G)$ is called the *nonconflicting completion semantics* of G. In both cases, the set C of nonconflicting continuations or completions cannot contain the empty trace.

The concept of derivation which has been applied to both languages and automata can also be applied to a set of nonconflicting completions or continuation.

Definition 2.8 For $\mathscr{C} \subseteq \Sigma^* \times 2^{\Sigma^*\omega}$ and $s \in \Sigma^*$, define

$$\mathscr{C}/s = \{\, (t,C) \in \Sigma^* \times 2^{\Sigma^*\omega} \mid (st,C) \in \mathscr{C} \,\} \;. \tag{2.11}$$

The following is an unpublished proof by Dr Robi Malik. It shows that the nonconflicting completions are preserved after derivation.

Proposition 2.1 Let G be an automaton and $c \in \mathrm{NConf}(G)$ be a trace then,

$$\mathrm{CC}^\omega(G/c) = \mathrm{CC}^\omega(G)/c \;. \tag{2.12}$$

Proof. Let G be an automaton and $c \in \mathrm{NConf}(G)$ trace.

First assume that $(s,C) \in \mathrm{CC}^\omega(G/c)$, and consider a test automaton T and state y_T such that $G \parallel T$ is nonblocking and $T \overset{cs}{\Rightarrow} y_T$. Then there exists a state x_T such that $T \overset{c}{\Rightarrow} x_T \overset{s}{\Rightarrow} y_T$. It follows that $(G/c) \parallel (T/c)$ is nonblocking as $G \parallel T$ is nonblocking. To see this let $t \in \Sigma^*$ be a trace and (z,z_T) be a state tuple such that $(G/c) \parallel (T/c) \overset{t}{\Rightarrow} (z,z_T)$. As $(G/c) \parallel (T/c) \overset{t}{\Rightarrow} (z,z_T)$ it holds that $G \parallel T \overset{ct}{\Rightarrow} (z,z_T)$. As $G \parallel T$ is nonblocking it must hold that (z,z_T) is nonblocking. As t and (z,z_T) where chosen arbitrarily it must be the case that $(G/c) \parallel (T/c)$ are nonblocking. Furthermore as $(s,C) \in \mathrm{CC}^\omega(G/c)$, $(G/c) \parallel (T/c)$ is nonblocking, and $T/c \overset{s}{\Rightarrow} y_T$ it holds that $y_T \overset{u}{\Rightarrow}$ for some $u \in C$. Since T and y_T was chosen arbitrarily, it follows that $(cs,C) \in \mathrm{CC}^\omega(G)$, and thus $(s,C) \in \mathrm{CC}^\omega(G)/c$.

Conversely, let $(s,C) \in \mathrm{CC}^\omega(G)/c$. By definition, this means that $(cs,C) \in \mathrm{CC}^\omega(G)$. Consider a test $T = \langle \Sigma, Q_T, \to_T, Q_T^\circ \rangle \Sigma$ such that $(G/c) \parallel T$ is nonblocking and $T \overset{s}{\Rightarrow} x_T$.

A new test automaton T^c is constructed as follows,

$$T^c = \langle \Sigma, \mathrm{NC{\scriptstyle ONF}}^\omega(G) \dot\cup Q_T, \rightarrow_N \cup \rightarrow_T \cup \rightarrow_{NT}, \{\varepsilon\} \rangle \tag{2.13}$$

where $\rightarrow_N$ and $\rightarrow_{NT}$ consist of the transitions

$$s \xrightarrow{\sigma}_N s\sigma \qquad \text{for all } s\sigma \in \mathrm{NC{\scriptstyle ONF}}^\omega(G)\ ; \tag{2.14}$$

$$c \xrightarrow{\tau}_{NT} x_T^\circ \qquad \text{for all } x_T^\circ \in Q_T^\circ\ . \tag{2.15}$$

Then $G \parallel T^c$ is nonblocking. To see this, let $t \in \Sigma^*$ such that $G \parallel T^c \stackrel{t}{\Rightarrow} (x_t^G, x_t^T)$, for some states x_t^G of G and x_t^T of T^c. If $x_t^T \in \mathrm{NC{\scriptstyle ONF}}(G)$, then by construction $t \in \mathrm{NC{\scriptstyle ONF}}(G)$, and given $G \stackrel{t}{\Rightarrow} x_t^G$, it follows by lemma 2.2 that there exists $v \in \Sigma^*$ such that $x_t^G \stackrel{v\omega}{\Rightarrow}$ and $tv \in \mathrm{NC{\scriptstyle ONF}}(G)$. Again by construction, and since $\mathrm{NC{\scriptstyle ONF}}(G)$ is prefix-closed by lemma 2.1, it follows that $(x_t^G, x_t^T) \stackrel{v\omega}{\Rightarrow}$. If on the other hand $x_t^T \in Q_T$, then by construction $t = cu$ for some $u \in \Sigma^*$, and

$$G \parallel T^c \stackrel{c}{\Rightarrow} (x_c^G, c) \stackrel{\tau}{\rightarrow} (x_c^G, x_T^\circ) \stackrel{u}{\Rightarrow} (x_t^G, x_t^T) \tag{2.16}$$

for some state x_c^G of G and some $x_T^\circ \in Q_T^\circ$. Clearly, $G \stackrel{c}{\Rightarrow} x_c^G \stackrel{u}{\Rightarrow} x_t^G$ and therefore $G/c \stackrel{u}{\Rightarrow} x_t^G$. Together with $T \stackrel{u}{\Rightarrow} x_t^T$, this implies $(G/c) \parallel T \stackrel{u}{\Rightarrow} (x_t^G, x_t^T)$. Since $(G/c) \parallel T$ is nonblocking, there exists $v \in \Sigma^*$ such that $(x_t^G, x_t^T) \stackrel{v\omega}{\Rightarrow}$. Since the state (x_t^G, x_t^T) was chosen arbitrarily, it follows that $G \parallel T^c$ is nonblocking.

By construction of T^c, it holds that $T^c \stackrel{c}{\Rightarrow}_N c \xrightarrow{\tau}_{NT} x_T^\circ$ for every $x_T^\circ \in Q_T^\circ$, and since $T \stackrel{s}{\Rightarrow} x_T$, it follows that $T^c \stackrel{cs}{\Rightarrow} x_T$. Since $G \parallel T^c$ is nonblocking and $(cs, C) \in \mathrm{CC}^\omega(G)$, there exists $t \in C$ such that $x_T \stackrel{t}{\Rightarrow}$. Since T was chosen arbitrarily, it follows that $(s, C) \in \mathrm{CC}^\omega(G/c)$. □

As a direct consequence of proposition 2.1 if $(c, C) \in \mathrm{CC}^\omega(G)$ then it is also the case that $(\varepsilon, C) \in \mathrm{CC}^\omega(G'/c)$.

Proposition 2.2 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an automaton. Let (c, C) be a pair of trace and language such that $c \in \mathrm{NC{\scriptstyle ONF}}(G)$. It holds that $(c, C) \in \mathrm{CC}^\omega(G)$ if and only if $(\varepsilon, C) \in \mathrm{CC}^\omega(G'/c)$.

Proof. This comes directly from 2.1. □

2.7 Compositional Nonblocking Verification

The one of the main reasons why conflict equivalence preserving abstractions are of interest is that they can be used to check whether a large system of concurrent pro-

cesses are conflicting or not. The straightforward approach to check whether automata $A_1, A_2, \ldots, A_n$ are conflicting is to construct the synchronous product

$$A_1 \parallel A_2 \parallel \cdots \parallel A_n \tag{2.17}$$

and check whether it is blocking. This is done by checking whether a terminal state can be reached from every reachable state. Using symbolic representations such as BDDs [3] or IDDs [39], this approach has been used to analyse very large models. Yet, the technique always remains limited by the amount of memory available to store representations of the synchronous product. As an alternative, *compositional verification* [12] seeks to replace automaton A_1, e.g., by a simpler version A_1', and analyse the simpler system

$$A_1' \parallel A_2 \parallel \cdots \parallel A_n \,. \tag{2.18}$$

If A_1 and A_1' are conflict equivalent, then (2.17) is nonblocking if and only if (2.18) is nonblocking. This is a consequence of the *congruence* properties of the conflict preorder [25]. The following results follow directly from definition 2.5.

Proposition 2.3 [25] $\lesssim_{\text{conf}}$ is a *pre-congruence* with respect to $\parallel$. That is, if $A \lesssim_{\text{conf}} B$, then $A \parallel T \lesssim_{\text{conf}} B \parallel T$ for every automaton T.

Proposition 2.4 [25] $\lesssim_{\text{conf}}$ *respects blocking*. That is, if $A \lesssim_{\text{conf}} B$ and B is nonblocking, then A also is nonblocking.

Compositional verification relies on the above two congruence properties and the following simple facts about hiding.

Lemma 2.3 Let $A = \langle \Sigma_A, Q_A, \rightarrow_A, Q_A^\circ \rangle$ be an automaton and $\Upsilon \subseteq \Sigma_A$.

(i) A is nonblocking if and only if $A \setminus \Upsilon$ is nonblocking.

(ii) If $B = \langle \Sigma_B, Q_B, \rightarrow_B, Q_B^\circ \rangle$ is an automaton such that $\Sigma_B \cap \Upsilon = \emptyset$, then $(A \setminus \Upsilon) \parallel B = (A \parallel B) \setminus \Upsilon$.

Property (ii) shows how *local* events are exploited in compositional verification. A component A_1 in a larger system such as (2.17) typically contains certain events that are not used in any of the remaining components $A_2, \ldots, A_n$. Such events are local to A_1, and their identity can be removed. These events can be replaced by the silent event τ, making it possible to simplify the automaton. Compositional verification is based on this fact and the congruence properties.

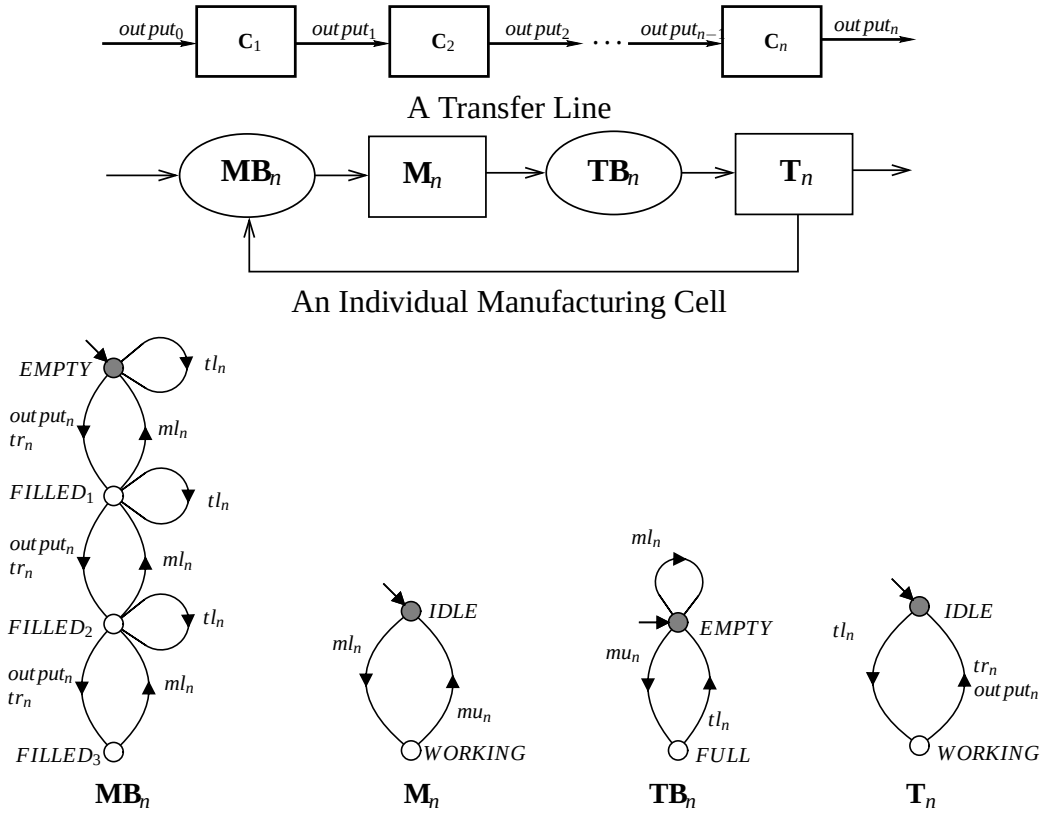


Figure 2.5: Manufacturing cell example.

Proposition 2.5 Let $A = \langle \Sigma_A, Q_A, \rightarrow_A, Q_A^\circ \rangle$ and $B = \langle \Sigma_B, Q_B, \rightarrow_B, Q_B^\circ \rangle$ be two automata, and let $\Upsilon \subseteq \Sigma_A$ such that $\Sigma_B \cap \Upsilon = \emptyset$. Furthermore, let A' be an automaton such that $A \setminus \Upsilon \lesssim_{\text{conf}} A'$. Then it holds that, if $A' \,\|\, B$ is nonblocking then $A \,\|\, B$ is nonblocking.

Proof. Let $A' \,\|\, B$ be nonblocking. Since $A \setminus \Upsilon \lesssim_{\text{conf}} A'$, it follows by proposition 2.3 that $(A \setminus \Upsilon) \,\|\, B \lesssim_{\text{conf}} A' \,\|\, B$, which implies by proposition 2.4 that $(A \setminus \Upsilon) \,\|\, B$ is nonblocking. Then it follows from lemma 2.3 that $(A \,\|\, B) \setminus \Upsilon = (A \setminus \Upsilon) \,\|\, B$ is nonblocking (ii), which means that $A \,\|\, B$ is nonblocking (i). □

Proposition 2.5 gives a basic way to exploit the conflict preorder when verifying a composed system to be nonblocking. The following example shows how such results can be used to model check a large system of composed automata.

Example 2.6 Figure 2.5 shows a discrete event system model of a factory, which is made up of a series of manufacturing cells. The model involves n manufacturing cells, where the output of the first manufacturing cell is used as the input of the second and so forth. Each manufacturing cell has a machine which does work on the work pieces which flow through the factory as well as a test unit which determines whether the work on the work piece is satisfactory. In addition there are two buffers used to store

work pieces before processing, one for the machine, the second for test unit. Given this representation of the factory it is desirable to determine whether it is nonconflicting.

The flow of work pieces through the nth manufacturing cell is modelled using the events $output_{n-1}$, $output_n$, tl_n, tu_n, tr_n, ml_n, and mu_n. A work piece entering the manufacturing cell is represented by the event $output_{n-1}$, which causes the machine buffer $\mathbf{MB}_n$ to be filled with one more work piece up to a maximum of three. The work piece can then be loaded into the machine $\mathbf{M}_n$ represented by the ml_n event. Once the machine has finished working, the work piece is placed in the test buffer $\mathbf{TB}_n$ with the event mu_n. The work piece can now be picked up by the test unit $\mathbf{T}_n$ using the event tl_n. At this point, the test unit can decide either to accept $(output_n)$ the work piece in which case it is sent on to the next manufacturing cell, or it can reject the work piece (tr_n) and have it sent back to the machine buffer so that it can have more work done on it. The test buffer $\mathbf{TB}_n$ will only allow the machine $\mathbf{M}_n$ to be loaded when $\mathbf{TB}_n$ is empty, this is to make certain that there will be a place for the work piece in $\mathbf{M}_n$ to be put when the machine is finished. $\mathbf{MB}_n$ only allows the test unit to be loaded when there is an empty space in $\mathbf{MB}_n$ for similar reasons.

The state space of this model grows exponentially in the number n of manufacturing cells. When $n = 7$, the number of reachable states in the model is equal to approximately 5.1 billion states, this has been found by BDDs to construct the state space symbolically. Using compositional verification based on conflict equivalence, the system can be proven to be nonblocking for arbitrary values of n while looking at far less states.

Composing the automata for the first manufacturing cell produces the subsystem

$$\mathbf{C}_1 = \mathbf{MB}_1 \parallel \mathbf{M}_1 \parallel \mathbf{TB}_1 \parallel \mathbf{T}_1 \tag{2.19}$$

with *local* events tl_n, tu_n, tr_n, ml_n, and mu_n, as well as 22 reachable states. These events are used only in the automata comprising $\mathbf{C}_1$, so they can be hidden, i.e., replaced by the silent event τ before composing $\mathbf{C}_1$ with further automata. While $\mathbf{C}_1$ has 22 reachable states, it can be shown to be conflict equivalent to the three state abstraction $\mathbf{C}'_{\mathbf{1}}$ in figure 2.6,

$$\mathbf{C}_1 \setminus \{tl_n, tu_n, tr_n, ml_n, mu_n\} \simeq_{\text{conf}} \mathbf{C}'_{\mathbf{1}} \,. \tag{2.20}$$

The same process can be applied to the second manufacturing cell. This will result $\mathbf{C}'_{\mathbf{2}}$ shown in figure 2.6. The next step of composing $\mathbf{C}'_{\mathbf{1}}$ and $\mathbf{C}'_{\mathbf{2}}$ results in a 15 state automaton, which after hiding the event $output_1$ can be replaced with the automaton $\mathbf{C}'_{\mathbf{1,2}}$ in figure 2.6. The same process can be applied to add $\mathbf{C}_3$ to the composition resulting in an automaton that is identical to both $\mathbf{C}'_{\mathbf{1}}$ and $\mathbf{C}'_{\mathbf{1,2}}$ with the exception that the different automata use $output_1$, $output_2$ and $output_3$ respectively. This process

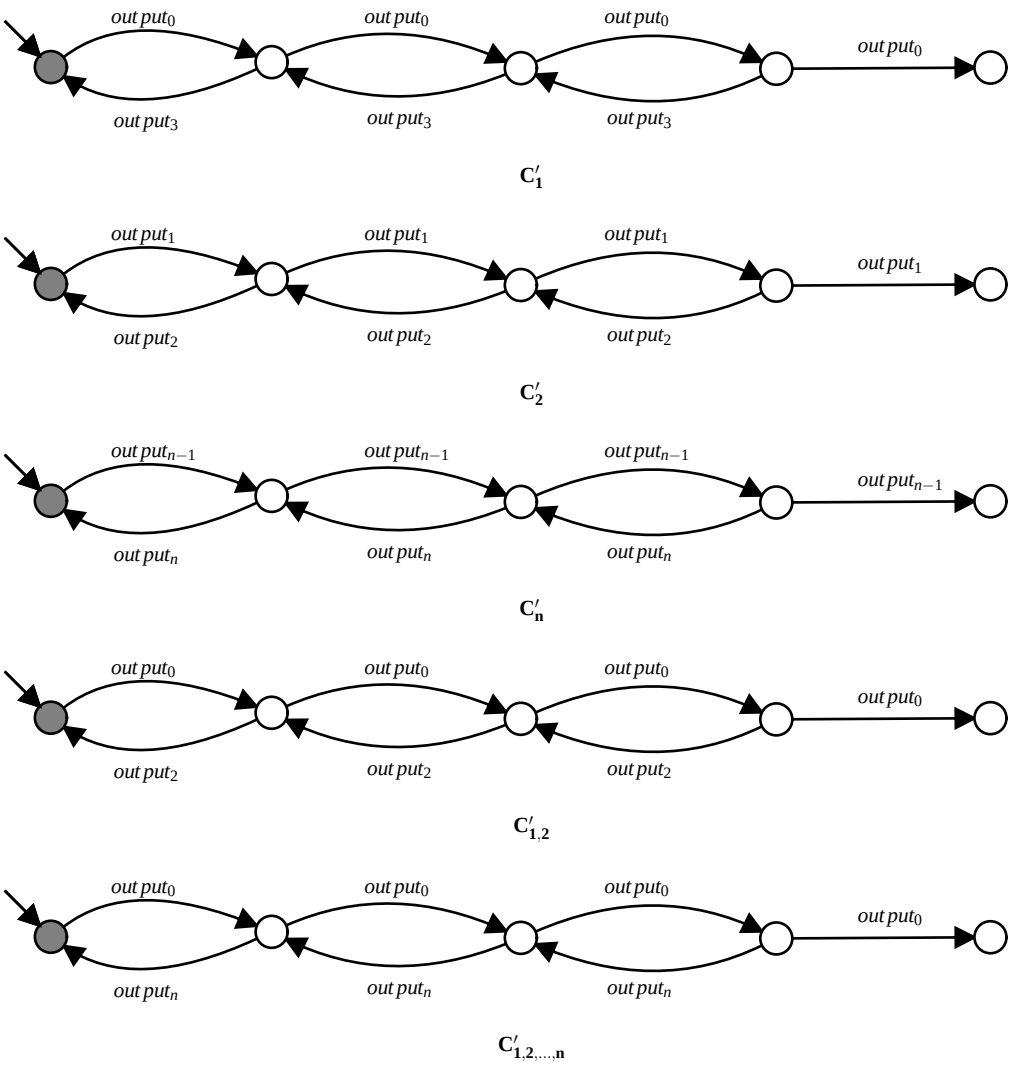


Figure 2.6: Abstractions for manufacturing cell example

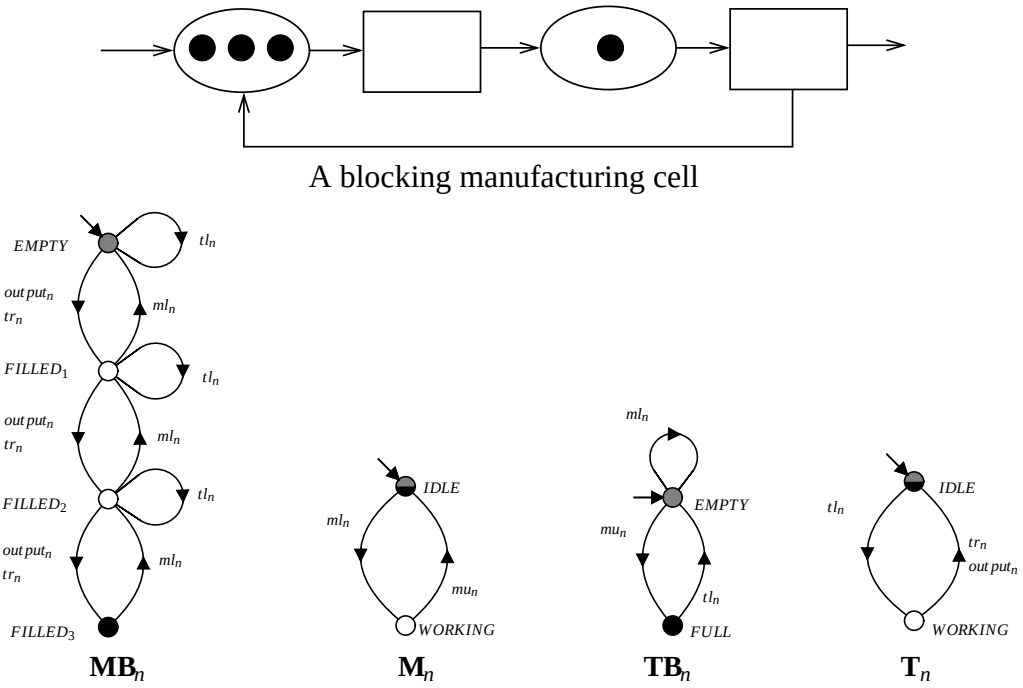


Figure 2.7: A manufacturing cell in state of deadlock.

can be repeated for all n manufacturing cells resulting in the automaton $\mathbf{C}_{1,2,\ldots,n}$. This automaton is blocking, thus the entire manufacturing cell model is blocking. Thus it is possible to use compositional model checking to determine whether this model is nonconflicting while at any given step only looking at an automaton with at most 22-reachable states regardless of how large n is.

The abstraction also highlights, the circumstances, in which a manufacturing cell is potentially blocking. For example if manufacturing cell $\mathbf{C}_1$ has four or more work pieces in it at one time the manufacturing cell can block. Figure 2.7 highlights one of the situations in which an individual manufacturing cell is blocking. For this figure the current state of the automaton is coloured black, if the current state is both marked and the current state, the state is coloured half black half gray. It highligts the situation where both the machine buffer and the test buffer is full. In this situation neither the machine nor the test unit may load a work piece, thus neither of the buffers can be emptied.

Example 2.7 The dining philosopher's problem [7] in concurrency is commonly used to describe the problem of deadlock in concurrent systems. It involves n philosophers sitting at a circular table with a large bowl of spaghetti in the centre. A fork is placed between each pair of philosophers, and each philosopher must eat with the two forks next to him or her. Here they each spend their time pondering or alternatively eating. In in order for a philosopher to eat they must pick up two forks, and they will not

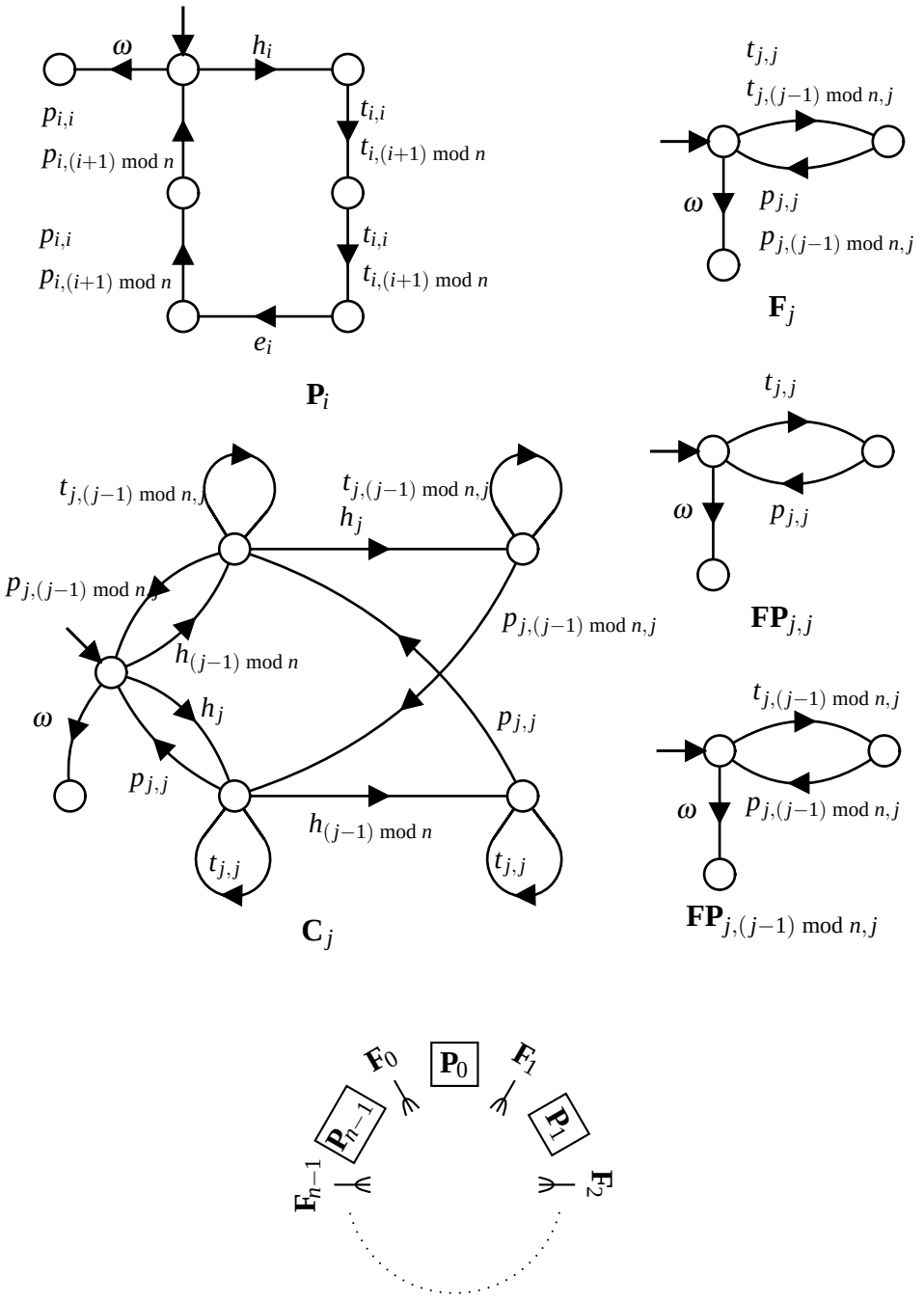


Figure 2.8: Dining philosophers example.

put their forks back down until after they have eaten. Because there is only one fork per philosopher, if the philosophers are unable to communicate with each other this can lead to a state where none of the philosophers are capable of eating, for example if all philosophers pick up their left fork. Figure 2.8 shows a discrete event system model of the dining philosopher problem which uses a coordinator to remove the presence of both deadlock and livelock. The diagram in figure 2.8 shows the philosophers $\mathbf{P}_0, \ldots, \mathbf{P}_{n-1}$ with the forks $\mathbf{F}_0, \ldots, \mathbf{F}_{n-1}$ between them, and the automata represent and attempt to control the system such that every philosopher can eventually get a chance to eat. Given this model we would like to prove that it does in fact solve the problem. That is to say it is impossible to reach a situation where it is impossible for any philosopher to eat. This can be done by proving that the model is nonconflicting. This is because only the initial state of this model can terminate, and it can be easily proven that from the initial state it is possible for any particular philosopher can eat.

Access to the forks is modelled using events $t_{i,j}$ and $p_{i,j}$, where $t_{i,j}$ means that philosopher $\mathbf{P}_i$ takes fork $\mathbf{F}_j$, and $p_{i,j}$ means that he or she puts it back. The automata $\mathbf{P}_i$, for $i = 0, \ldots, n-1$, model the behaviour of each philosopher $\mathbf{P}_i$: after a thinking phase, the philosopher gets hungry and signals his or her intention to eat (h_i), then he or she picks up both adjacent forks ($t_{i,i}$ and $t_{i,(i+1) \bmod n}$), eats (e_i), puts the forks back ($p_{i,i}$ and $p_{i,(i+1) \bmod n}$), and returns to the thinking phase. The fork automata $\mathbf{F}_j$ ensures that $\mathbf{F}_j$ can only be picked taken up one philosopher at a time whereas the automaton $\mathbf{FP}_{i,j}$ ensures that the philosopher $\mathbf{P}_i$ can only put down $\mathbf{F}_j$ after picking it up, this ensures mutual exclusion. Finally, the coordinators $\mathbf{C}_j$ sequence access to fork $\mathbf{F}_j$ by the two philosophers using it, such that the philosopher who gets hungry first also gets access first. For example if we consider the coordinator automaton for $\mathbf{F}_0$, it requires that before either philosopher $\mathbf{P}_0$ or $\mathbf{P}_1$ may take the fork ($t_{0,0}$ or $t_{1,0}$) they must first register their hunger (h_0 or h_1). Once philosopher $\mathbf{P}_0$ has registered his or her hunger if $\mathbf{P}_1$ has not registered hunger before them they may immediately take the fork $t_{0,0}$ otherwise they must wait for $\mathbf{P}_1$ to return the fork $p_{0,1}$. This is regardless of whether or not $\mathbf{P}_1$ has taken the fork yet.

The state space of this model grows exponentially in the number n of philosophers. It can be shown that when $n = 16$, the number of reachable states in the model is greater than $1.123 \cdot 10^{13}$ using symbolic model checking methods. For larger parameter values it becomes infeasible to model check the system using explicit or symbolic methods. Using compositional verification based on the conflict preorder, the system can be proven to be nonblocking for arbitrary values of n.

Composing the automata for philosophers $\mathbf{P}_0$ and $\mathbf{P}_1$ and the shared fork $\mathbf{F}_1$ produces

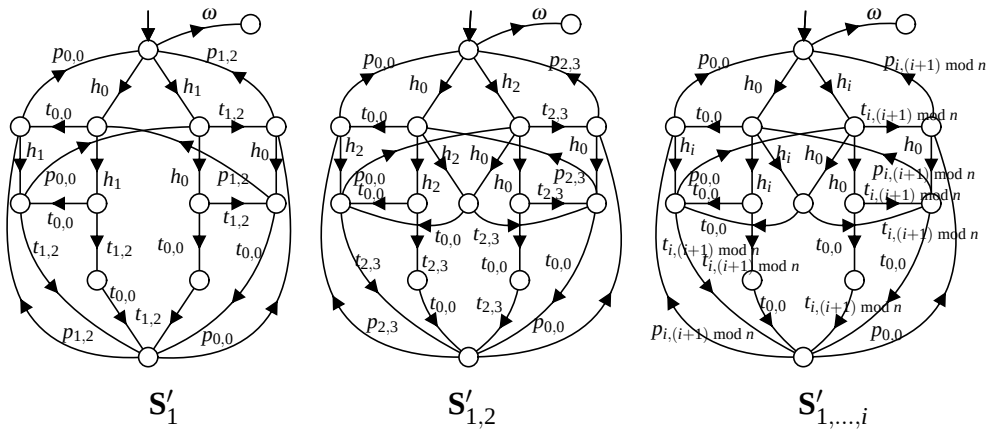


Figure 2.9: Abstractions for dining philosophers example.

the subsystem

$$\mathbf{S}_1 = \mathbf{FP}_{0,0} \parallel \mathbf{P}_0 \parallel \mathbf{FP}_{0,1} \parallel \mathbf{F}_1 \parallel \mathbf{C}_1 \parallel \mathbf{FP}_{1,1} \parallel \mathbf{P}_1 \parallel \mathbf{FP}_{1,2} \tag{2.21}$$

with *local* events e_0, $t_{0,1}$, $p_{0,1}$, $t_{1,1}$, $p_{1,1}$, and e_1 and 52 reachable states. These events are used only in the automata comprising $\mathbf{S}_1$, so they can be hidden, i.e., replaced by the silent event τ before composing $\mathbf{S}_1$ with further automata. While $\mathbf{S}_1$ has 52 reachable states, it can be shown to be less conflicting than (in fact, conflict equivalent to) the 12-state abstraction $\mathbf{S}'_1$ in figure 2.9,

$$\mathbf{S}_1 \setminus \{e_0, t_{0,1}, p_{0,1}, t_{1,1}, p_{1,1}, e_1\} \lesssim_{\text{conf}} \mathbf{S}'_1 \; . \tag{2.22}$$

The next step to add the automata for fork $\mathbf{F}_2$ and philosopher $\mathbf{P}_2$ to the system results in a 90-state automaton $\mathbf{S}_{1,2}$, with new local events. Hiding the local events leads to a new 13-state abstraction $\mathbf{S}'_{1,2}$ also shown in figure 2.9,

$$(\mathbf{S}'_1 \parallel \mathbf{F}_2 \parallel \mathbf{C}_2 \parallel \mathbf{FP}_{2,2} \parallel \mathbf{P}_2 \parallel \mathbf{FP}_{2,3}) \setminus \{h_1, t_{1,2}, p_{1,2}, t_{2,2}, p_{2,2}, e_2\} \lesssim_{\text{conf}} \mathbf{S}'_{1,2} \; . \tag{2.23}$$

Repeating this process by including the next fork $\mathbf{F}_3$ and philosopher $\mathbf{P}_3$ gives a 100-state automaton $\mathbf{S}_{1,2,3}$ and an abstraction $\mathbf{S}'_{1,2,3}$ identical to $\mathbf{S}'_{1,2}$ except that the events for fork $\mathbf{F}_2$ and philosopher $\mathbf{P}_2$ are replaced by the corresponding events for fork $\mathbf{F}_3$ and philosopher $\mathbf{P}_3$. By induction, it can be shown that the subsystem consisting of forks $\mathbf{F}_1, \ldots, \mathbf{F}_i$ and philosophers $\mathbf{P}_0, \ldots, \mathbf{P}_i$ is less conflicting than the abstraction $\mathbf{S}'_{1,\ldots,i}$ in figure 2.9. Composing this abstraction with the missing automata for fork $\mathbf{F}_0$ gives a 41-state automaton $\mathbf{F}_0 \parallel \mathbf{C}_0 \parallel \mathbf{S}'_{1,\ldots,i}$, which is easily shown to be nonblocking.

By repeated application of proposition 2.5 it follows that the dining philosophers model is nonblocking for all values of $n \geq 4$. Despite the enormous state space, this

is possible without ever considering an automaton with more than 100 states. The necessary tests for the conflict preorder have been completed in less than one second using the implementation described below in section 5.6. The most difficult is the test for (2.23), which takes 0.34 s to complete. Further performance data is given in section 5.7.

Chapter 3

Annotated Automata

As has explained in section 2.4, conflicts can been put into two distinct categories: deadlocks and livelocks. While it is quite difficult to categorise information about livelock in a compositional way, this is not the case for deadlock.

In [16] the set of failures is used to characterise processes with respect to how they reach a deadlock situation. Failure sets can be used to characterise the deadlock information contained within an automaton, and minimize with respect to it. Unfortunately simplifying an automaton solely with respect to failures is not guaranteed to preserve conflict equivalence as livelocks may be hidden.

This chapter introduces annotated automata as a means of using failures to simplify automata with respect to conflict equivalence. An annotated automaton is a standard automaton which in which each state is annotated with a set of events. These event sets are called annotations. An annotation signifies that any automaton which wishes to be nonblocking with the state which that annotation is associated with must be able to execute at least one event in that annotation. This is similar to nonconflicting completions mentioned in section 2.6. If an annotated automaton A is derived from the standard automaton G the annotations of A will be derived from the ready sets of the states in G, where ready sets are the complement of failure sets. The information contained in annotations can be used in several abstraction rules as well as a modified version of bisimulation equivalence in order to simplify automata with respect to conflict equivalence.

Unlike the later chapters, which seek to fully characterise conflict equivalence, this chapter only provides abstraction rules which can be used to simplify an automaton with respect to conflict equivalence. That said the methods developed in this chapter are fast abstraction rules which have been shown to be capable of improving the performance of compositional conflict checkers. Furthermore the idea of annotating states with their ready sets/one step nonconflicting completions is generalised in future

chapters to comparing the full nonconflicting completions.

This chapter is organized into several sections. Section 3.1 describes how a standard automaton can be converted into an annotated automaton, and back again. In addition it describes several abstractions which can be applied to annotated automata which preserve conflict-equivalence. Finally section 3.2 gives experimental results showing the effectiveness of using the abstractions in a compositional checker.

3.1 Annotated Automata

This section shows how annotations are used to bring automata in a more regular form to make simplification with respect to conflict equivalence more effective. Using the running example in figure 3.1, methods to construct an annotated automaton are described in 3.1.1 and 3.1.2, and three abstraction rules to simplify annotated automata are presented in 3.1.3–3.1.5. In 3.1.6, the complete abstraction procedure to simplify automata using annotations is presented.

3.1.1 Annotation

The states in a nondeterministic automaton carry several implicit requirements characterising their blocking or nonblocking behaviour in composition with other automata. For illustration, consider state q_0 in automaton G in figure 3.1. Its eligible event set is $\mathrm{Elig}_G(q_0) = \{\alpha, \beta, \gamma\}$; note that β is included because of the silent transition to q_4. Blocking will occur if state q_0 is composed with a state that does not enable at least one of the events α, β, or γ. Moreover, due to the silent transitions to states q_3 and q_4, any state composed with q_0 also needs to enable at least one event from their sets of eligible events, $\mathrm{Elig}_G(q_4) = \{\alpha, \beta\}$ and $\mathrm{Elig}_G(q_3) = \{\alpha\}$. In order to capture these nonblocking requirements in a more concise manner, the three eligible event sets are associated with state q_0 as *annotations*.

Definition 3.1 An *annotated automaton* is a 5-tuple $A = \langle \Sigma, Q, \rightarrow, Q^\circ, Ann \rangle$ such that $\langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ is an ordinary automaton without τ-transitions, and $Ann \subseteq Q \times 2^{\Sigma_\omega}$ is the *annotation relation*, which satisfies the following conditions:

(i) for every $x \in Q$, there exists $a \subseteq \Sigma_\omega$ such that $(x, a) \in Ann$;

(ii) for every $(x, a) \in Ann$, it holds that $a \subseteq \mathrm{Elig}_A(x)$.

An annotation is a set of events $a \subseteq \Sigma_\omega$ associated with a state $x \in Q$. The intended meaning of $(x, a) \in Ann$ is that, if the automaton is in state x, at least one of the events in a must be enabled in the synchronous composition of the entire system in order

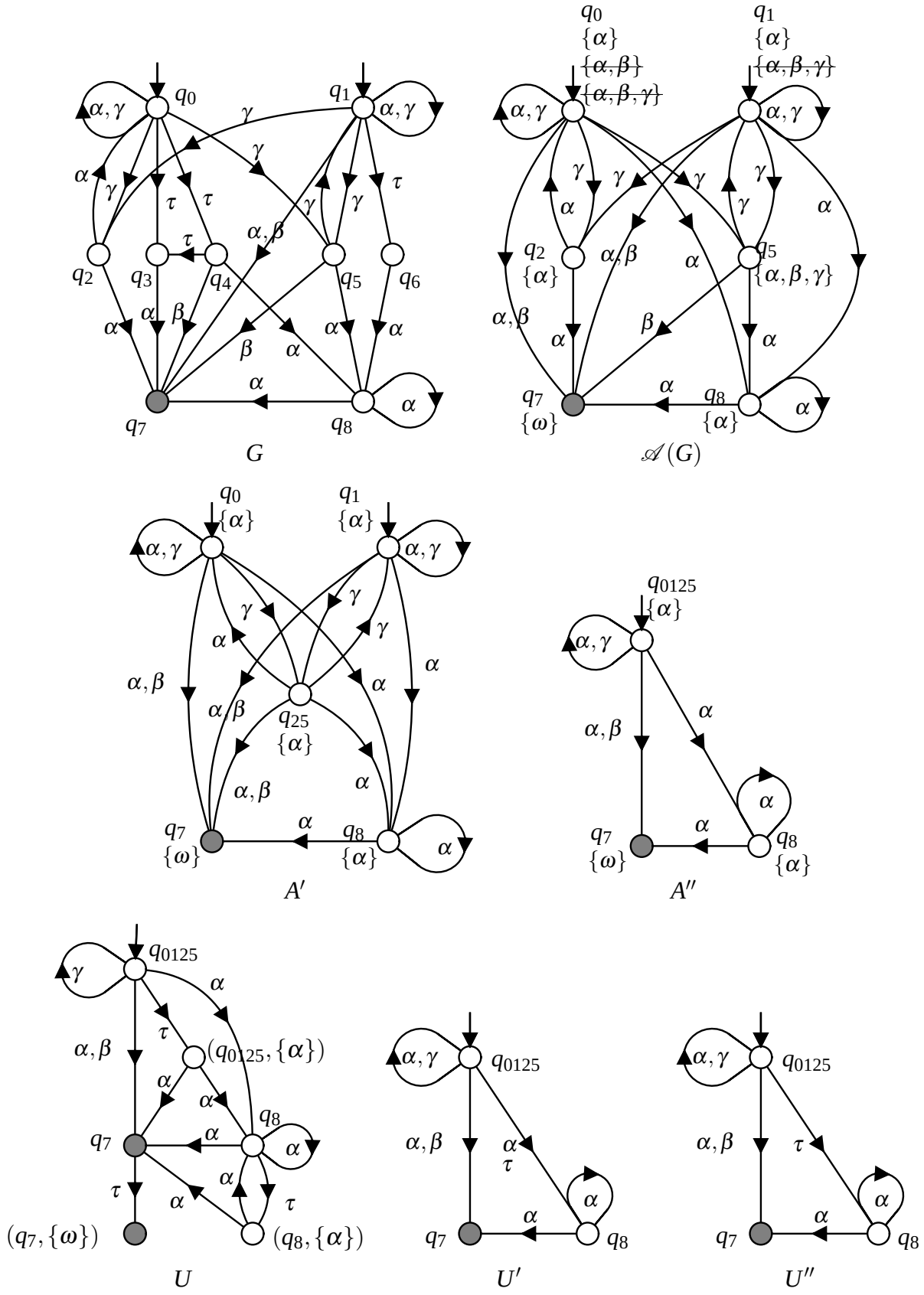


Figure 3.1: Simplification of automaton G using annotations gives $G \simeq_{\text{conf}} U''$.

to avert blocking. The empty set of events can also serve as an annotation, which is used to characterise deadlock states. Annotations are similar to *ready sets* [28] or the complements of *failure sets* [16], but they can only be used to partially characterise conflict equivalence.

The two requirements (i) and (ii) ensure that annotations capture the idea of nonblocking requirements correctly. Each state must have at least one annotation, and all annotations must be subsets of the eligible event set of their state. When annotating automata in practice, every state can be associated with its own eligible event set as an annotation, and this "maximal" annotation does not need to be stored explicitly in an annotated automaton as it can be inferred from the transitions.

The following definition shows how to transform an arbitrary nondeterministic automaton into an annotated automaton. - To do this for every state x in the automaton, all states y which x can reach silently are determined. Once these states are determined a copy of every outgoing transition of y is created with x as its new source state, in addition the eligible event set of y is calculated and added to x as an annotation. After this information has been added to the annotated automaton all silent transitions can be removed without losing any conflict information.

Definition 3.2 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an automaton. The *annotated form* of G is

$$\mathscr{A}(G) \;=\; \langle \Sigma, Q, \rightarrow_A, Q^\circ, Ann \rangle \;, \tag{3.1}$$

where

$$\rightarrow_A = \{\, (x,\sigma,y) \in Q \times \Sigma_\omega \times Q \mid x \stackrel{\varepsilon}{\Rightarrow} z \stackrel{\sigma}{\rightarrow} y \text{ for some } z \in Q \,\}\;; \tag{3.2}$$

$$Ann = \{\, (x, \mathrm{Elig}_G(y)) \mid x \stackrel{\varepsilon}{\Rightarrow} y \,\}\;. \tag{3.3}$$

The annotated form clearly satisfies the two conditions (i) and (ii) in definition 3.1, because $(x, \mathrm{Elig}_G(x)) \in Ann$ for every $x \in Q$, and $x \stackrel{\varepsilon}{\Rightarrow} y$ implies $\mathrm{Elig}_G(y) \subseteq \mathrm{Elig}_G(x)$.

The annotated form is obtained from the original automaton by replacing all silent transitions by the transitions originating from the silent successor states: if state z can be reached silently from state x, then all transitions originating from z are copied to x. Due to this removal of silent transitions, some states may become unreachable and then can be removed. To retain the nonblocking conditions associated with the originally silently reached states, their eligible event sets are added as annotations to the start states of the removed transitions.

Example 3.1 Figure 3.1 shows an automaton G and its annotated form $\mathscr{A}(G)$. As each state can be reached from itself after 0 silent transitions, it is associated with its own

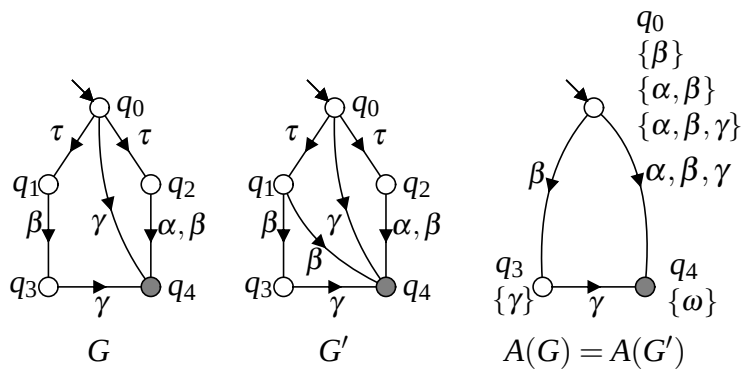


Figure 3.2: Two automata with equivalent annotated form

eligible event set as an annotation. The state q_0 collects all the outgoing transitions of q_3 and q_4, because it is connected to these two states by silent transitions, and annotations are added to q_0 for each of these two states. Similarly, q_1 has all the outgoing transitions and the annotation $\{\alpha\}$ of q_6. The states q_3, q_4, and q_6 have been deleted because they become unreachable after the removal of silent transitions.

Complexity 3.1 The annotated form $\mathscr{A}(G)$ of $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ has $|Q|$ states, up to $|Q|^2|\Sigma_\omega|$ transitions, and up to $|Q|^2$ annotations. Thus, its size is bounded by $O(|Q|^2|\Sigma|)$. The time complexity to construct $\mathscr{A}(G)$ is dominated by the computation of the transitive closure of the silent transitions, i.e., $O(|Q|^3)$ [27].

Annotation removes information, and it may well happen that two different automata have equal annotated forms. The following proposition shows that this can only happen if the two original automata are conflict equivalent, so the annotation procedure does indeed yield a standardised form with respect to conflict equivalence.

Proposition 3.1 Let G and H be two automata such that $\mathscr{A}(G) = \mathscr{A}(H)$. Then $G \simeq_{\text{conf}} H$.

Example 3.2 In figure 3.2 there are two automata G and G' with equivalent annotated forms. The only difference between G and G' is that the transition $q_1 \xrightarrow{\beta} q_4$ exists in G' but not in G. When the two automata are annotated this difference is removed however. This is because q_1 is only reachable via τ transition and therefore is unreachable in the annotated automaton, furthermore because $q_0 \xrightarrow{\tau} q_2 \xrightarrow{\beta} q_4$ in both automata the transition $q_0 \xrightarrow{\beta} q_4$ is added to the annotated automaton in both cases.

Conversely, it is not true that two conflict equivalent automata have the same annotated forms. Annotations cannot be used to characterise conflict equivalence. This is due to the fact that failures equivalence [16] does not imply conflict equivalence, and the same counterexample as given in [25] applies.

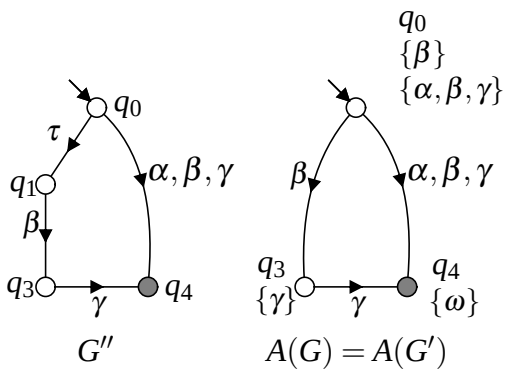


Figure 3.3: An automaton which is conflict-equivalent to the automata in figure 3.2

Example 3.3 The automaton G'' shown in figure 3.3 is conflict equivalent to the automata in figure 3.2 yet it does not have the same annotated form. This is because the annotated automaton of G has the annotation $\{\alpha,\beta\}$ whereas the annotated form of G'' does not. The three automata are conflict equivalent however because all three automata have the annotation $\{\beta\}$ in q_0. The annotation $\{\beta\}$ is strictly more restrictive than the annotation $\{\alpha,\beta\}$. Therefore the annotation $\{\alpha,\beta\}$ is redundant. This is gone into in more detail in section 3.1.3.

In order to prove proposition 3.1 it is necessary to first prove two lemmas that describe the relationship between paths in an automaton and its annotated form.

Lemma 3.1 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an automaton. Let $\mathscr{A}(G) = \langle \Sigma, Q, \rightarrow_A, Q^\circ, Ann \rangle$ be the annotated form of G. For all traces $s \in \Sigma^*$ and all events $\sigma \in \Sigma$, the annotated form has a path $x \xrightarrow{s\sigma}_A z$ if and only if there exists a path $x \stackrel{s}{\Rightarrow} y \xrightarrow{\sigma} z$ in G, for some $y \in Q$.

Proof. The claim is proved by induction on $|s|$.

In the base case, $s = \varepsilon$, the claim follows directly from the definition (3.2).

For the inductive step, let $s = t\sigma'$. Then note,

$$x \xrightarrow{s\sigma}_A z \quad \Longleftrightarrow \quad x \xrightarrow{t\sigma'\sigma}_A z \quad \Longleftrightarrow \quad x \xrightarrow{t\sigma'}_A y \xrightarrow{\sigma}_A z \quad \text{for some } y \in Q\ . \tag{3.4}$$

By inductive assumption, $x \xrightarrow{t\sigma'}_A y$ holds if and only if $x \stackrel{s}{\Rightarrow} y' \xrightarrow{\sigma'} y$ for some $y' \in Q$, and by (3.2) $y \xrightarrow{\sigma}_A z$ holds if and only if $y \stackrel{\varepsilon}{\Rightarrow} z' \xrightarrow{\sigma} z$ for some $z' \in Q$. Thus, (3.4) becomes equivalent to,

$$x \stackrel{s}{\Rightarrow} y' \xrightarrow{\sigma'} y \stackrel{\varepsilon}{\Rightarrow} z' \xrightarrow{\sigma} z \quad \text{for some } y', z' \in Q \quad \Longleftrightarrow \quad x \stackrel{s\sigma'}{\Rightarrow} z' \xrightarrow{\sigma} z \quad \text{for some } z' \in Q\ . \ \square$$

Lemma 3.2 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an automaton. Let $\mathscr{A}(G) = \langle \Sigma, Q, \rightarrow_A, Q^\circ, Ann \rangle$ be the annotated form of G. Also let $x, z \in Q$ and $s \in \Sigma^*$.

(i) If $x \overset{s}{\Rightarrow} z$, then there exists $z' \in Q$ such that $x \overset{s}{\Rightarrow}_A z'$ and $(z', \mathrm{Elig}_G(z)) \in \mathit{Ann}$.

(ii) If $x \overset{s}{\rightarrow}_A z$ and $(z,a) \in \mathit{Ann}$, then there exists $z' \in Q$ such that $x \overset{s}{\Rightarrow} z'$ and $\mathrm{Elig}_G(z') = a$.

Proof. (i) Let $x \overset{s}{\Rightarrow} z$. If $s = \varepsilon$ then $x \overset{\varepsilon}{\Rightarrow} z$, so $x \overset{\varepsilon}{\rightarrow}_A x$ with $(x, \mathrm{Elig}_G(z)) \in \mathit{Ann}$ by definition 3.2 (3.3). Otherwise, $s = t\sigma$ and thus $x \overset{t}{\Rightarrow} y \overset{\sigma}{\rightarrow} z' \overset{\varepsilon}{\Rightarrow} z$ for some $y, z' \in Q$. By lemma 3.1, it follows that $x \overset{t\sigma}{\Rightarrow}_A z'$, and $(z', \mathrm{Elig}_G(z)) \in \mathit{Ann}$ since $z' \overset{\varepsilon}{\Rightarrow} z$.

(ii) Let $x \overset{s}{\rightarrow}_A z$ and $(z,a) \in \mathit{Ann}$. By definition 3.2 (3.3), there exists $z' \in Q$ such that $z \overset{\varepsilon}{\Rightarrow} z'$ and $\mathrm{Elig}_G(z') = a$. If $s = \varepsilon$ then $x = z \overset{\varepsilon}{\Rightarrow} z'$ with $\mathrm{Elig}_G(z') = a$. Otherwise, $s = t\sigma$ and by lemma 3.1, there exists $y \in Q$ such that $x \overset{t}{\Rightarrow} y \overset{\sigma}{\rightarrow} z$. Then $x \overset{s}{\Rightarrow} z \overset{\varepsilon}{\Rightarrow} z'$ with $\mathrm{Elig}_G(z') = a$. □

Given these results, it is now possible to prove proposition 3.1.

Proposition 3.1 Let G and H be two automata such that $\mathscr{A}(G) = \mathscr{A}(H)$. Then $G \simeq_{\mathrm{conf}} H$.

Proof. Let $G = \langle \Sigma, Q_G, \rightarrow_G, Q_G^\circ \rangle$ and $H = \langle \Sigma, Q_H, \rightarrow_H, Q_H^\circ \rangle$, and let $T = \langle \Sigma, Q_T, \rightarrow_T, Q_T^\circ \rangle$ be an arbitrary automaton.

Assume that $G \parallel T$ is nonblocking. It is enough to show that this implies that $H \parallel T$ is nonblocking. Therefore, let $s \in (\Sigma \cup \Sigma_T)^*$ such that $H \parallel T \overset{s}{\Rightarrow} (x_H, x_T)$. Then $H \overset{P(s)}{\Longrightarrow} x_H$ according to (2.5), where $P\colon \Sigma \cup \Sigma_T \to \Sigma$ denotes the natural projection, and by lemma 3.2 (i), there exists a state $x_A \in Q_H$ such that $\mathscr{A}(G) = \mathscr{A}(H) \overset{P(s)}{\longrightarrow} x_A$ and $(x_A, \mathrm{Elig}_H(x_H)) \in \mathit{Ann}_H = \mathit{Ann}_G$. By lemma 3.2 (ii), there also exists a state $x_G \in Q_G$ such that $G \overset{P(s)}{\Longrightarrow} x_G$ and $\mathrm{Elig}_G(x_G) = \mathrm{Elig}_H(x_H)$. Thus, $G \parallel T \overset{s}{\Rightarrow} (x_G, x_T)$.

As $G \parallel T$ is nonblocking, there exists a trace $t \in (\Sigma \cup \Sigma_T)^*$ such that $(x_G, x_T) \overset{t\omega}{\Rightarrow}$. Clearly, $t\omega = u\sigma v$ for some $u \in (\Sigma_T \setminus \Sigma)^*$, $\sigma \in \Sigma_\omega$, and $v \in (\Sigma_\omega \cup \Sigma_T)^*$. Then $x_G \overset{u}{\Rightarrow}_G x_G \overset{\sigma}{\rightarrow}_G$, i.e., $\sigma \in \mathrm{Elig}_G(x_G) = \mathrm{Elig}_H(x_H)$. If $\sigma = \omega$, then clearly $H \parallel T \overset{s}{\Rightarrow} (x_H, x_T) \overset{u\omega}{\Rightarrow}$, which is enough to show that $H \parallel T$ is nonblocking. Otherwise, if $\sigma \in \Sigma$, let $y_H \in Q_H$ such that $H \overset{P(s)}{\Longrightarrow} x_H \overset{\sigma}{\rightarrow} y_H$. By lemma 3.1, this implies $\mathscr{A}(G) = \mathscr{A}(H) \overset{P(s)\sigma}{\longrightarrow} y_H$ and $G \overset{P(s)\sigma}{\Longrightarrow} y_H$. Since $u \in (\Sigma_T \setminus \Sigma)^*$, it also follows that $G \parallel T \overset{su\sigma}{\Longrightarrow} (y_H, y_T)$ for some state y_T of T. Since $G \parallel T$ is nonblocking, there exists a trace $w \in (\Sigma \cup \Sigma_T)^*$ such that $(y_H, y_T) \overset{w\omega}{\Rightarrow}$. Therefore,

$$H \parallel T \overset{s}{\Rightarrow} (x_H, x_T) \overset{u\sigma}{\Rightarrow} (y_H, y_T) \overset{w\omega}{\Rightarrow} \quad . \tag{3.5}$$

Since (x_H, x_T) was chosen arbitrarily, it follows that $H \parallel T$ is nonblocking. □

3.1.2 Unannotation

The annotation procedure can be reversed to obtain an ordinary automaton from a given annotated automaton. The reverse operation is called *unannotation* and is characterised by the following definition.

Definition 3.3 Let $A = \langle \Sigma, Q, \rightarrow, Q^\circ, Ann \rangle$ be an annotated automaton. An *unannotated form* of A is any automaton $U = \langle \Sigma, Q_U, \rightarrow_U, Q^\circ \rangle$ such that the following properties hold.

(i) $Q_U = Q \cup Ann$;

(ii) $x \xrightarrow{\tau}_U (x,a)$ for all $(x,a) \in Ann$, and these are the only τ-transitions in U;

(iii) If $x,y \in Q$, then $x \xrightarrow{\sigma}_U y$ if and only if $x \xrightarrow{\sigma} y$.

(iv) If $(x,a) \in Ann$ and $\sigma \in a$, then $(x,a) \xrightarrow{\sigma}_U$;

(v) If $(x,a) \xrightarrow{\sigma}_U y$, then $\sigma \in a$ and $x \xrightarrow{\sigma} y$.

The state space of an unannotated form consists of all the *original states* of the annotated automaton plus an additional so-called *annotation state* for each annotation (i), which is linked to its original state by a silent transition (ii). Furthermore, the unannotated form contains all the transitions of the annotated automaton (iii). In addition, the annotation states must have outgoing transitions for each event in their respective annotation (iv), and these transitions must lead to some successor state reached by the same event from the corresponding original state (v).

Given an annotated automaton A, an unannotated form can be constructed by including the states and transitions according to (i), (ii), and (iii), and by arbitrarily choosing for each annotation state (x,a) and each event $\sigma \in a$ a transition $x \xrightarrow{\sigma} y$, and then including the transition $(x,a) \xrightarrow{\sigma} y$ in the unannotated form. There are several possibilities to choose transitions satisfying points (iv) and (v), but the ambiguity does not cause problems with conflict-preserving abstraction.

Proposition 3.2 Let A be an annotated automaton, and let U_1 and U_2 be unannotated forms of A. Then $U_1 \simeq_{\text{conf}} U_2$.

This proposition which will be proven at the end of this section, confirms that unannotated forms are well-defined up to conflict equivalence, so the ambiguity in definition 3.3 does not affect the nonblocking property and can be exploited to minimise unannotated forms.

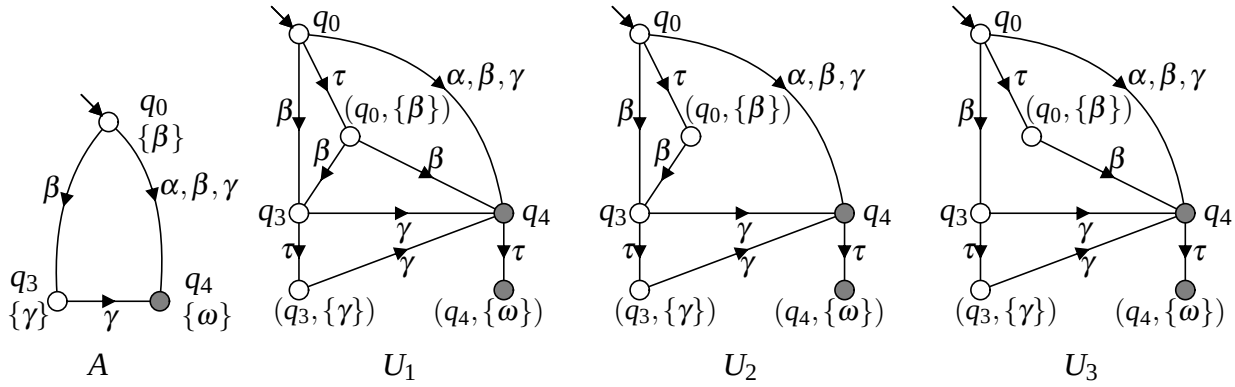


Figure 3.4: Three possible unanotations of an annotated automaton

Example 3.4 Figure 3.4 show an annotated automaton A alongside three possible unannotated forms of A. All three automata the annotations $(q_0,\{\beta\})$, $(q_3,\{\gamma\})$, and $(q_4,\{\omega\})$ are replaced by annotation states. These annotation states are can be reached by τ-transitions as defined in definition 3.3. Each annotation state must have at least one outgoing transition for each event in its annotation. As the annotation states $(q_3,\{\gamma\})$, and $(q_4,\{\omega\})$. Have only one possible outgoing transition which can be chosen in order to fulfill this requirement these states are identical in U_1, U_2, and U_3. The annotation state $(q_0,\{\beta\})$ has two outgoing β-transitions which it can choose from in order to fulfill the requirements of definition 3.3. As such it is possible to created a valid unnanotation of the automaton A by choosing to use either/both transitions.

Example 3.5 In figure 3.1, automaton U is an unannotated form of the annotated automaton A''. The three annotations in A'' have been replaced by annotation states $(q_7,\{\omega\})$ $(q_8,\{\alpha\})$, and $(q_{0125},\{\alpha\})$. Note that the transition $(q_{0125},\{\alpha\}) \xrightarrow{\alpha} q_{0125}$ is not included in U, although it could be inherited from q_{0125}.

Complexity 3.2 Given $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$, an unannotated form of $\mathscr{A}(G)$ has up to $|Q| + |Ann| \leq |Q| + |Q|^2$ states and up to $|\rightarrow| + |Ann| + |Ann||\Sigma_\omega| \leq |Q|^2|\Sigma_\omega|$ transitions. Its space complexity is $O(|Q|^2|\Sigma|)$, and this is also the time complexity to construct it from an annotated automaton. This worst-case is unusual in practice—in the experiments in section 3.2, the number of states after unannotation is almost always less than it was before annotation.

The following result confirms that unannotation is a reverse operation of the annotation procedure, up to conflict equivalence. Conflict equivalence is preserved by annotation and subsequent unannotation.

Proposition 3.3 Let G be an automaton, and let U be an unannotated form of $\mathscr{A}(G)$. Then $U \simeq_{\text{conf}} G$.

In the following sections, different methods are presented to simplify annotated automata. The simplification needs to be carried out in a conflict-preserving way, and this requires an appropriate notion of conflict equivalence of annotated automata. The following definition is justified by propositions 3.2 and 3.3, and by the fact that every annotated automaton has an unannotated form.

Definition 3.4 The two annotated automata A_1 and A_2 are conflict equivalent, written $A_1 \simeq_{\text{conf}} A_2$, if for every unannotated form U_1 of A_1 and for every unannotated form U_2 of A_2 it holds that $U_1 \simeq_{\text{conf}} U_2$.

It is now necessary to prove the two key results about unannotation. Unannotated forms are equal with respect to conflict equivalence (proposition 3.2), and conflict equivalence is preserved when annotating and unannotating again (proposition 3.3).

These results depend on the relationship between traces in an annotated automaton and its unannotated forms, which are first established. Lemma 3.3 shows that every nonempty path of an annotated automaton corresponds to an equivalent path of its unannotated form. Lemma 3.4 lifts this result to all paths of an unannotated form, considering separately the cases of original and annotation end states.

Lemma 3.3 Let $A = \langle \Sigma, Q, \rightarrow, Q^\circ, Ann \rangle$ be an annotated automaton, and let $U = \langle \Sigma, Q_U, \rightarrow_U, Q^\circ \rangle$ be an unannotated form of A. For all traces $s \in \Sigma^*$, all events $\sigma \in \Sigma$, and all states $x \in Q$, it holds that $x \xrightarrow{s\sigma} z$ if and only if $x \overset{s}{\Rightarrow}_U y \xrightarrow{\sigma}_U z$ for some $y \in Q_U$.

Proof. The claim is proved by induction on $|s|$.

First consider the base case $s = \varepsilon$. If $x \xrightarrow{\sigma} z$, it follows directly from definition 3.3 (iii) that $x \xrightarrow{\sigma}_U z$. Conversely, if $x \overset{\varepsilon}{\Rightarrow}_U y \xrightarrow{\sigma}_U z$, then by definition 3.3 (ii) either $x = y$ or $x \xrightarrow{\tau}_U y$. If $x = y \xrightarrow{\sigma}_U z$, then $x \xrightarrow{\sigma} z$ by definition 3.3 (iii). If $x \xrightarrow{\tau}_U y$, then $y = (x,a) \in Ann$ by definition 3.3 (ii), and $(x,a) = y \xrightarrow{\sigma}_U z$ implies $x \xrightarrow{\sigma} z$ by definition 3.3 (v).

For the inductive step, let $s = t\sigma'$, and first assume $x \xrightarrow{t\sigma'} y \xrightarrow{\sigma} z$. By inductive assumption, it follows that $x \overset{t\sigma'}{\Rightarrow}_U y$, and by definition 3.3 (iii) it holds that $y \xrightarrow{\sigma}_U z$. This implies $x \overset{t\sigma'}{\Rightarrow}_U y \xrightarrow{\sigma}_U z$. Conversely, assume that $x \overset{t\sigma'}{\Rightarrow}_U y \xrightarrow{\sigma}_U z$, i.e.,

$$x \overset{t}{\Rightarrow}_U x' \xrightarrow{\sigma'}_U y' \overset{\varepsilon}{\Rightarrow}_U y \xrightarrow{\sigma}_U z \,. \tag{3.6}$$

Then $x \xrightarrow{t\sigma'} y'$ by inductive assumption, and by definition 3.3 (ii), it either holds that $y' = y$, and thus $y' \xrightarrow{\sigma}_U z$, which implies $y' \xrightarrow{\sigma} z$ by definition 3.3 (iii); or there is an annotation $(y',a) \in Ann$ such that $y = (y',a)$, i.e., $(y',a) \xrightarrow{\sigma}_U z$ and thus $y' \xrightarrow{\sigma} z$ by definition 3.3 (v). In both cases, $x \xrightarrow{t\sigma'} y' \xrightarrow{\sigma} z$, i.e., $x \xrightarrow{s\sigma} z$. □

Lemma 3.4 Let $A = \langle \Sigma, Q, \rightarrow, Q^\circ, Ann \rangle$ be an annotated automaton, and let $U = \langle \Sigma, Q_U, \rightarrow_U, Q^\circ \rangle$ be an unannotated form of A.

(i) For all traces $s \in \Sigma^*$ and all states $x \in Q$, it holds that $A \xrightarrow{s} x$ if and only if $U \overset{s}{\Rightarrow} x$.

(ii) For all traces $s \in \Sigma^*$ and all annotations $(x,a) \in Ann$, it holds that $A \xrightarrow{s} x$ if and only if $U \overset{s}{\Rightarrow} (x,a)$.

Proof. (i) Firstly, if $s = \varepsilon$, then $A \xrightarrow{\varepsilon} x$ implies $x \in Q^\circ$ and thus $U \overset{\varepsilon}{\Rightarrow} x$, and conversely $U \overset{\varepsilon}{\Rightarrow} x$ with $x \in Q$ implies $x \in Q^\circ$ by definition 3.3 (ii) and thus $A \xrightarrow{\varepsilon} x$. Secondly, if $s = t\sigma$, the claim follows immediately from lemma 3.3.

(ii) Let $(x,a) \in Ann$. Then $x \xrightarrow{\tau}_U (x,a)$ by definition 3.3 (ii), and this is the only way how (x,a) can be reached in U. Then the claim follows from (i), because $x \in Q$ and thus $A \xrightarrow{s} x$ if and only if $U \overset{s}{\Rightarrow} x \xrightarrow{\tau} (x,a)$. □

The result that two unannotated forms of the same annotated automaton are conflict equivalent now becomes a consequence of lemmas 3.3 and 3.4.

Proposition 3.2 Let A be an annotated automaton, and let U_1 and U_2 be unannotated forms of A. Then $U_1 \simeq_{\text{conf}} U_2$.

Proof. Let $A = \langle \Sigma, Q, \to, Q^\circ, Ann \rangle$, and let $U_i = \langle \Sigma, Q \cup Ann, \to_i, Q^\circ \rangle$ for $i = 1,2$ be unannotated forms of A. Furthermore, let $T = \langle \Sigma, Q_T, \to_T, Q_T^\circ \rangle$ be an arbitrary automaton such that $U_1 \parallel T$ is nonblocking. It is enough to show that this implies that $U_2 \parallel T$ is nonblocking. Therefore, let $s \in (\Sigma \cup \Sigma_T)^*$ such that $U_2 \parallel T \overset{s}{\Rightarrow} (x, x_T)$, and consider two cases.

Case 1: $x = (x_a, a) \in Ann$. Then $U_2 \overset{P(s)}{\Longrightarrow} (x_a, a)$, which implies $A \xrightarrow{P(s)} x_a$ and $U_1 \overset{P(s)}{\Longrightarrow} (x_a, a)$ by lemma 3.4 (ii). Thus $U_1 \parallel T \overset{s}{\Rightarrow} ((x_a,a), x_T)$, and since $U_1 \parallel T$ is nonblocking, there exists $t \in \Sigma^*$ such that $U_1 \parallel T \overset{s}{\Rightarrow} ((x_a,a), x_T) \overset{t\omega}{\Rightarrow}$. Write $t\omega = u\sigma v$ with $u \in (\Sigma_T \setminus \Sigma)^*$, $\sigma \in \Sigma_\omega$, and $v \in (\Sigma_\omega \cup \Sigma_T)^*$. Then $U_1 \parallel T \overset{s}{\Rightarrow} ((x_a,a), x_T) \overset{u}{\Rightarrow} ((x_a,a), x_T') \xrightarrow{\sigma} (y_1, y_T)$, so $\sigma \in \text{Elig}_{U_1}((x_a,a)) = a = \text{Elig}_{U_2}((x_a,a))$ by definition 3.3 (iv) and (v), and thus $(x_a,a) \xrightarrow{\sigma}_2 y_2$ for some $y_2 \in Q$. Thus $U_2 \parallel T \overset{s}{\Rightarrow} ((x_a,a), x_T) \overset{u}{\Rightarrow} ((x_a,a), x_T') \xrightarrow{\sigma} (y_2, y_T)$. If $\sigma = \omega$, then clearly $U_2 \parallel T \overset{s}{\Rightarrow} ((x_a,a), x_T) \overset{u\omega}{\Rightarrow}$, which is enough to show that $U_2 \parallel T$ is nonblocking. Otherwise, $U_2 \parallel T \overset{su\sigma}{\Longrightarrow} (y_2, y_T)$ with $su\sigma \in (\Sigma \cup \Sigma_T)^*$ and $y_2 \in Q$, and the proof continues as in *Case 2*.

Case 2: $x \in Q$. Then $U_2 \overset{P(s)}{\Longrightarrow} x$ implies $A \xrightarrow{P(s)} x$ and $U_1 \overset{P(s)}{\Longrightarrow} x$ by lemma 3.4 (i). Thus $U_1 \parallel T \overset{s}{\Rightarrow} (x, x_T)$, and since $U_1 \parallel T$ is nonblocking, there exists $w \in \Sigma^*$ such that $U_1 \parallel T \overset{s}{\Rightarrow} (x, x_T) \overset{w\omega}{\Rightarrow} (y, y_T)$ where $y \in Q$. Therefore $x \overset{P(w)\omega}{\Longrightarrow}_1 y$, which implies $x \xrightarrow{P(w)\omega} y$ and $x \overset{P(w)\omega}{\Longrightarrow}_2 y$ by lemma 3.3. Then $U_2 \parallel T \overset{s}{\Rightarrow} (x, x_T) \overset{w\omega}{\Rightarrow}$, and since (x, x_T) was chosen arbitrarily, it follows that $U_2 \parallel T$ is nonblocking. □

The second main result about unannotation is that conflict equivalence is preserved when annotation is followed by unannotation. To prove this, it is helpful to first establish a lemma about annotations, namely that the annotated form of an automaton

is equal to the annotated form of its unannotation. Due to the way annotated forms are defined in this thesis, lemma 3.5 only applies to annotated forms of an ordinary automaton G, not to arbitrary annotated automata.

Lemma 3.5 Let G be an automaton, and let U be an unannotated form of $\mathscr{A}(G)$. Then $\mathscr{A}(U) = \mathscr{A}(G)$.

Proof. Let $\mathscr{A}(G) = \langle \Sigma, Q, \to, Q^\circ, Ann \rangle$, let $U = \langle \Sigma, Q_U, \to_U, Q^\circ \rangle$ be an unannotated form of $\mathscr{A}(G)$, and let $\mathscr{A}(U) = \langle \Sigma, Q_U, \to_{\mathscr{A}(U)}, Q^\circ, Ann_{\mathscr{A}(U)} \rangle$. It will be shown that the reachable parts of $\mathscr{A}(G)$ and $\mathscr{A}(U)$ are equal, i.e., that $\to = \to_{\mathscr{A}(U)|Q}$ and $Ann = Ann_{\mathscr{A}(U)|Q}$, where $\to_{\mathscr{A}(U)|Q} = \to_{\mathscr{A}(U)} \cap (Q \times \Sigma_\omega \times Q_U)$ and $Ann_{\mathscr{A}(U)|Q} = Ann_{\mathscr{A}(U)} \cap (Q \times 2^{\Sigma_\omega})$.

First, let $x \xrightarrow{\sigma} y$. Then $x \in Q$ and $x \xrightarrow{\sigma}_U y$ by definition 3.3 (iii), and $x \xrightarrow{\sigma}_{\mathscr{A}(U)} y$ by definition 3.2 (3.2), and $x \xrightarrow{\sigma}_{\mathscr{A}(U)|Q} y$ as $x \in Q$.

Conversely, let $x \xrightarrow{\sigma}_{\mathscr{A}(U)|Q} y$. Then $x \in Q$ and $x \overset{\varepsilon}{\Rightarrow}_U z \xrightarrow{\sigma}_U y$ for some $z \in Q_U$ by definition 3.2 (3.2). By definition 3.3 (ii), this means that either $x = z$, which implies $x \xrightarrow{\sigma}_U y$ and $x \xrightarrow{\sigma} y$ by definition 3.3 (iii), or $z = (x, a) \xrightarrow{\sigma}_U y$, which implies $x \xrightarrow{\sigma} y$ by definition 3.3 (v).

Second, let $(x, a) \in Ann$. Then $x \in Q$ and $x \xrightarrow{\tau}_U (x, a)$ by definition 3.3 (ii) and $\mathrm{Elig}_U((x, a)) = a$ by definition 3.3 (iv) and (v). By definition 3.2 (3.3), it follows that $(x, a) = (x, \mathrm{Elig}_U((x, a))) \in Ann_{\mathscr{A}(U)|Q}$.

Conversely, let $(x, a) \in Ann_{\mathscr{A}(U)|Q}$. Then $x \in Q$, and by definition 3.2 (3.3), there exists $y \in Q_U$ such that $x \overset{\varepsilon}{\Rightarrow}_U y$ and $\mathrm{Elig}_U(y) = a$. Here, $x \overset{\varepsilon}{\Rightarrow}_U y$ means that either $x = y$ or $x \xrightarrow{\tau}_U y$.

In the case $x = y$, note that $y = x \in Q$, and $\mathrm{Elig}_U(y) = \mathrm{Elig}_A(y) \cup \bigcup_{(z,a) \in Ann} a = \mathrm{Elig}_A(y)$ by definition 3.1 (ii), and $\mathrm{Elig}_A(y) = \mathrm{Elig}_G(y)$ by definition 3.2 (3.2). Therefore, $(x, a) = (y, \mathrm{Elig}_U(y)) = (y, \mathrm{Elig}_A(y)) = (y, \mathrm{Elig}_G(y)) \in Ann$.

In the case $x \xrightarrow{\tau}_U y$, note that $y \in Ann$ by definition 3.3 (ii). Then it follows from $\mathrm{Elig}_U(y) = a$ by definition 3.3 (iv) and (v) that $(x, a) = y \in Ann$. □

Proposition 3.3 Let G be an automaton, and let U be an unannotated form of $\mathscr{A}(G)$. Then $U \simeq_{\mathrm{conf}} G$.

Proof. By lemma 3.5, it holds that $\mathscr{A}(U) = \mathscr{A}(G)$, which implies $U \simeq_{\mathrm{conf}} G$ by proposition 3.1. □

3.1.3 Subsumption

Annotations are sets of events that must be enabled to avert blocking. More precisely, when a state is entered, at least one of the events in each of its annotations needs to be

enabled in order to avert blocking. This leads to the observation that certain annotations are redundant. For example, if a state has both the annotations $\{\alpha\}$ and $\{\alpha,\beta\}$, then the latter is implied by the former. The state already requires event α to be enabled, so the fact that α or β needs to be enabled adds no additional information. The annotation $\{\alpha,\beta\}$, being a superset of $\{\alpha\}$, is said to be covered or *subsumed* by $\{\alpha\}$, and subsumed annotations can be removed without affecting conflict equivalence.

This gives rise to the following *subsumption rule*: if an annotated automaton contains annotations (x,a) and (x,b) such that $a \subsetneqq b$, then the annotation (x,b) can be removed. The removal of subsumed annotations from an annotated automaton preserves conditions (i) and (ii) in definition 3.1, because no annotations are added and annotations can only be removed from states that have more than one annotation.

Example 3.6 In state q_0 of automaton $\mathscr{A}(G)$ in figure 3.1, the annotation $\{\alpha\}$ subsumes $\{\alpha,\beta\}$ and $\{\alpha,\beta,\gamma\}$, and the annotation $\{\alpha\}$ in state q_1 subsumes $\{\alpha,\beta,\gamma\}$. The subsumed annotations are struck out in the figure.

Proposition 3.4 Let $A = \langle \Sigma, Q, \rightarrow, Q^\circ, Ann\rangle$ and $A_{\text{sub}} = \langle \Sigma, Q, \rightarrow, Q^\circ, Ann_{\text{sub}}\rangle$ be two annotated automata such that $Ann_{\text{sub}} \subseteq Ann$ and for all $(x,a) \in Ann$ there exists $a_{\text{sub}} \subseteq a$ such that $(x,a_{\text{sub}}) \in Ann_{\text{sub}}$. Then $A \simeq_{\text{conf}} A_{\text{sub}}$.

Proof. Let $U = \langle \Sigma, Q \cup Ann, \rightarrow_U, Q^\circ\rangle$ and $U_{\text{sub}} = \langle \Sigma, Q \cup Ann_{\text{sub}}, \rightarrow_{U,\text{sub}}, Q^\circ\rangle$ be unannotated forms of A and A_{sub}, respectively. It is to be shown that $U \simeq_{\text{conf}} U_{\text{sub}}$. Therefore, let $T = \langle \Sigma, Q_T, \rightarrow_T, Q_T^\circ\rangle$ be an arbitrary automaton.

First, assume that $U \parallel T$ is nonblocking, and let $s \in (\Sigma \cup \Sigma_T)^*$ such that $U_{\text{sub}} \parallel T \stackrel{s}{\Rightarrow} (x,x_T)$. Then $U_{\text{sub}} \stackrel{P(s)}{\Longrightarrow} x \in Q \cup Ann_{\text{sub}}$. Consider two cases.

Case 1: $x = (x_a,a) \in Ann_{\text{sub}}$. From $U_{\text{sub}} \stackrel{P(s)}{\Longrightarrow} x = (x_a,a)$, it follows that $A_{\text{sub}} \stackrel{P(s)}{\longrightarrow} x_a$ by lemma 3.4 (ii), which implies $A \stackrel{P(s)}{\longrightarrow} x_a$ because A and A_{sub} have the same transition relations. Furthermore, since $(x_a,a) \in Ann_{\text{sub}} \subseteq Ann$, it follows by lemma 3.4 (ii) that $U \stackrel{P(s)}{\Longrightarrow} (x_a,a)$. This implies $U \parallel T \stackrel{s}{\Rightarrow} ((x_a,a),x_T)$, and since $U \parallel T$ is nonblocking, there exists $t \in \Sigma^*$ such that $U \parallel T \stackrel{s}{\Rightarrow} ((x_a,a),x_T) \stackrel{t\omega}{\Rightarrow}$. Write $t\omega = u\sigma v$ with $u \in (\Sigma_T \setminus \Sigma)^*$, $\sigma \in \Sigma_\omega$, and $v \in (\Sigma_\omega \cup \Sigma_T)^*$. Then $U \parallel T \stackrel{s}{\Rightarrow} ((x_a,a),x_T) \stackrel{u}{\Rightarrow} ((x_a,a),x_T') \stackrel{\sigma}{\rightarrow} (y,y_T)$, so $\sigma \in \text{Elig}_U((x_a,a)) = a = \text{Elig}_{U_{\text{sub}}}((x_a,a))$ by definition 3.3 (iv) and (v), and $(x_a,a) \stackrel{\sigma}{\rightarrow}_{U,\text{sub}} y_{\text{sub}}$ for some $y_{\text{sub}} \in Q$. If $\sigma = \omega$, then clearly $U_{\text{sub}} \parallel T \stackrel{s}{\Rightarrow} ((x_a,a),x_T) \stackrel{u\omega}{\Rightarrow}$, which is enough to show that $U_{\text{sub}} \parallel T$ is nonblocking. Otherwise, $U_{\text{sub}} \parallel T \stackrel{su\sigma}{\Longrightarrow} (y_{\text{sub}},y_T)$ with $su\sigma \in (\Sigma \cup \Sigma_T)^*$ and $y_{\text{sub}} \in Q$, and the proof continues as in *Case 2*.

Case 2: $x \in Q$. From $U_{\text{sub}} \stackrel{P(s)}{\Longrightarrow} x$, it follows that $A_{\text{sub}} \stackrel{P(s)}{\longrightarrow} x$ by lemma 3.4 (i), which implies $A \stackrel{P(s)}{\longrightarrow} x$ because A and A_{sub} have the same transition relations, which implies $U \stackrel{P(s)}{\Longrightarrow} x$ again by lemma 3.4 (i). Then $U \parallel T \stackrel{s}{\Rightarrow} (x,x_T)$, and since $U \parallel T$ is nonblocking,

there exists $w \in \Sigma^*$ such that $U \parallel T \stackrel{s}{\Rightarrow} (x, x_T) \stackrel{w\omega}{\Rightarrow} (z, z_T)$. This means $x \stackrel{P(w)\omega}{\Longrightarrow}_U z$, which implies $x \stackrel{P(w)\omega}{\longrightarrow} z$ by lemma 3.3, which implies $x \stackrel{P(w)\omega}{\longrightarrow}_{\text{sub}} z$ because A and A_{sub} have the same transition relations, which implies $x \stackrel{P(w)\omega}{\longrightarrow}_{U,\text{sub}} z$ again by lemma 3.3. Thus, $U_{\text{sub}} \parallel T \stackrel{s}{\Rightarrow} (x, x_T) \stackrel{w\omega}{\Rightarrow}$, and since (x, x_T) was chosen arbitrarily, it follows that $U_{\text{sub}} \parallel T$ is nonblocking.

Conversely, assume that $U_{\text{sub}} \parallel T$ is nonblocking, and let $s \in (\Sigma \cup \Sigma_T)^*$ such that $U \parallel T \stackrel{s}{\Rightarrow} (x, x_T)$. Then $U \stackrel{P(s)}{\Longrightarrow} x \in Q \cup Ann$. Consider two cases.

Case 1: $x = (x_a, a) \in Ann$. By assumption there exists $a_{\text{sub}} \subseteq a$ such that $(x_a, a_{\text{sub}}) \in Ann_{\text{sub}}$. From $U \stackrel{P(s)}{\Longrightarrow} x = (x_a, a)$, it follows that $A \stackrel{P(s)}{\longrightarrow} x_a$ by lemma 3.4 (ii), which implies $A_{\text{sub}} \stackrel{P(s)}{\longrightarrow} x_a$ because A and A_{sub} have the same transition relations. Therefore, $U_{\text{sub}} \stackrel{P(s)}{\Longrightarrow} x_a \stackrel{\tau}{\rightarrow} (x_a, a_{\text{sub}})$ by lemma 3.4 (i) and by definition 3.3 (ii). Thus, $U_{\text{sub}} \parallel T \stackrel{s}{\Rightarrow} ((x_a, a_{\text{sub}}), x_T)$, and since $U_{\text{sub}} \parallel T$ is nonblocking, there exists $t \in \Sigma^*$ such that $U_{\text{sub}} \parallel T \stackrel{s}{\Rightarrow} ((x_a, a_{\text{sub}}), x_T) \stackrel{t\omega}{\Rightarrow}$. Write $t\omega = u\sigma v$ with $u \in (\Sigma_T \setminus \Sigma)^*$, $\sigma \in \Sigma_\omega$, and $v \in (\Sigma_\omega \cup \Sigma_T)^*$. Then $U_{\text{sub}} \parallel T \stackrel{s}{\Rightarrow} ((x_a, a_{\text{sub}}), x_T) \stackrel{u}{\Rightarrow} ((x_a, a_{\text{sub}}), x'_T) \stackrel{\sigma}{\rightarrow} (y_{\text{sub}}, y_T)$, i.e., $\sigma \in \text{Elig}_{U_{\text{sub}}}((x_a, a_{\text{sub}})) = a_{\text{sub}} \subseteq a = \text{Elig}_U((x_a, a))$ by definition 3.3 (iv) and (v), and $(x_a, a) \stackrel{\sigma}{\rightarrow}_U y$ for some $y \in Q$. If $\sigma = \omega$, then clearly $U \parallel T \stackrel{s}{\Rightarrow} ((x_a, a), x_T) \stackrel{u\omega}{\Rightarrow}$, which is enough to show that $U \parallel T$ is nonblocking. Otherwise, $U \parallel T \stackrel{su\sigma}{\Longrightarrow} (y, y_T)$ with $su\sigma \in (\Sigma \cup \Sigma_T)^*$ and $y \in Q$, and the proof continues as in *Case 2*.

Case 2: $x \in Q$. From $U \stackrel{P(s)}{\Longrightarrow} x$, it follows that $A \stackrel{P(s)}{\longrightarrow} x$ by lemma 3.4 (i), which implies $A_{\text{sub}} \stackrel{P(s)}{\longrightarrow} x$ because A and A_{sub} have the same transition relations, which implies $U_{\text{sub}} \stackrel{P(s)}{\Longrightarrow} x$ again by lemma 3.4 (i). Then $U_{\text{sub}} \parallel T \stackrel{s}{\Rightarrow} (x, x_T)$, and since $U_{\text{sub}} \parallel T$ is nonblocking, there exists $w \in \Sigma^*$ such that $U_{\text{sub}} \parallel T \stackrel{s}{\Rightarrow} (x, x_T) \stackrel{w\omega}{\Rightarrow} (z, z_T)$. This means $x \stackrel{P(w)\omega}{\Longrightarrow}_{U,\text{sub}} z$, which by lemma 3.3 implies $x \stackrel{P(w)\omega}{\longrightarrow} z$, both in A and A_{sub}, and $x \stackrel{P(w)\omega}{\longrightarrow}_U z$. Thus, $U \parallel T \stackrel{s}{\Rightarrow} (x, x_T) \stackrel{w\omega}{\Rightarrow}$, and since (x, x_T) was chosen arbitrarily, it follows that $U \parallel T$ is nonblocking. □

Complexity 3.3 The annotated form $\mathscr{A}(G)$ of $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ has up to $|Q|$ annotations per state, which gives $O(|Q|^2)$ subsumption tests per state, and the cost of each test is $O(|\Sigma|)$. So the worst-case time complexity of the subsumption test for $\mathscr{A}(G)$ is $O(|Q|^3|\Sigma|)$. This makes subsumption one of the most expensive of the abstractions presented here, but experimental results show that it is worthwhile. The subsumption test is best done immediately while constructing annotated automata or introducing annotations, considerably reducing memory requirements.

3.1.4 Incoming Equivalence

Incoming equivalence [12] identifies two states as equivalent if they have exactly the same incoming transitions. The concept is extended to annotated automata as follows.

Definition 3.5 Let $A = \langle \Sigma, Q, \to, Q^\circ, Ann \rangle$ be an annotated automaton. The incoming equivalence relation $\sim_{\text{inc}} \subseteq Q \times Q$ is defined such that $x \sim_{\text{inc}} y$ if and only if the following conditions hold.

- $x \in Q^\circ$ if and only if $y \in Q^\circ$;
- For all states $z \in Q$ and all events $\sigma \in \Sigma_\omega$, it holds that $z \xrightarrow{\sigma} x$ if and only if $z \xrightarrow{\sigma} y$.

In [12], incoming equivalence is used as a restriction to make certain simplification rules applicable. Due to the improved regularity achieved by annotations, all incoming equivalent states in an annotated automaton can be merged. This merging is done using the standard automaton quotient, with the addition that, when merging several states into one, the resultant state receives the annotations of all original states.

Definition 3.6 Let $A = \langle \Sigma, Q, \to, Q^\circ, Ann \rangle$ be an annotated automaton, and let $\sim \subseteq Q \times Q$ be an equivalence relation. The *quotient automaton* of A modulo $\sim$ is $A/\!\sim = \langle \Sigma, Q/\!\sim, \to/\!\sim, \tilde{Q}^\circ, \tilde{A}nn \rangle$, where

$$\to/\!\sim = \{\, ([x], \sigma, [y]) \mid x \xrightarrow{\sigma} y \,\}\,; \tag{3.7}$$

$$\tilde{Q}^\circ = \{\, [x] \mid x \in Q^\circ \,\}\,; \tag{3.8}$$

$$\tilde{A}nn = \{\, ([x], a) \mid x \in Q \text{ and there exists } x' \sim x \text{ such that } (x', a) \in Ann \,\}\,. \tag{3.9}$$

Here, $[x] = \{\, x' \in Q \mid x' \sim x \,\}$ denotes the *equivalence class* of $x \in Q$ with respect to $\sim$, and $Q/\!\sim = \{\, [x] \mid x \in Q \,\}$ is the set of equivalence classes modulo $\sim$.

It is easily confirmed that the quotient $A/\!\sim$ of an annotated automaton A satisfies conditions (i) and (ii) in definition 3.1, because every merged state receives annotations from all its original states, and the eligible events sets are increased when merging.

Proposition 3.5 Let $A = \langle \Sigma, Q, \to, Q^\circ, Ann \rangle$ be an annotated automaton. Then $A \simeq_{\text{conf}} A/\!\sim_{\text{inc}}$.

This result is proven later on in this section. The merging of incoming equivalent states can be considered as a generalisation of the silent continuation rule for normal automata. This rule states that all incoming equivalent states which have outgoing τ-transitions can be merges [12]. An annotation symbolises a silent transition to an

implicit state. When incoming equivalent states are merged, the nondeterministic decisions of the predecessor states are deferred by one step, expressed by the merged annotations.

Example 3.7 The annotated automaton A' in figure 3.1 is the result of using incoming equivalence to simplify $\mathscr{A}(G)$. States q_2 and q_5 are incoming equivalent and have been merged. The resultant state q_{25} receives the annotations $\{\alpha\}$ and $\{\alpha,\beta,\gamma\}$, but only $\{\alpha\}$ remains because of subsumption.

Complexity 3.4 The complexity of partitioning an automaton based on incoming equivalence is $O(|Q|^2|\Sigma|)$. Two states are equivalent if they have equal sets of incoming transitions, which can be determined efficiently using hash codes. Hash codes can be set up in a single pass over all transitions of the automaton, of which there are up to $|Q|^2|\Sigma_\omega|$, and the construction of the simplified automaton is achieved by another loop over all transitions, in the same complexity [12]. However, the merging of some states may make other states incoming equivalent, so the abstraction should be repeated to ensure a minimal result. The maximum number of iterations is $|Q|$, as each merge except the last reduces the number of states, so the complexity to obtain a minimal abstraction by incoming equivalence is $O(|Q|^3|\Sigma|)$.

To prove the correctness of abstractions based on automaton quotients, such as the incoming equivalence abstraction, the relationship between the traces in an automaton A and its quotient $A/\!\sim$ needs to be established. It is well-known that every trace in A also has a corresponding trace in $A/\!\sim$. The following lemma 3.6 is quoted from [12] and holds for every equivalence relation. Conversely, not every path in a quotient automaton exists in the original automaton, but lemma 3.7 shows how such a path can be obtained if the quotient is constructed using incoming equivalence.

Lemma 3.6 Let $A = \langle \Sigma, Q, \to, Q^\circ, Ann\rangle$ be an annotated automaton, and let $\sim \subseteq Q \times Q$ be an equivalence relation. Then, for all states $x,y \in Q$ and all traces $s \in \Sigma^*$ such that $x \xrightarrow{s} y$ in A, it holds that $[x] \xrightarrow{s} [y]$ in $A/\!\sim$.

Proof. Let $x \xrightarrow{s} y$ in A with $s = \sigma_1 \ldots \sigma_n$. Then there exists states $x_0, \ldots, x_n \in Q$ such that

$$x = x_0 \xrightarrow{\sigma_1} x_1 \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_n} x_n = y\ . \tag{3.10}$$

By definition 3.6, it holds that $[x_{k-1}] \xrightarrow{\sigma_k} [x_k]$ for each $k = 1, \ldots, n$, which implies $[x] \xrightarrow{s} [y]$ in $A/\!\sim$. □

Lemma 3.7 Let $A = \langle \Sigma, Q, \to, Q^\circ, Ann\rangle$ be an annotated automaton, and let $\tilde{x}, \tilde{z} \in Q/\!\sim_{\text{inc}}$ be two states of $A/\!\sim_{\text{inc}}$.

(i) For all $s \in \Sigma^*$ and all $\sigma \in \Sigma$ such that $\tilde{x} \xrightarrow{s\sigma} \tilde{z}$, there exists $x \in \tilde{x}$ such that for all $z' \in \tilde{z}$ it holds that $x \xrightarrow{s\sigma} z'$.

(ii) For all $s \in \Sigma^*$ such that $A/{\sim_{\text{inc}}} \xrightarrow{s} \tilde{z}$ and for all $z' \in \tilde{z}$, it holds that $A \xrightarrow{s} z'$.

Proof. (i) The claim is proven by induction on $|s|$.

Base case: $s = \varepsilon$. As $\tilde{x} \xrightarrow{\sigma} \tilde{z}$, there must exist $x \in \tilde{x}$ and $z \in \tilde{z}$ such that $x \xrightarrow{\sigma} z$. Let $z' \in \tilde{z}$. Then $z \sim_{\text{inc}} z'$, and it follows from definition 3.5 that $x \xrightarrow{\sigma} z'$.

Inductive step: $s = t\sigma$. Assume that $\tilde{x} \xrightarrow{t} \tilde{y} \xrightarrow{\sigma} \tilde{z}$. Then there are states $y \in \tilde{y}$ and $z \in \tilde{z}$ such that $y \xrightarrow{\sigma} z$. By inductive assumption, there exists a state $x \in \tilde{x}$ such that $x \xrightarrow{t} y$. Let $z' \in \tilde{z}$. Then $z \sim_{\text{inc}} z'$, and it follows from definition 3.5 that $x \xrightarrow{t} y \xrightarrow{\sigma} z'$.

(ii) Let $\tilde{Q}^\circ = \{\, [x^\circ] \mid x^\circ \in Q^\circ \,\}$ be the set of initial states of $A/{\sim_{\text{inc}}}$.

If $s = \varepsilon$, then $\tilde{z} \in \tilde{Q}^\circ$ and thus $\tilde{z} = [x^\circ]$ for some $x^\circ \in Q^\circ$, which implies $x^\circ \in \tilde{z}$. Let $z' \in \tilde{z}$. Then $x^\circ \sim_{\text{inc}} z'$, which implies $z' \in Q^\circ$ by definition 3.5 and thus $A \xrightarrow{\varepsilon} z'$.

Otherwise $s = t\sigma$ for some $t \in \Sigma^*$ and $\sigma \in \Sigma$, and there exists $\tilde{x} \in \tilde{Q}^\circ$ such that $\tilde{x} \xrightarrow{t\sigma} \tilde{z}$. Let $z' \in \tilde{z}$. It follows from (i) that there exists $x \in \tilde{x}$ such that $x \xrightarrow{t\sigma} z'$. Since $\tilde{x} \in \tilde{Q}^\circ$, there exists $x^\circ \in \tilde{x}$ such that $x^\circ \in Q^\circ$. Then $x^\circ \sim_{\text{inc}} x$ implies $x \in Q^\circ$ and thus $A \xrightarrow{t\sigma} z'$. □

Using the above two lemmas and the properties of the paths of unannotated forms established in section **??**, the proof of proposition 3.5 proceeds using similar ideas to that of the *Active Events Rule* [12].

Proposition 3.5 Let $A = \langle \Sigma, Q, \rightarrow, Q^\circ, Ann \rangle$ be an annotated automaton. Then $A \simeq_{\text{conf}} A/{\sim_{\text{inc}}}$.

Proof. Let $U = \langle \Sigma, Q \cup Ann, \rightarrow_U, Q^\circ \rangle$ and $\tilde{U} = \langle \Sigma, Q/{\sim_{\text{inc}}} \cup \tilde{A}nn, \rightarrow_{\tilde{U}}, \tilde{Q}^\circ \rangle$ be unannotated forms of A and $\tilde{A} = A/{\sim_{\text{inc}}}$, respectively. It is to be shown that $U \simeq_{\text{conf}} \tilde{U}$. Therefore, let $T = \langle \Sigma, Q_T, \rightarrow_T, Q_T^\circ \rangle$ be an arbitrary automaton.

First, assume that $U \parallel T$ is nonblocking, and let $s \in (\Sigma \cup \Sigma_T)^*$ such that $\tilde{U} \parallel T \overset{s}{\Rightarrow} (\tilde{x}, x_T)$. Then $\tilde{U} \overset{P(s)}{\Longrightarrow} \tilde{x} \in Q/{\sim_{\text{inc}}} \cup \tilde{A}nn$. Consider two cases.

Case 1: $\tilde{x} = (\tilde{x}_a, a) \in \tilde{A}nn$. Then there exists $x_a \in \tilde{x}_a$ such that $(x_a, a) \in Ann$. From $\tilde{U} \overset{P(s)}{\Longrightarrow} \tilde{x} = (\tilde{x}_a, a)$, it follows that $\tilde{A} \overset{P(s)}{\longrightarrow} \tilde{x}_a$ by lemma 3.4 (ii), which implies $A \overset{P(s)}{\longrightarrow} x_a$ by lemma 3.7 (ii), and $U \overset{P(s)}{\Longrightarrow} (x_a, a)$ again by lemma 3.4 (ii). Thus, $U \parallel T \overset{s}{\Rightarrow} ((x_a, a), x_T)$, and since $U \parallel T$ is nonblocking, there exists $t \in \Sigma^*$ such that $U \parallel T \overset{s}{\Rightarrow} ((x_a, a), x_T) \overset{t\omega}{\Rightarrow}$. Write $t\omega = u\sigma v$ with $u \in (\Sigma_T \setminus \Sigma)^*$, $\sigma \in \Sigma_\omega$, and $v \in (\Sigma_\omega \cup \Sigma_T)^*$. Then $U \parallel T \overset{s}{\Rightarrow} ((x_a, a), x_T) \overset{u}{\Rightarrow} ((x_a, a), x'_T) \overset{\sigma}{\rightarrow} (y, y_T)$, i.e., $\sigma \in \text{Elig}_U((x_a, a)) = a = \text{Elig}_{\tilde{U}}((\tilde{x}_a, a))$ by definition 3.3 (iv) and (v), and $(\tilde{x}_a, a) \xrightarrow{\sigma}_{\tilde{U}} \tilde{y}$ for some $\tilde{y} \in Q/{\sim_{\text{inc}}}$. If $\sigma = \omega$, then clearly $\tilde{U} \parallel T \overset{s}{\Rightarrow} ((\tilde{x}_a, a), x_T) \overset{u}{\Rightarrow} ((\tilde{x}_a, a), x'_T) \overset{\omega}{\Rightarrow}$, which is enough to show that $\tilde{U} \parallel T$ is nonblocking. Otherwise, $\tilde{U} \parallel T \overset{s}{\Rightarrow} ((\tilde{x}_a, a), x_T) \overset{u}{\Rightarrow} ((\tilde{x}_a, a), x'_T) \overset{\sigma}{\rightarrow} (\tilde{y}, y_T)$ with $su\sigma \in (\Sigma \cup \Sigma_T)^*$ and $\tilde{y} \in Q/{\sim_{\text{inc}}}$, and the proof continues as in *Case 2*.

Case 2: $\tilde{x} \in Q/\sim_{\text{inc}}$. Then $\tilde{A} \xrightarrow{P(s)} \tilde{x}$ by lemma 3.4 (i). Then let $x \in \tilde{x}$, and it follows from lemma 3.7 (ii) that $A \xrightarrow{P(s)} x$, which implies $U \overset{P(s)}{\Longrightarrow} x$ again by lemma 3.4 (i). Thus, $U \parallel T \overset{P(s)}{\Longrightarrow} (x, x_T)$, and since $U \parallel T$ is nonblocking, there exists $w \in \Sigma^*$ such that $U \parallel T \overset{P(s)}{\Longrightarrow} (x, x_T) \overset{w\omega}{\Rightarrow} (z, z_T)$. Then $x \overset{P(w)\omega}{\Longrightarrow}_U z$, with $z \in Q$ by definition 3.3. This implies $x \xrightarrow{P(w)\omega} z$ by lemma 3.3, and thus $[x] \xrightarrow{P(w)\omega} [z]$ in $A/\sim_{\text{inc}}$ by lemma 3.6, which implies $\tilde{x} = [x] \overset{P(w)\omega}{\Longrightarrow}_{\tilde{U}} [z]$ again by lemma 3.3. Thus, $\tilde{U} \parallel T \overset{s}{\Rightarrow} (\tilde{x}, x_T) \overset{w\omega}{\Rightarrow}$, and since $(\tilde{x}, x_T)$ was chosen arbitrarily, it follows that $\tilde{U} \parallel T$ is nonblocking.

Conversely, assume that $\tilde{U} \parallel T$ is nonblocking, and let $s \in (\Sigma \cup \Sigma_T)^*$ such that $U \parallel T \overset{s}{\Rightarrow} (x, x_T)$. Then $U \overset{P(s)}{\Longrightarrow} x \in Q \cup Ann$. Consider two cases.

Case 1: $x = (x_a, a) \in Ann$. From $U \overset{P(s)}{\Longrightarrow} (x_a, a)$, by lemma 3.4 (ii) it follows that $A \xrightarrow{P(s)} x_a$, which implies $\tilde{A} \xrightarrow{P(s)} [x_a]$ by lemma 3.6. Note that $([x_a], a) \in \tilde{A}nn$ and thus $\tilde{U} \overset{P(s)}{\Longrightarrow} ([x_a], a)$ again by lemma 3.4 (ii). Thus, $\tilde{U} \parallel T \overset{s}{\Rightarrow} (([x_a], a), x_T)$, and since $\tilde{U} \parallel T$ is nonblocking, there exists $t \in \Sigma^*$ such that $\tilde{U} \parallel T \overset{s}{\Rightarrow} (([x_a], a), x_T) \overset{t\omega}{\Rightarrow}$. Write $t\omega = u\sigma v$ with $u \in (\Sigma_T \setminus \Sigma)^*$, $\sigma \in \Sigma_\omega$, and $v \in (\Sigma_\omega \cup \Sigma_T)^*$. Then $\tilde{U} \parallel T \overset{s}{\Rightarrow} (([x_a], a), x_T) \overset{u}{\Rightarrow} (([x_a], a), x'_T) \xrightarrow{\sigma} (\tilde{y}, y_T)$, i.e., $\sigma \in \text{Elig}_{\tilde{U}}(([x_a], a)) = a = \text{Elig}_U((x_a, a))$ by definition 3.3 (iv) and (v), and $(x_a, a) \xrightarrow{\sigma}_U y$ for some $y \in Q$. Therefore $U \parallel T \overset{s}{\Rightarrow} ((x_a, a), x_T) \overset{u}{\Rightarrow} ((x_a, a), x'_T) \xrightarrow{\sigma} (y, y_T)$ with $y \in Q$. If $\sigma = \omega$, then clearly $U \parallel T \overset{s}{\Rightarrow} ((x_a, a), x_T) \overset{u\omega}{\Rightarrow}$, which is enough to show that $U \parallel T$ is nonblocking. Otherwise, $U \parallel T \overset{su\sigma}{\Longrightarrow} (y, y_T)$ with $su\sigma \in (\Sigma \cup \Sigma_T)^*$ and $y \in Q$, and the proof continues as in *Case 2*.

Case 2: $x \in Q$. Then $A \xrightarrow{P(s)} x$ by lemma 3.4 (i), which implies $\tilde{A} \xrightarrow{P(s)} [x]$ by lemma 3.6. By definition 3.1, there exists $a \subseteq \text{Elig}_A(x)$ such that $(x, a) \in Ann$. Then $([x], a) \in \tilde{A}nn$, and $\tilde{U} \overset{P(s)}{\Longrightarrow} ([x], a)$ by lemma 3.4 (ii). Thus, $\tilde{U} \parallel T \overset{s}{\Rightarrow} (([x], a), x_T)$, and since $\tilde{U} \parallel T$ is nonblocking, there exists $t \in \Sigma^*$ such that $\tilde{U} \parallel T \overset{s}{\Rightarrow} (([x], a), x_T) \overset{t\omega}{\Rightarrow}$. Write $t\omega = u\sigma v$ with $u \in (\Sigma_T \setminus \Sigma)^*$, $\sigma \in \Sigma_\omega$, and $v \in (\Sigma_\omega \cup \Sigma_T)^*$. Then $\tilde{U} \parallel T \overset{s}{\Rightarrow} (([x], a), x_T) \overset{u}{\Rightarrow} (([x], a), x'_T) \xrightarrow{\sigma} (\tilde{y}, y_T)$. Clearly, $\sigma \in \text{Elig}_{\tilde{U}}(([x], a)) = a \subseteq \text{Elig}_A(x) = \text{Elig}_U(x)$ by definition 3.3 (iii) and (v). If $\sigma = \omega$, it already follows that $U \parallel T \overset{s}{\Rightarrow} (x, x_T) \overset{u\omega}{\Rightarrow}$, i.e., $U \parallel T$ is nonblocking. Otherwise $\sigma \in \text{Elig}_A(x)$ means that $x \xrightarrow{\sigma} y$ for some $y \in Q$. Then $\tilde{A} \xrightarrow{P(s)} [x] \xrightarrow{\sigma} [y]$ by definition 3.6 and $\tilde{U} \overset{P(s)\sigma}{\Longrightarrow} [y]$ by lemma 3.4 (i). Therefore $\tilde{U} \parallel T \overset{su\sigma}{\Longrightarrow} ([y], y_T)$, and since $\tilde{U} \parallel T$ is nonblocking, there exists $w \in \Sigma^*$ such that $\tilde{U} \parallel T \overset{su\sigma}{\Longrightarrow} ([y], y_T) \overset{w\omega}{\Rightarrow}$. Then $[y] \overset{P(w)\omega}{\Longrightarrow}_{\tilde{U}}$, and by lemma 3.7 (i) there exists $y' \in [y]$ such that $y' \overset{P(w)\omega}{\Longrightarrow}_U$. Thus $x \xrightarrow{\sigma} y \sim_{\text{inc}} y'$, which implies $x \xrightarrow{\sigma} y'$ by definition 3.5, and $x \xrightarrow{\sigma}_U y'$ by definition 3.3 (iii). Thus, $U \parallel T \overset{s}{\Rightarrow} (x, x_T) \overset{u\sigma}{\Rightarrow} (y', y_T) \overset{w\omega}{\Rightarrow}$, and since (x, x_T) was chosen arbitrarily, it follows that $U \parallel T$ is nonblocking. $\square$

3.1.5 Bisimulation

Bisimulation and *observation equivalence* [26] are general tools that have been used with considerable success to simplify automata during nonblocking verification [12, 33]. Bisimulation can also be applied to annotated automata, with the added restriction that bisimilar states must have the same annotations. Nevertheless, the removal of silent transitions can transform several conflict equivalent transition structures into the same annotated states, even if they are not originally observation equivalent. So bisimulation on the annotated automaton can be more effective, particularly after the removal of subsumed annotations.

Definition 3.7 Let $A_1 = \langle \Sigma, Q_1, \rightarrow_1, Q_1^\circ, Ann_1 \rangle$ and $A_2 = \langle \Sigma, Q_2, \rightarrow_2, Q_2^\circ, Ann_2 \rangle$ be two annotated automata. A relation $\approx \subseteq Q_1 \times Q_2$ is called a *bisimulation* between A_1 and A_2, if the following conditions hold for all states $x_1 \in Q_1$ and $x_2 \in Q_2$ such that $x_1 \approx x_2$.

- For all $\sigma \in \Sigma_\omega$, if $x_1 \xrightarrow{\sigma} y_1$ then there exists $y_2 \in Q_2$ such that $y_1 \approx y_2$ and $x_2 \xrightarrow{\sigma} y_2$.
- For all $\sigma \in \Sigma_\omega$, if $x_2 \xrightarrow{\sigma} y_2$ then there exists $y_1 \in Q_1$ such that $y_1 \approx y_2$ and $x_1 \xrightarrow{\sigma} y_1$.
- For all $a \subseteq \Sigma_\omega$, it holds that $(x_1, a) \in Ann_1$ if and only if $(x_2, a) \in Ann_2$.

A_1 and A_2 are *bisimulation equivalent* or *bisimilar*, written $A_1 \approx A_2$, if there exists a bisimulation $\approx$ between A_1 and A_2 such that, for every initial state $x_1^\circ \in Q_1^\circ$ there exists an initial state $x_2^\circ \in Q_2^\circ$ such that $x_1^\circ \approx x_2^\circ$, and vice versa.

It is easily confirmed that conditions (i) and (ii) in definition 3.1 are preserved under bisimilarity of annotated automata. This is because bisimilar states always have the same sets of annotations and eligible events.

Example 3.8 Automaton A'' in figure 3.1 is bisimilar to A'. States q_0, q_1, and q_{25} have been merged due to the fact that they have the same annotations and equivalent outgoing transitions. Note that this only becomes possible after annotation, subsumption, and incoming equivalence.

Proposition 3.6 Let A_1 and A_2 be annotated automata such that $A_1 \approx A_2$. Then $A_1 \simeq_{\text{conf}} A_2$.

Complexity 3.5 Given an annotated automaton, a coarsest bisimulation relation can be found in time complexity $O(|\rightarrow| \log |Q|)$ using the algorithm in [10]. The annotated form of $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ has $O(|Q|^2|\Sigma|)$ transitions, giving $O(|Q|^2|\Sigma| \log |Q|)$ time complexity for its simplification. An initial partition based on annotations can be established with lower time complexity.

We will now set out to prove proposition 3.6. This is best proven by showing that the unannotated forms of bisimilar annotated automata are bisimilar. For this purpose, the following standard definition of bisimulation for ordinary automata is used [26].

Definition 3.8 Let $G_1 = \langle \Sigma, Q_1, \rightarrow_1, Q_1^\circ \rangle$ and $G_2 = \langle \Sigma, Q_2, \rightarrow_2, Q_2^\circ \rangle$ be two automata. A relation $\approx \subseteq Q_1 \times Q_2$ is called a *bisimulation* between G_1 and G_2, if the following conditions hold for all states $x_1 \in Q_1$ and $x_2 \in Q_2$ such that $x_1 \approx x_2$.

(i) For all $\sigma \in \Sigma_{\tau,\omega}$, if $x_1 \xrightarrow{\sigma} y_1$ then there exists $y_2 \in Q_2$ such that $y_1 \approx y_2$ and $x_2 \xrightarrow{\sigma} y_2$.

(ii) For all $\sigma \in \Sigma_{\tau,\omega}$, if $x_2 \xrightarrow{\sigma} y_2$ then there exists $y_1 \in Q_1$ such that $y_1 \approx y_2$ and $x_1 \xrightarrow{\sigma} y_1$.

G_1 and G_2 are *bisimulation equivalent* or *bisimilar*, written $G_1 \approx G_2$, if there exists a bisimulation $\approx$ between G_1 and G_2 such that, for every initial state $x_1^\circ \in Q_1^\circ$ there exists an initial state $x_2^\circ \in Q_2^\circ$ such that $x_1^\circ \approx x_2^\circ$, and vice versa.

Although unannotated forms have been shown to be unique up to conflict equivalence in proposition 3.2, two unannotated forms of the same annotated automaton are not necessarily bisimilar. To prove the result about bisimulation, a unique unannotated form is needed.

Definition 3.9 Let $A = \langle \Sigma, Q, \rightarrow, Q^\circ, Ann \rangle$ be an annotated automaton. The *standard unannotation* of A is $\mathscr{U}(A) = \langle \Sigma, Q_U, \rightarrow_U, Q^\circ \rangle$ where $Q_U = Q \cup Ann$ and

$$\begin{aligned} \rightarrow_U = \rightarrow \cup \{\, (x, \tau, (x,a)) \in Q \times \{\tau\} \times Ann \,\} \cup \\ \{\, ((x,a), \sigma, y) \in Ann \times \Sigma_\omega \times Q \mid \sigma \in a \text{ and } x \xrightarrow{\sigma} y \,\} \end{aligned} \tag{3.11}$$

The standard unannotation resolves the ambiguity in points (iv) and (v) of definition 3.3 by simply including all possible transitions for every annotation state. This ensures uniqueness at the expense of minimality. It is easy to confirm that, for every annotated automaton A, the standard unannotation $\mathscr{U}(A)$ is indeed an unannotated form of A.

The standard unannotations of bisimilar automata can be shown to be bisimilar, and this is enough to complete the proof of proposition 3.6.

Lemma 3.8 Let $A_1 = \langle \Sigma, Q_1, \rightarrow_1, Q_1^\circ, Ann_1 \rangle$ and $A_2 = \langle \Sigma, Q_2, \rightarrow_2, Q_2^\circ, Ann_2 \rangle$ be two annotated automata such that $A_1 \approx A_2$. Then $\mathscr{U}(A_1) \approx \mathscr{U}(A_2)$.

Proof. Let $\mathscr{U}(A_i) = \langle \Sigma, Q_{U,i}, \rightarrow_{U,i}, Q_i^\circ \rangle$ where $Q_{U,i} = Q_i \cup Ann_i$ for $i = 1,2$, and let $\approx$ be a bisimulation between A_1 and A_2. Consider the relation $\approx_U \subseteq Q_{U,1} \times Q_{U,2}$ such that $x_1 \approx_U x_2$ if and only if one of the following two conditions holds:

$$x_1 \in Q_1,\ x_2 \in Q_2, \text{ and } x_1 \approx x_2 \qquad \text{or} \tag{3.12}$$

$$\text{there exists } a \subseteq \Sigma_\omega \text{ such that } x_1 = (x_1', a) \in Ann_1,\ x_2 = (x_2', a) \in Ann_2, \text{ and } x_1' \approx x_2'\ . \tag{3.13}$$

It is to be shown that $\approx_U$ is a bisimulation between $\mathscr{U}(A_1)$ and $\mathscr{U}(A_2)$. To see (i) in definition 3.8, let $x_1 \approx_U x_2$ and $x_1 \xrightarrow{\sigma}_{U,1} y_1$ for some $\sigma \in \Sigma_{\tau,\omega}$. Then either (3.12) or (3.13) holds.

If (3.12) holds, then $x_1 \approx x_2$ with $x_1 \in Q_1$ and $x_2 \in Q_2$. Then either $y_1 \in Q_1$ or $y_1 \in Ann_1$. If $y_1 \in Q_1$, then it follows from $x_1 \xrightarrow{\sigma}_{U,1} y_1$ that $x_1 \xrightarrow{\sigma}_1 y_1$ by definition 3.9. Since $x_1 \approx x_2$, by definition 3.7 there exists $y_2 \in Q_2$ such that $x_2 \xrightarrow{\sigma}_2 y_2$ and $y_1 \approx y_2$. Again by definition 3.9, this implies $x_2 \xrightarrow{\sigma}_{U,2} y_2$, and $y_1 \approx_U y_2$ according to (3.12). If on the other hand $y_1 \in Ann_1$, then $\sigma = \tau$ and $y_1 = (x_1, a)$ for some $a \subseteq \Sigma_\omega$ by definition 3.3. Since $x_1 \approx x_2$ and $(x_1, a) = y_1 \in Ann_1$, it follows from definition 3.7 that $(x_2, a) \in Ann_2$. Then $x_2 \xrightarrow{\tau}_{U,2} (x_2, a)$ by definition 3.9 and $y_1 = (x_1, a) \approx_U (x_2, a)$ by (3.13).

If (3.13) holds, then $x_1 = (x_1', a) \in Ann_1$ and $x_2 = (x_2', a) \in Ann_2$ for some $a \subseteq \Sigma_\omega$, and $x_1' \approx x_2'$. Then it follows from $(x_1', a) \xrightarrow{\sigma}_{U,1} y_1$ by definition 3.9 that $\sigma \in a$, $y_1 \in Q_1$, and $x_1' \xrightarrow{\sigma}_1 y_1$. Since $x_1' \approx x_2'$, there exists $y_2 \in Q_2$ such that $x_2' \xrightarrow{\sigma}_2 y_2$ and $y_1 \approx y_2$. Then $(x_2', a) \xrightarrow{\sigma}_{U,2} y_2$ by definition 3.9 since $\sigma \in a$, and $y_1 \approx_U y_2$ by (3.12) since $y_1 \approx y_2$.

This shows (i) in definition 3.8. The proof of (ii) is symmetric, and the condition on the initial states follows since $A_1 \approx A_2$ and A_i and $\mathscr{U}(A_i)$ have the same initial states.

□

Proposition 3.6 Let A_1 and A_2 be annotated automata such that $A_1 \approx A_2$. Then $A_1 \simeq_{\text{conf}} A_2$.

Proof. Let U_1 be an unannotated form of A_1, and let U_2 be an unannotated form of A_2. Then $U_1 \simeq_{\text{conf}} \mathscr{U}(A_1) \approx \mathscr{U}(A_2) \simeq_{\text{conf}} U_2$ by proposition 3.2 and lemma 3.8. The claim follows from results in [25], according to which bisimilar automata are conflict equivalent. □

3.1.6 Abstraction Procedure

This section explains how the above results can be used to minimise a given automaton with respect to conflict equivalence. Given an automaton G, the task is to compute a hopefully smaller abstraction G' conflict equivalent to G.

Given the complexity of the annotation procedure, it is advisable to reduce the size of the input automaton G using some standard means before constructing an annotated form. While not necessarily optimal for conflict equivalence, bisimulation or observation equivalence [26] can be computed efficiently and are known to achieve significant reduction, as is the removal of blocking states [12].

After simplification of the input automaton, the next step is to compute its annotated form $\mathscr{A}(G)$, which then is simplified in several steps. While constructing the annotated form, annotations can be checked for subsumption on the fly, suppressing the generation of any redundant annotations. The resulting annotated form is next simplified by merging incoming equivalent states, again checking for subsumption and removing annotations that become redundant. Then the result is minimised according to bisimulation equivalence.

After simplifying the annotated automaton, it is unannotated to obtain an ordinary automaton that is conflict equivalent to the input. There are different ways to construct an unannotated form that satisfies the conditions of definition 3.3, as there is considerable leeway in how outgoing transitions from annotation states can be chosen, and by making clever choices, the new annotation states can become bisimilar to original states or other annotation states, making it possible to further simplify the result.

An example of the abstraction procedure is shown in figure 3.1. Automaton G is first annotated to obtain $\mathscr{A}(G)$, with subsumption being tested on the fly to suppress some annotations struck out in the figure. Next incoming equivalence leads to the abstraction A', with another annotation being suppressed due to subsumption as discussed in example 3.7, and the result is further simplified using bisimulation, giving A''.

Since the annotated automaton cannot be simplified further, it is replaced by its unannotated form U. As explained in example 3.5, the transition $(q_{0125},\{\alpha\}) \xrightarrow{\alpha} q_{0125}$ is not included in U. This choice makes the states q_8, $(q_8,\{\alpha\})$, and $(q_{0125},\{\alpha\})$ observation equivalent, so they can be merged in addition to states q_7 and $(q_7,\{\omega\})$. This results in the observation equivalent abstraction U'. Furthermore, the transition $q_{0125} \xrightarrow{\alpha} q_8$ is redundant according to observation equivalence [8] and can be removed, giving the final result U''.

The abstraction steps in figure 3.1 can be justified by the propositions given in the previous sections. Note that, for every annotated automaton, there exists an unannotated form although it does not always have to be constructed explicitly. Let V and V' be unannotated forms of $\mathscr{A}(G)$ and A', respectively. Then $G \simeq_{\text{conf}} V$ by proposition 3.3 and $V \simeq_{\text{conf}} V' \simeq_{\text{conf}} U$ by proposition 3.4–3.6. Furthermore, U is observation equivalent to U' and U'', which implies $U \simeq_{\text{conf}} U''$ according to [25]. Thus,

$$G \simeq_{\text{conf}} V \simeq_{\text{conf}} V' \simeq_{\text{conf}} U \simeq_{\text{conf}} U' \simeq_{\text{conf}} U'' \,. \tag{3.14}$$

Overall, the automaton G with nine states and 25 transitions is simplified to the conflict-equivalent automaton U'' with three states and seven transitions.

3.2 Experimental Results

A conflict checker using annotated automata has been implemented in the DES software tool Supremica [1] and tested on the same set of industrial-scale and parametrised models as used previously in [12]. All these problems have been solved successfully, and the results are shown in Table 5.1.

After simplifying each individual component in a composed system, the algorithm selects a *candidate* set of automata for composition using strategies described in [12]. After synchronous composition and hiding of local events, the result is first simplified using observation equivalence and by removing obvious certain conflicts [12]. Then the annotated form is constructed and simplified using incoming equivalence and bisimulation. Subsumption is used during each of these steps. Finally, an unannotated form is obtained and further simplified by removing states with only silent outgoing transitions.

The *Annotating Method* described above has been compared to the *Heuristic Method* described in [12]. The heuristic compositional conflict checker of [12] selects and composes candidate sets of automata in the same way as the annotating method, but it uses a more straightforward set of abstraction rules to simplify automata. In addition to the *Certain Conflicts Rule* and observation equivalence, which are part of the preprocessing steps in the Annotating Method, the Heuristic Method also uses the *Active Events Rule*, the *Silent Continuation Rule*, the *Only Silent Incoming Rule*, and the *Only Silent Outgoing Rule* [12]. All these rules are directly applied to the transitions of an automaton, without computing an annotated form. This makes the rules simpler to apply, but they also have somewhat weaker abstraction potential, as it can be shown that all abstractions obtained using the above mentioned rules and more can in principle be achieved by simplifying an annotated automaton.

To make the Annotating and Heuristic Method comparable, they have been modified to ensure that both implementations select and compose the same automata in the same order, regardless of possible differences in the intermediate results. This is done to compare the effects of the different simplification methods, as opposed to comparing different choices of automata for composition (which often lead to dramatic changes). However, the chosen order of composition is no longer optimal, which explains the difference between the results in Table 3.1 and [12].

Table 3.1 shows the experimental results for nonblocking verification of 14 large models of industrial-scale applications and 9 very large parametrised models. Please

Table 3.1: Experimental results

		Annotating			**Heuristic**		
	Size	Peak States	Total States	Time [s]	Peak States	Total States	Time [s]
AGV	$2.6 \cdot 10^{7}$	10552	18054	28.1	1368	4097	4.1
AGVb	$2.3 \cdot 10^{7}$	975	1719	0.2	781	1524	0.1
verriegel3	$9.7 \cdot 10^{8}$	2346	12767	4.7	2856	14639	6.8
verriegel3b	$1.3 \cdot 10^{9}$	2346	11028	4.8	2537	11976	6.3
verriegel4	$4.5 \cdot 10^{10}$	3703	15286	5.4	2671	15106	6.1
verriegel4b	$6.3 \cdot 10^{10}$	2346	11827	4.6	2537	12968	6.3
big_bmw	$3.1 \cdot 10^{7}$	63	342	0.1	63	347	0.1
FMS	812544	86	206	0.0	125	279	0.1
SMS	312	18	119	0.0	18	120	0.0
PMS	$5.7 \cdot 10^{8}$	75	487	0.1	75	492	0.2
IPC	20592	107	195	0.0	107	195	0.1
ftechnik	$1.2 \cdot 10^{8}$	5631	21218	5.9	2450	15524	4.8
rhone_tough	$1.0 \cdot 10^{10}$	1584	5025	4.1	1584	5026	4.5
AIP	$1.0 \cdot 10^{9}$	6864	82542	30.3	6868	77512	24.7
256philo	$5.4 \cdot 10^{168}$	628	77419	21.8			
512philo	$2.9 \cdot 10^{337}$	628	156395	48.1			
1024philo	$8.5 \cdot 10^{674}$	628	314347	96.1			
128transfer	$1.6 \cdot 10^{231}$	43	11115	3.9	42	10966	10.7
256transfer	$2.4 \cdot 10^{462}$	43	22251	10.7	42	21974	9.3
512transfer	$5.8 \cdot 10^{924}$	43	44523	42.6	42	43990	34.7
128arbiter	$2.8 \cdot 10^{112}$	55	14669	10.4			
256arbiter	$5.4 \cdot 10^{224}$	55	29517	31.5			
512arbiter	$2.1 \cdot 10^{449}$	55	59213	58.1			

refer to [12] for a more detailed description of the models. The table shows the number of reachable states of the synchronous product of each model (Size), and the number of states of the largest automaton encountered during compositional verification (Peak States), the cumulative number of states constructed during verification (Total States), and the total verification time in seconds, for both the Annotating Method and the Heuristic Method,

All experiments were run on a standard laptop computer with a 2 GHz microprocessor and 4 GB of RAM, and controlled by state limits. If during abstraction some synchronous product has more than 10,000 states, its construction is aborted and another set of automata is composed instead. If no suitable set of automata for composition can be identified, a final attempt is made to construct and check the full synchronous product of all remaining automata whether it is nonblocking. If this attempt runs out of memory, the run is aborted and the corresponding table entries are left blank.

The annotating conflict checker performs much better than the heuristic method for the parametrised dining philosophers and tree arbiter problems, which cannot be solved by the heuristic method using the given state limits and candidate selection strategy. For the industrial applications, the two methods yield similar results, with the Annotating Method producing a smaller peak number of states in 5 cases, and the Heuristic Method producing a smaller peak number of states in 4 cases. The difference is particularly notable for the *AGV* and *ftechnik* models, where the annotating method results in larger automata. This seems to be caused by the annotating and unannotating steps, which may change the structure of an automaton in such a way that certain states are no longer observation equivalent.

Table 3.2 shows some information on the effectiveness of the individual steps taken by the annotating method. First, it shows for each model the total number of annotations created and removed by subsumption. Next, it shows the total number of states removed as unreachable after annotation (Ann.), the number of states removed by merging incoming equivalent ($\sim_{\text{inc}}$) and bisimilar ($\approx$) states, and the number of states added back in when constructing unannotated forms (Unann.). Note that $\approx$ refers to simplification of annotated automata and is in addition to observation equivalence simplification, which is performed on all automata before annotating.

In most cases, annotating helps to remove substantially more states than need to be added back during unannotation. The data clearly shows the importance of the subsumption step, which is performed directly while constructing the annotated form. While merging incoming equivalent and bisimilar states seems to have a limited effect for most industrial models, it has a marked effect for some of the more regular models in the dining philosophers and arbiter series.

Theses results show that conflict equivalence preserving abstractions can be used

Table 3.2: Rule Usage

	Annotations		**States**			
	Create	Subsume	Ann.	$\sim_{\text{inc}}$	$\approx$	Unann.
AGV	+63435	−58073	−1777	−34	−513	+5
AGVb	+328	−226	−0	−0	−0	+0
verriegel3	+3442	−759	−93	−7	−16	+37
verriegel3b	+3478	−777	−70	−1	−16	+19
verriegel4	+3875	−927	−93	−13	−32	+29
verriegel4b	+4578	−1540	−122	−1	−67	+42
big_bmw	+53	−27	−1	−0	−0	+1
FMS	+77	−26	−24	−0	−8	+11
SMS	+8	−8	−0	−0	−0	+0
PMS	+161	−103	−17	−9	−9	+7
IPC	+133	−58	−9	−0	−2	+4
ftechnik	+4785	−856	−26	−0	−0	+1
rhone_tough	+899	−491	−15	−0	−6	+13
AIP	+17303	−6644	−1600	−597	−216	+1054
256philo	+86128	−33106	−1756	−874	−9635	+0
512philo	+174192	−67133	−3548	−1770	−19491	+0
1024philo	+350320	−133683	−7132	−3562	−39203	+0
128transfer	+3721	−1289	−129	−0	−0	+1
256transfer	+7433	−2569	−257	−0	−0	+1
512transfer	+14857	−5129	−513	−0	−0	+1
128arbiter	+5475	−2769	−1002	−436	−61	+61
256arbiter	+11043	−5585	−2026	−884	−125	+125
512arbiter	+22179	−11217	−4074	−1780	−253	+253

to verify whether or not large systems are nonblocking. They further show that different abstraction methods can be superior to one another in different situations, and thus that it beneficial to have a multitude of abstraction methods available for different models. This chapter has introduced the method of abstracting automata using annotated automata. It further introduces several abstraction rules which can be used on an annotated automata in order to simplify the automaton while preserving conflict equivalence.

Chapter 4

Generalised Nonblocking

Despite its widespread use, the expressive powers of nonblocking are limited. To overcome its weaknesses, nonblocking has been modified and extended in several ways [9, 23, 29].

This chapter is concerned about *generalised nonblocking* [23], which adds to standard nonblocking the ability to restrict the set of states from which blocking is checked. This is useful for the verification of software components and of certain conditions in Hierarchical Interface-Based Supervisory Control [19, 20]. Of particular interest for the purposes of this chapter is how nonconflicting completions relate to generalised nonblocking equivalence.

Comparing two automata with respect to generalised nonblocking equivalence is in many ways similiar to comparing two automata with respect to standard nonblocking equivalence, but with a simplified semantics. This makes it a much easier equivalence relation to characterize, while still providing us insight into conflict-equivalence. In addition to this, all standard nonblocking problems can also be represented as generalised nonblocking problems. Thus, all methods of simplifying an automata with respect to generalised nonblocking can be potentially be applied to standard nonblocking.

This chapter is organised as follows. Section 4.1 introduces multi-coloured automata. Section 4.3 introduces a testing equivalence and preorder for generalised nonblocking, presents a semantic model, and proves results about its adequacy and finiteness. Afterwards, section 4.4 describes the canonical automaton as a standardised normal form with respect to generalised nonblocking, and proposes an algorithm to construct it.

4.1 Multi-coloured Automata

Because generalised nonblocking uses the proposition α to mark states which must terminate, the definition of automata which we introduced in section 2.2, and is used throughout the rest of this thesis, is not well suited to dealing with generalised nonblocking. This section gives a definition of multi-coloured automata and describes how the major automata operations behave with respect to them. *Multi-coloured* automata extend the traditional concept of *marked states* to multiple simultaneous marking conditions, by labelling states with different *colours* or *propositions*. In most other respect multi-coloured automata are identical to regular automata. The generalised nonblocking property [23] is defined using these propositions. The following definition is introduced in [23] and based on similar ideas in [5, 29].

Definition 4.1 A *multi-coloured automaton* is a tuple $G = \langle \Sigma, \Pi, Q, \rightarrow, Q^\circ, \Xi \rangle$ where Σ is a finite set of *events*, Π is a finite set of *propositions* or *colours*, Q is a set of *states*, $\rightarrow \subseteq Q \times \Sigma_\tau \times Q$ is the *state transition relation*, $Q^\circ \subseteq Q$ is the set of *initial states*, and $\Xi \colon \Pi \rightarrow 2^Q$ defines the set of marked states for each proposition in Π. G is called *finite-state* if the state set Q is finite.

Multi-coloured automata behave identically to the automata introduced in section 2.2 in most respects. The main difference is how π-marked languages are defined. For $\pi \in \Pi$, the *π-marked language* $\mathscr{L}^\pi(x) = \{ s \in \Sigma^* \mid x \stackrel{s}{\Rightarrow} \Xi(\pi) \}$ contains the traces that lead from x to some state marked π. The language and the π-marked language of an automaton G are $\mathscr{L}(G) = \mathscr{L}(Q^\circ)$ and $\mathscr{L}^\pi(G) = \mathscr{L}^\pi(Q^\circ)$.

Synchronous composition models the parallel execution of two or more automata, and is done using lock-step synchronisation [16]. This is the same operation which was introduced in section 2.3 except the Ξ is also synchronised.

Definition 4.2 Let $G = \langle \Sigma, \Pi, Q_G, \rightarrow_G, Q_G^\circ, \Xi_G \rangle$ and $H = \langle \Sigma, \Pi, Q_H, \rightarrow_H, Q_H^\circ, \Xi_H \rangle$ be multi-coloured automata. The *synchronous product* of G and H is

$$G \,\|\, H = \langle \Sigma, \Pi, Q_G \times Q_H, \rightarrow, Q_G^\circ \times Q_H^\circ, \Xi \rangle \tag{4.1}$$

where

$$\begin{array}{ll} (x_G, x_H) \xrightarrow{\sigma} (y_G, y_H) & \text{if } \sigma \in \Sigma,\ x_G \xrightarrow{\sigma}_G y_G, \text{ and } x_H \xrightarrow{\sigma}_H y_H; \\ (x_G, x_H) \xrightarrow{\tau} (y_G, x_H) & \text{if } x_G \xrightarrow{\tau}_G y_G; \\ (x_G, x_H) \xrightarrow{\tau} (x_G, y_H) & \text{if } x_H \xrightarrow{\tau}_H y_H; \end{array}$$

and $\Xi(\pi) = \Xi_G(\pi) \times \Xi_H(\pi)$ for each $\pi \in \Pi$.

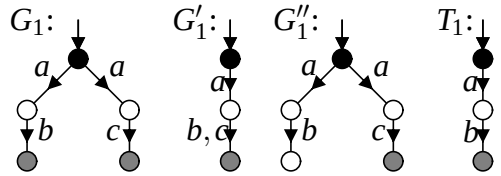


Figure 4.1: Generalised nonblocking equivalence.

4.2 Generalised Nonblocking

Nonblocking is generalised in [23], using two propositions α and ω. The intended meaning is that ω represents terminal states, while α specifies a set of states from which terminal states are required to be reachable. This is in contrast to standard nonblocking where terminal states must be reachable from all reachable states.

Definition 4.3 Let $G = \langle \Sigma, \Pi, Q, \rightarrow, Q^\circ, \Xi \rangle$ be a multi-coloured automaton with $\alpha, \omega \in \Pi$.

- G is *ω-nonblocking* or *standard nonblocking*, if for all states $x \in Q$ such that $G \Rightarrow x$ it also holds that $x \Rightarrow \Xi(\omega)$. Otherwise, G is *ω-blocking*.

- G is *(α, ω)-nonblocking*, or *generalised nonblocking* if for all states $x \in \Xi(\alpha)$ such that $G \Rightarrow x$ it also holds that $x \Rightarrow \Xi(\omega)$. Otherwise, G is *(α, ω)-blocking*.

4.3 Generalised Nonblocking Equivalence

In the same way that the nonblocking property of a system can be verified by abstracting components with respect to conflict equivalence the generalised nonblocking property can be verified by abstracting with respect to generalised nonblocking equivalence.

For example, automaton G_1 in figure 4.1 may be replaced by G_1' while preserving the generalised nonblocking property of the system $G_1 \parallel G_2 \parallel \cdots \parallel G_n$. If the remainder $G_2 \parallel \cdots \parallel G_n$ of the system has an α-marked initial state, the composed system is (α, ω)-nonblocking if and only if it can reach an ω-marked state after executing the trace *ab* or *ac*, regardless of whether G_1 or G_1' is used.

On the other hand, generalised nonblocking is not preserved if G_1 is replaced by G_1'' in figure 4.1. If $G_2 \parallel \cdots \parallel G_n$ has an α-marked initial state and can only reach an ω-marked state after executing the trace *ab*, like automaton T_1 in figure 4.1, then $G_1 \parallel G_2 \parallel \cdots \parallel G_n$ is (α, ω)-nonblocking while $G_1'' \parallel G_2 \parallel \cdots \parallel G_n$ is (α, ω)-blocking. This is the same as conflict equivalence introduced in section 2.4 with the exception that generalized nonblocking is used instead of nonblocking.

4.3.1 The Generalised Nonblocking Preorder

A notion of process *equivalence* to perform abstractions preserving generalised nonblocking is described in [23]. This section generalises these definitions and introduces a *preorder*, which makes it possible to reason not only about equivalence but also about *refinement*. The definitions are based on the traditional testing framework [6, 15] that defines preorders and equivalences relating processes based on their responses to *tests*. In the context of generalised nonblocking, a test can be an arbitrary automaton, and the test's response is the observation whether the test is (α,ω)-nonblocking in combination with the given automaton or not. Two automata are considered as equivalent, if the responses of all tests are equal.

Definition 4.4 Let G and H be two multi-coloured automata with $\alpha,\omega \in \Pi$.

- G is *less* (α,ω)*-conflicting* than H, written $G \lesssim_{(\alpha,\omega)} H$, if for every multi-coloured automaton T such that $H \parallel T$ is (α,ω)-nonblocking, $G \parallel T$ also is (α,ω)-nonblocking.
- G and H are (α,ω)*-conflict equivalent*, written $G \simeq_{(\alpha,\omega)} H$, if $G \lesssim_{(\alpha,\omega)} H$ and $H \lesssim_{(\alpha,\omega)} G$.

The relation $\lesssim_{(\alpha,\omega)}$ defines the *generalised nonblocking* preorder. An automaton G is less (α,ω)-conflicting than H if there are fewer tests T that are (α,ω)-blocking in combination with G than in combination with H. Two automata are (α,ω)-conflict equivalent if they are (α,ω)-blocking in combination with exactly the same tests. Given the composition $G_1 \parallel G_2 \parallel \cdots \parallel G_n$, if $G_1 \simeq_{(\alpha,\omega)} G_1'$, then G_1 can be replaced by G_1' without affecting the generalised nonblocking property of the composition.

Example 4.1 figure 4.1 shows four multi-coloured automata. α-marked states are black whereas ω-marked states are gray. States which have no marking associated with them have no colouring. Automata G_1 and G_1' in figure 4.1 are (α,ω)-conflict equivalent, while G_1 and G_1'' are not, because $G_1 \parallel T_1$ is (α,ω)-nonblocking and $G_1'' \parallel T_1$ is (α,ω)-blocking. Furthermore, it can be shown that $G_1 \lesssim_{(\alpha,\omega)} G_1''$.

4.3.2 Congruence Properties

An important question concerning preorders such as $\lesssim_{(\alpha,\omega)}$ is their relationship to process-algebraic operations. For compositional verification, the equivalence used must be well-behaved with respect to synchronous composition and hiding. These so-called *congruence* properties have been established in [25] for standard nonblocking and in [23] for generalised nonblocking equivalence, and can easily be extended to the generalised nonblocking preorder.

Definition 4.5 Let $\lesssim$ be a preorder on the set of multi-coloured automata.

- $\lesssim$ is a *pre-congruence* with respect to $\|$ if, for all multi-coloured automata G, H, and T such that $G \lesssim H$, it follows that $G \| T \lesssim H \| T$.
- $\lesssim$ *respects* (α, ω)*-nonblocking* if, for all multi-coloured automata G and H such that $G \lesssim H$, if H is (α, ω)-nonblocking then G also is (α, ω)-nonblocking.

Proposition 4.1 $\lesssim_{(\alpha,\omega)}$ is a pre-congruence with respect to $\|$.

Proof. Let G, H, and T be such that $G \lesssim_{(\alpha,\omega)} H$, and let T' be an arbitrary multi-coloured automaton such that $(H \| T) \| T'$ is (α, ω)-nonblocking. Then clearly, $H \| (T \| T') = (H \| T) \| T'$ is (α, ω)-nonblocking, and since $G \lesssim_{(\alpha,\omega)} H$ it follows that $(G \| T) \| T' = G \| (T \| T')$ is (α, ω)-nonblocking. Since T' was chosen arbitrarily, it follows that $G \| T \lesssim_{(\alpha,\omega)} H \| T$. □

Proposition 4.2 $\lesssim_{(\alpha,\omega)}$ respects (α, ω)-nonblocking.

Proof. Note that there exists a multi-coloured automaton U such that $G \| U = G$ for every multi-coloured automaton G. Let $G \lesssim_{(\alpha,\omega)} H$, and let H be (α, ω)-nonblocking. Then $H \| U = H$ is (α, ω)-nonblocking. Since $G \lesssim_{(\alpha,\omega)} H$, it follows that $G = G \| U$ is (α, ω)-nonblocking. □

Thus, the generalised nonblocking equivalence is a congruence with respect to synchronous composition and respects (α, ω)-nonblocking. This is enough to justify the correctness of a compositional verification approach such as the one outlined at the beginning of section 4.3.

Similarly to standard nonblocking [25], the generalised nonblocking preorder turns out to be the coarsest pre-congruence with respect to synchronous composition that respects (α, ω)-nonblocking. In other words, any preorder that relates multi-coloured automata according to their generalised nonblocking behaviour and preserves synchronous composition is contained in the generalised nonblocking preorder. Therefore, the generalised nonblocking preorder is the best possible process refinement for reasoning about generalised nonblocking.

Proposition 4.3 Let $\lesssim$ be a pre-congruence with respect to $\|$ which respects (α, ω)-nonblocking. Then $G \lesssim H$ implies $G \lesssim_{(\alpha,\omega)} H$.

Proof. Let $G \lesssim H$, and let T be a multi-coloured automaton such that $H \| T$ is (α, ω)-nonblocking. Then $G \| T \lesssim H \| T$ since $\lesssim$ is a pre-congruence with respect to $\|$. Since $\lesssim$ respects blocking it follows that $G \| T$ is (α, ω)-nonblocking. Since G, H, and T were chosen arbitrarily, it follows that $G \lesssim_{(\alpha,\omega)} H$. □

4.3.3 Characterising the Preorder

In addition to the test-based definition of a process preorder, it is desirable to have a characterisation that can be derived from the state structure of an automaton [13]. This section introduces the generalised nonconflicting completion semantics as an algebraic model of the generalised nonblocking preorder and equivalence, which can be derived from the state and transitions of a multi-coloured automaton in such a way that the model can be represented finitely for every finite-state automaton. This model will be used in the following section to construct a canonical automaton.

The following definition restates the generalised nonblocking preorder as a state-based criterion. To check whether an automaton G is less (α,ω)-conflicting than another automaton H, it is enough to collect the ω-marked languages of all α-marked states of G and check whether H contains larger languages associated with the same α-markings. This idea is formalised by the concept of being state-wise less (α,ω)-conflicting, which turns out to be equivalent to the generalised nonblocking preorder.

Definition 4.6 Let $G = \langle \Sigma, \Pi, Q_G, \rightarrow_G, Q_G^\circ, \Xi_G \rangle$ and $H = \langle \Sigma, \Pi, Q_H, \rightarrow_H, Q_H^\circ, \Xi_H \rangle$ be multi-coloured automata with $\alpha, \omega \in \Pi$. G is said to be *state-wise less (α,ω)-conflicting* than H if the following property holds for every $s \in \Sigma^*$: for every $x_G \in \Xi_G(\alpha)$ such that $G \stackrel{s}{\Rightarrow} x_G$ there exists $x_H \in \Xi_H(\alpha)$ such that $H \stackrel{s}{\Rightarrow} x_H$ and $\mathscr{L}^\omega(x_H) \subseteq \mathscr{L}^\omega(x_G)$.

Proposition 4.4 Let $G = \langle \Sigma, \Pi, Q_G, \rightarrow_G, Q_G^\circ, \Xi_G \rangle$ and $H = \langle \Sigma, \Pi, Q_H, \rightarrow_H, Q_H^\circ, \Xi_H \rangle$ be multi-coloured automata with $\alpha, \omega \in \Pi$. G is state-wise less (α,ω)-conflicting than H if and only if G is less (α,ω)-conflicting than H.

Proof. First assume that G is state-wise less (α,ω)-conflicting than H, and let $T = \langle \Sigma, \Pi, Q_T, \rightarrow_T, Q_T^\circ, \Xi_T \rangle$ be an automaton such that $H \parallel T$ is (α,ω)-nonblocking. Let $G \parallel T \stackrel{s}{\Rightarrow} (x_G, x_T) \in \Xi_G(\alpha) \times \Xi_T(\alpha)$. Clearly $G \stackrel{s}{\Rightarrow} x_G \in \Xi_G(\alpha)$, and since G is state-wise less (α,ω)-conflicting than H, there exists a state $x_H \in \Xi_H(\alpha)$ such that $H \stackrel{s}{\Rightarrow} x_H$ and $\mathscr{L}^\omega(x_H) \subseteq \mathscr{L}^\omega(x_G)$. Thus, $H \parallel T \stackrel{s}{\Rightarrow} (x_H, x_T) \in \Xi_H(\alpha) \times \Xi_T(\alpha)$, and since $H \parallel T$ is (α,ω)-nonblocking, there exists a trace $t \in \Sigma^*$ such that $(x_H, x_T) \stackrel{t}{\Rightarrow} \Xi_H(\omega) \times \Xi_T(\omega)$. Then, $t \in \mathscr{L}^\omega(x_H) \subseteq \mathscr{L}^\omega(x_G)$, which implies $x_G \stackrel{t}{\Rightarrow}_G \Xi_G(\omega)$, and therefore $(x_G, x_T) \stackrel{t}{\Rightarrow} \Xi_G(\omega) \times \Xi_T(\omega)$. Since s, x_G, and x_T were chosen arbitrarily, it follows that $G \parallel T$ is (α,ω)-nonblocking.

Second, assume that G is less (α,ω)-conflicting than H. Let $s \in \Sigma^*$ and $G \stackrel{s}{\Rightarrow} x_G \in \Xi_G(\alpha)$. Construct a deterministic automaton $T = \langle \Sigma, \Pi, Q_T, \rightarrow_T, Q_T^\circ, \Xi_T \rangle$ such that $\mathscr{L}(T) = \Sigma^*$, $\mathscr{L}^\alpha(T) = \{s\}$, and $\mathscr{L}^\omega(T) = \Sigma^* \setminus s\mathscr{L}^\omega(x_G)$. Since T is deterministic, there exists a unique state $x_T \in Q_T$ such that $T \stackrel{s}{\Rightarrow} x_T$, which satisfies $x_T \in \Xi_T(\alpha)$ and $\mathscr{L}^\omega(x_T) = \Sigma^* \setminus \mathscr{L}^\omega(x_G)$. Then $G \parallel T$ is (α,ω)-blocking, because $G \parallel T \stackrel{s}{\Rightarrow} (x_G, x_T) \in \Xi_G(\alpha) \times \Xi_T(\alpha)$ and $\mathscr{L}^\omega(x_G) \cap \mathscr{L}^\omega(x_T) = \emptyset$. Since G is less (α,ω)-conflicting than H,

it follows that $H \parallel T$ is (α,ω)-blocking. This means that there exists $u \in \Sigma^*$, $y_H \in Q_H$, and $y_T \in Q_T$ such that $H \parallel T \stackrel{u}{\Rightarrow} (y_H, y_T) \in \Xi_H(\alpha) \times \Xi_T(\alpha)$ and $\mathscr{L}^\omega(y_H) \cap \mathscr{L}^\omega(y_T) = \emptyset$. Then $y_T \in \Xi_T(\alpha)$, and by construction of T it follows that $u = s$ and $y_T = x_T$. This implies $H \stackrel{s}{\Rightarrow} y_H \in \Xi_H(\alpha)$ and $\mathscr{L}^\omega(y_H) \cap (\Sigma^* \setminus \mathscr{L}^\omega(x_G)) = \mathscr{L}^\omega(y_H) \cap \mathscr{L}^\omega(x_T) = \mathscr{L}^\omega(y_H) \cap \mathscr{L}^\omega(y_T) = \emptyset$, i.e., $\mathscr{L}^\omega(y_H) \subseteq \mathscr{L}^\omega(x_G)$. Thus, y_H satisfies the requirements given for x_H in definition 4.6, so G is state-wise less (α,ω)-conflicting than H $\square$

Proposition 4.4 is the key to constructing a process-algebraic model of generalised nonblocking. Essentially, generalised nonblocking can be characterised by the sets of ω-marked languages associated with the α-marked states or, more precisely, with the traces leading to α-marked states.

Definition 4.7 Let $G = \langle \Sigma, \Pi, Q, \rightarrow, Q^\circ, \Xi \rangle$ be a multi-coloured automaton with $\alpha, \omega \in \Pi$. The *generalised nonconflicting completion semantics* for G is defined as

$$\begin{aligned} \mathrm{CC}^\omega_{(\alpha,\omega)}(G) \;=\; \{\, (c,C) \in \Sigma^* \times 2^{\Sigma^*} \mid \text{There exists } x \in \Xi(\alpha) \text{ such that } G \stackrel{c}{\Rightarrow} x \text{ and} \\ \mathscr{L}^\omega(x) \subseteq C \,\} \,. \end{aligned} \tag{4.2}$$

If $(c,C) \in \mathrm{CC}^\omega_{(\alpha,\omega)}(G)$, then C is called a *nonconflicting completion* for c in G.

Assume G contains an α-marked state x reachable via trace $c \in \Sigma^*$, i.e., $G \stackrel{c}{\Rightarrow} x \in \Xi(\alpha)$. Then the marked language $\mathscr{L}^\omega(x)$ of x clearly is a nonconflicting completion for c in G, i.e.,

$$(c, \mathscr{L}^\omega(x)) \in \mathrm{CC}^\omega_{(\alpha,\omega)}(G) \,. \tag{4.3}$$

Furthermore, all superlanguages of $\mathscr{L}^\omega(x)$ are also nonconflicting completions,

$$(c,C) \in \mathrm{CC}^\omega_{(\alpha,\omega)}(G) \quad \text{for all } C \supseteq \mathscr{L}^\omega(x) \,. \tag{4.4}$$

If G is finite-state, then there exists only a finite number of α-states x and thus only a finite number of associated ω-marked languages $\mathscr{L}^\omega(x)$. This means that all nonconflicting completions can be obtained as supersets of the ω-marked language of some state x, of which there are only finitely many. Therefore, the following closure operations are used.

Definition 4.8 For $\mathrm{CC} \subseteq \Sigma^* \times 2^{\Sigma^*}$, the *upward closure* $\mathrm{CC}^\uparrow$ and the *reduced form* $\mathrm{CC}^\downarrow$ are

$$\mathrm{CC}^\uparrow = \{\, (c,C') \in \Sigma^* \times 2^{\Sigma^*} \mid \text{There exists } (c,C) \in \mathrm{CC} \text{ such that } C \subseteq C' \,\} \,; \tag{4.5}$$

$$\mathrm{CC}^\downarrow = \{(c,C) \in \mathrm{CC} \mid \text{For all } (c,C') \in \mathrm{CC} \text{ where } C' \subseteq C \text{ it holds that } C' = C\}. \tag{4.6}$$

Example 4.2 The generalised nonconflicting completion semantics of automaton G_1 in figure 4.1 is

$$\mathrm{CC}^{\omega}_{(\alpha,\omega)}(G_1) = \{(\varepsilon, \{ab, ac\})\}^{\uparrow} . \tag{4.7}$$

Example 4.3 The generalised nonconflicting completion semantics of automaton G_4 in figure 4.2 is

$$\mathrm{CC}^{\omega}_{(\alpha,\omega)}(G_4) = \{\, (a^n, a^+b) \mid n \geq 0 \,\}^{\uparrow} . \tag{4.8}$$

The ω-marked language of the α-marked state q_0 is $\mathscr{L}^{\omega}(q_0) = a^+b$, and since this state can be reached after any number of a events, this language is associated with all traces a^n for $n \geq 0$. The ω-marked language of the second α-marked state q_1 is $\mathscr{L}^{\omega}(q_1) = a^*b \supseteq \mathscr{L}^{\omega}(q_0)$, and as a superlanguage of the already listed language, it is automatically included in the upward closure.

Not every nonconflicting completion semantics CC can be reconstructed from its reduced form $\mathrm{CC}^{\downarrow}$. In infinite structures, it is not guaranteed for $(c,C) \in \mathrm{CC}$ that there exists a minimal subset $C' \subseteq C$ such that $(c,C') \in \mathrm{CC}$. However, if the set of nonconflicting completions C that appear in CC is finite, then the existence of minimal subsets is guaranteed. Thus, if G is a finite-state automaton, then it indeed holds that

$$\mathrm{CC}^{\omega}_{(\alpha,\omega)}(G)^{\downarrow\uparrow} = \mathrm{CC}^{\omega}_{(\alpha,\omega)}(G) . \tag{4.9}$$

The following main result of this section states that the generalised nonconflicting completion semantics indeed characterises the generalised nonblocking preorder. If an automaton G is less (α,ω)-conflicting than automaton H, then the generalised nonconflicting completion semantics of G is contained in that of H.

Proposition 4.5 Let $G = \langle \Sigma, \Pi, Q_G, \rightarrow_G, Q^{\circ}_G, \Xi_G \rangle$ and $H = \langle \Sigma, \Pi, Q_H, \rightarrow_H, Q^{\circ}_H, \Xi_H \rangle$ be multi-coloured automata with $\alpha, \omega \in \Pi$. Then $G \lesssim_{(\alpha,\omega)} H$ if and only if $\mathrm{CC}^{\omega}_{(\alpha,\omega)}(G) \subseteq \mathrm{CC}^{\omega}_{(\alpha,\omega)}(H)$.

Proof. First let $G \lesssim_{(\alpha,\omega)} H$ and $(c,C) \in \mathrm{CC}^{\omega}_{(\alpha,\omega)}(G)$. Then there exists $x_G \in \Xi_G(\alpha)$ such that $G \stackrel{c}{\Rightarrow} x_G$ and $\mathscr{L}^{\omega}(x_G) \subseteq C$. By proposition 4.4, G is state-wise less (α,ω)-conflicting than H, so there exists $x_H \in \Xi_H(\alpha)$ such that $H \stackrel{c}{\Rightarrow} x_H$ and $\mathscr{L}^{\omega}(x_H) \subseteq \mathscr{L}^{\omega}(x_G) \subseteq C$. This already implies $(c,C) \in \mathrm{CC}^{\omega}_{(\alpha,\omega)}(H)$.

Second let $\mathrm{CC}^{\omega}_{(\alpha,\omega)}(G) \subseteq \mathrm{CC}^{\omega}_{(\alpha,\omega)}(H)$. By proposition 4.4, it is sufficient to show that G is state-wise less (α,ω)-conflicting than H. Therefore, let $s \in \Sigma^*$ and $x_G \in \Xi_G(\alpha)$ such that $G \stackrel{s}{\Rightarrow} x_G$. Then $(s, \mathscr{L}^{\omega}(x_G)) \in \mathrm{CC}^{\omega}_{(\alpha,\omega)}(G) \subseteq \mathrm{CC}^{\omega}_{(\alpha,\omega)}(H)$. By definition of $\mathrm{CC}^{\omega}_{(\alpha,\omega)}(H)$, there exists $x_H \in \Xi_H(\alpha)$ such that $H \stackrel{s}{\Rightarrow} x_H$ and $\mathscr{L}^{\omega}(x_H) \subseteq \mathscr{L}^{\omega}(x_G)$. Thus, x_H satisfies the conditions of definition 4.6, so G is state-wise less (α,ω)-conflicting than H. □

4.3.4 Relationship to Standard Nonblocking

The nonconflicting completion semantics introduced in definition 2.7 can also be applied to multi-coloured automata.

Definition 4.9 [25] Let $G = \langle \Sigma, \Pi, Q, \rightarrow, Q^\circ, \Xi \rangle$ be a multi-coloured automaton with $\omega \in \Pi$. The *nonconflicting completion semantics* of G is

$$\begin{aligned} \mathrm{CC}^{\omega}(G) = \{\, (c,C) \in \Sigma^* \times 2^{\Sigma^*} \mid & \text{For every automaton } T \text{ such that } G \,\|\, T \text{ is } \omega\text{-} \\ & \text{nonblocking and } T \overset{c}{\Rightarrow} x, \text{ there exists } t \in C \text{ with } x \overset{t}{\Rightarrow}_T \Xi_T(\omega) \,\} \\ & . \end{aligned} \tag{4.10}$$

As discussed in section 2.6 the idea of the nonconflicting completion semantics of an automaton G is that each nonconflicting completion represents a requirement that needs to be satisfied by any test that is to be nonblocking in combination with G. If the test can execute the trace c associated with a nonconflicting completion C, then, in order to be nonblocking in combination with G, the test must be able to terminate with at least one of the traces $t \in C$.

The following result shows that the generalised nonconflicting completion semantics can be explained in the same way: if a pair (c,C) is contained in the semantics, then every test that can enter an α-marked state after trace c must be able to terminate with at least one of the traces in C, in order to be (α,ω)-nonblocking in combination with G.

Proposition 4.6 Let $G = \langle \Sigma, \Pi, Q, \rightarrow, Q^\circ, \Xi \rangle$ be a multi-coloured automaton with $\alpha, \omega \in \Pi$. The generalised nonconflicting completion semantics can be alternatively characterised as

$$\begin{aligned} \mathrm{CC}^{\omega}_{(\alpha,\omega)}(G) \;=\; \{\, (c,C) \in \Sigma^* \times 2^{\Sigma^*} \mid & \text{For every automaton } T \text{ such that } G \,\|\, T \text{ is} \\ & (\alpha,\omega)\text{-nonblocking and } T \overset{c}{\Rightarrow} x \in \Xi_T(\alpha), \text{ there exists } t \in C \\ & \text{with } x \overset{t}{\Rightarrow}_T \Xi_T(\omega) \,\} \,. \end{aligned} \tag{4.11}$$

Proof. Let $(c,C) \in \mathrm{CC}^{\omega}_{(\alpha,\omega)}(G)$ and $T = \langle \Sigma, \Pi, Q_T, \rightarrow_T, Q_T^\circ, \Xi_T \rangle$ such that $G \,\|\, T$ is (α,ω)-nonblocking and $T \overset{c}{\Rightarrow} x_T \in \Xi_T(\alpha)$. Since $(c,C) \in \mathrm{CC}^{\omega}_{(\alpha,\omega)}(G)$, there exists $x \in \Xi(\alpha)$ such that $G \overset{c}{\Rightarrow} x$ and $\mathscr{L}^\omega(x) \subseteq C$. Then $G \,\|\, T \overset{c}{\Rightarrow} (x, x_T) \in \Xi(\alpha) \times \Xi_T(\alpha)$, and since $G \,\|\, T$ is (α,ω)-nonblocking there exists $t \in \Sigma^*$ such that $(x, x_T) \overset{t}{\Rightarrow} \Xi(\omega) \times \Xi_T(\omega)$. This implies $x_T \overset{t}{\Rightarrow}_T \Xi_T(\omega)$ and $t \in \mathscr{L}^\omega(x) \subseteq C$.

Now let $(c,C) \in \Sigma^* \times 2^{\Sigma^*}$, and assume that for every automaton $T = \langle \Sigma, \Pi, Q_T, \rightarrow_T, Q_T^\circ, \Xi_T \rangle$ such that $G \,\|\, T$ is (α,ω)-nonblocking and $T \overset{c}{\Rightarrow} x \in \Xi_T(\alpha)$, there exists $t \in C$ such that $x \overset{t}{\Rightarrow}_T \Xi_T(\omega)$. Consider a deterministic automaton $T = \langle \Sigma, \Pi, Q_T, \rightarrow_T, Q_T^\circ,$

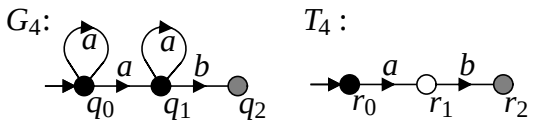


Figure 4.2: Standard nonconflicting completion semantics may be not well-founded.

$\Xi_T\rangle$ such that $\mathscr{L}(T)=\Sigma^*$, $\mathscr{L}^\alpha(T)=\{c\}$, and $\mathscr{L}^\omega(T)=c(\Sigma^*\setminus C)$. There exists exactly one state $x_T\in\Xi_T(\alpha)$, which also satisfies $T\stackrel{c}{\Rightarrow}x_T$ and $\mathscr{L}^\omega(x_T)=\Sigma^*\setminus C$, so there does not exist $t\in C$ such that $x_T\stackrel{t}{\Rightarrow}\Xi_T(\omega)$. By assumption it follows that $G\parallel T$ is (α,ω)-blocking. Then there exists a state $y\in\Xi(\alpha)\times\Xi_T(\alpha)$ such that $G\parallel T\Rightarrow y$ and $\mathscr{L}^\omega(y)=\emptyset$. By construction of T, there exists $x\in\Xi(\alpha)$ such that $y=(x,x_T)$ and $G\parallel T\stackrel{c}{\Rightarrow}y=(x,x_T)$, and furthermore $\emptyset=\mathscr{L}^\omega(y)=\mathscr{L}^\omega(x)\cap\mathscr{L}^\omega(x_T)=\mathscr{L}^\omega(x)\cap(\Sigma^*\setminus C)$, which implies $\mathscr{L}^\omega(x)\subseteq C$. It follows that $(c,C)\in\mathrm{CC}^\omega_{(\alpha,\omega)}(G)$ by definition. □

This shows that the standard and generalised nonconflicting completion semantics are closely related to each other. Yet, there are also important differences. While the generalised nonconflicting completion semantics only is closed via upward closure, in standard nonblocking there are interdependencies between states that lead to further closure properties.

Example 4.4 [25] In order to be ω-nonblocking in combination with automaton G_4 in figure 4.2, a test must initially be able to accept at least one of the traces $ab,aab,aaab,\ldots$ Therefore, $\mathrm{CC}^\omega(G_4)$ contains the pair $(\varepsilon,\{a^+b\})$. Furthermore, any such test must be able to execute a in its initial state, and any test executing a initially must also be able to cope with G_4 being put back to its initial state q_0 by executing the selfloop in q_0. Therefore, such a test also has to accept at least one of the traces $aab,aaab,aaaab,\ldots$ in its initial state. It follows that $\mathrm{CC}^\omega(G_4)$ contains all the pairs $(\varepsilon,\{a^na^*b\})$ for $n\geq 1$.

This example shows that, even for a finite-state automaton, the standard nonconflicting completion semantics is not necessarily *well-founded*, and in general cannot be described by listing a finite set of minimal nonconflicting completions. For generalised nonblocking, this is possible. Due to the presence of α-markings, there always is the possibility for a test to be not α-marked for certain states.

Example 4.5 Consider automaton G_4 in figure 4.2 in combination with test T_4. Clearly, $G_4\parallel T_4$ is (α,ω)-nonblocking, because the only reachable α-marked state of the synchronous product $G_4\parallel T_4$ is the initial state, from which both automata can terminate by executing trace ab. However, the test T_4 cannot execute any trace $t\in\{a^na^*b\}$ for $n>1$, so unlike the case of standard nonblocking, $(\varepsilon,\{aaa^*b\})\notin\mathrm{CC}^\omega_{(\alpha,\omega)}(G_4)$.

The presence of α-markings makes the nonconflicting completions for different traces independent from each other. This leads to a simpler semantic model with a finite characterisation. It also means that some abstractions possible for standard nonblocking are not applicable to generalised nonblocking.

4.4 Canonical Automaton

For compositional reasoning, it is necessary to modify automata in such a way that generalised nonblocking equivalence is preserved. This is facilitated by the fact that the generalised nonconflicting completion semantics can be represented finitely. This section explains how the generalised nonconflicting completion semantics can be used to construct a canonical form for any given finite-state automaton, which is generalised nonblocking equivalent to the original automaton, and such that the canonical forms of any two generalised nonblocking equivalent automata are equal.

4.4.1 Construction from Semantics

To ensure uniqueness, the canonical form is constructed directly from the generalised nonconflicting completion semantics. More precisely, it is shown in the following how to construct a *canonical automaton* $\mathscr{CA}(\mathrm{CC})$ for any given model

$$\mathrm{CC} \subseteq \Sigma^* \times 2^{\Sigma^*} . \tag{4.12}$$

Afterwards, an algorithm will be given to compute the canonical automaton for any given multi-coloured automaton G.

The canonical automaton consists of two parts, called the *upper* and *lower automaton*. The upper automaton of CC essentially is a minimal deterministic recogniser of the language covered by CC,

$$\mathscr{L}(\mathrm{CC}) = \{c \in \Sigma^* \mid \text{There exists } C \subseteq \Sigma^* \text{ such that } (c,C) \in \mathrm{CC}\}. \tag{4.13}$$

The lower automaton consists of minimal deterministic recognisers of all the nonconflicting completions in CC, which are linked to transitions from the corresponding states in the upper automaton.

To ensure uniqueness, the upper automaton needs to be minimised in such a way that traces leading to equal nonconflicting completions in the future are mapped to the same state of the upper automaton. The following definition provides the necessary equality for any given model CC.

Definition 4.10 Let $\mathrm{CC} \subseteq \Sigma^* \times 2^{\Sigma^*}$. Two traces $c_1, c_2 \in \Sigma^*$ are said to be *equivalent modulo* CC, written $c_1 \equiv_{\mathrm{CC}} c_2$, if for all $t \in \Sigma^*$ and all $C \subseteq \Sigma^*$, it holds that $(c_1 t, C) \in \mathrm{CC}$ if and only if $(c_2 t, C) \in \mathrm{CC}$.

Given this definition, the state set of the upper automaton is

$$U_{\mathrm{CC}} = \overline{\mathscr{L}(\mathrm{CC})}/\equiv_{\mathrm{CC}} , \tag{4.14}$$

and the transitions of the upper automaton are

$$[s]_{\mathrm{CC}} \xrightarrow{\sigma}_{U,\mathrm{CC}} [s\sigma]_{\mathrm{CC}} \quad \text{for all } s\sigma \in \overline{\mathscr{L}(\mathrm{CC})}. \tag{4.15}$$

Here, $[s]_{\mathrm{CC}} = \{ s' \in \overline{\mathscr{L}(\mathrm{CC})} \mid s \equiv_{\mathrm{CC}} s' \}$ denotes the *equivalence class* of s modulo $\equiv_{\mathrm{CC}}$, and for $L \subseteq \Sigma^*$, the notation $L/\equiv_{\mathrm{CC}} = \{ [s]_{\mathrm{CC}} \mid s \in L \}$ represents its partition into equivalence classes.

The lower automaton consists of deterministic recognisers for all the nonconflicting completions. It includes states accepting each of the following languages,

$$V_{\mathrm{CC}} = \{ C\omega/t \mid \text{There exists } c \in \Sigma^* \text{ such that } (c, C) \in \mathrm{CC}, \text{ and } t \in \overline{C} \} . \tag{4.16}$$

Here, $L/s = \{ t \in \Sigma^* \mid st \in L \}$ denotes the continuation language of $L \subseteq \Sigma^*$ after $s \in \Sigma^*$. To ensure minimality and thus uniqueness, it is convenient to identify the states of the lower automaton with the languages in V_{CC}. Accordingly, the transitions of the lower automaton are

$$L \xrightarrow{\sigma}_{V,\mathrm{CC}} L/\sigma \quad \text{for all } L \in V_{\mathrm{CC}} \text{ and } \sigma \in \Sigma \cap \overline{L}. \tag{4.17}$$

A lower-automaton state in $L \in V_{\mathrm{CC}}$ is marked ω if and only if $\omega \in L$. This ensures that the ω-marked languages of these states are equal to the languages they represent, i.e.,

$$\mathscr{L}^{\omega}(L\omega) = L \quad \text{for each } L\omega \in V_{\mathrm{CC}} . \tag{4.18}$$

To complete the lower automaton, each nonconflicting completion in CC is associated with its own α-marked state. The α-marked states may only be accessed from the upper automaton and therefore need to be distinct from any lower-automaton state. Therefore, the following additional states are used,

$$V_{\mathrm{CC}}^{\alpha} = \{ (C, \alpha) \mid \text{There exists } c \in \Sigma^* \text{ such that } (c, C) \in \mathrm{CC} \} . \tag{4.19}$$

Given these state sets and transitions, the *canonical automaton* for CC is con-

structed as follows,

$$\mathscr{CA}(\mathrm{CC}) = \langle \Sigma, \{\alpha, \omega\}, Q_{\mathrm{CA}}, \rightarrow_{\mathrm{CA}}, Q^{\circ}_{\mathrm{CA}}, \Xi_{\mathrm{CA}} \rangle \tag{4.20}$$

where

- $Q_{\mathrm{CA}} = U_{\mathrm{CC}} \cup V_{\mathrm{CC}} \cup V^{\alpha}_{\mathrm{CC}}$;
- $\rightarrow[\mathrm{CA}] = \ \rightarrow[U, \mathrm{CC}] \cup \rightarrow[V, \mathrm{CC}] \cup \{ ([c]_{\mathrm{CC}}, \tau, (C, \alpha)) \mid (c, C) \in \mathrm{CC} \} \cup \{ ((C, \alpha), \tau, C\omega) \mid (C, \alpha) \in V^{\alpha}_{\mathrm{CC}} \}$;
- $Q^{\circ}_{\mathrm{CA}} = \{[\varepsilon]_{\mathrm{CC}}\} \setminus \{\emptyset\}$;
- $\Xi_{\mathrm{CA}}(\alpha) = V^{\alpha}_{\mathrm{CC}}$;
- $\Xi_{\mathrm{CA}}(\omega) = \{ C \in V_{\mathrm{CC}} \mid \omega \in C \}$.

The canonical automaton has a simple regular form, but it is not necessarily minimal. For example, the α-marked states can be merged into their successors, if those successors do not have other incoming transitions. The potential for reduction becomes clear in example 4.6 below.

The following result confirms that the canonical automaton construction preserves generalised nonblocking in that the generalised nonconflicting completion semantics of the canonical automaton is equal to the upwards closure of the model CC, from which the automaton was constructed.

Proposition 4.7 Let $\mathrm{CC} \subseteq \Sigma^* \times 2^{\Sigma^*}$. Then

$$\mathrm{CC}^{\omega}_{(\alpha,\omega)}(\mathscr{CA}(\mathrm{CC})) = \mathrm{CC}^{\uparrow} . \tag{4.21}$$

Proof. First, let $(c, C) \in \mathrm{CC}^{\omega}_{(\alpha,\omega)}(\mathscr{CA}(\mathrm{CC}))$. Then there exists $x \in \Xi_{\mathrm{CA}}(\alpha)$ such that $\mathscr{CA}(\mathrm{CC}) \stackrel{c}{\Rightarrow} x$ and $\mathscr{L}^{\omega}(x) \subseteq C$. By construction, this means that $x \in V^{\alpha}_{\mathrm{CC}}$, so $x = (C', \alpha)$ for some $(c', C') \in \mathrm{CC}$. Also by construction of the upper automaton, since $\mathscr{CA}(\mathrm{CC}) \stackrel{c}{\Rightarrow} x = (C', \alpha)$, it follows that $\mathscr{CA}(\mathrm{CC}) \stackrel{c}{\rightarrow} [c]_{\mathrm{CC}} \stackrel{\tau}{\rightarrow} (C', \alpha)$, which implies $(c, C') \in \mathrm{CC}$. Furthermore by construction of the lower automaton, $C' = \mathscr{L}^{\omega}(C'\omega) = \mathscr{L}^{\omega}((C', \alpha)) = \mathscr{L}^{\omega}(x) \subseteq C$, so it follows from $(c, C') \in \mathrm{CC}$ that $(c, C) \in \mathrm{CC}^{\uparrow}$.

Second, let $(c, C) \in \mathrm{CC}^{\uparrow}$. Then there exists $C' \subseteq C$ such that $(c, C') \in \mathrm{CC}$. By construction of the upper automaton, $\mathscr{CA}(\mathrm{CC}) \stackrel{c}{\rightarrow} [c]_{\mathrm{CC}} \stackrel{\tau}{\rightarrow} (C', \alpha) \in \Xi_{\mathrm{CA}}(\alpha)$, and by construction of the lower automaton, $(C', \alpha) \stackrel{\tau}{\rightarrow} C'\omega$ and $\mathscr{L}^{\omega}((C', \alpha)) = \mathscr{L}^{\omega}(C'\omega) = C' \subseteq C$. Thus, given $\mathscr{CA}(\mathrm{CC}) \stackrel{c}{\Rightarrow} (C', \alpha) \in \Xi_{\mathrm{CA}}(\alpha)$ and $\mathscr{L}^{\omega}((C', \alpha)) \subseteq C$, it follows by definition of $\mathrm{CC}^{\omega}_{(\alpha,\omega)}$ that $(c, C) \in \mathrm{CC}^{\omega}_{(\alpha,\omega)}(\mathscr{CA}(\mathrm{CC}))$. □

The canonical automaton can be constructed for any model CC, but the result is only finite-state if the set of nonconflicting completions in CC is finite, and the upper automaton has a finite-state representation. These conditions can be ensured when CC is obtained from the generalised nonconflicting completion semantics of a finite-state automaton. In this case, the upper automaton is finite-state because of the finite number of α-states from which nonconflicting completions can originate, and although the set of nonconflicting completions is typically infinite due to upwards closure, it is enough to construct the canonical automaton using only minimal nonconflicting completions.

Definition 4.11 The *canonical form* of a finite-state multi-coloured automaton G is

$$\mathscr{CA}(G) = \mathscr{CA}(\mathrm{CC}^{\omega}_{(\alpha,\omega)}(G)^{\downarrow}) \; . \tag{4.22}$$

As explained above, the canonical form of an automaton G is finite-state as long as G is finite-state. Given the previous results, it is not difficult to show that the canonical form is unique for all generalised nonblocking equivalent automata.

Proposition 4.8 Let G and H be two finite-state multi-coloured automata. Then

$$G \simeq_{(\alpha,\omega)} H \;\text{ if and only if }\; \mathscr{CA}(G) = \mathscr{CA}(H) \; . \tag{4.23}$$

Proof. First assume that $G \simeq_{(\alpha,\omega)} H$. It follows that $\mathrm{CC}^{\omega}_{(\alpha,\omega)}(G) = \mathrm{CC}^{\omega}_{(\alpha,\omega)}(H)$ by proposition 4.5, which implies $\mathscr{CA}(G) = \mathscr{CA}(\mathrm{CC}^{\omega}_{(\alpha,\omega)}(G)^{\downarrow}) = \mathscr{CA}(\mathrm{CC}^{\omega}_{(\alpha,\omega)}(H)^{\downarrow}) = \mathscr{CA}(H)$ by definition.

Second assume that $\mathscr{CA}(G) = \mathscr{CA}(H)$. From the fact that G is finite-state and proposition 4.7, it follows that

$$\begin{aligned}
\mathrm{CC}^{\omega}_{(\alpha,\omega)}(G) &= \mathrm{CC}^{\omega}_{(\alpha,\omega)}(G)^{\downarrow\uparrow} \\
&= \mathrm{CC}^{\omega}_{(\alpha,\omega)}(\mathscr{CA}(\mathrm{CC}^{\omega}_{(\alpha,\omega)}(G)^{\downarrow})) \\
&= \mathrm{CC}^{\omega}_{(\alpha,\omega)}(\mathscr{CA}(G)) \\
&= \mathrm{CC}^{\omega}_{(\alpha,\omega)}(\mathscr{CA}(H)) \\
&= \mathrm{CC}^{\omega}_{(\alpha,\omega)}(\mathscr{CA}(\mathrm{CC}^{\omega}_{(\alpha,\omega)}(H)^{\downarrow})) \\
&= \mathrm{CC}^{\omega}_{(\alpha,\omega)}(H)^{\downarrow\uparrow} \\
&= \mathrm{CC}^{\omega}_{(\alpha,\omega)}(H) \; .
\end{aligned} \tag{4.24}$$

By proposition 4.5, this implies $G \simeq_{(\alpha,\omega)} H$. □

Proposition 4.8 shows that the canonical automaton can be used for identification of generalised nonblocking equivalent automata. To determine whether two finite-state

automata are generalised nonblocking equivalent, it is enough to construct their canonical automata and check whether they are equal.

Canonical automata can also be used to test the generalised nonblocking preorder. To check whether $G \lesssim_{(\alpha,\omega)} H$, it is possible to inspect all α-marked states of the synchronous product of the canonical forms of G and H and compare the associated languages. For every ω-marked language of an α-marked state of G, there needs to be a sublanguage associated with some corresponding α-marked state of H. The languages can be compared polynomially since they are represented deterministically in the canonical automata. However, the test for language inclusion requires only a deterministic representation for one of the two languages compared, and it is enough to construct only the canonical automaton of H to check whether $G \lesssim_{(\alpha,\omega)} H$.

4.4.2 Algorithmic Construction

In the previous section, the canonical automaton has been constructed from a semantic model CC, and its uniqueness has been established. This section proposes an algorithm that, given a finite-state multi-coloured automaton $G = \langle \Sigma, \Pi, Q, \rightarrow, Q^\circ, \Xi \rangle$, computes its canonical form $\mathscr{CA}(G)$.

The first step in the computation of the canonical automaton is the construction of the lower automaton, because it contains the languages associated with all α-marked states, which are needed to ensure minimality of the upper automaton.

The lower automaton consists of the minimal deterministic recognisers of all the ω-marked languages of all α-marked states of G. To construct it, the first step is to remove from G all states from where no ω-marked can be reached, that is, its state set is restricted to

$$R_\omega = \{\, x \in Q \mid x \rightarrow \Xi(\omega) \,\} \,. \tag{4.25}$$

Then subset construction [18] is used to construct a deterministic recogniser V^{det} of all nonconflicting completion languages of G. The subset construction starts with initial state sets corresponding to each α-marked state and continues until all reachable state sets have been explored. More precisely,

$$V^{\text{det}} = \langle \Sigma, \{\omega\}, 2^R_\omega, \rightarrow_V, Q^\circ_V, \Xi_V \rangle \tag{4.26}$$

where

- $X \xrightarrow{\sigma}_V Y$ for $X, Y \subseteq R_\omega$ and $\sigma \in \Sigma$ if and only if $Y = \{\, y \in R_\omega \mid X \stackrel{\sigma}{\Rightarrow} y \,\}$ and $Y \neq \emptyset$;
- $X \in Q^\circ_V$ if and only if $X = \{\, x \in R_\omega \mid x_\alpha \stackrel{\varepsilon}{\Rightarrow} x \,\}$ for some $x_\alpha \in \Xi(\alpha)$;
- $\Xi_V(\omega) = \{\, X \subseteq R_\omega \mid X \cap \Xi(\omega) \neq \emptyset \,\}$.

This automaton is then minimised using Hopcroft's algorithm [17] to obtain a unique and minimal lower automaton V. For each initial state x° of the minimised lower automaton, a new α-marked state x^α is created and linked via a τ-transition to x°. These α-marked state comprise the state set V^α. In order to link this automaton to the upper automaton later, a map is kept that links the α-marked states of G to their corresponding states in V^α.

Next, the upper automaton is constructed. In order to ensure that it accepts precisely the language $\mathscr{L}(\mathrm{CC}^\omega_{(\alpha,\omega)}(G)) = \mathscr{L}^\alpha(G)$, the state set of G is restricted to states from where an α-marked can be reached, i.e., to

$$R_\alpha = \{\, x \in Q \mid x \to \Xi(\alpha) \,\} \,. \tag{4.27}$$

Then a second subset construction is used to obtain a deterministic recogniser U^{det} of $\mathscr{L}^\alpha(G)$.

In order to establish uniqueness with respect to $\equiv_{\mathrm{CC}^\omega_{(\alpha,\omega)}(G)}$, for each state set $X \subseteq R_\alpha$ in this subset construction, the associated set of minimal nonconflicting completions,

$$\begin{aligned}\mathrm{CC}^\omega_{(\alpha,\omega)}(X) = \{\, C \subseteq \Sigma^* \mid \text{There exists } c \in \Sigma^* \text{ such that } G \overset{c}{\Rightarrow} X \text{ and } (c,C) \in \\ \mathrm{CC}^\omega_{(\alpha,\omega)}(G)^\downarrow \,\} \,,\end{aligned} \tag{4.28}$$

needs to be determined. Therefore, each state set X in the subset construction is associated with the set of all initial states of the lower automaton V that have been associated with some α-marked state contained in X. The ω-marked languages of these states are checked for language inclusion, and the initial states associated with non-minimal languages are removed from the set of languages associated with X. The ω-marked languages of the remaining states make up the set $\mathrm{CC}^\omega_{(\alpha,\omega)}(X)$.

Now the automaton U^{det} is minimised subject to an initial partition based on the sets (4.28). Two subset states $X, Y \subseteq R_\alpha$ can only be merged if

$$\mathrm{CC}^\omega_{(\alpha,\omega)}(X) = \mathrm{CC}^\omega_{(\alpha,\omega)}(Y) \,. \tag{4.29}$$

This is done using Hopcroft's algorithm [17] with an initial partition based on the minimised sets of α-marked states, which satisfies (4.29). The result is a unique minimal upper automaton with states partitioned in the coarsest possible way that respects $\equiv_{\mathrm{CC}}$.

The final step in the construction of the canonical automaton is to link the upper and lower automata. Each state $[X]$ of the minimised upper automaton is linked via a τ-transition to all the α-marked states in V^α that have been associated with some α-marked state of G contained in one of the state sets associated with the merged state $[X]$.

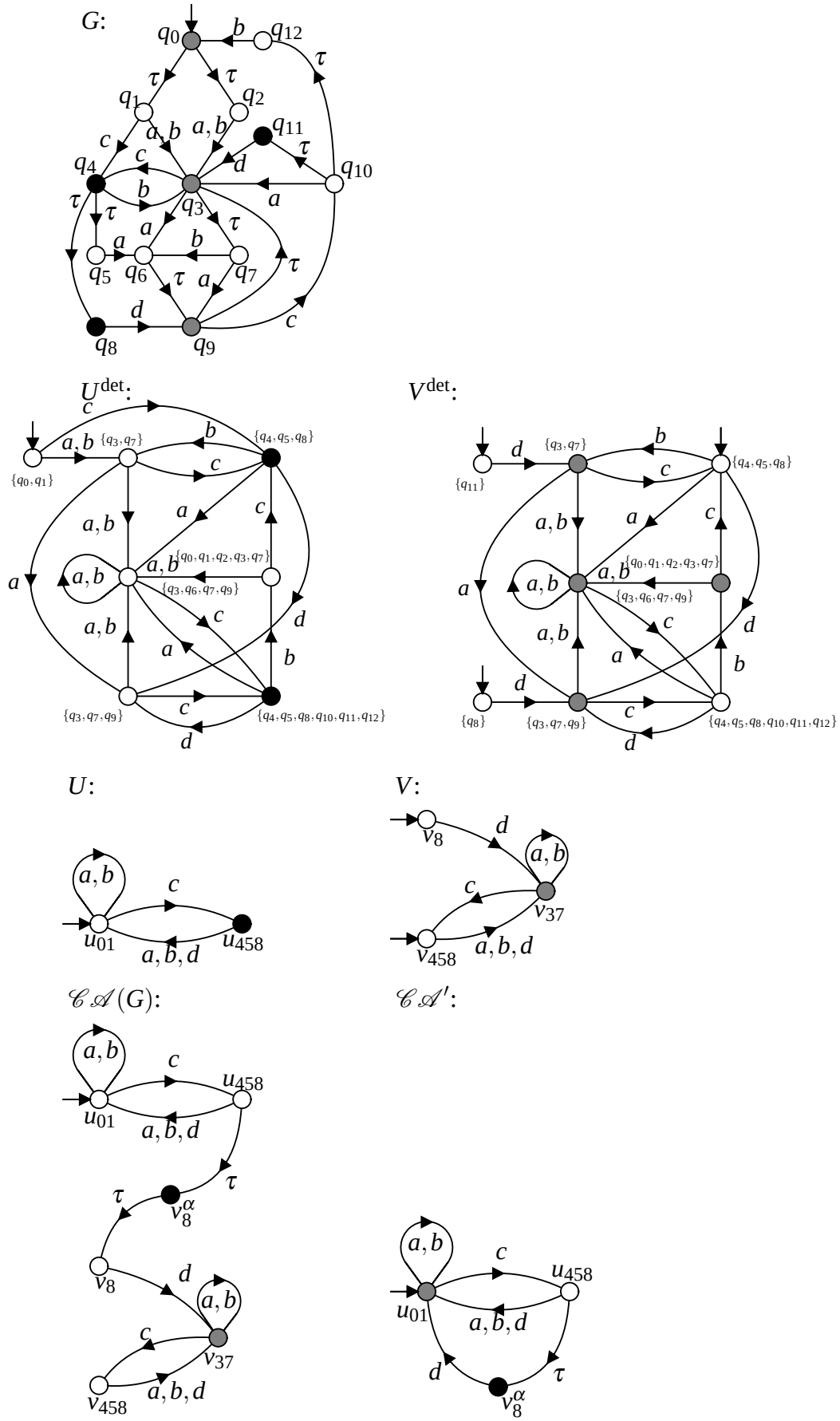


Figure 4.3: Example construction of canonical automaton.

Example 4.6 Figure 4.3 demonstrates the process of construction of the canonical form $\mathscr{CA}(G)$ of automaton G.

The first step is to apply subset construction starting from the three α-marked states q_4, q_8, and q_{11}. This results in the deterministic automaton V^{det} also shown in figure 4.3. Its three initial states $\{q_8\}$, $\{q_{11}\}$, and $\{q_4, q_5, q_8\}$ correspond to the three α-marked states of G, from which the subset construction originates—the α-marked state q_4 is expanded to $\{q_4, q_5, q_8\}$ because of its outgoing τ-transitions.

Next, the intermediate lower automaton V^{det} is minimised using Hopcroft's algorithm, resulting in the lower automaton V. After merging, this automaton has only two initial states: state v_8 corresponds to the original α-marked states q_8 and q_{11}, while v_{458} corresponds to the original α-marked state q_4. It can already be seen that the ω-marked language of v_8 is contained in the ω-marked language of v_{458}.

Next, to construct the upper automaton, subset construction is applied to G to obtain its deterministic form U^{det}. Owing to the fact that α- and ω-marked states are reachable from all states of G, this automaton is very similar to the intermediate lower automaton V^{det}. The α-marked states of U^{det} are $\{q_4, q_5, q_8\}$ and $\{q_4, q_5, q_8, q_{10}, q_{11}, q_{12}\}$. These states are both associated with the lower-automaton initial states v_8 and v_{458}, however since $\mathscr{L}^\omega(v_8) \subseteq \mathscr{L}^\omega(v_{458})$, only v_8 is considered. Both α-marked states are associated with equal sets of lower-automaton initial states, so they may be merged during minimisation. And indeed, minimisation results in the automaton U with only one α-marked state u_{458}.

Finally, the upper and lower automata are linked, resulting in the canonical automaton $\mathscr{CA}(G)$. The only α-marked state of the upper automaton is u_{458}, which is to be associated with v_8 in the lower automaton. Therefore, the new α-marked state v_8^α is created and linked via silent transitions to u_{458} and v_8.

It becomes clear that the canonical automaton, although unique, is not minimal. Since v_8^α has only one outgoing τ-transition that leads to state v_8 with no other incoming transitions, states v_8^α and v_8 can be merged while preserving generalised nonblocking equivalence. Furthermore, the language of lower-automaton state v_{37} is equal to the language of upper-automaton state u_{01}, and since for lower-automaton states only the language is relevant, v_{37} can be replaced by u_{01}. This results in the automaton $\mathscr{CA}'$, which is generalised nonblocking equivalent to $\mathscr{CA}(G)$ and to G.

The algorithm to construct the canonical automaton is exponential. The upper and lower automaton are obtained through subset construction, and the number of states of

the canonical automaton is bounded by

$$\begin{aligned} |U_{\mathrm{CC}}| + |V_{\mathrm{CC}}| + |V_{\mathrm{CC}}^{\alpha}| &\leq 2^{|Q|} + 2^{|Q|} + |\Xi(\alpha)| \\ &= O(2^{|Q|}) \ . \end{aligned} \tag{4.30}$$

To estimate the number of transitions, note that the upper and lower automaton are deterministic automata linked by two τ-transitions for each α-marked state. Thus, the number of transitions of the canonical automaton is bounded by

$$|\Sigma||U_{\mathrm{CC}}| + |\Sigma||V_{\mathrm{CC}}| + 2|V_{\mathrm{CC}}^{\alpha}| = O(|\Sigma|2^{|Q|}) \ . \tag{4.31}$$

The construction of the upper automaton requires tests for language inclusion to see whether languages associated to different α-marked states are contained in each other. There are up to $\frac{1}{2}|\Xi(\alpha)|(|\Xi(\alpha)|-1)$ pairs of α-marked states that need to be compared, and each test in the worst case requires construction of a synchronous product of two deterministic automata with $2^{|Q|}$ states each. The time complexity of the language inclusion check is determined by the number of transitions of the synchronous product, which is bounded by $|\Sigma|(2^{|Q|})^2 = |\Sigma|4^{|Q|}$. In practice, the test can often be completed much faster, because identical states of G can be recognised in the subset construction, and because the test can stop early when language inclusion is not satisfied. Still, the worst-case time complexity of the algorithm to construct the canonical form is

$$O(|\Sigma||\Xi(\alpha)|^2 4^{|Q|}) = O(|\Sigma||Q|^2 4^{|Q|}) \ . \tag{4.32}$$

Despite its exponential complexity, subset construction is known to be well-behaved in many practical cases. In [34], subset construction has been used for compositional verification of safety properties of very large discrete-event systems models. Such results suggest that the canonical automaton may be a useful tool for compositional verification of generalised nonblocking.

Chapter 5

Comparing Two Automata with Respect to the Conflict Preorder

In this chapter we introduce a concrete algorithm with which it is possible to compare to arbitrary automata with respect to the conflict preorder, and thus by extension conflict equivalence.

The ability to compare two automata with respect to conflict equivalence is essential in order to construct a canonical form of automata with respect to conflict equivalence, as will be described in chapter 6. Furthermore, being able to calculate whether one automaton is more conflicting than another is useful within the field of supervisory control. In [24] it is described how to design interface automata for subsystems. One of the requirements for such interfaces is that they should be more conflicting than their subsystem.

In this chapter we introduce a state-based method of calculating whether one automaton is less conflicting than another using **LC** – *Pairs*. An **LC** – *Pair* is a pair of state sets which represent nonconflicting completions. We then go on to show that these **LC** – *Pairs* can be used to determine whether an automaton is less conflicting than another by looking at a finite number of state sets.

In addition the algorithm to test the conflict preorder has been implemented in the discrete event systems tool Supremica [1], and has been used to compare several automata. We give experimental results which show that while the algorithm in the worst case runs in linear exponential time, in practice for many automata of non-trivial size the algorithm can calculate an answer within seconds. As any two automata can be compared with respect to fair testing using conflict equivalence it is also possible to use this algorithm to test fair testing equivalence. In [31] an algorithm for testing fair testing equivalence is presented. While both algorithms have linear exponential time complexity, the algorithm presented in this chapter has lower time complexity than the

fair testing algorithm, furthermore the algorithm from that paper has not been implemented to the best of our knowledge whereas our algorithm has.

In the following, Section 5.1 introduces how nonconflicting completions can be used to compare automata with respect to the conflict preorder. Section 5.2 introduces less conflicting pairs and shows how they can be used to describe nonconflicting completions. Section 5.3 shows how less conflicting pairs can be used to calculate the set of certain conflicts of an automaton. Section 5.4 describes how less conflicting pairs can be used to characterise the conflict preorder. Afterwards, Section 5.5 proposes an algorithm to calculate less conflicting pairs for finite-state automata. Section 5.6 describes an implementation of the algorithm. Section 5.7 presents the results from using the algorithm to compare several automata together.

5.1 The Conflict Preorder and Nonconflicting Completions

As has already been described in section 2.6 a nonconflicting completion of the automaton A is a pair (c,C) of trace and language such that for every test automaton T and state x_T such that $A \parallel T$ is nonblocking and $T \xrightarrow{c} x_T$ it holds that $x_T \xrightarrow{s\omega}$ for some $s\omega \in C$. That is in order for a test automaton which is capable of performing the trace c to be nonblocking with A, all processes which can be reached in T after c must be capable of performing at least one trace in C.

In this subsection we introduce how we can use nonconflicting completions to test whether two automata are less conflicting. It is already known that nonconflicting completions can be used to compare two automata with respect to conflict-equivalence, but it was unknown how to use nonconflicting completions in a practical algorithm to compare two automata with respect to conflict-equivalence. There were two main problems in implementing such an algorithm. Firstly in general the set of nonconflicting completions of any given automaton is infinite. Second and more importantly previously is was not known how to calculate for any given tuple (c,C) and automaton A whether $(c,C) \in \mathrm{CC}^{\omega}(A)$. In this section we will show how nonconflicting completions can be used to test whether two automata are conflict-equivalent. In subsequent subsections we will go on to show how nonconflicting completions can be calculated using *LC* pairs.

Theorem 5.1 Let $A = \langle \Sigma, Q_A, \rightarrow_A, Q_A^{\circ} \rangle$ and $B = \langle \Sigma, Q_B, \rightarrow_B, Q_B^{\circ} \rangle$ be two automata. A is less conflicting than B if and only if for all $c \in \Sigma^*$ and all $x_A \in Q_A$ such that $A \stackrel{c}{\Rightarrow} x_A$ it holds that $(c, \mathscr{L}^{\omega}(x_A)) \in \mathrm{CC}^{\omega}(B)$.

Proof. First let us assume that for all $c \in \Sigma^*$ and all $x_A \in Q_A$ such that $A \stackrel{c}{\Rightarrow} x_A$ it holds that $(c, \mathscr{L}^\omega(x_A)) \in \mathrm{CC}^\omega(B)$. We will show that A is less conflicting than B.

Let T be a test automaton such that $B \parallel T$ is nonblocking. Let c be a trace and (x_A, x_T) be pair of states such that $A \parallel T \stackrel{c}{\Rightarrow} (x_A, x_T)$. From definition 2.7 as $(c, \mathscr{L}^\omega(x_A))$ $\in \mathrm{CC}^\omega(B))$ and $B \parallel T$ is nonblocking the state x_T must be able to do at least one trace $s\omega \in \mathscr{L}^\omega(x_A)$. Therefore $(x_A, x_T) \stackrel{s\omega}{\Rightarrow}$ and $A \parallel T$ is nonblocking.

Now let us assume that A is less conflicting than B we will show that for all $c \in \Sigma^*$ and all $x_A \in Q_A$ such that $A \stackrel{c}{\Rightarrow} x_A$ it holds that $(c, \mathscr{L}^\omega(x_A)) \in \mathrm{CC}^\omega(B)$.

Let $c \in \Sigma^*$ be a trace and $x_A \in Q_A$ be a state such that $A \stackrel{c}{\Rightarrow} x_A$. From definition 2.7 it holds that $(c, \mathscr{L}^\omega(x_A)) \in CC(B)$ if for any given test automaton T and state x_T such that $B \parallel T$ is nonblocking and $T \stackrel{c}{\rightarrow} x_T$ then $x_T \stackrel{s\omega}{\rightarrow}$ where $s\omega \in \mathscr{L}^\omega(x_A)$. Let T be an automaton and state x_T such that $B \parallel T$ is nonblocking and $T \stackrel{c}{\rightarrow} x_T$. As $A \stackrel{c}{\Rightarrow} x_A$ it holds that $A \parallel T \stackrel{c}{\Rightarrow} (x_A, x_T)$. Furthermore as $B \parallel T$ is nonblocking and $A \lesssim_{\mathrm{conf}} B$ it must also be the case that $A \parallel T$ is nonblocking, therefore there exists a trace $s\omega \in \mathscr{L}^\omega(x_A)$ such that $(x_A, x_T) \stackrel{s\omega}{\rightarrow}$. $x_T \stackrel{s\omega}{\rightarrow}$ can do this trace therefore $(c, \mathscr{L}^\omega(x_A)) \in \mathrm{CC}^\omega(B)$. □

5.2 Less Conflicting Pairs

Determining whether or not the pair (ε, C) for any given language C is in fact a non-conflicting completion is non trivial. In order to be able to determine whether C is in fact a nonconflicting completion of the automaton it is neccessary to identify the sets of states that the automaton may reach over the language C. This is done using the well-known *subset construction* [18]. To capture termination, the usual powerset state space is extended by a special state ω entered only after termination.

Definition 5.1 The *deterministic state space* of automaton $A = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ is

$$Q_A^{\mathrm{det}} = 2^Q \cup \{\omega\} \,, \tag{5.1}$$

and the *deterministic transition function* $\delta_A^{\mathrm{det}} \colon Q^{\mathrm{det}} \times (\Sigma \cup \{\omega\}) \rightarrow Q^{\mathrm{det}}$ for A is defined as

$$\delta_A^{\mathrm{det}}(X, \sigma) = \begin{cases} \omega, & \text{if } \sigma = \omega \text{ and } X \stackrel{\omega}{\Rightarrow}; \\ \{y \in Q \mid X \stackrel{\sigma}{\Rightarrow} y\}, & \text{otherwise.} \end{cases} \tag{5.2}$$

The deterministic transition function δ_A^{det} is extended to traces $s \in \Sigma^* \cup \Sigma^*\omega$ in the standard way. Note that $\delta_A^{\mathrm{det}}(X, s)$ is defined for every trace $s \in \Sigma^* \cup \Sigma^*\omega$; if none of the states in X accepts the trace s, this is indicated by $\delta_A^{\mathrm{det}}(X, s) = \emptyset$. This is also true for termination: if ω is enabled in some state in X, then $\delta_A^{\mathrm{det}}(X, \omega) = \omega$, otherwise $\delta_A^{\mathrm{det}}(X, \omega) = \emptyset$.

In order to determine whether or not the language C is a nonconflicting completion of the automaton B, we will use a possibly nondeterministic automaton A to represent the language of C. We will then compare the state sets which both A and B can reach in parallel to one another. Therefore, the deterministic transition function is also applied to pairs $\mathbf{X} = (X_A, X_B)$ of state sets $X_A \subseteq Q_A$ and $X_B \subseteq Q_B$.

$$\delta^{\det}_{A,B}(\mathbf{X}, s) = \delta^{\det}_{A,B}(X_A, X_B, s) = (\delta^{\det}_A(X_A, s), \delta^{\det}_B(X_B, s)) \ . \tag{5.3}$$

We now give a definition of less conflicting pairs. A pair of state sets (X_A, X_B) is a less conflicting pair if and only if the language represented by X_A is a nonconflicting completion of the automaton represented by X_B. The set of less conflicting pairs is defined hierarchichally in such a way that lower ranked less conflicting pairs can be used find higher ranked less conflicting pairs.

Definition 5.2 Let $A = \langle \Sigma, Q_A, \rightarrow_A, Q^\circ_A \rangle$ and $B = \langle \Sigma, Q_B, \rightarrow_B, Q^\circ_B \rangle$ be automata. The set $\mathbf{LC}(A,B) \subseteq Q^{\det}_A \times Q^{\det}_B$ of *less conflicting pairs* for A and B is inductively defined by

$$\begin{aligned}
\mathbf{LC}^0(A,B) =& \{\omega\} \times Q^{\det}_B \cup \{\, (X_A, X_B) \mid X_B \subseteq Q_B \text{ and there exists} \\
& x_B \in X_B \text{ with } \mathscr{L}^\omega(x_B) = \emptyset \,\} \ ; && (5.4) \\
\mathbf{LC}^{n+1}(A,B) =& \{\, (X_A, X_B) \mid \text{there exists } x_B \in X_B \text{ such that for all } t \in \Sigma^*, \text{ if} && (5.5) \\
& x_B \stackrel{t\omega}{\Rightarrow} \text{ then there exists } r \sqsubseteq t\omega \text{ such that } \delta^{\det}_{A,B}(X_A, X_B, r) \in \\
& \mathbf{LC}^i(A,B) \text{ for some } i \leq n \,\} \ ; \\
\mathbf{LC}(A,B) =& \bigcup_{n \geq 0} \mathbf{LC}^n(A,B) \ . && (5.6)
\end{aligned}$$

Remark 5.1 If $(X_A, X_B) \notin \mathbf{LC}(A,B)$, then according to (5.5), for every state $x_B \in X_B$, there exists $t \in \Sigma^*$ such that $x_B \stackrel{t\omega}{\Rightarrow}$, and $\delta^{\det}(X_A, X_B, r) \notin \mathbf{LC}(A,B)$ for all prefixes $r \sqsubseteq t\omega$.

The idea of definition 5.2 is to classify a pair (X_A, X_B) as less conflicting, if the marked language of X_A is a *nonconflicting completion* [25] for the process with initial states X_B. That is, every test that is nonconflicting in combination with each of the states in X_B can terminate with at least one trace from the marked language of X_A. Or conversely, every test that cannot terminate using any of the traces in the marked language of X_A also is conflicting with X_B (see lemma 5.3 below).

The first state set X_A of a pair (X_A, X_B) is just used to represent a *language* of possible completions. If state sets X_A and Y_A have the same languages, then all pairs (X_A, X_B) and (Y_A, X_B) have exactly the same less conflicting status. For the second state set X_B on the other hand, the complete nondeterministic behaviour is relevant.

A pair (ω, X_B) is considered as "less conflicting" (5.4), since termination has already been achieved in A. If X_B contains a state x_B such that $\mathscr{L}^\omega(x_B) = \emptyset$, then

(X_A, X_B) also is less conflicting (5.4), because conflict is guaranteed in X_B. For other pairs (X_A, X_B), it must be checked whether X_B contains a requirement to avert blocking matching that given by the language of X_A (5.5).

Example 5.1 Consider again the automata A_0 and B_0 in figure 2.2. It is the case that $(\{a_0\}, \{b_0\}) \in \mathbf{LC}^1(A_0, B_0)$. There are three ways to terminate from b_0, by executing ω or $\alpha\beta\omega$ or $\alpha\alpha\beta\omega$. All three traces are possible in a_0, each taking the pair $(\{a_0\}, \{b_0\})$ to the deterministic successor $(\omega, \omega) \in \mathbf{LC}^0(A_0, B_0)$. This is enough to confirm that (5.5) is satisfied.

On the other hand, $(\{a_0\}, \{b_2\}) \notin \mathbf{LC}^1(A_0, B_0)$. From state a_0, blocking occurs with a test T that can only execute $\beta\omega$, but this test is nonblocking with b_2. It holds that $b_2 \xrightarrow{\beta\omega}$, where trace $\beta\omega$ has the prefixes ε, β, and $\beta\omega$, but $\delta^{\det}_{A_0,B_0}(\{a_0\}, \{b_2\}, \varepsilon) = (\{a_0\}, \{b_2\}) \notin \mathbf{LC}^0(A_0, B_0)$, $\delta^{\det}_{A_0,B_0}(\{a_0\}, \{b_2\}, \beta) = (\emptyset, \{b_4\}) \notin \mathbf{LC}^0(A_0, B_0)$, and finally $\delta^{\det}_{A_0,B_0}(\{a_0\}, \{b_2\}, \beta\omega) = (\emptyset, \omega) \notin \mathbf{LC}^0(A_0, B_0)$. Therefore, (5.5) is not satisfied and $(\{a_0\}, \{b_2\}) \notin \mathbf{LC}^1(A_0, B_0)$. It can also be shown that $(\{a_0\}, \{b_2\}) \notin \mathbf{LC}(A_0, B_0)$.

For a *level-1* less conflicting pair $(X_A, X_B) \in \mathbf{LC}^1(A, B)$, if X_B does not contain blocking states, then there must exist a state $x_B \in X_B$ such that $\mathscr{L}^\omega(x_B) \subseteq \mathscr{L}^\omega(X_A)$. This is not the case for every less conflicting pair, as some nonblocking requirements are only implicitly contained in the automaton. To show that (X_A, X_B) is a less conflicting pair, it is enough to find a state in $x_B \in X_B$ that can cover an initial segment of $\mathscr{L}^\omega(X_A)$, as long as a less conflicting pair of a *lower level* is reached afterwards.

Example 5.2 Consider again automata A_2 and B_2 in figure 2.4. By definition, $(\omega, \omega) \in \mathbf{LC}^0(A_2, B_2)$, and following from this, $(\{a_1\}, \{b_0, b_1\}) \in \mathbf{LC}^1(A_2, B_2)$, because the marked language of a_1 is $\alpha^+\omega$, which also is the marked language of b_1.

Now consider the pair $(\{a_0\}, \{b_0, b_1\})$. State a_0 has the marked language $\alpha\alpha^+\omega$, i.e., to avert blocking from a_0, a test must be able to execute at least one of the traces in $\alpha\alpha^+\omega$. Although this language is not directly associated with any state in B_2, the nonblocking requirement is implicitly present in state b_1. If blocking is to be averted from state b_1, event α must be possible. After executing α, state b_0 is entered, from where it is always possible to silently return to state b_1 with marked language $\alpha^+\omega$. Therefore, in order to avert blocking from state b_1, it is necessary to execute α and afterwards be able to terminate using one of the traces in $\alpha^+\omega$. This amounts to the implicit nonblocking requirement to execute a trace from $\alpha\alpha^+\omega$ in state b_1.

Therefore $(\{a_0\}, \{b_0, b_1\}) \notin \mathbf{LC}^1(A_2, B_2)$, but $(\{a_0\}, \{b_0, b_1\}) \in \mathbf{LC}^2(A_2, B_2)$ according to (5.5): every trace that leads to a terminal state from state b_1 has the prefix α, and $\delta^{\det}_{A_2,B_2}(\{a_0\}, \{b_0, b_1\}, \alpha) = (\{a_1\}, \{b_0, b_1\}) \in \mathbf{LC}^1(A_2, B_2)$.

As shown in the example, some nonblocking requirements have to be constructed using a saturation operation that combines two previously found nonblocking requirements. The level n of a less conflicting pair $(X_A, X_B) \in \mathbf{LC}^n(A,B)$ represents the nesting depth of applications of this saturation operation.

The level of less conflicting pairs can also be seen as measuring *progress* towards termination. When moving to the next level, a strongly connected component is exited and some state combinations become unreachable. The idea of progress is essential for conflict semantics. By (5.6), every less conflicting pair must be in a set $\mathbf{LC}^n(A,B)$ for some $n \in \mathbb{N}$, even for infinite-state systems.

In the following lemma we shows that under all circumstances where (X_A, X_B) is not in $\mathbf{LC}(A,B)$, that there exists an automaton which is both nonblocking with X_B and incapable of performing any trace in $\mathscr{L}^\omega(X_A)$. This automaton is a counterexample which shows that $\mathscr{L}^\omega(X_A)$ is not a nonconflicting completion of B.

Lemma 5.1 Let $A = \langle \Sigma, Q_A, \to_A, Q_A^\circ \rangle$ and $B = \langle \Sigma, Q_B, \to_B, Q_B^\circ \rangle$ be automata. Let $\mathbf{X} = (X_A, X_B) \notin \mathbf{LC}(A,B)$. Let $B' = \langle \Sigma, Q, \to, Q^\circ \rangle X_B[B]$.

Then there exists a deterministic automaton $T_\mathbf{X} = \langle \Sigma, Q_T, \to_T, Q_T^\circ\{x_T^\circ\} \rangle$ such that both the following conditions hold.

(i) $\mathscr{L}^\omega(X_A) \cap \mathscr{L}^\omega(T) = \emptyset$.

(ii) $B' \parallel T$ is nonblocking.

Proof. Construct the deterministic automaton $T_\mathbf{X} = \langle \Sigma, Q_T, \to_T, Q_T^\circ\{x_T^\circ\} \rangle$ such that

$$\mathscr{L}(T_\mathbf{X}) = \{ s \in \Sigma^* \cup \Sigma^*\omega \mid \delta_{A,B}^{\det}(\mathbf{X}, r) \notin \mathbf{LC}(A,B) \text{ for all } r \sqsubseteq s \} . \tag{5.7}$$

This language is prefix-closed by construction and nonempty because $\mathbf{X} \notin \mathbf{LC}(A,B)$. Therefore, $T_\mathbf{X}$ is a well-defined automaton.

(i) Let $x_A \in X_A$. If $x_A \overset{t\omega}{\Rightarrow}$ for some $t \in \Sigma^*$, then $\delta_{A,B}^{\det}(\mathbf{X}, t\omega) = (\omega, Y_B) \in \mathbf{LC}^0(A,B) \subseteq \mathbf{LC}(A,B)$ for some $Y_B \in Q_B^{\det}$ by definition 5.1 and 5.2. It follows from (5.7) that $t\omega \notin \mathscr{L}(T_\mathbf{X})$, and thus $(x_A, x_T^\circ) \overset{t\omega}{\Rightarrow}$ does not hold. Since $t \in \Sigma^*$ was chosen arbitrarily, it follows that $\mathscr{L}^\omega(x_A, x_T^\circ) = \emptyset$. Therefore $\mathscr{L}^\omega(X_A) \cap \mathscr{L}^\omega(T) = \emptyset$.

(ii) Let $x_B \in X_B$, $y_B \in Q_B$, $y_T \in Q_T$, and $s \in \Sigma^*$ such that $B \parallel T \overset{s}{\Rightarrow} (y_B, y_T)$. From (5.7) it follows that $\delta_{A,B}^{\det}(\mathbf{X}, s) \notin \mathbf{LC}(A,B)$. Let $\delta_{A,B}^{\det}(\mathbf{X}, s) = \mathbf{Y}$. Then $\mathbf{Y} \notin \mathbf{LC}(A,B)$, so there exists a trace $t \in \Sigma^*$ such that $y_B \overset{t\omega}{\Rightarrow}$ and for all $r \sqsubseteq t$ it holds that $\delta_{A,B}^{\det}(\mathbf{Y}, r) \notin \mathbf{LC}(A,B)$ (see remark 5.1). Thus $x_B \overset{s}{\Rightarrow} y_B \overset{t\omega}{\Rightarrow}$ and for all prefixes $u \sqsubseteq st\omega$, it holds that $\delta_{A,B}^{\det}(\mathbf{X}, u) \notin \mathbf{LC}(A,B)$. Then $st\omega \in \mathscr{L}(T_\mathbf{X})$ according to (5.7), and since $T_\mathbf{X}$ is deterministic, it follows that $y_T \overset{t\omega}{\Rightarrow}$. Therefore, $(y_B, y_T) \overset{t\omega}{\Rightarrow}$, As s, x_B, y_B and y_T were chose arbitrarily $B' \parallel T$ is nonblocking. □

We further show that if the tuple $(X_A, X_B) \in \mathbf{LC}(A,B)$ then for any test automaton T, and any state x_T from the automaton T it must be the case that either $X_A \parallel x_T$ is nonblocking or $X_B \parallel x_T$ can reach a blocking state.

Lemma 5.2 Let $A = \langle \Sigma, Q_A, \rightarrow_A, Q_A^\circ \rangle$, $B = \langle \Sigma, Q_B, \rightarrow_B, Q_B^\circ \rangle$, and $T = \langle \Sigma, Q_T, \rightarrow_T, Q_T^\circ \rangle$ be automata, and let $x_T \in Q_T$ be a (possibly unreachable) state. For every less conflicting pair $(X_A, X_B) \in \mathbf{LC}(A,B)$, at least one of the following conditions holds.

(i) $X_A = \omega$, or $X_A \subseteq Q_A$ and there exists $x_A \in X_A$ such that $\mathscr{L}^\omega(x_A, x_T) \neq \emptyset$.

(ii) There exists states $x_B \in X_B$, $y_B \in Q_B$, and $y_T \in Q_T$ such that $(x_B, x_T) \Rightarrow (y_B, y_T)$ and $\mathscr{L}^\omega(y_B, y_T) = \emptyset$.

(Here and in the following, the notation $\mathscr{L}^\omega(x_A, x_T)$ is abused to be a shorthand for $\mathscr{L}^\omega((x_A, x_T))$.)

Proof. As (X_A, X_B) is a less conflicting pair, it holds that $(X_A, X_B) \in \mathbf{LC}^n(A,B)$ for some $n \in \mathbb{N}$. The claim is shown by induction on n.

If $(X_A, X_B) \in \mathbf{LC}^0(A,B)$ then by (5.4) it holds that $X_A = \omega$, or $X_B \subseteq Q_B$ and there exists $x_B \in X_B$ such that $\mathscr{L}^\omega(x_B) = \emptyset$. In the first case (i) holds, and in the second case (ii) holds as $(x_B, x_T) \xrightarrow{\varepsilon} (x_B, x_T)$ and $\mathscr{L}^\omega(x_B, x_T) = \mathscr{L}^\omega(x_B) \cap \mathscr{L}^\omega(x_T) = \emptyset$.

Now assume the claim holds for all $i \leq n$, i.e., for all $(X_A, X_B) \in \mathbf{LC}^i(A,B)$, one of the conditions (i) or (ii) holds, and consider $(X_A, X_B) \in \mathbf{LC}^{n+1}(A,B)$. By (5.5), there exists $x_B \in X_B$ such that for all $t \in \Sigma^*$, if $x_B \overset{t\omega}{\Rightarrow}$ then there exists a prefix $r \sqsubseteq t\omega$ such that $\delta^{\det}_{A,B}(X_A, X_B, r) \in \mathbf{LC}^i(A,B)$ for some $i \leq n$. If $\mathscr{L}^\omega(x_B, x_T) = \emptyset$, (ii) follows immediately as $(x_B, x_T) \xrightarrow{\varepsilon} (x_B, x_T)$. Therefore assume that $\mathscr{L}^\omega(x_B, x_T) \neq \emptyset$, i.e., there exists $t \in \Sigma^*$ such that $(x_B, x_T) \overset{t\omega}{\Rightarrow}$. Then $x_B \overset{t\omega}{\Rightarrow}$, so there exists $r \sqsubseteq t\omega$ such that $\delta^{\det}_{A,B}(X_A, X_B, r) \in \mathbf{LC}^i(A,B)$ for some $i \leq n$. As $r \sqsubseteq t\omega$ and $x_T \overset{t\omega}{\Rightarrow}$, it also holds that $x_T \overset{r}{\Rightarrow} y_T$ for some $y_T \in Q_T$. Let $\delta^{\det}_{A,B}(X_A, X_B, r) = (Y_A, Y_B)$. By inductive assumption, (i) or (ii) holds for $(Y_A, Y_B) \in \mathbf{LC}^i(A,B)$ and y_T.

(i) In this case, either $Y_A = \omega$, or $Y_A \subseteq Q_A$ and there exists $y_A \in Y_A$ and $u \in \Sigma^*$ such that $(y_A, y_T) \overset{u\omega}{\Rightarrow}$. If $Y_A = \omega$, then $\delta^{\det}_A(X_A, r) = Y_A = \omega$ and according to definition 5.1 there exists $r_A \in \Sigma^*$ such that $r = r_A\omega$, and there exists states $x_A \in X_A$ and $y_A \in Q_A$ such that $x_A \overset{r_A}{\Rightarrow} y_A \overset{\omega}{\Rightarrow}$, i.e., $(x_A, x_T) \overset{r_A\omega}{\Longrightarrow}$. If there exists $y_A \in Y_A$ and $u \in \Sigma^*$ such that $(y_A, y_T) \overset{u\omega}{\Rightarrow}$, then since $\delta^{\det}_A(X_A, r) = Y_A$, there exists $x_A \in X_A$ such that $x_A \overset{r}{\Rightarrow} y_A$, i.e., $(x_A, x_T) \overset{r}{\Rightarrow} (y_A, y_T) \overset{u\omega}{\Rightarrow}$. In both cases, (i) holds for (X_A, X_B) and x_T.

(ii) If there exists a state $y_B \in Y_B$ such that $(y_B, y_T) \Rightarrow (z_B, z_T)$ where $\mathscr{L}^\omega(z_B, z_T) = \emptyset$, then since $\delta^{\det}_B(X_B, r) = Y_B$, there exists $x_B \in X_B$ such that $x_B \overset{r}{\Rightarrow} y_B$, which implies $(x_B, x_T) \overset{r}{\Rightarrow} (y_B, y_T) \Rightarrow (z_B, z_T)$ with $\mathscr{L}^\omega(z_B, z_T) = \emptyset$. Thus, (ii) holds for (X_A, X_B) and x_T. $\square$

We further go on to show that if (X_A, X_B) is in $\mathbf{LC}(A,B)$ that it is in fact the case that $\mathscr{L}^\omega(X_A)$ is a nonconflicting completion of X_B.

Lemma 5.3 Let $A = \langle \Sigma, Q_A, \rightarrow_A, Q_A^\circ \rangle$, $B = \langle \Sigma, Q, \rightarrow, X_B \rangle [B]$ be automata. Let $X_A \subseteq Q_A$. Then $(\varepsilon, \mathscr{L}^\omega(X_A)) \in \mathrm{CC}^\omega(B)$ if and only if $(X_A, X_B) \in \mathbf{LC}(A,B)$

Proof. First we will prove that if $(X_A, X_B) \notin \mathbf{LC}(A,B)$ then $(\varepsilon, \mathscr{L}^\omega(X_A)) \notin \mathrm{CC}^\omega(B)$.

From lemma 5.1 their exists an automaton T such that $\mathscr{L}^\omega(Y_A) \cap \mathscr{L}^\omega(Q_T^\circ) = \emptyset$ and $B \parallel T$ is nonblocking. Clearly $T \xrightarrow{\varepsilon} Q_T^\circ$ therefore $(\varepsilon, \mathscr{L}^\omega(X_A)) \notin \mathrm{CC}^\omega(B)$.

Second we will prove that if $(X_A, X_B) \in \mathbf{LC}(A,B)$ then $(\varepsilon, \mathscr{L}^\omega(X_A)) \in \mathrm{CC}^\omega(B)$.

Let $T = \langle \Sigma, Q_T, \rightarrow_T, Q_T^\circ \rangle$ be an automaton such that $B \parallel T$ is nonblocking we will show that $\mathscr{L}^\omega(X_A) \cap \mathscr{L}(Q_T^\circ) \neq \emptyset$.

As $(X_A, X_B) \in \mathbf{LC}(A,B)$ from lemma 5.2 either there exists a state $x_A \in X_A$ such that $\mathscr{L}^\omega(x_A) \cap \mathscr{L}^\omega(Q_T^\circ) \neq \emptyset$ or there exists states $x_B \in X_B$, $y_B \in Q_B$, and $y_T \in Q_T$ such that $(x_B, x_T) \Rightarrow (y_B, y_T)$ and $\mathscr{L}^\omega(y_B, y_T) = \emptyset$ In the first case as $\mathscr{L}^\omega(x_A) \subseteq \mathscr{L}^\omega(X_A)$ it holds that $\mathscr{L}^\omega(X_A) \cap \mathscr{L}(Q_T^\circ) \neq \emptyset$. In the second case $B \parallel T \Rightarrow (y_B, y_T)$, as $\mathscr{L}^\omega(y_B, y_T) = \emptyset$ it would hold that $B \parallel T$ is blocking. As this contradicts our assumption about T it must be the first case.

□

5.3 Less Conflicting Pairs and Certain Conflicts

Less conflicting pairs can be used to characterise the set of *certain conflicts* of an automaton as defined in 2.5. This shows the close link between the conflict preorder and the set of certain conflicts. If a pair $(\emptyset, X_B)$ is a less conflicting pair then, since termination is impossible from $\emptyset$, conflict must be also present in X_B. In this case, every trace leading to X_B must be a trace of certain conflicts. This observation leads to the following alternative characterisation of the set of certain conflicts.

Theorem 5.2 The set of certain conflicts of $B = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ can also be written as

$$\textsc{Conf}(B) = \{ s \in \Sigma^* \mid (\emptyset, \delta_B^{\det}(Q^\circ, r)) \in \mathbf{LC}(O,B) \text{ for some prefix } r \sqsubseteq s \}\,, \qquad (5.8)$$

where $O = \langle \Sigma, \emptyset, \emptyset, \emptyset \rangle$ stands for the empty automaton.

Proof. First let $s \in \Sigma^*$ such that $(\emptyset, \delta_B^{\det}(Q^\circ, r)) \in \mathbf{LC}(O,B)$ for some $r \sqsubseteq s$, and let $T = \langle \Sigma, Q_T, \rightarrow_T, Q_T^\circ \rangle$ be an automaton such that $T \stackrel{s}{\Rightarrow}$. It is to be shown that $B \parallel T$ is blocking. Since $T \stackrel{s}{\Rightarrow}$ and $r \sqsubseteq s$, it holds that $T \stackrel{r}{\Rightarrow} x_T$ for some state $x_T \in Q_T$. Since $(\emptyset, \delta_B^{\det}(Q^\circ, r)) \in \mathbf{LC}(O,B)$, either (i) or (ii) in lemma 5.2 holds. However, (i) is impossible as the first state set of the pair is empty, so (ii) must be true. Thus, there exists

a state $x \in \delta_B^{\det}(Q^\circ, r)$ such that $(x, x_T) \Rightarrow (y, y_T)$ where $\mathscr{L}^\omega(y, y_T) = \emptyset$. Then $B \parallel T$ is blocking as $B \parallel T \stackrel{r}{\Rightarrow} (x, x_T) \Rightarrow (y, y_T)$.

Conversely, let $s \in \Sigma^*$ such that $(\emptyset, \delta_B^{\det}(Q^\circ, r)) \notin \mathbf{LC}(O, B)$ for every prefix $r \sqsubseteq s$. It is to be shown that $s \in \mathrm{NCONF}(B)$. Consider the deterministic automaton T such that

$$\mathscr{L}(T) = \{ t \in \Sigma^* \mid (\emptyset, \delta_B^{\det}(Q^\circ, r)) \notin \mathbf{LC}(O, B) \text{ for all } r \sqsubseteq t \} . \tag{5.9}$$

T is a well-defined automaton as $\mathscr{L}(T)$ is prefix-closed by construction. It remains to be shown that $B \parallel T$ is nonblocking. Let $B \parallel T \stackrel{t}{\Rightarrow} (x, x_T)$. Then $t \in \mathscr{L}(T)$, and by definition of T (5.9), it holds that $(\emptyset, \delta_B^{\det}(Q^\circ, t)) \notin \mathbf{LC}(O, B)$, and the same holds for all prefixes of t. Also $x \in \delta_B^{\det}(Q^\circ, t)$, so there exists a trace $u \in \Sigma^*$ such that $x \stackrel{u\omega}{\Rightarrow}$, and for every prefix $r \sqsubseteq u\omega$, it holds that $\delta_{O,B}^{\det}(\emptyset, \delta_B^{\det}(Q^\circ, t), r) \notin \mathbf{LC}(O, B)$ (see remark 5.1). By definition (5.9), it follows that $tu\omega \in \mathscr{L}(T)$, and since T is deterministic also $x_T \stackrel{u\omega}{\Rightarrow}$. Therefore, $B \parallel T \stackrel{t}{\Rightarrow} (x, x_T) \stackrel{u\omega}{\Rightarrow}$, i.e., $B \parallel T$ is nonblocking. □

The result of theorem 5.2 shows how less conflicting pairs generalise certain conflicts for the case when two automata are compared, and in combination with the algorithm in section 5.5, less conflicting pairs lead to an alternative presentation of the algorithm [22] to compute the set of certain conflicts.

5.4 Testing the Conflict Preorder

Given the less conflicting pairs for two automata A and B, it is possible to determine whether $A \lesssim_{\mathrm{conf}} B$. Automaton A is less conflicting than B if every test T that is nonconflicting in combination with B also is nonconflicting with A. To check this condition, it is enough to consider traces $B \parallel T \stackrel{s}{\Rightarrow} (x_B, x_T)$, and check whether termination is also possible for every state x_A of A such that $A \parallel T \stackrel{s}{\Rightarrow} (x_A, x_T)$. This amounts to checking whether $(\{x_A\}, X_B) \in \mathbf{LC}(A, B)$ when $A \stackrel{s}{\Rightarrow} x_A$ and $\delta_B^{\det}(Q_B^\circ, s) = X_B$.

However, this condition does not apply to traces of certain conflicts. If $s \in \mathrm{CONF}(B)$, then every test T that can execute s is in conflict with B. In this case, A can still be less conflicting than B, no matter whether A can or cannot execute the trace s and terminate afterwards. This observation leads to the following result.

Theorem 5.3 Let $A = \langle \Sigma, Q_A, \rightarrow_A, Q_A^\circ \rangle$ and $B = \langle \Sigma, Q_B, \rightarrow_B, Q_B^\circ \rangle$ be two automata. A is less conflicting than B if and only if for all $s \in \mathrm{NCONF}(B)$ and all $x_A \in Q_A$ such that $A \stackrel{s}{\Rightarrow} x_A$ it holds that $(\{x_A\}, X_B) \in \mathbf{LC}(A, B)$, where $\delta_B^{\det}(Q_B^\circ, s) = X_B$.

Proof. We will show that it holds that all $s \in \mathrm{NCONF}(B)$ and all $x_A \in Q_A$ such that $A \stackrel{s}{\Rightarrow} x_A$ it holds that $(\{x_A\}, X_B) \in \mathbf{LC}(A, B)$, where $\delta_B^{\det}(Q_B^\circ, s) = X_B$, if and only if it holds that all $c \in \Sigma^*$ and all $x_A \in Q_A$ such that $A \stackrel{c}{\Rightarrow} x_A$ it holds that $(c, \mathscr{L}^\omega(x_A)) \in \mathrm{CC}^\omega(B)$.

First we will show that if all $s \in \mathrm{NCONF}(B)$ and all $x_A \in Q_A$ such that $A \overset{s}{\Rightarrow} x_A$ it holds that $(\{x_A\}, X_B) \in \mathbf{LC}(A,B)$, where $\delta_B^{\det}(Q_B^\circ, s) = X_B$, then it holds that all $c \in \Sigma^*$ and all $x_A \in Q_A$ such that $A \overset{c}{\Rightarrow} x_A$ it holds that $(c, \mathscr{L}^\omega(x_A)) \in \mathrm{CC}^\omega(B)$.

Let $c \in \Sigma^*$ be a trace and $x_A \in Q_A$ be a state such that $A \overset{c}{\Rightarrow} x_A$, we will show that $(c, \mathscr{L}^\omega_(x_A) \in \mathrm{CC}^\omega(B))$.

First let us consider the case where $c \in \mathrm{CONF}(B)$. In this case $(c, \emptyset) \in \mathrm{CC}^\omega(B)$, which in turn implies that $(c, \mathscr{L}^\omega(x_A)) \in \mathrm{CC}^\omega(B)$ as $\emptyset \subseteq \mathscr{L}^\omega(x_A)$.

Second let us consider the case where $c \in \mathrm{NCONF}(B)$. In this case from assumption it holds that $(\{x_A\}, X_B) \in \mathbf{LC}(A,B) \in \mathbf{LC}(A,B)$, where $\delta_B^{\det}(Q_B^\circ, s) = X_B$. From lemma 5.3 it holds that $(\varepsilon, \mathscr{L}^\omega(x_A)) \in \mathrm{CC}^\omega(X_B)$. It then holds from proposition 2.2 that $(c, \mathscr{L}^\omega(x_A)) \in \mathrm{CC}^\omega(B)$ as $c \in \mathrm{NCONF}(B)$ and $\delta^{\det} B \xrightarrow{c} X_B$.

Now let us prove that if for all $c \in \Sigma^*$ and all $x_A \in Q_A$ such that $A \overset{c}{\Rightarrow} x_A$ it holds that $(c, \mathscr{L}^\omega(x_A)) \in \mathrm{CC}^\omega(B)$ it must hold that for all $s \in \mathrm{NCONF}(B)$ and all $x_A \in Q_A$ such that $A \overset{s}{\Rightarrow} x_A$ it holds that $(\{x_A\}, X_B) \in \mathbf{LC}(A,B)$, where $\delta_B^{\det}(Q_B^\circ, s) = X_B$.

Let $s \in \mathrm{NCONF}(B)$ be a trace and x_A be a state such that $A \overset{s}{\Rightarrow} x_A$ we will show that it must hold that $(\{x_A\}, X_B) \in \mathbf{LC}(A,B)$, where $\delta_B^{\det}(Q_B^\circ, s) = X_B$.

As $A \xrightarrow{s} x_A$ it must hold that $(s, \mathscr{L}^\omega(x_A)) \in \mathrm{CC}^\omega(B)$. As $(s, \mathscr{L}^\omega(x_A)) \in \mathrm{CC}^\omega(B)$ and $\delta_B^{\det}(Q_B^\circ, s) = X_B$ from proposition 2.2 it must hold that $(\varepsilon, \mathscr{L}^\omega(x_A)) \in \mathrm{CC}^\omega(X_B)$. Finally from lemma 5.3 as $(\varepsilon, \mathscr{L}^\omega(x_A)) \in \mathrm{CC}^\omega(X_B)$ it must hold that $(X_A, X_B) \in \mathbf{LC}(A,B)$.

□

Example 5.3 Consider again automata A_0 and B_0 in figure 2.2. Recall that $\mathrm{CONF}(B_0) = \alpha\Sigma^*$ from example 2.5, so the only state in A_0 that can be reached by a trace $s \notin \mathrm{CONF}(B_0)$ is a_0. Therefore, it is enough to check the pair $(\{a_0\}, \{b_0\})$ according to theorem 5.3, and it has been shown in example 5.1 that $(\{a_0\}, \{b_0\}) \in \mathbf{LC}^1(A_0, B_0)$. It follows that $A_0 \lesssim_{\mathrm{conf}} B_0$. This conclusion is made despite the fact that $(\{a_0\}, \{b_2\}) \notin \mathbf{LC}(A_0, B_0)$, because $(\{a_0\}, \{b_2\})$ is only reachable by traces $\alpha^n \in \mathrm{CONF}(B_0)$, $n \geq 2$.

When using theorem 5.3 to determine whether an automaton A is less conflicting than some blocking automaton B, the set of certain conflicts of B must be known first. This can be achieved using theorem 5.2, which makes it possible to classify state sets in the subset construction of B as certain conflicts. If a state set $X_B \subseteq Q_B$ is found to represent certain conflicts, i.e., $(\emptyset, X_B) \in \mathbf{LC}(O,B)$ according to theorem 5.2, then $(X_A, X_B) \in \mathbf{LC}(A,B)$ for every state set $X_A \subseteq Q_A$. Successors reached only from such pairs are also certain conflicts of B and should not be considered when testing whether $A \lesssim_{\mathrm{conf}} B$ according to theorem 5.3.

Example 5.4 Consider again automata A_1 and B_1 in figure 2.3. Composing A_1 with a deterministic version of B_1 results in the following four pairs of states in A_1 and sets

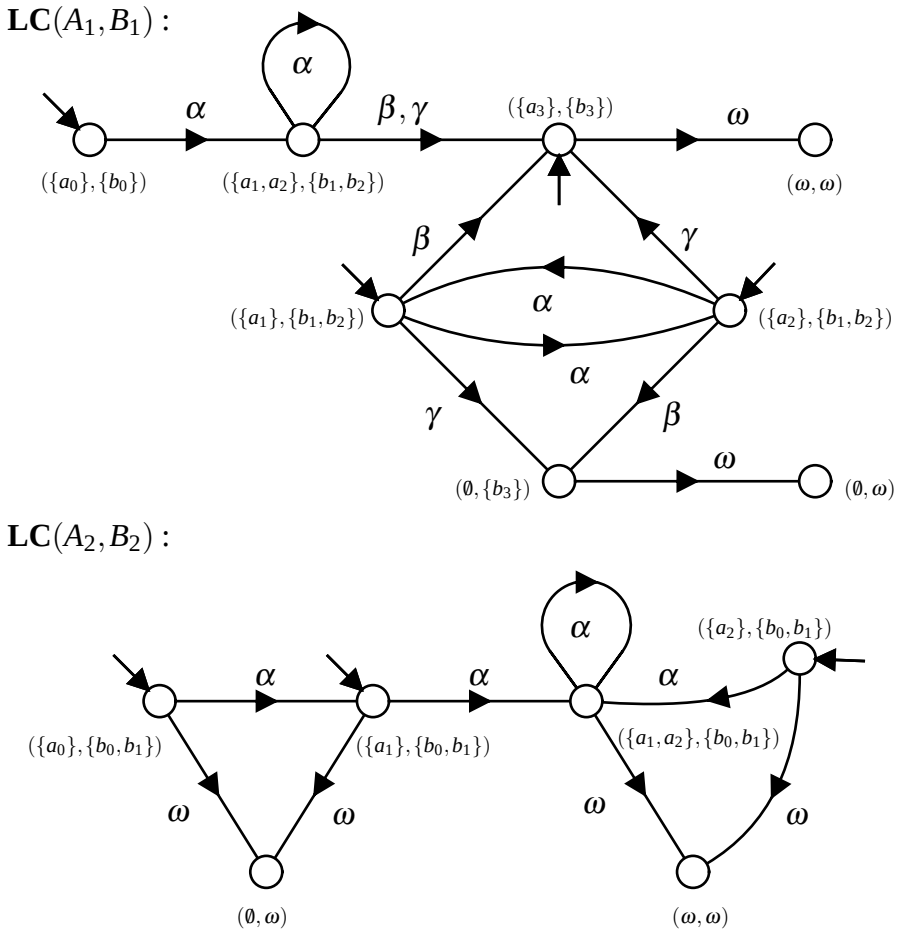


Figure 5.1: Less conflicting pairs for the automata pairs in figure 2.3 and 2.4.

of states in B_1 that should be tested according to theorem 5.3 to determine whether $A_1 \lesssim_{\text{conf}} B_1$:

$$(\{a_0\},\{b_0\}) \quad (\{a_1\},\{b_1,b_2\}) \quad (\{a_2\},\{b_1,b_2\}) \quad (\{a_3\},\{b_3\})\,. \tag{5.10}$$

All four pairs need to be considered as B_1 is nonblocking and thus $\textsc{Conf}(B_1) = \emptyset$.

The graph to the left in figure 5.1 shows these four pairs and their deterministic successors. The four pairs (5.10) are marked as initial states, and the arrows in the graph represent the deterministic transition function. Although the deterministic transition function is defined for all state set pairs and events, arrows to $(\emptyset,\emptyset)$ are suppressed for clarity of presentation.

The following less conflicting pairs to compare A_1 to B_1 are determined from the graph:

$$(\omega,\omega) \in \mathbf{LC}^0(A_1,B_1)\,; \tag{5.11}$$

$$(\{a_0\},\{b_0\}),\ (\{a_1,a_2\},\{b_1,b_2\}),\ (\{a_3\},\{b_3\}) \in \mathbf{LC}^1(A_1,B_1)\,. \tag{5.12}$$

For example, $(\{a_1,a_2\},\{b_1,b_2\}) \in \mathbf{LC}^1(A_1,B_1)$, because all the ways to reach termination from state b_1, i.e., all traces in $\mathscr{L}^\omega(b_1) = \alpha^*\beta\omega$ take the pair $(\{a_1,a_2\},\{b_1,b_2\})$ to $(\omega,\omega) \in LC^0(A_1,B_1)$. No further pairs are found in $\mathbf{LC}^2(A_1,B_1)$, so $\mathbf{LC}(A_1,B_1)$ consists only of the pairs listed above. For example, $(\{a_1\},\{b_1,b_2\}) \notin \mathbf{LC}^2(A_1,B_1)$, because the traces $\alpha\beta\omega \in \mathscr{L}^\omega(b_1)$ and $\gamma\omega \in \mathscr{L}^\omega(b_2)$ do not have any prefixes that reach a pair in $\mathbf{LC}^1(A_1,B_1)$.

As $(\{a_1\},\{b_1,b_2\}) \notin \mathbf{LC}(A_1,B_1)$, it follows from theorem 5.3 that A_1 is *not* less conflicting than B_1.

Example 5.5 Consider again automata A_2 and B_2 in figure 2.4. Note that $\textsc{Conf}(B_2) = \emptyset$. By composing A_2 with a deterministic version of B_2, it becomes clear that the only pairs that need to be tested to determine whether $A_2 \lesssim_{\text{conf}} B_2$ according to theorem 5.3 are $(\{a_0\},\{b_0,b_1\})$ reached after ε, $(\{a_1\},\{b_0,b_1\})$ reached after α^+, and $(\{a_2\},\{b_0,b_1\})$ reached after $\alpha\alpha^+$.

The graph with these pairs and their deterministic successors is shown to the right in figure 5.1, with the three crucial pairs marked as initial. The following less conflicting pairs are discovered (see example 5.2):

$$(\omega,\omega) \in \mathbf{LC}^0(A_2,B_2)\,; \tag{5.13}$$

$$(\{a_1\},\ \{b_0,b_1\}),\ (\{a_1,a_2\},\{b_0,b_1\}),\ (\{a_2\},\{b_0,b_1\}) \in \mathbf{LC}^1(A_2,B_2)\,; \tag{5.14}$$

$$(\{a_0\},\{b_0,b_1\}) \in \mathbf{LC}^2(A_2,B_2)\,. \tag{5.15}$$

As the three crucial pairs are all in $\mathbf{LC}(A_2, B_2)$, it follows from theorem 5.3 that $A_2 \lesssim_{\text{conf}} B_2$.

The result of theorem 5.3 is related to the decision procedure for fair testing [31]. The fair testing decision procedure starts by composing the automaton A with a determinised form of B, which gives rise to the same state set combinations that need to be considered as in theorem 5.3. From this point on, the two methods differ. The fair testing decision procedure annotates each state of the synchronous product of A and the determinised form of B with automata representing the associated refusal trees, and searches for matching automata (or more precisely, for matching *productive sub-automata*) within these annotations. The method based on less conflicting pairs avoids some of the resulting complexity by performing the complete decision on the flat state space of the synchronous product of the determinised forms of A and B.

Another consequence of theorem 5.3 is that if $A \lesssim_{\text{conf}} B$ and the trace $s \in \text{NCONF}(B)$ it must be the case that $A/s \lesssim_{\text{conf}} B/s$.

Proposition 5.1 Let $A = \langle \Sigma, Q_A, \rightarrow_A, Q_A^\circ \rangle$ and $B = \langle \Sigma, Q_B, \rightarrow_B, Q_B^\circ \rangle$ be two automata, such that $A \lesssim_{\text{conf}} B$. Let $s \in \text{NCONF}(B)$. Let $A/s = \langle \Sigma, Q_A, \rightarrow_A, X_A \rangle$ and $B/s = \langle \Sigma, Q_B, \rightarrow_B, X_B \rangle$

Then $A/s \lesssim_{\text{conf}} B/s$

Proof. We will prove that $A/s \lesssim_{\text{conf}} B/s$ using theorem 5.3, by proving that for all $t \in \text{NCONF}(B/s)$ and all $y_A \in Q_A$ such that $A \stackrel{s}{\Rightarrow} x_A$ it holds that $(\{x_A\}, X_B) \in \mathbf{LC}(A, B)$, where $\delta_B^{\text{det}}(Q_B^\circ, s) = X_B$.

Let $t \in \text{NCONF}(B/s)$ be a trace. Let $y_A \in Q_A$ be a state such that $\tilde{A} \stackrel{t}{\Rightarrow} y_A$ and $Y_B \subseteq Q_B$ be a state set such that $X_B \stackrel{t}{\rightarrow} Y_B$. As $\det(B) \stackrel{s}{\rightarrow} X_B \stackrel{t}{\rightarrow} Y_B \rightarrow \omega$, $st \in \text{NCONF}(B)$. Furthermore as $\det(A) \stackrel{s}{\rightarrow} X_A$, $X_A \stackrel{t}{\Rightarrow} y_A$. From theorem 5.3 as $st \in \text{NCONF}(B)$ and $A \stackrel{st}{\rightarrow} y_A$, $(\{y_A\}, Y_B) \in LC$.

Therefore as $A \lesssim_{\text{conf}} B$, $(\{y_A\}, Y_B) \in LC$. □

5.5 Algorithm to Compute Less Conflicting Pairs

This section proposes a method to effectively compute the less conflicting pairs for two given finite-state automata A and B. This is done in a nested iteration. Assuming that the set $\mathbf{LC}^n(A, B)$ is already known, the set $\mathbf{LC}^{n+1}(A, B)$ is computed in a secondary iteration based on *more conflicting triples*.

Definition 5.3 Let $A = \langle \Sigma, Q_A, \rightarrow_A, Q_A^\circ \rangle$ and $B = \langle \Sigma, Q_B, \rightarrow_B, Q_B^\circ \rangle$ be automata. The set $\mathbf{MC}^n(A, B) \subseteq Q_A^{\text{det}} \times Q_B^{\text{det}} \times Q_B$ of n^{th} level *more conflicting triples* for A and B is

defined inductively as follows.

$$\mathbf{MC}_0^n(A,B) = \{\, (\emptyset, \omega, x_B) \mid x_B \in Q_B \,\}\,; \tag{5.16}$$

$$\begin{aligned}\mathbf{MC}_{m+1}^n(A,B) = \{\, &(X_A,X_B,x_B) \mid (X_A,X_B) \notin \mathbf{LC}^n(A,B) \text{ and } x_B \in X_B \text{ and there} \\ &\text{exists a triple } (Y_A,Y_B,y_B) \in \mathbf{MC}_m^n(A,B) \text{ and } \sigma \in \Sigma \text{ such that} \\ &\delta_{A,B}^{\det}(X_A,X_B,\sigma) = (Y_A,Y_B) \text{ and } x_B \overset{\sigma}{\Rightarrow} y_B \,\}\,;\end{aligned} \tag{5.17}$$

$$\mathbf{MC}^n(A,B) = \bigcup_{m\geq 0} \mathbf{MC}_m^n(A,B)\,. \tag{5.18}$$

For a pair (X_A,X_B) to be a less conflicting pair, according to definition 5.2 there must be a state $x_B \in X_B$ such that every trace that takes x_B to termination in B has a prefix that leads to another less conflicting pair. A triple (X_A,X_B,x_B) is considered "more conflicting" if (X_A,X_B) is not yet known to be a less conflicting pair, and the state $x_B \in X_B$ cannot be used to confirm the above property. Therefore, lemma 5.4 shows that a triple (X_A,X_B,x_B) is n^{th}-level "more conflicting" if and only if the state $x_B \in X_B$ can reach termination without passing through a pair in $\mathbf{LC}^n$.

If (X_A,X_B,x_B) is "more conflicting" for all $x_B \in X_B$, then the pair (X_A,X_B) cannot be a less conflicting pair. Otherwise, if there exists at least one state $x_B \in X_B$ such that (X_A,X_B,x_B) is not "more conflicting", then (X_A,X_B) is added to set of less conflicting pairs in the next iteration. Theorem 5.4 below confirms the correctness of this approach.

Lemma 5.4 Let $A = \langle \Sigma, Q_A, \rightarrow_A, Q_A^\circ \rangle$ and $B = \langle \Sigma, Q_B, \rightarrow_B, Q_B^\circ \rangle$ be automata, let $n \in \mathbb{N}$ and $(X_A,X_B,x_B) \in Q_A^{\det} \times Q_B^{\det} \times Q_B$. The following statements are equivalent.

(i) $(X_A,X_B,x_B) \in \mathbf{MC}^n(A,B)$;

(ii) There exists a trace $s \in \Sigma^*\omega \cup \{\varepsilon\}$ such that $\delta_{A,B}^{\det}(X_A,X_B,s) = (\emptyset,\omega)$ and $x_B \overset{s}{\Rightarrow}$, and $\delta_{A,B}^{\det}(X_A,X_B,r) \notin \mathbf{LC}^n(A,B)$ for all prefixes $r \sqsubseteq s$.

Proof. First let $(X_A,X_B,x_B) \in \mathbf{MC}^n(A,B)$, i.e., $(X_A,X_B,x_B) \in \mathbf{MC}_m^n(A,B)$ for some $m \in \mathbb{N}$. It is shown by induction on m that (ii) holds.

In the base case, $m = 0$, and by definition $(X_A,X_B,x_B) \in \mathbf{MC}_0^n(A,B)$ means that $(X_A,X_B) = (\emptyset,\omega)$. Then consider $s = \varepsilon$, and note $\delta_{A,B}^{\det}(X_A,X_B,\varepsilon) = (X_A,X_B) = (\emptyset,\omega)$ and $x_B \overset{\varepsilon}{\Rightarrow}$. Clearly $r \sqsubseteq \varepsilon$ implies $r = \varepsilon$, and $\delta_{A,B}^{\det}(X_A,X_B,\varepsilon) = (\emptyset,\omega) \notin \mathbf{LC}(A,B) \supseteq \mathbf{LC}^n(A,B)$ by lemma 5.2.

Now consider $(X_A,X_B,x_B) \in \mathbf{MC}_{m+1}^n(A,B)$. It follows from definition 5.3 that $(X_A,X_B) \notin LC^n(A,B)$ and $x_B \in X_B$, and there exists $(Y_A,Y_B,y_B) \in \mathbf{MC}_m^n(A,B)$ and $\sigma \in \Sigma$ such that $\delta_{A,B}^{\det}(X_A,X_B,\sigma) = (Y_A,Y_B)$ and $x_B \overset{\sigma}{\Rightarrow} y_B$. By inductive assumption, there exists a trace $s \in \Sigma^*\omega \cup \{\varepsilon\}$ such that $\delta_{A,B}^{\det}(Y_A,Y_B,s) = (\emptyset,\omega)$ and $y_B \overset{s}{\Rightarrow}$, and for all $r \sqsubseteq s$ it

holds that $\delta^{\det}_{A,B}(Y_A, Y_B, r) \notin \mathbf{LC}^n(A,B)$. Then $\delta^{\det}_{A,B}(X_A, X_B, \sigma s) = \delta^{\det}_{A,B}(Y_A, Y_B, s) = (\emptyset, \omega)$ and $x_B \overset{\sigma}{\Rightarrow} y_B \overset{s}{\Rightarrow}$, and for all $r \sqsubseteq \sigma s$ it holds that $\delta^{\det}_{A,B}(X_A, X_B, r) \notin \mathbf{LC}^n(A,B)$.

Conversely, let $s \in \Sigma^*\omega \cup \{\varepsilon\}$ such that (ii) holds. This means that $\delta^{\det}_{A,B}(X_A, X_B, s) = (\emptyset, \omega)$ and $x_B \overset{s}{\Rightarrow}$, and $\delta^{\det}_{A,B}(X_A, X_B, r) \notin \mathbf{LC}^n(A,B)$ for all $r \sqsubseteq s$. It is shown by induction on $m = |s|$ that $(X_A, X_B, x_B) \in \mathbf{MC}^n_m(A,B)$.

In the base case, where $m = 0$ and $s = \varepsilon$, it holds by definition that $(X_A, X_B) = \delta^{\det}_{A,B}(X_A, X_B, \varepsilon) = (\emptyset, \omega) \in \mathbf{MC}^n_0(A,B)$.

Now let $s = \sigma t$ such that $|t| = m$, and $\delta^{\det}_{A,B}(X_A, X_B, s) = (\emptyset, \omega)$ and $x_B \overset{s}{\Rightarrow}$, and $\delta^{\det}_{A,B}(X_A, X_B, r) \notin \mathbf{LC}^n(A,B)$ for all prefixes $r \sqsubseteq s$. Write $\delta^{\det}_{A,B}(X_A, X_B, \sigma) = (Y_A, Y_B)$ and $x_B \overset{\sigma}{\Rightarrow} y_B \overset{t}{\Rightarrow}$. Then $y_B \overset{t}{\Rightarrow}$ and $\delta^{\det}_{A,B}(Y_A, Y_B, t) = \delta^{\det}_{A,B}(X_A, X_B, \sigma t) = \delta^{\det}_{A,B}(X_A, X_B, s) = (\emptyset, \omega)$ and $\delta^{\det}_{A,B}(Y_A, Y_B, r) \notin \mathbf{LC}^n(A,B)$ for all $r \sqsubseteq t$. Then $(Y_A, Y_B, y_B) \in \mathbf{MC}^n_m(A,B)$ by inductive assumption, and by definition 5.3 it follows that $(X_A, X_B, x_B) \in \mathbf{MC}^n_{m+1}(A,B)$. □

Theorem 5.4 Let $A = \langle \Sigma, Q_A, \to_A, Q^\circ_A \rangle$ and $B = \langle \Sigma, Q_B, \to_B, Q^\circ_B \rangle$ be automata, and let $n \in \mathbb{N}$. Then

$$\mathbf{LC}^{n+1}(A,B) = \{\, (X_A, X_B) \in Q^{\det}_A \times Q^{\det}_B \mid (X_A, X_B, x_B) \notin \mathbf{MC}^n(A,B) \text{ for some } x_B \in X_B \,\}. \tag{5.19}$$

Proof. Let $(X_A, X_B) \in LC^{n+1}(A,B)$. Then by definition 5.2, there exists $x_B \in X_B$ such that for all $t \in \Sigma^*$ such that $x_B \overset{t\omega}{\Rightarrow}$, there exists $r \sqsubseteq t\omega$ such that $\delta^{\det}_{A,B}(X_A, X_B, r) \in \mathbf{LC}^i(A,B)$ for some $i \leq n$. Equivalently, this means that if there does not exist a trace $t \in \Sigma^*$ such that $x_B \overset{t\omega}{\Rightarrow}$ and for all prefixes $r \sqsubseteq t\omega$ it holds that $\delta^{\det}_{A,B}(X_A, X_B, r) \notin \mathbf{LC}^n(A,B)$. Then $(X_A, X_B, x_B) \notin \mathbf{MC}^n(A,B)$ because otherwise such a trace would exist by lemma 5.4.

Conversely, let $x_B \in X_B$ such that $(X_A, X_B, x_B) \notin \mathbf{MC}^n(A,B)$. To check the condition in definition 5.2 (5.5), consider $t \in \Sigma^*$ such that $x_B \overset{t\omega}{\Rightarrow}$. Then clearly $\delta^{\det}_B(X_B, t\omega) = \omega$. By definition 5.1, it holds that either $\delta^{\det}_A(X_A, t\omega) = \omega$ or $\delta^{\det}_A(X_A, t\omega) = \emptyset$. If $\delta^{\det}_A(X_A, t\omega) = \omega$, then $\delta^{\det}_{A,B}(X_A, X_B, t\omega) = (\omega, \omega) \in \mathbf{LC}^0(A,B)$. Otherwise $\delta^{\det}_A(X_A, t\omega) = \emptyset$ and thus $\delta^{\det}_{A,B}(X_A, X_B, t\omega) = (\emptyset, \omega)$, and by lemma 5.4 there must exist $r \sqsubseteq t\omega$ such that $\delta^{\det}_{A,B}(X_A, X_B, r) \in \mathbf{LC}^n(A,B)$ as otherwise $(X_A, X_B, x_B) \in \mathbf{MC}^n(A,B)$. In both cases, $\delta^{\det}_{A,B}(X_A, X_B, r) \in \mathbf{LC}^i(A,B)$ for some $r \sqsubseteq t\omega$ and $i \leq n$. Since $t \in \Sigma^*$ with $x_B \overset{t\omega}{\Rightarrow}$ was chosen arbitrarily, it follows from definition 5.2 (5.5) that $(X_A, X_B) \in \mathbf{LC}^{n+1}(A,B)$. □

Example 5.6 Figure 5.2 shows a graph representing the more conflicting triples to check whether $A_2 \lesssim_{\text{conf}} B_2$ in figure 2.4. The arrows in the graph represent the deterministic transition function in combination with the transition relation of B_2. An arrow $(X_A, X_B, x_B) \overset{\sigma}{\to} (Y_A, Y_B, y_B)$ indicates that $\delta^{\det}_{A_2,B_2}(X_A, X_B, \sigma) = (Y_A, Y_B)$ and $x_B \overset{\sigma}{\Rightarrow} y_B$.

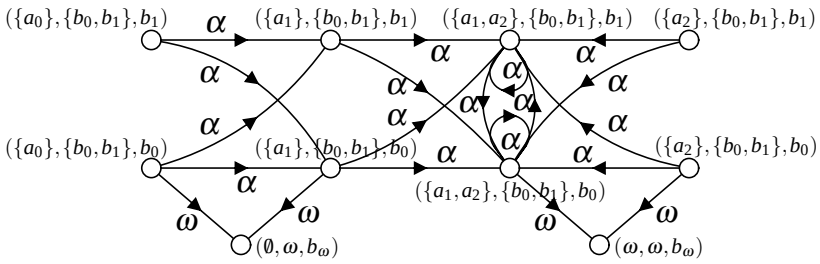


Figure 5.2: Calculating more conflicting triples for automata A_2 and B_2 in figure 2.4.

In the first iteration to compute $\mathbf{MC}^0(A_2,B_2)$, first the triple $(\emptyset,\omega,b_\omega)$ is added to $\mathbf{MC}^0_0(A_2,B_2)$. Next, the triples $(\{a_0\},\{b_0,b_1\},b_0)$ and $(\{a_1\},\{b_0,b_1\},b_0)$ are added to $\mathbf{MC}^0_1(A_2,B_2)$ as they can immediately reach $(\emptyset,\omega,b_\omega)$. Finally, $(\{a_0\},\{b_0,b_1\},b_1)$ is also added to $\mathbf{MC}^0_2(A_2,B_2)$ as it can reach the triple $(\{a_1\},\{b_0,b_1\},b_0)\in\mathbf{MC}^0_1(A_2,B_2)$. No further triples are found to be in $\mathbf{MC}^0_3(A_2,B_2)$. Therefore, $(\{a_1\},\{b_0,b_1\},b_1)\notin\mathbf{MC}^0(A_2,B_2)$, so it follows from theorem 5.4 that $(\{a_1\},\{b_0,b_1\})\in\mathbf{LC}^1(A_2,B_2)$, and likewise $(\{a_1,a_2\},\{b_0,b_1\}),(\{a_2\},\{b_0,b_1\})\in\mathbf{LC}^1(A_2,B_2)$.

In the next iteration to compute $\mathbf{MC}^1(A_2,B_2)$, we note that $(\{a_1\},\{b_0,b_1\},b_0)\notin\mathbf{MC}^1_1(A_2,B_2)$ as $(\{a_1\},\{b_0,b_1\})\in\mathbf{LC}^1(A_2,B_2)$. $(\{a_0\},\{b_0,b_1\},b_0)\in\mathbf{MC}^1_1(A_2,B_2)$ because of the transition to $(\emptyset,\omega,b_\omega)\in\mathbf{MC}^1_0(A_2,B_2)$, but now $(\{a_0\},\{b_0,b_1\},b_1)\notin\mathbf{MC}^1_2(A_2,B_2)$ because $(\{a_1\},\{b_0,b_1\},b_0)\notin\mathbf{MC}^1_1(A_2,B_2)$. Accordingly, the pair $(\{a_0\},\{b_0,b_1\})$ is added to $\mathbf{LC}^2(A_2,B_2)$.

In a final iteration to compute $\mathbf{MC}^2(A_2,B_2)$, only one more conflicting triple is found, $(\emptyset,\omega,b_\omega)\in\mathbf{MC}^2_0(A_2,B_2)$. No further pairs are added in $\mathbf{LC}^3(A_2,B_2)$. At this point, the iteration terminates, having found exactly the four less conflicting pairs given in example 5.5, (5.14) and (5.15).

To determine whether an automaton A is less conflicting than an automaton B, we first needed to determine the set of certain conflicts of B, and then find all the state-set pairs for A and B that are reachable from a pair like $(\{x_A\},X_B)$ associated with some trace that is not a certain conflict of B. The more conflicting triples can be constructed as they are discovered during the backwards search from the terminal states.

The complexity of each iteration of the more conflicting triples computation is determined by the number of arrows in the graph, which is bounded by $|\Sigma|\cdot|Q_B|^2\cdot 2^{|Q_A|}\cdot 2^{|Q_B|}$, because the powerset transitions are deterministic, which is not the case for the transitions of B. Each iteration except the last adds at least one less conflicting pair, so the number of iterations is bounded by $2^{|Q_A|}\cdot 2^{|Q_B|}$. The complexity of this loop dominates all other tasks of the computation. Therefore, the worst-case time complexity to

Algorithm 1 Construct Deterministic State Space

1: $Stack \leftarrow \{(Q_A^\circ, Q_B^\circ), (\emptyset, Q_B^\circ)\}$
2: $Pairs \leftarrow \{(Q_A^\circ, Q_B^\circ), (\emptyset, Q_B^\circ)\}$
3: **while** $Stack \neq \emptyset$ **do**
4: $\quad (X_A, X_B) \leftarrow Stack.pop()$
5: $\quad$ **for all** $\sigma \in \Sigma \cup \{\omega\}$ **do**
6: $\quad\quad (Y_A, Y_B) \leftarrow \delta_{A,B}^{\det}(X_A, X_B, \sigma)$
7: $\quad\quad$ **if** $(Y_A, Y_B) \notin Pairs$ **then**
8: $\quad\quad\quad Pairs \leftarrow Pairs \cup \{(Y_A, Y_B)\}$
9: $\quad\quad\quad Stack.push((Y_A, Y_B))$
10: $\quad\quad$ **end if**
11: $\quad$ **end for**
12: $\quad$ **for all** $x_A \in X_A$ **do**
13: $\quad\quad$ **if** $(\{x_A\}, X_B) \notin Pairs$ **then**
14: $\quad\quad\quad Pairs \leftarrow Pairs \cup \{(\{x_A\}, X_B)\}$
15: $\quad\quad\quad Stack.push((\{x_A\}, X_B))$
16: $\quad\quad$ **end if**
17: $\quad$ **end for**
18: **end while**

determine whether $A \lesssim_{\text{conf}} B$ using less conflicting pairs is

$$O(|\Sigma| \cdot |Q_B|^2 \cdot 4^{|Q_A|} \cdot 4^{|Q_B|}) \;=\; O(|\Sigma| \cdot |Q_B|^2 \cdot 2^{2|Q_A|+2|Q_B|}) \,. \tag{5.20}$$

This shows that the conflict preorder can be tested in linear exponential time, as is the case for the fair testing preorder. Yet, the complexity (5.20) is better than the time complexity of the decision procedure for fair testing, which is $O(|Q_A| \cdot |Q_B| \cdot 2^{3|Q_A|+5|Q_B|})$ [31].

5.6 Implementation

To determine for two automata A and B whether $A \lesssim_{\text{conf}} B$, the implementation performs three steps, presented as separate algorithms. First Algorithm 1 computes the set of reachable state-set pairs, second Algorithm 2 determines which of these pairs are less conflicting pairs, and third Algorithm 3 examines the computed pairs to determine whether $A \lesssim_{\text{conf}} B$ based on theorem 5.3.

In the first step, given two automata $A = \langle \Sigma, Q_A, \rightarrow_A, Q_A^\circ \rangle$ and $B = \langle \Sigma, Q_B, \rightarrow_B, Q_B^\circ \rangle$, Algorithm 1 performs a depth-first search to collect the set *Pairs* of all reachable state-set pairs $(X_A, X_B) \in Q_A^{\det} \times Q_B^{\det}$, using a *Stack* of pairs yet to be expanded. The search begins with the initial state-set pair (Q_A°, Q_B°) and with $(\emptyset, Q_B^\circ)$, in order to calculate both the composed deterministic state space of A and B and the set of certain conflicts

of B according to theorem 5.2. For each state-set pair (X_A, X_B), the loop in lines 5–11 finds all successors and adds them into the set of pairs. In addition, the loop in lines 12–17 adds the pairs $(\{x_A\}, X_B)$ for each $x_A \in X_A$. This is done because these pairs have to be checked for containment in $\mathbf{LC}(A,B)$ according to theorem 5.3.

Algorithm 1 constructs the state-set pairs to detect the set of certain conflicts and to test the less conflicting condition in one iteration. For blocking automata, it may be more efficient to discover all certain conflicts first and use them to prune the search for the remaining pairs.

In the second step, Algorithm 2 calculates the set $\mathbf{LC}(A,B)$ of less conflicting pairs for A and B, using more conflicting triples as described in section 5.5.

The loop in lines 2–6 collects all pairs in $\mathbf{LC}^0(A,B)$ according to definition 5.2 (5.4) and adds them to the set LC. Then the loop starting in line 7 adds to this the pairs in the next level $\mathbf{LC}^n(A,B)$ by collecting the corresponding set of more conflicting triples. According to definition 5.3 (5.16), this iteration starts with the triples in $\mathbf{MC}_0^n(A,B) = \{\,(\emptyset, \omega, x_B) \mid x_B \in Q_B\,\}$, which can be restricted to states reached by ω as no other transitions lead to $\omega \in Q_B^{\det}$. Then the algorithm looks backward to visit the predecessors of each triple (X_A, X_B, x_B). To find pairs (Y_A, Y_B) such that $\delta_{A,B}^{\det}(Y_A, Y_B, \sigma) = (X_A, X_B)$ in line 13 efficiently, it is advisable to remember the backwards transition relation during the construction of the state-space in Algorithm 1.

Finally, after all the more conflicting triples for the current level n are found, the loop in lines 24–31 adds the new less conflicting pairs for $LC^{n+1}(A,B)$ to the set LC. According to theorem 5.4, this is done by checking each pair (X_A, X_B) if there is any triple $(X_A, X_B, x_B) \notin \mathbf{MC}_0^n(A,B)$ with $x_B \in X_B$. If a new less conflicting pair is discovered during this iteration, line 28 ensures that the main loop starting in line 7 is executed once more to check for less conflicting pairs of the next level.

Lastly, Algorithm 3 is invoked to determine whether $A \lesssim_{\text{conf}} B$ based on theorem 5.3. The reachable state-set pairs are explored a second time to see if the relevant pairs are in fact less conflicting pairs. In line 5, the search stops when encountering a pair (X_A, X_B) with $X_A = \emptyset$, as such pairs cannot lead to a pair $(\{x_A\}, X_B)$. And when $(\emptyset, X_B) \in \mathbf{LC}(A,B)$, then X_B and its successors represent certain conflicts according to theorem 5.2, so according to theorem 5.3, these pairs are not explored further either. For the remaining pairs (X_A, X_B), the loop in lines 6–10 checks for states $x_A \in X_A$ such that $(\{x_A\}, X_B)$ is not a less conflicting pair—if such a pair exists then A cannot be less conflicting than B according to theorem 5.3. If no such pair exists, the loop in lines 11–17 proceeds to visit the successors. If no relevant pair $(\{x_A\}, X_B) \notin \mathbf{LC}(A,B)$ can be found after visiting all reachable state-set pairs, the algorithm terminates and reports that $A \lesssim_{\text{conf}} B$.

Algorithm 2 Collect **LC**-pairs

$LC \leftarrow \emptyset$
for all $(X_A, X_B) \in Pairs$ **do**
  if $X_A = \{\omega\}$ or X_B contains a blocking state **then**
    $LC \leftarrow LC \cup \{(X_A, X_B)\}$
  end if
end for
repeat
  $Stack \leftarrow \{\, (\emptyset, \{\omega\}, x_B) \mid B \xrightarrow{t\omega} x_B \,\}$
  $MC \leftarrow \{\, (\emptyset, \{\omega\}, x_B) \mid B \xrightarrow{t\omega} x_B \,\}$
  while $Stack \neq \emptyset$ **do**
    $(X_A, X_B, x_B) \leftarrow Stack.pop()$
    for all $\sigma \in \Sigma$ **do**
      for all $(Y_A, Y_B) \in Pairs \setminus LC$ such that $\delta^{\det}_{A,B}(Y_A, Y_B, \sigma) = (X_A, X_B)$ **do**
        for all $y_B \in Y_B$ such that $x_B \xrightarrow{\sigma} y_B$ **do**
          if $(Y_A, Y_B, y_B) \notin MC$ **then**
            $MC \leftarrow MC \cup \{(Y_A, Y_B, y_B)\}$
            $Stack.push((Y_A, Y_B, y_B))$
          end if
        end for
      end for
    end for
  end while
  $unchanged \leftarrow true$
  for all $(X_A, X_B) \in Pairs$ **do**
    for all $x_B \in X_B$ **do**
      if $(X_A, X_B, x_B) \notin MC$ **then**
        $LC \leftarrow LC \cup \{(X_A, X_B)\}$
        $unchanged \leftarrow false$
      end if
    end for
  end for
until *unchanged*

Algorithm 3 Check for Less Conflicting

```
1: Stack ← {(Q°_A, Q°_B)}
2: Pairs ← {(Q°_A, Q°_B)}
3: while Stack ≠ ∅ do
4:    (X_A, X_B) ← Stack.pop()
5:    if X_A ≠ ∅ and (∅, X_B) ∉ LC then
6:       for all x_A ∈ X_A do
7:          if ({x_A}, X_B) ∉ LC then
8:             return false
9:          end if
10:      end for
11:      for all σ ∈ Σ ∪ {ω} do
12:         (Y_A, Y_B) ← δ^det_{A,B}(X_A, X_B, σ)
13:         if (Y_A, Y_B) ∉ Pairs then
14:            Pairs ← Pairs ∪ {(Y_A, Y_B)}
15:            Stack.push((Y_A, Y_B))
16:         end if
17:      end for
18:   end if
19: end while
20: return true
```

5.7 Experimental Results

The algorithm to test the conflict preorder has been tested on pairs of moderately large automata obtained during attempts at compositional nonblocking verification of discrete event systems models of industrial applications [12]. The results are summarised in Table 5.1.

The first six test cases are the checks needed to verify the dining philosophers example as discussed in example 2.7. The other automata pairs have been obtained during compositional verification of a manufacturing system model using abstraction [12, 35]. Each test case seeks to compare an automaton constructed during compositional verification to a conflict equivalent abstraction that was computed automatically. Some abstractions have been modified to produce test cases where the conflict preorder is not satisfied.

Table 5.1 shows for each test case the number of states of each of the two automata composed (**States**), the number of reachable state-set pairs in the combined deterministic state space (**Pairs**), the largest number of more conflicting triples constructed during the iterations of Algorithm 2 (**Triples**), and the number of less conflicting pairs (**LC-Pairs**). The number in column $|\mathbf{LC}^k|$ indicates the number of new pairs discovered at level k, not the total number of pairs at that level. Furthermore, the table displays the execution time (**Time**) of each test, and whether or not the conflict preorder is satis-

fied (**Res.**). All experiments were run on a standard laptop computer using a single 2.3 GHz CPU and 3.8 GB of RAM.

Despite the exponential complexity of the algorithm, all test cases except one have been solved successfully by the implementation, which processes automata with a few thousand states in a matter of seconds. The level of less conflicting pairs, which also has an exponential worst-case, does not exceed four in any test. These results suggest that the conflict preorder can be tested in a reasonable time for moderately large automata derived from practical applications.

Table 5.1: Experimental results

Instance	**States**		**Pairs**	**Triples**	**LC-Pairs**					**Time**	**Res.**
$A \lesssim_{\text{conf}} B$	$\lvert Q_A\rvert$	$\lvert Q_B\rvert$	$\lvert Pairs\rvert$	$\lvert MC\rvert$	$\lvert\mathbf{LC}^0\rvert$	$\lvert\mathbf{LC}^1\rvert$	$\lvert\mathbf{LC}^2\rvert$	$\lvert\mathbf{LC}^3\rvert$	$\lvert\mathbf{LC}^4\rvert$	[s]	$\lesssim_{\text{conf}}$
$\mathbf{S}_1 \lesssim_{\text{conf}} \mathbf{S}'_1$	52	12	198	13	2	90	0	0	0	0.10	*true*
$\mathbf{S}'_1 \lesssim_{\text{conf}} \mathbf{S}_1$	12	52	80	149	1	25	0	0	0	0.14	*true*
$\mathbf{S}_{1,2} \lesssim_{\text{conf}} \mathbf{S}'_{1,2}$	90	13	483	65	2	193	22	0	0	0.19	*true*
$\mathbf{S}'_{1,2} \lesssim_{\text{conf}} \mathbf{S}_{1,2}$	13	90	200	1055	1	44	0	0	0	0.33	*true*
$\mathbf{S}_{1,2,3} \lesssim_{\text{conf}} \mathbf{S}'_{1,2,3}$	100	13	529	53	2	236	14	0	0	0.19	*true*
$\mathbf{S}'_{1,2,3} \lesssim_{\text{conf}} \mathbf{S}_{1,2,3}$	13	100	154	1054	1	34	0	0	0	0.36	*true*
1. a)	126	34	280	38	11	52	0	0	0	0.08	*true*
b)	34	126	183	57	87	50	0	0	0	0.04	*true*
2. a)	102	33	224	33	6	47	0	0	0	0.01	*true*
b)	33	102	143	120	15	37	0	0	0	0.02	*true*
3. a)	624	615	7800	17618	13	5584	47	0	0	2.60	*true*
b)	615	624	3402	9202	17	2813	0	0	0	1.17	*true*
4. a)	1141	1048	17318	42876	205	11173	906	56	0	5.96	*true*
b)	1048	1141	17625	45705	169	11654	990	0	0	6.09	*true*
5. a)	679	431	3538	1362	52	1561	222	0	0	0.24	*true*
b)	431	679	1633	1795	59	784	0	0	0	0.13	*true*
6. a)	165	153	1293	686	2	764	0	0	0	0.15	*true*
b)	153	165	871	888	1	601	0	0	0	0.12	*true*
7. a)	306	255	4145	3951	2	2809	128	0	0	0.69	*true*
b)	255	306	1889	4522	1	1500	0	0	0	0.38	*true*
8. a)	808	598	23169	16195	2	19681	0	0	0	4.42	*true*
b)	598	808	11369	22677	2	8845	0	0	0	3.11	*true*
9. a)	3853	78	55537	10333	2	33231	198	50	46	32.55	*true*
b)	78	3853	4003	>440000	1	out of memory					
10. a)	8304	5927	70214	28269	2450	40459	15	0	0	7.38	*true*
b)	5927	8304	37140	37243	2029	24313	0	0	0	5.40	*true*
11. a)	1773	1766	5976	3325	328	3734	0	0	0	0.59	*true*
b)	1766	1773	5956	3309	335	3720	0	0	0	0.45	*true*
12. a)	498	487	2777	1756	2	1367	70	0	0	0.15	*true*
b)	487	498	2998	1677	2	1583	0	0	0	0.14	*true*
13. a)	424	392	2176	818	93	986	2	0	0	0.21	*true*
b)	392	424	1341	809	87	822	810	0	0	0.15	*true*
14. a)	385	231	13487	44240	2	9253	389	0	0	6.54	*false*
b)	231	385	5884	33618	2	6382	0	0	0	4.73	*true*
15. a)	620	455	4978	9875	14	1774	1	0	0	2.47	*false*
b)	455	620	3205	5408	19	5389	0	0	0	0.77	*true*
16. a)	120	49	1156	750	284	343	0	0	0	0.12	*false*
b)	49	120	715	715	268	244	0	0	0	0.59	*true*
17. a)	306	169	5119	5520	2	2113	0	0	0	2.28	*false*
b)	169	306	1770	2547	2	949	60	0	0	0.34	*true*

Chapter 6

Conflict Normal Form

Two automata A and B are defined as being conflict equivalent, if when compared with an arbitrary test automaton T, $A \parallel T$ is nonblocking if and only if $B \parallel T$ is also nonblocking. Two automata are conflict equivalent if and only if they are conflicting in exactly the same situations. This describes conflict equivalence in terms of how two conflict equivalent automata can be used. It does not however describe what in the two automata's structure causes the automata A and B to be conflict equivalent.

Another description of conflict equivalence is the nonconflicting completion semantics. This describes conflict equivalence in terms of the traces which the test automaton T must be capable of performing in order for $A \parallel T$ to be nonblocking. The limitation of this method is that there can be a potentially infinite number of nonconflicting completions associated with any given automaton.

The previous chapter describes how to determine whether or not a given language L is in fact a nonconflicting completion of the automaton A. The chapter further shows that for every finite-state automaton B there is a finite set of languages which need to be compared for inclusion in the set of nonconflicting completions of A, in order to determine whether B is less conflicting that A.

This chapter builds upon this and shows that we can find a finite, minimal set of nonconflicting continuations of the automaton A. We further show that this minimal set of continuations uniquely characterises the nonconflicting continuations of A. This means that the minimal set of continuations characterises all the potential conflicts in A, and only the potential conflicts in A. In addition if an automaton B is conflict equivalent with A, it will have an identical minimal set of nonconflicting continuations to the automaton A. This minimal set will be characterised as the conflict normal form, which is comprised of a *trunk* automaton, paired with a set of nonconflicting requirements based upon that *trunk*. This trunk and its nonconflicting requirements minimally represent the nonconflicting continuations of A.

Figure 6.1 shows an example of an automaton A and its resulting conflict normal form, while showing intermediate steps. A is a nondeterministic automaton. The trunk automaton $trunk(A)$ of the automaton A is constructed as a deterministic recogniser of A's language, with the exception that all states in $trunk(A)$ are marked. Once the trunk automaton is constructed, a set of nonconflicting requirements of A is also constructed. For example the state $(\{1,2\},\mathscr{L}^{\omega}(1))$ represents that any test automaton which can reach the state $\{1,2\}$ must be capable of performing at least one trace in $\mathscr{L}^{\omega}(1)$. This is represented by the state $\{1,2\}$ being linked to $(\{1,2\},\mathscr{L}^{\omega}(1))$ by a τ transition. This set is capable of characterising the nonconflicting completions of A but is not unique, this is the initial conflict normal form of A. A series of refinement steps is then applied to this nonconflicting requirement set until it is considered irreducible. This irreducible set combined with $trunk(A)$ uniquely characterises the automaton A's nonconflicting completion and is the conflict normal form of A.

This chapter is divided into sections. In the section 6.1 we introduce requirements, requirement sets, requirement automata, and all the notation surrounding them. In section 6.2 we show how to construct the trunk automaton, and the initial requirement set for a given automaton. In section 6.3 we show how a requirement set can be refined into a simpler equivalent requirement set. In section 6.4 we show that the refinements in section three can be used to find an irreducible requirement set which is unique for any given conflict equivalence class. This is combined with the results from section 6.2 to show that the trunk and unique requirement set are canonical.

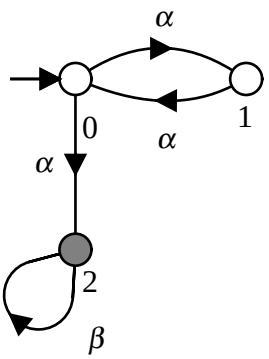
6.1 Notation

6.1.1 Nonconflicting Requirements

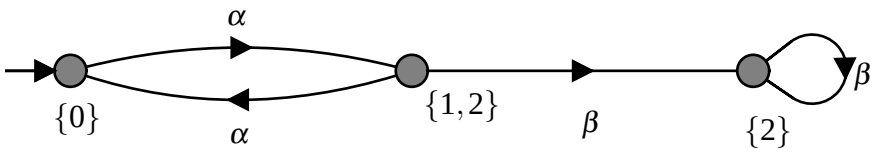
In this subsection we introduce the concept of nonconflicting requirements. We first define what a nonconflicting requirement is, next we define how different requirements relate to one another over $\rightarrow$, we then define how requirements are grouped together, finally we show how requirements can be compared to one another.

A nonconflicting requirement is a pair of state and language. Each nonconflicting requirement (x,L) is used to represent a set of nonconflicting continuations, where L represents the continuation and the state x represents the set of traces $\{s \in \Sigma^* | G \xrightarrow{s} x\}$. The trunk automaton G is the automaton which is used to define the set of states which reach x. All the nonconflicting requirements for a given normal form will always have the same trunk.

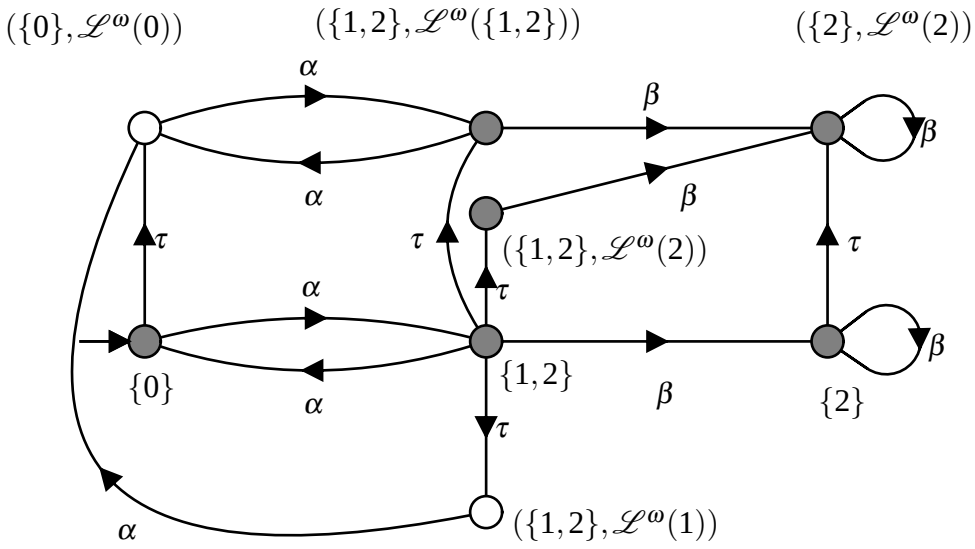
We place various restrictions on what can be a nonconflicting requirement in order to prevent obviously redundant requirements from being allowed. The first is that the



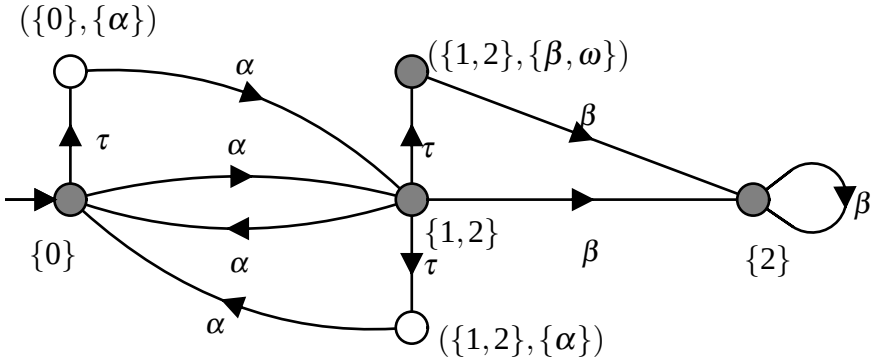
A



trunk(*A*)



Initial requirement automaton of *A*



Conflict Normal Form of *A*

Figure 6.1: Example of an automaton and its Conflict Normal Form

language of the requirement must be prefix-free. A language L is a prefix-free language if for every trace s contained in L, L contains no prefix of s.

Definition 6.1 Let $L \in \Sigma^*\omega$ be a language.

L is a prefix-free language if and only if for all $s,t \in L$, $s \sqsubseteq t$ implies $s = t$.

Example 6.1 Let $L = \{\alpha, \alpha\beta, \beta\}$ be a language. This language is not prefix-free. This is because both α and $\alpha\beta$ are elements of L, while $\alpha \sqsubseteq \alpha\beta$. Because L is not prefix-free, if L is being used to represent a nonconflicting continuation we can immediately state L contains redundant traces. This is because a nonconflicting continuation is satisfied as long as any trace in L is accepted by the test automaton. If the test automaton T can perform $\alpha\beta$ that automatically implies that T can perform α. Therefore whenever the trace $\alpha\beta$ could be used to satisfy L, the trace α can be used instead. This is because α is a prefix of $\alpha\beta$.

Let $M = \{\alpha, \beta\}$ be a language. This language is prefix-free.

We further define the function prune such that for any given language L, we can find its appropriate prefix-free language.

Definition 6.2 Let L be a language.

$prune(L) = \{s \in L | \forall t \in L \text{ if } t \sqsubseteq s \text{ then } s = t\}$

Example 6.2 Consider the languages L and M from the previous example $prune(L) = M$. This is because α is a prefix of $\alpha\beta$, thus $\alpha\beta$ is pruned back to the event α.

A useful property of the prune relation which will be used later on in this chapter is that prune is commutative with language derivation.

Lemma 6.1 Let L be a language. Let $s \in \overline{prune(L)}$.

$prune(L)/s = prune(L/s)$.

Proof. First we will prove that $prune(L)/s \subseteq prune(L/s)$. Let $t \in prune(L)/s$ be a trace, by definition 2.3 it is the case that $st \in prune(L)$. Therefore $st \in L$ and for every trace $p \in L$ such that $p \sqsubseteq st$ it holds that $p = st$. Therefore $t \in L/s$, and for every trace $p \in L/s$ such that $p \sqsubseteq t$ it holds that $p = t$, therefore t must be a trace in $prune(L/s)$.

Next we will prove that $prune(L/s) \subseteq prune(L)/s$. Let $t \in prune(L/s)$, by definition $t \in L/s$ and for every trace $p \in L/s$ such that $p \sqsubseteq t$ it holds that $p = t$, furthermore $st \in L$. As $s \in \overline{prune(L)}$ by assumption and $prune(L)$ is prefix-free for every trace $p \in prune(L)$ such that $p \sqsubseteq s$ it holds that $p = s$, therefore there exists no $p \in L$ such that $p \sqsubset s \sqsubseteq st$, furthermore as there exists no trace $p \in L/s$ such that $p \sqsubseteq t$, there exists no trace $sp \in L$ such that $sp \sqsubseteq st$ therefore $st \in prune(L)$, and consequently $t \in prune(L)/s$. □

Each requirement pair (x,L) must satisfy several conditions in order for us to consider (x,L) to be well formed. Firstly as has already been stated the language L must be prefix-free. This is because as has been previously shown languages which are not prefix-free have traces which are trivially redundant. Second $L = \emptyset$ if and only if $\mathscr{L}^\omega(x) = \emptyset$, in this case we don't wish to use (x,L) to express certain conflicts. This is because certain conflicts can be more easily and better dealt with outside the refinement process for nonconflicting requirements. We also require that $L \subseteq \mathscr{L}^\omega(x)$, this has a dual purpose. We restrict L to traces which are in $L(x)$ because we do not wish requirements to be capable of performing traces which the original automaton cannot. We further restrict L to traces in $\mathscr{L}^\omega(x)$ because it is nonsensical to require a test automaton to be capable of reaching a blocking state, in order to not block. Finally we require that $\varepsilon \notin L$ as such a requirement is redundant.

Definition 6.3 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be a deterministic automaton. Let x be a state. Let L be a language.

(x,L) is a requirement pair if and only if (x,L) fulfills the following properties

(i) L is prefix-free.

(ii) $L = \emptyset$ if and only if $x \not\rightarrow \omega$.

(iii) $L \subseteq \overline{\mathscr{L}^\omega(x)}$.

(iv) $\varepsilon \notin L$.

We define the relation between continuation pairs $\lesssim_{\text{conf}}$. For any two given continuation pairs $(x,L),(x,M)$ we consider $(x,L) \lesssim_{\text{conf}} (x,M)$ to be true if satisfying (x,M) implies that (x,L) must also be satisfied. This is the case if for all traces $t \in M$ there exists a trace $p \in L$ such that $p \sqsubseteq t$. If this is the case no matter what trace t is used to satisfy (x,M), the requirement (x,L) will be satisfiable by some trace p. Because of this we can consider the nonconflicting requirement (x,M) as implying (x,L). This is an important concept which will be used extensively while finding the minimal nonconflicting requirement set.

Example 6.3 Figure 6.2 represents a determistic automaton with a requirement set. The state x has the requirements α and $(\beta\alpha)^*\omega$, the state y has the requirement $\alpha(\beta\alpha)^*\omega$, and the state $\perp$ has the requirement $\emptyset$. If we look at the requirement $(\beta\alpha)^*\omega$ we will notice that it is prefix-free. I.E. there is no trace in this language which is a prefix of another trace in the language. Furthermore all the traces $(\beta\alpha)^*\omega$ lead to a state which can reach the state ω.

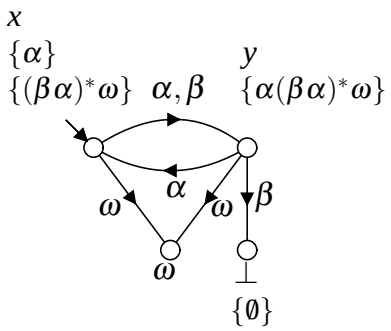


Figure 6.2: Example of an automaton and its requirement set

We define a relation between requirement tuples $\lesssim_{\text{conf}}$. For any two given requirement tuples $(x,L),(x,M)$, $(x,L) \lesssim_{\text{conf}} (x,M)$ if the only way that the requirement (x,M) can be satisfied is if the requirement (x,L) is also satisfied. In this situation it can be considered that the requirement (x,M) makes the requirement (x,L) redundant.

Definition 6.4 Let (x,L) and (x,M) be two continuation pairs. We define the relation $\lesssim_{\text{conf}}$ such that $(x,L) \lesssim_{\text{conf}} (x,M)$ if and only if $M \subseteq L\Sigma^*_\omega$

Example 6.4 Let $(x,\{\alpha,\beta\})$ and $(x,\{\alpha\alpha\})$ be two requirements. $(x,\{\alpha,\beta\}) \lesssim_{\text{conf}} (x,\{\alpha\alpha\})$. This is because the only completion of $(x,\{\alpha\alpha\})$, is a suffix of α. If we instead consider the requirement $(x,\{\alpha\alpha,\beta,\gamma\})$ this requirement would be incomparable to $(x,\{\alpha,\beta\})$. This is because $(x,\{\alpha\alpha,\beta,\gamma\})$ can be satisfied by γ without satisfying $(x,\{\alpha,\beta\})$ where as $(x,\{\alpha,\beta\})$ can in turn be satisfied by α without satisfying $(x,\{\alpha\alpha,\beta,\gamma\})$.

Note $\lesssim_{\text{conf}}$ was defined for automata in section 2.4.

Lemma 6.2 $\lesssim_{\text{conf}}$ is a transitive relation.

$\lesssim_{\text{conf}}$ is a reflexive relation.

$\lesssim_{\text{conf}}$ is an antisymmetric relation.

Proof. First we will show that $\lesssim_{\text{conf}}$ is transitive. Let x be a state, and let L_1,L_2,L_3 be languages such that $(x,L_1) \lesssim_{\text{conf}} (x,L_2) \lesssim_{\text{conf}} (x,L_3)$.

We will prove that $(x,L_1) \lesssim_{\text{conf}} (x,L_3)$.

$L_1 \subseteq L_2\Sigma^*_\omega \subseteq L_3\Sigma^*_\omega\Sigma^*_\omega = L_3\Sigma^*_\omega$. and therefore $(x,L_1) \lesssim_{\text{conf}} (x,L_2)$

Next we will prove that $\lesssim_{\text{conf}}$ is reflexive.

Let (x,L) be a requirement pair. Clearly it holds that $L \subseteq L\Sigma^*_\omega$ and thus $(x,L) \lesssim_{\text{conf}} (x,L)$ by definition.

Lastly we will prove that $\lesssim_{\text{conf}}$ is antisymmetric. Let (x,L) and (x,M) be two pairs such that $(x,L) \lesssim_{\text{conf}} (x,M)$ and $(x,M) \lesssim_{\text{conf}} (x,L)$.

We will prove that $L \subseteq M$. Let $s \in L$ be a trace. We will show that $s \in M$. As $L \subseteq M\Sigma^*_\omega$ there exists a trace $t \in M$ such that $t \sqsubseteq s$. Furthermore as $M \subseteq L\Sigma^*_\omega$ there exists a trace $r \in L$ such that $r \sqsubseteq t$. This implies that $r \sqsubseteq s$. From the definition of a requirement pair, L is prefix-free, therefore $r = s = t$ and thus $s = t \in M$.

The proof for $M \subseteq L$ is analagous. □

Nonconflicting requirements are also related to one another over the $\rightarrow$ relation. For any two nonconflicting requirements (x,L) and (y,M), and trace s, it holds that $(x,L) \xrightarrow{s} (y,M)$ if $s \in \overline{L} - L$, $x \xrightarrow{s} y$, and $M = L/s$. This is because after the trace s, the trunk automaton will reach the state y, and once it reachs this state it will still be neccessary to perform at least one trace in $M = L/s$ in order to complete L. We thus define the transition relation $\rightarrow_{R(G)}$ for an arbitrary automaton G.

Definition 6.5 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be a deterministic automaton.

Then $\rightarrow_{R(G)} = \{((x,L), \sigma, (\delta(x,\sigma), L/\sigma)) | \sigma \in \overline{L} - L \text{ and } L \in \Sigma^* \cup \Sigma^*\omega\}$

Example 6.5 Consider the trunk automaton in figure 6.1. The initial conflict normal form has the nonconflicting requirement $(\{q_0\}, \alpha(\alpha\alpha)^*\beta^+\omega)$. After the event α, the requirement will transition from $\{q_1\}$ to $\{q_1, q_2\}$. In addition as α has already been executed, the new requirement becomes $(\alpha\alpha)^*\beta^+\omega)$, thus $(\{q_0\}, \alpha(\alpha\alpha)^*\beta^+\omega) \xrightarrow{\alpha} (\{q_1,q_2\}, (\alpha\alpha)^*\beta^+\omega)$. Similarly, $(\{q_1,q_2\}, (\alpha\alpha)^*\beta^+\omega) \xrightarrow{\alpha} (\{q_0\}, \alpha(\alpha\alpha)^*\beta^+\omega)$, and $(\{q_1,q_2\}, (\alpha\alpha)^*\beta^+\omega) \xrightarrow{\beta} (\{q_2\}, \beta^*\omega)$. This corresponds to the transitions related to the initial conflict form in figure 6.1.

In addition if a nonconflicting requirement has a trace in its language which does not end in ω this trace transitions back into the original automaton. For example $(\{q_0\}, \{\alpha\}) \xrightarrow{\alpha} \{q_1, q_2\}$.

$\downarrow_G$ is a closure relation for sets of requirements. It ensures that for all requirements (x,L) and (y,M) such that $(x,L) \rightarrow (y,M)$, it holds that $(y,M) \in \downarrow_G (R)$ if $(x,L) \in \downarrow_G (R)$.

Definition 6.6 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be deterministic automaton. Let R be a set of requirement pairs of G.

Then $\downarrow_G (R) = \{(\delta_G(x,s), L/s) | (x,L) \in R \text{ and } s \in \overline{L} - L)\}$

Nonconflicting requirements are grouped into requirement sets. A well-formed requirement set is a set of nonconflicting requirements which is closed under $\downarrow_G$. That is to say if R is a requirement set, then for each requirement $(x,L) \in R$ if $(x,L) \rightarrow (y,M)$ for some (y,M) then $(y,M) \in R$.

Definition 6.7 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be a deterministic automaton, and $R \subseteq Q \times 2^{\Sigma^*_\omega}$ be a set of pairs.

Then R is a requirement set of G if the following holds for all (x,L) in R.

(i) (x,L) is a requirement.

(ii) For all (y,M) such that $(x,L) \to (y,M)$ it holds that $(y,M) \in R$.

Example 6.6 We will consider the automaton $trunk(G)$ from figure 6.1. The nonconflicting requirement set $R = \{(q_0, \alpha(\alpha\alpha)^*\beta^+\omega)\}$ would not be a well-formed requirement set of $trunk(G)$. This is because $(q_0, \alpha(\alpha\alpha)^*\beta^+\omega) \to (\{q_1,q_2\}, (\alpha\alpha)^*\beta^+\omega)$ and $(q_0, \alpha(\alpha\alpha)^*\beta^+\omega) \to (\{q_2\}, \beta^*\omega)$, but neither of these two requirements are contained within R. The nonconflicting requirement set $R' = \{(q_0, \alpha(\alpha\alpha)^*\beta^+\omega),\ (\{q_1,q_2\}, (\alpha\alpha)^*\beta^+\omega), (\{q_2\}, \beta^*\omega)\}$ is well-formed however.

For any set of nonconflicting requirements R, it holds that $\downarrow_G (R)$ is a requirement set. That is to say that closing R under $\downarrow_G$ always results in a well-formed requirement set.

Lemma 6.3 Let $G = \langle \Sigma, Q, \to, Q^\circ \rangle$ be a deterministic automaton. Let R be a set of requirement pairs of G.

Then $\downarrow_G (R)$ is a requirement set of G.

Proof. Let $(x,L) \in \downarrow_G (R)$.

First we will show that (x,L) is in fact a well-formed requirement of G as defined in definition 6.3.

From the definition of $\downarrow_G$ there exists $(w,J) \in R$ and $s \in \overline{J} - J$ such that $(w,J) \xrightarrow{s} (x,L)$. As (w,J) is a requirement pair J is prefix-free therefore $J/s = L$ is also prefix-free. This satisfies requirement i. Second, as $s \in \overline{J}$ and $J/s = L$ it is the case that $L \neq \emptyset$. Furthermore as (w,J) is a requirement pair $\overline{J} \subseteq \overline{\mathscr{L}^\omega(w)}$ from requirement iii, therefore as $s \in \overline{J}$ and G is deterministic $w \xrightarrow{s} x \to \omega$, thus (x,L) satisfies requirement ii. In addition $J \subseteq \mathscr{L}^\omega(w)$ therefore $J/s = L \subseteq \mathscr{L}^\omega(w)/s = \mathscr{L}^\omega(x)$ thus satisfying requirement iii. Finally as $s \notin J$ it holds that $\varepsilon \notin J/s = L$ therefore (x,L) satisfies requirement iv.

Next we will prove that for all $t \in \overline{L}$, $(\delta_G(x,t), L/t) \in \downarrow_G (R)$. For the first case there exists $(w,J) \in R$ and $s \in \overline{J}$ such that $(w,J) \xrightarrow{s} (x,L)$. $\delta_G(w,st) = \delta_G(x,t)$ furthermore $J/st = L/t$ therefore $(\delta_G(x,t), L/t) \in \downarrow_G (R)$ from the definition of $\downarrow_G$. □

Next we will show that the relation $(x,L) \lesssim_{\text{conf}} (x,M)$ is preserved by $\to$. That is to say if (y,N) and (y,O) are two nonconflicting requirements and s is a trace such that $(x,L) \xrightarrow{s} (y,N)$ and $(x,M) \xrightarrow{s} (y,O)$ then $(y,N) \lesssim_{\text{conf}} (y,O)$ also.

Lemma 6.4 Let (x,L) and (x,M) be two requirement pairs such that $(x,L) \lesssim_{\text{conf}} (x,M)$.

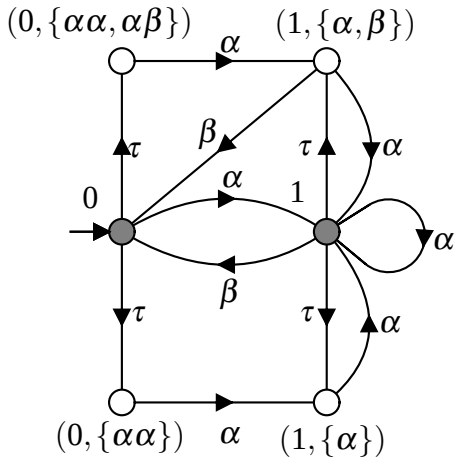


Figure 6.3: Example requirements

Let $s \in \Sigma^*$ be a trace and let (y,N) and (y,O) be two requirement pairs such that $(x,L) \xrightarrow{s} (y,N)$ and $(x,M) \xrightarrow{s} (y,O)$.

Then $(y,N) \lesssim_{\text{conf}} (y,O)$.

Proof.

$$
\begin{aligned}
M &\subseteq L\Sigma_\omega{}^* \quad \text{as } (x,L) \lesssim_{\text{conf}} (x,M) \\
M/s &\subseteq L\Sigma_\omega{}^*/s \\
O &\subseteq L\Sigma_\omega{}^*/s \quad \text{as } (x,M) \xrightarrow{s} (y,O) \\
O &\subseteq L/s\Sigma_\omega{}^* \quad \text{as } L \text{ is prefix-free} \\
O &\subseteq N\Sigma_\omega{}^* \quad \text{as } (x,L) \xrightarrow{s} (y,N)
\end{aligned}
$$

□

Example 6.7 Consider the automaton shown in figure 6.3. Because $(0,\{\alpha\alpha,\alpha\beta\}) \lesssim_{\text{conf}} (0,\{\alpha\alpha\})$ it must also hold that $(1,\{\alpha,\beta\}) \lesssim_{\text{conf}} (1,\{\alpha\})$.

6.1.2 Requirement Automata

In this subsection we introduce how to construct the standard automata representation of a trunk automaton and requirement set. We further give several lemmas showing the conditions upon which states are reached.

The requirement automaton $RA(G,R)$ for a given automaton G and requirement set R is created by combining G and R together. To do this we give $RA(G,R)$ all the transitions in G as well as all the transitions in R as defined in definition 6.5. We then connect G to R, firstly by adding the τ transition $x \xrightarrow{\tau} (x,L)$ for every requirement

$(x,L) \in R$ and secondly by adding the transition $(x,L) \xrightarrow{\sigma} y$ for every $(x,L) \in R$, where $\sigma \in L$ and $x \xrightarrow{\sigma} y$ in G.

Definition 6.8 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be a deterministic automaton. Let R be a requirement set of G.

Then $RA(G,R) = (Q \cup R, \Sigma, \rightarrow_A, X)$ where $\rightarrow_A$ is defined such that for all $x,y \in Q \cup R$ and all $\sigma \in \Sigma_\omega^*$. $x \xrightarrow{\sigma}_A y$ if and only if one of the following holds

- $x \xrightarrow{\sigma} y$.
- $x \xrightarrow{\sigma}_{R(G)} y$.
- $x = (q,L), \sigma \in L$ and $q \xrightarrow{\sigma} y$.

Furthermore $x \xrightarrow{\tau}_A y$ if and only if $x \in Q$ and $y = (x,L)$ such that $(x,L) \in R$.

Remark 6.1 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be a deterministic automaton. Let R be a requirement set of G. Let s be a trace in Σ^*. Let (x,L) be nonconflicting requirement such that $RA(G,R) \stackrel{s}{\Rightarrow} (x,L)$

Then $(s,L) \in \mathrm{CC}(RA(G,R))$.

Example 6.8 Let us consider $trunk(G)$ from figure 6.1. Let $R = \{\{q_0\}, \{\alpha\}, \{q_1,q_2\}, \{\alpha\}, \{q_1,q_2\}, \{\beta,\omega\}\}$. Then $RA(trunk(G),R)$ is the final automaton in 6.1.

In order to reason about the automaton $RA(G,R)$ for any given automaton G and requirement set R, we introduce the following lemmas. Lemma 6.5 shows that for any given trace s, and requirement (x,L), the requirement (x,L) can only be reached if the state x can also be reached.

Lemma 6.5 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be a deterministic automaton. Let R be a requirement set of G. Let $(x,L) \in R$ be a requirement. Let $w \in Q$ be a state. Let $s \in \Sigma^*$ be a trace such that $w \stackrel{s}{\Rightarrow} (x,L)$

Then $w \xrightarrow{s} x$.

Proof. We will prove the claim via induction on $|s|$.

In the base case $s = \varepsilon$. From the construction of $RA(G,R)$ the only silent transitions are of the form $y \xrightarrow{\tau} (y,M)$ where $(y,M) \in R$. Therefore $w \xrightarrow{\varepsilon} (x,L)$ implies that $x = w$. $w \xrightarrow{\varepsilon} w$.

Now let us consider the case where the property is true for the trace s, we will prove that it is also true for $s\sigma$.

Let (x,L) be a requirement such that $w \stackrel{s\sigma}{\Rightarrow} (x,L)$. There are only two basic transitions into (x,L), The first is $x \xrightarrow{\tau} (x,L)$, this case trivially implies that $w \stackrel{s\sigma}{\Rightarrow} x \xrightarrow{\tau} (x,L)$. In the

second case $(y,M) \xrightarrow{\sigma} (x,L)$ for some (y,M) where $y \xrightarrow{\sigma} x$ and $M/\sigma = L$. From the inductive assumption if $w \xrightarrow{s} (y,M)$ then $w \xrightarrow{s} y \xrightarrow{\sigma} x$. □

Lemma 6.6 goes on to show that for any given state x, that $RA(G,R)$ can reach x under exactly the same circumstances in which x is reachable in G.

Lemma 6.6 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be a deterministic automaton. Let R be a requirement set of G. Let $w,x \in Q$ be states.

Then $w \stackrel{s}{\Rightarrow} x$ if and only if $\delta_G(w,s) = x$

Proof. First we will prove that if $\delta_G(w,s) = x$ then $w \stackrel{s}{\Rightarrow} x$. This comes directly from the construction of $RA(G,R)$ as $RA(G,R)$ contains all the transitions in G.

Now we will prove that if $w \stackrel{s}{\Rightarrow} x$ then $\delta_G(w,s) = x$ via induction on $|s|$.

In the base case $s = \varepsilon$. $\delta_G(w,\varepsilon) = w$. From definition 6.8 the only τ transitions in $RA(G,R)$ go to states in R. Therefore it holds that $w \xrightarrow{\varepsilon} y$ for some $y \in Q$ if and only if $y = w$.

Let us assume that the property holds for the trace s, we will prove that it must be true for $s\sigma$.

Let x be a state in Q such that $w \stackrel{s\sigma}{\Rightarrow} x$.

There are only two types of transitions which reach x, either there exists $y \in Q$ such that $w \stackrel{s}{\Rightarrow} y \xrightarrow{\sigma} x$ or there exists $(y,L) \in R$ such that $w \stackrel{s}{\Rightarrow} (y,L) \xrightarrow{\sigma} x$.

In the first case, from the inductive assumption $y = \delta_G(w,s)$, therefore x must equal $\delta_G(w,s\sigma)$.

In the second case there exists $(y,L) \in R$ such that $y \xrightarrow{\sigma} x$ and $\sigma \in L$ where $w \stackrel{s}{\Rightarrow} (y,L)$. From lemma 6.5, $w \stackrel{s}{\Rightarrow} y$, and from the inductive assumption $w = \delta_G(x^\circ, s)$ therefore x must equal $\delta_G(w,s\sigma)$. □

We further show that for any two requirement automata, if they both share the same trunk then for any given state x in that trunk, both requirement automata can reach x under exactly the same situations.

Corollary 6.1 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be a deterministic automaton. Let R,S be two requirement sets of G. Let $w,x \in Q$ be states.

Then $w \stackrel{s}{\Rightarrow}_S x$ if $w \stackrel{s}{\Rightarrow}_R x$

Proof. From lemma 6.6 using R as a requirement set of G as $w \stackrel{s}{\Rightarrow}_R x$ it follows that $\delta_G(w,s) = x$. Again applying lemma 6.6 this time using S as a requirement set of G as $\delta_G(w,s) = x$ it follows that $w \stackrel{s}{\Rightarrow}_S x$. □

We further show that for any two requirement automata which share the same trunk, if one has a requirement for the state x which is reachable after a given trace s, and the

other automaton has a requirement for the state x, that automaton can also reach the requirement on the trace s.

Corollary 6.2 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be a deterministic automaton. Let R, S be two requirement sets of G. Let $(x, L) \in R$ and $(x, M) \in S$ be two requirements.

Then $RA(G,R) \stackrel{s}{\Rightarrow} (x,L)$ implies that $RA(G,S) \stackrel{s}{\Rightarrow} (x,M)$.

Proof. From lemma 6.5 as $RA(G,R) \stackrel{s}{\Rightarrow} (x,L)$ it follows that $RA(G,R) \stackrel{s}{\Rightarrow} x$. Applying lemma 6.1 as $RA(G,R) \stackrel{s}{\Rightarrow} x$ it follows that $RA(G,S) \stackrel{s}{\Rightarrow} x$. Finally from the construction of $RA(G,S)$ as $(x,M) \in S$ it holds that $x \stackrel{\tau}{\rightarrow} (x,M)$ therefore $RA(G,S) \stackrel{s}{\Rightarrow} x \stackrel{\varepsilon}{\Rightarrow} (x,M)$. □

Now we show that the set of certain conflicts of a requirement automaton and its trunk are equivalent.

Lemma 6.7 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be a deterministic automaton.

Then $\mathrm{NCONF}(RA(G,R)) = \mathrm{NCONF}(G)$.

Proof. Let s be a trace in $\mathrm{NCONF}(RA(G,R))$. Let x be a state such that $G \stackrel{s}{\rightarrow} x$. From lemma 6.6 it holds that $RA(G,R) \stackrel{s}{\rightarrow} x$. As $s \in \mathrm{NCONF}(RA(G,R))$ it must hold that x is not blocking, therefore there exists some trace $t\omega$ such that $x \stackrel{t\omega}{\Rightarrow}$ in $RA(G,R)$. Thus from lemma 6.6 $x \stackrel{t\omega}{\rightarrow}$. Therefore x is nonblocking in G and as G is deterministic $s \in \mathrm{NCONF}(G)$.

Now we will prove that $\mathrm{NCONF}(G) \subseteq \mathrm{NCONF}(RA(G,R))$. Let T be a deterministic automaton such that $L(T) = \mathrm{NCONF}(G)$, we will prove that $RA(G,R) \parallel T$ is nonblocking and thus that $\mathrm{NCONF}(G) \subseteq \mathrm{NCONF}(RA(G,R))$. First as $L(T) = \mathrm{NCONF}(G)$ and T is deterministic, $G \parallel T$ is nonblocking.

Let s be a trace and q and q_T be two states such that $RA(G,R) \parallel T \Rightarrow (q, q_T)$. There are two cases, either $q \in Q$ or $q \in R$. In the first case $RA(G,R) \stackrel{s}{\Rightarrow} q$, therefore from lemma 6.6 it also holds that $G \stackrel{s}{\rightarrow} q$. Thus $G \parallel T \stackrel{s}{\Rightarrow} (q, q_T)$. As $G \parallel T$ is nonblocking, (q, q_T) is also nonblocking, thus there exists a trace $t\omega$ such that $(q, q_T) \stackrel{t\omega}{\Rightarrow}$ in $G \parallel T$. As $RA(G,R)$ contains all the transitions in G, $(q, q_T) \stackrel{t\omega}{\Rightarrow}$ in a $RA(G,R) \parallel T$ also. In the second case $q \in R$, let $(x,L) = q$. It holds that $RA(G,R) \stackrel{s}{\Rightarrow} (x,L)$, therefore from 6.5 it must also hold that $G \stackrel{s}{\rightarrow} x$. Thus $G \parallel T \stackrel{s}{\Rightarrow} (x, q_T)$. As $G \parallel T$ is nonblocking (x, q_T) is also nonblocking. Furthermore as R is a requirement set of G it must hold that $L \subseteq \overline{\mathscr{L}^\omega(x)}$. As (x, q_T) is nonblocking in $G \parallel T$ it holds that $\mathscr{L}^\omega(x) \neq \emptyset$, thus from definition 6.3 it must hold that $L \neq \emptyset$, thus there exists a trace $t \in L$, such that $t \in \overline{\mathscr{L}^\omega(x)}$. As $t \in \overline{\mathscr{L}^\omega(x)}$ it holds that $x \stackrel{t}{\rightarrow} y$ where $y \in Q$ and y is nonblocking. As $G \stackrel{s}{\rightarrow} x \stackrel{t}{\rightarrow} y$ and G is deterministic $st \in \mathrm{NCONF}(G)$, therefore $((x,L), q_T) \stackrel{t}{\rightarrow} (y, y_T)$ in $RA(G,R) \parallel T$ where $y_T \in Q_T$. Furthermore (y, y_T) has already been proven nonblocking. □

We can also show that automata derivation is commutative with the construction of a requirement automaton with respect to conflict equivalence.

Lemma 6.8 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be a deterministic automaton. Let R be a requirement set of G. Let s be a trace in Σ^*

Then $RA(G,R)/s \simeq_{\text{conf}} RA(G/s,R)$.

Proof. We will first prove that $RA(G,R)/s \lesssim_{\text{conf}} RA(G/s,R)$. Let T be an automaton such that $RA(G/s,R) \parallel T$ is nonblocking. Let t be a trace such that $RA(G,R)/s \parallel T \xrightarrow{t} (y \in Q \cup R, y_T)$.

We will show that $RA(G/s,R) \parallel T \stackrel{t}{\Rightarrow} (y,y_T)$. From definitions 2.3 and 6.8 the initial state set of $RA(G,R)/s$ is $\{x \in Q \cup R | RA(G,R) \stackrel{s}{\Rightarrow} x\}$. Therefore there must exist a state q such that $RA(G,R) \stackrel{s}{\Rightarrow} q \stackrel{t}{\Rightarrow} y$ because $RA(G,R)/s \stackrel{t}{\Rightarrow} y$. There are two cases: either $q \in Q$ or $q \in R$. In the first case from lemma 6.6 it holds that $G \stackrel{s}{\Rightarrow} q$ therefore q is an initial state of $RA(G/s,R)$ and $RA(G/s,R) \parallel T \stackrel{t}{\Rightarrow} (y,y_T)$. In the second case $q = (x,L) \in R$ in which case from lemma 6.5 it holds that $G \stackrel{s}{\Rightarrow} x$, in which case x is in the initial state set of $RA(G/s,R)$ and $RA(G/s,R) \parallel T \stackrel{\varepsilon}{\Rightarrow} ((x,L) = q, \_) \stackrel{t}{\Rightarrow} (y,y_T)$.

As $RA(G/s,R) \parallel T$ is nonblocking (y,y_T) must be nonblocking. Since (y,y_T) was chosen arbitrarily $RA(G,R)/s$ must also be nonblocking.

Now we will prove $RA(G/s,R) \lesssim_{\text{conf}} RA(G,R)/s$. Let T be an automaton such that $RA(G/s,R) \parallel T$ is nonblocking. Let t be a trace such that $RA(G/s,R) \parallel T \xrightarrow{t} (y \in Q \cup R, y_T)$.

We will show that $RA(G,R)/s \parallel T \stackrel{t}{\Rightarrow} (y,y_T)$. From definitions 2.3 and 6.8 the initial state set of $RA(G/s,R)$ is $\{x \in Q | G \stackrel{s}{\Rightarrow} x\}$. Therefore there must exist a state $x \in Q$ such that $G \xrightarrow{s} x$, furthermore as G is deterministic this state is unique. Thus $x \xrightarrow{t} y$ because $RA(G/s,R) \stackrel{t}{\Rightarrow} y$. From lemma 6.6 it holds that $RA(G,R) \stackrel{s}{\Rightarrow} x$ therefore x is an initial state of $RA(G,R)/s$ and $RA(G,R)/s \parallel T \stackrel{t}{\Rightarrow} (y,y_T)$.

As $RA(G,R)/s \parallel T$ is nonblocking (y,y_T) must be nonblocking. Since (y,y_T) was chosen arbitrarily $RA(G/s,R)$ must also be nonblocking.

□

6.2 Construction

In this section we define how to construct a trunk automaton and a requirement set for any given automaton G. Subsection 6.2.1 introduces how the trunk automaton is constructed, in addition to several properties of this trunk automaton. Subsection 6.2.2 describes how to construct the initial requirement set of any given automaton. This

requirement set can be refined into a unique requirement set using the refinement rules described in section 6.3.

6.2.1 Trunk

In this subsection we introduce how for any given finite state automaton G the trunk automaton $trunk(G)$ can be constructed. We further show that this automaton is well-formed, and common for all conflict equivalent automata. Lastly we give some useful lemmas describing the automaton's behaviour.

As has been previously stated, a nonconflicting requirement is made up of a state and a language (x, L) where for all traces $s \in \Sigma^*$ such that $G \xrightarrow{s} x$, the language L is a conflicting continuation of G. Therefore the trunk automaton must be constructed in such a way that for all $s, t \in \Sigma^*$ and x in $trunk(G)$, if $trunk(G) \xrightarrow{s} x$ and $trunk(G) \xrightarrow{t} x$, then the nonconflicting continuations of s and t must be equal. To accomplish this the trunk automaton is created by taking the subset construction of G and merging states which are conflict equivalent, as will be described in this section.

First we introduce a relation by which state sets are considered conflict equivalent. Given the automaton $A = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ and two state sets $X_1, X_2 \subseteq Q$, we consider X_1 and X_2 to be conflict equivalent if and only if $\langle \Sigma, Q, \rightarrow, X_1 \rangle \simeq_{\text{conf}} \langle \Sigma, Q, \rightarrow, X_2 \rangle$. This relation is used to determine which state sets should be merged.

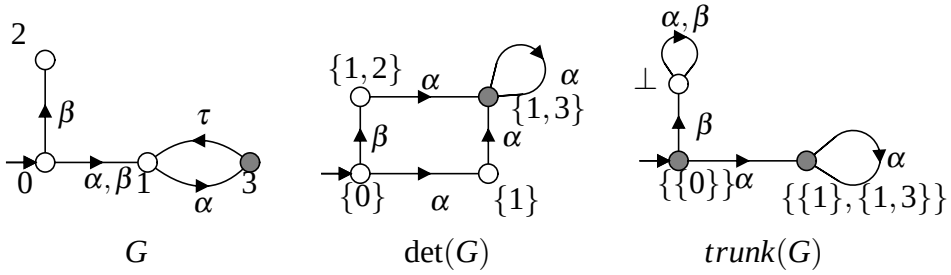
Definition 6.9 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an automaton. Let $X_1, X_2 \subseteq Q$ be two state sets. $X_1 \simeq_{\text{conf}} X_2$ if $\langle \Sigma, Q, \rightarrow, X_1 \rangle \simeq_{\text{conf}} \langle \Sigma, Q, \rightarrow, X_2 \rangle$.

Definition 6.10 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an automaton.
Then $trunk(G) = \langle \Sigma, Q_{conf}, \rightarrow_{conf}, [Q^\circ]_{conf} \rangle$.
Where $Q_{conf} = 2^Q / \simeq_{\text{conf}} \cup \{\bot, \omega\}$.

All state sets which are in the same conflict equivalence class, are grouped into the same state via the $[.]_{conf}$ relation. In addition states in the trunk are merged by the $[[.]]_{conf}$ relation. $[[.]]_{conf}$ also specifically marks out the equivalence class of state sets which are definitely conflicting as being the dump state $\bot$.

Definition 6.11 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an automaton.
$[.]_{conf} : 2^Q \rightarrow Q_{conf}$ where $[X]_{conf} = \{Y \subseteq Q | X \simeq_{\text{conf}} Y\}$
$[[.]]_{conf} : 2^Q \rightarrow Q_{conf}$ where $[[X]]_{conf} = \begin{cases} \bot & \text{if } \varepsilon \in \text{CONF}(X) \\ [x]_{conf} & \text{otherwise} \end{cases}$
where $\text{CONF}(X) = \text{CONF}(\langle \Sigma, Q, \rightarrow, X \rangle)$

We now describe how the transition relation of *trunk* is constructed.



G $\quad$ $\det(G)$ $\quad$ $trunk(G)$

Figure 6.4: An automaton G and it determinised and trunk automaton

Definition 6.12 Let $G = \langle \Sigma, Q, \to, Q^\circ \rangle$ be an automaton.

$\to_{conf} : Q_{conf} \times \Sigma_\omega \cap \{\omega\} \to Q_{conf}$

$$\delta_{conf}(\tilde{X}, \sigma) = \begin{cases} \omega & \text{if } \tilde{X} \neq \bot \text{ and } \sigma = \omega \\ [[\delta_{det}(X, \sigma)]]_{conf} & \text{if } \tilde{X} = [X]_{conf} \\ \bot & \text{if } \tilde{X} = \bot \end{cases}$$

Lastly we put these two together to form the trunk automaton.

Example 6.9 Figure 6.2.1 shows an automaton G, as well as $\det(G)$ and $trunk(G)$. The first step to creating $trunk(G)$, is to construct $\det(G)$. Once we have constructed $\det(G)$ we notice that the state $\{1,2\}$ is in fact a certain conflict, as 2 is blocking, thus $\{1,2\}$ becomes $\bot$. Because $\bot$ represents certain conflicts, even though $\{1,2\}$ can transition to $\{1,3\}$ on an α event, $\bot$ cannot. Furthermore states $\{1\}$ and $\{1,3\}$ are conflict equivalent. This is because both require that any test automaton must be capable of performing an infinite string of α events. Thus in the trunk automaton the states $\{1\}$ and $\{1,3\}$ are merged into the state $\{\{1\},\{1,3\}\}$.

In order to be certain that this is a proper definition of the trunk automaton we must first ensure that the function δ_{conf} is well-defined.

Lemma 6.9 Let $G = \langle \Sigma, Q, \to, Q^\circ \rangle$ be an automaton.

Then δ_{conf} is well-defined.

Proof. Let $\tilde{X} \in \tilde{Q}$ and $X, Y \subseteq Q$ be such that $\tilde{X} = [X]_{conf} = [Y]_{conf}$, we will prove that for all $\sigma \in \Sigma$, $[[\delta_{det}(X,\sigma)]]_{conf} = [[\delta_{det}(Y,\sigma)]]_{conf}$. Because $[X]_{conf} = [Y]_{conf}$ it must hold that $\langle \Sigma, Q, \to, X \rangle \simeq_{\text{conf}} \langle \Sigma, Q, \to, Y \rangle$ and thus $\textsc{Conf}(X) = \textsc{Conf}(Y)$.

There are two cases of σ. Either $\sigma \in \textsc{Conf}(X) = \textsc{Conf}(Y)$ or $\sigma \notin \textsc{Conf}(X) = \textsc{Conf}(Y)$.

In the first case $\sigma \in \textsc{Conf}(X) = \textsc{Conf}(Y)$. From theorem 5.2 as $\sigma \in \textsc{Conf}(X)$ there exists a trace $s \sqsubseteq \sigma$ such that $(\emptyset, \delta_{det}(X,s)) \in LC(O,G)$. From theorem 5.2 as $\varepsilon \notin \textsc{Conf}(X)$ it holds that $(\emptyset, \delta_{det}(X,\varepsilon)) \notin LC(O,G)$, therefore it must be the case that $(\emptyset, \delta_{det}(X,\sigma)) \in LC(O,G)$. From theorem 5.2 as $(\emptyset, \delta_{det}(X,\sigma)) \in LC(O,G)$ it

holds that $\varepsilon \in \text{CONF}(\delta_{det}(X,\sigma))$, therefore $[[\delta_{det}(X,\sigma)]]_{conf} = \bot$. The proof that $[[\delta_{det}(Y,\sigma)]]_{conf} = \bot$ is analagous.

In the second case $\sigma \notin \text{CONF}(X) = \text{CONF}(Y)$. From proposition 5.1 it holds that $[[\delta_{det}(X,\sigma)]]_{conf} = [[\delta_{det}(Y,\sigma)]]_{conf}$ □

Now we must show that for any two automata A and B, if $A \simeq_{\text{conf}} B$ then $trunk(A)$ is isomorphic to $trunk(B)$.

Theorem 6.1 Let $A\langle\Sigma, Q_A, \rightarrow_A, Q_A^\circ\rangle, B = \langle\Sigma, Q_B, \rightarrow_B, Q_B^\circ\rangle$ be two automata such that $A \simeq_{\text{conf}} B$.

Then it holds that $trunk(A) = trunk(B)$.

Proof. We will prove that $trunk(A)$ is isomorphic to $trunk(B)$.

First we we define the relation $\simeq_{\text{trunk}}$ between states. Let $\tilde{X} \in \tilde{Q}_A$ and $\tilde{Y} \in \tilde{Q}_B$ be two states. It holds that $\tilde{X} \simeq_{\text{trunk}} \tilde{Y}$ if and only if either there exists $X \subseteq Q_A$ and $Y \subseteq Q_B$ such that $[X]_{conf} = \tilde{X}$, $[Y]_{conf} = \tilde{Y}$ and $\langle\Sigma, Q_A, \rightarrow_A, X\rangle \simeq_{\text{conf}} \langle\Sigma, Q_B, \rightarrow_B, Y\rangle$ or $\tilde{X} = \tilde{Y} = \bot$.

We will prove that for all $\tilde{X} \in \tilde{Q}_A$, $\tilde{Y}, \tilde{Z} \in \tilde{Q}_B$ such that $\tilde{X} \simeq_{\text{trunk}} \tilde{Y}$ and $\tilde{X} \simeq_{\text{trunk}} \tilde{Z}$, it must also hold that $\tilde{Y} = \tilde{Z}$. From the definition of $\simeq_{\text{trunk}}$ either there must exist $X \subseteq Q_A$, $Y \subseteq Q_B$, and $Z \subseteq Q_B$ such that $[X]_{conf} = \tilde{X}$, $[Y]_{conf} = \tilde{Y}$, $[Z]_{conf} = \tilde{Z}$, and $\langle\Sigma, Q_A, \rightarrow_A, X\rangle \simeq_{\text{conf}} \langle\Sigma, Q_B, \rightarrow_B, Y\rangle \simeq_{\text{conf}} \langle\Sigma, Q_B, \rightarrow_B, Z\rangle$, or $\tilde{X} = \tilde{Y} = \tilde{Z} = \bot$. In the latter case the property is proven trivially. From definition 6.10 $[Y]_{conf} = \{W \subseteq Q_B | W \simeq_{\text{conf}} Y\}$ and $[Z]_{conf} = \{W \subseteq Q_B | W \simeq_{\text{conf}} Z\}$. As $\langle\Sigma, Q_B, \rightarrow_B, Y\rangle \simeq_{\text{conf}} \langle\Sigma, Q_B, \rightarrow_B, Z\rangle$ $W \simeq_{\text{conf}} Y$ is equivalent to $W \simeq_{\text{conf}} Z$ therefore $[Y]_{conf} = \{W \subseteq Q_B | W \simeq_{\text{conf}} Y\} = \{W \subseteq Q_B | W \simeq_{\text{conf}} Z\} = [Z]_{conf}$, thus $\tilde{Y} = \tilde{Z}$.

The proof to show that for all $\tilde{X} \in \tilde{Q}_B$, $\tilde{Y}, \tilde{Z} \in \tilde{Q}_A$ such that $\tilde{X} \simeq_{\text{trunk}} \tilde{Y}$ and $\tilde{X} \simeq_{\text{trunk}} \tilde{Z}$, it must also hold that $\tilde{Y} = \tilde{Z}$ is analagous.

This shows that for any state $\tilde{X} \in \tilde{Q}_A$ there can be at most one state $\tilde{Y} \in \tilde{Q}_B$ such that $\tilde{X} \simeq_{\text{trunk}} \tilde{Y}$, and vice versa. Therefore we can show that if for all $s \in \Sigma^*$ it holds that $\delta^A_{conf}(s) \simeq_{\text{trunk}} \delta^B_{conf}(s)$, that is enough to show that $trunk(A)$ and $trunk(B)$ are isomorphic.

Let $s \in \Sigma^*$ be a trace. We will prove that $\delta^A_{conf}(s) \simeq_{\text{trunk}} \delta^B_{conf}(s)$, via induction on $|s|$.

In the base case $s = \varepsilon$. From construction $\delta^A_{conf}(\varepsilon) = [[Q_A^\circ]]_{conf}$ and $\delta^B_{conf}(\varepsilon) = [[Q_B^\circ]]_{conf}$. There are two cases, either $\varepsilon \in \text{CONF}(Q_A^\circ)$ or $\varepsilon \notin \text{CONF}(Q_A^\circ)$. In the first case $\text{CONF}(Q_A^\circ) = \text{CONF}(A) = \text{CONF}(B) = \text{CONF}(Q_B^\circ)$, therefore $\delta^A_{conf}(\varepsilon) = [[Q_A^\circ]]_{conf} = \bot = \delta^B_{conf}(\varepsilon) = [[Q_B^\circ]]_{conf}$ and thus $\delta^A_{conf}(s) \simeq_{\text{trunk}} \delta^B_{conf}(s)$. In the second case $[[Q_A^\circ]]_{conf} = [Q_A^\circ]_{conf}$ and $[[Q_B^\circ]]_{conf} = [Q_B^\circ]_{conf}$. From the definition of $[.]_{conf}$ it must hold that $A \in [Q_A^\circ]_{conf}$ and $B \in [Q_B^\circ]_{conf}$. As $A \simeq_{\text{conf}} B$ thus $[[Q_A^\circ]]_{conf} \simeq_{\text{trunk}} [[Q_B^\circ]]_{conf}$.

Now we consider the inductive case let us assume that the property holds for s, we will show that it must hold for $s\sigma$.

From the inductive assumption $\delta^A_{conf}(s) \simeq_{\text{trunk}} \delta^B_{conf}(s)$, therefore either $\delta^A_{conf}(s) = \delta^B_{conf}(s) = \bot$ or there exists $X_A \in \delta^A_{conf}(s)$ and $XB \in \delta^B_{conf}(s)$ such that $\langle \Sigma, Q_A, \rightarrow_A, X_A\rangle \simeq_{\text{conf}} \langle \Sigma, Q_B, \rightarrow_B, X_B\rangle$. Let $Y_A \subseteq Q_A$ and $Y_B \subseteq Q_B$ be two state sets such that $X_A \rightarrow_{\det(A)} [\sigma]Y_A$ and $X_B \rightarrow_{\det(B)} [\sigma]Y_B$. In the first case directly from the definition of δ_{conf} it holds that $\delta^A_{conf}(s\sigma) = \delta^B_{conf}(s\sigma) = \bot$. From the definition of δ_{conf} it holds that $\delta^A_{conf}(X_A, \sigma) = [[Y_A]]_{conf}$ and $\delta^B_{conf}(X_B, \sigma) = [[Y_B]]_{conf}$. Here again there are two cases, either $\sigma \in \text{CONF}(X_A) = \text{CONF}(X_B)$ or $\sigma \notin \text{CONF}(X_A) = \text{CONF}(X_B)$. In the first case $\varepsilon \in \text{CONF}(Y_A) = \text{CONF}(Y_B)$ therefore $[[Y_A]]_{conf} = [[Y_B]]_{conf} = \bot$. In the second case from proposition 5.1 as $\langle \Sigma, Q_A, \rightarrow_A, X_A\rangle \simeq_{\text{conf}} \langle \Sigma, Q_B, \rightarrow_B, X_B\rangle$, $\sigma \in \text{NCONF}(X_A) = \text{NCONF}(X_B)$, and $X_A \rightarrow_{\det(A)} [\sigma]Y_A$ and $X_B \rightarrow_{\det(B)} [\sigma]Y_B$, it must hold that $\langle \Sigma, Q_A, \rightarrow_A, Y_A\rangle \simeq_{\text{conf}} \langle \Sigma, Q_A, \rightarrow_A, Y_B\rangle$, therefore $[[Y_A]]_{conf} \simeq_{\text{trunk}} [[Y_B]]_{conf}$. □

Now we will give some useful lemmas about this trunk automaton.

First for any trace s as long as that trace is not a certain conflict in A if there exists a state X such that $\det(A) \xrightarrow{s} X$, then the conflict equivalence class of X is reachable in $trunk(A)$ on s.

Lemma 6.10 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ\rangle$. Let $\tilde{X} \in \tilde{Q}$. Let $X \in \tilde{X}$. Let $s \in \text{NCONF}(X) \cap \Sigma^*$ be a trace.

$\delta_{\det(G)}(X, s) \in \delta_{trunk}(\tilde{X}, s)$.

Proof. We will prove the claim via induction on $|s|$.

In the base case $s = \varepsilon$. In this case $\delta_{conf}(\tilde{X}, \varepsilon) = \tilde{X}$, Let $Y = \delta_{det(G)}(X, \varepsilon)$. $Y = \delta_{det(G)}(X, \varepsilon)$, $X \simeq_{\text{conf}} Y$ thus $Y \in \tilde{X}$.

Now let us assume we have proven the property for the trace s. We will now prove that it must also hold for the trace $s\sigma$. From the inductive assumption $\delta\det(G)(X, s) \in \delta_{conf}(\tilde{X}, s)$, as there exists a state set in $\delta_{conf}(\tilde{X}, s)$ it must hold that $\delta_{conf}(\tilde{X}, s) \neq \bot$. Let $\delta\det(G)(X, s) = Y$ and $\delta_{conf}(\tilde{X}, s) = \tilde{Y}$. From definition 6.10 it holds that $\delta_{conf}(\tilde{Y}, \sigma) = [[\delta\det(G)(Y, \sigma)]]_{conf}$. From assumption $s\sigma \notin \text{CONF}(X)$ therefore $\varepsilon \notin \text{CONF}(\delta_{\det(G)}(Y, \sigma))$. Thus $[[\delta_{\det(G)}(Y, \sigma)]]_{conf} = [\delta_{\det(G)}(Y, \sigma)]_{conf}$. Therefore it is the case that $\delta_{\det(G)}(Y, \sigma) \in [\delta_{\det(G)}(Y, \sigma)]_{conf}$. □

Lemma 6.11 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ\rangle$. Let $X \in \tilde{X}$. Let $\tilde{X} = [[X]]_{conf}$. Let $s \in \text{CONF}(X) \cap \Sigma^*$ be a trace.

$\delta_{conf}(\tilde{X}, s) = \bot$.

Proof. We will prove the claim via induction on $|s|$.

In the base case $s = \varepsilon$ and $\delta_{conf}(\tilde{X}, \varepsilon) = \tilde{X}$. From definition $[[X]]_{conf} = \bot$ if $\varepsilon \in \textsc{Conf}(X)$.

Now let us assume that the property holds for s. We will prove that it must also hold for $s\sigma$.

We will consider two cases, in the first case $s \in \textsc{Conf}(X)$ in the second $s \notin \textsc{Conf}(X)$. For the first case by the inductive assumption $\delta_{conf}(\tilde{X}, s) = \bot$, therefore by definition $\delta_{conf}(\tilde{X}, s\sigma) = \bot$. In the second case $s \in \textsc{NConf}(G)$. There must exist some $X \subset Q$ such that that $X \xrightarrow{s}_{\det(G)} X$, otherwise $s\sigma$ could not be in $\textsc{Conf}(X)$. From lemma 6.10 $trunk(G) \xrightarrow{s} [[X]]_{conf}$, and from construction of $trunk(G)$ $[[X]]_{conf} \xrightarrow{\sigma} [[\delta_{det}(X, \sigma)]]_{conf}$, as $s\sigma \in \textsc{Conf}(G)$, $\varepsilon \in \textsc{Conf}(\delta_{det}(X, \sigma))$, thus $[[\delta_{det}(X, \sigma)]]_{conf} = \bot$. □

Second for any given automaton G, the trunk of G has exactly the same nonconflicting language.

Lemma 6.12 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$.

Then $\textsc{NConf}(G) = \textsc{NConf}(trunk(G))$

Proof. First we will prove that $\textsc{NConf}(G) \subseteq \textsc{NConf}(trunk(G))$. From lemma 6.13 $trunk(G) \lesssim_{\text{conf}} G$. As a direct consequence $\textsc{NConf}(G) \subseteq \textsc{NConf}(trunk(G))$.

Now we will prove that $\textsc{Conf}(G) \subseteq \textsc{Conf}(trunk(G))$.

Let $s \in \textsc{Conf}(G) = \textsc{Conf}(Q^\circ)$. From lemma 6.11 $\delta_{trunk}([[Q^\circ]]_{conf}, s) = \bot$. As $\bot$ is blocking $s \in \textsc{Conf}(trunk(G))$. □

Lastly for any given automaton A. The trunk automaton of A is always less conflicting than A.

Lemma 6.13 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$.

Then $trunk(G) \lesssim_{\text{conf}} G$.

Proof. Let T be a test automaton such that $G \parallel T$ is nonblocking. We will prove that $C(G) \parallel T$ is also nonblocking.

Let $s \in \Sigma^*$ be a trace and (X, x_T) be a state such that $trunk(G) \parallel T \xrightarrow{s} (X, x_T)$. Let (x, x_T) be a state such that $(G \parallel T \xrightarrow{s} (x, x_T)$. As $G \parallel T$ is nonblocking there must exist a trace $t\omega$ such that $st\omega \in \textsc{NConf}(G)$ and $(x, x_T) \xrightarrow{t\omega}$. Let $Y \subseteq Q$ be a state set such that $det(G) \xrightarrow{st} Y$. From lemma 6.10 $C(G) \xrightarrow{st} [[Y]]_{conf}$, as $st \in \mathscr{L}(T)$ and $G \parallel T$ is nonblocking $st \in \textsc{NConf}(G)$. Therefore $\varepsilon \notin \textsc{Conf}(Y)$ therefore $[[Y]]_{conf} \neq \bot$, thus $[[Y]]_{conf} \xrightarrow{\omega} \omega$, thus $(X, x_T) \xrightarrow{t} ([[Y]]_{conf}, y_T) \xrightarrow{\omega}$. □

6.2.2 The Initial Requirement Set

In this section we introduce how to generate a correct, though not yet unique, continuation set for the automaton G. In addition we will prove that this continuation set is well-formed, and that G is conflict equivalent with this requirement set.

The initial continuation set is created in such a way that for every state $\tilde{X}$ in $trunk(G)$, if the state x in the original automaton G is in the state $\tilde{X}$, then we give $\tilde{X}$ the continuation $\mathscr{L}^{\omega}(x) \cap \mathscr{L}^{\omega}_{trunk}(\tilde{X})$. In practice this continuation is equal to the marked language of x minus any certain conflicts. The requirement set is then closed under $\downarrow_G$.

Definition 6.13 Let $G = \langle \Sigma, Q, \rightarrow, Q^{\circ} \rangle$ be an automaton. Let $trunk(G) = \langle \Sigma, Q_{trunk}, \rightarrow_{trunk}, Q^{\circ}_{trunk} \rangle$. Let $R' = \{(\tilde{X}, \mathscr{L}^{\omega}_{G}(x) \cap \mathscr{L}^{\omega}_{trunk}(\tilde{X})) | \tilde{X} \in Q_{trunk}, X \in \tilde{X}, x \in X\}$
$R(G) = \downarrow_G (R')$

We must now prove that $R(G)$ is well-formed according to definition 6.7. To do this we will prove that all the elements of R' are well-formed continuation pairs. After this has been proven it is enough to use lemma 6.3, to state that $\downarrow_G (R')$ is well-formed.

Lemma 6.14 Let $G = \langle \Sigma, Q, \rightarrow, Q^{\circ} \rangle$ be an automaton. Let $R = R(G)$.
R is a requirement set of $trunk(G)$

Proof. First we will prove that all the pairs in $R' = \{(\tilde{X}, \mathscr{L}^{\omega}_{G}(x) \cap \mathscr{L}^{\omega}_{trunk}(\tilde{X})) | \tilde{X} \in \tilde{Q}_{trunk}, X \in \tilde{X}, x \in X\}$ are requirement pairs.

Let $(\tilde{X}, \mathscr{L}^{\omega}_{G}(x) \cap \mathscr{L}^{\omega}_{trunk}(\tilde{X}))$ be a pair such that $\tilde{X} \in Q_C, X \in \tilde{X}$, and $x \in X$. We will show that this pair conforms to the definition of a well-formed nonconflicting requirement given in definition 6.3.

(i) First $\mathscr{L}^{\omega}(x) \cap \mathscr{L}^{\omega}_{trunk}(\tilde{X})$ is prefix-free as it contains only traces which end in ω which is terminal, thus it satisfies condition i.

(ii) The second requirement is that $\mathscr{L}^{\omega}_{G}(x) \cap \mathscr{L}^{\omega}_{trunk}(\tilde{X})) = \emptyset$ if and only if $\mathscr{L}^{\omega}_{trunk}(\tilde{X}) = \emptyset$. First we will consider the case where $\mathscr{L}^{\omega}_{trunk}(\tilde{X}) = \emptyset$, the property obviously holds in this instance as anything intersected with $\emptyset$ equals $\emptyset$.

Now we will prove that if $\mathscr{L}^{\omega}_{trunk}(\tilde{X}) \neq \emptyset$ that $\mathscr{L}^{\omega}_{G}(x) \cap \mathscr{L}^{\omega}_{trunk}(\tilde{X})) \neq \emptyset$. To do this we will prove that there must exist at least one trace $t\omega \in \mathscr{L}^{\omega}_{A}(x)$ where $t\omega \in \mathscr{L}^{\omega}_{trunk}(\tilde{X}))$. First we will show that there exists a trace $t\omega \in \mathscr{L}^{\omega}(x) - \textsc{Conf}(X)$. From the definition of trunk automata give in definition 6.12 it must hold that $\tilde{X} \neq \perp$. This is because $\mathscr{L}^{\omega}_{trunk}(\perp)$ always equals $\emptyset$. Furthermore as $[[X]]_{conf} \neq \perp$, it holds that $\varepsilon \notin \textsc{Conf}(X)$, therefore there must exist at least one trace $t\omega \in \mathscr{L}^{\omega}(x) - \textsc{Conf}(X)$. Furthermore as $t \in \textsc{NConf}(X)$ and $X \in \tilde{X}$, $\delta_{conf}(\tilde{X}, t) \neq \perp$, therefore $\delta_{conf}(\tilde{X}, t) \xrightarrow{\omega}$.

(iii) This requirement comes directly from the definition.

(iv) Fourthly $\mathscr{L}^{\omega}(x) \cap \mathscr{L}^{\omega}_{trunk}(\tilde{X})$ only contains traces which end in ω, therefore ε cannot be an element of $\mathscr{L}^{\omega}(x) \cap \mathscr{L}^{\omega}_{trunk}(\tilde{X})$.

As all the pairs in $\{(\tilde{X}, \mathscr{L}^{\omega}(x) - \textsc{Conf}(X)) | \tilde{X} \in Q_C, X \in \tilde{X}, x \in X\}$ are requirement pairs $\downarrow_G \{(\tilde{X}, \mathscr{L}^{\omega}(x) - \textsc{Conf}(X)) | \tilde{X} \in Q_C, X \in \tilde{X}, x \in X\} = R(A)$ is a requirement set of $trunk(A)$. □

We now go on to show that all the requirements in R represent nonconflicting requirement of the automaton G.

Lemma 6.15 Let $G = \langle \Sigma, Q, \rightarrow, Q^{\circ} \rangle$ be an automaton. Let $(\tilde{X}, L) \in R(G)$ be a requirement. Let $s \in \Sigma^* \cap \mathrm{N}\textsc{Conf}(G)$ be a trace such that $RA(trunk(G), R(G)) \xrightarrow{s} (\tilde{X}, L)$.

It holds that $(s, L) \in \mathrm{CC}(G)$.

Proof. Let T be a test automaton and x_T be a state such that $G \parallel T$ is nonblocking and $T \xrightarrow{s} x_T$. We will show that their exists $t \in L$ such that $x_T \xrightarrow{t}$.

First let us assume that $(\tilde{X}, L) \in R'(G)$. From definition 6.13 it holds that $R' = \{(\tilde{X}, \mathscr{L}^{\omega}_A(x) \cap \mathscr{L}^{\omega}_{trunk}(\tilde{X})) | \tilde{X} \in Q_{trunk}, X \in \tilde{X}, x \in X\}$. Therefore there exists a state set $X \in \tilde{X}, x \in X$ such that $L = \mathscr{L}^{\omega}_G(x) \cap \mathscr{L}^{\omega}_{trunk}(\tilde{X})$.

From lemma 6.5 it holds that $RA(trunk(G), R(G)) \xrightarrow{s} \tilde{X}$ as $RA(trunk(G), R(G)) \xrightarrow{s} (\tilde{X}, L)$. From lemma 6.10 $\det(G) \xrightarrow{s} X$ where $X \in \tilde{X}$ as $trunk(G) \xrightarrow{s} \tilde{X}$. As $\det(G) \xrightarrow{s} X$ it holds that $G \stackrel{s}{\Rightarrow} x$. Therefore $G \parallel T \stackrel{s}{\Rightarrow} (x, x_T)$. As $G \parallel T$ is nonblocking it holds that (x, x_T) is nonblocking. Thus there exists $u\omega$ such that $(x, x_T) \stackrel{u\omega}{\Rightarrow}$. We will show that $u\omega$ is an element of L, to do this we will show that it is in both $\mathscr{L}^{\omega}_G(x)$ and $\mathscr{L}^{\omega}_{trunk}(\tilde{X})$. As $x \stackrel{u\omega}{\Rightarrow}$ it is the case that $u\omega \in \mathscr{L}^{\omega}_G(x)$. Now we must show that $u\omega \in \mathscr{L}^{\omega}_{trunk}(\tilde{X})$. As $G \parallel T$ is nonblocking it holds that $u\omega \notin \textsc{Conf}(X)$ as $\det(G) \stackrel{s}{\Rightarrow} X$. Let Y be the state set such that $X \xrightarrow{u} Y$. From lemma 6.10 it holds that $\tilde{X} \xrightarrow{u} \tilde{Y}$, where $\tilde{Y} \neq \bot$, therefore $\tilde{Y} \xrightarrow{\omega}$, therefore $t\omega \in \mathscr{L}^{\omega}_{trunk}(\tilde{X})$ and further $t\omega \in L$.

Now we will assume that $(\tilde{X}, L) \in R(G)$. From definitions 6.13 and 6.6 it holds that $R(G) = \{(\tilde{Z}, N) | (\tilde{Y}, M) \rightarrow (\tilde{Z}, N) \text{ where } (\tilde{Y}, M) \in R'(G)\}$. Thus there exists a nonconflicting requirement $(Y, M) \in R'(G)$ such that $(\tilde{Y}, M) \xrightarrow{u} (\tilde{X}, L)$. Let $Y \in \tilde{Y}$ be a state set. From lemma 6.10 it holds that $Y \xrightarrow{s} X$ where $X \in \tilde{X}$ as $\tilde{Y} \xrightarrow{s} \tilde{X}$. Furthermore as $(\varepsilon, M) \in \mathrm{CC}(\langle \Sigma, Q, \rightarrow, Y \rangle$ it holds that $(\varepsilon, L) \in \mathrm{CC}(\langle \Sigma, Q, \rightarrow, X \rangle)$. Furthermore from lemma 6.10 $\det(G) \xrightarrow{s} W$ where $W \in \tilde{X}$ as $trunk(G) \xrightarrow{s} \tilde{X}$. Furthermore as both X and W are in $\tilde{X}$ it holds that $X \simeq_{\text{conf}} W$. Therefore $(\varepsilon, L) \in \mathrm{CC}(\langle \Sigma, Q, \rightarrow, W \rangle)$ and thus $(s, L) \in \mathrm{CC}(G)$. □

Now we must prove that for any given automaton G, that $A(trunk(G), R(G))$ is always conflict equivalent with G.

Theorem 6.2 Let $G = \langle \Sigma, Q_G, \rightarrow_G, Q_G^\circ \rangle$. Let $trunk = trunk(G)$. Let $R = R(G)$

Then $RA(trunk, R) \simeq_{\text{conf}} G$, where $RA(trunk, R)$ is constructed according to the definition 6.8

Proof. First let us prove that $RA(trunk, R)$ is less conflicting than G.

Let T be an automaton such that $G \parallel T$ is nonblocking. Let s be a trace in Σ^* and let $(\tilde{W}, x_T)$ be two states such that $RA(trunk, R) \parallel T \xrightarrow{s} (\tilde{W}, x_T)$.

We will now prove that the state $(\tilde{W}, x_T)$ is not blocking. Either $\tilde{W} \in R$ or $\tilde{W} \in \tilde{Q}$.

First we consider the case were $\tilde{W} \in \tilde{Q}$. From lemma 6.6 as $RA(trunk, R) \xrightarrow{s} \tilde{W}$, $trunk \xrightarrow{s} \tilde{W}$, furthermore from lemma 6.13 $trunk(G) \lesssim_{\text{conf}} G$. Thus there must exist some trace $t\omega$ such that $\tilde{W} \xrightarrow{t\omega}_{trunk}$, because $G \parallel T$ is nonblocking. Finally from lemma 6.6 it holds that $\tilde{W} \xrightarrow{t\omega}_{RA}$ as $\tilde{W} \xrightarrow{t\omega}_{trunk}$.

Next we consider the case where $(\tilde{X}, L) \in R$. From lemma 6.15 it holds that $(s, L) \in \text{CC}(G)$ as $(\tilde{X}, L) \in R(G)$ and $s \in \text{NCONF}(G)$ and $G \xRightarrow{s} (X, L)$. Therefore as $G \parallel T$ is nonblocking, there exists a trace $t \in L$ such that $x_T \xrightarrow{t}$. $((\tilde{X}, L), x_T) \xrightarrow{t} (\tilde{Y} \in Q_{trunk}, y \in Q_T)$, as this class of states has already been proven nonblocking it holds that $((\tilde{X}, L), x_T)$ is nonblocking.

Second let us prove that G is less conflicting than $RA(trunk, R)$

Let T be an automaton such that $RA(trunk, R) \parallel T$ is nonblocking. Let s be a trace in Σ^*. Let (x, x_T) be a state such that $G \parallel T \xrightarrow{s} (x, x_T)$. It must hold that $s \in \text{NCONF}(RA(trunk, R))$ as $G \parallel T$ is nonblocking.

From lemma 6.7 it holds that $s \in \text{NCONF}(trunk)$. Thus from lemma 6.12 it holds that $s \in \text{NCONF}(RA(trunk, R))$. Let X be a state set such that $\det(G) \xrightarrow{s} X$. Let $x \in X$. From 6.10 it holds that $trunk \xrightarrow{s} [[X]]_{conf}$. From definition 6.13 it holds that $([[X]]_{conf}, \mathscr{L}^\omega(x) \cap \mathscr{L}^\omega([[X]]_{conf})) \in R$. Thus from definition 6.8 it is the case that $RA(trunk, R)$ $[[X]]_{conf} \xRightarrow{\varepsilon} ([[X]]_{conf}, \mathscr{L}^\omega(x) \cap \mathscr{L}^\omega([[X]]_{conf}))$. As $RA(trunk, R) \parallel T$ is nonblocking, x_T must be capable of performing at least one trace in $t\omega \in \mathscr{L}^\omega(x)$, therefore $(x, x_T) \xrightarrow{t\omega}$ □

6.3 Refinement

If we are given a trunk automaton G, as well as a nonconflicting requirement set R, it is possible to progressively refine the requirement set R into progressively smaller refinement sets using refinement relations. In this section we will describe two refinements. The first refinement is the strongly connected requirements rule. If R and S are two requirement sets such that applying the strongly connected requirement rule to R results in S then $R \succ_\psi S$. The second refinement is the requirement subsumption rule. Again if R and S are two requirement sets such that applying the requirement subsumption

rule to R results in S then $R \succ_\phi S$. We will also prove that the use of each of these refinements preserves conflict equivalence. We will further go on to show that as long as R is a finite set, it is always possible to iteratively refine R using both the strongly connected requirement rule and the requirement subsumption rule until we reach an irreducible requirement set. That is to say a requirement set which can no longer be refined by either ψ or ϕ. Furthermore the number of times the strongly connected requirement rule as well as the requirement subsumption rule will need to be applied to in order to find this irreducible requirement set will themselves be finite.

6.3.1 The Strongly Connected Requirements Rule

The first refinement rule is the strongly connected requirements rule. For this refinement we notice that for a given nonconflicting requirement set R, it is possible that R may contain two requirements (x,L) and (y,M) such that $(x,L) \rightarrow (y,M)$ but $(y,M) \not\rightarrow (x,L)$, that is to say (x,L) and (y,M) are only weakly connected. Because (x,L) and (y,M) are only weakly connected the requirement (x,L) can be split from (y,M). We do this by adding every traces $s \in L$ such that $(x,L) \xrightarrow{s} (y,M)$ to L, and then pruning the resulting language of L back to the shortest accepting traces. This then leads to a set of nonconflicting requirements such that if (x,L) and (y,M) are connected, then they must also be strongly connected. This transformation will weaken the requirement (x,L), that is to say that the transformed requirement is less conflicting than (x,L), yet it does so in such a way that all the conflict information, which has been removed from (x,L), is still contained in (y,M).

Example 6.10 Take for example the automaton $RA(G.R)$ in figure 6.5. The requirement $(0,\{\alpha\beta,\gamma\})$ in this automaton is weakly connected to the requirement $(1,\{\beta\})$, because of this we can split $(0,\{\alpha\beta,\gamma\})$ from $(1,\{\beta\})$, thus resulting in the automaton $RA(G,S)$. In order to transform $(0,\{\alpha\beta,\gamma\})$ into $(0,\{\alpha,\gamma\})$, we first take all the traces which enter the state $(1,\{\beta\})$ from $(0,\{\alpha\beta,\gamma\})$, which in this case is only α, and add them to $(0,\{\alpha\beta,\gamma\})$, giving us $(0,\{\alpha\beta,\gamma,\alpha\})$. The language of this requirement is not prefix-free, therefore we use the prune function on it, this in turn removes the trace $\alpha\beta$ from the language as α is a prefix of it. Even though the nonconflicting requirement $(0,\{\alpha,\gamma\})$ is weaker than $(0,\{\alpha\beta,\gamma\})$, if the requirement $(0,\{\alpha,\gamma\})$ is satisfied by an α transition it reaches the state 1 which has the requirement $(1,\{\beta\})$, and will thus have to perform a β transition. Because of this we can say that the requirement $(0,\{\alpha\beta,\gamma\})$ is still implied.

Example 6.11 Figure 6.6 shows an example where the strongly connected component is larger than a single state. In this example the nonconflicting requirements

$(0,(\beta\beta)^*\alpha\alpha)$ and $(0,\beta(\beta\beta)^*\alpha\alpha)$ are strongly connected to one another, but only weakly connected to the requirement $(1,\{\alpha\})$. Because of this they can be transformed into the requirements $(0,(\beta\beta)^*\alpha)$ and $(0,\beta(\beta\beta)^*\alpha)$ respectively. If we look specifically at how $(0,\beta(\beta\beta)^*\alpha\alpha)$ is transformed into $(0,\beta(\beta\beta)^*\alpha)$, again we add all traces which enter the state $(1,\{\alpha\})$, in this case it is the language $\beta(\beta\beta)^*\alpha$, which results in the requirement $(0,\beta(\beta\beta)^*\alpha\alpha\cup\beta(\beta\beta)^*\alpha)$, and as all traces in the language $\beta(\beta\beta)^*\alpha\alpha$ have a prefix in the language $\beta(\beta\beta)^*\alpha$ this is then pruned back to the requirement $(0,\beta(\beta\beta)^*\alpha)$. Again each of these requirements is strictly less conflicting than their original, but it is also still the case that under all circumstances in which the new requirement is satisfied the state 1 will be reached in $RA(G,S)$. As 1 has the nonconflicting requirement $(1,\{\alpha\})$, it will still be the case that in order for the system as a whole to be nonblocking it will have to be able to perform the extra α transition.

Example 6.12 Figure 6.7 shows an example where multiple strongly connected components are pruned in a single step. In this example there are four different strongly connected components.

- $(0,\alpha(\alpha\alpha)^*(\beta^*\omega))$ and $(1,(\alpha\alpha)^*(\beta^*\omega))$.
- $(1,\beta^*\omega)$.
- $(1,(\alpha\alpha)^+(\beta^*\omega))$.
- $(2,\beta^*\omega)$.

Other than $(2,\beta^*\omega)$ all of these components are connected to at least one other strongly connected component, therefore all of these components can be pruned back. The strongly connected component made up of $(0,\alpha(\alpha\alpha)^*(\beta^*\omega))$ and $(1,(\alpha\alpha)^*(\beta^*\omega))$ can have all of its traces which lead to the components $(2,\beta^*\omega)$ pruned back thus giving us a component made of the requirements $(0,\alpha(\alpha\alpha)^*(\beta|\omega))$ and $(1,(\alpha\alpha)^*(\beta|\omega))$. In the same way $(1,\beta^*\omega)$ is pruned back to $(1,\{\beta,\omega\})$. Finally the requirement $(1,(\alpha\alpha)^+(\beta^*\omega))$ is pruned back to the requirement $(1,\{\alpha\})$ as the requirement transitions to a new strongly connected component as soon as an α event occurs. All of these operations are commutative with one another, thus all simplifications can be done at the same time.

Definition 6.14 Let $G=\langle\Sigma,Q,\rightarrow,Q^\circ\rangle$ be a deterministic automaton.

Let (x,L) be a nonconflicting requirement pair.

$\psi(x,L)=(x,prune(L\cup\{s\in\overline{L}|(x,L)\xrightarrow{s}(y,M)\not\rightarrow(x,L)\text{ for some }(y,M)\}))$

Let R be a requirement set of G. Then $\psi(R)=\{\psi(x,L)|(x,L)\in R\}$.

Let R and S be two requirement sets of G. Then $R\succ_\psi S$ if and only if $\psi(R)=S$.

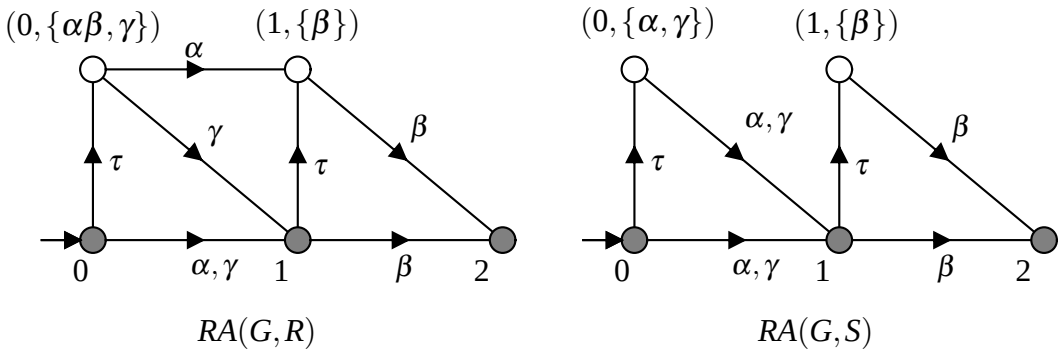


Figure 6.5: An example of pruning weakly connected components

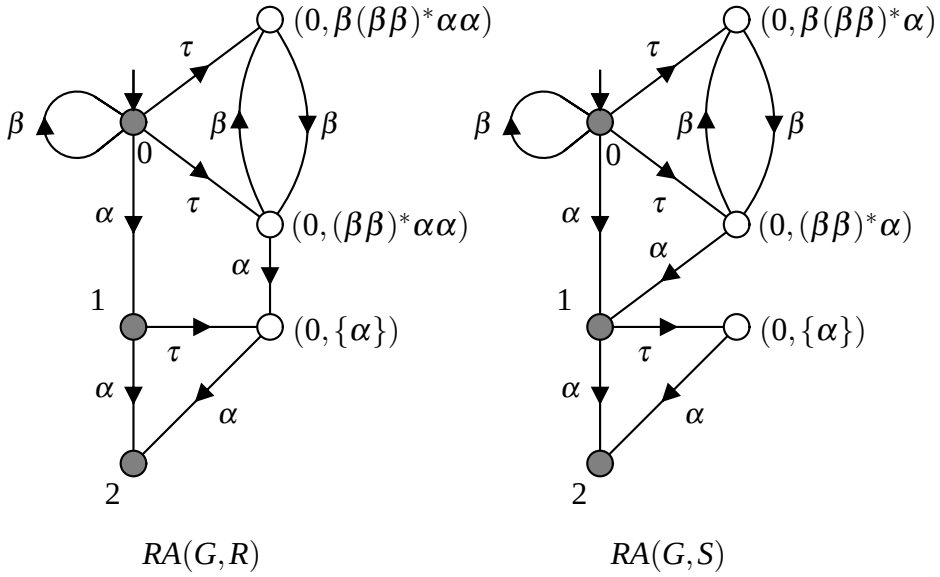


Figure 6.6: An example of pruning weakly connected components which contains loops

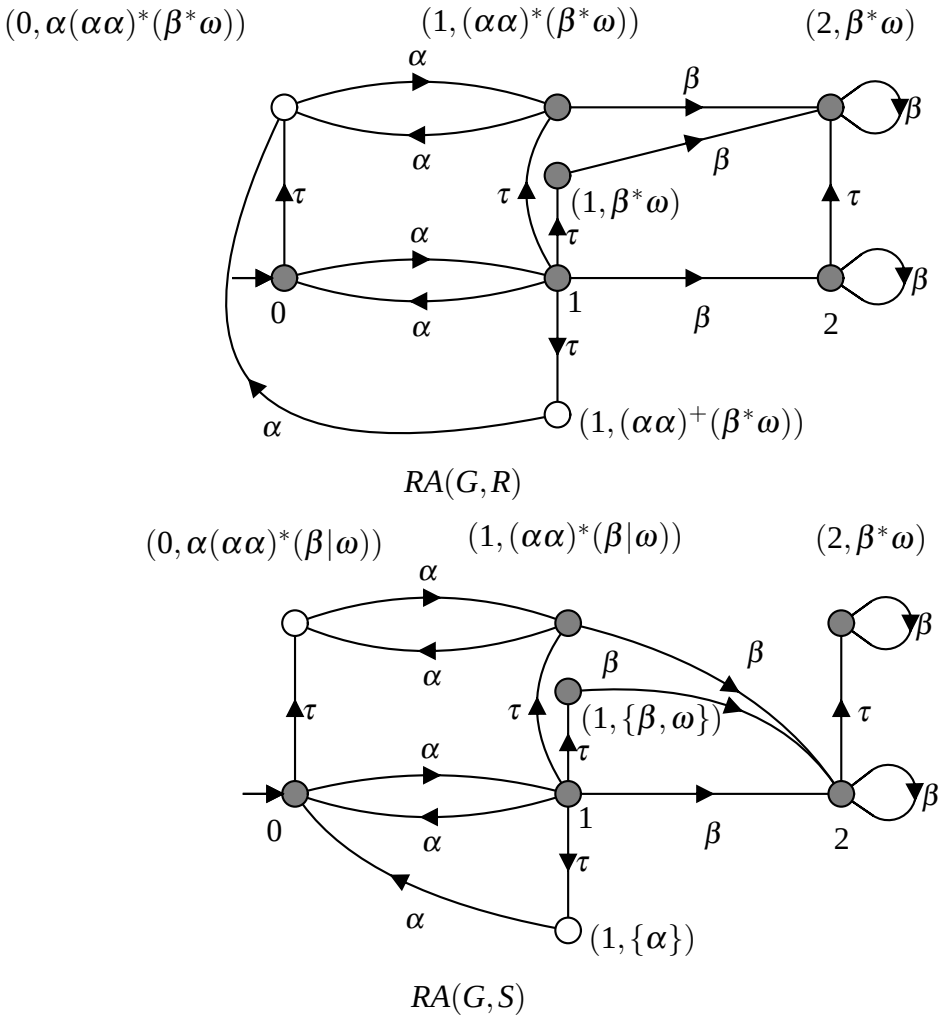


Figure 6.7: An example of pruning multiple components at the same time

Now that we have defined the strongly connected requirements rule, we must go on to prove that it only produces well-defined requirements to be used with subsequent refinement steps, in addition to preserving conflict equivalence.

We will do this by first showing that applying ψ to a single nonconflicting requirement, always results in another nonconflicting requirement. We will then show that the ψ function preserves the $\rightarrow$ relation between nonconflicting requirements. Next we will show that applying ψ to a well-formed requirement set, always results in another well-formed requirement set. Finally we will prove that the requirement automaton of the original requirement set is conflict-equivalent to the requirement automaton of the new requirement set.

Lemma 6.16 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be a deterministic automaton. Let (x, L) be a requirement set of G.

Then $\psi(x, L)$ is also a nonconflicting requirement of G, according to definition 6.3.

Proof. Let $(x, M) = \psi(x, L)$. We will show that $\psi(x, L)$ fulfills all the conditions of being a requirement described in definition 6.3. From definition 6.14

$\psi(x, L) = (x, \mathit{prune}(L \cup \{s \in \overline{L} | (x, L) \xrightarrow{s} (y, M) \not\rightarrow (x, L) \text{ for some } (y, M)\}))$

(i) Any language which is the result of *prune* is prefix-free, therefore (x, M) satisfies condition i

(ii) As (x, L) is a requirement of G the language $L = \emptyset$ if and only if $x \not\rightarrow \omega$. We will prove that $M = \emptyset$ if and only if $L = \emptyset$. It is obvious that if $L = \emptyset$ then $M = \emptyset$ as from definition 6.14 the language M only contains traces from $\overline{L}$ and $\overline{\emptyset} = \emptyset$. Now we will show that if $L \neq \emptyset$ then $M \neq \emptyset$. Let $N = L \cup \{s \in \overline{L} | (x, L) \xrightarrow{s} (y, O) \not\rightarrow (x, L) \text{ for some } (y, O)\}$. As $L \neq \emptyset$ it is also the case that $N \neq \emptyset$ as N is a superset of L. As $N \neq \emptyset$ there exists a trace $s \in N$. From definition 6.2 as $s \in N$, either $s \in \mathit{prune}(N)$ or there exists $p \in N$ such that $p \sqsubseteq s$. In either case $\mathit{prune}(N)$ is non empty. As $M = \mathit{prune}(N) = \mathit{prune}(L \cup \{s \in \overline{L} | (x, L) \xrightarrow{s} (y, O) \not\rightarrow (x, L) \text{ for some } (y, O)\})$. Thus (x, M) satisfies condition ii.

(iii) As (x, L) is a requirement of G, $L \subseteq \overline{\mathscr{L}^\omega(x)}$ this implies that $\overline{L} \subseteq \overline{\mathscr{L}^\omega(x)}$. As M only contains traces which are in $\overline{L}$, $M \subseteq \overline{L} \subseteq \overline{\mathscr{L}^\omega(x)}$, therefore (x, M) satisfies condition iii.

(iv) As (x, L) is a requirement $\varepsilon \notin L$. Furthermore $\varepsilon \notin \{s \in \overline{L} | (x, L) \xrightarrow{s} (y, M) \not\rightarrow (x, L) \text{ for some } (y, M)\}$ as $(x, L) \xrightarrow{\varepsilon} (x, L) \rightarrow (x, L)$. As $\varepsilon \notin L \cup \{s \in \overline{L} | (x, L) \xrightarrow{s} (y, M) \not\rightarrow (x, L) \text{ for some } (y, M)\}$ it must not be in $\mathit{prune}(L \cup \{s \in \overline{L} | (x, L) \xrightarrow{s} (y, M) \not\rightarrow (x, L) \text{ for some } (y, M)\})$ either. $\square$

We next go on to show that the ψ function preserves the $\rightarrow$ relation. Let (x,L) and (x,M) be two nonconflicting requirements such that $\psi(x,L) = (x,M)$, it will always hold that if (x,M) can reach another nonconflicting requirement (y,O) using the trace s then (x,L) will also be able to reach a requirement (y,N) using the same trace, and that $\psi(y,N) = (y,O)$.

Example 6.13 Consider the nonconflicting requirements in example 6.10. If we look at the requirements $(0,(\beta\beta)^*\alpha\alpha)$ and $(0,\beta(\beta\beta)^*\alpha\alpha)$ in R as well as the requirements $(0,(\beta\beta)^*\alpha)$ and $(0,\beta(\beta\beta)^*\alpha)$ in S it is the case that $\psi(0,(\beta\beta)^*\alpha\alpha) = (0,(\beta\beta)^*\alpha)$ and $\psi(0,\beta(\beta\beta)^*\alpha\alpha) = (0,\beta(\beta\beta)^*\alpha)$. Under all circumstances when $(0,(\beta\beta)^*\alpha) \rightarrow (0,\beta(\beta\beta)^*\alpha)$, it is also the case $(0,(\beta\beta)^*\alpha\alpha) \rightarrow (0,\beta(\beta\beta)^*\alpha\alpha)$.

Lemma 6.17 Let (x,L) and (x,M) be two requirement pairs such that $\psi(x,L) = (x,M)$. Let $s \in \overline{M} - M$ be a trace. Let (y,N) and (y,O) be two requirement pairs such that $(x,L) \xrightarrow{s} (y,N)$ and $(x,M) \xrightarrow{s} (y,O)$

Then $\psi(y,N) = (y,O)$

Proof. Let Q be a language such that $(y,Q) = \psi(y,N)$. We will prove that $Q = O$.

$$O = prune(L \cup \{t \in \overline{L} | (x,L) \xrightarrow{t} (z,P) \not\rightarrow (x,L)\})/s$$

This is because from definition 6.14 it holds that
$M = prune(L \cup \{t \in \overline{L} | (x,L) \xrightarrow{t} (z,P) \not\rightarrow (x,L)\})$ as $M \neq \{\varepsilon\}$ as there exists $s \in \overline{M} - M$. Furthermore as $(x,M) \xrightarrow{s} (y,O)$ it holds that $M/s = O$.

$$\begin{aligned}
O &= prune(L \cup \{t \in \overline{L} | (x,L) \xrightarrow{t} (z,P) \not\rightarrow (x,L)\})/s \\
&= prune(L/s \cup \{t \in \overline{L} | (x,L) \xrightarrow{t} (z,P) \not\rightarrow (x,L)\}/s) \text{ from lemma 6.1} \\
&= prune(L/s \cup \{st \in \overline{L} | (x,L) \xrightarrow{st} (z,P) \not\rightarrow (x,L)\}) \\
&= prune(L/s \cup \{t \in \overline{L/s} | (x,L) \xrightarrow{st} (z,P) \not\rightarrow (x,L)\}) \\
&= prune(L/s \cup \{t \in \overline{L/s} | (x,L) \xrightarrow{st} (z,P) \not\rightarrow (x,L)\}) \\
&= prune(L/s \cup \{t \in \overline{L/s} | (x,L) \xrightarrow{st} (z,P) \not\rightarrow (y,N)\})
\end{aligned}$$

This is because $(y,N) \rightarrow (x,L)$. We prove this by contradiction. Let us assume that $(y,N) \not\rightarrow (x,L)$. $M = prune(\{t \in \overline{L} | (x,L) \xrightarrow{t} (z,P) \not\rightarrow (x,L)\})$. As $(y,N) \not\rightarrow (x,L)$ it also holds that $s \in \{t \in \overline{L} | (x,L) \xrightarrow{t} (z,P) \not\rightarrow (x,L)\}$. Therefore $prune(\{t \in \overline{L} | (x,L) \xrightarrow{t} (z,P) \not\rightarrow (x,L)\}) = M$ must contain some trace $p \in M$ such that $p \sqsubseteq s$. As M is prefix-

free this contradicts the assumption that $s \in \overline{M} - M$.

$$\begin{aligned} O &= \mathit{prune}(L/s \cup \{t \in \overline{L/s} | (x,L) \xrightarrow{st} (z,P) \not\rightarrow (y,N)\}) \\ &= \mathit{prune}(\{N \cup \{t \in \overline{N} | (x,L) \xrightarrow{st} (z,P) \not\rightarrow (y,N)\}) \\ &= \mathit{prune}(\{N \cup \{t \in \overline{N} | (x,L) \xrightarrow{s} (y,N) \xrightarrow{t} (z,P) \not\rightarrow (y,N)\}) \\ &= \mathit{prune}(\{N \cup \{t \in \overline{N} | (y,N) \xrightarrow{t} (z,P) \not\rightarrow (y,N)\}) \\ &= Q \end{aligned}$$

This is because as $(x,L) \xrightarrow{s} (y,N)$ it holds that $L/s = N$, and as $(x,L) \xrightarrow{st} (z,P)$ it also follows $(x,L) \xrightarrow{s} (y,N) \xrightarrow{t} (z,P)$. Furthermore as $(x,L) \xrightarrow{s} (y,N)$ is true by assumption it can be removed. Lastly $Q = \mathit{prune}(\{t \in \overline{N} | (y,N) \xrightarrow{t} (z,P) \not\rightarrow (y,N)\})$ by definition 6.14 □

Now we show that if the strongly connected requirements rule is used on a well-formed requirement set that it will always result in another well-formed requirement set. This is important as it ensures that we can continue using more refinements on the result of the ψ operation.

Lemma 6.18 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be a deterministic automaton. Let R be a requirement set of G.

Then $\psi(R)$ is a requirement set of G.

Proof. Let $(x,M) \in \psi(R)$. From definition 6.14 there exists $(x,L) \in R$ such that $\psi(x,L) = (x,M)$.

(i) Lemma 6.16 shows that (x,M) must be a requirement of G.

(ii) Next we will prove that for all (y,O) such that $(x,M) \xrightarrow{s} (y,O)$ for some s, it must hold that $(y,O) \in \psi(R)$.

As $(x,M) \xrightarrow{s}$ it holds that $s \in \overline{M} - M$. Furthermore as M only contains traces in $\overline{L}$ it must also hold that $s \in \overline{L}$. Thus there exists (y,N) such that $(x,L) \xrightarrow{s} (y,N)$. As R is a requirement set of G it holds that $(y,N) \in R$. From lemma 6.17 as $(x,L) \xrightarrow{s} (y,N)$ and $(x,M) \xrightarrow{s} (y,O)$, it must also hold that $\psi(y,N) = (y,O)$. Therefore (y,O) must be in $\psi(R)$.

□

Finally we must show that for any given trunk automaton G and requirement set R, that $RA(G,R) \simeq_{\text{conf}} RA(G,\psi(R))$. This ensures that ψ preserves conflict-equivalence. Because of this all requirement sets which are produced by this rule stay within the

same conflict equivalence class. Because of this we can be certain that the trunk and requirement set represent a canonical form of the original automaton's conflict equivalence class.

Theorem 6.3 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be a deterministic automaton. Let R, S be two requirement sets of G such that $R \succ_\psi S$.

Then $RA(G,R) \simeq_{\text{conf}} RA(G,S)$.

Proof. First we will prove that $RA(G,S) \lesssim_{\text{conf}} RA(G,R)$. Let T be an automaton such that $RA(G,R) \parallel T$ is nonblocking, let us prove that $RA(G,S) \parallel T$ is also nonblocking.

Let $s \in \Sigma^*$ be a trace such that there exists (q, x_T) where $RA(G,S) \parallel T \stackrel{s}{\Rightarrow} (q, x_T)$. Either $q \in Q$ or $q \in S$, we will prove that (q, x_T) is nonblocking in both cases.

In the first case $q \in Q$. From corollary 6.1 it holds that $RA(G,R) \parallel T \stackrel{s}{\Rightarrow} (q, x_T)$ as $RA(G,S) \parallel T \stackrel{s}{\rightarrow} (q, x_T)$. Since (q, x_T) is nonblocking in $RA(G,R) \parallel T$ there must exist a trace $t\omega$ such that $(q, x_T) \stackrel{t\omega}{\Rightarrow}_R$. From corollary 6.1 it holds that $q \stackrel{t\omega}{\Rightarrow}_S$ as $q \stackrel{t\omega}{\Rightarrow}_R$ therefore (q, x_T) must be nonblocking in $RA(G,S) \parallel T$.

In the second case $q \in S$. Let $(x, L) \in S$ be a requirement such that $(x, L) = q$. From definition 6.14 there must exist some requirement $(x, N) \in R$ such that $\psi(x, N) = (x, L)$ therefore $(x, L) = (x, \mathit{prune}(N \cup \{t \in \overline{N} | (x, N) \stackrel{t}{\rightarrow} (y, M) \not\rightarrow (x, N) \text{ for some } (y, M)\}))$. From corollary 6.2 as $RA(G,S) \stackrel{s}{\Rightarrow} ((x, L), x_T)$ it also holds that $RA(G,R) \stackrel{s}{\Rightarrow} ((x, N), x_T)$. From remark 6.1 it holds that $(s, N) \in \text{CC}(RA(G,R))$ as $(x, N) \in R$ and $G \stackrel{s}{\rightarrow} x$. Therefore there must exist some trace $t \in N$ such that $((x, N), x_T) \stackrel{t}{\rightarrow}$ as $RA(G,R) \parallel T$ is nonblocking. As $t \in N$, it must also be the case that $t \in (N \cup \{t \in \overline{N} | (x, N) \stackrel{t}{\rightarrow} (y, M) \not\rightarrow (x, N)$ for some $(y, M)\})$. From definition 6.2, as $t \in (N \cup \{t \in \overline{N} | (x, L) \stackrel{t}{\rightarrow} (y, M) \not\rightarrow (x, N)\})$ there must exist some trace $p \in \mathit{prune}(N \cup \{t \in \overline{N} | (x, L) \stackrel{t}{\rightarrow} (y, M) \not\rightarrow (x, N)\})$ such that $p \sqsubseteq t$. Therefore $((x, L), x_T) \stackrel{p}{\Rightarrow} (y, y_T)$ for some $(y \in Q, y_T \in Q_T)$. We have already proven that (y, y_T) must be nonblocking. Therefore $((x, L), x_T)$ must be nonblocking.

Let T be an automaton such that $RA(G,S) \parallel T$ is nonblocking, we will prove that $RA(G,R) \parallel T$ is also nonblocking.

Let $s \in \Sigma^*$ be a trace such that either there exists (q, x_T) where $RA(G,R) \parallel T \stackrel{s}{\Rightarrow} (q, x_T)$. Either $q \in Q$ or $q \in R$, we will prove that (q, x_T) is nonblocking in both cases.

In the first case $q \in Q$. From corollary 6.1 it holds that $RA(G,S) \parallel T \stackrel{s}{\Rightarrow} (q, x_T)$. Since (q, x_T) is nonblocking in $RA(G,S) \parallel T$ there must exist $q \stackrel{t\omega}{\Rightarrow}_{AS}$. From corollary 6.1 for all $t \in \Sigma^*$ if $q \stackrel{t\omega}{\Rightarrow}_{AS}$ then $q \stackrel{t\omega}{\Rightarrow}_{AR}$ therefore if (q, x_T) is nonblocking in $RA(G,R) \parallel T$ it must also be nonblocking in $RA(G,S) \parallel T$.

In the second case $q \in R$. Let $q = (x, L)$ and let s be a trace such that $RA(G,R) \parallel T \stackrel{s}{\Rightarrow} ((x, L), x_T)$ for some state x_T. We will prove that $((x, L), x_T)$ is nonblocking.

Let $O(x, L) = |\{(y, M) | (x, L) \rightarrow (y, M)\}|$, $O(x, L)$ must be finite if $|R|$ is finite, as R is closed on $\rightarrow$

We will prove the claim via induction on $O(x,L)$

In the base case $O(x,L) = 1$. From definition 6.14 there must exist a requirement $(x,N) \in S$ such that $(x,N) = \psi(x,L)$. Thus $(x,N) = (x, prune(L \cup \{t \in \overline{L} | (x,L) \xrightarrow{t} (y,M) \not\to (x,L)\}))$. For all $t \in N$ it holds that either $t \in L$ or $(x,L) \xrightarrow{t} (y,L/t) \not\to (x,L)$ where $x \xrightarrow{t} y$. From corollary 6.2 it holds that $RA(G,S) \parallel T \stackrel{s}{\Rightarrow} ((x,N),x_T)$ therefore $((x,N),x_T)$ is nonblocking. Thus $((x,N),x_T)$ must be capable of performing at least one trace $t \in N$. Thus it holds that $((x,L),x_T) \xrightarrow{t}$. There are two cases, either $t \in L$ or $t \notin L$. In the first case $((x,L),x_T) \xrightarrow{t} (y,y_T)$ where $x \xrightarrow{t} y$, this state has already been proven nonblocking. In the second case $((x,L),x_T) \xrightarrow{t} (y,L/t),y_T)$ where $x \xrightarrow{t} y$. As $(y,L/t) \not\to (x,L)$ it must hold that $O(y,L/t) < O(x,L)$ as (x,L) can reach all the requirements $(y,L/t)$ can. As $O(x,L) = 1$, $(y,L/t) = 0$. This is absurd as $(suc(x,t),L/t) \to (suc(x,t),L/t)$ therefore t must have been in L.

Let us consider the case where $O(x,L) = n+1$. From the definition 6.14 there must exist a requirement $(x,N) \in S$ such that $(x,N) = \psi(x,L)$. Therefore $(x,N) = (x, prune(L \cup \{t \in \overline{L} | (x,L) \xrightarrow{t} (y,M) \not\to (x,L)\}))$. For all $t \in N$ it holds that either $t \in L$ or $(x,L) \xrightarrow{t} (y,L/s) \not\to (x,L)$ where $x \xrightarrow{t} y$. From corollary 6.2 it holds that $RA(G,S) \parallel T \stackrel{s}{\Rightarrow} ((x,N),x_T)$ therefore $((x,N),x_T)$ is nonblocking. Thus $((x,N),x_T)$ must be capable of performing at least one trace $t \in N$. Thus it holds that $((x,L),x_T) \xrightarrow{t}$. There are two cases, either $t \in L$ or $t \notin L$. In the first case $((x,L),x_T) \xrightarrow{t} (y,y_T)$ where $x \xrightarrow{t} y$ this state has already been proven nonblocking. In the second $((x,L),x_T) \xrightarrow{t} ((y,L/t),y_T)$. As $(y,L/t) \not\to (x,L)$ it must hold that $O(y,L/t) < O(x,L)$ as (x,L) can reach all the requirements $(suc(x,t),L/t)$ can. From the inductive assumption it holds that $(y,L/t)$ must be nonblocking, therefore as $(x,L) \to (y,L/t)$ it must hold that (x,L) is also nonblocking.

□

6.3.2 Requirement Subsumption

The second refinement rule is subsumption. If one nonconflicting requirement is less conflicting than another requirement then the less conflicting requirement is subsumed. If the requirement set R contains two requirements (x,L) and (x,M) such that $(x,L) \lesssim_{\text{conf}} (x,M)$, then the requirement (x,L) can be subsumed by (x,M). This is because all the blocking information implied by (x,L) is also implied by (x,M). All situations which cause (x,L) to block, will also cause (x,M) to block. Simply removing (x,L) will result in a non-well-formed requirement set. However, there is likely to exist at least one requirements $(y,N) \in R$ such that $(y,N) \to (x,L)$. Thus if (x,L) is simply removed from R, the resulting requirement set will no longer be closed under $\to$. For this reason we must also transform the requirement (y,N). To do this we add any trace s, such that $(y,N) \xrightarrow{s} (x,L)$ to the language of N, and then prune N back to the shortest of these

traces. In this way we can be certain that (y,N) can no longer reach (x,L).

Example 6.14 An example of this is shown in figure 6.8. The requirement set R contains the requirements $(1,\{\alpha,\beta\})$ and $(1,\{\alpha\})$. As $(1,\{\alpha,\beta\}) \lesssim_{\text{conf}} (1,\{alpha\})$, we can subsume the requirement $(1,\{\alpha,\beta\})$. At the same time we prune back the requirement $(0,\{\alpha\alpha,\alpha\beta\})$ back to the shortest traces which go through $(1,\{\alpha,\beta\})$ thus getting the requiremnt $(0,\{\alpha\})$. This gives us the requirement set S. We can be certain that this requirement is conflict equivalent with S as while $(0,\{\alpha\}) \lesssim_{\text{conf}} (0,\{\alpha\alpha,\alpha\beta\})$, as soon as $(0,\{\alpha\})$ is satisfied by α the requirement $(1,\{\alpha\})$ is reached. Thus the requirement $(0,\{\alpha\alpha\})$ is implied.

Example 6.15 This will not work for all situations however. Consider the situation where $(x,M) \to (x,L)$. In this case we would have to transform (x,M), thus making it less conflicting. Figure 6.9 shows an example of using this refinement on such an automaton. In this example the requirement $(0,(\beta^*\alpha^+)^*\gamma) \lesssim_{\text{conf}} (0,(\beta^*\alpha^+)^+\gamma)$. However it is not possible to remove $(0,(\beta^*\alpha^+)^*\gamma)$, as to do so we must transform the requirement $(0,(\beta^*\alpha^+)^+\gamma)$ into $(0,\beta^*\alpha)$. Where the original automaton required that the event γ must occur after an indeterminate number of α and β transitions, the new automaton does not require γ to be able to occur at all.

Example 6.16 There are cases however where $(x,M) \to (x,L)$, and it is still possible to remove (x,M). This is the case if for all traces $s \in L$, it holds that $(x,M) \xrightarrow{s}$ does not go through (x,L). If this is the case we can say that while (x,M) will be transformed into a weaker requirement, that weaker requirement will still be stronger than (x,L). Figure 6.10 shows an example of such an automaton. In this automaton $(0,(\beta\alpha)^*\alpha) \lesssim_{\text{conf}} (0,\alpha(\beta\alpha)^*\alpha)$. Furthermore $(0,\alpha(\beta\alpha)^*\alpha)$ is transformed into the requirement $(0,\{\alpha\})$. This requirement is weaker than the requirement $(0,\alpha(\beta\alpha)^*\alpha)$, but stronger than the requirement $(0,(\beta\alpha)^*\alpha)$. Furthermore as satisfying the requirement $(0,\{\alpha\})$ means returning to the state 0 in the trunk, and the state 0 has the requirement $(0,\{\alpha\})$ we can say that the requirement $(0,\{\alpha\alpha\})$ is implied, this requirement is stronger than both requirements.

Because of this we introduce the new stronger relation $<_{\text{conf}}$. In order for the two requirements (x,L) and (x,M) to be $(x,L) <_{\text{conf}} (x,M)$ it must hold that $(x,L) \lesssim_{\text{conf}} (x,M)$ *and* that either (x,M) cannot reach (x,L), or if it can reach (x,L), all traces which can reach (x,L) from (x,M), will also satisfy (x,L).

Example 6.17 Lastly as any test automaton T which wishes to be nonblocking with x will need to be able to perform at least one trace in $\mathscr{L}^\omega(x)$ we can say that the requirement $(x,\mathscr{L}^\omega(x))$ is implied by the trunk. Furthermore these requirements can be

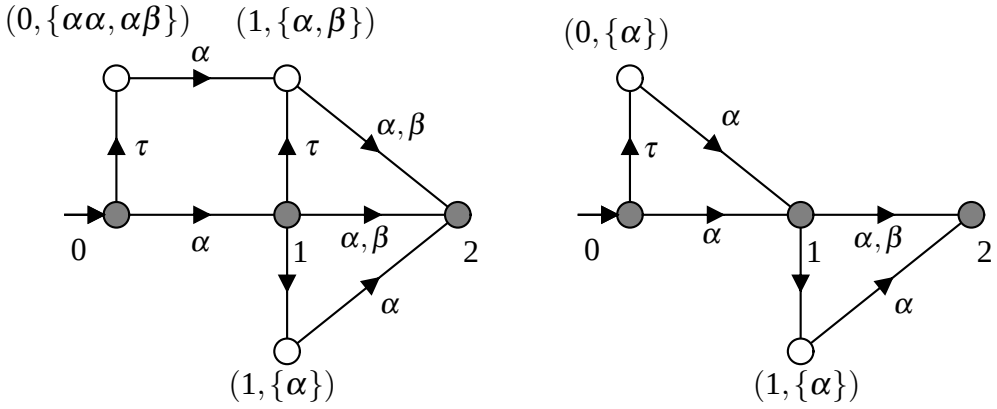


Figure 6.8: An application of requirement subsumption.

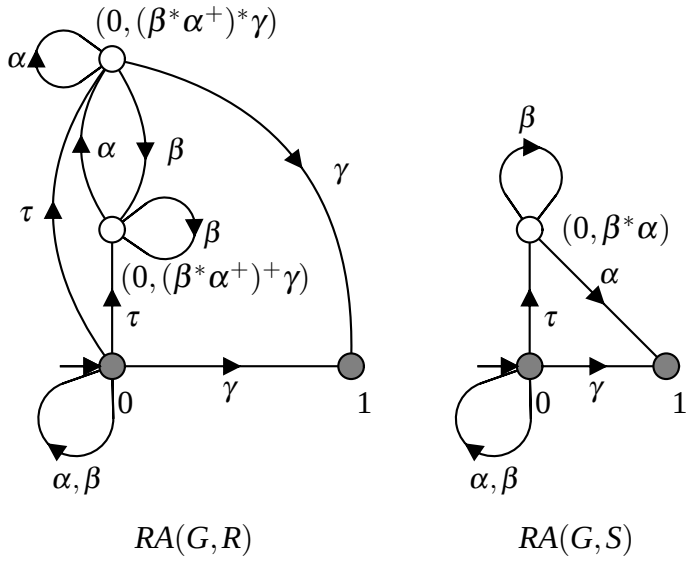


Figure 6.9: An incorrect application of requirement subsumption.

used to subsume explicit requirements. Also as implicit requirements are implied by the trunk which is never modified when we subsume requirements using an implicit requirement we can use the weaker relation between requirement $\lesssim_{\text{conf}}$. Figure 6.11 shows an example of a requirement automaton where a requirement can be subsumed using an implicit requirement. The requirement $(0,\{\alpha,\beta,\omega\}) \lesssim_{\text{conf}} (0,\mathscr{L}^{\omega}(0)=\{\alpha\omega,\beta\omega,\omega\})$. Because of this we can remove $(0,\{\alpha,\beta,\omega\})$ from the requirement set R.

Definition 6.15 Let $G=\langle\Sigma,Q,\rightarrow,Q^{\circ}\rangle$ be a deterministic automaton. Let (x,L) and (x,M) be two requirement pairs of G. Then $(x,L) <_{\text{conf}} (x,M)$ if and only if

- $(x,L) \lesssim_{\text{conf}} (x,M)$.
- For all $s \in \overline{L}-L$, $(\delta(x,s),M/s) \neq (x,L)$

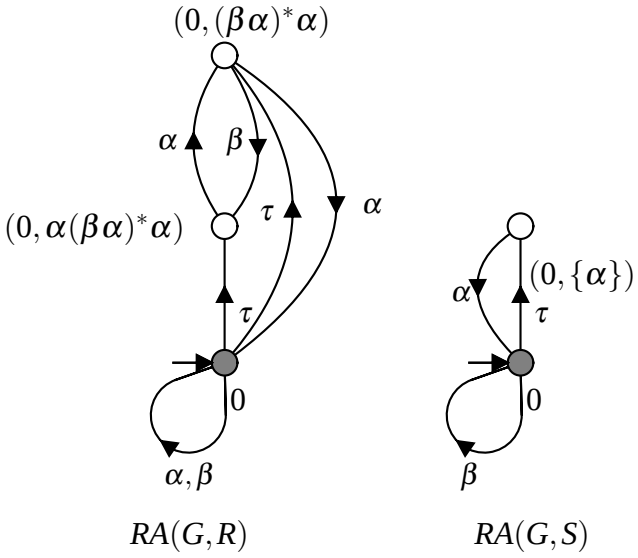


Figure 6.10: An application of requirement subsumption using the $<_{\text{conf}}$ relation.

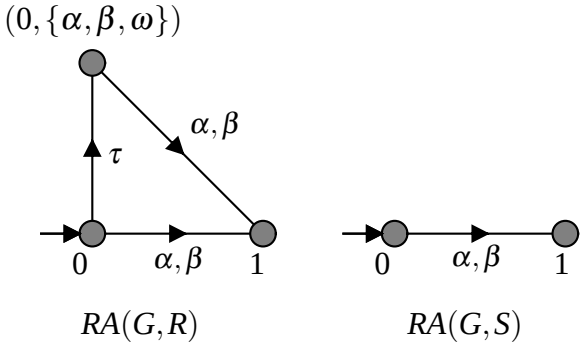


Figure 6.11: An application of requirement subsumption using an implicit requirement.

We also introduce the language $L_{(x,L)}$ of a requirement. $L_{(x,L)}(y,N)$ is the set of all traces such that $(y,N) \rightarrow (x,L)$.

Definition 6.16 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be a deterministic automaton. Let (x,L) and (y,N) be requirement pairs.

$L_{(x,L)}(y,N) = \{s \in \overline{N} | ((y,N) \xrightarrow{s} (x,L)\}$.

We now give the definition of the requirement subsumption rule. A requirement $(x,L) \in R$ is considered to be unnecessary for one of two reasons. The first reason is if there exists a nonconflicting requirement $(x,M) \in R$ such that $(x,L) <_{\text{conf}} (x,M)$ as has been previously discussed. The second is if $(x,L) \lesssim_{\text{conf}} (x,\mathscr{L}^\omega(x))$. This is because the requirement $(x,\mathscr{L}^\omega(x))$ is implied by the trunk. Furthermore in this case we can use the weaker relation $\lesssim_{\text{conf}}$. This is because the requirement $(x,\mathscr{L}^\omega(x))$ is implied by the trunk rather than being explictly referred to in R. Thus there is no way that $(x,\mathscr{L}^\omega(x))$ will be transformed.

Definition 6.17 Let R,S be requirement sets of G.

Let (y,N) be a requirement then $\phi_{(x,L)}(y,N) = \{(y, \mathit{prune}(N \cup L_{(x,L)}(y,N)))\}$.

$\phi_{(x,L)}(R) = \{\phi_{(x,L)}(y,N) | (y,N) \in R \text{ and } (y,N) \neq (x,L)\}$.

$R \succ_\phi S$ if and only if there exists $(x,L) \in R$ such that either there exists $(x,M) \in R$ where $(x,L) <_{\text{conf}} (x,M)$ or $(x,L) \lesssim_{\text{conf}} (x,\mathscr{L}^\omega(x))$ and $S = \phi_{(x,L)}(R)$.

As with the strongly connected requirements rule, we must prove that requirement subsumption produces well-formed requirement sets and that it preserves conflict-equivalence.

The format for proving this is of the same form as for the previous subsection. We first prove that applying ϕ to a single nonconflicting requirement, always results in another nonconflicting requirement. We then show that the ϕ function preserves the $\rightarrow$ relation between nonconflicting requirements. Next we show that applying ϕ to a well-formed requirement set, always results in another well-formed requirement set. Finally we will prove that the requirement automaton of the original requirement set is conflict-equivalent to the requirement automaton of the new requirement set.

Lemma 6.19 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be a deterministic automaton. Let (x,L) and (y,M) be two requirements pairs of G such that $(x,L) \neq (y,M)$.

Then $\phi_{(x,L)}(y,M)$ is a requirement pair of G.

Proof. Let $(y,N) = \phi_{(x,L)}(y,M)$. We will proceed to show that (y,N) satisfies all the conditions of being a requirement given in definition 6.3.

(i) From the definition of *prune*, $N = \mathit{prune}(M \cup L_{(x,L)}(y,M)$ must be prefix-free.

(ii) As (y,M) is a requirement of G it must hold that $M = \emptyset$ if and only if $\mathscr{L}^{\omega}(x) = \emptyset$. We will prove that $N = \emptyset$ if and only if $M = \emptyset$. It is obvious that if $M = \emptyset$ then $N = \emptyset$ as from definition N only contains traces from $\overline{M}$ and $\overline{\emptyset} = \emptyset$. It is equally obvious that if $N = \emptyset$ then $M = \emptyset$. This is because $N = \mathit{prune}(M \cup L(x,L)(y,M)$, it is obvious that $(M \cup L_{(x,L)}(y,M)) \supseteq M$ and *prune* cannot remove every trace in $(M \cup L_{(x,L)}(y,M))$.

(iii) As (y,M) is a requirement of G, $M \subseteq \overline{\mathscr{L}^{\omega}(y)}$ this implies that $\overline{M} \subseteq \overline{\mathscr{L}^{\omega}(y)}$. As N only contains traces which are in $\overline{M}$, $M \subseteq \overline{M} \subseteq \overline{\mathscr{L}^{\omega}(y)}$

(iv) As (y,M) is a requirement $\varepsilon \notin M$. Furthermore $\varepsilon \notin L_{(x,L)}(y,M)$as $(y,M) \xrightarrow{\varepsilon} (y,M)$ and $(y,M) \neq (x,L)$ from assumption. As $\varepsilon \notin M \cup L_{(x,L)}(y,M))$ it must not be in $\mathit{prune}(M \cup L_{(x,L)}(y,M))$ either. □

We now show that the ϕ function preserves the $\rightarrow$ relation. This principally has the same meaning as lemma 6.17 has for the strongly connected requirements relation. Again we are merely showing that if a transformed requirement (x,M) can transition to another requirement (y,O), then it must also be the case that the original requirement (x,L) can reach a requirement (y,N) such that (y,N) will be transformed into (y,O).

Lemma 6.20 Let $G = \langle \Sigma, Q, \rightarrow, Q^{\circ} \rangle$ be a deterministic automaton. Let $(x,L),(y,M)$, (y,N) be three requirements pairs of G such that $\phi_{(x,L)}(y,M) = (y,N)$.

Let $s \in \overline{N}$ be a trace.

Then $\phi_{(x,L)}(\delta(y,s),M/s) = (\delta_G(y,s),N/s)$

Proof. Let $(\delta_G(y,s),O) = \phi_{(x,L)}(\delta(y,s),M/s)$. We will prove that $O = N/s$.

$$
\begin{aligned}
N/s &= \mathit{prune}(M \cup L_{(x,L)}(y,M))/s \\
&= \mathit{prune}(M/s \cup L_{(x,L)}(y,M)/s) && \text{from lemma 6.1} \\
&= \mathit{prune}(M/s \cup \{t \in \overline{M} | (\delta(y,t),M/t) = (x,L)\}/s) && \text{from definition 6.17} \\
&= \mathit{prune}(M/s \cup \{st \in \overline{M} | (\delta(y,st),M/st) = (x,L)\}) \\
&= \mathit{prune}(M/s \cup \{t \in \overline{M/s} | (\delta(y,st),M/st) = (x,L)\}) \\
&= \mathit{prune}(M/s \cup L_{(x,L)}(\delta_G(y,s),M/s)) \\
&= O && \text{from definition 6.17}
\end{aligned}
$$

□

Again we can show that for any well-formed requirement set, if we subsume a requirement, then the resulting requirement set will also be well-formed. This ensures

that we can continually apply the strongly connected requirements rule and the requirement subsumption rule without problems.

Lemma 6.21 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be a deterministic automaton. Let R be a requirement set of G. Let $(x,L) \in R$, such that either there exists $(x,M) \in R$ where $(x,L) <_{\text{conf}} (x,M)$ or $(x,L) \lesssim_{\text{conf}} (x, \mathscr{L}^\omega(x))$.

Then $\phi_{(x,L)}(R)$ is a requirement set of G

Proof. Let (y,N) be a pair in $\phi(x,L)(R)$. From definition 6.17 there exists $(y,M) \neq (x,L) \in (R)$ such that $\phi_{(x,L)}(y,M) = (y,N)$.

(i) Lemma 6.16 shows that (y,N) must be a requirement of G as $\phi_{(x,L)}(y,M) = (y,N)$.

(ii) Next we will prove that for all (z,P) such that $(y,N) \xrightarrow{s} (z,P)$ for some s, it must hold that $(z,P) \in \psi(R)$.

As $(y,N) \xrightarrow{s}$ it holds that $s \in \overline{M} - M$. Furthermore as M only contains traces in $\overline{M}$ it must also hold that $s \in \overline{M}$. Thus there exists (z,O) such that $(y,M) \xrightarrow{s} (z,O)$. As R is a requirement set of G it holds that $(z,O) \in R$. From lemma 6.17 as $(y,M) \xrightarrow{s} (z,O)$ and $(y,N) \xrightarrow{s} (z,P)$, it must also hold that $\psi(z,O) = (z,P)$. Therefore (z,P) must be in $\psi(R)$. □

Finally we show that requirement subsumption preserves conflict-equivalence. That is that for any given trunk automaton G and any two requirement sets R and S of that trunk automaton such that $R \succ_\phi S$ it holds that $RA(G,R) \simeq_{\text{conf}} RA(G,S)$. This ensures that repeated uses of the strongly connected requirements rule and of the requirement subsumption rule always result in requirement automata in the same equivalence class.

Theorem 6.4 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be a deterministic automaton. Let R,S be two requirement sets of G, such that $R \succ_\phi S$.

Then $RA(G,R) \simeq_{\text{conf}} RA(G,S)$.

Proof. As $R \succ_\phi S$ there must exist a requirement tuple $(w,J) \in R$ such that either $(w,J) \lesssim_{\text{conf}} (w, \mathscr{L}^\omega(w))$ or there exists $(w,K) \in R$ where $(w,J) <_{\text{conf}} (w,J)$, and $S = \phi_{(w,J)}(R)$

Let T be an automaton such that $RA(G,R) \parallel T$ is nonblocking, let us prove that $RA(G,S) \parallel T$ is also nonblocking.

Let $s \in \Sigma^*$ be a trace and $(q,x_T) \in S$ be a requirement where $RA(G,S) \parallel T \overset{s}{\Rightarrow} (q,x_T)$. Either $q \in Q$ or $q \in S$.

In the first case $q \in Q$. From corollary 6.1 it holds that $RA(G,R) \parallel T \overset{s}{\Rightarrow} (q,x_T)$. Furthermore from corollary 6.1 for all $t \in \Sigma^*$ if $q \overset{t\omega}{\Rightarrow}_{AR}$ then $q \overset{t\omega}{\Rightarrow}_{AS}$ therefore if (q,x_T) is nonblocking in $RA(G,R) \parallel T$ it must also be nonblocking in $RA(G,S) \parallel T$.

In the second case $q \in S$. Let $q = (x,L)$. From definition 6.17 there exists a requirement in $(x,M) \in R$ such that $\phi(x,M) = (x,L) = (x, prune(M \cup L_{(w,J)}(x,M)))$. From corollary 6.1 it holds that $RA(G,R) \parallel T \stackrel{s}{\Rightarrow} ((x,M),x_T)$. From remark 6.1 it holds that $(s,M) \in \mathrm{CC}(RA(G,R))$ as $(x,M) \in R$ and $G \stackrel{s}{\rightarrow} x$. Therefore x_T can perform some trace $t \in M$ as $RA(G,R) \parallel T$ is nonblocking. As $t \in M \cup L_{(w,J)}(x,M)$ from definition 6.2 there must exist some trace $p \in prune(M \cup L_{(w,J)}(x,M)$ such that $p \sqsubseteq t$. $((x,L),x_T) \stackrel{p}{\Rightarrow} (\delta_G(x,p),\delta_T(x_T,p))$. This state has already been proven nonblocking.

Let T be an automaton such that $RA(G,S) \parallel T$ is nonblocking, we will prove that $RA(G,R) \parallel T$ is also nonblocking.

Let (q,q_T) be a state and s be a trace such that $RA(G,R) \parallel T \stackrel{s}{\rightarrow} (q,x_T)$.

There are three cases. First $q \in Q$, second $q = (w,J)$, and third $q \in R$.

First we will consider the case where $q \in Q$. From corollary 6.1 as $RA(G,R) \stackrel{s}{\Rightarrow} (q,x_T)$ it holds that $RA(G,S) \parallel T \stackrel{s}{\Rightarrow} (q,x_T)$. Furthermore from corollary 6.1 for all $t \in \Sigma^*$ if $q \stackrel{t\omega}{\Rightarrow}_{AS}$ then $q \stackrel{t\omega}{\Rightarrow}_{AR}$ therefore as (q,x_T) is nonblocking in $RA(G,R) \parallel T$ it must also be nonblocking in $RA(G,S) \parallel T$.

Second we consider the case where $q = (w,J)$. From the definition of $\succ_\phi$, either $(w,J) \lesssim_{\mathrm{conf}} (w,\mathscr{L}^\omega(w))$ or there exists a requirement $(w,K) \in R$ such that $(w,J) <_{\mathrm{conf}} (w,K)$. In the first case $(w,J) \lesssim_{\mathrm{conf}} (w,\mathscr{L}^\omega(w))$. From lemma 6.5 it holds that $RA(G,R) \parallel T \stackrel{s}{\rightarrow} (w,x_T)$ as $RA(G,R) \stackrel{s}{\rightarrow} ((w,J),x_T)$. As this state has already been proven nonblocking there exists a trace $t\omega \in \mathscr{L}^\omega(w)$ such that $(w,x_T) \stackrel{t\omega}{\rightarrow}$. From definition 6.4 as $(w,J) \lesssim_{\mathrm{conf}} (w,\mathscr{L}^\omega(w)$ it holds that $\mathscr{L}^\omega(w) \subseteq J\Sigma^*_\omega$, therefore there must exist some trace $p \sqsubseteq t\omega$ such that $p \in J$. Therefore $((w,J),x_T) \stackrel{p}{\Rightarrow} (\delta_G(w,p),\delta_T(x_T,p))$. The state $(\delta_G(w,p),\delta_T(x_T,p))$ has already been proven nonblocking. In the second case there exists a requirement $(w,K) \in R$ such that $(w,J) <_{\mathrm{conf}} (w,K)$. From definition 6.17 it holds that $\phi_{(w,J)}(R) = S$, $\phi_{(w,J)}(w,K) \in S$. From definition 6.17 it holds that $\phi_{(w,J)}(w,K) = (w, prune(K \cup L_{(w,J)}(w,K)))$. Let $(\phi_{(w,J)}(w,K),x_T) = (w,I)$ from corollary 6.2 it holds that $RA(G,S) \parallel T \stackrel{s}{\rightarrow} ((w,I),x_T)$ as $RA(G,R) \parallel T \stackrel{s}{\rightarrow} ((w,J),x_T)$. As $((w,I),x_T)$ is nonblocking there must exist a trace $t \in I$ such that $((w,I),x_T) \stackrel{t}{\rightarrow}$. From definition 6.17 it holds that $I = prune(K \cup L_{(w,J)}(w,K))$, further from definition 6.2 it holds that $prune(K \cup L_{(w,J)}(w,K)) \subseteq K \cup L_{(w,J)}(w,K)$, therefore t must be an element of either K or $L_{(w,J)}(w,K)$. We will first consider the case where $t \in K$. As $(w,J) \lesssim_{\mathrm{conf}} (w,K)$ it holds that $K \subseteq J\Sigma^*_\omega$ therefore there exists $p \in J$ such that $p \sqsubseteq t$. Therefore $((w,J),x_T) \stackrel{p}{\Rightarrow} (\delta_G(w,p),\delta_T(x_T,p))$ as $((w,I),x_T) \stackrel{t}{\Rightarrow}$. The state $(\delta_G(w,p),\delta_T(x_T,p))$ has already been proven nonblocking. In the second case $t \in L_{(w,J)}(w,K) = \{u \in \overline{K} | (\delta(w,u),K/u) = (w,J)\}$. Therefore $t \in \overline{K}$ and $(\delta(w,t),K/t) = (w,J)$. As $(w,J) \lesssim_{\mathrm{conf}} (w,K)$, it holds that $K \subseteq J\Sigma^*_\omega$, which further implies that $\overline{K} \subseteq \overline{J}\Sigma^*_\omega$, therefore $t \in \overline{J}\Sigma^*_\omega$. From the definition of $<_{\mathrm{conf}}$, if $t \in \overline{J} - J$ then $(\delta(w,t),K/t) \neq (w,J)$, therefore $t \notin \overline{J} - J$. Therefore $t \in \overline{J}\Sigma^*_\omega - \overline{J} + J = J\Sigma^*_\omega$. As $t \in J\Sigma^*_\omega$ there exists $p \in J$ such that $p \sqsubseteq t$. There-

fore $((w,J),x_T) \stackrel{p}{\Rightarrow} (\delta_G(w,p),\delta_T(x_T,p))$. The state $(\delta_G(w,p),\delta_T(x_T,p))$ has already been proven nonblocking.

Thirdly we consider the case where $q \in R$. Let $(x,L) = q$. From definition 6.17 there exists a requirement $(x,M) \in S$ such that $\phi_{w,J}(x,L) = (x,M)$. From assumption $RA(G,R) \parallel T \stackrel{s}{\Rightarrow} ((x,L),x_T)$, therefore from corollary 6.2 it holds that $RA(G,S) \parallel T \stackrel{s}{\Rightarrow} ((x,M),x_T)$. As $((x,M),x_T)$ is nonblocking there must exist a trace $t \in M$ such that $((x,M),x_T) \stackrel{t}{\rightarrow}$. From definition 6.17 as $\phi_{w,J}(x,L) = (x,M)$ it holds that $(x,M) = (x,prune(L \cup L_{(w,J)}(x,L)))$. As $prune(L \cup L_{(w,J)}(x,L)) \subseteq L \cup L_{(w,J)}(x,L)$ the trace t must either be in L or in $L_{(w,J)}(x,L)$. In the first case as $t \in L$ it holds that $((x,L),x_T) \stackrel{t}{\Rightarrow} (\delta_G(x,t),\delta_T(x_T,t))$. The state $(\delta_G(x,t),\delta_T(x_T,t))$ has already been proven nonblocking. In he second case $t \in L_{(w,J)}(x,L) = \{u \in \overline{L} | (\delta(x,u),L/u) = (w,J)\}$. Thus $(x,L) \stackrel{t}{\rightarrow} (w,J)$, therefore $((x,L),x_T) \stackrel{t}{\Rightarrow} ((w,J),\delta_T(x_T,t))$. The state $((w,J),\delta_T(x_T,t))$ has already been proven nonblocking. □

6.4 Irreducible Requirement Sets

Using the refinement rules introduced in section 6.3 it is possible to refine any finite requirement set into an irreducible requirement set. This section is divided into two subsections. Subsection 6.4.1 first defines what it means to be an irreducible requirement set, then proves that any finite requirement set can be reduced to an irreducible requirement set, it then finally describes a few properties of an irreducible requirement set. Subsection 6.4.2 shows that there is a unique requirement set for every conflict equivalence class.

6.4.1 Properties

In the previous section we described the strongly connected component rule as well as the requirement subsumption rule. We further showed that each successive application of these rules resulted in well-formed requirement sets, as well as conflict-equivalent requirement automata. In this subsection we will show that these two rules can be used iteratively in order to reach an irreducible requirement set. We will further go on to show that given a particular trunk automaton G, there exists a unique irreducible requirement set for each conflict-equivalence class.

We first define the relation $\succ$ between requirement sets which are found by successive application of the refinement rules described in section 6.3.

Definition 6.18 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an automaton. Let R and S be two requirement sets of G.

Then $R \succ S$ if and only if one of the following holds.

(i) $R \succ_{\psi} S$.

(ii) $R \succ_{\phi} S$.

(iii) If there exists some requirement set T such that $R \succ T \succ S$.

Lemma 6.22 Let $G = \langle \Sigma, Q, \rightarrow, Q^{\circ} \rangle$ be an automaton. Let R and S be two requirement sets of G such that $R \succ S$.

Then $RA(G,R) \simeq_{\text{conf}} RA(G,S)$.

Proof. This can be proved using induction based upon theorems 6.3 and 6.4 □

We further define what it means for a requirement set R to be irreducible.

Definition 6.19 Let $G = \langle \Sigma, Q, \rightarrow, Q^{\circ} \rangle$ be a deterministic automaton. Let R be a well-formed requirement set with respect to G.

R is ψ-irreducible if and only if for every requirement set S of G, if $R \succ_{\psi} S$ then $R = S$.

R is ϕ-irreducible if and only if for every requirement set S of G, if $R \succ_{\phi} S$ then $R = S$.

R is irreducible if and only if R is both ψ-irreducible and ϕ-irreducible.

Example 6.18 If we use requirement automata to represent requirement sets, figure 6.12 gives an example of how the strongly connected requirement rule and requirement subsumption can be used to simplify a requirement set until it is irreducible. The example starts with the requirement set R. R can be simplified using the strongly connected requirements rule into the requirement set S as previously shown in example 6.12. After this the requirement $(1,(\alpha\alpha)^*(\beta|\omega))$ can be subsumed from S. This is because $(1,(\alpha\alpha)^*(\beta|\omega)) <_{\text{conf}} (1,\{\beta,\omega\})$. Thus S can be simplified into T by removing $(1,(\alpha\alpha)^*(\beta|\omega))$ and pruning $(0,\alpha(\alpha\alpha)^*(\beta|\omega))$ back to the requirement $(0,\alpha)$. Lastly the requirement $(2,\beta^*\omega)$ can also be subsumed, this time using the implied requirement $(2,\mathscr{L}^{\omega}(2))$ which is equal to $(2,\beta^*\omega)$. While $(2,\beta^*\omega) \not<_{\text{conf}} (2,\mathscr{L}^{\omega}(2) = \beta^*\omega)$ as they both equal one another, as $(2,\mathscr{L}^{\omega}(2))$ is a requirement which is implied by the trunk, we can use the weaker relation $\lesssim_{\text{conf}}$ to subsume $(2,\beta^*\omega)$. This transforms T into the requirement set V which is irreducible.

Furthermore we can be certain that $RA(G,R) \simeq_{\text{conf}} RA(G,S) \simeq_{\text{conf}} RA(G,T) \simeq_{\text{conf}} RA(G,V)$. As from theorem 6.3 it holds that $RA(G,R) \simeq_{\text{conf}} RA(G,S)$ and from theorem 6.4 it holds that $RA(G,S) \simeq_{\text{conf}} RA(G,T)$ and $RA(G,T) \simeq_{\text{conf}} RA(G,V)$.

In order to show that successive applications of the refinement rules of section three result in an irreducible requirement set we will show that at each step the requirement rules make the requirement set smaller. To do this we first show that successive applications of the strongly connected requirements rule R have no effect. That is to say if R has been simplified using the strongly connected requirements rule once it cannot be simplified using the strongly connected requirements rule again. In order for it to be possible to use the strongly connected requirements rule again it is first neccessary to apply the requirement subsumption rule. Thus a requirement set simplified by the strongly connected components rule can be considered to be smaller with respect to whether it can be simplified by ψ.

Lemma 6.23 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be a deterministic automaton. Let R, R', R'' be three requirement sets of G such that $R \succ_\psi R' \succ_\psi R''$

Then $R' = R''$

Proof. Let (x,L) be a requirement in R' such that $L \neq \{\varepsilon\}$. We will show for all traces $s \in \overline{L}$ that $(\delta(x,s), L/s) \rightarrow (x,L)$ if and only if $L/s \neq \{\varepsilon\}$. Let $s \in \overline{L}$. We will first show that if $(\delta(x,s), L/s) \rightarrow (x,L)$ then $L/s \neq \{\varepsilon\}$. This is simple as if $L/s = \{\varepsilon\}$, $(\delta(x,s), L/s)$ can only reach $\delta(x,s), \{\varepsilon\})$. Second we will show that if $L/s \neq \{\varepsilon\}$ then $(\delta(x,s), L/s) \rightarrow (x,L)$. From the construction of R' there exists a requirement pair (x,M) such that $L = prune(\{t \in \overline{M} | (\delta_G(x,t), M/t) \not\rightarrow (x,M)\})$. As $L/s \neq \{\varepsilon\}$, $L = prune(\{t \in \overline{M} | (\delta_G(x,t), M/t) \not\rightarrow (x,M)\})$, and as L is prefix-free $\delta_G(x,s), M/s) \xrightarrow{u} (x,M)$ for some u. The trace $su \in \overline{L}$. Furthermore $\phi(x,M) = (x,L)$ therefore from lemma 6.20 $\psi(\delta_G(x,su), M/su) = (\delta_G(x,su), L/su)$. As $\delta_G(x,su), M/su = (x,M)$ and $\psi(x,M) = (x,L)$ it follows that $(\delta_G(x,su), L/su) = (x,L)$, thus $(\delta(x,s), L/s) \xrightarrow{u} (x,L)$. $(\delta(x,s), L/s) \rightarrow (x,L)$ if and only if $L/s \neq \{\varepsilon\}$ is equivalent to $(\delta(x,s), L/s) \not\rightarrow (x,L)$ if and only if $L/s = \{\varepsilon\}$.

Let (x,L) be a requirement in R'. We will show that $\psi(x,L) = (x,L)$. In the first case $L = \{\varepsilon\}$. In this case $\psi(x,L) = (x,L)$ directly from the definition of ψ. In the second case $\psi(x,L) = (x, prune(\{s \in \overline{L} | (\delta(x,s), L/s) \not\rightarrow (x,L)\}))$. As for all $s \in \overline{L}$ $(\delta(x,s), L/s) \not\rightarrow (x,L)$ if and only if $L/s = \{\varepsilon\}$. $\psi(x,L) = (x, prune(\{s \in \overline{L} | L/s = \{\varepsilon\}))$ as L is prefix-free $L/s = \{\varepsilon\}$ if and only if $s \in L$ therefore $\psi(x,L) = (x, prune(L))$. Again because L is prefix-free $prune(L) = L$ thus $\psi(x,L) = (x,L)$.

Lastly we will prove that $R' = R''$. From assumption $\psi(R') = R''$ therefore it is the case that $\{\psi(x,L) | (x,L) \in R'\} = R''$. As $\psi(x,L) = (x,L)$ $\{(x,L) | (x,L) \in R'\} = R' = R''$. □

We can further show that applying the strongly connected requirements rule to a requirement set R will never increase the number of requirements in R. Thus the strongly

connected requirement rule never makes R bigger. Thus applying the strongly connected requirement rule will always result in a requirement set which is either smaller than or equal to R with respect to the number of requirements it contains.

Lemma 6.24 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be a deterministic automaton. Let R, R' be two requirement sets of G such that $R \succ_\psi R'$.

Then $|R| \geq |R'|$

Proof. This come directly from the definition of ψ. $\psi(R)$ creates at most one tuple for each pair in the original requirement set R. □

We now show that each application of the requirement subsumption rule strictly reduces the number of requirements in R.

Lemma 6.25 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be a deterministic automaton. Let R, R' be two requirement sets of G such that $R \succ_\phi R'$

Then $|R| > |R'|$

Proof. This also comes directly from the definition of ϕ as R' can only have at most one requirement pair for each pair in R excluding the requirement (x, L). As such $|\phi_{(x,L)}(R)|$ can have at most $|R| - 1$ elements. □

We now define a function with which we can compare the size of two requirement sets. The function $||.||$ assigns a number to each requirement set such that requirement sets are ordered primarily based upon the number of nonconflicting requirements they contain and secondarily based upon whether they can be refined by the strongly connected requirements rule.

Definition 6.20 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be a deterministic automaton. Let R a requirement set of G.

Then $||R|| = 2|R|$ if $R \succ_\phi R$ or otherwise $||R|| = 2|R| + 1$.

We now show that whenever either the strongly connected requirements rule or the requirement subsumption rule is applied to a requirement set R, then the resulting requirement set is always smaller with respect to $||.||$. As $||.||$ cannot be negative, we can therefore say that for any finite requirement set R, a finite number of applications of the strongly connected requirements rule or the requirement subsumption rule will lead to an irreducible requirement set.

Theorem 6.5 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be a deterministic automaton. Let R and S be two requirement sets of G such that $R \succ S$ and $R \neq S$.

Then $||R|| > ||S||$.

Proof. There are two cases either $R \succ_\phi S$ or $R \succ_\psi S$. In the first case because $R \neq S$ and $R \succ_\phi S$ it must be the case that $||R|| = 2|R| + 1$. From lemma 6.24 it holds that $|R| \geq |S|$ and from 6.23 $S \succ_\phi S$, therefore $||S|| \leq 2|R|$, and the proposition is proven.

In the second case $||R||$ either equals $2|R| + 1$ or $2|R|$, thus $||R|| \geq 2|R|$, and $||S||$ either equals $2|S| + 1$ or $2|S|$, thus $||S|| \leq 2|S| + 1$. From lemma 6.25 it holds that $|R| > |S|$, therefore $||S|| \leq 2(|R| - 1) + 1$, which is equivalent to $||S|| \leq 2|R| - 1$. Thus the case is proven. □

Example 6.19 Consider the requirement automata shown in figure 6.12. In this example the requirement set R is successively refined using the strongly connected requirement rule and the requirement subsumption rule until it eventually results in the irreducible requirement set V. The first requirement set R has five requirements and can be reduced by the strongly connected requirement rule, thus $||R|| = 2 \times 5 + 1 = 11$. R is then refined using the strongly connected requirement rule into the requirement set S. This requirement set still has exactly five requirements but can no longer be reduced by the strongly connected requirement rule, thus $||S|| = 10$. The requirement $(1, (\alpha\alpha)^*(\beta|\omega))$ is then subsumed by $(1, \{\beta, \omega\})$. This results in the requirement set T. T has four requirements in it and also cannot be refined by the strongly connected requirements rule, thus $||T|| = 8$. Finally we subsume the nonconflicting requirement $(2, \beta^*\omega)$ using implied requirement $(2, \mathscr{L}^\omega(2))$. This results in the requirement set V which has three requirements. $||V|| = 6$. V cannot no longer be refined by either the strongly connected requirements rule or the requirement subsumption rule, thus it is considered irreducible. We further notice that after each step $||.||$ decreases in size.

Lastly we give some properties of irreducible requirement sets. The first property applies to requirement sets which are irreducible with respect to the strongly connected requirement rule. If the requirement set R is ψ irreducible then for every requirement (x, L) and (y, M) in R such that $(x, L) \to (y, M)$ it holds that (y, M) and (x, L) are strongly connected.

Lemma 6.26 Let $G = \langle \Sigma, Q, \to, Q^\circ \rangle$ be an automaton, and let R be a $\psi-$irreducible requirement set of G. Let (x, L) and (y, M) be two requirements in R such that $(x, L) \to (y, M)$.

It holds that $(y, M) \to (x, L)$

Proof. As $(x, L) \to (y, M)$ there must exist a trace s such that $(x, L) \xrightarrow{s} (y, M)$. As R is R irreducible, $\psi(R) = R$, therefore from definition 6.14 there must exist a requirement $(x, N) \in R$ such that $\psi(x, N) = (x, L)$. From lemma 6.17 it must hold that there exists a nonconflicting requirement $(y, O) \in R$ such that $(x, N) \xrightarrow{s} (y, O)$ and $\psi(y, O) = (y, M)$.

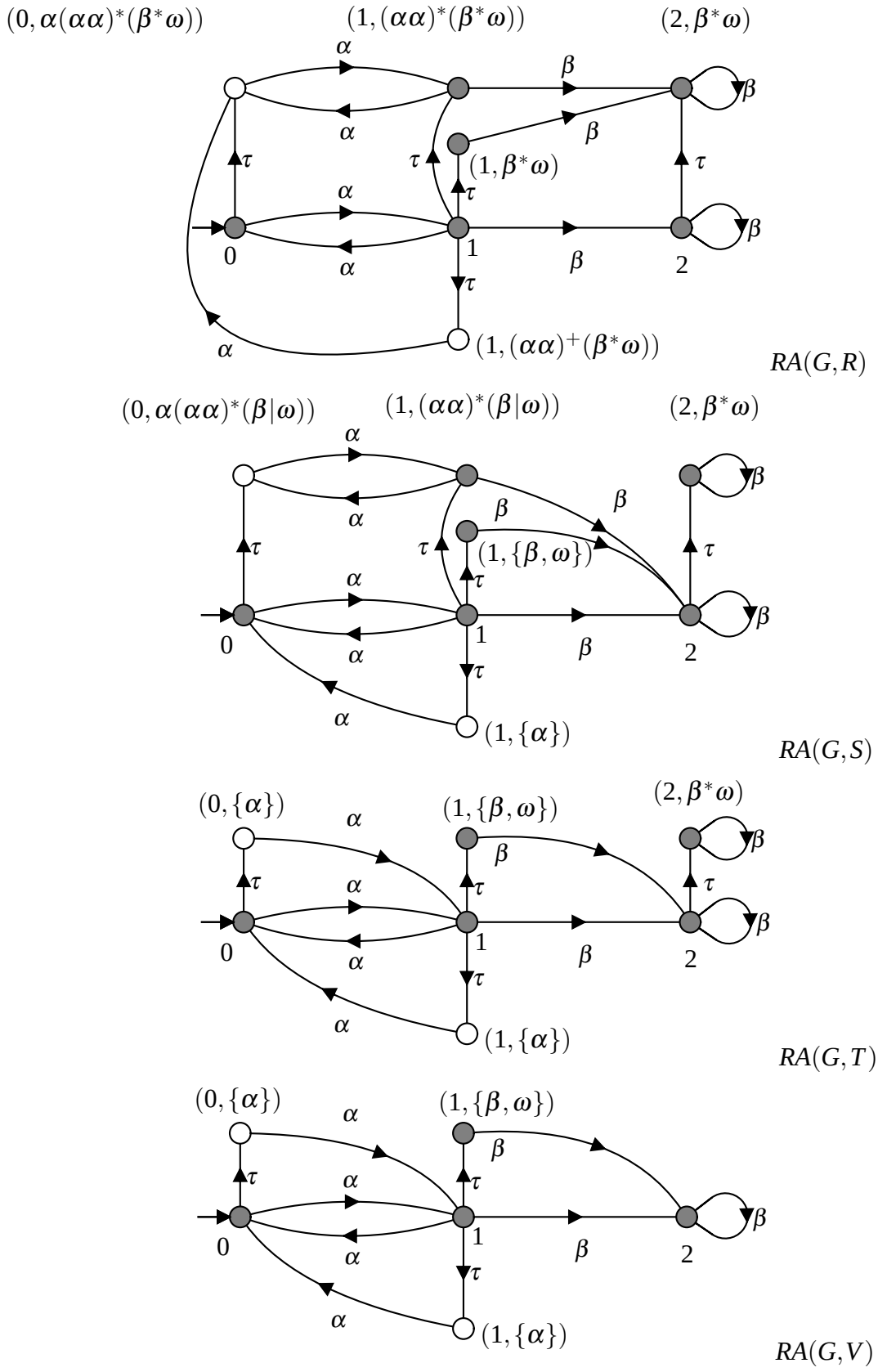


Figure 6.12: Example of four automata such that $R \succ_\psi S \succ_\phi T \succ_\phi V$

As $(x,L) \xrightarrow{s} (y,M)$, it holds that $s \notin L$, and as L is prefix-free it holds that there exists no trace $p \sqsubseteq s$ such that $p \in L$. Furthermore as $s \notin L$, $\psi(x,N) = (x,L)$, and $(x,N) \xrightarrow{s} (y,O)$ it holds that $(y,O) \xrightarrow{t} (x,N)$ for some trace t. As $(y,O) \xrightarrow{t}$ there exists a trace $u \in O$ such that $t \sqsubseteq u$. Furthermore as $(x,N) \xrightarrow{s} (y,O)$ all possible traces p and v such that $pv = t$ it is the case that $(y,O) \xrightarrow{p} (.,.) \xrightarrow{v} (x,L) \xrightarrow{s} (y,O)$, therefore $(y,M) \xrightarrow{t}$ to some requirement (x,P), which from lemma 6.17 is equal to $\psi(x,N) = (x,P) = (x,L)$. □

We next show that for any requirement set R which is $\phi-$irreducible, if R contains the nonconflicting requirement (x,L) then there exists no nonconflicting requirement $(x,M) \in R$ such that (x,M) is strictly more conflicting than (x,L).

Lemma 6.27 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an automaton, and let R be a $\phi-$irreducible requirement set of G. Let (x,L) and (x,M) be two requirements in R.

$(x,L) \not<_{\text{conf}} (x,M)$

Proof. Let us assume that $(x,L) <_{\text{conf}} (x,M)$. Then according to definition 6.17 it holds that $R \succ_\phi \phi_{(x,L)}(R)$. From lemma 6.25 it holds that $|\phi_{(x,L)}(R)| < |R|$ therefore $\phi_{(x,L)}(R) \neq R$, thus R is not $\phi-$irreducible. This contradicts our assumptions, thus $(x,L) \not<_{\text{conf}} (x,M)$. □

We further show that for any requirement set R which is $\psi-$irreducible, if R has the nonconflicting requirement (x,L), then (x,L) is not less conflicting than $(x,\mathscr{L}^\omega(x))$.

Lemma 6.28 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an automaton, and let R be a $\phi-$irreducible requirement set of G. Let (x,L) be a requirement in R.

$(x,L) \not\lesssim_{\text{conf}} (x,\mathscr{L}^\omega(x))$

Proof. Let us assume that $(x,L) \lesssim_{\text{conf}} (x,\mathscr{L}^\omega(x))$. Then according to definition 6.17 it holds that $R \succ_\phi \phi_{(x,L)}(R)$. From lemma 6.25 it holds that $|\phi_{(x,L)}(R)| < |R|$ therefore $\phi_{(x,L)}(R) \neq R$, thus R is not $\phi-$irreducible. This contradicts our assumptions, thus $(x,L) \not\lesssim_{\text{conf}} (x,\mathscr{L}^\omega(x))$. □

6.4.2 Uniqueness

In this section we show that their is a unique requirement set for each conflict equivalence class. In addition to this we will show that for any given trunk automaton G, and any two irreducible nonconflicting requirement sets R and S, $RA(G,R) \simeq_{\text{conf}} RA(G,S)$ if and only if $R = S$. The proof for this is quite involved and is split between several different lemmas. Here we will give an overview of the proof so that the reader can have a reasonable understanding about the importance of each individual lemma. Theorem 6.6 shows that every nonconflicting requirement $(x,K) \in R$ is also in S. To do

this we consider a nonconflicting requirement (x,L) in R such that there exists no requirement in R which is $\gtrsim_{\text{conf}} (x,L)$, and $(x,K) \lesssim_{\text{conf}} (x,L)$. Because R is an irreducible requirement set, $(x,K) \not<_{\text{conf}} (x,L)$ otherwise (x,K) would have been removed, and thus $(x,K) \to (x,L)$. Furthermore this implies that $(x,L) \to (x,K)$ again as R is irreducible any requirements which are not strongly connected will have been abstracted. As this is the case any well-formed nonconflicting requirement set which contains (x,L) must also contain (x,K), therefore it is enough to show that $(x,L) \in S$.

The picture in figure 6.15 gives a graphical representation of how we find a requirement $(x,O) \in S$ such that $(x,L) \lesssim_{\text{conf}} (x,O)$. Given our nonconflicting requirement (x,L) we first use lemma 6.29 in order to find the requirements (y,L') and (y,O') and the trace s such that $(x,L) \xrightarrow{s} (y,L')$ and $(y,L') \lesssim_{\text{conf}} (y,O')$. We then construct the nonconflicting requirement (y,M') which is an element of neither R or S such that $M' = L' \cap \overline{O'}$, this is a nonconflicting requirement which has the property that $(y,L') \lesssim_{\text{conf}} (y,M') \lesssim_{\text{conf}} (y,O')$. We use lemma 6.29 on this to find the trace t and the requirements (z,M'') and (z,N'') such that $(y,M') \xrightarrow{t} (z,M'')$ and $(z,M'') \lesssim_{\text{conf}} (z,N'')$. We further state that there exists a requirement (z,L'') such that $(y,L') \xrightarrow{t} (z,L'')$ and $M'' \subseteq L''$. To this we apply the lemma 6.31 to find the nonconflicting requirement $(z,L'/tu)$ which all three of these requirements converge upon after the trace u. As R is irreducible and $(x,L) \xrightarrow{stu} (z,L'/tu)$ there exists a trace v such that $(z,L'/tu) \xrightarrow{v} (x,L)$. Furthermore as $(y,M') \xrightarrow{tu} (x,L)$ and $(y,M') \lesssim_{\text{conf}} (y,O')$, $(y,O') \xrightarrow{tu} (x,O)$ such that $(x,L) \lesssim_{\text{conf}} (x,O)$.

We can use the same method we used to find $(x,O) \in S$ such that $(x,L) \lesssim_{\text{conf}} (x,O)$, to find another requirement $(x,P) \in R$ such that $(x,O) \lesssim_{\text{conf}} (x,P)$. Thus $(x,L) \lesssim_{\text{conf}} (x,O) \lesssim_{\text{conf}} (x,P)$, as $\lesssim_{\text{conf}}$ is transitive, and the only nonconflicting requirement in R which is more conflicting than (x,L) is (x,L), $(x,P) = (x,L)$. Furthermore as $\lesssim_{\text{conf}}$ is antisymmetric $(x,O) = (x,L)$. Therefore $(x,L) \in S$. An identical argument can be made for why every nonconflicting requirement in S must also be contained in R.

Lemma 6.29 Let $G = \langle \Sigma, Q, \to, Q^\circ \rangle$ be a deterministic automaton. Let R,S be two irreducible requirement sets such that $RA(G,R) \lesssim_{\text{conf}} RA(G,S)$. Let (x,L) be a requirement. Let $(x,K) \in R$ be a requirement such that $(x,L) \lesssim_{\text{conf}} (x,K)$. Let $(x,J) \in R \cup S$ be a requirement such that $L \subseteq J$.

Then there exists a trace $t \in \Sigma^*$, a requirement (y,L') such that $(x,L) \xrightarrow{t} (y,L')$, and there exists a requirement $(y,O') \in S$ such that $(y,L') \lesssim_{\text{conf}} (y,O')$.

Proof. As x is reachable in G there exists a trace s such that $G \xrightarrow{s} x$. Let s be such a trace.

We will prove the claim by first showing that if $RA(G,R) \lesssim_{\text{conf}} RA(G,S)$ it also holds that $RA(G/s,R) \lesssim_{\text{conf}} RA(G/s,S)$. We will then show that $RA(G/s,R) \lesssim_{\text{conf}}$

$RA(G/s,S)$ can only be true if the requirements (y,L') and (y,O') exist.

From theorem 5.1 if $s \in \mathrm{NCONF}(RA(G,S))$ as $RA(G,R) \lesssim_{\text{conf}} RA(G,S)$ it would also hold that $RA(G,R)/s \lesssim_{\text{conf}} RA(G,S)/s$. We will prove that $s \in \mathrm{NCONF}(RA(G,S))$ by contradiction. Let us assume that $s \in CONF(RA(G,S))$. From lemma 6.7 it holds that $\mathrm{CONF}(RA(G,S)) = \mathrm{CONF}(G)$. As G is deterministic and $G \xrightarrow{s} x$ it must hold that $\mathscr{L}^{\omega}(x) = \emptyset$. Note that $(x,K) \lesssim_{\text{conf}} (x,\emptyset)$. This contradicts our assumption that R is irreducible as from lemma 6.28 it holds that $(x,K) \lesssim_{\text{conf}} (x,\mathscr{L}^{\omega}(x))$ because $(x,K) \in R$ by assumption. Therefore $s \in \mathrm{NCONF}(RA(G,S))$. Furthermore lemma 6.8 shows that $RA(G,R)/s \simeq_{\text{conf}} RA(G/s,R)$ and $RA(G,S)/s \simeq_{\text{conf}} RA(G/s,S)$, thus $RA(G/s,R) \lesssim_{\text{conf}} RA(G/s,S)$

Let T be a deterministic test automaton such that $L(T) = \overline{\mathscr{L}^{\omega}(x)} - L\Sigma_{\omega}^{*}$. We will show that $RA(G,R) \parallel T$ is blocking. As x is the initial state of $RA(G/s,R)$ and $(x,K) \in R$ it holds that $RA(G/s,R) \parallel T \xrightarrow{\varepsilon} ((x,K),Q_T^{\circ})$ where q_T is the initial state of T. Furthermore as $(x,L) \lesssim_{\text{conf}} (x,K)$ it holds that $K \subseteq L\Sigma_{\omega}^{*}$. As $L(T)$ contains no traces in $L\Sigma_{\omega}^{*}$ it follows that $((x,K),Q_T^{\circ})$ is blocking. Therefore $RA(G,R) \parallel T$ is blocking.

As $RA(G/s,R) \lesssim_{\text{conf}} RA(G/s,S)$ it follows that $RA(G,S) \parallel T$ must also be blocking. Let t be a trace and (q,q_T) be a state such that $RA(G/s,S) \parallel T \xrightarrow{t} (q,q_T)$. We will show that (q,q_T) is only blocking if $q = (y,O')$ where $(x,L) \xrightarrow{t} (y,L/t) = (y,L')$ such that $(y,L') \lesssim_{\text{conf}} (y,O')$

We will consider four possible cases of t and q.

(i) $t \notin \overline{L\Sigma_{\omega}^{*}}$ and $q \in Q$

(ii) $t \notin \overline{L\Sigma_{\omega}^{*}}$ and $q \in S$

(iii) $t \in \overline{L\Sigma_{\omega}^{*}}$ and $q \in Q$

(iv) $t \in \overline{L\Sigma_{\omega}^{*}}$ and $q \in S$

We will show that cases $1-3$ are nonblocking, and that for case 4, $q = (y,O')$.

(i) Let $t \notin \overline{L\Sigma_{\omega}^{*}}$ and $q \in Q$. As $t \notin \overline{L\Sigma_{\omega}^{*}}$ it holds that $\mathscr{L}^{\omega}(q_T)$ is equal too $\mathscr{L}^{\omega}(x)/t$ as $L(q_T) = \mathscr{L}^{\omega}(x)/t - L\Sigma_{\omega}^{*} = \mathscr{L}^{\omega}(x)/t - \emptyset$. Furthermore $\mathscr{L}^{\omega}(x)/t \neq \emptyset$ as $t \in \overline{\mathscr{L}^{\omega}(x)/t}$ as $T \overset{t}{\Rightarrow}$, therefore (q,q_T) is nonblocking.

(ii) Let $t \notin \overline{L\Sigma_{\omega}^{*}}$ and $q = (y,O') \in S$. As $t \in \mathscr{L}^{\omega}(x)$ as $T \xrightarrow{t}$ and (y,O') is a requirement set of G it follows that $O' \neq \emptyset$ from definition 6.3, therefore there exists at least one trace $u \in O'$. Furthermore as (y,O') is a requirement of G, it holds that $O' \subseteq \mathscr{L}^{\omega}(y) = \mathscr{L}^{\omega}(x)/t$. Thus $((y,O'),q_T) \xrightarrow{u} (z,z_T)$ for some state pair (z,z_T). As such states have already been proven to be nonblocking $((y,O'),q_T)$ is also nonblocking.

(iii) Let $t \in \overline{L}$ and $q \in Q$. As $L \subseteq J$ and $(x,J) \in R \cup S$ by assumption, it follows that $(x,J) \xrightarrow{t} (q,J/t)$ where $J/t \in R \cup S$ as both R and S are requirement sets. Furthermore as both R and S are irreducible from lemma 6.28 it holds that $(q,J/t) \not\lesssim_{\text{conf}} (q,\mathscr{L}^{\omega}(q))$, therefore there exists a trace $u\omega \in \mathscr{L}^{\omega}(q)$ which is not in $(J/t)\Sigma^*_{\omega}$. As $L/t \subseteq J/t$ it follows that $u\omega$ is not in $(L/t)\Sigma^*_{\omega}$ either. Thus $(q,q_T) \xrightarrow{u\omega}$ as $L(q_T) = (\mathscr{L}^{\omega}(x)/t = \mathscr{L}^{\omega}(q)) - (L/t)\Sigma^*_{\omega}$ and $t \in \mathscr{L}^{\omega}(q)$ but $t \notin (L/t)\Sigma^*_{\omega}$.

(iv) $t \in \overline{L}$ and $q = (y,O') \in S$. $t \in \overline{L} - L$ as T cannot perform any trace in $L\Sigma^*_{\omega}$, therefore there exists a pair (y,L') such that $(x,L) \xrightarrow{t} (y,L')$. Now we will prove if (y,O') is blocking that $(y,L') \lesssim_{\text{conf}} (y,O')$. Let u be a trace in O' such that $u \notin (L/t)\Sigma^*_{\omega}$. As $O' \subseteq \overline{\mathscr{L}^{\omega}(y)}$ it holds that $((y,O'),q_T) \xrightarrow{u} (z,z_T)$ for some pair of states $(z \in Q, z_T \in Q_T)$. As such states have already been proven nonblocking it holds that if O' can be perform any trace $u \notin (L/t)\Sigma^*_{\omega}$ that $((y,O'),q_T)$ is nonblocking. Thus if $((y,O'),q_T)$ is blocking $O' \subseteq (L/t = L')\Sigma^*_{\omega}$, which is equivalent to $(y,L') \lesssim_{\text{conf}} (y,O')$. As this is the only possible type of blocking state and $(x,L) \xrightarrow{t} (y,L')$ the property holds.

Therefore there must exist (y,L') and $(y,O') \in S$ such that $(x,L) \to (y,L')$ and $(y,L') \lesssim_{\text{conf}} (y,O')$. □

Lemma 6.30 Let $G = \langle \Sigma, Q, \to, Q^{\circ} \rangle$ be a deterministic automaton. Let $R \in R^{-}_{G}$ be a requirement set. Let $(x,L),(x,N) \in R$ and (x,M) be three requirement tuples such that $M \subseteq L$ and $(x,M) \lesssim_{\text{conf}} (x,N)$

Then there exists a trace $s \in \overline{L} - L$ such that the following holds.

(i) $M/s \subseteq L/s$

(ii) $(x,L/s) \lesssim_{\text{conf}} (x,L)$

(iii) $(x,M/s) \lesssim_{\text{conf}} (x,L)$

(iv) $(x,N) \xrightarrow{s} (x,L)$

(v) $(x,L/s) \in R$.

Figure 6.13 show a graphical representation of this lemma. We start with the requirements $(x,L),(x,M)$, and (x,N) which are in a triangular relation such that $M \subseteq L$ and (x,N) is more conflicting than both (x,L) and (x,M), we then show that their must exist some trace trace s and requirements $(x,L/s),(x,M/s)$, and $(x,N/s = L)$ such that these requirements share the same triangular relation.

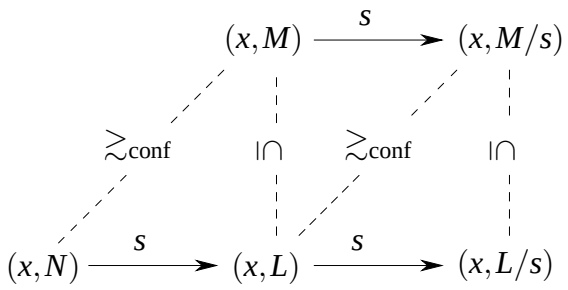


Figure 6.13: A graphical representation of a triangular relation of Nonconflicting Requirements and how it propogates

Proof. First we will prove that $(x,L) \lesssim_{\text{conf}} (x,N)$. Let $t \in N$ be a trace. As $(x,M) \lesssim_{\text{conf}} (x,N)$ it holds that $N \subseteq M\Sigma^*_\omega$, furthermore as $M \subseteq L$ it must be the case that $N \subseteq M\Sigma^*_\omega \subseteq L\Sigma^*_\omega$, therefore $(x,L) \lesssim_{\text{conf}} (x,N)$.

From lemma 6.27, as both $(x,L),(x,N) \in R$ $(x,L) \not<_{\text{conf}} (x,N)$. Furthermore as $(x,L) \lesssim_{\text{conf}} (x,N)$ there must exist a trace $s \in \overline{L} - L$ such that $(x,N) \xrightarrow{s} (x,L)$, otherwise from definition 6.15 it holds that $(x,L) <_{\text{conf}} (x,N)$, this proves requirement iv. As $s \in \overline{L}$, and R is a requirement set $(x,L) \xrightarrow{s} (x,L/s) \in R$ this proves requirement v. As $M \subseteq L$, and $M/s \subseteq L/s$ this proves requirement i.

Next we will prove ii, that $(x,L/s) \lesssim_{\text{conf}} (x,L)$. As $(x,L) \lesssim_{\text{conf}} (x,N)$, $N \subseteq L\Sigma^*_\omega$, therefore as s in both $\overline{L}$ and $\overline{N}$, $N/s \subseteq (L/s)\Sigma^*_\omega$. $L = N/s$ therefore $(x,L/s) \lesssim_{\text{conf}} (x,L)$.

Thirdly we will prove iii, that $(x,M/s) \lesssim_{\text{conf}} (x,L)$. We will first prove that $s \in \overline{M}$. As $s \in \overline{N}$ there must exist some trace $t \in N$ such that $s \sqsubseteq t$, furthermore as $N \subseteq M\Sigma^*_\omega$, there must exists a trace $p \sqsubseteq t$, such that $p \in M$. Furthermore as $M \subseteq L$, $p \in L$. As both $p \sqsubseteq t$ and $s \sqsubseteq t$, it holds that either $p \sqsubseteq s \sqsubseteq t$ or $s \sqsubseteq p \sqsubseteq t$, as $s \in \overline{L} - L$ and L is prefix-free, $s \sqsubseteq p \sqsubseteq t$. Therefore $s \in \overline{M}$, thus $(x,M) \xrightarrow{s} (x,M/s)$ and as First $(x,M) \lesssim_{\text{conf}} (x,N)$ therefore $N \subseteq M\Sigma^*_\omega$, this implies that $N/s \subseteq (M/s)\Sigma^*_\omega$ as both N and M can perform s. Furthermore $N/s = L$ therefore $(x,M/s) \lesssim_{\text{conf}} (x,L)$. □

We can now use lemma 6.30 to show that given three requirement tuples in the triangular relation all three tuples must eventually converge on the same requirement tuple. Figure 6.14 demonstrates the rationale for this lemma, because from lemma 6.30 given three nonconflicting requirements within the triangular relation we can always find another three nonconflicting requirements which are in the same triangular relation which are also less conflicting. As R is a finite set, we must eventually run out of less conflicting requirements, therefore the relation must converge on a particular nonconflicting requirement.

Lemma 6.31 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be a deterministic automaton. Let $R \in R^-_G$ be a requirement set. Let $(x,L),(x,N) \in R$ and (x,M) be three requirement tuples such that

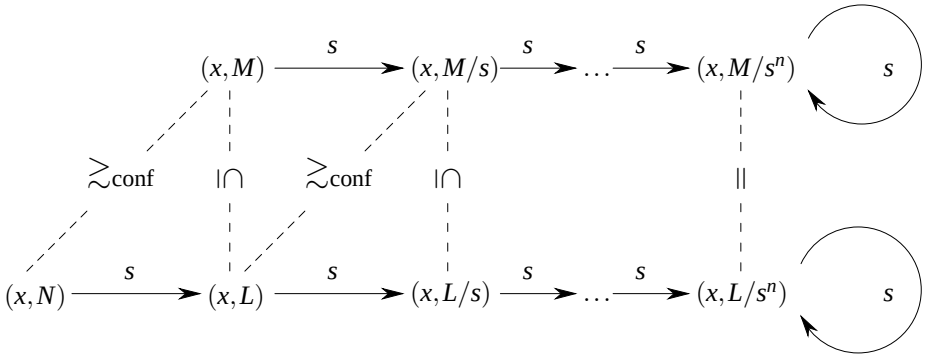


Figure 6.14: A graphical representation of why the triangular relation must eventually converge on a single nonconflicting requirement

$M \subseteq L$ and $(x,M) \lesssim_{\text{conf}} (x,N)$. Let $|\lesssim^*_{\text{conf}} (x,L)|$ be the number of requirements in R which are $\lesssim_{\text{conf}} (x,L)$.

Then there exists a trace $s \in \overline{L} - L$ such that $x = \delta_G(x,s)$ and $L/s = N/s = M/s$

Proof. We will prove the claim via induction on $|\lesssim^*_{\text{conf}} (x,L)|$.

In the base case $|\lesssim^*_{\text{conf}} (x,L)| = 1$. Therefore the only requirement that is less conflicting than (x,L) is (x,L). From lemma 6.30 as G is a deterministic automaton $R \in R^-_G$, $(x,L),(x,N) \in R$ and (x,M) is a requirement tuple, such that $M \subseteq L$. and $(x,M) \lesssim_{\text{conf}} (x,N)$ there exists $s \in \overline{L}$ such that $(x,L/s) \lesssim_{\text{conf}} (x,L)$, $(x,M/s) \lesssim_{\text{conf}} (x,L)$, $M/s \subseteq L/s$, and $(x,N) \xrightarrow{s} (x,L)$. As $(x,N) \xrightarrow{s} (x,L)$ $\delta(x,s) = x$ and $N/s = L$. Furthermore from the base case assumption $(x,L/s) = (x,L) = (x,N/s)$ as there are no other requirements in R which are less conflicting. Now we must prove that $M/s = L/s$. It is already known that $M/s \subseteq L/s$ therefore we must only prove that $L/s \subseteq M/s$. Let t be a trace in L/s. As $(x,M/s) \lesssim_{\text{conf}} (x,L)$, and $L = L/s$, $L/s \subseteq (M/s)\Sigma^*_\omega$. Therefore t must be in $(M/s)\Sigma^*_\omega$. This means that there exists $p \in M/s$ such that $p \sqsubseteq t$. As $M/s \subseteq L/s$ $p \in L/s$ and as L/s is prefix-free $t = p$. As t was chosen arbitrarily $L/s \subseteq M/s$. Thus $L/s = M/s = N/s$.

Now let us consider the inductive case where the property holds for $|\lesssim^*_{\text{conf}} (x,L)| \leq n$, we will prove that the property must hold for $|\lesssim^*_{\text{conf}} (x,L)| = n+1$. From lemma 6.30 there exists $s \in \overline{L}$ such that $(x,L/s) \lesssim_{\text{conf}} (x,L)$, $(x,M/s) \lesssim_{\text{conf}} (x,L)$, $M/s \subseteq L/s$, and $(x,N) \xrightarrow{s} (x,L)$. As $(x,L/s) \lesssim_{\text{conf}} (x,L)$ either $L/s = L$ or $|\lesssim^*_{\text{conf}} (x,L/s)| \leq n$. In the first case prove as for the base case, in the second from the inductive assumption as G is a deterministic automaton, R is irreducible, $(x,L),(x,N) \in R$ and (x,M) is a requirement tuple, such that $M \subseteq L$, and $(x,M) \lesssim_{\text{conf}} (x,N)$ there exists t such that $x = \delta(x,s) = \delta(x,t)$ and $L/st = M/st = N/st$. □

Lemma 6.32 Let $G = \langle \Sigma, Q, \to, Q^\circ \rangle$ be a deterministic automaton. Let R and S be two irreducible requirement sets $RA(G,S) \lesssim_{\text{conf}} RA(G,R)$. Let $(x,L) \in R$. Let $(y,L') \in R$ be

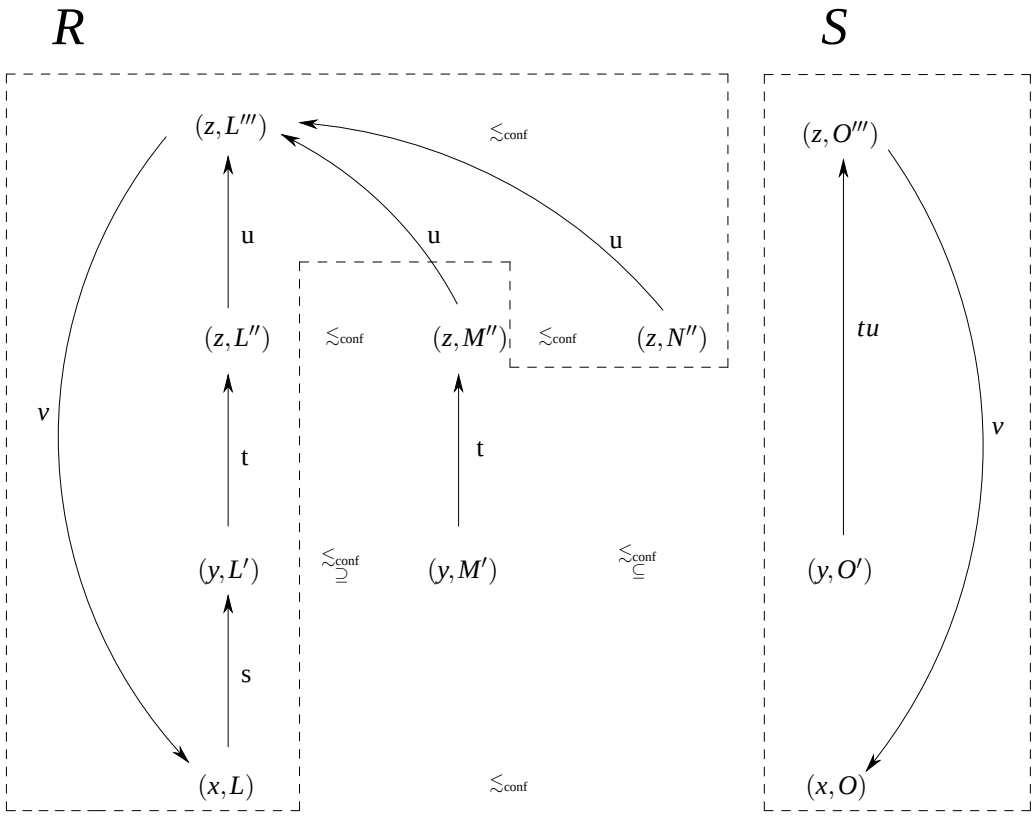


Figure 6.15: A graphical representation of lemma 6.32.

a nonconflicting requirement such that $(x,L) \to (y,L')$. Let $(y,O') \in S$ be a nonconflicting requirement such that $(y,L') \lesssim_{\text{conf}} (y,O')$.

Then there exists $(x,O) \in S$ such that $(x,L) \lesssim_{\text{conf}} (x,O)$.

The proof for lemma 6.32 involves a large number of different requirements. As such figure 6.15 is provided in order to more easily keep track of all the different requirements and how they relate to one another.

Proof. Let $M' = L' \cap \overline{O'}$

We will now prove that $(x,M') \lesssim_{\text{conf}} (x,O')$. Let t be a trace in O'. We will prove that it is also in $M'\Sigma^*_\omega$. Because $(x,L') \lesssim_{\text{conf}} (x,O')$ it holds that $O' \subseteq L'\Sigma^*_\omega$. This implies that their exists $p \in L'$ and $u \in \Sigma^*_\omega$ such that $pu = t$. As $t \in O'$ it holds that $p \in \overline{O'}$ therefore $p \in M'$. Thus $pu = t \in M'\Sigma^*_\omega$. Thus $(x,M') \lesssim_{\text{conf}} (x,O')$.

$(x,M') \lesssim_{\text{conf}} (x,O')$. Therefore from lemma 6.29 as $G = \langle \Sigma, Q, \to, Q^\circ \rangle$ is a deterministic automaton. Both R and S are irreducible. (y,M') is a requirement. $(y,O') \in S$ such that $(y,M') \lesssim_{\text{conf}} (y,O')$. $(y,L') \in S \cup S$ such that $H \subseteq N$. there exists a trace $t \in \Sigma^*$, (z,M''), and $(Z,N'') \in R$ such that $(x,M') \xrightarrow{t} (z,M')$ and $(z,M'') \lesssim_{\text{conf}} (z,N'')$.

Let $L'' = L'/t$. From lemma 6.31 as G is a deterministic automaton, S is irreducible. $(z,L''),(z,N'') \in R$ and (z,M'') are requirement pairs such that $M'' \subseteq L''$

and $(z,M'') \lesssim_{\text{conf}} (z,N'')$. Their exists a trace u such that $z = \delta(z,u)$ and $(z,L''/u) = (z,M''/u) = (z,N'')$.

Furthermore as S is irreducible, from lemma 6.26 there must exist a trace v such that $(z,L'/tu) = (z,L''/u) \xrightarrow{v} (x,L)$ otherwise ψ would have been applied. Let $(x,O) = (x,O'/tuv)$. As $(y,M') \xrightarrow{tu} (z,M''/u) \xrightarrow{v} (x,L)$, $tuv \in \overline{M'}$ furthermore from the construction of $\overline{M'} \subseteq \overline{O'}$ thus $tuv \in \overline{O'}$. Therefore $(y,O') \xrightarrow{tuv} (\delta(y,tuv),O'/tuv) = (x,O)$. Thus as S is a nonconflicting requirement set, (x,O) must be in S. Furthermore from lemma 6.4 as $(y,L') \lesssim_{\text{conf}} (y,O')$, $tuv \in \overline{L'}$, $(x,L) = (\delta_G(y,tuv),L'/tuv) \lesssim_{\text{conf}} (x,O) = (\delta(y,tuv),O'/tuv)$. □

We can now use lemmas 6.29 and 6.32 to prove that for any given trunk automaton G, there is a unique irreducible requirement set for each conflict-equivalence class.

Theorem 6.6 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be a deterministic automaton. Let R and S be two irreducible requirement sets such that $RA(G,R) \simeq_{\text{conf}} RA(G,S)$

Then it holds that $R = S$.

Proof. Let $(x,K) \in R$. As R is finite and $\lesssim_{\text{conf}}$ is antisymmetric there exists a requirement $(x,L) \in R$ such that $(x,K) \lesssim_{\text{conf}} (x,L)$ and for every $(x,M) \in R$ either $(x,L) \not\lesssim_{\text{conf}} (x,M)$ or $(x,L) = (x,M)$.

We will proceed to prove that $(x,L) \in S$.

From lemma 6.29 there exists $(y,M) \in R$ and $(y,N) \in S$ such that $(x,L) \rightarrow (y,M)$ and $(y,M) \lesssim_{\text{conf}} (y,N)$.

From lemma 6.32 there exists $(x,O) \in S$ such that $(x,L) \lesssim_{\text{conf}} (x,O)$.

From lemma 6.29 there exists $(y,M) \in S$ and $(y,N) \in R$ such that $(x,L) \rightarrow (y,M)$ and $(y,M) \lesssim_{\text{conf}} (y,N)$.

From lemma 6.32 there exists $(x,P) \in R$ such that $(x,O) \lesssim_{\text{conf}} (x,P)$.

As $\lesssim_{\text{conf}}$ is transitive $(x,L) \lesssim_{\text{conf}} (x,O) \lesssim_{\text{conf}} (x,P)$. As the only pair in R which is more conflicting than (x,L) is (x,L), $(x,P) = (x,L)$. As $\lesssim_{\text{conf}}$ is antisymmetric $(x,L) = (x,O) = (x,P)$. Thus $(x,L) \in S$. We will now prove that this implies $(x,K) \in S$.

From lemma 6.27 it holds that $(x,K) \not\lesssim_{\text{conf}} (x,L)$ as R is irreducible. Furthermore from definition 6.15 it must be the case that $(x,K) \rightarrow (x,L)$ as $(x,K) \lesssim_{\text{conf}} (x,L)$. From lemma 6.26 it holds that $(x,L) \rightarrow (x,K)$ as $(x,K) \rightarrow (x,L)$. From definition 6.7, $(x,K) \in S$ as S is a requirement set, $(x,L) \in S$ and $(x,L) \rightarrow (x,K)$. - □

We now have all the material we need to define $\hat{R}$, which is the unique irreducible requirement set for any given automaton G.

Definition 6.21 Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be a finite state automaton. Let R be an irreducible requirement set such that $R(G) \succ R$.

$\hat{R}(G) = R$.

We know that $\hat{R}$ is a well-defined function because theorem 6.6 shows that there is a unique irreducible requirement set for each conflict-equivalence class. Futhermore theorem 6.5 shows that every finite nonconflicting requirement set can be refined to an irreducible nonconflicting requirement set using the refinement rules ψ and ϕ.

Theorem 6.7 Let $G = \langle \Sigma, Q_G, \rightarrow_G, Q_G^\circ \rangle$ and $H = \langle \Sigma, Q_H, \rightarrow_H, Q_H^\circ \rangle$ be two automata.
Then $trunk(G) = trunk(H)$ and $\hat{R}(G) = \hat{R}(H)$ if and only if $G \simeq_{\text{conf}} H$.

Proof.

$$
\begin{aligned}
& G \simeq_{\text{conf}} H \\
\iff \quad & G \simeq_{\text{conf}} H \wedge trunk(G) = trunk(H) && \text{from theorem 6.1} \\
\iff \quad & RA(trunk(G), R(G)) \simeq_{\text{conf}} RA(trunk(H), R(H)) \\
& \wedge trunk(G) = trunk(H) && \text{from theorem 6.2} \\
\iff \quad & RA(trunk(G), \hat{R}(G)) \simeq_{\text{conf}} RA(trunk(H), \hat{R}(H)) && \text{from lemma 6.22} \\
& \wedge trunk(G) = trunk(H) && \text{as } R(G) \succ \hat{R}(G) \\
\iff \quad & RA(trunk(G), \hat{R}(G)) \simeq_{\text{conf}} RA(trunk(G), \hat{R}(H)) \\
& \wedge trunk(G) = trunk(H) && \text{as } trunk(G) = trunk(H) \\
\iff \quad & \hat{R}(G) = \hat{R}(H) \wedge trunk(G) = trunk(H) && \text{from theorem 6.6}
\end{aligned}
$$

□

Chapter 7

Conclusion

Verifying whether or not a DES is nonblocking is an NP-Hard problem [14]. However with the use of abstraction it is possible to verify large systems in many cases. In order to verify the nonblocking property, the best equivalence relation to abstract with respect to is conflict equivalence. In chapter 3 we presented annotated automata as a method of abstracting automata with respect to conflict equivalence. Annotated automata take ideas used in failures equivalence, in order to simplify automata with respect to conflict equivalence. The chapter further shows that annotated automata can be used with a compositional nonblocking checker in order to verify whether large discrete event models are nonblocking.

In addition, this thesis set out to gain a greater understanding of conflict equivalence using some of the ideas first developed in chapter 3. In chapter 5 we introduce an algorithm for determining whether one automaton is less conflicting than another. This makes it possible to compare automata with respect to the conflict preorder and also allows us to determine why a given automaton is not less conflicting than another. One of the main uses of the algorithm is to establish a contract for an automaton. If an automaton A is more conflicting than the automaton B the automaton A can be used as a contract for the automaton B with respect to nonblocking. This is because in every situation in which B is blocking, the automaton A is also blocking, therefore if A is nonblocking in that situation we know that B is nonblocking also. The LC pairs algorithm is used for this purpose in [24]. It is also of interest for those who desire to refine automata into a less conflicting automaton which performs the same function. A method of determining whether two automata are conflict equivalent is also neccessary in order to construct the conflict normal form described in chapter 6.

The conflict normal form allows the simplification of any given finite state automaton down to another finite state automaton which uniquely represents its conflict equivalence. This gives us a much greater idea of what makes two automata conflict equiv-

alent by being able to focus uniquely on those elements of the structure of the original automaton which relate to conflicts. As such the conflict normal form is critical in order to understand what could make a particular automaton conflicting with another automaton, as well as for the purpose of studying how different automata are similar and/or different with respect to the situations in which they are conflicting. Furthermore the conflict normal form creates an excellent foundation for understanding how to simplify automata with respect to conflict equivalence.

Possible future work includes implementing the method of constructing the conflict normal form described in chapter 6. Also of interest is the improvement of annotated automata using the knowlegde of conflict equivalence gained in chapters 5 and 6. This is of interest as simplification automata using annotated automata has a worst-case time complexity of $O(n^2)$ as opposed to the worst case exponential time complexity of the conflict normal form.

Another interesting task would be to extend the concept of conflict equivalence. Conflict equivalence makes no assumption about context. In order for two automata to be conflict equivalent to one another they must equivalent when synchronised with any arbitrary test automaton. If we take into account extra information about the context of a model it could be possible to achieve better abstraction. For example a certain event might only ever be blocked by a specific automaton in a model, even though it is synchronised on by many others. Finally it would be interesting to investigate how the conflict normal form could be used in order to construct useful contracts for finite state automata.

Bibliography

[1] Åkesson, K., Fabian, M., Flordal, H., Malik, R.: Supremica—an integrated environment for verification, synthesis and simulation of discrete event systems. In: Proc. 8th Int. Workshop on Discrete Event Systems, WODES '06, pp. 384–385. Ann Arbor, MI, USA (2006)

[2] Brinksma, E., Rensink, A., Vogler, W.: Fair testing. In: I. Lee, S.A. Smolka (eds.) Proc. 6th Int. Conf. Concurrency Theory, CONCUR '95, *LNCS*, vol. 962, pp. 313–327. Springer (1995)

[3] Bryant, R.E.: Graph-based algorithms for Boolean function manipulation. IEEE Trans. Comput. **35**(8), 677–691 (1986)

[4] Brzozowski, J.A.: Derivatives of regular expressions. J. ACM **11**(24), 481–494 (1964). DOI 10.1145/321239.321249

[5] Clarke Jr., E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (1999)

[6] De Nicola, R., Hennessy, M.C.B.: Testing equivalences for processes. Theoretical Comput. Sci. **34**(1–2), 83–133 (1984). DOI 10.1016/0304-3975(84)90113-0

[7] Dijkstra, E.W.: Hierarchical ordering of sequential processes. Acta Inf. **1**(2), 115–138 (1971)

[8] Eloranta, J.: Minimizing the number of transitions with respect to observation equivalence. BIT **31**(4), 397–419 (1991)

[9] Fabian, M., Kumar, R.: Mutually nonblocking supervisory control of discrete event systems. In: Proc. 36th IEEE Conf. Decision and Control, CDC '97, pp. 2970–2975. San Diego, CA, USA (1997)

[10] Fernandez, J.C.: An implementation of an efficient algorithm for bisimulation equivalence. Sci. Comput. Programming **13**, 219–236 (1990)

[11] Flordal, H., Malik, R.: Modular nonblocking verification using conflict equivalence. In: Proc. 8th Int. Workshop on Discrete Event Systems, WODES '06, pp. 100–106. Ann Arbor, MI, USA (2006)

[12] Flordal, H., Malik, R.: Compositional verification in supervisory control. SIAM J. Control and Optimization **48**(3), 1914–1938 (2009). DOI 10.1137/070695526

[13] van Glabbeek, R.J.: The linear time — branching time spectrum I: The semantics of concrete, sequential processes. In: J.A. Bergstra, A. Ponse, S.A. Smolka (eds.) Handbook of Process Algebra, pp. 3–99. Elsevier (2001)

[14] Gohari, P., Wonham, W.M.: On the complexity of supervisory control design in the RW framework. IEEE Trans. Syst., Man, Cybern. (2000)

[15] Hennessy, M.: Algebraic Theory of Processes. MIT Press (1988)

[16] Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall (1985)

[17] Hopcroft, J.E.: An $n \log n$ algorithm for minimizing states in a finite automaton. In: Z. Kohavi, A. Paz (eds.) Theory of Machines and Computations, pp. 189–196. Academic Press, New York, NY, USA (1971)

[18] Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley (2001)

[19] Leduc, R., Malik, R.: A compositional approach for verifying hierarchical interface-based supervisory control. In: Proc. 10th Int. Workshop on Discrete Event Systems, WODES '10, pp. 114–120. Berlin, Germany (2010). DOI 10.3182/20100830-3-DE-4013.00019

[20] Leduc, R.J., Brandin, B.A., Lawford, M., Wonham, W.M.: Hierarchical interface-based supervisory control—part I: Serial case. IEEE Trans. Autom. Control **50**(9), 1322–1335 (2005)

[21] Malik, R.: On the set of certain conflicts of a given language. In: Proc. 7th Int. Workshop on Discrete Event Systems, WODES '04, pp. 277–282. Reims, France (2004)

[22] Malik, R.: The language of certain conflicts of a nondeterministic process. Working Paper 05/2010, Dept. of Computer Science, University of Waikato, Hamilton, New Zealand (2010)

[23] Malik, R., Leduc, R.: Generalised nonblocking. In: Proc. 9th Int. Workshop on Discrete Event Systems, WODES '08, pp. 340–345. Göteborg, Sweden (2008). DOI 10.1109/WODES.2008.4605969

[24] Malik, R., Leduc, R.: Hierarchical interface-based supervisory control using the conflict preorder. In: Proc. 11th Int. Workshop on Discrete Event Systems, WODES '12, pp. 163–168. Guadalajara, Mexico (2012)

[25] Malik, R., Streader, D., Reeves, S.: Conflicts and fair testing. Int. J. Found. Comput. Sci. **17**(4), 797–813 (2006). DOI 10.1142/S012905410600411X

[26] Milner, R.: Communication and concurrency. Series in Computer Science. Prentice-Hall (1989)

[27] Nuutila, E.: Efficient Transitive Closure Compuation in Large Digraphs, *Acta Polytechnica Scandinavica, Mathematics and Computing in Engineering Series*, vol. 74. Finnish Academy of Technology, Helsinki, Finland (1995)

[28] Olderog, E.R., Hoare, C.A.R.: Specification-oriented semantics for communicating processes. Acta Inf. **23**(1), 9–66 (1986)

[29] de Queiroz, M.H., Cury, J.E.R., Wonham, W.M.: Multi-tasking supervisory control of discrete-event systems. In: Proc. 7th Int. Workshop on Discrete Event Systems, WODES '04, pp. 175–180. Reims, France (2004)

[30] Ramadge, P.J.G., Wonham, W.M.: The control of discrete event systems. Proc. IEEE **77**(1), 81–98 (1989)

[31] Rensink, A., Vogler, W.: Fair testing. Information and Computation **205**(2), 125–198 (2007). DOI 10.1016/j.ic.2006.06.002

[32] Roscoe, A.W.: The Theory and Practice of Concurrency. Prentice-Hall (1997)

[33] Su, R., van Schuppen, J.H., Rooda, J.E., Hofkamp, A.T.: Nonconflict check by using sequential automaton abstractions based on weak observation equivalence. Automatica **46**(6), 968–978 (2010). DOI 10.1016/j.automatica.2010.02.025

[34] Ware, S., Malik, R.: The use of language projection for compositional verification of discrete event systems. In: Proc. 9th Int. Workshop on Discrete Event Systems, WODES '08, pp. 322–327. Göteborg, Sweden (2008). DOI 10.1109/WODES.2008.4605966

[35] Ware, S., Malik, R.: Compositional nonblocking verification using annotated automata. In: Proc. 10th Int. Workshop on Discrete Event Systems, WODES '10, pp. 374–379. Berlin, Germany (2010). DOI 10.3182/20100830-3-DE-4013.00060

[36] Ware, S., Malik, R.: A process-algebraic semantics for generalised nonblocking. In: Proc. CATS 2011—Computing: The Australasian Theory Symposium, pp. 75–84. Perth, Australia (2011)

[37] Ware, S., Malik, R.: A state-based characterisation of the conflict preorder. In: Proc. 10th Int. Workshop on the Foundations of Coordination Languages and Software Architectures, FOCLASA 2011, pp. 34–48. Aachen, Germany (2011). DOI 10.4204/EPTCS.58.3

[38] Ware, S., Malik, R.: Conflict-preserving abstraction of discrete event systems using annotated automata. Discrete Event Dyn. Syst. **22**(4), 451–477 (2012). DOI 10.1007/s10626-012-0133-3

[39] Zhang, Z.H., Wonham, W.M.: STCT: An efficient algorithm for supervisory control design. In: B. Caillaud, P. Darondeau, L. Lavagno, X. Xie (eds.) Synthesis and Control of Discrete Event Systems, pp. 77–100. Kluwer (2002)