Local reversibility and the Calculus of Covalent Bonding

Thesis submitted for the degree of
Doctor of Philosophy
at the University of Leicester

by
Stefan Kuhn
Department of Informatics
University of Leicester

2019

Abstract

We introduce a process calculus with a new action prefixing operator that allows to model locally controlled reversibility. Since the observation of covalent bonding in chemical reactions is the starting point of our work we call the process calculus the Calculus of Covalent Bonding (CCB). The calculus is based on CCSK, but adds an operator of the form $(s; b)$, where s is a sequence of actions. Action b can only be executed once all actions in s are done and executing it requires to undo one action in s. By this we achieve control over when reverse actions happen without the need of a global control or a memory. The calculus also allows spontaneous undoing and sequences of forward and reverse actions.
We give a formal definition of the calculus using Structural Operational Semantics rules. We also show properties of the calculus, in particular we show that it can model out-of-causal order reversibility.
In order to demonstrate the use of our calculus we model the hydration of formaldehyde in water into methanediol. We use a system of four molecules, where various paths are possible to produce the final result. These path include a range of chemical situations. Our calculus can model most of them. We also model Base Excision Repair (BER), a bio-chemical process, on a higher level. This show that our calculus is also useful to model more general situations than the chemical reactions it was derived from.

Acknowledgments

I would like to thank my supervisor Dr. Irek Ulidowski for his continuous support throughout this work. My thanks also go to my second supervisor Prof. Reiko Heckel. All PhD students in the Department of Informatics at University of Leicester helped me not only with professional discussions, but also with sustaining a great atmosphere and camaraderie in the PhD offices. I am indebted to Dr. Nils Schlörer and the whole team at the NMR facility of the Department of Chemistry at Universität zu Köln for help with the chemical side of this work. Of course all remaining errors are my own.

Contents

List of Figures **5**

1 Introduction **8**
1.1 Contributions 10
1.2 Thesis outline 12

2 Autoprotolysis of water **14**
2.1 Modelling the autoprotolysis of water 14
2.2 Chemically equivalent processes 19
2.3 Conclusion 24

3 Process calculi and the modelling of biochemical reactions **25**
3.1 Basics of process calculi and rule-based calculi 25
3.2 Related work 28
3.2.1 Early developments in chemistry and computer science 28
3.2.2 Process calculi for chemistry and biology 29
3.2.3 Reversibility in computer science 32
3.2.4 Reversibility and causality 33
3.3 Chemistry in process calculi 37
3.4 Modelling of autoprotolysis of water in process calculi 40
3.4.1 CCSK 40
3.4.2 CCSK with execution control 43
3.4.3 CCS-R 46
3.4.4 κ-calculus 48
3.4.5 P Systems 51
3.5 Conclusion 53

4 A Calculus of Covalent Bonding **55**
4.1 Definition of the calculus 55
4.2 Properties of CCB 70
4.2.1 Concerted actions 79
4.3 CCB without weak actions 81
4.4 Expressiveness 92
4.5 Conclusion 94

5 The hydration of formaldehyde in water **95**
5.1 The most common mechanism of the reaction 96
5.2 Other paths through the reaction 97
5.3 A CCB model of the hydration of formaldehyde in water 100

5.3.1 The main path through the reaction . . . 103
5.3.2 The base-catalysed path . . . 106
5.3.3 The acid-catalysed path . . . 107
5.3.4 Other paths . . . 111
5.4 Conclusion . . . 113

6 Chemical process equivalence 116
6.1 Definition of chemical process equivalence . . . 117
6.2 Properties of the equivalence relation . . . 121
6.3 Chemical process equivalence in the hydration of formaldehyde . . . 123
6.4 Behavioural equivalence . . . 127
6.5 Conclusion . . . 128

7 Base Excision Repair 129
7.1 Description of Base Excision Repair . . . 129
7.2 Modelling BER . . . 130
7.3 Conclusion . . . 136

8 Simulation software 137
8.1 Software architecture . . . 137
8.2 User interfaces . . . 143
8.2.1 Command-line interface . . . 143
8.2.2 Graphical user interface . . . 143
8.3 Process equivalence . . . 150
8.4 Testing . . . 154
8.5 Conclusion . . . 155

9 Conclusion 156
9.1 Summary . . . 156
9.2 Evaluation of CCB . . . 157
9.3 Future work . . . 158
9.3.1 Improve chemical modelling . . . 158
9.3.2 Other future work . . . 160

Appendix Record of the execution of the hydration of formaldehyde in water using CCBsimulation 162

Bibliography 174

List of Figures

2.1 Autoprotolysis of water. 14

2.2 The processes from Figure 2.3. 20

2.3 All paths for the autoprotolysis of water. 21

3.1 SOS rules for CCS. 27

3.2 Inference tree for the transition $(a.\mathbf{0} \mid \overline{a}.\mathbf{0})\backslash\{a\} \xrightarrow{\tau} (\mathbf{0} \mid \mathbf{0})\backslash\{a\}$. 27

3.3 Forward SOS rules for CCSK. 41

3.4 Reverse SOS rules for CCSK. $\rightarrow$ indicates a forward transition, $\rightsquigarrow$ a reverse transition. 42

3.5 SOS rules for multisets. 43

3.6 SOS rules for the control operator $\langle\rangle$. $\rightarrow$ indicates a forward transition, $\rightsquigarrow$ a reverse transition. 44

3.7 SOS rules for CCS-R. 47

3.8 SOS rules for CCS-R with multisets. 47

3.9 A possible modelling of the autoprotolysis of water using the κ calculus. 50

3.10 A possible modelling of the autoprotolysis of water using P Systems. 52

4.1 Predicates std and fsh. 58

4.2 Forward SOS rules for CCB. 58

4.3 Functions k and keys. 59

4.4 Reverse SOS rules for CCB. 60

4.5 SOS rules for concerted actions in CCB. 60

4.6 Structural congruence rule sc. 61

4.7 Reduction rules for CCB. 61

4.8 Inference tree for Example 4.3. 64

4.9 Inference tree for Example 4.4. 65

4.10 Inference tree for Example 4.7. 69

4.11 Algorithm for deciding consistency of processes. 71

4.12 SOS rules for CCB_f. 82

4.13 Rearrangement algorithm. 89

5.1 Hydration of formaldehyde in water into methanediol. 95

5.2 The most common path through the hydration of formaldehyde. . . . 96

5.3 Acid-catalysed hydration of formaldehyde in water. 97

5.4 Base-catalysed hydration of formaldehyde in water. 98

5.5 Other compounds possible in the reaction of formaldehyde with water. 98

5.6 All possible reactions in a system of formaldehyde and three water molecules. 99

5.7 The three main reaction paths in the hydration of formaldehyde. . . . 100

6.1 Function αkeys defining α conversion of key k into key l in CCB and helper function αk, with $k, l \in \mathcal{K}$. 118

6.2 Function αsubscripts defining α conversion of subscript k into subscript l in CCB and helper function αs, αsr, and αsc, with $k, l \in \mathcal{S}$. . 119

6.3 Function s and predicate freshsubscript. 120

6.4 A full representation of the two processes P_1 and P_2. 124

7.1 A *UDG* unit whilst performing a step along a DNA strand. 131

7.2 A three base pair DNA fragment. 133

7.3 The repaired DNA fragment. 135

8.1 The main classes of the software package `CCBsimulation`. 142

8.2 A record of the program `CCBcommandline`. 146

8.3 A screenshot of the program `CCBgui`. 147

8.4 Screenshots of the execution of the base excision repair in CCBsimulation. 148

8.5 Screenshots of the execution of the base excision repair in CCBsimulation. 149

8.6 An algorithm to rewrite a CCB process P_1 into a chemically equivalent processes P_2. 153

8.7 CCBgui demonstrating chemical process equivalence of two processes. 154

9.1 Forward SOS rules of CCB-S. 160

9.2 Reverse SOS rules of CCB-S. 160

9.3 SOS rules for concerted actions of CCB-S. 161

Chapter 1

Introduction

There are many different computation tasks which involve undoing of previously performed steps or actions. Consider a computation where the action a causes the action b, written $a < b$, and where the action c occurs independently of a and b. There are three executions of this computation that preserve *causality*, namely abc, acb and cab. We note that a always comes before b. There are several conceptually different ways of undoing these actions [117]. *Backtracking* is undoing in precisely the reverse order in which they happened. So, undo b undo c undo a is the backtrack of the execution acb. *Reversing* is a more general form of undoing: here actions can be undone in any order provided causality is preserved (meaning that causes cannot be undone before effects). For example, undo c undo b undo a is a reversal of acb for the events a, b and c above.

There are networks of reactions in biochemistry, however, where actions are undone seemingly *out-of-causal order*. The creation and breaking of molecular bonds between the proteins involved in the ERK (extracellular signal-regulated kinases) signalling pathway is a good example of this phenomenon [93]. Let us assume for simplicity that the creation of molecular bonds is represented by actions a, b, c where, as above, $a < b$ and c is independent of a and b. In the ERK pathway, the molecular bonds are broken in the following order: undo a, undo b, undo c, which seems to undo the cause a before the effect b.

In this thesis we introduce a calculus called the Calculus of Covalent Bonding (CCB) to capture these situations. The novel features of our calculus are introduced via an

example of a catalytic reaction. Consider two molecules A and B that are only able to bind if assisted by the catalyst C. We assume:

$$A \stackrel{def}{=} (a;p).A',\ B \stackrel{def}{=} (b,p).B' \text{ and } C \stackrel{def}{=} (a,b).C'$$

Here, (b,p) and (a,b) are sequences of actions, which can be executed in any order. So atom B can do actions b and p, and atom C can do actions a and b in any order. No other operators, e. g. for choice, are allowed inside s. This is very much like in [27, 93]. For atom A, we use a new prefix operator $(s;p).P$ where s is a sequence of actions or executed actions and p is a *weak* action. We call these actions weak, because, as we will see later, they are temporary and facilitate other, more permanent, actions. How actions can work together, or synchronize, is determined by a synchronization function γ. In our case, this is the function $\gamma(a,a) = c$, $\gamma(b,b) = d$ and $\gamma(p,p) = q$. This function produces new actions c, d and q. The molecules A, B and C can bond by performing synchronously the matching actions according to γ. In the $(s;p)$ operator, which is the new feature here, p can happen only if all actions in s have already taken place. The simple operators like (b,p) can be considered as the new operator with the weak action p can be left out resulting in the reduced prefix $(s).P$ (as in B and C above). If a weak action p exists, it brings reversibility into our model. Once p takes place, one of the executed actions in s must be undone immediately: this is our new mechanism for triggering reverse computation. We shall model these two almost simultaneous events as a transition of *concerted actions*.[1] This is a simple but realistic representation of the mechanism of covalent bonding, the most common type of chemical bonds between atoms, hence our calculus is called a *Calculus of Covalent Bonding*.

Returning to our example, we represent the system of molecules A, B and C as

$$((a;p).A' \mid (b,p).B' \mid (a,b).C') \setminus \{a,b,p\}$$

where ' | ' is the parallel composition and '\' the restriction as in CCS [78]. We note that A and B cannot interact initially since $\gamma(a,b)$ is not defined. But they can both interact with C, which is represented by the following two transitions:

$$(a;p).A' \mid (b,p).B' \mid (a,b).C' \xrightarrow{c[1]} (a[\mathbf{1}];p).A' \mid (b,p).B' \mid (a[\mathbf{1}],b).C' \xrightarrow{d[2]}$$
$$(a[1];p).A' \mid (b[\mathbf{2}],p).B' \mid (a[1],b[\mathbf{2}]).C'$$

[1] In chemistry a concerted reaction is defined as a reaction where all bonds are made or broken in a single step without a reaction intermediate, also not an unstable intermediate (entry "concerted reaction" of [85]).

where 1 and 2 are *communication keys* [94, 90] indicating which pairs of actions created bonds. New communication keys are highlighted in **bold**. The bold markup is not part of the syntax, it is only there to help the reader. Molecules A and B can now do p synchronously, producing the action q. This causes immediately the breaking of the bond c, which means undoing actions a in A and C, leaving A and B bonded. We model such pairs of events by pairs of concerted actions:

$$\begin{aligned}&(a[1];p).A' \mid (b[2],p).B' \mid (a[1],b[2]).C'\\ &\xrightarrow{\{q[3],\underline{c}[1]\}} (\boldsymbol{a};p[\mathbf{3}]).A' \mid (b[2],p[\mathbf{3}]).B' \mid (\boldsymbol{a},b[2]).C'\end{aligned}$$

The action $\boldsymbol{a}$ is bold since it has lost its communication key 1. The bond 3 on a weak action p is unstable and thus gets *promoted* to a stable stronger bond on a and p (of B), which is represented by the following rewrite:

$$(a;p[3]).A' \mid (b[2],p[3]).B' \mid (a,b[2]).C' \Rightarrow (a[\mathbf{3}];p).A' \mid (b[2],p[3]).B' \mid (a,b[2]).C'$$

Finally, the catalyst dissolves the bond with B:

$$(a[\mathbf{3}];p).A' \mid (b[2],p[3]).B' \mid (a,b[2]).C' \xrightarrow{\underline{d}[2]} (a[3];p).A' \mid (\boldsymbol{b},p[3]).B' \mid (a,\boldsymbol{b}).C'$$

We note that A and B are now bonded although the synchronisation function did not allow it to happen initially. The main consequence of this is that the bond between $a[3]$ and $p[3]$ is *irreversible*, namely it cannot be undone. Looking at the pattern of doing and undoing of bonds we obtain the following sequence:

$$c[1]d[2]q[3]\underline{c}[1]\underline{d}[2]$$

Since creation of bonds c and d causes the bond q, and since c and d are undone whilst q is not, we have here an example of out-of-causal order computation.

1.1 Contributions

This thesis gives a formal definition of a novel calculus. This is done using *Structural Operational Semantics* (SOS for short) style [95]. This includes novel SOS rules for concerted actions and three rewrite rules that prescribe when bonds on weak actions can be promoted to strong action bonds. The calculus models doing and undoing of communications and does not require global control. By using the new prefixing operator we can define within one process if a certain forward transition triggers the

undoing of a previously done action. Since this information is contained within the definition of a process, the mechanism for undoing is purely local.

We show that out-of-causal order computation can be modelled in this calculus. Hence, in general, the *causal consistency* property [26] does not hold. There are reachable states that can only be arrived at by a mixture of forward and reverse steps.

We compare the new calculus to existing calculi. We argue that causal consistency holds in a restricted version of our calculus, where no weak actions are contained. This sub-calculus with the reduced prefixing operator $(s).P$ satisfies causal consistency. Therefore the full calculus is in effect a conservative extension of a causally consistent reversible process calculus. We show that CCB is a well behaved calculus by proving a number of useful properties. These include a forward diamond property and a reverse diamond property for simple transitions. We also show that we can reach states by doing forward and reverse transitions which cannot be reached by computing forwards alone.

We can model actual chemical reactions as well as higher-level biological processes in CCB. We model a basic reaction, the autoprotolysis of water, containing two molecules, with one possible forward and reverse transition. A more complicated example is the hydration of formaldehyde in water, where a variety of transitions, both forward and reverse, and different paths between states, are possible. We model base excision repair (BER) as an example of a higher-level biological process. When doing so we are able to represent different forms of reversibility, including out-of-causal order reversibility, and computation can proceed in any direction without external control. We present the possible (chemically valid) sequences which result from our model and show that they cover most of the actually possible reactions. We also explain that a large majority of reactions that our model produces are chemically valid (see Section 5.4 for details).

We define an equivalence, which we call *chemical process equivalence*. This is based on identifying actions and processes by subscripts. Swaps of such subscripts and keys allow us to transform equivalent processes into each other. We show that this is a useful equivalence in our calculus, since it allows us to model chemical reaction networks realistically (see Section 6.4 for details).

We also present a software tool, which allows for the simulation of CCB processes. It offers a command line interface and a graphical user interface, where a process can be entered, checked for correctness, and possible transitions can be listed. The user can then execute these and follow the evolution of the initial process. If a CCB

process is used to model a chemical system this is effectively a reaction simulation system. Chemical process equivalence has been implemented in this software by using a graph isomorphism as an intermediate step in establishing equivalence or non-equivalence of two processes. This allows fast computation of such equivalences.

1.2 Thesis outline

In Chapter 2 we introduce CCB informally using a simple example, the autoprotolysis of water.

Chapter 3 explains the background of the work. This includes an introduction to process calculi, gives an overview of modelling efforts in biology and how process calculi are used for this. We demonstrate how reversibility is handled in various existing calculi. Finally we explain how our new calculus relates to other calculi and biological and chemical modelling previously presented in the literature.

In Chapter 4 we give a formal definition of CCB. It is presented using Structural Operational Semantics style semantics. We then discuss various properties that hold in the calculus. We also define a simplified sub-calculus of CCB without weak actions. A further sub-calculus of this is CCB without weak actions and forward-only transitions.

We demonstrate the usefulness of our calculus in Chapter 5 by modelling the hydration of formaldehyde in water. Formaldehyde and two water molecules are modelled as appropriate compositions of carbon, oxygen and hydrogen atoms. Then, the molecules are composed into a process, and the computation of the process is represented by sequences of pairs of concerted actions.

In Chapter 6 we introduce an extension of our calculus with identifiable processes and actions. We show that this can lead to syntactically different processes which are actually equivalent. We give a definition of this equivalence, which we call chemical equivalence.

Another example, the base excision repair (BER) mechanism for DNA repair, is given in Chapter 7. This is a higher level example from biology where our calculus is not used to represent chemical processes directly, but is modelling aggregated reactions. The calculus can deal with the situations on both levels.

A simulation software for CCB is presented in Chapter 8. We describe the architecture of the software as well as the user interface. We show the usage of the software for simulating the examples in Chapters 5 and 7.

Finally in Chapter 9 we summarise the methods and achievements of this thesis. We also describe limitations and areas for further work.

Some results presented in this thesis were published in [59, 60, 61, 58]. Specifically, the example from Chapter 1 was used in [59, 60, 61] and the example from Chapter 2 was used in [59]. CCB was presented in [60, 61] similar to Chapter 4. A preliminary version of Chapter 7 was printed in [58].

Chapter 2

Autoprotolysis of water

In this chapter we introduce our calculus informally, concentrating on the new general prefixing operator $(s; b).P$ which produces pairs of *concerted* actions, while presenting an intuitive model of the autoprotolysis of water. We also use this to introduce some chemical concepts we will use later. A more detailed justification why this modelling is appropriate and how it relates to chemical knowledge and the existing research will be given in Chapter 3.

2.1 Modelling the autoprotolysis of water

We consider a reaction that transfers a hydrogen atom between two water molecules. Since the reaction takes place in water it is also known as *autoprotolysis of water*. The reaction is shown in Figure 2.1. Here, O indicates an oxygen atom and H a hydrogen atom. The lines indicate bonds. Charges on atoms are shown by $\oplus$ and $\ominus$. The meaning of the curved arrows and the dots will be explained in the next paragraph. The reaction is reversible and it takes place at a relatively low rate, making pure water slightly conductive.

In order to model this reaction we need to understand what it is that makes it happen. The main factor is that the oxygen atom in the water molecule is *nucleophilic*, meaning it has the tendency to bond to another atomic nucleus, which would serve

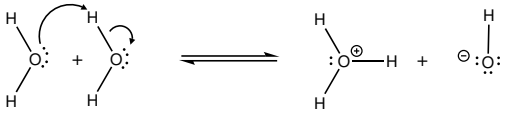

Figure 2.1: Autoprotolysis of water.

as an *electrophile*. This is because oxygen has a high electro-negativity, therefore it attracts electrons and has an abundance of electrons around it. The electrons around the atomic nucleus are arranged on electron shells, where only those in the outer shell participate in bonding. Oxygen has four electrons in its outer shell which are not involved in the initial bonding with hydrogen atoms. These electrons form two *lone pair* of two electrons each, which can form new bonds (lone pairs are shown in Figure 2.1 by pairs of dots). All this makes oxygen nucleophilic: it tends to connect to other atomic nuclei by forming bonds from its lone pairs. Since oxygen attracts electrons, the hydrogen atoms in water have a positive partial charge and the oxygen atom a negative partial charge. The reaction starts with the oxygen atom (in one water molecule) being attracted by a hydrogen atom in another water molecule due to their opposite charges (this attraction is called a *hydrogen bond*). Due to the nucleophilicity of the oxygen atom, a bond can form to the hydrogen atom. This bond is formed out of the electrons of one of the lone pairs of the oxygen atom. The curved arrows in Figure 2.1 indicate the movements of the electrons. Since a hydrogen atom cannot have more than one bond the creation of a new bond is compensated by breaking the existing hydrogen-oxygen bond (again indicated by a curved arrow). When this happens, both electrons, which form the hydrogen-oxygen bond, are left with the oxygen atom. Since a hydrogen atom consists of one electron and one proton, it is only the proton that is transferred, so the process can be called a proton transfer as well as a hydrogen transfer. The forming of the new bond and the breaking of the old bond are *concerted*, namely they happen together without a stable intermediate configuration. As a result we have reached the state where one oxygen atom has three bonds to hydrogen atoms and is positively charged, represented on the right side of Figure 2.1. This molecule is called hydronium and can be written as H_3O^+. The other oxygen atom bonds to only one hydrogen atom and is negatively charged, having an electron in surplus. This molecule is a hydroxide and can be written as OH^-.

We notice the reaction is reversible: the oxygen atom, which has lost a hydrogen atom, can pull back one of the hydrogen atoms from the other molecule, the H_3O^+ molecule. This is the case since the negatively charged oxygen atom is a strong nucleophile and the hydrogen atoms in the H_3O^+ molecule are all positively charged. Therefore, any of the hydrogen atoms can be removed, making both oxygen atoms formally uncharged, and restoring the two water molecules. In Figure 2.1 the curved arrows are given for the reaction going from left to right. Since the reaction is reversible (indicated by the double arrow) there are correspondent electron movements when going from right to left. These are not given in line with usual conventions, but can be inferred.

We shall model the hydrogen and oxygen atoms as processes H and O as follows, where h, o are actions representing the bonding capabilities of the atoms and n, p representing negative and positive charges, respectively. H' and O' are process constants.

$$\begin{array}{lcl} H & \stackrel{def}{=} & (h;p).H' \\ O & \stackrel{def}{=} & (o,o,n).O' \end{array}$$

We use a general prefixing construct $(s;b).P$ where s is a sequence of actions or executed actions, and b is a *weak* action. The prefixing operator must not used (as in the definition of O). In such a case at most one of the actions in the sequence can be a weak action. Informally, actions in s can take place in any order and b can happen if all actions in s have already taken place. Once b takes place, it must be accompanied by undoing immediately one of the actions in s. The weak action in the sequence, as we shall see later, is important for rewriting operations.

We use a synchronisation function γ which tells us which actions can combine to produce bonds between atoms. We define γ in the style of ACP ([36], [3]), where actions ho, np, nh, no represent the created bonds and the communication function is a partial function $\gamma : \mathcal{A} \times \mathcal{A} \to \mathcal{A}$. :

$$\begin{array}{lcllcl} \gamma(h,o) & = & ho & \gamma(n,p) & = & np \\ \gamma(n,h) & = & nh & & & \end{array}$$

Each water molecule is a structure consisting of two hydrogen atoms and one oxygen atom which are bonded appropriately. We shall use subscripts to name the individual copies of atoms and actions; for example H_1 is a specific copy of hydrogen atom defined by $(h_1;p).H_1'$, similarly for O_1 defined as $(o_1,o_2,n).O_1'$. The atoms are composed with the parallel composition operator "|" using the communication keys (which are natural numbers) to combine actions into bonds. So a water molecule is modelled by the following process, where the key 1 shows that h_1 of H_1 has bonded with o_1 of O_1 (correspondingly for key 2). The restriction $\backslash\{h_1, h_2, o_1, o_2\}$ ensures that these actions cannot happen on their own, but only together with their partners, forming a bond.

$$((h_1[1];p).H_1' \mid (h_2[2];p).H_2' \mid (o_1[1],o_2[2],n).O_1') \setminus \{h_1, h_2, o_1, o_2\}$$

The system of two water molecules in Figure 2.1 is represented by the parallel composition of two water processes, where the restriction $\backslash\{n, p\}$ represses actions

n, p from taking place separately by forcing them to combine into bonds (according to γ).

$$(((h_1[1];p).H_1' \mid (h_2[2];p).H_2' \mid (o_1[1], o_2[2], n).O_1') \setminus \{h_1, h_2, o_1, o_2\} \mid$$
$$((h_3[3];p).H_3' \mid (h_4[4];p).H_4' \mid (o_3[3], o_4[4], n).O_2') \setminus \{h_3, h_4, o_3, o_4\}) \setminus \{n, p\}$$

Following a general principle in process calculi in the style of CCB we can move the restritions to the outside. The rule used can be written as $(P \mid Q) \setminus L = P \setminus L \mid Q$ if the actions of L are not used in Q. Applying this gives us a water molecule modelled as follows:

$$((h_1[1];p).H_1' \mid (h_2[2];p).H_2' \mid (o_1[1], o_2[2], n).O_1') \mid (h_3[3];p).H_3' \mid$$
$$(h_4[4];p).H_4' \mid (o_3[3], o_4[4], n).O_2')) \setminus \{h_1, h_2, o_1, o_2\} \setminus \{h_3, h_4, o_3, o_4\} \setminus \{n, p\}$$

Note the $\underline{h_i}$, $\underline{o_j}$, and $\underline{n}$ are not restricted: this allows us to break bonds via concerted actions involving these actions. We will see an example of this in the next step.

Notice that we leave out the restrictions to improve readability in the following description.

Now actions n in O_1 and p in H_3 can combine (we use the new key 5), representing a transfer of a proton from one atom of oxygen (O_2 in our model) to another one (O_1 in our model). Since a hydrogen atom consists of a proton and an electron, and the electron stays in such a transfer, it can either be called a proton transfer or the transfer of a (positively charged) hydrogen atoms. We show the transfer from O_2 to O_1, where $\rightarrow$ is a transition relation denoting a reaction taking place, here a creation of a bond (changes highlighted in **bold**):

$$(h_1[1];p).H_1' \mid (h_2[2];p).H_2' \mid (o_1[1], o_2[2], n).O_1') \mid$$
$$(h_3[3];p).H_3' \mid (h_4[4];p).H_4' \mid (o_3[3], o_4[4], n).O_2')$$
$$\rightarrow$$
$$(h_1[1];p).H_1' \mid (h_2[2];p).H_2' \mid (o_1[1], o_2[2], n\mathbf{[5]}).O_1' \mid (h_3[3];p\mathbf{[5]}).H_3'$$
$$\mid (h_4[4];p).H_4' \mid (o_3[3], o_4[4], n).O_2'$$

The creation of the bond with key 5 forces us to break the bond with key 3 (between h_3 and o_3) due to the property of the operator $(s; b).P$ discussed earlier:

$$
\begin{aligned}
&(h_1[1]; p).H_1' \mid (h_2[2]; p).H_2' \mid (o_1[1], o_2[2], n).O_1') \mid \\
&(h_3[3]; p).H_3' \mid (h_4[4]; p).H_4' \mid (o_3[3], o_4[4], n).O_2') \\
&\rightarrow \\
&(h_1[1]; p).H_1' \mid (h_2[2]; p).H_2' \mid (o_1[1], o_2[2], n[5]).O_1' \mid (\boldsymbol{h_3}; p[5]).H_3' \\
&\mid (h_4[4]; p).H_4' \mid (\boldsymbol{o_3}, o_4[4], n).O_2'
\end{aligned}
$$

These two reactions happen almost simultaneously so we represent them as a pair of *concerted actions*. In line with the convention in process calculi, where the action performed is written above the transition arrow, we write a pair of reactions above the transition arrow, to express the concerted actions. We enclose them in {} to indicate concerted actions.

$$
\begin{aligned}
&(h_1[1]; p).H_1' \mid (h_2[2]; p).H_2' \mid (o_1[1], o_2[2], n).O_1' \mid (h_3[3]; p).H_3' \\
&\mid (h_4[4]; p).H_4' \mid (o_3[3], o_4[4], n).O_2') \\
&\xrightarrow{\{np[5], \underline{h_3 o_3}[3]\}} \\
&((h_1[1]; p).H_1' \mid (h_2[2]; p).H_2' \mid (o_1[1], o_2[2], n\boldsymbol{[5]}).O_1' \mid (\boldsymbol{h_3}; p\boldsymbol{[5]}).H_3' \\
&\mid (h_4[4]; p).H_4' \mid (\boldsymbol{o_3}, o_4[4], n).O_2'
\end{aligned}
$$

We have now arrived at the state on the right hand side in Figure 2.1. There are weak bonds between n and p (denoted by key 5) and *strong* bonds between h_i and o_j for all appropriate i, j. Since H_3 is weakly bonded to O_1 and its strong capability h_3 has become available, the bond 5 gets promoted to the stronger bond, releasing the capability p of H_3. We represent this change as a rewrite denoted by $\Rightarrow$ (which is not as a transition) and we obtain the following process:

$$
\begin{aligned}
&((h_1[1]; p).H_1' \mid (h_2[2]; p).H_2' \mid (o_1[1], o_2[2], n\boldsymbol{[5]}).O_1' \mid (\boldsymbol{h_3}; p\boldsymbol{[5]}).H_3' \\
&\mid (h_4[4]; p).H_4' \mid (\boldsymbol{o_3}, o_4[4], n).O_2' \\
&\Rightarrow ((h_1[1]; p).H_1' \mid (h_2[2]; p).H_2' \mid (o_1[1], o_2[2], n[5]).O_1') \mid (h_3\boldsymbol{[5]}; \boldsymbol{p}).H_3') \\
&\mid (h_4[4]; p).H_4' \mid (o_3, o_4[4], n).O_2'
\end{aligned}
$$

Oxygen O_1 is still blocked, which represents it being fully bonded (and positively charged). Oxygen O_2 has a free n capability and can remove any of the hydrogen atoms from O_1. As a result the process can reverse to its original state.

We show this by again transferring H_3. We then execute promotion again:

$$
\begin{aligned}
&(((h_1[1];p).H_1' \mid (h_2[2];p).H_2' \mid (o_1[1],o_2[2],n[5]).O_1') \mid (h_3[5];p).H_3') \\
&\mid (h_4[4];p).H_4' \mid (o_3,o_4[4],n).O_2' \\
&\xrightarrow{\{np[3],\underline{nh_3}[5]\}} \\
&(((h_1[1];p).H_1' \mid (h_2[2];p).H_2' \mid (o_1[1],o_2[2],\boldsymbol{n}).O_1') \mid (\boldsymbol{h_3};\boldsymbol{p}[\mathbf{3}]).H_3') \\
&\mid (h_4[4];p).H_4' \mid (o_3,o_4[4],\boldsymbol{n}[\mathbf{3}]).O_2' \\
&\Rightarrow \\
&(((h_1[1];p).H_1' \mid (h_2[2];p).H_2' \mid (o_1[1],o_2[2],\boldsymbol{n}).O_1') \mid (\boldsymbol{h_3}[\mathbf{3}];\boldsymbol{p}).H_3') \\
&\mid (h_4[4];p).H_4' \mid (\boldsymbol{o_3}[\mathbf{3}],o_4[4],\boldsymbol{n}).O_2'
\end{aligned}
$$

This corresponds to the original process, which we started with, after moving the restrictions to the outside. Remember that we left out the restrictions only to improve readability. If we re-add them we get:

$$
\begin{aligned}
&((h_1[1];p).H_1' \mid (h_2[2];p).H_2' \mid (o_1[1],o_2[2],n).O_1' \mid (h_3[3];p).H_3' \\
&\mid (h_4[4];p).H_4' \mid (o_3[3],o_4[4],n).O_2') \setminus \{h_1,h_2,o_1,o_2\} \setminus \{h_3,h_4,o_3,o_4\} \setminus \{n,p\}
\end{aligned}
$$

We can therefore apply the movement of restrictions the reverse way we applied it originally and get:

$$
\begin{aligned}
&(((h_1[1];p).H_1' \mid (h_2[2];p).H_2' \mid (o_1[1],o_2[2],n).O_1') \setminus \{h_1,h_2,o_1,o_2\} \mid \\
&((h_3[3];p).H_3' \mid (h_4[4];p).H_4' \mid (o_3[3],o_4[4],n).O_2') \setminus \{h_3,h_4,o_3,o_4\}) \setminus \{n,p\}
\end{aligned}
$$

2.2 Chemically equivalent processes

So far we have chosen a particular hydrogen atom to transfer. It would be equally possible to use any other hydrogen atom. Since we have marked our actions and processes by subscript numbers, we are able to distinguish the various situations here. Furthermore, the use of keys is arbitrary and many more syntactically different processes can be generated by using different keys. If we disregard different keys linking the same actions, but annotate actions and processes by subscripts, we can distinguish the processes shown in Figure 2.3. In Figure 2.2 the processes are shown in a graph with the possible transitions. The numbers in Figure 2.2 correspond to those in Figure 2.3.

1 $(h_1[1];p).H'_1 \mid (h_2[2];p).H'_2 \mid (o_1[1], o_2[2], n).O'_1 \mid$
$(h_3[3];p).H'_3 \mid (h_4[4];p).H'_4 \mid (o_3[3], o_4[4], n).O'_2)$

2 $(h_1[1];p).H'_1 \mid (h_2[2];p).H'_2 \mid (o_1[1], o_2[2], n[5]).O'_1 \mid$
$(h_3[5];p).H'_3 \mid (h_4[4];p).H'_4 \mid (o_3, o_4[4], n).O'_2)$

3 $(h_1[1];p).H'_1 \mid (h_2[2];p).H'_2 \mid (o_1[1], o_2[2], n[5]).O'_1 \mid$
$(h_3[3];p).H'_3 \mid (h_4[5];p).H'_4 \mid (o_3[3], o_4, n).O'_2)$

4 $(h_1[1];p).H'_1 \mid (h_2[5];p).H'_2 \mid (o_1[1], o_2, n).O'_1 \mid$
$(h_3[3];p).H'_3 \mid (h_4[4];p).H'_4 \mid (o_3[3], o_4[4], n[5]).O'_2)$

5 $(h_1[5];p).H'_1 \mid (h_2[2];p).H'_2 \mid (o_1, o_2[2], n).O'_1 \mid$
$(h_3[3];p).H'_3 \mid (h_4[4];p).H'_4 \mid (o_3[3], o_4[4], n[5]).O'_2)$

6 $(h_1[6];p).H'_1 \mid (h_2[2];p).H'_2 \mid (o_1[5], o_2[2], n).O'_1 \mid$
$(h_3[3];p).H'_3 \mid (h_4[5];p).H'_4 \mid (o_3[3], o_4[6], n).O'_2)$

7 $(h_1[1];p).H'_1 \mid (h_2[6];p).H'_2 \mid (o_1[1], o_2[5], n).O'_1 \mid$
$(h_3[3];p).H'_3 \mid (h_4[5];p).H'_4 \mid (o_3[3], o_4[6], n).O'_2)$

8 $(h_1[6];p).H'_1 \mid (h_2[2];p).H'_2 \mid (o_1[5], o_2[2], n).O'_1 \mid$
$(h_3[5];p).H'_3 \mid (h_4[4];p).H'_4 \mid (o_3[6], o_4[4], n).O'_2)$

9 $(h_1[1];p).H'_1 \mid (h_2[6];p).H'_2 \mid (o_1[1], o_2[5], n).O'_1 \mid$
$(h_3[5];p).H'_3 \mid (h_4[4];p).H'_4 \mid (o_3[6], o_4[4], n).O'_2)$

10 $(h_1[6];p).H'_1 \mid (h_2[2];p).H'_2 \mid (o_1[5], o_2[2], n[7]).O'_1 \mid$
$(h_3[7];p).H'_3 \mid (h_4[5];p).H'_4 \mid (o_3, o_4[6], n).O'_2)$

11 $(h_1[6];p).H'_1 \mid (h_2[7];p).H'_2 \mid (o_1[5], o_2, n).O'_1 \mid$
$(h_3[3];p).H'_3 \mid (h_4[5];p).H'_4 \mid (o_3[3], o_4[6], n[7]).O'_2)$

12 $(h_1[6];p).H'_1 \mid (h_2[7];p).H'_2 \mid (o_1[5], o_2, n).O'_1 \mid$
$(h_3[5];p).H'_3 \mid (h_4[4];p).H'_4 \mid (o_3[6], o_4[4], n[7]).O'_2)$

13 $(h_1[6];p).H'_1 \mid (h_2[2];p).H'_2 \mid (o_1[5], o_2[2], n[7]).O'_1 \mid$
$(h_3[5];p).H'_3 \mid (h_4[7];p).H'_4 \mid (o_3[6], o_4, n).O'_2)$

14 $(h_1[6];p).H'_1 \mid (h_2[8];p).H'_2 \mid (o_1[5], o_2[7], n).O'_1 \mid$
$(h_3[5];p).H'_3 \mid (h_4[7];p).H'_4 \mid (o_3[6], o_4[8], n).O'_2)$

Figure 2.2: The processes from Figure 2.3. Restrictions are left out for clarity and possible differences due to keys used are ignored.

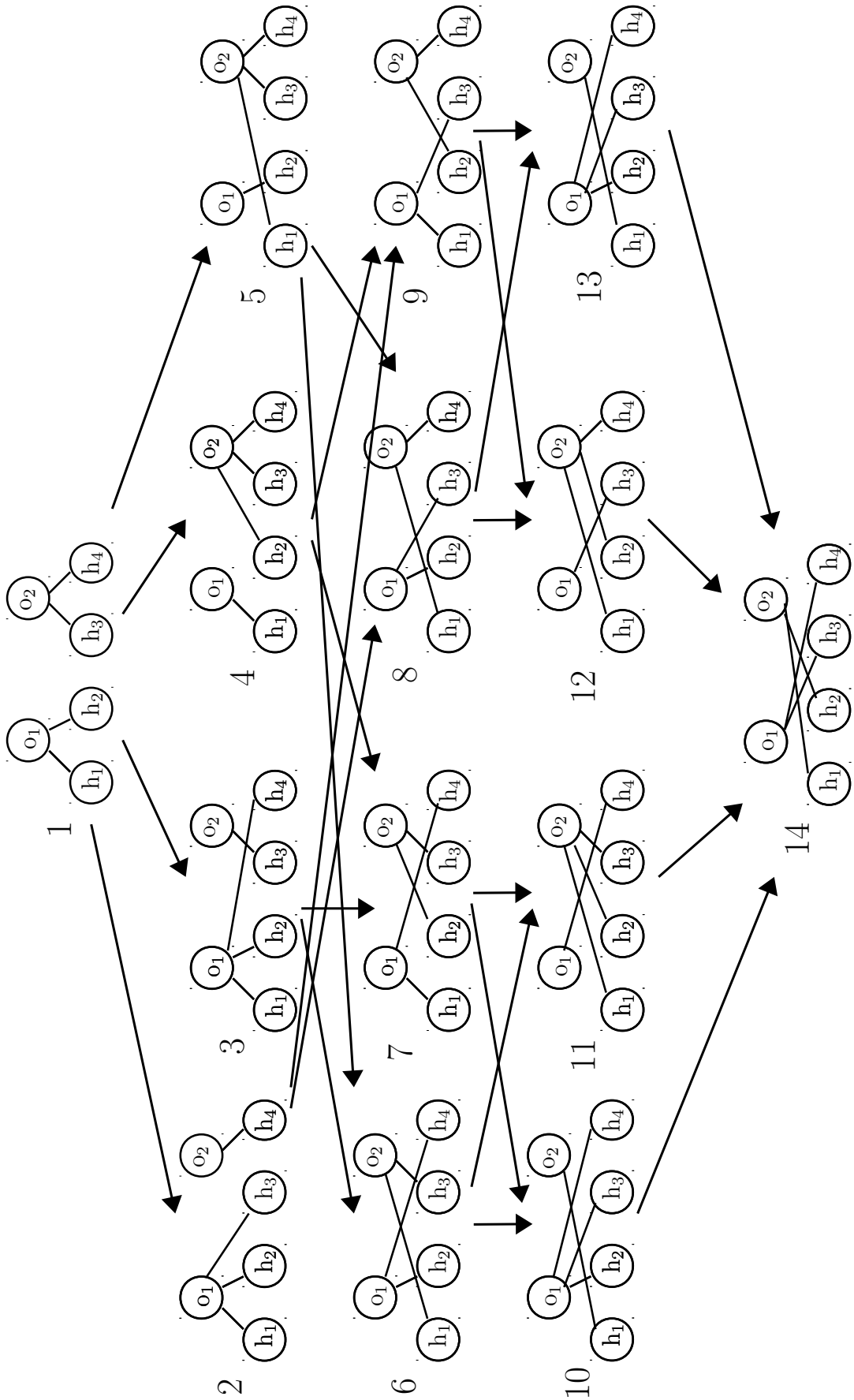

Figure 2.3: All paths for the autoprotolysis of water. The numbers correspond to the processes shown in Figure 2.2. Some transitions require changes of keys.

From a chemical point of view there are only two possible configurations, namely a system of two water molecules (processes 1, 6, 7, 8, 9, and 14 in Figure 2.3) and a system of one H_3O^+ and one HO^- molecule (processes 2, 3, 4, 5, 10, 11, 12, and 13 in Figure 2.3). Processes 6, 7, 8, and 9 have two hydrogen atoms swapped between the oxygen atoms, and process 14 has all hydrogen atoms swapped between the oxygen atoms.

In order to model this equality in our calculus, we need two operations, which we discuss informally here. Firstly, we must be able to change a key in the process to some other fresh key. The new key must not be used in the process so far. For example, in Figure 2.2 process 5 is:

$$(h_1[5];p).H_1' \mid (h_2[2];p).H_2' \mid (o_1, o_2[2], n).O_1' \mid$$
$$(h_3[3];p).H_3' \mid (h_4[4];p).H_4' \mid (o_3[3], o_4[4], n[5]).O_2')$$

From this process we can transition to process 6 in Figure 2.2. This would happen by the following transition and promotion:

$$(h_1[5];p).H_1' \mid (h_2[2];p).H_2' \mid (o_1, o_2[2], n).O_1' \mid$$
$$(h_3[3];p).H_3' \mid (h_4[4];p).H_4' \mid (o_3[3], o_4[4], n[5]).O_2')$$
$$\xrightarrow{\{np[6], \underline{h_4}n[4]\}}\Rightarrow$$
$$(h_1[5];p).H_1' \mid (h_2[2];p).H_2' \mid (o_1[6], o_2[2], n).O_1' \mid$$
$$(h_3[3];p).H_3' \mid (h_4[6];p).H_4' \mid (o_3[3], o_4[5], n).O_2')$$

On the other hand, Figure 2.2 tells us that process 6 is actually as follows (this was derived by a transition from process 3, which leads to process 6 as well):

$$(h_1[6];p).H_1' \mid (h_2[2];p).H_2' \mid (o_1[5], o_2[2], n).O_1' \mid$$
$$(h_3[3];p).H_3' \mid (h_4[5];p).H_4' \mid (o_3[3], o_4[6], n).O_2')$$

If we swap keys 5 and 6, these processes are syntactically identical (swapping is a short name for changing both keys to the other key via a third variable to avoid conflicts). Clearly this is possible, since as long as keys link the same actions, their naming is arbitrary. Figure 2.3 implies several such changes in order for the transitions to be valid.

Secondly, we can also disregard subscripts (which were introduced to distinguish copies of atoms of the same type). Informally we can say that if we remove the subscripts on the actions and process names and change keys accordingly, then only

two processes can be distinguished. In Chapter 6 we will call this chemical process equivalence. To demonstrate this we use processes 1 and 14. Process 1 is

$$(h_1[1];p).H_1' \mid (h_2[2];p).H_2' \mid (o_1[1],o_2[2],n).O_1' \mid$$
$$(h_3[3];p).H_3' \mid (h_4[4];p).H_4' \mid (o_3[3],o_4[4],n).O_2'$$

and becomes (with keys and order of processes unchanged):

$$(h[1];p).H' \mid (h[2];p).H' \mid (o[1],o[2],n).O' \mid$$
$$(h[3];p).H' \mid (h[4];p).H' \mid (o[3],o[4],n).O' \tag{1}$$

Process 14 is

$$(h_1[6];p).H_1' \mid (h_2[8];p).H_2' \mid (o_1[5],o_2[7],n).O_1' \mid$$
$$(h_3[5];p).H_3' \mid (h_4[7];p).H_4' \mid (o_3[6],o_4[8],n).O_2')$$

and becomes (with keys 6, 8, 5, 7 being changed to 1, 2, 3, 4 respectively and the two processes representing oxygen atoms being swapped)

$$(h[1];p).H' \mid (h[2];p).H' \mid (o[1],o[2],n).O') \mid$$
$$(h[3];p).H' \mid (h[4];p).H' \mid (o[3],o[4],n).O' \tag{2}$$

As we can see the resulting processes (1) and (2) are syntactically identical.

We will elaborate on these further in Chapter 6, for now we notice that the subscripts allow us to distinguish cases which would otherwise be written identically. On the other hand, the naming of keys has no semantic meaning. Overall there are two possible states in a system of two oxygen atoms and four hydrogen atoms if keys and subscripts are disregarded completly, 14 different states if subscripts are taken into account (these states are shown in Figure 2.3), and an infinite number of states if keys from $\mathbb{N}$ are considered different.

Example 2.1. Note that in water the n of O_1 can combine with the p of one of its hydrogen atoms, say H_1. Due to the property of the operator $(h_1[1];p).H_1'$, this must be followed immediately by breaking the bond 1 on h_1 with O_1, giving

$$((h_1;p[5]).H_1' \mid (h_2[2];p).H_2' \mid (o_1,o_2[2],n[5]).O_1')) \setminus \{h_1,h_2,h_3,o_1,o_2\}.$$

Now the system is converted to a system which is equivalent to the original formulation of water by promoting the bond 5 to a strong bond:

$$((h_1[5];p).H_1' \mid (h_2[2];p).H_2' \mid (o_1[5],o_2[2],n).O_1')) \setminus \{h_1,h_2,h_3,o_1,o_2\}$$

So our calculus allows a reaction which does not change the actual state of the system. We have replaced the original key 1 with a new key 5, but we will see in Chapter 6 that this change is not indicating an actual change in the system.

Remark 2.1. It should be noted that there are two types of bonding modelled here. Firstly, we have the initial bonds where two atoms contribute an electron each. Secondly, the *dative or coordinate bonds* are formed where both electrons come from one atom (an oxygen atom in this case). Both are *covalent bonds*, and once formed they cannot be distinguished. Specifically, in the oxygen atom with three bonds all bonds are the same and no distinction can be made. If one of the bonds is broken by a deprotonation (as in the autoprotolysis of water) the two electrons are left behind and they form a lone pair. If the broken bond was not previously formed as a dative bond, the electrons changed their "rôle". This is done in our modelling by promoting the bond from the n action to the o action. For the hydrogen atom we have a similar situation: Once the proton transfer has happened, the new bond is the same as the old one, so we perform the promotion. We can have only one bond at a time on the hydrogen atom, therefore we need the ; operator to model this situation, whereas the oxygen atom can have two or three bonds, so we model all bonds as a multiset.

2.3 Conclusion

We have seen in this chapter that we can model a simple covalent chemical reaction with our new calculus. The main feature of the new calculus is a new prefix operator $(s;b).P$. Whilst we have not yet given a formal definition we could already see the possibilities of the calculus and some challenges we may encounter when working with it.

Chapter 3

Process calculi and the modelling of biochemical reactions

In this chapter we first provide the basics of process calculi and rule-based systems to set the foundations for further discussions. We then provide a literature review focusing on the issues outlined in Chapter 1. After reviewing the literature we discuss how our approach is different to what has been done so far. In order to understand the differences better we model the reaction from Chapter 2 in various calculi.

3.1 Basics of process calculi and rule-based calculi

In Chapter 2 we used a certain syntax to model a network of chemical reactions. This was based on process calculi without providing an exact definition of the concepts of process calculi and other modelling options. Therefore, we give an introduction into *process calculi* here and contrast them with *rule-based systems*. We explain process calculi using *Calculus of Communicating Systems* (CCS) [79, 78]. The basic concepts are similar in other flavours of process calculi.

The basic entities of CCS are *agents* or *processes* (we will generally use the term process here). Processes offer *actions* (also called *channels* in some process calculi [80], p. 27) to perform. If two processes are composed in parallel and the actions on two processes match, the processes can do a *synchronisation*. A process with an action ready to perform is written as $a.P$. If it is in parallel with a process where

$\overline{a}$ is ready, these processes can sychronize. The overall system is then performing a transition.

Formally, the syntax of CCS is as follows, where P and Q are typical process terms:

$$P ::= \mathbf{0} \;\Big|\; S|S \stackrel{def}{=} P \;\Big|\; a.P \;\Big|\; P+Q \;\Big|\; P|Q \;\Big|\; P[a/b] \;\Big|\; P\backslash L, L \subset \mathcal{A}$$

$\mathbf{0}$ is a deadlocked process, which can do nothing. S is a member of the set of process identifiers or constants $\mathcal{PI}$, which have a defining equation $S \stackrel{def}{=} P$. A term $a.P$ is the prefixing operator. a is a member of set $\mathcal{L}$, where $\mathcal{L} = \mathcal{A} \cup \overline{\mathcal{A}}$. $\mathcal{A}$ is the infinite set of action names typically ranged over by a, b, c... and $\overline{\mathcal{A}}$ is the set of co-names typically ranged over by $\overline{a}$, $\overline{b}$, $\overline{c}$... $P + Q$ indicates a choice to execute one of two processes. $P \mid Q$ represents two processes in parallel. Processes $P[a/b]$ and $P \setminus L$ represent P with a relabelling $[a/b]$ or restriction $\backslash L$ applied to it.

Example 3.1. We use an example to demonstrate the execution of a CCS process. Our process is defined as:

$$(a.\mathbf{0} \mid \overline{a}.\mathbf{0})\backslash\{a\}$$

We have two processes in parallel. Both have an action ready to perform, shown by the use of the prefix. Since the actions $\overline{a}$ is the co-name of a, the two processes can synchronise. The restriction prevents a and $\overline{a}$ from happening on their own, so a must always be part of a synchronisation. For communications, also called *internal* or *silent* actions, we introduce the special label τ. We define $Act = \tau \cup \mathcal{L}$. Therefore, in our example the transition is:

$$(a.\mathbf{0} \mid \overline{a}.\mathbf{0})\backslash\{a\} \stackrel{\tau}{\rightarrow} (\mathbf{0} \mid \mathbf{0})\backslash\{a\}$$

In order to formalise the transitions operational semantics can be used. Structural Operational Semantics (SOS) have been introduced in [95]. It defines the behaviour of a process by giving all possible transitions. The processes and transitions define a graph with processes as nodes and transitions as edges. More formally, such a graph is defined by a labelled transition system (LTS). An LTS is a structure $(St, AL, \rightarrow : \subseteq St \times AL \times St)$ with St representing the set of states, AL the set of action labels and $\rightarrow : \subseteq St \times AL \times St$ the labelled transition relation. For CCS, the set of states St is the set of processes. The action labels are Act. The labelled transition relation is defined by SOS rules. The SOS rules for CCS are given in Figure 3.1.

$$\textsf{act}\ \frac{}{a.P \xrightarrow{a} P} \qquad \textsf{sum1}\ \frac{P \xrightarrow{a} P'}{P + Q \xrightarrow{a} P' + Q}$$

$$\textsf{sum2}\ \frac{Q \xrightarrow{a} Q'}{P + Q \xrightarrow{a} P + Q'} \qquad \textsf{com1}\ \frac{P \xrightarrow{a} P'}{P \mid Q \xrightarrow{a} P' \mid Q}$$

$$\textsf{com2}\ \frac{Q \xrightarrow{a} Q'}{P \mid Q \xrightarrow{a} P \mid Q'} \qquad \textsf{com3}\ \frac{P \xrightarrow{a} P' \quad Q \xrightarrow{\overline{a}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

$$\textsf{res}\ \frac{P \xrightarrow{a} P'}{P\backslash L \xrightarrow{a} P'\backslash L}\ \{a, \overline{a}\} \notin L \qquad \textsf{con}\ \frac{P \xrightarrow{a} P'}{S \xrightarrow{a} P'}\ S \stackrel{def}{=} P$$

$$\textsf{rel}\ \frac{P \xrightarrow{a} P'}{P[b\backslash a] \xrightarrow{b} P'[b\backslash a]}$$

Figure 3.1: SOS rules for CCS.

In order to show the validity of a transition we can use an inference tree. This gives the SOS rules which are used to build a particular transition. The inference tree for our example is shown in Figure 3.2.

$$\textsf{res}\ \frac{\textsf{com3}\ \dfrac{\textsf{act}\ \dfrac{}{a.\mathbf{0} \xrightarrow{a} \mathbf{0}} \qquad \textsf{act}\ \dfrac{}{\overline{a}.\mathbf{0} \xrightarrow{\overline{a}} \mathbf{0}}}{a.\mathbf{0} \mid \overline{a}.\mathbf{0} \xrightarrow{\tau} \mathbf{0} \mid \mathbf{0}}}{(a.\mathbf{0} \mid \overline{a}.\mathbf{0})\backslash\{a\} \xrightarrow{\tau} (\mathbf{0} \mid \mathbf{0})\backslash\{a\}}$$

Figure 3.2: Inference tree for the transition $(a.\mathbf{0} \mid \overline{a}.\mathbf{0})\backslash\{a\} \xrightarrow{\tau} (\mathbf{0} \mid \mathbf{0})\backslash\{a\}$.

As we have seen the SOS rules work on the structure of the processes. They do not make any assumptions about particular actions in the processes being used. Rule-based calculi, on the other hand, have rules which tell how particular systems can behave. As an example, we use the *Calculus of Chemical Systems* [96]. This works on chemical species typically identified by capital letters.

Example 3.2. Two chemical species E and S, acting in parallel, are an example of a process in the Calculus of Chemical Systems. A rule for a complexation of E and S would then be:

$$E + S \to E - S$$

where $E - S$ is the new product. The rule here gives the particular species it is working on and the result is a new species, for which we would need new rules if we want it to do anything useful. In contrast SOS rules work for a multitude of processes provided they have the right structure.

3.2 Related work

3.2.1 Early developments in chemistry and computer science

Chemists have started to look at the speed of reactions and the rates achieved as soon as the concept of chemical reactions was established.[1] Since chemical reactions can be interlinked (they may use the same reactants, the product of one reaction is the reactant of another one, and there may be circles involving several steps), there can be complicated interactions. Such a system of compounds linked by reactions has a particular behaviour over time, and this can be modelled using a set of ordinary differential equations (ODEs). Starting from defined concentrations of each compound in a system, the ODEs can then be used to compute how the concentrations evolve over time and which concentrations exist in a stable state of the system (assuming there is one). An important milestone was achieved by Joseph L. Doob and others and popularized by Dan Gillespie (hence the name "Gillespie algorithm") in [41], enabling the fast calculation of such ODEs with limited computing power. Models based on ODEs were widely applied in chemistry as well as in biology and various techniques have been used to enable the simulation of complex systems with large number of species and molecules. A good overview of the use of ODEs in chemistry is given in [44]. Whilst such methods accurately show the dynamic behaviour, it was soon noted that they do not deal with the objects involved. In particular, any properties of the objects, which make reactions possible, are not considered. This became even more important when biochemical processes, which do not only involve small molecules, but also macromolecules, cells, and membranes, were modelled. This lack of understanding of the objects was raised in [37] and the use of computer science methods like the λ-calculus was suggested.

In computer science, process calculi had been established as a means of formally modelling concurrent systems. The principles of process calculi were explained in Section 3.1. Popular process calculi include Calculus of communicating systems (CCS) [78, 79], Communicating Sequential Processes (CSP) [46], and the π-calculus [80]. In a process calculus, the processes contain information about what they can potentially do, so this seems to fulfill the suggested need to have a better modelling of the objects involved.

[1] An overview is given in [109], pp. 11 ff.

3.2.2 Process calculi for chemistry and biology

A connection between process calculi and chemistry has been made in [7]. Here, the Chemical Abstract Machine (CHAM) is introduced as a method to define the semantics of a calculus. As opposed to other semantics for calculi, the algebraic terms ("molecules") can interact with each other no matter what order they are written in - this is similar to molecules being in a solution. CHAM also introduces membranes to group atoms and rules which simulate heating and cooling of solutions. The actual rules determining the reactions must be defined for a particular CHAM. Therefore, a CHAM can be used to execute processes of arbitrary process calculi. A CHAM can also model actual chemistry depending on the rules used.

Starting with Regev et al. [100, 103, 104, 105, 102] process calculi, specifically the π-calculus, were used to model biochemical systems. For this the biological units in question are represented as processes, their possible actions are modelled as channel ports and binding and unbinding is represented as establishing and breaking a communication, respectively, on a channel. So there is an analogy between processes in concurrent computing and biological entities in natural systems, and between communication in computer systems and interactions between entities in biological systems. In [105] proteins were modelled as collection of processes, the functional subunits of proteins (called domains) as processes with the active parts of domains, the residues, being (loosely) represented by actions. Since the π-calculus allows message passing, modifications can be modelled by passing appropriate channel names. Restriction is used to model molecular complexes and compartments. The aim of this work as stated in [104] was mainly the representation of biological knowledge and to enable computer-based analysis of these representations. A simulation software BioSPI [98, 100] was also introduced.

In [100, 103, 104, 105] only the reactions taking place were represented and not their rates. Since the rates can be vastly different and make some reactions irrelevant even if they are possible, it seemed a natural extension to include reaction rates in the model. This was done in [98] using the stochastic π-calculus, which was introduced previously in [97]. Here, for each reaction a rate is given. In a BioSPI simulation, the rates together with the concentrations of the molecules are used to decide which reactions are executed if there is a choice. This enables the simulation of the concentrations and of the overall state of the system over time.

With a framework which includes reaction rates, it was possible to simulate systems of interlinked reactions and to see how they react to changes. This is relevant for medical research since treatments using drugs change some of the reactions forming

the overall biochemistry of the organism, aiming for an effect not necessarily directly connected to that change. Many drugs for example block a binding site on a molecule and so reduce the rate of a particular reaction (a complete suppression is usually not possible, so a very low rate is considered a blocking). Since, as we have seen, the network of reactions can be very complicated, the effects of such a change to the final results of the network are complicated to determine. A good model of the system in a calculus including reaction rates can be used to test in silico the effects of a drug on the biological system, which is the ultimate aim of the discipline of systems biology. A typical example is [4], where the reactions involved in severe asthma are simulated and the influence of inhibitors is calculated to find a good potential drug. The software used here is Bio-PEPA [15].

Bio-PEPA is an extension of PEPA [45], which was intended for performance analysis of computer systems. PEPA combines a process algebra with timing information and Bio-PEPA uses this to represent reaction rates. In a Bio-PEPA simulation the entities modelled are species. Every interaction of molecules creates a new species, and there is no tracking of different "original" components. The transitions between the entities model the transformations between species. There is no tracking of mechanisms or modelling of the underlying causes. The focus is on the rates and the development of concentrations over time. In that respect Bio-PEPA is similar to previous work done on reaction dynamics, using ODEs and Gillespie's algorithm.

Modelling biological processes using process algebras has been applied to various areas since the original publications. Apart from biochemical interactions (e.g metabolic pathways [9], gene regulatory networks [62, 76], the cell-division cycle [10], inflammatory processes [68], RNA synthesis [16]), also processes involving organisms as entities were modelled, e.g. immune defence [42], epidemiology [114, 77], or ecological processes [111, 112, 113]. On the other hand, also the calculi used have been extended with new elements of syntax and associated rules for more realistic modelling of biological and chemical examples. Apart from the languages based on process calculi mentioned so far, the following calculi were suggested:

3.2.2.1 P Systems

Biological processes take place in separated compartments (e.g. cells) and the possibility of interaction depends on the compartments in which the molecules are. The first attempt at capturing this was done with P system [87]. A P system contains several membranes, which can be inside each other. For every membrane a set of rules is defined, making P Systems a rule-based system. Objects are located inside

membranes, are processed by rules, can enter and leave membranes, and membranes can be dissolved. In the original definition, membranes cannot be created, this feature was added in later work [83]. P systems were later extended to model various properties of systems, for example P systems with symport/antiport [86] add the capability to only allow certain objects in a defined direction to pass. An example of a P system is given in Section 3.4.5.

3.2.2.2 BioAmbients

BioAmbients [101] is based on the Ambient calculus [12], which was originally devised for mobile computing outside the biological context and is an extension of the π-calculus. Here, similarly to P systems, processes can be inside compartments (and compartments can be in other compartments as well). The ability to enter, leave or merge compartments is modelled by channels, and communications can be restricted to processes in the same, neighbouring or parent/child compartments. BioAmbients is a process calculus, as opposed to P Systems.

3.2.2.3 Brane Calculi

Brane Calculi ("brane" is short for membrane) [11], a family of process calculi, model membranes not only as compartments that define which interactions can happen, but also as part of the interactions. As a result, the membranes can be transformed to gain new capabilities. Typical examples are viruses entering cells by interacting with their membranes. The Projective Brane Calculus [29] is a variant where interactions are directed outward or inward from a membrane.

3.2.2.4 Language for Biochemical Systems/Calculus of Chemical Systems

The Language for Biochemical Systems (LBS) is introduced in [88]. It is based on a "Calculus of Chemical Systems"[96]. This is a rule-based system, where reaction rules are explicitly given. A particular feature is modularity, meaning that a certain pattern (for example a signalling cascade) can be modelled abstractly and then be instantiated with certain compounds instead of the variables used in the pattern. The "Calculus of Chemical Systems" is explained in Section 3.1.

3.2.2.5 BIOCHAM

The Biochemical Abstract Machine (BIOCHAM) [35, 34] is a rule-based system for biological networks. It also models temporal properties of a system in a logic based language and enables validation of the temporal properties. Extensive software support is available [1] for BIOCHAM.

3.2.3 Reversibility in computer science

In the 1950s, it was discovered that a certain amount of energy is dissipated into the environment of a computing machine if information is lost, which became known as *Landauer's principle* (suggested by Rolf Landauer in [63], although earlier work exists). This drew attention to *reversible computing*, which would not loose information and be therefore much more energy-efficient. Early theoretical studies in the area were focused on reversible finite state machine [51], reversible logic [63], and reversible Turing machines [69].

Following from this, work has been done to better understand reversibility, and apply it. In the area of applications, attempts have been made to build reversible circuits, which are of particular significance in the area of quantum computing. Since this thesis does not deal with the area, we recommend [74] for an overview.

In the area of programming languages, there have been several attempts at reversible languages. The most prominent one of these is Janus, originally started in the 1980s, and recently continued at Copenhagen University [123]. Other examples are R [39, 73]. In a reversible language, every statement is reversible, i.e. can be run in both directions. That excludes certain operations (most prominently the assignment), which have to be replaced by reversible operations - for example the assignment would be replaced by repeated increment/decrement operations or a swap, which are reversible. Reversible languages are not directly Turing complete, since all functions implemented must be injective, whereas Turing machines can compute non-injective functions. It has been shown that using some methods, most prominently Landauer embedding and Bennett's method, all functions can be made injective [2]. Janus has also been extended to include features of an object-oriented programming language, whilst still being reversible, leading to the ROOPL language [43].

Another strand of research is the extension of conventional programming languages to become reversible. A good overview is given in [71]. Generally speaking, this

can be done by saving previous states when progressing with the computation. This technique, known as *checkpointing*, is used e.g. in [31] to make multiparty sessions reversible. Recent work on reversing imperative programs is presented in [47] and extended in [48] to reverse parallel C-like programs. Here, undo information is saved locally for individual statements, making reversing independent from checkpoints.

Reversibility has also been explored for process calculi (see 3.2.4). For Turing machines, [6] has shown that any computation of a Turing machine can be simulated reversibly, with increased requirements for space and time resources. Reversible cellular automata have been used to simulate reversible circuits [81]. Finally, Petri nets have been studied for reversibility. Traditionally, a reversible Petri net is defined as being able to return to its inital markings from any state, thus reversibility is a global property. In contrast, the possibility to undo specific computations is a local property [5, 89].

The reversible languages and models mentioned have a range of applications. A significant area is the possibility for error recovery, contributing to dependability of software and systems. A typical example is reversible robot control [107], for which the domain-specific language RASQ has been proposed [108]. Reversibility has become standard in databases, since transactions, which can be undone, are widely used. In software development, a comparable technique is reversible debugging. In a reversible debugger, a developer can go back and forth in his code during debugging, which avoids restarting the program to get to a previous state [32]. UndoDB is a commercially available reversible debugger for C/C++ [118].

3.2.4 Reversibility and causality

The calculi described in so far do not explicitly address reversibility. This does not mean that they do not include reversibility, since the obvious cases of reversibility in natural processes make modelling reversibility necessary for a faithful modelling, but there is no explicit relationship between doing and undoing one action. For example, if we have a ligand binding to a protein and, perhaps after some other useful actions, unbinding from it this could be modelled in a CCS-like calculus like this:

$$Ligand \stackrel{def}{=} bind.x.unbind.Ligand$$

Here, the ligand can bind (given there is a complimentary action available), then perform its task x and unbind. Note that, if several copies exist, the unbind action may only unbind from where it is actually bound to. In the π-calculus a private

channel can achieve this, as demonstrated for example in [103], p. 235. Of course the unbinding is the inverse of the binding, but this is not expressed in the calculus. A human reader can infer this from the action names, but in the calculus the names are arbitrary. Since the actions done so far determine what can potentially be undone, it seems that a closer modelling of this dependency might give new insights into the underlying mechanism of the processes.

The first attempt at this was RCCS ("R" indicating reversibility) [26], later extended to include recursion and called CCS-R (we will use CCS-R in the following) [27]. These are reversible calculi based on CCS. Reversibility is achieved by adding memory to the processes. These memories record which actions have been done and also any non-deterministic choices (+) that have been discarded. By using the information from the memory, it is possible to undo past actions and restore discarded branches, so that it is ultimately possible to go back to the state in which the process started. This system of memories has consequences with respect to *causality* of the undoing process. In an execution, an action a causes action b if in all possible paths a comes before b. In a CCS-like calculus, causality exists for example between prefixes: Clearly, in the process $a.b$ action a must happen before b in all executions. In the reverse trace therefore b must be undone before a. If undoing is done in the reverse order of the forward path it is called *backtracking*. It may still be possible to undo in a different order if actions are not *causally dependent*. For example, the process $a \mid b$ can clearly perform the traces $a.b$ and $b.a$. In CCS-R the undoing could take both paths no matter which path the forward transitions took.

Example 3.3. The process would be in CCS-R:

$$<> (a|b).\mathbf{0}_1$$

The execution of b and a in this order makes the process:

$$<> (a|b).\mathbf{0}_1 \xrightarrow{1,r,b} \xrightarrow{1,s,a} (< (s,a)|(r,b) >)\mathbf{0}_1$$

From here we could reverse the order when going back to the initial state:

$$(< (s,a)|(r,b) >)\mathbf{0}_1 \xleftarrow{1,r,b} \xleftarrow{1,r,a} <> (a|b).\mathbf{0}_1$$

This principle of reversibility is called *causally consistent* reversibility. It was shown for CCS-R in [26]. Moreover, it is often understood as a general characteristic of reversible calculi, e.g. in [21], p. 3. There are many instances of causally consistent processes in biology. An example is a protein which can prevent the working of two

other proteins by binding to their active sites and making them unavailable. Such a protein needs to be able to bind to one of the two proteins it blocks, unbind (which is a reversing of the binding) and then bind to the other protein to block. Clearly here the system goes back to a state in which it has been before, and from there it takes an alternative path. An example of the use of CCS-R is given in Section 3.4.3.

Another reversible calculus based on CCS is CCSK, introduced in [94] and [90]. CCSK is derived from CCS by a procedure which can be used to derive a reversible calculus from any standard process calculus with SOS rules. Instead of memories, the actions are decorated with keys indicating which actions took place and potentially which communications happened. Any alternative paths from the choice operator + are left in place. In this way, all information about the past and all potential futures are in the process term. CCSK offers causally consistent reversibility, similar to CCS-R. An example of the use of CCSK is given in Section 3.4.1. Here, as well as in CCS-R, the communications done are remembered and therefore only those can be undone, avoiding the need for private channels as in the π-calculus.

Looking at biological processes, a restriction to causally consistent reversibility seems too narrow, considering the example in Chapter 1, which clearly shows *out-of-causal order* reversibility. An attempt to incorporate this into a calculus was made in [93]. The process is split here into two parts: The actual process and a controller. A new type of prefix operator is used, where several actions are grouped in a set of actions. In such a group the actions can be done in any order, and only once all actions are done we can (potentially) progress to the next prefix. The controller gives the order(s) in which the actions can be performed. Effectively the process only holds the information about its current state, whereas all information about what can happen is in the controller. Due to this there is not restriction to backtracking or causally consistent undoing. This calculus is very flexible, but splits information in two entities. Ideally, the modelling of the process should be so that the order of actions emerges out of it. An example of the use of CCSK with controllers is given in Section 3.4.2. Out-of-causal order computation was also studied in [92, 91].

There is a distinction made between *controlled* and *uncontrolled reversibility* in [67, 65]. In uncontrolled reversibility, of which CCSK and CCS-R are examples, there is no hint to when a reverse execution should be preferred over a forward execuction, but there is information which actions can potentially be undone. In controlled reversibility, represented by [93], there is a specification of when the calculation should go forwards or reverse. In contrast in our calculus restricts the choice of reversible actions in a subprocess depending on the state of this subprocess. We therefore call this *locally controlled reversibility*.

Reversibility for the π-calculus was addressed in [64, 65, 66] and in [20].

The κ-calculus [28] is based on graph-rewriting. Here, entities (e.g. proteins) are defined as having several sites, which can potentially bind to other entities. The rules tell that a link can be formed or broken if the sites in the involved entities are in certain states. These rules allow a very flexible modelling of events and causalities. They make κ-calculus a rule-based calculus. Similarly to CCSK with controllers, we have a separation between entities and rules. Here the rules comprise all entities involved, whereas in CCSK with controllers each controller is local for an entity. An example of the use of κ-calculus is given in Section 3.4.4. Software support for the κ-calculus is available, e.g. KaSim [8].

As we have seen, calculi imply certain types of causality. Generally speaking, there are the following possibilities if we look at two particular events (which we call a and b) in a system:

- Causality: a causes b if b can only happen once a has happened.
- Conflict: a and b are in conflict if only one of them can happen.
- Concurrency: a and b are concurrent if they can happen in any order.

We also get backtracking, causally consistent reversibility and out-of-causal-order reversibility as the types of reversibility in a reversible calculus. We have seen that some calculi imply certain types of reversibility. A framework explicitly modelling events are *event structures* [121]. In an event structure the events and their causal dependency (and conflict in some cases) are given. An event structure can be translated into a transition system and vice versa [120]. Various calculi have either been extended to include causal modelling [23] or have been examined with respect to their causal expressiveness [25, 24].

Finally, there is a wide range of works from biology and chemistry which are loosely related to the field. Biological modelling implies the use of some type of language (even if not formally defined or named). Apart from informal verbal and graphical models, Systems Biology Markup Language (SBML) provides a standardized, formal language for modelling [50]. There is extensive software support for SBML. Ideally, such a model should be close to modelling in mathematical languages and a translation should be possible. In chemistry, there are reaction prediction systems, which try to tell the possible reactions in a system based on chemical knowledge (a typical example was the software EROS [49]). From there it is a logical step to ask

if the transitions in a biological systems (including proteins) can be inferred from some general rule. Research in that area has indeed been performed, in biology the automated reconstruction of pathways has been attempted. Success in that area has been limited though, due to the complexity of the systems in question. The algorithms therefore often work without explicit rules. A typical example is [56] which a) generalizes the molecules into fingerprints and b) uses Support Vector Machines (SVM) to find rules about which reactions happen. This means that explicit rules are not visible, making it very difficult to learn about the structure of the rules.

3.3 Chemistry in process calculi

As we have seen in Section 3.2 process calculi were used to model biological processes. The focus often was on quantitative modelling and not so much on the structure of the networks. In many cases, the network is fully described, sometimes even with reverse reactions given explicitly. In an actual biolological system, the network is an emergent property of the entities involved. So ideally having modelled all the entities, the network should emerge out of this. We want to try to do this as much as possible in this thesis. Therefore we focus on modelling how the network is built from the components instead of the behaviour of a pre-defined network.

For biological reactions such a modelling is difficult to do. The entities involved are very large and complex (a protein contains hundreds to tens of thousands of amino acids, each composed of around 20 atoms) and their functions are the result of complex interactions on several levels. Whilst there is extensive research on biological processes, we are still far away from an understanding which would enable us, for example, to infer the functions of a protein from its structure.

We have therefore decided to look at classical chemical reactions. These are the basis for the more complicated biological reactions and there has been research into how and why they work for a long time. Even though our understanding of chemical mechanisms is not complete, it seems a more appropriate area for our research.

Looking at chemical reactions, there are several ways in which atoms, molecules and their interactions can be viewed. The most accurate view (according to current research) are quantum chemical computations. Here, the electronic properties of atoms are calculated, using the Schrödinger Equation, and then combined to get electronic properties of a molecule. These determine the chemical properties of a molecule, including its bonding abilities. Unfortunately, quantum chemistry

computations for multi-atom systems are computationally expensive or, in practice, impossible for larger molecules. Various approximation methods have been developed. These include molecular orbitals (MO) and valence bonds (VB) methods. Still this is a very low level modelling and does not seem suitable for an approach using process calculi.[2]

Chemists had already tried to find rules for reactions before quantum theory was available. These rules are of course approximations, but they have served science and industry alike. There is a vast array of such approximations, but many are based on the concept of atoms consisting of nuclei and electrons around them in shells. The electrons can be shared between atoms, by this forming a *bond*. This model is combined with a set of rules. A typical example is the octet rule, which says that an atom has the most stable bonds if it has eight (hence the octet) electrons in its outer shell.[3] For example carbon has four electrons in its outer shell, so it can form four bonds to gain four additional electrons. Like all approximations there are limits (the octet rule only applies to carbon, nitrogen, oxygen, the halogens, and some metals), but the rule is still useful. Such approximations are also something we can model using process calculi, because there are entities and interactions.

The interactions between atoms are normally called bonds. The most important type of chemical bonds and the one we are going to model are *covalent bonds*. Covalent bonds are intramolecular bonds, i.e. they are formed between atoms which are part of the same molecule. In a covalent bond two electrons are shared between two atoms, the electrons forming the bond. As we have seen in Chapter 2, there are two ways for a covalent bond to form: If there are two atoms which both expose a single electron they can react with each other. The other possibility is that one atom has an empty site and another atom a pair of two electrons (a *lone pair*) exposed. The two electrons then form a *dative covalent bond*. There are other intramolecular bonds, namely metallic bonds (where electrons are shared between a multitude of atoms) and ionic bonds (where atoms are negatively or positively charged and bond by the electronic attraction). Finally there are intermolecular bonds, where atoms in different molecules interact without the molecules forming a single molecule.[4]

Since the chemical rules mentioned above focus on atoms and bonds as entities, we decided to model chemical reactions by treating atoms as entities and bonding as communication events, similar to what is done in the biological examples. We will

[2]For an overview of computational chemistry and quantum chemistry see [72], in particular Chapter 4.3 discussing molecular orbitals (MO) and valence bonds (VB) methods.

[3]See [13] Chapter 9.4 for details.

[4]For details on chemical bonds see [13] Chapters 9.2, 9.4, and 12.5.

try to model chemical rules that are generally used to get a notation which enables transitions which are as close as possible to what is possible in reality.

As noted we will model covalent bonding. This involves sharing of electron pairs between atoms. Since each of the atoms involved contributes to the bond this can be modelled by actions faithfully. Also dative covalent bonding can faithfully be modelled by actions on atoms. It should be mentioned that there are subcategories of covalent, for example π- and σ-bonds (which are the most common ones), but for our purposes we do not distinguish them.

In this research we have decided not to include other types of bonds than covalent bonds. Also intermolecular forces will mostly not be considered with the exception of hydrogen bonds. Since hydrogen bonds play a major rôle they will be dealt with. Other intermolecular forces, which are generally of limited interest, are not included.

Against this background we have decided to model atoms as entities and their interactions as actions in some process calculus suitable for this purpose. A particular point of attention will be how reactions emerge out of the entities and what type of causality we find there. Looking at what has been done so far, one possibility would be to use the π-calculus. This offers mobility, which, since it allows to model changes to the processes, has been useful in the biological examples, where states of entities change. In our case, we would like to keep our entities immutable, since we have chosen to model atoms where changes to state are not a natural concept (except by bonding, but this will be covered by the actions). Therefore, a calculus based on CCS seems more appropriate. Also the work done for CCSK and CCS-R can be used as a starting point. This also implies that our calculus will not be rule-based (as for example Bio-PEPA or LBS) but it will be based on a process calculus.

Finally, it should be noted that we have deliberately decided not to model the electrons as entities. In our approach they are modelled in the actions respectively bonds. Also, our modelling is not on molecular level, but on atomic level. This is because chemical reactions start with atoms interacting, which leads to a change on molecular level. In our case, this behaviour should be emergent from the atomic interactions.

3.4 Modelling of autoprotolysis of water in process calculi

In Chapter 2 we have modelled a simple reaction of two water molecules using our new calculus. In order to better understand what existing calculi can do and how our new calculus compares to them we model the autoprotolysis of water in each of the existing process calculi.

3.4.1 CCSK

We need a reversible calculus to model the process, since a bond is first formed and later broken during the process. Firstly we look at a reversible version of CCS, derived by the procedure given in [90]. This calculus, called CCSK, keeps all properties of CCS, but adds reversibility.

The syntax of CCSK [90] is as follows:

$$P ::= \mathbf{0} \;\Big|\; a.P \;\Big|\; a[k].P \;\Big|\; P + Q \;\Big|\; P|Q \;\Big|\; P[b/a] \;\Big|\; P\backslash a \;\Big|\; S|S \stackrel{def}{=} P$$

where $\mathbf{0}$ is the zero agent, $a.P$ is the action a followed by P, $a[k].P$ is the past action a with key k followed by P, $P + Q$ is the choice of P or Q, $P \mid Q$ is P in parallel with Q, $P[b/a]$ is P with all actions a renamed to b, $P\backslash A$ is P restricted by A and S is an agent identifier defined by another agent.

Semantics are provided by SOS rules. There are forward and reverse SOS rules, given in Figures 3.3 and 3.4. The predicate std is similar to std in Figure 4.1 in Chapter 4.1 and ensures the rules only apply for reachable processes. The predicate fsh is similar to fsh in Figure 4.1 in Chapter 4.1and prevents keys from being used twiced. This calculus is similar to CCS, but it adds keys as a way to memorize past actions instead of dropping them.

To model the system of two water molecules in CCSK, we still need oxygen and hydrogen atoms. The modelling most faithful to the real situation is this:

$$\begin{aligned} H &\stackrel{def}{=} \overline{a}.H' \\ O &\stackrel{def}{=} a.a.a.O' \end{aligned}$$

Here, an oxygen atom is modelled with three prefixes and a hydrogen atom with the corresponding co-name. As in CCS, these can synchronise. A synchronisation

$$\frac{\mathsf{std}(X)}{a.X \xrightarrow{a[m]} a[m].X} \qquad \frac{X \xrightarrow{b[n]} X'}{a[m].X \xrightarrow{b[n]} a[m].X'}\ m \neq n$$

$$\frac{X \xrightarrow{a[m]} X' \quad \mathsf{std}(Y)}{X + Y \xrightarrow{a[m]} X' + Y} \qquad \frac{Y \xrightarrow{a[m]} Y' \quad \mathsf{std}(X)}{X + Y \xrightarrow{a[m]} X + Y'}$$

$$\frac{X \xrightarrow{a[m]} X' \quad \mathsf{fsh}[m](Y)}{X \mid Y \xrightarrow{a[m]} X' \mid Y} \qquad \frac{Y \xrightarrow{a[m]} Y' \quad \mathsf{fsh}[m](X)}{X \mid Y \xrightarrow{a[m]} X \mid Y'}$$

$$\frac{X \xrightarrow{a[m]} X' \quad Y \xrightarrow{\overline{a}[m]} Y'}{X \mid Y \xrightarrow{\tau[m]} X' \mid Y'}$$

$$\frac{X \xrightarrow{a[m]} X'}{X\backslash A \xrightarrow{a[m]} X'\backslash A}\ a \notin A \cup \overline{A} \qquad \frac{X \xrightarrow{a[m]} X'}{X[f] \xrightarrow{f(a)[m]} X'[f]}$$

Figure 3.3: Forward SOS rules for CCSK.

(represented by a key) forms a bond until undone. In our modelling, the actions make sure the oxygen atom can bond to two or three hydrogen atoms as required and a hydrogen atom can bond to exactly one other atom. We can then model our system of two oxygen atoms and four hydrogen atoms:

$$(\overline{a}.H_1' \mid \overline{a}.H_2' \mid a.a.a.O_1' \mid \overline{a}.H_3' \mid \overline{a}.H_4' \mid a.a.a.O_2') \setminus \{a\}.$$

This system now can develop to two water molecules. This happens by four communications of a and $\overline{a}$, so that two hydrogen atoms are bonded to each oxygen atom:

$$(\overline{a}[1].H_1' \mid \overline{a}[2].H_2' \mid a[1].a[2].a.O_1' \mid \overline{a}[3].H_3' \mid \overline{a}[4].H_4' \mid a[3].a[4].a.O_2') \setminus \{a\}.$$

It can also develop to a OH^- and a H_3O^+. For this, three communications happen with one oxygen atom and the remaining hydrogen atom could go to the other oxygen atom:

$$(\overline{a}[1].H_1' \mid \overline{a}[2].H_2' \mid a[1].a.a.O_1' \mid \overline{a}[3].H_3' \mid \overline{a}[4].H_4' \mid a[2].a[3].a[4].O_2') \setminus \{a\}.$$

Any of these bonds can be broken by backtracking, which is the benefit of a reversible calculus. So we could go from the two water molecules to OH and a H_3O by breaking the bond between one the hydrogen atoms and the oxygen atom it is bonded to and re-bonding to the other oxygen atom. A problem is that three hydrogen atoms could

$$\frac{\mathsf{std}(X)}{a.X \overset{a[m]}{\rightsquigarrow} a[m].X} \qquad \frac{X \overset{b[n]}{\rightsquigarrow} X'}{a[m].X \overset{b[n]}{\rightsquigarrow} a[m].X'}\ m \neq n$$

$$\frac{X \overset{a[m]}{\rightsquigarrow} X' \quad \mathsf{std}(Y)}{X + Y \overset{a[m]}{\rightsquigarrow} X' + Y} \qquad \frac{Y \overset{a[m]}{\rightsquigarrow} Y' \quad \mathsf{std}(X)}{X + Y \xrightarrow{a[m]} X + Y'}$$

$$\frac{X \overset{a[m]}{\rightsquigarrow} X' \quad \mathsf{fsh}[m](Y)}{X \mid Y \overset{a[m]}{\rightsquigarrow} X' \mid Y} \qquad \frac{Y \overset{a[m]}{\rightsquigarrow} Y' \quad \mathsf{fsh}[m](X)}{X \mid Y \overset{a[m]}{\rightsquigarrow} X \mid Y'}$$

$$\frac{X \overset{a[m]}{\rightsquigarrow} X' \quad Y \overset{\overline{a}[m]}{\rightsquigarrow} Y'}{X \mid Y \overset{\tau[m]}{\rightsquigarrow} X' \mid Y'}$$

$$\frac{X \overset{a[m]}{\rightsquigarrow} X'}{X\backslash A \overset{a[m]}{\rightsquigarrow} X'\backslash A}\ a \notin A \cup \overline{A} \qquad \frac{X \overset{f(a)[m]}{\rightsquigarrow} X'}{X[f] \overset{f(a)[m]}{\rightsquigarrow} X'[f]}$$

Figure 3.4: Reverse SOS rules for CCSK. $\rightarrow$ indicates a forward transition, $\rightsquigarrow$ a reverse transition.

bond directly to an oxygen atom, without two waters ever existing. This would not happen in reality, since, as we have seen in Chapter 2, it is the partial charge of the water which enables the transfer. These charges are not at all modelled in this representation.

In order for the hydrogen transfer to happen, a spontaneous dissociation of a hydrogen atom would be needed. This could then bind to another oxygen atom thus completing the transfer. Whilst such dissociations exist, it is a different mechanism than the transfer, which is supported by the partial charge. The dissociation would also allow for protons to flow freely between water molecules, which is not the situation in reality. There could even be several water molecules loosing all of their protons without any of them being bonded, which is quite unrealistic. Finally, the hydrogen atoms bonded to one oxygen atom must dissolve in reverse order as they were bonded, which is not at all the case in the real world.

Overall, we can model the final products in CCSK, but the processes taking place are not well represented. The main reason for this is that the calculus is restricted to backtracking for undoing actions.

3.4.2 CCSK with execution control

CCSK is combined with an *execution controller* in [93] to achieve the behaviour to be modelled. The syntax of CCSK, as given above, is extended by a new multiset prefix to become:

$$P ::= \mathbf{0} \;\Big|\; (s).P \;\Big|\; P + Q \;\Big|\; P|Q \;\Big|\; P[b/a] \;\Big|\; P\backslash a$$

where s is a multiset of actions or past actions.

The SOS rules in Figure 3.3 and 3.4 are extended by the rules in Figure 3.5. The syntax of a controller is as follows:

$$C ::= c \;\Big|\; a.C \;\Big|\; a[k].C \;\Big|\; C + D \;\Big|\; C|D$$

The syntax of C is similar to that of a process, with c being a controller identifier, a an action from $\mathcal{A} \cup \overline{\mathcal{A}} \cup \underline{\mathcal{A}} \cup \underline{\overline{\mathcal{A}}}$, $a[k]$ a past action with a key, and $+$ and $|$ being the choice and parallel operators as in CCS. A process P controlled by a controller C is written as $P\langle C\rangle$. The SOS rules for controllers are the standard SOS rules for CCS, except that the prefixes include the new actions $\underline{\mathcal{A}} \cup \underline{\overline{\mathcal{A}}}$. The SOS rules for the control operator $\langle\rangle$ are given in Figure 3.6. The rules are Ordered SOS rules [116, 82], where $(\mathsf{cf1}) > (\mathsf{cf2})$ means that $\mathsf{cf2}$ may only be applied if no $\mathsf{cf1}$ rule is applicable. As a consequence a process P controlled by C is allowed to compute until the action to be executed in P can be executed in C. If an action from $\underline{\mathcal{A}} \cup \underline{\overline{\mathcal{A}}}$ can be undone in C, the process P can reverse until that action is undone.

$$\frac{\mathsf{std}(X)}{(\alpha, s).X \xrightarrow{\alpha[n]} (\alpha[n], s).X} \qquad \frac{X \xrightarrow{\mu[n]} X' \quad \mathsf{fsh}[n](s')}{(s').X \xrightarrow{\mu[n]} (s').X'}$$

$$\frac{\mathsf{std}(X)}{(\alpha, s).X \overset{\alpha[n]}{\rightsquigarrow} (\alpha[n], s).X} \qquad \frac{X \overset{\mu[n]}{\rightsquigarrow} X' \quad \mathsf{fsh}[n](s')}{(s').X \overset{\mu[n]}{\rightsquigarrow} (s').X'}$$

Figure 3.5: SOS rules for multisets. s is a sequence of actions or past actions. $\rightarrow$ indicates a forward transition, $\rightsquigarrow$ a reverse transition.

For our example the processes (atoms) could be modelled as follows:

$$\begin{aligned} H &\overset{def}{=} (\overline{a}, \overline{b}).H' \\ O &\overset{def}{=} (a, a, b).O' \end{aligned}$$

$$\mathsf{cf1}\ \frac{X \xrightarrow{\alpha[n]} X' \quad Y \xrightarrow{\alpha[n]} Y'}{X\langle Y\rangle \xrightarrow{\alpha[n]} X'\langle Y'\rangle} \quad > \quad \mathsf{cf2}\ \frac{X \xrightarrow{\beta[n]} X' \quad Y \xrightarrow{\alpha'[n]} Y'}{X\langle Y\rangle \xrightarrow{\alpha[n]} X'\langle Y\rangle}$$

$$\mathsf{cr1}\ \frac{X \overset{\alpha[n]}{\rightsquigarrow} X' \quad Y \xrightarrow{\underline{\alpha}[n]} Y'}{X\langle Y\rangle \overset{\alpha[n]}{\rightsquigarrow} X'\langle Y'\rangle} \quad > \quad \mathsf{cr2}\ \frac{X \overset{\beta[n]}{\rightsquigarrow} X' \quad Y \xrightarrow{\underline{\alpha'}[n]} Y'}{X\langle Y\rangle \overset{\beta[n]}{\rightsquigarrow} X'\langle Y\rangle}$$

Figure 3.6: SOS rules for the control operator $\langle\rangle$. $\rightarrow$ indicates a forward transition, $\rightsquigarrow$ a reverse transition.

Here, the new multiset prefixing operator is used. All actions or past actions in the multiset can be done or undone in any order. What actually happens is determined by a controller associated with each agent. The controller tells which actions the agent can do and in which order. The following controllers achieve the desired behaviour of our system:

$$\begin{aligned}
C_H &\overset{def}{=} a.C'_H \\
C'_H &\overset{def}{=} \underline{a}.C_H + b.\underline{a}.C''_H \\
C''_H &\overset{def}{=} \underline{b}.C_H + a.\underline{b}.C'_H \\
C_O &\overset{def}{=} a.a.C'_O \\
C'_O &\overset{def}{=} b.\underline{b}.C'_O + \underline{a}.C''_O \\
C''_O &\overset{def}{=} \underline{a}.C_O + a.C'_O
\end{aligned}$$

Using instances of controllers with subscripts, we can describe our system of two oxygen atoms and four hydrogen atoms:

$$(O_1\langle C_{O_1}\rangle \mid O_2\langle C_{O_2}\rangle \mid H_1\langle C_{H_1}\rangle \mid H_2\langle C_{H_2}\rangle \mid H_3\langle C_{H_3}\rangle \mid H_4\langle C_{H_4}\rangle)\backslash\{a,\overline{a},b,\overline{b}\}$$

The processes form the water molecules as follows (leaving out the restrictions for clarity):

$$\begin{aligned}
&(a,a,b).O_1\langle C_{O_1}\rangle \mid (a,a,b).O_2\langle C_{O_2}\rangle \mid (\overline{a},\overline{b}).H_1\langle C_{H_1}\rangle \mid (\overline{a},\overline{b}).H_2\langle C_{H_2}\rangle \mid \\
&(\overline{a},\overline{b}).H_3\langle C_{H_3}\rangle \mid (\overline{a},\overline{b}).H_4\langle C_{H_4}\rangle \xrightarrow{*} (a[1],a[2],b).O_1\langle b.\underline{b}.C'_{O_1} + \underline{a}.C''_O\rangle \mid \\
&(a[3],a[4],b).O_2\langle b.\underline{b}.C'_{O_2} + \underline{a}.C''_O\rangle \mid (\overline{a}[1],\overline{b}).H_1\langle \underline{a}.C_{H_1} + b.\underline{a}.C''_{H_1}\rangle \mid \\
&(\overline{a}[2],\overline{b}).H_2\langle \underline{a}.C_{H_2} + b.\underline{a}.C''_{H_2}\rangle \mid (\overline{a}[3],\overline{b}).H_3\langle \underline{a}.C_{H_3} + b.\underline{a}.C''_{H_3}\rangle \mid \\
&(\overline{a}[4],\overline{b}).H_4\langle \underline{a}.C_{H_4} + b.\underline{a}.C''_{H_4}\rangle
\end{aligned}$$

From here, the controllers allow one oxygen atom and one hydrogen atom to form a bond b:

$$\begin{aligned}&\xrightarrow{*} (a[1], a[2], b[5]).O_1\langle \underline{b}.C'_{O_1}\rangle \mid (a[3], a[4], b).O_2\langle b.\underline{b}.C'_{O_2} + \underline{a}.C''_O\rangle \mid \\ &(\overline{a}[1], \overline{b}).H_1\langle \underline{a}.C_{H_1} + b.\underline{a}.C''_{H_1}\rangle \mid (\overline{a}[2], \overline{b}).H_2\langle \underline{a}.C_{H_2} + b.\underline{a}.C''_{H_2}\rangle \mid \\ &(\overline{a}[3], \overline{b}[5]).H_3\langle \underline{a}.C''_{H_3}\rangle \mid (\overline{a}[4], \overline{b}).H_4\langle \underline{a}.C_{H_4} + b.\underline{a}.C''_{H_4}\rangle\end{aligned}$$

For H_3 this must be followed by breaking a and O_2 can also take that route. So we obtain:

$$\begin{aligned}&\xrightarrow{*} (a[1], a[2], b[5]).O_1\langle \underline{b}.C'_{O_1}\rangle \mid (a, a[4], b).O_2\langle \underline{a}.C_O + a.C'_O\rangle \mid \\ &(\overline{a}[1], \overline{b}).H_1\langle \underline{a}.C_{H_1} + b.\underline{a}.C''_{H_1}\rangle \mid (\overline{a}[2], \overline{b}).H_2\langle \underline{a}.C_{H_2} + b.\underline{a}.C''_{H_2}\rangle \mid \\ &(\overline{a}, \overline{b}[5]).H_3\langle \underline{b}.C_H + a.\underline{b}.C'_H\rangle \mid (\overline{a}[4], \overline{b}).H_4\langle \underline{a}.C_{H_4} + b.\underline{a}.C''_{H_4}\rangle\end{aligned}$$

We have now described the $OH^- - HO_3^+$ situation as desired. One problem is that the system does not require the breaking of a and forming of b on the hydrogen atom to happen together. Even though timing is not modelled here, it is not very realistic to assume that the system can stay in the double bonded hydrogen state for any period of time. In our modelling, we could even have any number of other actions happening in between the forming and breaking of the bonds, which is very unrealistic. Another problem is that a transfer of a hydrogen atom back to O_2 can happen, but only by transferring H_3, since O_1 according to its controller must break the b bond. Breaking keys 1 or 2, which in reality is equally possible, is not allowed in our modelling. Even if we introduce more paths into our controller, there is always the problem that the three hydrogen atoms on H_3O^+ are different, because one of them has a different past, as shown by the keys being on a in H_1 and H_2 and on b in H_3. Also the controllers C_{H_1} and C_{H_2} are in a different state from the controller C_{H_3}.

The modelling done so far gives all steps in the controller. This is not needed. We could have left out the controller on the hydrogen atoms completely, modelling the systems as follows:

$$(O_1\langle C_{O_1}\rangle \mid O_2\langle C_{O_2}\rangle \mid H_1 \mid H_2 \mid H_3 \mid H_4)\backslash\{a, \overline{a}, b, \overline{b}\}$$

The transitions possible are still the same as above, since the hydrogen atoms need to interact with oxygen atoms and the oxygen atoms are fully controlled.

The calculus enables out-of-causal-order reversibility. This is achieved by making the process a very basic entity. In our case it just holds the information which bonds

exist in it and which actions are potentially happening, but no information about the causal order of actions. The process has information about the state, but it has no information about transitions. The controller in contrast holds all the possible orders of execution, it has complete information about the transitions. So we have two entities, one holding state information and one holding transition information. Ideally, we would want one entity comprising state and transitions.

3.4.3 CCS-R

In [27] CCS is combined with an external memory of past actions to achieve reversibility. CCS-R also has multi-actions similar to CCSK with execution control (actions are separated by $\mid$). The syntax is defined as follows:

$$P ::= \mathbf{0} \mid \Sigma a_i.P_i \mid \Pi M_i[P_i]_{u_i} \mid K \mid (\nu x)P$$

Here Σ represents the $+$ and Π the $\mid$ operator. M is a memory defined as:

$$M ::= \langle def : K(u)\rangle.M \mid \langle u, a, P\rangle.M \mid \langle\rangle$$

For memories, $\langle\rangle$ is an empty memory and $\langle u, a, P\rangle.M$ is a memory representing a past transition. In this, P is a dropped process if a choice was exectued, a is the action executed, and u is the process communicated with in case of a communication. Finally, $\langle def : K(u)\rangle.M$ is used in case the executed process contained a process identifier. It enables to re-introduce the process identifier if an action is undone.

The SOS rules for CCS-R are given in Figure 3.7. Here, we have rule com, which means that a process u proceeds by doing an action a and dropping a process P, with P being stored in the memory together with the action a. The back rule then enables the backtracking of this, using the information in the memory to reconstruct the original process, including the dropped process P. $\underline{a}$ represents an irreversible action, which is not added to memory in rule com'. Rule syn allows the synchronisation of two parallel processes. The remaining rules are standard SOS rules, where $\leftrightarrow$ represents the transitions in both directions.

[27] also introduces multi-actions similar to CCSK with execution control. The syntax is:

$$\mu ::= \epsilon \mid (\mu \mid a)$$

where ϵ is an empty multiset and a is a single action. The SOS rules are extended by those in Figure 3.8.

$$\textsf{com}\ \frac{}{M[a.Q+P]_u \xrightarrow{u,r,a} \langle r,a,P\rangle.M[Q]_u} \qquad \textsf{back}\ \frac{}{\langle r,a,P\rangle.M[Q]_u \xleftarrow{u,r,a} M[a.Q+P]_u}$$

$$\textsf{com'}\ \frac{}{M[\underline{a}.Q+P]_u \xrightarrow{u,r,a} M[Q]_u} \qquad \zeta\ \frac{P \overset{u,r,a}{\leftrightarrow} P'}{\langle\rangle[P] \overset{u,r,a}{\leftrightarrow} \langle\rangle[P']}$$

$$\textsf{par}\ \frac{P \overset{u,r,a}{\leftrightarrow} P'}{P \mid Q \overset{u,r,a}{\leftrightarrow} P' \mid Q} \qquad \textsf{res}\ \frac{P \overset{u,r,a}{\leftrightarrow} P' \quad a \neq c,\overline{c}}{(\nu c)P \overset{u,r,a}{\leftrightarrow} (\nu c)P'}$$

$$\textsf{syn}\ \frac{P_1 \overset{u,r,a}{\leftrightarrow} P_1' \quad P_2 \overset{u,r,\overline{a}}{\leftrightarrow} P_2'}{P_1 \mid P_2 \overset{u,r,a}{\leftrightarrow} P_1' \mid P_2'} \qquad \equiv\ \frac{P \equiv P' \quad P' \overset{u,r,a}{\leftrightarrow} Q' \quad Q \equiv Q'}{P \overset{u,r,a}{\leftrightarrow} Q}$$

Figure 3.7: SOS rules for CCS-R.

$$\mu\textsf{-com}\ \frac{}{\langle\gamma;P\rangle.M[(\mu \mid a).Q]_u \xrightarrow{u,r,a} \langle\gamma \mid (r,a);P\rangle.M[Q]_u}$$

$$\mu\textsf{-back}\ \frac{}{\langle\gamma \mid (r,a);P\rangle.M[\mu.Q]_u \xleftarrow{u,r,a} \langle\gamma;P\rangle.M[(\mu \mid a).Q]_u}$$

Figure 3.8: SOS rules for CCS-R with multisets.

For our example the processes can be represented, using CCS-R and multisets, as follows:

$$\begin{aligned} H &\overset{def}{=} (\overline{a}|\overline{b}).\mathbf{0} \\ O &\overset{def}{=} (a|a).b.\mathbf{0} \end{aligned}$$

Notice that we have modelled hydrogen with a set of two actions here. In our molecules the processes get an identifier and an empty memory:

$$(<> (\overline{a}|\overline{b}).\mathbf{0}_1 \mid\ <> (\overline{a}|\overline{b}).\mathbf{0}_2 \mid (a|a).b.\mathbf{0}_5 \mid \overline{a}.\mathbf{0}_3' \mid (\overline{a}|\overline{b}).\mathbf{0}_4 \mid (a|a).b.\mathbf{0}_6)\ \setminus \{a\}$$

We can now form the initial molecules:

$$\begin{aligned} &(< 5,a > (\overline{b}).\mathbf{0}_1 \mid\ < 5,a > (\overline{b}).\mathbf{0}_2 \mid\ < (1,a)|(2,a) > b.\mathbf{0}_5 \mid\ < 6,a > (\overline{b}).\mathbf{0}_3 \\ &\mid\ < 6,a > (\overline{b}).\mathbf{0}_4 \mid\ < (3,a)|(4.a) > b.\mathbf{0}_6)\ \setminus \{a\} \end{aligned}$$

Here, the actions have been executed and "moved" into the memory. For example, the subprocess 1 ($<> (\overline{a}|\overline{b}).\mathbf{0}_1$) has executed $\overline{a}$. So this is now no longer part of the process, but the memory of process 1 ($< 5,a >$ after the execution) contains action a and the information that there was a communication with process 5. We could also have got the situation that one oxygen atom binds to three hydrogen atoms and the other oxygen atom to one only.

We could now make the hydrogen transfer happen by forming a b bond from one of the hydrogen atoms to the other oxygen atom. We can then undo the a bond of that hydrogen atom, since the reverting can be done in any order inside a multi-action group. So these two steps can be modelled as follows:

$$\xrightarrow{1,6,b} (< (5,a)|(6,b) > \mathbf{0}_1 \mid < 5,a > (\overline{b}).\mathbf{0}_2 \mid < (1,a)|(2,a) > b.\mathbf{0}_5 \mid < 6,a > (\overline{b}).\mathbf{0}_3 \mid < 6,a > (\overline{b}).\mathbf{0}_4 \mid < (3,a)|(4,a).(1,b) > \mathbf{0}_6) \setminus \{a\}$$
$$\xleftarrow{1,5,a} (< (6,b) > (a).\mathbf{0}_1 \mid < 5,a > (\overline{b}).\mathbf{0}_2 \mid < (2,a) > (a).b.\mathbf{0}_5 \mid < 6,a > (\overline{b}).\mathbf{0}_3 \mid < 6,a > (\overline{b}).\mathbf{0}_4 \mid < (3,a)|(4,a).(1,b) > \mathbf{0}_6) \setminus \{a\}$$

This is our system in the desired state, and with a proton transfer as a step to get there. Similarly to the situation when using CCSK there are still some problems:

1. The hydrogen atom can stay in the double bonded state and there is nothing to show that this is a transition state only.
2. The hydrogen atoms in the H_3O^+ molecule are different due to having a different history.
3. The reversal of the hydrogen transfer can only work with 1, since 5 needs to do an a action as next step.
4. A hydrogen atom could be bonded to one oxygen atom with two bonds in parallel.

We could also have modelled just one type of action similarly to Section 3.4.1. The processes would be:

$$H \stackrel{def}{=} (\overline{a}).\mathbf{0}$$
$$O \stackrel{def}{=} (a|a|a).\mathbf{0}$$

That would have lead to the same problem we saw with CCSK, namely that only the spontaneous dissocation enables the proton transfer.

3.4.4 κ-calculus

The κ-calculus was introduced in [28] and is based on graph rewriting. It is aimed at modelling proteins. Proteins have sites, which can bond to other sites. The formal syntax of a κ-process is as follows:

$$S := \mathbf{0} \mid A(\rho) \mid S, S \mid (x)(S)$$

where **0** is the empty process, S, S is a group of processes, (x) is the new name constructor, and A is a protein with an interface ρ. The interface consists of a number of sites which can be free or bound and hidden or visible. Edges are formed between sites. Rules are used to determine which edges can be formed or broken. Sites can be *hidden* or *visible*. Only visible sites can form part of a reaction. Sites can also have states. This can for example be used to model phosphorylation.

A simple example of a system (not using hidden sites or states) could consist of a kinase (K) and a target (T). The agents are:

$$K(a)$$
$$T(x, y)$$

A rule allowing the kinase to bind to the target is:

$$K(a), T(x) \leftrightarrow K(a!1), T(x!1)$$

The rule requires a to be ready on K and x on T. Notice it does not say anything about y, since sites which are not relevant for the rule can be left out. On the right both a and x are bound (indicated by !) and the index 1 shows the bond. If we apply the rule we get this transition:

$K(a), T(x, y) \rightarrow K(a!1), T(x!1, y)$

In order to model the autoprotolysis of water in the κ-calculus we start with atoms as agents. For oxygen and hydrogen the agent signatures are:

$$O(a, a, c)$$

$$H(d)$$

Oxygen has three interaction sites named a, a and c and hydrogen has exactly one, named d. This corresponds to the number of bonds the atoms can hold. There is nothing said about which actions can happen in which order. In our modelling we do not use the possibility to have different states for the interaction sites, since we can model everything using the interaction of the atoms. The following rule captures the bonding of a hydrogen atom to an oxygen atom:

$$O(a), H(d) \rightarrow O(a!1), H(d!1) \tag{3.1}$$

Rewrite (3.1) describes that an oxygen atom and a hydrogen atom can combine on any of the a sites of the oxygen atom, given an a site and d site of a hydrogen

atom are available, i.e. not yet bonded. Again we do not fully specify the agents, for oxygen we only say an a must be available, the rest could be in any state. We get the following system of two oxygen atoms and four hydrogen atoms by forming groups of atoms using rule (3.1):

$$O(a!1, a!2, c), O(a!3, a!4, c), H(d!1), H(d!2), H(d!3), H(d!4)$$

A graphical illustration of the situation before applying the rule is shown in Figure 3.9 a) and in in Figure 3.9 b) after applying the rule.

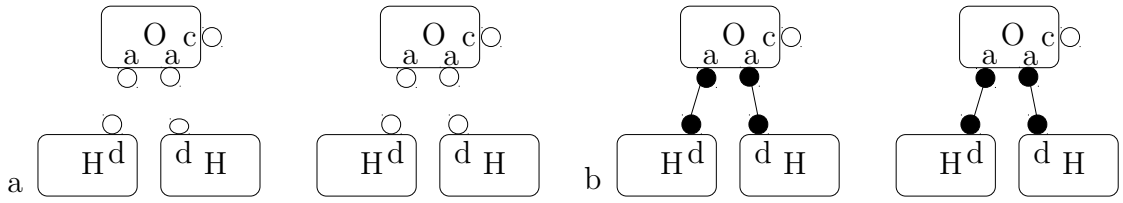

Figure 3.9: A possible modelling of the autoprotolysis of water using the κ calculus. Squares represent atoms, filled circles indicate bound sites, empty circles indicate free sites. The style of illustration follows [28].

As explained, the keys indicate bonds which have been formed. The bonding is as expected in two molecules of water and has been facilitated by the rule given. By adding an appropriate rule we could also make the bonding of hydrogen atoms to oxygen atoms reversible. We now need a rule for the hydrogen transfer. This can be expressed as follows:

$$O(c), H(d!1), O(a!1) \rightarrow O(c!2), H(d!2), O(a) \tag{3.2}$$

The meaning of this is that if we have an oxygen atom which has a free c site and a hydrogen bond to another oxygen atom, then the c site can bind to the hydrogen atom with the existing bond being broken. Our process can evolve as follows using rule (3.2):

$$O(a!1, a!2, c!5), O(a, a!4, c), H(d!1), H(d!2), H(d!5), H(d!4)$$

From that point we can now reverse, again using rule (3.2), and we get:

$$O(a!1, a!2, c), O(a, a!4, c!3), H(d!1), H(d!2), H(d!3), H(d!4)$$

Both transitions are using the same rule, which breaks the bond between a hydrogen atom and an oxygen atom and forms a new bond from the same hydrogen atom to another oxygen atom.

A problem with this is that one of bonds is now to c and not to a, as it was initially. We could solve this by either having a rewrite rule similar to *promotion* in our calculus or by having an extra rule where the a does the transfer. Overall we can nicely simulate our system, but we have the separation of processes and rules. Whilst this is a possible modelling, there is a separate body of knowledge about transitions. Ideally we want these to be an emergent property of the processes.

3.4.5 P Systems

P systems were originally designed in order to model biological processes taking place in cells. Because of this, membranes (which separate the inside environment of the cell from the outside environment and are selectively permeable) play an important rôle. In our chemical examples we do not have membranes, but we could still use the idea to model chemical processes. The syntax of a P system is defined as follows (from [87]):

$$\Pi = (V, \mu, w_1, ..., w_m, (R_1, \rho_1), ..., (R_m, \rho_m), i_0)$$

where

1. V is an alphabet, its members are called objects. They represent the biochemical entities we want to model.

2. μ is a membrane structure of degree m. Membranes can be nested, but there must be an outer membrane containing the other membranes.

3. $w_1, ..., w_m$ are words from the alphabet V. They represent the initial content of the membranes 1 to m. ρ_i is a partial order relation over R_i, specifying priority of rules in R_i.

4. $R_1, ..., R_m$ are the rules for each membrane. Each rule is of the form $u \to v$ where u, v are words from V

5. i_0 identifies the output membrane. P systems are intended to have a result in a certain membrane once no rule can be executed any more.

If we want to model the autoprotolysis of water, we could do this by creating a system with three "compartments", where two compartments are inside the outer one and contain an oxygen atom and two hydrogen atoms each, representing atoms.

We would then give a rule which enables an "OHH" to move out of its compartment. The outer compartment would then have a rule which enables two such water molecules to interact. The membranes have the purpose to distinguish unconnected atoms (represented in the inner compartments) from actual water molecules, which are represented in the outer compartment. Since in a P system we do not hold connectivity information we need to represent that information differently.

Our initial system is modelled as follows (the five elements mentioned above are written on separate lines):

$$
\begin{aligned}
&\Pi_1 = (\\
&\quad HOXN,\\
&\quad [_1[_2]_2[_3]_3]_1,\\
&\quad \lambda, HHO, HHO,\\
&\quad HHOHHO \rightarrow XN, HHO \rightarrow (HHO, out), HHO \rightarrow (HHO, out),\\
&\quad 1\\
&)
\end{aligned}
$$

The system is represented in Figure 3.10.a): We have two inner membranes in an outer membrane (described by $[_1[_2]_2[_3]_3]_1$), where membrane 1 (the outer membrane) is empty (represented by λ) and the inner membranes both contain an entity HHO. This is representing unconnected hydrogen and oxygen atoms in our case. The rules say that the HHO objects from the inner membrane can move to the outer membrane. This represents the formation of water molecules. In the outer membrane two HHO objects can react to hydroNium (H_3O^+) and hydroXide (OH^-), represented by the letters X and N respectively, the results of the autoprotolysis. We also have defined membrane 1 (that is the outer membrane) as the output membrane. Whilst we have chosen the names of our objects to resemble our molecules this would not be necessary. The rules ensure that we get the desired result, which we interpret as molecules again. The *out* in the second and third rule enables the reaction to start, and it takes place in the first membrane (the output membrane).

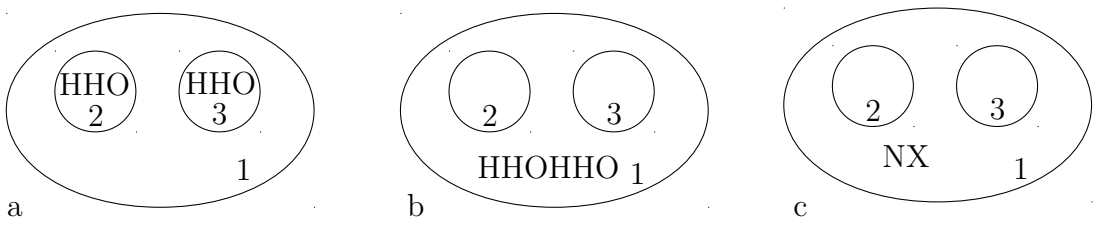

Figure 3.10: A possible modelling of the autoprotolysis of water using P Systems.

To see how this happens exactly we follow the steps. In the initial state, the only rules which can apply are those for the inner membranes 2 and 3. So we get:

$$\Pi_1 = (HOXN, [_1[_2]_2[_3]_3]_1, HHOHHO, \lambda, \lambda, HHOHHO \to XN,$$
$$HHO \to (HHO, out), HHO \to (HHO, out), 1$$

We represent this in Figure 3.10.b.

Then, in the outer membrane the rule applies and we get:

$$\Pi_1 = (HOXN, [_1[_2]_2[_3]_3]_1, XN, \lambda, \lambda, HHOHHO \to XN,$$
$$HHO \to (HHO, out), HHO \to (HHO, out), 1$$

The output membrane now contains the result, a hydroxide and a hydronium ion, as shown in Figure 3.10.c). We could also have added rules to reverse the process (e.g. $XN \to HHOHHO$ in the outer membrane). Reverse rules must in any case be given explicitly.

Whilst we have managed to model the reaction, the modelling does not seem very natural. This is mainly because we cannot handle a water molecule (or water ion) as a composition of atoms and there is no connectivity information. This means that in our rules we have to use the alphabet V to describe the compounds. If a new compound is formed we need to give a new name to it, forgetting about its components. This means that the rules in P systems are overall too simple for our purpose. The atom based modelling described in Section 3.3 is not possible in P systems.

3.5 Conclusion

We have seen that there is an extensive work on the application of process calculi to biology. Various calculi were developed emphasizing different aspects of biological systems. Whilst reversibility is contained in most of them, an explicit notion of reversibility was developed only recently. Similarly, causality is increasingly dealt with, where out-of-causal order reversibility is gaining attention, whereas traditionally backtracking was the standard form of reversibility in many calculi. We have also seen that process calculi have mainly been applied to higher-level biological systems and not to chemical reactions. At the same time the mechanisms of chem-

ical reactions have been studied intensively. Therefore we decided to look at such reactions for exploring the concepts of reversibility and causality in process calculi.

A short examination of existing calculi showed that we can model a simple chemical reaction using them. We have also seen weaknesses of the calculi. In particular CCSK and CCS-R restrict the possible reactions more than is the case in reality because they rely on backtracking. CCSK with execution control allows a better modelling, but uses external controls which include all possible traces. The κ-calculus allows a good modelling, but also has external rules. Finally we showed that P Systems are not very suitable for our purpose.

Chapter 4

A Calculus of Covalent Bonding

We now give a formal description of the calculus introduced in Chapters 1 and 2, which we call the Calculus of Covalent Bonding, in short CCB. We specify a labelled transition system with structural operational semantics for the transitions. We also give properties for the calculus in order to get a better understanding of the possibilities and limitations of the calculus. In this chapter we do not work with the subscripts suggested in Section 2 to keep the calculus more pure.

4.1 Definition of the calculus

In this section we define our new calculus CCB. First, we introduce some preliminary notions and notations.

Let $\mathcal{A}$ be the set of (forward) action labels, ranged over by a, b, c, d, e, f. We partition $\mathcal{A}$ into the set of *strong actions*, written as $\mathcal{SA}$, and the set of *weak actions*, written as $\mathcal{WA}$. Reverse (or past) action labels are members of $\underline{\mathcal{A}}$, with typical members $\underline{a}, \underline{b}, \underline{c}, \underline{d}, \underline{e}, \underline{f}$, and represent undoing of actions. The set $\mathcal{P}(\mathcal{A} \cup \underline{\mathcal{A}})$ is ranged over by L. Remember from the example in Chapter 2 that the weak actions, in conjunction with the prefixing operator $(s; b).P$, allowed the breaking of existing bonds when a new bond is formed. They also allow moving of keys by rewriting rules to restore the basic state of atoms.

Let $\mathcal{K}$ be an infinite set of *communication keys* (or *keys* for short) [94, 90], ranged over by k, l, m, n. The Cartesian product $\mathcal{A} \times \mathcal{K}$, denoted by $\mathcal{AK}$, represents past actions, which are written as $a[k]$ for $a \in \mathcal{A}$ and $k \in \mathcal{K}$. Correspondingly, we have the set $\underline{\mathcal{A}}\mathcal{K}$ that represents undoing of past actions. We use α, β to identify actions

which are either from $\mathcal{A}$ or $\mathcal{AK}$. It would be useful to consider sequences of actions or past actions, namely the elements of $(\mathcal{A} \cup \mathcal{AK})^*$, which are ranged over by s, s' and sequences of purely past actions, namely the elements of $\mathcal{AK}^*$, which are ranged over by t, t'. The empty sequence is denoted by ϵ. We use the notation "α,s" and "s,s'" to denote a concatenation of elements, which can be strings or single actions.

We shall also use two sets of auxiliary action labels, namely the set $(\mathcal{A}) = \{(a) \mid a \in \mathcal{A}\}$, and its product with the set of keys, namely $(\mathcal{A})\mathcal{K}$. These labels will be used in the auxiliary rules when defining the semantics of CCB. They denote the execution of a weak action, which makes it possible in the SOS rules to force breaking of a bond for those actions only.

We now define the Calculus of Covalent Bonding. The syntax of CCB is given below where P is a process term:

$$P ::= S | S \stackrel{def}{=} P \;\Big|\; (s;b).P \;\Big|\; P|Q \;\Big|\; P \backslash L$$

The set of process identifiers (constants) $\mathcal{PI}$ contains typical elements S and T. Each process identifier S has a defining equation $S \stackrel{def}{=} P$ where P contains only forward actions (and no past actions). There is also a special identifier $\mathbf{0}$, denoting the deadlocked process, which has no defining equation. For restrictions $L \subseteq \mathcal{A}$ holds.

The calculus does not contain the nondeterministic choice operator $+$, which is a feature of many similar calculi, including CCS. The reason for this decision is that in our chemical modelling we decided to model any changes to the atoms by means of the bonds respectively actions. We have seen an example of this in Chapter 2, where an oxygen was modelled as $O \stackrel{def}{=} (o, o, n).O'$. It can have either two or three bonds, and since one of them is a weak action, the tendency is to go back to have two bonds. This way we have modelled the behaviour of the atom without nondeterministic choice. This does not mean there is no nondeterminism in the calculus, though. We actually introduce a new type of nondeterminism, as we will see. In Chapter 7 we use several multisets in one process and, together with the use of the $(s;b)$ operator, can model two different behaviours of the process.

We have a general prefixing operator $(s;b).P$, where s is a non-empty sequence of actions or past actions. This operator extends the prefixing operator in [93]. The action b is a weak action and it can be omitted, in which case the prefixing is written as $(s).P$ and is called the *simple prefix*. The simple prefix (which is still a sequence)

is the prefixing operator in [93]. One of the actions in s in $(s).P$ may be a weak action from $\mathcal{WA}$. A weak action in s is only allowed for the simple prefix, in the $(s;b)$ operator b is the only allowed weak action. If s is a sequence that contains a single action, then the action is a strong action and the operator is the prefixing operator of CCS [78]. We omit trailing $\mathbf{0}$s so, for example, $(s).\mathbf{0}$ is written as (s). The new feature of the operator $(s;b).P$ is the execution of the weak action b, which can happen only after all the actions in s have taken place. Performing b then forces undoing one of the past actions in s (by the concert rule in Figure 4.5). If a $(s;b)$ operator is followed by another sequence of actions there is a non-deterministic choice of either doing b or progressing to the next sequence of actions.

$P \mid Q$ represents two systems P and Q which can perform actions or reverse actions on their own, or which can interact with each other according to a communication function γ. As in the calculus ACP [36], the communication function is a partial function $\gamma : \mathcal{A} \times \mathcal{A} \to \mathcal{A}$ which is commutative and associative. The function γ is used in the operational semantics to define when two processes can interact. Processes P and Q in $P \mid Q$ can also perform a pair of concerted actions, which is the new feature of our calculus. We also have the ACP-like restriction operator $\backslash L$, where L is a set of labels. It prevents actions from taking place and, due to the synchronisation algebra used, it also blocks communication. If $\gamma(a,b) = c$ then $a.P$ and $b.Q$ cannot communicate in $(a.P \mid b.Q) \setminus c$. Note that we do not use the usual relabelling operator $[f]$, where $f : \mathcal{A} \to \mathcal{A}$, in CCB, which could be easily added.

The set of *process terms* is ranged over by P, Q and R and is denoted by Proc. In the setting of CCB these terms are simply called *processes*. A context $C[\,]$ is a process term containing a *hole*, represented by $[\,]$. Formally, contexts are defined by the following syntax: $C ::= [\,] \mid (s;b).C \mid P|C \mid C|P \mid C\backslash L$. The term $C[Q]$ denotes the result of filling the hole in the context $C[\,]$ with the process Q. We say that R is a *subprocess* of P if P is $C[R]$ for some context $C[\,]$.

We define the semantics of our calculus by a labelled transition system, LTS for short. In Section 3.1 an LTS was defined as a structure $(St, AL, \rightarrow : \subseteq St \times AL \times St)$ with St representing the set of states, AL the set of action labels and $\rightarrow : \subseteq St \times AL \times St$ the labelled transition relation. The set of states St is the set Proc. In practice, all our results and examples hold for *consistent* processes, namely processes reachable from standard processes (see Definition 4.4). The action labels are the forward actions $\mathcal{AK}$, the reverse actions $\underline{\mathcal{A}}\mathcal{K}$ and the *pairs of concerted actions* $\mathcal{AK} \times \underline{\mathcal{A}}\mathcal{K}$. The labelled transition relation is defined by SOS rules (Figures 4.2–4.6) and rewrite rules (Figure 4.7), where the rules in Figures 4.2–4.4 are influenced by [90]. Note that sequences s and t are members of $(\mathcal{A} \cup \mathcal{AK})^*$ and $\mathcal{AK}^*$ respectively in Figures 4.2–4.5.

$$\frac{}{\mathsf{std}(\mathbf{0})} \qquad \frac{}{\mathsf{fsh}[m](\mathbf{0})}$$

$$\frac{\mathsf{std}(P)}{\mathsf{std}(S)}\ S \stackrel{def}{=} P \qquad \frac{\mathsf{fsh}[m](P)}{\mathsf{fsh}[m](S)}\ S \stackrel{def}{=} P$$

$$\frac{\mathsf{k}(s) = \emptyset \quad \mathsf{std}(P)}{\mathsf{std}((s;b).P)} \qquad \frac{m \notin \mathsf{k}(s) \quad \mathsf{fsh}[m](P)}{\mathsf{fsh}[m]((s;b).P)}$$

$$\frac{\mathsf{std}(P) \quad \mathsf{std}(Q)}{\mathsf{std}(P \mid Q)} \qquad \frac{m \notin \mathsf{k}(s) \quad m \neq n \quad \mathsf{fsh}[m](P)}{\mathsf{fsh}[m]((s;b[n]).P)}$$

$$\frac{\mathsf{std}(P)}{\mathsf{std}(P \setminus L)} \qquad \frac{\mathsf{fsh}[m](P) \quad \mathsf{fsh}[m](Q)}{\mathsf{fsh}[m](P \mid Q)} \qquad \frac{\mathsf{fsh}[m](P)}{\mathsf{fsh}[m](P \setminus L)}$$

Figure 4.1: Predicates std and fsh.

We now introduce and explain the SOS rules before returning to the rewrite rules. Let r be an SOS rule for an operator f of CCB as in Figures 4.2–4.5. Transitions above the horizontal bar in r are called *premises*. The set of premises is written as $pre(r)$. The transition below the bar in r is the *conclusion* and is written as $con(r)$. We use two predicates $\mathsf{std}(P) : \mathsf{Proc}$ and $\mathsf{fsh}[m](P) : \mathcal{K} \times \mathsf{Proc}$ in our SOS rules. They are defined in Figure 4.1, and they use two auxiliary functions $\mathsf{k}(s) : (\mathcal{A} \cup \mathcal{A}\mathcal{K})^* \to \mathcal{P}(\mathcal{K})$ and $\mathsf{keys}(P) : \mathsf{Proc} \to \mathcal{P}(\mathcal{K})$, which are defined in Figure 4.3. Informally $\mathsf{keys}(P)$ associates with each term P the set of its keys. A process P is standard, written $\mathsf{std}(P)$, if it contains no past actions (hence no keys). A key n is fresh in Q, written $\mathsf{fsh}[n](Q)$, if Q contains no past action with the key n. We extend the notion of fresh keys to the sequences of actions and past actions s and t via the function $\mathsf{k}()$.

Figure 4.1 defines the predicates by induction over the process terms. We could also have said that $\mathsf{std}(P)$ is true if $\mathsf{keys}(P) = \emptyset$ and that $\mathsf{fsh}[m](P)$ is true if $i \notin \mathsf{keys}(P)$.

$$\mathsf{act1}\ \frac{\mathsf{std}(P) \quad \mathsf{fsh}[k](s, s')}{(s, a, s'; b).P \xrightarrow{a[k]} (s, a[k], s'; b).P} \qquad \mathsf{act2}\ \frac{P \xrightarrow{a[k]} P' \quad \mathsf{fsh}[k](t)}{(t; b).P \xrightarrow{a[k]} (t; b).P'}$$

$$\mathsf{par}\ \frac{P \xrightarrow{a[k]} P' \quad \mathsf{fsh}[k](Q)}{P \mid Q \xrightarrow{a[k]} P' \mid Q} \qquad \mathsf{com}\ \frac{P \xrightarrow{a[k]} P' \quad Q \xrightarrow{d[k]} Q'}{P \mid Q \xrightarrow{c[k]} P' \mid Q'}\ (*)$$

$$\mathsf{res}\ \frac{P \xrightarrow{a[k]} P'}{P \backslash L \xrightarrow{a[k]} P' \backslash L}\ a \notin L \qquad \mathsf{con}\ \frac{P \xrightarrow{a[k]} P'}{S \xrightarrow{a[k]} P'}\ S \stackrel{def}{=} P$$

Figure 4.2: Forward SOS rules for CCB. The condition (*) is $\gamma(a, d) = c$, and $b \in \mathcal{W}\mathcal{A}$. Remember that s is a mixed sequence of actions and past actions and t is a sequence of purely past actions

$$\mathsf{k} : (\mathcal{A} \cup \mathcal{A}\mathcal{K})^* \to \mathcal{P}(\mathcal{K})$$
$$\mathsf{k}(\epsilon) = \emptyset$$
$$\mathsf{k}(\alpha : s) = \begin{cases} \{l\} \cup \mathsf{k}(s) & \text{if } \alpha = a[l], a \in \mathcal{A}, l \in \mathcal{K} \\ \mathsf{k}(s) & \text{if } \alpha \in \mathcal{A} \end{cases}$$

$$\mathsf{keys} : \mathsf{Proc} \to \mathcal{P}(\mathcal{K})$$
$$\mathsf{keys}(\mathbf{0}) = \emptyset$$
$$\mathsf{keys}(S) = \mathsf{keys}(X) \quad \text{if } S \stackrel{def}{=} X$$
$$\mathsf{keys}((s; b).X) = \mathsf{k}(s) \cup \mathsf{k}(b) \cup \mathsf{keys}(X)$$
$$\mathsf{keys}(X \mid Y) = \mathsf{keys}(X) \cup \mathsf{keys}(Y)$$
$$\mathsf{keys}(X \setminus L) = \mathsf{keys}(X)$$

Figure 4.3: Functions k and keys.

$$\textsf{rev act1}\ \frac{\textsf{std}(P)}{(s, a[k], s'; b).P \xrightarrow{\underline{a}[k]} (s, a, s'; b).P} \qquad \textsf{rev act2}\ \frac{P \xrightarrow{\underline{a}[k]} P'}{(t; b).P \xrightarrow{\underline{a}[k]} (t; b).P'}$$

$$\textsf{rev par}\ \frac{P \xrightarrow{\underline{a}[k]} P' \quad \textsf{fsh}[k](Q)}{P \mid Q \xrightarrow{\underline{a}[k]} P' \mid Q} \qquad \textsf{rev com}\ \frac{P \xrightarrow{\underline{a}[k]} P' \quad Q \xrightarrow{\underline{d}[k]} Q'}{P \mid Q \xrightarrow{\underline{c}[k]} P' \mid Q'}\ (*)$$

$$\textsf{rev res}\ \frac{P \xrightarrow{\underline{a}[k]} P'}{P\backslash L \xrightarrow{\underline{a}[k]} P'\backslash L}\ \underline{a} \notin L \qquad \textsf{rev con}\ \frac{P \xrightarrow{\underline{a}[k]} P'}{P \xrightarrow{\underline{a}[k]} S}\ S \stackrel{def}{=} P'$$

Figure 4.4: Reverse SOS rules for CCB. The condition (*) is $\gamma(a, d) = c$, and $b \in \mathcal{WA}$.

$$\textsf{aux1}\ \frac{\textsf{std}(P) \quad \textsf{fsh}[k](t)}{(t; b).P \xrightarrow{(b)[k]} (t; b[k]).P} \qquad \textsf{aux2}\ \frac{P \xrightarrow{(b)[k]} P' \quad \textsf{fsh}[k](t)}{(t; b').P \xrightarrow{(b)[k]} (t; b').P'}$$

$$\textsf{concert}\ \frac{P \xrightarrow{(b)[k]} P' \quad P' \xrightarrow{\underline{a}[l]} P'' \qquad Q \xrightarrow{\alpha[k]} Q' \quad Q' \xrightarrow{\underline{d}[l]} Q''}{P \mid Q \xrightarrow{\{e[k], \underline{f}[l]\}} P'' \mid Q''}(*)$$

$$\textsf{concert act}\ \frac{P \xrightarrow{\{a[k], \underline{h}[l]\}} P' \quad \textsf{fsh}[k](t)}{(t; b).P \xrightarrow{\{a[k], \underline{h}[l]\}} (t; b).P'}$$

$$\textsf{concert par}\ \frac{P \xrightarrow{\{a[k], \underline{h}[l]\}} P' \quad \textsf{fsh}[k](Q) \quad \textsf{fsh}[l](Q)}{P \mid Q \xrightarrow{\{a[k], \underline{h}[l]\}} P' \mid Q}$$

$$\textsf{concert res}\ \frac{P \xrightarrow{\{a[k], \underline{h}[l]\}} P'}{P\backslash L \xrightarrow{\{a[k], \underline{h}[l]\}} P'\backslash L}(**)$$

Figure 4.5: SOS rules for concerted actions in CCB. The condition (*) is 1. α is c or (c) and $\gamma(b, c) = e$ for some $c \in \mathcal{A}$, and 2. $\gamma(a, d) = f$. The condition (**) is $a, \underline{h} \notin L \cup (L)$. Recall that $t \in \mathcal{AK}^*$, and $b \in \mathcal{WA}$.

$$\frac{P \Rightarrow Q \quad Q \xrightarrow{\mu} Q' \quad Q' \Rightarrow P'}{P \xrightarrow{\mu} P'}$$

Figure 4.6: Structural congruence rule sc when $\mu \in \mathcal{AK} \cup (\mathcal{AK} \times \underline{\mathcal{A}}\mathcal{K})$, and rev sc when $\mu \in \underline{\mathcal{A}}\mathcal{K}$.

red1 :	$P \mid Q \Rightarrow Q \mid P$	
red2 :	$P \mid (Q \mid R) \Rightarrow (P \mid Q) \mid R$	
red3 :	$(P \mid Q) \mid R \Rightarrow P \mid (Q \mid R)$	
red4 :	$P \mid \mathbf{0} \Rightarrow P$	
red5 :	$(P \mid Q)\backslash L \Rightarrow P\backslash L \mid Q$	if $(\gamma(\mathsf{fn}(P) \times \mathsf{fn}(Q)) \cup \mathsf{fn}(Q)) \cap L = \emptyset$
red6 :	$P\backslash L \mid Q \Rightarrow (P \mid Q)\backslash L$	if $(\gamma(\mathsf{fn}(P) \times \mathsf{fn}(Q)) \cup \mathsf{fn}(Q)) \cap L = \emptyset$
prom :	$(s, a, s'; b[k]).P \Rightarrow (s, a[k], s'; b).P$	if $a \in \mathcal{SA}, b \in \mathcal{WA}$
move-r :	$(s, a, s', b[k], s'').P \Rightarrow (s, a[k], s', b, s'').P$	if $a \in \mathcal{SA}, b \in \mathcal{WA}$
move-l :	$(s, b[k], s', a, s'').P \Rightarrow (s, b, s', a[k], s'').P$	if $a \in \mathcal{SA}, b \in \mathcal{WA}$

Figure 4.7: Reduction rules for CCB. Sequences s, s', s'' are members of $(\mathcal{A} \cup \mathcal{AK})^*$.

Example 4.1. We illustrate how processes compute forwards using the new prefixing operator. Consider a standard process $(a;b).(c) \mid (a,d,c)$ and the communication function γ given by $\gamma(a,a) = a$ and $\gamma(c,c) = c$. We have

$$(a;b).(c) \mid (a,d,c) \xrightarrow{a[1]} (a[1];b).(c) \mid (a[1],d,c)$$

by the SOS rules act1 and com from Figure 4.2. This is because (c) is standard and the key 1 is fresh in ε. The next step of computation involves a communication of the actions c, which we obtain by rules act2 and com:

$$(a[1];b).(c) \mid (a[1],d,c) \xrightarrow{c[2]} (a[1];b).(c[2]) \mid (a[1],d,c[2])$$

We note that the key 2 is fresh in $a[1]$. Finally, the action d takes place by act1 and, informally, the symmetric version of par.

$$(a[1];b).(c[2]) \mid (a[1],d,c[2]) \xrightarrow{d[3]} (a[1];b).(c[2]) \mid (a[1],d[3],c[2])$$

Formally, we use par, the structural congruence rule sc in Figure 4.6 and the reduction rule red1 in Figure 4.7.

The next example illustrates how some of the reverse SOS rules work.

Example 4.2. Consider $(a[1],b).(c).S$ where $S \stackrel{def}{=} (a,b).(c).S$. We have

$$(a[1],b).(c).S \xrightarrow{\underline{a}[1]} (a,b).(c).S$$

by rev act1 since $(c).S$ is standard. Since $(a,b).(c).S$ is the definition of S we obtain by rule rev con $(a[1],b).(c).S \xrightarrow{\underline{a}[1]} S$.

Figure 4.5 contains the SOS rules that define the new concerted actions transitions. The main rule is the rule concert that defines when a pair of concerted actions takes place. We also have two auxiliary rules aux1 and aux2 which define only an auxiliary transition relation needed in the concert rule. The auxiliary rules are only applicable if a weak action is beyond the semicolon, whereas a weak action in the simple prefix is dealt with by standard rules and can therefore not become part of a concerted action. Note that the concert rule uses *lookahead* [115]. Also note that transitions in aux1 and aux2 use the auxiliary labels $(b)[k]$ for all $b \in \mathcal{WA}$ and $k \in \mathcal{K}$. The rule concert par requires that k is fresh in Q, correspondingly as in par. Moreover, we need to ensure that when we reverse h with the key l in P we do not leave out any actions with the key l in Q which make up a multiaction communication with the key l. Hence, we also include the premise $\mathsf{fsh}[l](Q)$ in concert par. The rule concert

act requires, correspondingly as act, that k is fresh in t. Our operational semantics guarantees that if a standard process evolves to $(t; b).P$, for some P, and P reverses an action with the key l, then l is fresh in t. Hence, we do not include $\mathsf{fsh}[l](t)$ in the premises of concert act. Next, we illustrate how concerted actions transitions work.

Example 4.3. Consider the process $(a; b) \mid a \mid b$ with $\gamma(a, a) = c$ and $\gamma(b, b) = d$. After the initial synchronisation of actions a, which produces the transition $c[1]$, we have a transition with a pair of concerted actions by rule concert in Figure 4.5

$$(a[1]; b) \mid a[1] \mid b \xrightarrow{\{d[2],\underline{c}[1]\}} (a; b[2]) \mid a \mid b[2]$$

since $(a[1]; b) \xrightarrow{(b)[2]} (a[1]; b[2])$ by aux1, $(a[1]; b[2]) \xrightarrow{\underline{a}[1]} (a; b[2])$ by rev act1, and since $a[1] \mid b \xrightarrow{b[2]} a[1] \mid b[2] \xrightarrow{\underline{a}[1]} a \mid b[2]$ by par and rev par. The inference tree for this example is shown in Figure 4.8.

Example 4.4. Consider $(a[1]; b) \mid (a[1]; b) \mid e$ with $\gamma(a, a) = c$ and $\gamma(b, b) = d$. We clearly have the following pair of concerted actions

$$(a[1]; b) \mid (a[1]; b) \mid e \xrightarrow{\{d[2],\underline{c}[1]\}} (a; b[2]) \mid (a; b[2]) \mid e.$$

Note that action d is possible since both processes break the same bond. If the two a actions would have different keys it would not be possible to perform d since our rules do not allow to break two bonds in a concerted action. The inference tree for this example is shown in Figure 4.9.

There are processes with weak actions that can potentially communicate but there are no concerted actions transitions due to our SOS rules:

Example 4.5. Consider $(a[1]; b) \mid (e[2]; b) \mid (a[1], e[2])$ with $\gamma(a, a) = c$ and $\gamma(b, b) = d$. The process cannot perform any concerted actions: Although $(a[1]; b) \xrightarrow{(b)[l]} \xrightarrow{\underline{a}[1]} (a; b[l])$, for any l different from 1 and 2, but $(e[2]; b) \mid (a[1], e[2])$ cannot perform the auxiliary $(b[l])$ transition since there are no SOS rules for parallel composition and auxiliary actions (b). This forces us to treat $(a[1]; b)$ and $(e[2]; b)$ as P and Q in the concert rule, respectively, and we notice that we cannot undo a communication on a or e.

Overall, the transitions in Figures 4.2–4.5 are labelled with $a[k] \in \mathcal{AK}$, or with $\underline{c}[l] \in \underline{\mathcal{A}}\mathcal{K}$, or with concerted actions $(a[k], \underline{c}[l])$.

We also have the usual structural congruence rules sc and rev sc in Figure 4.6, which potentially combine reductions (defined below) with transitions.

$$\dfrac{\dfrac{\mathsf{std}(0) \qquad \mathsf{fsh}[2](a[1])}{(a[1];b) \xrightarrow{(b)[2]} (a[1];b[2])}\,\text{aux1} \qquad \dfrac{\dfrac{\mathsf{std}(0) \qquad \mathsf{fsh}[1](\emptyset)}{(a[1];b[2]) \xrightarrow{\underline{a}[1]} (a;b[2])}\,\text{rev act1} \qquad \dfrac{\dfrac{\mathsf{std}(0) \qquad \mathsf{fsh}[2](\emptyset)}{b \xrightarrow{b[2]} b[2]}\,\text{act1}}{a[1] \mid b \xrightarrow{b[2]} a[1] \mid b[2]}\,\text{par}}{a[1] \mid b[2] \xrightarrow{\underline{a}[1]} a \mid b[2]}\,\text{par}}{(a[1];b) \mid a[1] \mid b \xrightarrow{\{d[2],\underline{c}[1]\}} (a;b[2]) \mid a \mid b[2]}\,\text{concert}$$

Figure 4.8: Inference tree for Example 4.3.

$$\text{concert}\dfrac{\text{aux1}\dfrac{\mathsf{std}(0) \quad \mathsf{fsh}[2](a[1])}{(a[1];b) \xrightarrow{(b)[2]} (a[1];b[2])} \quad \text{rev act1}\dfrac{\mathsf{std}(0) \quad \mathsf{fsh}[1](\emptyset)}{(a[1];b[2]) \xrightarrow{\underline{a}[1]} (a;b[2])} \quad \text{act1}\dfrac{\mathsf{std}(0) \quad \mathsf{fsh}[2](a[1])}{(a[1];b) \xrightarrow{(b)[2]} (a[1];b[2])} \quad \text{rev act1}\dfrac{\mathsf{std}(0) \quad \mathsf{fsh}[1](\emptyset)}{a[1];b[2]) \xrightarrow{\underline{a}[1]} (a;b[2])}}{\text{concert par}\dfrac{(a[1];b) \mid (a[1];b) \xrightarrow{\{d[2],\underline{c}[1]\}} (a;b[2]) \mid (a;b[2])}{(a[1];b) \mid (a[1];b) \mid e \xrightarrow{\{d[2],\underline{c}[1]\}} (a;b[2]) \mid (a;b[2]) \mid e}}$$

Figure 4.9: Inference tree for Example 4.4.

Next, we introduce our reduction relation which is given by the reduction (rewrite) rules in Figure 4.7. The reduction relation is needed to define *promotion* of actions. First we define the function fn for *free names* of processes.

Definition 4.1. The function $\mathsf{fn} : \mathsf{Proc} \to \mathcal{P}(\mathcal{K})$ is given as follows:

$$\mathsf{fn} : \mathsf{Proc} \to \mathcal{P}(\mathcal{K})$$

$$\mathsf{fn}(\mathbf{0}) = \emptyset$$

$$\mathsf{fn}(S) = \mathsf{fn}(P) \text{ if } S \stackrel{def}{=} P$$

$$\mathsf{fn}((\alpha : s; b).P) = \{\alpha\} \cup \mathsf{fn}(s; b).P)$$

$$\mathsf{fn}((a; b).P) = \{a, b\} \cup \mathsf{fn}(P)$$

$$\mathsf{fn}(P \mid Q) = \mathsf{fn}(P) \cup \mathsf{fn}(Q) \cup \gamma(\mathsf{fn}(P) \times \mathsf{fn}(Q))$$

$$\mathsf{fn}(P \backslash L) = \mathsf{fn}(P) \backslash L$$

Our reduction rules have names such as, for example, red and we write red: $P \Rightarrow Q$ to indicate that the reduction rule $P \Rightarrow Q$ is called red. The process P in the rule $P \Rightarrow Q$ is called a *redex*, and the process Q is called a *contractum*. A reduction rule $P \Rightarrow Q$ can be seen as a prescription for deriving rewrites $C[P] \Rightarrow C[Q]$ for arbitrary context $C[\,]$. A P redex may be replaced by its contractum Q in an arbitrary context $C[\,]$ giving rise to a reduction step: $C[P] \Rightarrow C[Q]$.

Definition 4.2. The reduction relation $\Rightarrow$ is the smallest reflexive and transitive relation on CCB processes that is preserved by all contexts, and that satisfies the rules in Figure 4.7.

Note that we do not want $\Rightarrow$ to be symmetric as we wish to apply prom only from left to right.

The rewrite rules in Figure 4.7 include prom, $\mathsf{move\text{-}r}$, and $\mathsf{move\text{-}l}$ which promote weak bonds (here b) to strong bonds (here a). The rule prom applies to the full version of our prefix operator (with the ; construct), and $\mathsf{move\text{-}r}$ and $\mathsf{move\text{-}l}$ apply only to the simple prefix. These three rules are here to model what happens in chemical systems: a bond on a weak action is temporary and as soon as there is a strong action that can accommodate that bond (as the result of concerted actions) the bond establishes itself on the strong action thus releasing the weak action. In order to align the use of these three rules to what happens in chemical reactions, we insist that they are used as soon as they becomes applicable: this is made precise

in Definition 4.3. We could have used the idea of ordering on SOS rules and rewrite rules [116, 82] to specify that the rewrite rules prom, move-r and move-r are higher in the ordering than all SOS rules and the remaining rewrite rules, implying that they should be applied first when deriving transitions. Alternatively, we could have tried to employ some of the techniques presented in [18] to define our transition relation. This would require the use of negative information in the premises, and the definitions in the style as those in [116, 82]. However, since we combine SOS rules with rewrite rules, we opted for a directly defined transition relation.

We now define the transition relation for the labelled transition system for CCB. Recall that the states of the LTS are processes in Proc and the labels are members of $\mathcal{A}$, $\mathcal{AK}$, $(\mathcal{A})\mathcal{K}$ and the concerted actions labels in $\mathcal{AK} \times \underline{\mathcal{A}}\mathcal{K}$. Let $d : \mathsf{Proc} \rightarrow \mathbb{N}$ be the operator depth function defined by $d(P) = 0$ if P is a constant, and $d(f(P_1, \ldots, P_n)) = 1 + max\{d(P_i) | 1 \leq i \leq n\}$ otherwise, where f is an operator of CCB. The transition relation is given as follows:

Definition 4.3. We associate to Proc and $\mathcal{AK} \cup \underline{\mathcal{A}}\mathcal{K} \cup (\mathcal{A})\mathcal{K} \cup (\mathcal{AK} \times \underline{\mathcal{A}}\mathcal{K})$ a transition relation $\rightarrow$ given by $\bigcup_{l<\omega} \rightarrow^l$, where transition relations $\rightarrow^l \subseteq \mathsf{Proc} \times \mathcal{AK} \cup \underline{\mathcal{A}}\mathcal{K} \cup (\mathcal{A})\mathcal{K} \cup (\mathcal{AK} \times \underline{\mathcal{A}}\mathcal{K}) \times \mathsf{Proc}$ are as follows, with $b \in \mathcal{AK}$ and $\mu \in \mathcal{AK} \cup \underline{\mathcal{A}}\mathcal{K} \cup (\mathcal{AK} \times \underline{\mathcal{A}}\mathcal{K})$:

1. $P \xrightarrow{(b)[k]} P' \in \rightarrow^l$ if $d(P) = l$, $P \xrightarrow{(b)[k]} P' = \rho(con(r))$, where r is either aux1 or aux2, and each premise in $pre(r)$ is a valid transition in $\bigcup_{k<l} \rightarrow^k$ or a valid predicate.

2. $P \xrightarrow{\mu} P' \in \rightarrow^l$ if $d(P) = l$, $P \Rightarrow Q$, for some Q such that Q does not contain any prom, move-r and move-l redex, $Q \xrightarrow{\mu} Q' = con(r)$, for some rule r where each member of $pre(r)$ is either a valid transition in $\bigcup_{k<l} \rightarrow^k$, a valid rewrite or a valid predicate, and $Q' \Rightarrow P'$.

The first part of the definition specifies the auxiliary transitions using rules aux1 and aux2. The second part tells us how to use the remaining rules to define transitions. If P has no prom, move-r and move-l redex, then we apply our rules in a standard way. Otherwise, we are required to reduce P to Q with prom, move-r and move-l first, then we define a transition of Q to Q' in a standard way, and finally we reduce Q' to P' (if needed). This implies that if P has a prom, move-r or move-l redex, then we must use one of the structural congruence rules in Figure 4.6. And, if we use any of these rules, then the reduced process Q must no longer have any prom, move-r and move-l redex.

The next example illustrates the application of the promotion rewrite rule.

Example 4.6. The transition $(a[1];b) \mid a[1] \mid b \xrightarrow{\{d[2],\underline{c}[1]\}} (a;b[2]) \mid a \mid b[2]$ from Example 4.3 cannot be followed by a communication of actions a because there is a prom redex $(a;b[2])$ in $(a;b[2]) \mid a \mid b[2]$. The rewrite of this redex takes priority: the bond 2 moves from the weak b to the strong a by prom:

$$(a;b[2]) \mid a \mid b[2] \Rightarrow (a[2];b) \mid a \mid b[2]$$

As a result, we can bond on the weak b again and, importantly, the $a[2]$ to $b[2]$ bond is irreversible as $\gamma(a,b)$ is undefined. Note that reaching this bond by computing forwards alone is not possible.

We shall call henceforth the transitions derived by the forward SOS rules the *forward transitions* and, the transitions derived by the reverse SOS rules the *reverse transitions*. Correspondingly, there are the *concerted (action)* transitions.

Example 4.7. The prom and move rules can give rise to transitions which would be blocked by restrictions otherwise. We start with process $(e|(a;a')|b) \setminus c$ with $\gamma(a,b) = c$, $\gamma(a,e) = d$ and $\gamma(b,a') = f$. It would seem that the transition c cannot happen because of the restriction, even though a and b are ready. On the other hand we can execute this sequence: $(e|(a;a')|b) \setminus c \xrightarrow{d[1]} (e[1]|(a[1];a')|b) \setminus c \xrightarrow{\{f[2],\underline{f}[1]\}} (e|(a;a'[2])|b[2]) \setminus c \Rightarrow\ (e|(a[2];a')|b[2]) \setminus c$. The c is now a past action according to the keys, but has only happened indirectly, ignoring the restriction. The inference tree for this example is shown in Figure 4.10.

Whilst we have defined the semantics of CBB using SOS rules, there are other possibilities to achieve this. One was suggested in [14] and is based on Nominal Logic described in [40]. Here, we give an informal overview how the semantics of CCB could be defined using Nominal Logic. For this, we would define sorts. Based on this, a *nominal signature* Σ, which is a triple (Δ, A, F), can be described. Here Δ is a set of base sorts ranged over by σ, A is a set of atom sorts ranged over by $\mathbb{A}$, and F is a set of operators $f(\sigma_1 \times ... \times \sigma_n) \to \sigma$. For our calculus we would define a base sort O for processes and an atom sort C for actions. Instead of a labelled transition system (LTS), as we did previously, we define a *nominal transition system specification* (NTSS), which is a triple (Σ, R, D), where Σ is a nominal signature (as defined before), R is a set of (transition) relation symbols, and D is a set of derivation rules. In our calculus, a transition relation symbol for example for the action prefix would be:

$$transition(a,x) : (C \times O) \to O$$

Here, $transition(a,P)$ would correspond to a process $(s,a,s';b).P$. We would also define freshness of keys fresh, standard of processes std, and the synchronisation

$$\text{par}\frac{\text{act1}\dfrac{\mathsf{std}(0) \qquad \mathsf{fsh}[7](i[1], j[2])}{(i[1], j[2], k) \xrightarrow{k[7]} (i[1], j[2], k[7])}}{(i[1], j[2], k) \mid (f[3], g[4], h) \xrightarrow{k[7]} (i[1], j[2], k[7]) \mid (f[3], g[4], h)} \quad \textbf{(1)}$$

$$\text{par}\frac{\text{act1}\dfrac{\mathsf{std}(0) \qquad \mathsf{fsh}[4](f[3])}{(f[3], g[4], h) \xrightarrow{\underline{g}[4]} (f[3], g, h)}}{(i[1], j[2], k[7]) \mid (f[3], g[4], h) \xrightarrow{\underline{g}[4]} (i[1], j[2], k[7]) \mid (f[3], g, h)} \quad \textbf{(2)}$$

$$\text{aux1}\ \frac{\mathsf{std}(0) \qquad \mathsf{fsh}[7](a[1], b[2], c[3], d[4])}{(a[1], b[2], c[3], d[4]; e) \xrightarrow{(e)[7]} (a[1], b[2], c[3], d[4]; e[7])} \quad \textbf{(3)}$$

$$\text{rev act1}\frac{\mathsf{std}(0) \qquad \mathsf{fsh}[4]((a[1], b[2], c[3], e[7])}{\text{sc}\dfrac{(a[1], b[2], c[3], d[4]; e[7]) \xrightarrow{\underline{d}[4]} (a[1], b[2], c[3], d; e[7]) \qquad (a[1], b[2], c[3], d; e[7]) \Rightarrow (a[1], b[2], c[3], d[7]; e)}{(a[1], b[2], c[3], d[4]; e[7]) \xrightarrow{\underline{d}[4]} (a[1], b[2], c[3], d[7]; e)}} \quad \textbf{(4)}$$

$$\text{concert}\ \frac{\textbf{(1)} \qquad \textbf{(2)} \qquad \textbf{(3)} \qquad \textbf{(4)}}{(a[1], b[2], c[3], d[4]; e) \mid (f[3], g[4], h) \mid (i[1], j[2], k) \xrightarrow{\{l[7], \underline{p}[4]\}} (a[1], b[2], c[3], d[7]; e) \mid (f[3], g, h) \mid (i[1], j[2], k[7])}$$

Figure 4.10: Inference tree for Example 4.7.

function with respect to nominal signatures. Using these functions, we could finally define derivation rules. For example, the equivalent rule of act1 would be:

$$\frac{\mathsf{std}(P) \quad \mathsf{fsh}[k](s, s')}{transition(a, (s, a, s'; b).x) \xrightarrow{transition(a)} (s, a[k], s'; b).x}$$

In a similar manner, all transition rules could be defined based on nominal signatures.

4.2 Properties of CCB

In this section we establish several properties of the LTS for CCB. We start by showing the expected properties of keys, namely that when an action takes place it uses a fresh key, and when a past action is undone its key is removed from the resulting process. We also show that the reverse transitions invert the corresponding forward transitions, and vice versa.

Definition 4.4. A process P is *consistent* if $Q \rightarrow^* P$ for some process Q such that $\mathsf{std}(Q)$.

Remark 4.1. It is decideable if a process P is consistent. Firstly, we define the root of a process as the process stripped of all keys. This can be done using a function similar to the pruning map in Section 4.3, but including weak actions. We can show that every process has exactly one root, as done in [90], p. 82. It can be checked algorithmically if a process is consistent. An algorithm could work as given in Figure 4.11, with the resulting process being the root of a consistent process. We assume that keys have been used in ascending order in order to keep the argument clear. If this would not have been the case, we would need additional rules for finding the last key used, which would be possible.

To demonstrate this, we take a sample process with some past actions:

$((a).(b[3]).\mathbf{0} \mid (c[2]).(d[3]).\mathbf{0})\backslash\{c\}$

The synchronisation function is empty. This process is not std, so if it is consistent, we should be able to use the algorithm to show this. The highest key is 3. There are two actions with this key, so it must be the result of a communication. Actions b and d are not part of the synchronisation function, so we determine the process not to be consistent at this point. If $\gamma(a, d)$ would be part of the synchronisation function, we could continue. At this point, we determine that the multiset before (b) has fresh actions, which again means that the process is not consistent. For the

1 Let P be the process to check
2 *while* ($\mathsf{keys}(P) \neq \emptyset$)
3 Let k be the highest key in p
4 Let *act* be the set of actions with key k in p
5 *if* ($|act| == 2$) *then*
6 *if* (*actions are not in* γ *or actions are not in parallel*) *then*
7 P not consistent; end.
8 *else*
9 remove k from P
10 *else*
11 *if* (*act restricted*)
12 P not consistent; end.
13 *else*
14 remove k from P
15 If the removed key was the last past action in a multiset and there is a multiset before it, check that the previous multiset in this process has only past action and potentially a $;b$ element. If not, P is not consistent; end.
16 P is consistent.

Figure 4.11: Algorithm for deciding consistency of processes.

second subprocess, the multiset before (d) has only past actions, which is in line with a consistent process. On the other hand, c is a past action whilst c is restricted, so this again makes the process not consistent.

If the process would be

$$((a[1]).(b[3]).\mathbf{0} \mid (c[2]).(d[3]).\mathbf{0})\backslash\{\}$$

with $\gamma(b, d) = e$, it would be consistent. Whilst executing the algorithm, we remove all keys and finish the algorithm with the result of the process being consistent.

Proposition 4.1. *Let P be consistent. Then*
1. If $P \xrightarrow{a[k]} Q$ then $k \notin \mathsf{keys}(P)$ and $\mathsf{keys}(Q) = \mathsf{keys}(P) \cup \{k\}$ for all Q.
2. If $P \xrightarrow{\underline{a}[k]} Q$ then $k \in \mathsf{keys}(P)$ and $\mathsf{keys}(Q) = \mathsf{keys}(P) \setminus \{k\}$ for all Q.
3. If $P \xrightarrow{a[k]} P'$ then $P' \xrightarrow{\underline{a}[k]} P$. If $P' \xrightarrow{\underline{a}[k]} P$ and P' has no move-r *or* move-l *redexes, then $P \xrightarrow{a[k]} P'$.*

Proof. 1. We prove Proposition 4.1.1 by induction on the depth of the inference tree of $P \xrightarrow{\mu[k]} Q$.

1. Base case follows for processes with the inference tree of depth 0.

2. Inductive hypothesis: We assume that Proposition 4.1.1 holds for all subprocesses R of P and all $\nu[l]$, namely if R is a consistent process and $R \xrightarrow{\nu[l]} R'$ for some R' then $k \notin \mathsf{keys}(R)$ and $\mathsf{keys}(R') = \mathsf{keys}(R) \cup l$.

3. Induction step: We show that Proposition 4.1.1 holds for P. For this we consider cases depending on the structure of P:

   (a) $P \equiv (s; b).R$: There are two cases:

      i. $P \xrightarrow{\mu[k]} P'$ by rule act1 in Figure 4.2: Assume P is consistent, $P \xrightarrow{\mu[k]} P'$ and s is $\mu : s'$. Since $P \xrightarrow{\mu[k]} P'$ by rule act1 we get $\mathsf{fsh}[k](s)$, meaning k does not appear in s and $\mathsf{std}(R)$ which means R contains no keys, hence $k \notin \mathsf{keys}(P)$. Also $P \xrightarrow{\mu[k]} P'$ by act1 means $(\mu : s'; b).R \xrightarrow{\mu[k]} (\mu[k] : s'; b).R$. That means we can calculate $\mathsf{keys}(P')$ as follows: $\mathsf{keys}(P') = \mathsf{keys}((\mu[k] : s'; b).R) = \{k\} \cup \mathsf{keys}((\mu : s'; b).R) = \{k\} \cup \mathsf{keys}(P)$ as required.

      ii. $P \xrightarrow{\mu[k]} P'$ by rule act2 in Figure 4.2: Assume P is consistent and $P \xrightarrow{\mu[k]} P'$. We deduce that $P \equiv (t; b).R$ and $P' \equiv (t; b).R'$. Recall that t denotes a sequence of past actions, namely an element from $\mathcal{AK}^*$. The premises of act2 ensure that $\mathsf{fsh}[k](t)$ and $R \xrightarrow{\mu[k]} R'$. R is consistent since P is consistent by assumption. Since R is consistent and $R \xrightarrow{\mu[k]} R'$ then, by the inductive hypothesis, $k \notin \mathsf{keys}(R)$ and $\mathsf{keys}(R') = \mathsf{keys}(R) \cup \{k\}$. Since $k \notin \mathsf{keys}(R)$ and $\mathsf{fsh}[k](t)$ we obtain $k \notin \mathsf{keys}((t; b).P)$ as required. We can also calculate $\mathsf{keys}(P')$: $\mathsf{keys}(P') = \mathsf{keys}((t; b).R') = \mathsf{keys}((t; b)) \cup \mathsf{keys}(R') = \mathsf{keys}((t; b)) \cup \mathsf{keys}(R) \cup \{k\} = \mathsf{keys}((t; b).R) \cup \{k\} = \mathsf{keys}(P) \cup \{k\}$ as required.

   (b) $P \equiv R \mid Q$: There are two cases:

      i. This happens by rule par in Figure 4.2. Assume P is consistent and $P \xrightarrow{\mu[k]} P'$. Since $P \xrightarrow{\mu[k]} P'$, by rule par, $R \xrightarrow{\mu[k]} R'$ must hold as well as $\mathsf{fsh}[k](Q)$, meaning k does not appear in Q. Since $k \notin \mathsf{keys}(R)$ by the inductive hypothesis and $k \notin \mathsf{keys}(Q)$, it follows $k \notin \mathsf{keys}(R \mid Q)$ since $\mathsf{keys}(R \mid Q) = \mathsf{keys}(R) \cup \mathsf{keys}(Q)$ according to the definition of keys in Section 4. Thus $k \notin \mathsf{keys}(P)$ since $\mathsf{keys}(P) = \mathsf{keys}(R \mid Q)$ as above.
      Also $\mathsf{keys}(P') = \mathsf{keys}(R') \cup \mathsf{keys}(Q)$ according to the definition of keys.
      Also $\mathsf{keys}(R') = \mathsf{keys}(R) \cup \{k\}$ by the inductive hypothesis. Hence,

$\mathsf{keys}(P') = \mathsf{keys}(R) \cup \{k\} \cup \mathsf{keys}(Q) = \mathsf{keys}(R) \cup \mathsf{keys}(Q) \cup \{k\} = \mathsf{keys}(P) \cup \{k\}$ as required.

ii. This happens by rule com in Figure 4.2. Assume P is consistent and $P \xrightarrow{\mu[k]} P'$. Since $P \xrightarrow{\mu[k]} P'$, by rule con, $R \xrightarrow{\nu_1[k]} R'$, $Q \xrightarrow{\nu_2[k]} Q'$ and $\gamma(nu_1, nu_2) = \mu$ must hold without loss of generality. The inductive hypothesis in this case is not only true about R, but also Q, so that if Q is a consistent process and $Q \xrightarrow{\nu_2[k]} Q'$ then $k \notin \mathsf{keys}(Q)$ and $\mathsf{keys}(Q') = \mathsf{keys}(Q) \cup k$. We deduce that $k \notin \mathsf{keys}(P)$ since $\mathsf{keys}(P) = \mathsf{keys}(R \mid Q) = \mathsf{keys}(R) \cup \mathsf{keys}(Q)$ and $k \notin \mathsf{keys}(R)$ and $k \notin \mathsf{keys}(Q)$.
We can calculate $\mathsf{keys}(P')$: $\mathsf{keys}(P') = \mathsf{keys}(R' \cup \mathsf{keys}(Q)') = \mathsf{keys}(R') \cup \mathsf{keys}(Q') = \mathsf{keys}(R) \cup k \cup \mathsf{keys}(Q) \cup k = \mathsf{keys}(R) \cup \mathsf{keys}(Q) \cup k = \mathsf{keys}(R \mid Q) \cup k = \mathsf{keys}(p) \cup k$ as required.

(c) $P \equiv R \setminus L$: This is by rule res in Figure 4.2. Assume P is consistent and $P \xrightarrow{\mu[k]} P'$. Since $P \xrightarrow{\mu[k]} P'$ by rule res we obtain $R \xrightarrow{\mu[k]} R'$ and $\mu \notin L$. Since $k \notin \mathsf{keys}(R)$ by the inductive hypothesis it follows that $k \notin \mathsf{keys}(R \setminus L)$ since restriction does not change the keys of the process by definition of keys. Also $\mathsf{keys}(P) = \mathsf{keys}(R)$ and $\mathsf{keys}(P') = \mathsf{keys}(R')$ according to the definition of keys. Since by the inductive hypothesis $\mathsf{keys}(R') = \mathsf{keys}(R) \cup \{k\}$ we can calculate $\mathsf{keys}(P')$: $\mathsf{keys}(P') = \mathsf{keys}(R' \setminus L) = \mathsf{keys}(R') = \mathsf{keys}(R) \cup \{k\} = \mathsf{keys}(P) \cup \{k\}$ as required.

(d) $P \equiv S$ with $S \stackrel{def}{=} R$: Assume P is consistent and $P \xrightarrow{\mu[k]} P'$ by rule con in Figure 4.2. Hence $R \xrightarrow{\mu[k]} P'$ by con and, by the inductive hypothesis (as the depth of the inference tree for $R \xrightarrow{\mu[k]} P'$ is clearly less than the depth of the inference tree for $P \xrightarrow{\mu[k]} P'$) $k \notin \mathsf{keys}(R)$ and $\mathsf{keys}(S) = \mathsf{keys}(R) \cup k$. Since the definition of keys includes $\mathsf{keys}(S) = \mathsf{keys}(R)$ if $S \stackrel{def}{=} R$, it follows that $k \notin \mathsf{keys}(P)$ and $\mathsf{keys}(P') = \mathsf{keys}(P) \cup k$ as required.

2. The proof of part 2 is by induction on the depth of inference trees for transitions and since it is very similar to the proof of part 1 above it is omitted.

3. We prove part 3 by considering each implication separately:

1. $P \xrightarrow{\mu[k]} P' \Rightarrow P' \xrightarrow{\underline{\mu}[k]} P$: We use induction on the depth of the inference tree of $P \xrightarrow{\mu[k]} P'$.

(a) Base case follows for processes with the inference tree of depth 0.

(b) Inductive hypothesis: We assume that the property holds for all subprocesses R of P, namely if R is a consistent process and if $R \xrightarrow{\nu[l]} R'$ for some R' then $R' \xrightarrow{\underline{\nu}[l]} R$.

(c) Induction step: We consider cases depending on the structure of P. Assume that P is consistent and $P \xrightarrow{a[k]} P'$.

i. $P \equiv (s; b).R$: Here we distinguish two cases:

A. $P \xrightarrow{\mu[k]} P'$ by rule act1 if s contains at least one fresh action: Here s is of the structure $\mu : s', t$ and the transition is $(\mu : s', t; b).R \xrightarrow{a[k]} (\mu[k] : s', t; b).R$ by act1. This implies that $\mathsf{std}(R)$ and $\mathsf{fsh}[k](s' \cup t)$. Because of this $(\mu[k] : s', t; b).R \xrightarrow{\underline{\mu}[k]} (\mu : s', t; b).R$ can happen by rule rev act1 and $P' \xrightarrow{\underline{\mu}[k]} P$ where $P' \equiv (\mu[k] : s', t; b).R$ as required.

B. $P \xrightarrow{\mu[k]} P'$ by rule act2 if there are no fresh actions in s: Here the transition must happen in R. Since, by the inductive hypothesis, the property is true for R, namely $R' \xrightarrow{\underline{\mu}[k]} R$, we deduce by rule rev act2 that $P' \xrightarrow{\underline{\mu}[k]} P$.

ii. $P \equiv Q \mid R$: There are two cases:

A. Transition by rule par: We deduce without loss of generality $R \xrightarrow{\mu[k]} R'$ and $\mathsf{fsh}[k](Q)$. By the inductive hypothesis $R' \xrightarrow{\underline{\mu}[k]} R$ must hold. By rev par, $Q \mid R' \xrightarrow{\underline{\mu}[k]} Q \mid R$ follows. Hence, $P' \xrightarrow{\underline{\mu}[k]} P$ where $P' \equiv Q \mid R'$.

B. Transition by rule com: Assume that P is consistent, $\gamma(\nu_1, \nu_2) = \mu$ and $P \xrightarrow{\mu[k]} P'$. Since this happens by rule com $Q \xrightarrow{\nu_1[k]} Q'$ and $R \xrightarrow{\nu_2[k]} R'$. Because of the inductive hypothesis, $Q' \xrightarrow{\underline{\nu_1}[k]} Q$ and $R' \xrightarrow{\underline{\nu_2}[k]} R$. By rule rev com we obtain $Q' \mid R' \xrightarrow{\underline{\mu}[k]} Q \mid R$. Hence, $P \xrightarrow{\underline{\mu}[k]} P$ where $P' \equiv Q' \mid R'$.

iii. $P \equiv R \setminus L$: This is by rule res. For $P \xrightarrow{a[k]} P'$ to be valid $a \notin L$ and $R \xrightarrow{\mu[k]} R'$ must be true. Hence, by the inductive hypothesis, $R \xrightarrow{\underline{\mu}[k]} R'$. Since $\mu \notin L$, by rule rev res, $R' \setminus L \xrightarrow{\underline{\mu}[k]} R \setminus L$ and, hence, $P' \xrightarrow{\underline{\mu}[k]} P$ where $P' \equiv R' \setminus L$.

iv. $P \equiv S$ with $S \stackrel{def}{=} R$: $P \xrightarrow{\mu[k]} P'$ by rule con in Figure 4.2. There is a transition $R \xrightarrow{\mu[k]} R'$ and, by the inductive hypothesis, $R' \xrightarrow{\underline{\mu}[k]} R$ is true. Since $R' \xrightarrow{\underline{\mu}[k]} R$ and $S \stackrel{def}{=} R$ we get $R' \xrightarrow{\underline{a}[k]} S$ by rev con. Hence $P' \xrightarrow{\underline{\mu}} P$ where $P' \equiv R'$.

2. $P \xrightarrow{\mu[k]} P' \Rightarrow P' \xrightarrow{\underline{\mu}[k]} P$: The proof works similar to the first part of proof of Proposition 4.1.3 and is thus omitted.

Next, we introduce some notation. We define a new transition relation $\longmapsto$ by $P \overset{a[k]}{\longmapsto} Q$ if $P \xrightarrow{a[k]} Q$ or $P \xrightarrow{\underline{a}[k]} Q$. Process P is called the *source* and Q the *target* of $P \overset{a[k]}{\longmapsto} Q$. We will use $t, t', t_1, \ldots$ to denote transitions, for example $t : P \overset{a[k]}{\longmapsto} Q$. Two $\longmapsto$ transitions are *coinitial* if they have the same source, and they are *cofinal* if their targets are identical.

We define when two transitions are concurrent.

Definition 4.5. Two coinitial transitions $P \overset{a[k]}{\longmapsto} P'$ and $P \overset{b[l]}{\longmapsto} P''$ are *concurrent* if there exists $M \neq P$ such that $P' \overset{b[l]}{\longmapsto} M$ and $P'' \overset{a[k]}{\longmapsto} M$.

Note that two concurrent transitions are coinitial and, together with the two transitions (with the target M) required by Definition 4.5, they form a "diamond" structure with the nodes P, P', P'' and M.

When transitions in Definition 4.5 are forward, we may not be able to complete the diamond as the following example shows. In such case, we say that the transitions are in *conflict*. Consider $(a) \mid (b) \mid (b)$ with $\gamma(a, b) = c$. The two coinitial transitions below are in conflict:

$$\begin{array}{lcl} (a) \mid (b) \mid (b) & \xrightarrow{c[1]} & (a[1]) \mid (b[1]) \mid (b) \\ (a) \mid (b) \mid (b) & \xrightarrow{c[2]} & (a[2]) \mid (b) \mid (b[2]) \end{array}$$

However, coinitial reverse transitions are concurrent. We shall denote the syntactical equality of process expressions by $\equiv$.

Proposition 4.2 (Reverse Diamond)**.** *Let P be a consistent process and let $t' : P \xrightarrow{\underline{a}[k]} P'$ and $t'' : P \xrightarrow{\underline{b}[l]} P''$ with $l \neq k$. Then t' and t'' are concurrent.*

Proof. We prove Proposition 4.2 by induction on the depth of the inference tree for transition of P.

1. Base case: Processes with an inference tree of depth 0 have no reverse transitions, so the proposition is valid.

2. Inductive hypothesis: We assume that for all subprocesses R of P and all $\underline{c}[m], \underline{d}[n]$, if R is a consistent process, $R \xrightarrow{\underline{c}[m]} R'$ and $R \xrightarrow{\underline{d}[n]} R''$, with $m \neq n$, then there is an N so that $R' \xrightarrow{\underline{d}[n]} N$ and $R'' \xrightarrow{\underline{c}[m]} N$.

3. Induction step: We consider cases depending on the structure of P:

   (a) $P \equiv (s; b).R$ with s containing two or more past actions: This includes the the case $P \equiv (t; b).R$. This is by rule rev act1. With s'

being the sequence obtained from s by removing $a[k]$ and $b[l]$ with $k, l \notin \mathsf{keys}(s')$ we have $(a[k], b[l], s'; c).R \xrightarrow{\underline{a}[k]} (a, b[l], ts; c).R \xrightarrow{\underline{b}[l]} (a, b, s'; c).R$ or $(a[k], b[l], s'; c).R \xrightarrow{\underline{b}[l]} (a[k], b, s'; c).R \xrightarrow{\underline{a}[k]} (a, b, s'; c).R$. Let $M \equiv (a, b, s'; c).U \mid T$ as required.

(b) $P \equiv (s; b).R$ with s containing one or none past action: We cannot deduce the required transitions $P \xrightarrow{\underline{a}[k]} P'$ and $P \xrightarrow{\underline{b}[l]} P''$ for any a, b, k, l and $l \neq k$ by any SOS rule. Hence the proposition is vacuously valid.

(c) $P \equiv Q \mid R$: There are three cases:

i. $P \xrightarrow{\underline{a}[k]} P'$ by rule rev par and $P \xrightarrow{\underline{b}[l]} P'$ by rule rev par. There are two subcases here:

A. Transitions in the same subprocess: Assume without loss of generality $Q \xrightarrow{\underline{a}[k]} Q'$ and $Q \xrightarrow{\underline{b}[l]} Q''$. By the inductive hypothesis there is an N so that $Q' \xrightarrow{\underline{b}[l]} N$ and $Q'' \xrightarrow{\underline{a}[k]} N$. We can conclude by using rule rev par that $Q' \mid R \xrightarrow{\underline{b}[l]} N \mid R$ and $Q'' \mid R \xrightarrow{\underline{a}[k]} N \mid R$. With $M \equiv N \mid R$ we get the result.

B. Transitions in different subprocesses: Assume without loss of generality that $Q \xrightarrow{\underline{a}[k]} Q'$ and $R \xrightarrow{\underline{b}[l]} R'$. By rule rev par $Q \mid R \xrightarrow{\underline{a}[k]} Q' \mid R \xrightarrow{\underline{b}[l]} Q' \mid R'$ and $Q \mid R \xrightarrow{\underline{b}[l]} Q \mid R' \xrightarrow{\underline{a}[k]} Q' \mid R'$ are valid. These form the required reversal diamond with $M \equiv Q' \mid R'$.

ii. $P \xrightarrow{\underline{a}[k]} P'$ by rule rev com and $P \xrightarrow{\underline{b}[l]} P'$ by rule rev par: Without loss of generality this covers all cases with one rev par and one rev com transition. We assume that $\underline{a}[k]$ is by rule rev com, that $\gamma(a_1, a_2) = a$ and that $\underline{b}[l]$ is by rev par. We also assume that b happens in Q, so that $Q \xrightarrow{\underline{b}[l]} Q'$ and $\mathsf{fsh}[l](R)$, and that $Q \xrightarrow{\underline{a_1}[k]} Q''$ and $R \xrightarrow{\underline{a_2}[k]} R'$. We know that $l \neq k$ because $\mathsf{fsh}[l](R)$ and $R \xrightarrow{\underline{a_2}[k]} R''$, a transition which could not happen if $l = k$, since according to Proposition 4.1.2 a key cannot be fresh for a reverse transition to happen with this key. By the inductive hypothesis there is an N so that $Q' \xrightarrow{\underline{a_1}[k]} N$ and $Q'' \xrightarrow{\underline{b}[l]} N$. Using the rev com rule we can deduce $P \xrightarrow{\underline{a}[k]} Q'' \mid R'$, $P \xrightarrow{\underline{b}[l]} Q' \mid R$, $Q'' \mid R' \xrightarrow{\underline{b}[l]} N \mid R'$ and $Q' \mid R \xrightarrow{\underline{a}[k]} N \mid R'$. Taking $M \equiv N \mid R'$ we get the result.

iii. $P \xrightarrow{\underline{a}[k]} P'$ by rev com and $P \xrightarrow{\underline{b}[l]} P''$ by rev com: Without loss of generality this covers all cases with two rev com transitions. We assume $\gamma(a_1, a_2) = a$ and $\gamma(b_1, b_2) = b$. Also $Q \xrightarrow{\underline{a_1}[k]} Q'$, $Q \xrightarrow{\underline{b_1}[l]} Q''$, $R \xrightarrow{\underline{a_2}[k]} R'$ and $R \xrightarrow{\underline{b_2}[l]} R''$. Since $Q \xrightarrow{\underline{a_1}[k]} Q'$ and $Q \xrightarrow{\underline{b_1}[l]} Q''$ by the inductive hypothesis it follows that there is an N so that $Q' \xrightarrow{\underline{b_1}[l]} N$ and $Q'' \xrightarrow{\underline{a_1}[k]} N$ and since $R \xrightarrow{\underline{a_2}[k]} R'$ and $R \xrightarrow{\underline{b_2}[l]} R''$ there is an N' so that $R' \xrightarrow{\underline{b_2}[l]} N'$ and $R'' \xrightarrow{\underline{a_2}[k]} N'$. By rule rev par it follows that

$P \xrightarrow{a[k]} Q' \mid R' \xrightarrow{b[l]} N \mid N'$ and $P \xrightarrow{b[l]} Q'' \mid R'' \xrightarrow{a[k]} N \mid N'$. Let $M \equiv N \mid N'$ as required.

(d) Cases $P \equiv R \setminus L$ and $P \equiv S$ with $S \stackrel{def}{=} R$ follow in a standard way by using rules rev res and rev con in Figure 4.4, respectively, and the inductive hypothesis.

Before we show that coinitial forward transitions are concurrent if they result in cofinal computations, we introduce *traces*. A trace is a sequence of composable forward and reverse transitions over CCB. Traces are ranged over by $\sigma, \sigma', \sigma_1, \ldots$. Two transitions are composable if the target of the first transition is the source of the second transition. The composition of transitions and traces is denoted by ';'. The *source* of a trace is the source of the first transition of the trace, and the *target* of a trace is the target of the last transition in the trace. As with transitions, two traces are *coinitial* if they have the same source, and they are *cofinal* if their targets are identical. The syntactical equality of transitions is also denoted by $\equiv$.

Proposition 4.3 (Forward Diamond)**.** *If P is a consistent process and $t_1 \equiv P \xrightarrow{a[k]} P'$, $t_2 \equiv P \xrightarrow{b[l]} P''$, with $l \neq k$, and $P' \rightarrow^* R$ and $P'' \rightarrow^* R$, for some R, then there is $M \not\equiv P$ such that $P' \xrightarrow{b[l]} M$, $P'' \xrightarrow{a[k]} M$ and $M \rightarrow^* R$.*

Proof. By induction on the depth of the inference tree for transition of P.

1. Base case: obvious.

2. Inductive hypothesis: We assume that Proposition 4.3 holds for all subprocesses R of P and all $c[m], d[n]$, namely if R is a consistent process, $t'_1 \equiv R \xrightarrow{c[m]} R'$ and $t'_2 \equiv R \xrightarrow{d[n]} R''$ with $m \neq n$, and $t'_1; \sigma'_1$ and $t'_2; \sigma'_2$, for some σ'_1 and σ'_2, are cofinal then there is an N so that $R' \xrightarrow{d[n]} N$, $R'' \xrightarrow{c[m]} N$ and $N \rightarrow^* N'$ is cofinal with σ'_1 and σ'_2.

3. Induction step: We assume $t_1 \equiv P \xrightarrow{a[k]} P'$ and $t_2 \equiv P \xrightarrow{b[l]} P''$, and consider cases depending on the structure of P:

   (a) $P \equiv (s; b).R$ with s containing two or more fresh actions: This happens by rule act1. s is of the structure a, b, s' with $k, l \notin \mathsf{keys}(s')$ so that $P \equiv (a, b, s'; b).R$. So we have $t_3 \equiv P' \xrightarrow{b[l]} (a[k], b[l], s'; c).R$ and $t_4 \equiv P'' \xrightarrow{a[k]} (a[k], b[l], s'; c).R$. With $M \equiv (a[k], b[l], s'; c).R$ this gives us the result.

   (b) $P \equiv Q \mid R$:

      i. $P \xrightarrow{a[k]} P'$ by rule par and $P \xrightarrow{b[l]} P'$ by rule par.

A. Transitions in the same subprocess: Assume without loss of generality $Q \xrightarrow{a[k]} Q'$ and $Q \xrightarrow{b[l]} Q''$. By the inductive hypothesis there is an N so that $Q' \xrightarrow{b[l]} N$ and $Q'' \xrightarrow{a[k]} N$ and there is a T so that $Q' \rightarrow^* T$ and $Q'' \rightarrow^* T$ implying that $a_1[k]$ and $b[l]$ do not exclude the execution of each other in Q. We can conclude, by rule par, that $Q' \mid R \xrightarrow{b[l]} N \mid R$ and $Q'' \mid R \xrightarrow{a[k]} N \mid R$. Taking $M \equiv N \mid R$ gives the result.

B. Transitions in different subprocesses: Assume without loss of generality that $Q \xrightarrow{a[k]} Q'$ and $R \xrightarrow{b[l]} R'$. By rule par $Q \mid R \xrightarrow{a[k]} Q' \mid R \xrightarrow{b[l]} Q' \mid R'$ and $Q \mid R \xrightarrow{b[l]} Q \mid R' \xrightarrow{a[k]} Q' \mid R'$ are valid. These form the required forward diamond with $M \equiv Q' \mid R'$.

ii. $P \xrightarrow{a[k]} P'$ by rule com and $P \xrightarrow{b[l]} P'$ by rule par: Without loss of generality this covers all cases with one par and one com transition. We assume that $a[k]$ happens by rule com, that $\gamma(a_1, a_2) = a$ and that $b[l]$ happens by rule par. We also assume that b happens in Q, so that $Q \xrightarrow{b[l]} Q'$ and $\mathsf{fsh}[l](R)$, and that $Q \xrightarrow{a_1[k]} Q''$ and $R \xrightarrow{a_2[k]} R'$. Also there is a T so that $Q' \rightarrow^* T$ and $Q'' \rightarrow^* T$ implying that $a_1[k]$ and $b[l]$ do not exclude the execution of each other in Q. By the inductive hypothesis there is an N so that $Q' \xrightarrow{a_1[k]} N$ and $Q'' \xrightarrow{b[l]} N$. By com we deduce that $P \xrightarrow{a[k]} Q'' \mid R'$, $P \xrightarrow{b[l]} Q' \mid R$, $Q'' \mid R' \xrightarrow{b[l]} N \mid R'$ and $Q' \mid R \xrightarrow{a[k]} N \mid R'$. With $M \equiv N \mid R'$ this gives us the result.

iii. $P \xrightarrow{a[k]} P'$ by com and $P \xrightarrow{b[l]} P''$ by com: Without loss of generality this covers all cases with two com transitions. We assume that $\gamma(a_1, a_2) = a$ and $\gamma(b_1, b_2) = b$. Also $Q \xrightarrow{a_1[k]} Q'$, $Q \xrightarrow{b_1[l]} Q''$, $R \xrightarrow{a_2[k]} R'$ and $R \xrightarrow{b_2[l]} R''$. Since $Q \xrightarrow{a_1[k]} Q'$ and $Q \xrightarrow{b_1[l]} Q''$ by the inductive hypothesis it follows that there is an N so that $Q' \xrightarrow{b_1[l]} N$ and $Q'' \xrightarrow{a_1[k]} N$ and since $R \xrightarrow{a_2[k]} R'$ and $R \xrightarrow{b_2[l]} R''$ there must be an N' so that $R' \xrightarrow{b_2[l]} N'$ and $R'' \xrightarrow{a_2[k]} N'$. Also, there is a T so that $Q' \rightarrow^* T$ and $Q'' \rightarrow^* T$, implying that $a_1[k]$ and $b_1[l]$ do not exclude the execution of each other in Q and there is a T' so that $R' \rightarrow^* T'$ and $R'' \rightarrow^* T'$, implying that $a_2[k]$ and $b_2[l]$ do not exclude the execution of each other in R. By par $P \xrightarrow{a[k]} Q' \mid R' \xrightarrow{b[l]} N \mid N'$ and $P \xrightarrow{b[l]} Q'' \mid R'' \xrightarrow{a[k]} N \mid N'$. We let $M \equiv N \mid N'$ as required.

(c) cases $P \equiv R \setminus L$ and $P \equiv S$ with $S \stackrel{def}{=} R$ follow a standard way.

The next subsection explores some properties of transitions by concerted actions.

4.2.1 Concerted actions

The properties of keys corresponding to those in parts 1 and 2 of Proposition 4.1 hold also for the transitions by concerted actions in CCB.

Proposition 4.4. *Let P be consistent. If $P \xrightarrow{\{\mu[k],\underline{\nu}[l]\}} Q$ then $k \notin \mathsf{keys}(P)$, $l \in \mathsf{keys}(P)$ and $\mathsf{keys}(Q) = (\mathsf{keys}(P) \cup \{k\}) \setminus \{l\}$ for all Q.*

Proof. By induction on the depth of the inference tree of $P \xrightarrow{\{\mu[k],\underline{\nu}[l]\}} Q$.

1. Base case follows for processes with the inference tree of depth 0.
2. Inductive hypothesis: We assume that Proposition 4.4 holds for all subprocesses R of P and all $\nu[l]$ and all $\mu[k]$, namely if R is a consistent process and $R \xrightarrow{\{\mu[k],\underline{\nu}[l]\}} R'$ for some R' then $k \notin \mathsf{keys}(R)$, $l \in \mathsf{keys}(R)$ and $\mathsf{keys}(R') = (\mathsf{keys}(R) \cup \{k\}) \setminus \{l\}$.
3. Induction step: We show that Proposition 4.4 holds for P. For this we consider cases depending on the structure of P:
   (a) $P \equiv R \mid Q$: There are two cases here:
      i. The transition is by the rule concert in Figure 4.5: Assume P is consistent and $P \xrightarrow{\{\mu[k],\underline{\nu}[l]\}} P'$ for some P'. Since $P \xrightarrow{\{\mu[k],\underline{\nu}[l]\}} P'$, by rule concert, we have $R \xrightarrow{(\mu_1)[k]} R'$, $Q \xrightarrow{(\mu_2)[k]} Q'$ or $Q \xrightarrow{\mu_2[k]} Q'$, $R' \xrightarrow{\underline{\nu_1}[l]} R''$, $Q' \xrightarrow{\underline{\nu_2}[l]} Q''$, $\gamma(\nu_1, \nu_2) = \nu$ and $\gamma(\mu_1, \mu_2) = \mu$ must hold without loss of generality. Also, $P' \equiv R'' \mid Q''$. We can calculate $\mathsf{keys}(P)$: $\mathsf{keys}(P) = \mathsf{keys}(R) \cup \mathsf{keys}(Q)$. Since R and Q can both perform a transition with k, we know, by argument similar to that used in the proof of Proposition 4.1.1, that $k \notin \mathsf{keys}(R)$ and $k \notin \mathsf{keys}(Q)$, and therefore $k \notin \mathsf{keys}(P)$ as required. Since R' and Q' can perform undoing of actions with the key l, we know that $l \in \mathsf{keys}(R')$ and $l \in \mathsf{keys}(Q')$ by Proposition 4.1.2. Transitions of R and Q to R' and Q' respectively do not involve l, so $l \in \mathsf{keys}(R)$ and $l \in \mathsf{keys}(Q)$ must hold. Since $\mathsf{keys}(P) = \mathsf{keys}(R \cup Q)$ we deduce that $l \in \mathsf{keys}(P)$ as required. Because of Proposition 4.1.1 and Proposition 4.1.2 we can calculate $\mathsf{keys}(Q'')$: $\mathsf{keys}(Q'') = \mathsf{keys}(Q') \setminus \{l\} = (\mathsf{keys}(Q) \cup \{k\}) \setminus \{l\}$, $\mathsf{keys}(R'')$: $\mathsf{keys}(R'') = \mathsf{keys}(R') \setminus \{l\} = (\mathsf{keys}(R) \cup \{k\}) \setminus \{l\}$, and $\mathsf{keys}(P')$: $\mathsf{keys}(P') = \mathsf{keys}(R'') \cup \mathsf{keys}(Q'') = (\mathsf{keys}(R) \cup \{k\}) \setminus \{l\} \cup (\mathsf{keys}(Q) \cup \{k\}) \setminus \{l\} = (\mathsf{keys}(R) \cup \mathsf{keys}(Q) \cup \{k\}) \setminus \{l\} = (\mathsf{keys}(P) \cup \{k\}) \setminus \{l\}$ as required.

ii. The transition is by concert par rule in Figure 4.5: We assume without loss of generality $R \xrightarrow{\{\mu[k],\underline{\nu}[l]\}} R'$ and $\mathsf{fsh}[k](Q)$, and $P' \equiv R' \mid Q$. By the inductive hypothesis we have $k \notin \mathsf{keys}(R)$, $l \in \mathsf{keys}(R)$ and $\mathsf{keys}(R') = (\mathsf{keys}(R) \cup \{k\}) \setminus \{l\}$. Since $\mathsf{keys}(P) = \mathsf{keys}(R \mid Q) = \mathsf{keys}(R) \cup \mathsf{keys}(Q)$ and $\mathsf{fsh}[k](Q)$ and $k \notin \mathsf{keys}(R)$ we deduce that $k \notin \mathsf{keys}(P)$. Also, since $l \in \mathsf{keys}(R)$ and $\mathsf{keys}(P) = \mathsf{keys}(R \mid Q) = \mathsf{keys}(R) \cup \mathsf{keys}(Q)$ and we deduce that $l \in \mathsf{keys}(P)$. Hence, we have $\mathsf{keys}(P') = \mathsf{keys}(R') \cup \mathsf{keys}(Q) = (\mathsf{keys}(R) \cup \{k\}) \setminus \{l\} \cup \mathsf{keys}(Q)$. Since $l \notin \mathsf{keys}(Q)$ by concert par, we have $(\mathsf{keys}(R) \cup \{k\}) \setminus \{l\} \cup \mathsf{keys}(Q) = (\mathsf{keys}(R \mid Q) \cup \{k\}) \setminus \{l\} = (\mathsf{keys}(P) \cup \{k\}) \setminus \{l\}$ as required.

(b) $P \equiv (t; b).R$: The transition is by concert act rule in Figure 4.5. Assume P is consistent and $P \xrightarrow{\{\mu[k],\underline{\nu}[l]\}} P'$. We deduce that $P' \equiv (t; b).R'$ for some R'. Recall that t denotes a sequence of past actions, namely an element from $\mathcal{AK}^*$. The premises of concert act ensure that $\mathsf{fsh}[k](t)$ and $R \xrightarrow{\{\mu[k],\underline{\nu}[l]\}} R'$. The process R is consistent since P is consistent. Since R is consistent and $R \xrightarrow{\{\mu[k],\underline{\nu}[l]\}} R'$ then, by the inductive hypothesis, $k \notin \mathsf{keys}(R)$, $l \in \mathsf{keys}(R)$ and $\mathsf{keys}(R') = (\mathsf{keys}(R) \cup \{k\}) \setminus \{l\}$. Since $k \notin \mathsf{keys}(R)$ and $\mathsf{fsh}[k](t)$ we obtain $k \notin \mathsf{keys}((t; b).R) = \mathsf{keys}(P)$ as required. Since $l \in \mathsf{keys}(R)$ we obtain $l \in \mathsf{keys}((t; b).R) = \mathsf{keys}(P)$ as required. It is clear by the rules act2 and concert act that since $l \in \mathsf{keys}(R)$ the key l is not among the keys in t in $(t; b).R$. We now calculate $\mathsf{keys}(P')$: $\mathsf{keys}(P') = \mathsf{keys}((t; b).R') = \mathsf{k}(t) \cup \mathsf{k}(b) \cup \mathsf{keys}(R') = \mathsf{k}(t) \cup \mathsf{k}(b) \cup (\mathsf{keys}(R) \cup \{k\}) \setminus \{l\} = ((\mathsf{k}(t) \cup \mathsf{k}(b) \cup \mathsf{keys}(R)) \cup \{k\}) \setminus \{l\} = (\mathsf{keys}((t; b).R) \cup \{k\}) \setminus \{l\} = (\mathsf{keys}(P) \cup \{k\}) \setminus \{l\}$ as required.

(c) $P \equiv R \setminus L$: The transition must be by concert res from Figure 4.5. Assume P is consistent and $P \xrightarrow{\{\mu[k],\underline{\nu}[l]\}} P'$. Since $P \xrightarrow{\{\mu[k],\underline{\nu}[l]\}} P'$ by rule concert res we obtain $R \xrightarrow{\{\mu[k],\underline{\nu}[l]\}} R'$ and $\mu, \nu \notin L \cup (L)$. Since $k \notin \mathsf{keys}(R)$ by the inductive hypothesis, it follows that $k \notin \mathsf{keys}(R \setminus L)$ since restriction does not change the keys of the process by definition of keys. Since $l \in \mathsf{keys}(R)$ by the inductive hypothesis, it follows that $l \in \mathsf{keys}(R \setminus L)$. Also $\mathsf{keys}(P) = \mathsf{keys}(R)$ and $\mathsf{keys}(P') = \mathsf{keys}(R')$ according to the definition of keys. Since by the inductive hypothesis $\mathsf{keys}(R') = \mathsf{keys}(R) \cup \{k\}$ we can calculate $\mathsf{keys}(P')$: $\mathsf{keys}(P') = \mathsf{keys}(R' \setminus L) = \mathsf{keys}(R') = (\mathsf{keys}(R) \cup \{k\}) \setminus \{l\} = (\mathsf{keys}(P) \cup \{k\}) \setminus \{l\}$ as required.

(d) There are no transitions by concerted actions for $P \equiv (s; c).R$ and $P \equiv S$ with $S \stackrel{def}{=} R$.

The property corresponding to part 3 of Proposition 4.1, namely $P \xrightarrow{\{\mu[k],\underline{\nu}[l]\}} P'$ if and only if $P' \xrightarrow{\{\nu[l],\underline{\mu}[k]\}} \}P$ does not hold in general but only in certain circumstances, which we now describe. Consider $(a[k];b).Q \mid R$ and c, d, for any Q, R, such that $\gamma(a,c) = d = \gamma(b,c)$ with $R \xrightarrow{c[l]} R'$ and $R' \xrightarrow{\underline{c}[k]} R''$. We obtain, by concert and prom rules,

$$(a[k];b).Q \mid R \xrightarrow{\{d[l],\underline{d}[k]\}} (a;b[l]).Q \mid R'' \Rightarrow (a[l];b).Q \mid R''$$

Since $R'' \xrightarrow{c[k]} R' \xrightarrow{\underline{c}[l]} R$, we get, again by concert and prom rules

$$(a[l];b).Q \mid R'' \xrightarrow{\{d[k],\underline{d}[l]\}} (a;b[l]).Q \mid R \Rightarrow (a[k];b).Q \mid R$$

Assume that R is $(c, c[k]).R_1$ for some R_1. If we take S as $(a[l];b).Q \mid R''$, then the following result could be seen as corresponding to part 3 of Proposition 4.1:

Proposition 4.5. *Consider $(a[k];b).Q$ for any Q and c, d such that $\gamma(a,c) = d = \gamma(b,c)$. There exist R, S and l such that $(a[k];b).Q \mid R \xrightarrow{\{d[l],\underline{d}[k]\}} S$ if and only if $S \xrightarrow{\{d[k],\underline{d}[l]\}} (a[k];b).Q \mid R$.*

4.3 CCB without weak actions

In this section we discuss the main properties of the sub-calculus of CCB that uses no weak actions. Our prefix operator is thus $(s).P$, where s contains only strong actions. We call this calculus CCB_s. Its SOS rules are as for CCB except that the rules in Figure 4.5 do not apply as there are no weak actions. The congruence rules prom, move-r and move-l also do not apply since there are no weak actions. We shall use μ, ν to denote strong actions in this section.

We shall also consider the forward-only version of CCB_s called CCB_f. The syntax of CCB_f is

$$P ::= S | S \stackrel{def}{=} P \;\Big|\; (s).P \;\Big|\; P|Q \;\Big|\; P\backslash L$$

and the SOS rules are given in Figure 4.12; we also have the reduction rules from Figure 4.7 (without prom, move-r and move-l) which, together with rules in Figure 4.12, generate the transition relation $\rightarrow_f$ for CCB_f. Note that we do not record past actions (act_f rule); hence CCB_f is similar to the core of ACP. We note that CCB_s is different from CCSK [94, 90] as it uses multiset prefixing (very much like in [27, 93]) and ACP-like communication. Alternatively, we could have derived the SOS rules of CCB_f from those of CCB_s by applying a transformation which stripes from every rule all keys in the labels and all past actions in the processes, using the pruning map π defined in the next paragraph. Together with the removal of

$$\mathsf{act}_f\ \frac{}{(s,a,s').P \xrightarrow{a}_f (s,s').P} \qquad \mathsf{par}_f\ \frac{P \xrightarrow{a}_f P'}{P \mid Q \xrightarrow{a}_f P' \mid Q}$$

$$\mathsf{com}_f\ \frac{P \xrightarrow{a}_f P' \quad Q \xrightarrow{b}_f Q'}{P \mid Q \xrightarrow{c}_f P' \mid Q'}(*) \qquad \mathsf{res}_f\ \frac{P \xrightarrow{a}_f P'}{P\backslash L \xrightarrow{a}_f P'\backslash L}\ a \notin L$$

$$\mathsf{con}_f\ \frac{P \xrightarrow{a}_f P'}{S \xrightarrow{a}_f P'}\ S \stackrel{def}{=} P \qquad \mathsf{sc}\ \frac{P \Rightarrow^* Q \quad Q \xrightarrow{a}_f Q' \quad Q' \Rightarrow^* P'}{P \xrightarrow{a}_f P'}$$

Figure 4.12: SOS rules for CCB_f. We have $a, b, c \in \mathcal{SA}$ and $(*)$ is $\gamma(a,b) = c$.

the semicolons and the weak actions that transforms CCB rules into CCB_f rules. Rule act1 from Figure 4.2 in this way becomes rule act_f. Rule act2 from Figure 4.2 is no longer relevant since we have no past actions and therefore no sets of purely past actions. Rules par, com, res, and con from Figure 4.2 are transformed into the respective rules of CCB_f. Rule sc is identical except that in CCB_f it applies to transitions from CCB_f.

We show firstly that $\rightarrow$ for CCB_s is essentially conservative over $\rightarrow_f$. A process of CCB_s is converted to a CCB_f process by "pruning" past actions:

Definition 4.6. The pruning map $\pi : \mathsf{Proc}_{\mathrm{CCB}_s} \rightarrow \mathsf{Proc}_{\mathrm{CCB}_f}$ is defined as follows, where $t \in \mathcal{AK}^*$ and $s, s' \in \mathcal{A}^*$:

$$\begin{array}{lll} \pi(\mathbf{0}) = \mathbf{0} & \pi((s,t,s').P) = (s,s').\pi(P) & \pi((t).P) = \pi(P) \\ \pi(P \mid Q) = \pi(P) \mid \pi(Q) & \pi(P \setminus L) = \pi(P) \setminus L & \pi(S) = \pi(P) \text{ if } S \stackrel{def}{=} P \end{array}$$

Theorem 4.1 (Conservation)**.** *Let* $P \in \mathsf{Proc}_{\mathrm{CCB}_s}$.
1. If $P \xrightarrow{\mu[k]} Q$ *then* $\pi(P) \xrightarrow{\mu}_f \pi(Q)$.
2. If $\pi(P) \xrightarrow{\mu}_f Q$, *then for any* $k \in \mathcal{K}\backslash\mathsf{keys}(P)$ *there is* Q' *such that* $P \xrightarrow{\mu[k]} Q'$ *and* $\pi(Q') = Q$.

Proof. 1. We use induction on the depth of the inference tree of $P \xrightarrow{\mu[k]} Q$.

(a) Base case: obvious.

(b) Inductive hypothesis: We assume that for all subprocesses R of P and all $\nu[l]$, if R is a consistent process and if $R \xrightarrow{\nu[l]} R'$ for some R' then $\pi(R) \xrightarrow{\nu}_f \pi(R')$.

(c) Induction step: We consider cases depending on the structure of P. Assume that P is consistent and $P \xrightarrow{\mu[k]} P'$ in all cases.

i. $P \equiv (s;b).R$: There are two cases:

A. $P \xrightarrow{\mu[k]} P'$ by act1 in Figure 4.2: Assume $s = \mu : s', t$ where $s' \in \mathcal{A}^*$ and $t \in \mathcal{A}\mathcal{K}^*$. Then $(\mu : s', t; b).R \xrightarrow{\mu[k]} (\mu[k] : s', t; b).R$. We can apply π to the processes: $\pi((\mu : s', t; b).R) = (\mu : s').R = \pi(P)$ and $\pi((\mu[k] : s', t; b).R) = (s').R = \pi(P')$. The transition $(\mu : s').R \xrightarrow{\mu}_f (s').R$ is by act1 and we get $\pi(P) \xrightarrow{\mu}_f \pi(P')$.

B. $P \xrightarrow{\mu[k]} P'$ by act2 in Figure 4.2: We deduce that $P \equiv (t; b).R$, $\mathsf{fsh}[k](t)$ and $P' \equiv (t; b).R'$. Recall that t contains only past actions. The transition must be $(t; b).R \xrightarrow{\mu[k]} (t; b).R'$, and we get by act2 $R \xrightarrow{\mu[k]} R'$ and $\mathsf{fsh}[k](t)$. Applying π to the processes we get $\pi((t; b).R) = \pi(R)$ and $\pi((t; b).R') = \pi(R')$. Since $R \xrightarrow{\mu[k]} R'$, by the inductive hypothesis, we obtain $\pi(R) \xrightarrow{\mu}_f \pi(R')$. The last implies $\pi((t; b).R) \xrightarrow{\mu}_f \pi((t; b).R')$ which is $\pi(P) \xrightarrow{\mu} \pi(P')$.

ii. $P \equiv R \mid Q$: There are two cases:

A. This happens by par in Figure 4.2. Since $P \xrightarrow{\mu[k]} P'$, by rule par, $R \xrightarrow{\mu[k]} R'$ and $\mathsf{fsh}[k](Q)$ must hold. By the inductive hypothesis $\pi(R) \xrightarrow{\mu}_f \pi(R')$ is true. By rule par $\pi(R) \mid \pi(Q) \xrightarrow{\mu}_f \pi(R') \mid \pi(Q)$ holds, which means $\pi(R \mid Q) \xrightarrow{\mu}_f \pi(R' \mid Q)$ and $\pi(P) \xrightarrow{\mu}_S \pi(P')$ as required.

B. This is by com in Figure 4.2. The inductive hypothesis in this case is not only true about R, but also about Q, so that if Q is consistent and $Q \xrightarrow{\mu_1[k]} Q'$ then $\pi(Q) \xrightarrow{\mu_1}_f \pi(Q')$. Assume P is consistent, $P \xrightarrow{\mu[k]} P'$ and $\gamma(\mu_1, \mu_2) = \mu$. We can calculate $\pi(P) = \pi(R \mid Q) = \pi(R) \mid \pi(Q)$ and $\pi(P') = \pi(R' \mid Q') = \pi(R') \mid \pi(Q')$. Since $\pi(R) \xrightarrow{\mu}_f \pi(R')$ and $\pi(Q) \xrightarrow{\mu}_f \pi(Q')$ it follows that according to rule com $\pi(R) \mid \pi(Q) \xrightarrow{\mu}_f \pi(R') \mid \pi(Q')$ which means $\pi(P) \xrightarrow{\mu}_f \pi(P')$ as required.

iii. Cases for $P \equiv R \setminus L$ and $P \equiv S$ with $S \stackrel{def}{=} R$ are straightforward.

2. We prove Theorem 4.1.2 by using induction on the definition of $\pi(P)$, which means on the structure of P.

(a) Base case: obvious.

(b) Inductive hypothesis: We assume that for all subprocesses R of P and all ν, if $\pi(R) \xrightarrow{\nu}_f R'$, then for any $l \in \mathcal{K} \setminus \mathsf{keys}(R)$ there is R'' such that $R \xrightarrow{\nu[l]} R''$ and $\pi(R'') = R'$.

(c) Induction step: We show the result for $\pi(P)$ as well. we consider cases depending on the structure of P. Assume that P is consistent and $\pi(P) \xrightarrow{\mu}_f Q$ for some Q in all cases.

i. $P \equiv (s, s'; b)$ and $\pi((s, s'; b).R) = (s').\pi(R)$: Here rule act_f applies with s' being of the structure ν, s''. The transition is $(\nu, s'').R \xrightarrow{\nu}_f (s'').R)$. Then with $k \in \mathcal{K} \backslash \mathsf{keys}(P)$ we have $(\nu, s'').R \xrightarrow{\nu[k]} (\nu[k], s'').R$ and we let $Q' \equiv (\nu[k], s'').R$ with $\pi(Q') \equiv (s'').R$.

ii. $P \equiv (t; b).R$ and $\pi((t; b).R) = \pi(R)$: Here the transition is in R by $\pi(R) \xrightarrow{\nu}_f R'$. By the inductive hypothesis $R \xrightarrow{\mu[k]} R''$ and $\pi(R'') = R'$ for some R''. We deduce that $\pi(P) \xrightarrow{\nu}_f R'$ and $P \xrightarrow{\mu[k]} R''$. Let $Q' \equiv R''$ with the required properties.

iii. $P \equiv R \mid T$ and $\pi(P) = \pi(R \mid T)$: There are two cases:

A. The transition is by par_f. Assume $\pi(R \mid T) \xrightarrow{\mu}_f Q$ for some Q. So $\pi(R \mid T) \xrightarrow{\mu}_f Q$ implies without loss of generality $\pi(R) \xrightarrow{\mu}_f R'$ for some R' and $Q \equiv R' \mid \pi(T)$ since $\pi(P) = \pi(R \mid T) = \pi(R) \mid \pi(T)$. Since $\pi(R) \xrightarrow{\mu}_f R'$ by the inductive hypothesis there exists a U and a $k \in \mathcal{K} \setminus (\mathsf{keys}(R) \cup \mathsf{keys}(T))$ such that $R \xrightarrow{\mu[k]} U$ and $\pi(U) = R'$. Since $R \xrightarrow{\mu[k]} U$ and since $\mathsf{fsh}[k](T)$ we obtain by rule par $R \mid T \xrightarrow{\mu[k]} U \mid T$. Now we calculate $\pi(U \mid T) = \pi(U) \mid \pi(T) = R' \mid \pi(T) = Q$. Let $Q' \equiv U \mid T$ with the required properties.

B. The transition is by com_f. Since $\pi(P) \xrightarrow{\mu} P'$, by rule com_f, $R \xrightarrow{\nu_1} R'$, $T \xrightarrow{\nu_1} T'$ and $\gamma(\nu_1, \nu_2) = \mu$ for some R', T' must hold without loss of generality. The inductive hypothesis in this case is not only true about R, so there is an R'' such that $R \xrightarrow{\nu_1[k]} R''$ and $\pi(R'') = R'$ and a T'' such that $T \xrightarrow{\nu_2[k]} T''$ and $\pi(T'') = T'$. We can deduce that $R \mid T \xrightarrow{\mu[k]} R'' \mid T''$. Also $\pi(R'' \mid T'') = \pi(R'') \mid \pi(T'') = R' \mid T' = P'$. Let $Q' \equiv R'' \mid T''$. This has the required properties.

iv. Cases for $P \equiv R \setminus L$ and $P \equiv S$ with $S \stackrel{def}{=} R$ are straightforward.

We now consider the causal consistency property, first defined and discussed in [26, 66], for CCB_s. We define when a set of keys is a cause for another key:

Definition 4.7. The set of causes of a key k in P is defined as follows:

$$cau(\mathbf{0}, k) = \emptyset \qquad cau(P \backslash L, k) = cau(P, k)$$

$$cau((s).P, k) = \mathsf{k}(s) \cup cau(P, k) \text{ if } k \in \mathsf{keys}(P) \qquad cau((\mu[k] : s).P, k) = \emptyset$$

$$cau((s).P, k) = \emptyset \text{ if } k \notin \mathsf{keys}(P) \qquad cau(S) = cau(P) \text{ if } S \stackrel{def}{=} P$$

$$cau(P \mid Q, k) = cau(P, k) \cup cau(Q, k)$$

If one of two coinitial transitions is forward and the other reverse, either they are concurrent (by Defintion 4.5) or the forward transition depends causally on the reverse one.

Proposition 4.6. *If $t_1 \equiv P \xrightarrow{\underline{\mu}[k]} P'$ and $t_2 \equiv P \xrightarrow{\nu[l]} P''$, then either t_1 and t_2 are concurrent or $k \in cau(P'', l)$.*

Proof. By induction on the depth of inference trees for transition of P.

1. Base case: Obvious.

2. Inductive hypothesis: We assume that for all subprocesses R of P and all $\underline{c}[m], d[n]$, if R is a consistent process, $t_1 \equiv R \xrightarrow{\underline{c}[m]} R'$ and $t_2 \equiv R \xrightarrow{d[n]} R''$ then either t_1 and t_2 are concurrent or $m \in cau(P'', n)$.

3. Induction step: We show Proposition 4.6 for P. By Definition 4.5 there is either an M such that $M \neq P$, $P' \xrightarrow{\nu[l]} M$ and $P'' \xrightarrow{\underline{\mu}[k]} M$, or $k \in cau(P'', l)$ holds. We consider cases based on the structure of P:

   (a) $P \equiv (s; b).R$ with s containing at least one past action and at least one fresh action. The transitions t_1 and t_2 are by rev act1 respectively act1. Since there is nothing in the rules giving any of them precedence, the transitions are concurrent as required. With s' being the sequence obtained from s by removing actions ν and $\mu[k]$ we have $t_1 \equiv (s', \nu, \mu[k]; b).R \xrightarrow{\underline{\mu}[k]} (s', \nu, \mu, ; b).R$ and $t_2 \equiv (s', \nu, \mu[k]; b).R \xrightarrow{\nu[l]} (s', \nu[l], \mu[k]; b).R$. We now deduce $M \equiv (s', \nu[l], \mu; b).R$, and the properties required for ν and $\mu[k]$ being concurrent hold: namely $(s', \nu, \mu, ; b).R \xrightarrow{\nu[l]} M$ and $(s', \nu[l], \mu[k]; b).R \xrightarrow{\underline{\mu}[k]} M$.

   (b) $P \equiv (s; b).R$ where s contains only fresh actions: We cannot have the required transition t_2. Hence Proposition 4.6 is vacuously valid.

   (c) $P \equiv (s; b).R$ where s contains only past actions: There is $R \xrightarrow{\nu[l]} R'$, and a $\mu[k] \in s$ so that $t_2 \equiv (s; b).R \xrightarrow{\nu[l]} (s; b).R'$ and $t_1 \equiv (s; b).R \xrightarrow{\underline{\mu}[k]} (s'; b).R$ for some s'. These transitions are not concurrent since doing any action in R prevents undoing of actions in s and vice versa. Hence $k \in cau(P'', l)$ holds with $P'' \equiv (s; b).R'$.

   (d) $P \equiv Q \mid R$: There are three cases:

      i. t_1 by rule rev par and t_2 by rule par. There are two sub-cases:

         A. Transitions in the same subprocess: Assume without loss of generality $Q \xrightarrow{\underline{\mu}[k]} Q'$ and $Q \xrightarrow{\nu[l]} Q''$. By the inductive hypothesis these transitions are either concurrent or $k \in cau(Q'', l)$.

- concurrent transitions: By the inductive hypothesis there is an N so that $Q' \xrightarrow{\nu[l]} N$ and $Q'' \xrightarrow{\underline{mu}[k]} N$. We can conclude by using rule rev par that $Q' \mid R \xrightarrow{\nu[l]} N \mid R$ and $Q'' \mid R \xrightarrow{\underline{\mu}[k]} N \mid R$. Letting $M \equiv N \mid R$ we get the required result.
- $k \in cau(Q'', l)$: Since $cau(P \mid Q, k) = cau(P, k) \cup cau(Q, k)$ according to Definition 4.7 we get $cau(Q'' \mid R, l) = cau(Q'', l) \cup cau(R, l)$. If $k \in cau(Q'', l)$ then $k \in cau(Q'', l) \cup cau(R, l)$ must be true as well and, hence, $k \in cau(Q'' \mid R)$. Since $P'' \equiv Q'' \mid R$ it follows that $k \in cau(P'', l)$ as required.

B. Transitions in different subprocesses: Assume without loss of generality that $Q \xrightarrow{\underline{\mu}[k]} Q'$ and $R \xrightarrow{\nu[l]} R'$. These transitions are concurrent. By rule rev par $Q \mid R \xrightarrow{\underline{\mu}[k]} Q' \mid R \xrightarrow{\nu[l]} Q' \mid R'$ and $Q \mid R \xrightarrow{\nu[l]} Q \mid R' \xrightarrow{\underline{\mu}[k]} Q' \mid R'$ are valid. These form the diamond required for concurrent transitions with $M \equiv Q' \mid R'$.

ii. $P \xrightarrow{\underline{\mu}[k]} P'$ by rule rev com and $P \xrightarrow{\nu[l]} P''$ by rule par: Without loss of generality this covers all cases with one par and one rev com or one rev par and one com transition. We assume that $P \xrightarrow{\underline{\mu}[k]} P'$ happens by rule rev com, where $\gamma(\mu_1, \mu_2) = \mu$, and that $P \xrightarrow{\nu[l]} P''$ happens by rule par. We also assume that ν happens in Q, so that $Q \xrightarrow{\nu[l]} Q'$, and that $Q \xrightarrow{\underline{\mu_1}[k]} Q''$ and $R \xrightarrow{\underline{\mu_2}[k]} R'$. We know that $\mathsf{fsh}[l](R)$ because of the preconditions of the par rule and that $l \neq k$ because $\mathsf{fsh}[l](R)$ and $R \xrightarrow{\underline{\mu_2}[k]} R''$, a transition which could not happen if $l = k$, since according to Proposition 4.1.2 a key cannot be fresh for a reverse transition to happen with this key. By the inductive hypothesis transition $Q \xrightarrow{\nu[l]} Q'$ and $Q \xrightarrow{\underline{\mu_1}[k]} Q''$ are either concurrent or $k \in cau(Q', l)$.

A. concurrent transitions: By the inductive hypothesis there is an N so that $Q' \xrightarrow{\underline{\mu_1}[k]} N$ and $Q'' \xrightarrow{\nu[l]} N$. Using the rev com rule we can deduce that $P \xrightarrow{\underline{\mu}[k]} Q'' \mid R'$, $P \xrightarrow{\nu[l]} Q' \mid R$, $Q'' \mid R' \xrightarrow{\nu[l]} N \mid R'$ and $Q' \mid R \xrightarrow{\underline{\mu}[k]} N \mid R'$. Letting $M \equiv N \mid R'$ we obtain the result.

B. $k \in cau(Q', l)$: $k \in cau(Q', l)$ implies $k \in (cau(Q', l) \cup cau(R, l))$ according to Definition 4.7, which is $k \in cau(Q' \mid R, l)$. Since $P'' = Q' \mid R$ we have $k \in cau(P'', l)$ as required.

iii. $P \xrightarrow{\underline{\mu}[k]} P'$ by rev com and $P \xrightarrow{\nu[l]} P''$ by com: Without loss of generality this covers all cases with one com and one rev com transition. We assume that $\gamma(\mu_1, \mu_2) = \mu$ and $\gamma(\nu_1, \nu_2) = \nu$. Also $Q \xrightarrow{\underline{\mu_1}[k]} Q'$, $Q \xrightarrow{\nu_1[l]} Q''$, $R \xrightarrow{\underline{\mu_2}[k]} R'$ and $R \xrightarrow{\nu_2[l]} R''$. By the inductive hypothesis

the transitions μ_1 and ν_1 must be either concurrent or $k \in cau(Q'', l)$. For transitions μ_2 and ν_2 the same holds. So we distinguish three cases:

A. two concurrent transitions: Since $Q \xrightarrow{\underline{\mu_1}[k]} Q'$ and $Q \xrightarrow{\nu_1[l]} Q''$ by the inductive hypothesis it follows that there is an N so that $Q' \xrightarrow{\nu_1[l]} N$, and $Q'' \xrightarrow{\underline{\mu_1}[k]} N$ and since $R \xrightarrow{\underline{\mu_2}[k]} R'$ and $R \xrightarrow{\nu_2[l]} R''$ there is an N' so that $R' \xrightarrow{\nu_2[l]} N'$ and $R'' \xrightarrow{\underline{\mu_2}[k]} N'$. By rev par and par it follows that $P \xrightarrow{\underline{\mu}[k]} Q' \mid R' \xrightarrow{\nu[l]} N \mid N'$ and $P \xrightarrow{\nu[l]} Q'' \mid R'' \xrightarrow{\mu[k]} N \mid N'$. Letting $M \equiv N \mid N'$ gives the result.

B. $k \in cau(Q'', l)$ and $k \in cau(R'', l)$: Since $P'' \equiv Q'' \mid R''$ we can calculate $cau(P'', l) = cau(Q'' \mid R'', l) = cau(Q'', l) \cup cau(R'', l)$. Since $k \in cau(Q'', l)$ and $k \in cau(R'', l)$ it follows that $k \in cau(Q'', l) \cup cau(R'', l)$ and that $k \in cau(P'', l)$ as required.

C. $k \in cau(Q'', l)$, and $R \xrightarrow{\underline{\mu_2}[k]} R'$ and $R \xrightarrow{\nu_2[l]} R''$ are concurrent: Since $P'' \equiv Q'' \mid R''$ we get $cau(P'', l) = cau(Q'' \mid R'', l) = cau(Q'', l) \cup cau(R'', l)$. Since $k \in cau(Q'', l)$ it follows that $k \in cau(Q'', l) \cup cau(R'', l)$ and that $k \in cau(P'', l)$ as required. This also applies when $k \in cau(R'', l)$, so $Q \xrightarrow{\underline{\mu_1}[k]} Q'$ and $Q \xrightarrow{\nu_1[l]} Q''$ are concurrent.

(e) $P \equiv R \setminus L$: The transitions $R \setminus L \xrightarrow{\underline{\mu}[k]} R' \setminus L$ and $R \setminus L \xrightarrow{\nu[l]} R'' \setminus L$ are by rule rev res in Figure 4.4 respectively res in Figure 4.2. We assume without loss of generality that $R \xrightarrow{\underline{\mu}[k]} R'$, $R \xrightarrow{\nu[l]} R''$ and $\mu, \nu \notin L$. By the inductive hypothesis $R \xrightarrow{\underline{\mu}[k]} R'$ and $R \xrightarrow{\nu[l]} R''$ are either concurrent or $k \in cau(R'', l)$.

i. concurrent transitions: By the inductive hypothesis there is an N so that $R' \xrightarrow{\underline{\mu}[l]} N$ and $R'' \xrightarrow{\nu[k]} N$. Consider $M \equiv N \setminus L$. Since $\mu, \nu \notin L$ by rule rev res respectively res we deduce $P \xrightarrow{\underline{\mu}[k]} R' \setminus L \xrightarrow{\nu[l]} M$ and $P \xrightarrow{\nu[l]} R'' \setminus L \xrightarrow{\underline{\mu}[k]} M$.

ii. $k \in cau(R'', l)$: Since $cau(P \setminus L, k) = cau(P, k)$ according to Definition 4.7 we calculate $cau(R'' \setminus L, l) = cau(R'', l)$. If $k \in cau(R'', l)$ it follows that $k \in cau(R'' \setminus L, l)$ as required.

(f) $P \equiv S$ with $S \stackrel{def}{=} R$: similar to the $P \equiv R \setminus L$ case.

Next we introduce further notation on *traces*. We denote the reverse transition corresponding to a forward transition t (and the forward transition corresponding to a reverse transition t) as $t^\bullet$. Similarly to denoting the reverse transitions by $^\bullet$,

we denote the reverse trace of σ as $\sigma^\bullet$. The empty trace with a process P is written as ϵ_P, and when the process is the source or the target of the transition t, we write $\epsilon_{source(t)}$ respectively $\epsilon_{target(t)}$. Similarly as with forward and reverse transitions, we shall use *forward traces* (traces composed of forward transitions only) and *reverse traces* (traces composed of reverse transitions only).

We can now define causal equivalence between traces.

Definition 4.8. *Causally equivalent* traces are defined by the least equivalence relation $\asymp$ which is closed under composition and obeys the following rules, where t_1 is $P \xmapsto{a[k]} Q$, t_2 is $P \xmapsto{b[l]} R$, d_1 is $Q \xmapsto{b[l]} S$ and d_2 is $R \xmapsto{a[k]} S$:

$$t_1; d_1 \asymp t_2; d_2 \quad t; t^\bullet \asymp \epsilon_{source(t)} \quad t^\bullet; t \asymp \epsilon_{target(t)}$$

The first relation states that the concurrent transitions t_1 and t_2 are causally independent, hence they can happen in any order. The trace $t_1; d_1$ forms a diamond with $t_2; d_2$, so the traces are causally equivalent. The remaining relations state that doing a transition and its reverse version is the same as doing nothing.

The next two results are needed to prove causal consistency for CCB_s; they follow closely [26, 66]. The first states that any computation has a causally equivalent version in which we first compute in reverse for a while and then we only compute forwards. The second result says that a trace which has a forward-only coinitial and cofinal and causally equivalent trace can always be shortened to a forward-only trace.

Proposition 4.7 (Rearrangement)**.** *If σ is a trace then there exist forward traces σ_1 and σ_2 such that $\sigma \asymp \sigma_1^\bullet; \sigma_2$.*

Proof. We give a constructive proof. We show that any trace can be transformed to the form required by Proposition4.7. Any trace must be either σ, $\sigma^\bullet$ or $\sigma_1^\bullet; \sigma_2$ or it must be of the form $\sigma_1^\bullet; \sigma_2^*; t_1; t_2^\bullet; \sigma_3$ where σ, σ_1 and σ_2 are forward traces and σ_3 is composed of any number of forward and reverse transitions. In other words, this means that we can identify the earliest pair of forward-reverse transitions. The instructions for the transformation to the required form are in the algorithm in Figure 4.13.

We show that the algorithm terminates in all cases with the required result. Executing the inner while loop (lines 9 to 13) once decreases the length of the resulting σ_2 by 1 and increases the length of the resulting σ_3 by 1.

After the inner while loop terminates we have $length(\sigma_2') = length(\sigma_2)$. The trace $\sigma_1^\bullet; t^\bullet$ forms a new reverse only trace whose length is increased by 1 compared to $\sigma_1^\bullet$.

1 Let σ_{input} be our trace

2 *while* ($\sigma_{input} \neq \sigma \wedge \sigma_{input} \neq \sigma^{\bullet} \wedge \sigma_{input} \neq \sigma_1^{\bullet}; \sigma_2$ for all forward traces $\sigma, \sigma_1, \sigma_2$)

3 σ_{input} is of the form $\sigma_1^{\bullet}; \sigma_2; t_1; t_2^{\bullet}; \sigma_3$ for some (possibly new) $\sigma_1, \sigma_2, t_1, t_2, \sigma_3$, where $\sigma_1, \sigma_2, \sigma_3$ are forward traces, t_1, t_2 are forward transitions (any of $\sigma_1, \sigma_2, \sigma_3$ could be empty sequences ϵ)

Let $length(\sigma_1) = n$, $length(\sigma_2) = k$, $length(\sigma_3) = l$. Distance from start to the pair $t_1; t_2^{\bullet}$ is $n + k$

4 Let $t_1 \equiv P \xrightarrow{\mu[m]} P'$, $t_2^{\bullet} \equiv P' \xrightarrow{\underline{\nu}[n]} P''$ (so $t_2 \equiv P'' \xrightarrow{\nu[n]} P'$)

5 *if* ($\mu[m] = \nu[n]$) *then*

6 Since $\mu[m] = \nu[k]$ we get $P \equiv P''$ and, by Definition 4.8, transitions $t_1; t_2^{\bullet}$ are replaced by ϵ in σ_{input}

7 σ_{input} is now $\sigma_1^{\bullet}; \sigma_2; \sigma_3$

8 *else*

9 *while* ($\sigma_2 \neq \epsilon$)

10 σ_{input} is $\sigma_1^{\bullet}; \sigma_2; t_1; t_2^{\bullet}; \sigma_3$ for some (possibly new) σ_2, t_1, t_2 and σ_3, with σ_1 and σ_2 being forward traces and t_1 and t_2 being forward transitions

11 According to Proposition 4.1.3 there must be a transition $t_1^{\bullet} \equiv P' \xrightarrow{\underline{\mu}[m]} P$. $t_1^{\bullet}$ and $t_2^{\bullet}$ form a diamond with two transitions $t_3^{\bullet} \equiv P \xrightarrow{\underline{\nu}[n]} M$ and $t_4^{\bullet} \equiv P'' \xrightarrow{\underline{\mu}[m]} M$ for some M according to Proposition 4.2. According to Proposition 4.1.3 there must be a transition $t_4 \equiv M \xrightarrow{\mu[m]} P''$. The trace $t_3^{\bullet}; t_4$ replaces $t_1; t_2^{\bullet}$ in σ_{input} since the traces are coinitial and cofinal

12 σ_{input} is now $\sigma_1^{\bullet}; \sigma_2; t_3^{\bullet}; t_4; \sigma_3$

13 *end while* (at this point σ_2 is ϵ)

14 σ_{input} is now $\sigma_1^{\bullet}; t^{\bullet}; \sigma_2'; \sigma_3$ for some t, σ_2' where σ_2' is forwards only

15 *end if*

16 *end while* (at this point $\sigma_{input} = \sigma \vee \sigma_{input} = \sigma^{\bullet} \vee \sigma_{input} = \sigma_1^{\bullet}; \sigma_2$, for some forward traces σ, σ_1 and σ_2)

Figure 4.13: Rearrangement algorithm.

Executing the outer while loop once decreases the length of the sequence $t_2^\bullet;\sigma_3$, which is the "unprocessed" part of $\sigma_1^\bullet;\sigma_2;t_1;t_2^\bullet;\sigma_3$, and changes the structure of σ_{input} so it is no longer as required by Proposition 4.7. The sequence $\sigma_1^\bullet;\sigma_2;t_1$ is of the form required by Proposition 4.7 and its length is increased by the outer while loop. This is because at the end of the outer while loop we have $\sigma_1^\bullet;t^\bullet;\sigma_2';\sigma_3$. The part $\sigma_1^\bullet;t^\bullet;\sigma_2'$ of this trace is in the correct form and its length is increased by 1, since $t^\bullet$ is added, $\sigma_1^\bullet$ is unchanged, and $length(\sigma_2') = length(\sigma_2)$. The sequence σ_3 is the "unprocessed" part, which has been shortened by one transition, in comparison to $t_2^\bullet;\sigma_3$ since σ_3 is unchanged. Hence the algorithm terminates once σ_3 has been completely processed and the final σ_{input} has the required form with σ_3 being empty.

Proposition 4.8 (Shortening)**.** *If σ_1 and σ_2 are coinitial and cofinal traces, with σ_2 a forward trace, then there exists a forward trace σ_1' of length at most that of σ_1 such that $\sigma_1' \asymp \sigma_2$.*

Proof. By induction on the length of σ_1. If σ_1 is a forward trace then the proposition holds. If not, then by Proposition 4.7 we assume σ_1 to be $\sigma'^\bullet;\sigma$ for some forward sequences σ and σ'. There is only one sub-trace $t_1^\bullet;t_2$ in $\sigma'^\bullet;\sigma$ where the first transition $t_1^\bullet$ is reverse and the second transition t_2 is forward. We assume $t_1^\bullet \equiv P \xrightarrow{\underline{\mu}[k]} P'$ and $t_2 \equiv P' \xrightarrow{\nu[l]} P''$. There is a transition $t' \equiv R \xrightarrow{\mu[k]} R'$ in σ_1 for some R, R', otherwise σ_1 could not be cofinal with σ_2. Proposition 4.1.3 implies that there is a transition $t_1 \equiv P' \xrightarrow{\mu[k]} P$. Transitions t_1 and t_2 are either concurrent, in conflict, or $l \in cau(P,k)$ or $k \in cau(P'',l)$. The possibility of t_1 and t_2 being in conflict is excluded since we have t_1 and t_2 in a valid trace, namely $P' \xrightarrow{\nu[l]} P'' \rightarrow^* R \xrightarrow{\mu[k]} R'$. Also, $k \in cau(P'',l)$ is impossible since we perform $\nu[l]$ before $\mu[k]$ in the trace $P \xrightarrow{\underline{\mu}[k]} P' \xrightarrow{\nu[l]} P'' \rightarrow^* R \xrightarrow{\mu[k]} R'$. Finally, $l \in cau(P,k)$ cannot hold because otherwise, since $P' \xrightarrow{\mu[k]} P$, some $\alpha[l]$ transition must have happened in P' before $P' \xrightarrow{\mu[k]} P$. This contradicts $P' \xrightarrow{\nu[l]} P''$ (key l is not fresh in P' due to the $\alpha[l]$ action, hence $P' \xrightarrow{\nu[l]} P'''$ is not possible for any P'''). Hence $l \notin cau(P,k)$ and, overall, the only possibility is that t_1 and t_2 are concurrent.

The transitions $t_1^\bullet$ and t_2 form a diamond with two transitions $t_3 \equiv P \xrightarrow{\nu[l]} P'''$ and $t_4^\bullet \equiv P''' \xrightarrow{\underline{\mu}[k]} P''$ for some P''' according to Proposition 4.2. The trace $t_3;t_4^\bullet$ can replace $t_1^\bullet;t_2$ in $\sigma'^\bullet;\sigma$ since the traces are coinitial and cofinal. We can repeat this process of "moving" an equivalent version of $t_4^\bullet$ to the right (by following the steps described above) until the resulting $t^\bullet$ (with the label $\underline{\mu}[k]$) is directly to the left of $t' \equiv R \xrightarrow{\mu[k]} R'$. These transitions are then $Q \xrightarrow{\underline{\mu}[k]} R \xrightarrow{\mu[k]} R'$ for some Q (where $Q = R$). Using Definition 4.8 we can remove them. The resulting trace is shorter than σ_1 and we can repeat the process until the trace is forwards only.

Next we have the second important result for CCB_s.

Theorem 4.2 (Causal consistency)**.** *Let σ_1 and σ_2 be traces. Then $\sigma_1 \asymp \sigma_2$ if and only if σ_1 and σ_2 are coinitial and cofinal.*

Proof. The statement, if $\sigma_1 \asymp \sigma_2$ then σ_1 and σ_2 are coinitial and cofinal, follows directly from the construction of $\asymp$ in Definition 4.8.

Next, we show that if σ_1 and σ_2 are coinitial and cofinal then $\sigma_1 \asymp \sigma_2$. We use induction on the sum of the lengths of σ_1 and σ_2 and on the distance between the end of σ_1 and the earliest pair of transitions t_1 in σ_1 and t_2 in σ_2 which are not equal. The coinitial and cofinal traces σ_1 and σ_2 are rewritten as compositions of reverse and forward traces by Proposition 4.7. There are three cases:

1. t_1 is forward and t_2 is reverse: We can assume that $\sigma_1 = \sigma^\bullet; t_1; \sigma'$ and $\sigma_2 = \sigma^\bullet; t_2^\bullet; \sigma''$ by Proposition 4.7. Since t_1 is forward $t_1; \sigma'$ must be forward, whereas $t_2^\bullet; \sigma''$ must start with at least one reverse transition. Since σ_1 and σ_2 are coinitial and cofinal $t_1; \sigma'$ and $t_2^\bullet; \sigma''$ are also coinitial and cofinal. We notice that the assumption of Proposition 4.8 is valid for $t_1; \sigma'$ and $t_2^\bullet; \sigma''$, hence there is a forward trace σ''' shorter than $t_2^\bullet; \sigma''$ such that $\sigma''' \asymp t_2^\bullet; \sigma''$. So the trace $\sigma^\bullet; \sigma'''$ is shorter than $\sigma_2 = \sigma^\bullet; t_2^\bullet; \sigma''$, and the result follows by induction.

2. t_1 and t_2 are forward: Assume $t_1 \equiv R \xrightarrow{\mu[k]} R'$ and $t_2 \equiv R \xrightarrow{\nu[l]} R''$ for some R, R', R''. Transitions t_1 and t_2 must be be concurrent according to Proposition 4.3. Since σ_1 and σ_2 are cofinal, there must be transitions $t_1' \equiv P \xrightarrow{\nu[l]} P'$ and $t_2' \equiv Q \xrightarrow{\mu[k]} Q'$ for some P, P', Q, Q' at some later stage in σ_1 and σ_2. We look at the case of $t_1' \equiv P \xrightarrow{\nu[l]} P'$ in σ_1 here, the other case follows in the corresponding way. All transitions from t_1 to t_1' are concurrent since we can do $\mu[k]$ either as the first transition $R \xrightarrow{\mu[k]} R'$ of σ_1 or as the last transition in the sub-trace $t_2; t^*; t_2'$. Hence, we can rearrange the transitions by the technique used in the proof of Proposition 4.7 so that $R \xrightarrow{\nu[l]} R'''$ comes first. This transition is identical to t_2. So we have moved the first non-matching pair of transitions in σ_1 and σ_2 to the right, and the result follows by induction.

3. $t_1^\bullet$ and $t_2^\bullet$ are reverse: Transitions $t_1^\bullet \equiv R \xrightarrow{\underline{\mu}[k]} R'$ and $t_2^\bullet \equiv R \xrightarrow{\underline{\nu}[l]} R''$, for some R, R' and R'', are undoing different actions. Since σ_1 and σ_2 are cofinal, there is either a transition $t_1'^\bullet \equiv P'' \xrightarrow{\underline{\mu}[k]} P'''$, for some P'' and P''' in σ_2, or a transition $t_1' \equiv P \xrightarrow{\mu[k]} P'$, for some P and P' in σ_1, redoing the action undone in $t_1^\bullet$. For $t_2^\bullet$ there is either a transition $t_2'^\bullet \equiv S'' \xrightarrow{\underline{\mu}[k]} S'''$, for some S'' and S''' in σ_1, or a transition $t_2' \equiv S \xrightarrow{\mu[k]} S'$, for some S and S' in σ_2, redoing the action undone in $t_1^\bullet$. We treat these cases separately:

(a) $t_1'^{\bullet} \equiv P'' \xrightarrow{\underline{\mu}[k]} P'''$ in σ_2 or $t_2'^{\bullet} \equiv S'' \xrightarrow{\underline{\nu}[l]} S'''$ in σ_1: We consider $t_1'^{\bullet} \equiv P'' \xrightarrow{\underline{\mu}[k]} P'''$ in σ_2, the other case follows correspondingly. The traces $t_1^{\bullet}$ and $t_2^{\bullet}$ are concurrent according to Proposition 4.2. All transitions from $t_2^{\bullet}$ to $t_1'^{\bullet}$ must be concurrent since from R we can do $\mu[k]$ either as the first transition as in σ_1 or as the last one in this sub-trace. So we can rearrange the transitions by a similar technique of "swaps" as used in the proof of Proposition 4.7 so that $R \xrightarrow{\underline{\mu}[k]} R'''$ comes first. This transition is identical to $t_1^{\bullet}$, and we have moved the first non-matching pair of transitions in σ_1 and σ_2 to the right, hence the case follows by induction.

(b) $t_1' \equiv P \xrightarrow{\mu[k]} P'$ in σ_1 and $t_2' \equiv S \xrightarrow{\nu[l]} S'$ in σ_2 and $P'' \xrightarrow{\underline{\mu}[k]} P'''$ not in σ_2 and $S'' \xrightarrow{\underline{\nu}[l]} S'''$ not in σ_1: We consider $t_1' \equiv P \xrightarrow{\mu[k]} P'$ in σ_1 only, the case $t_2' \equiv S \xrightarrow{\nu[l]} S'$ in σ_2 follows correspondingly. We show that the transitions from $t_1^{\bullet}$ up to t_1' are concurrent. In fact, we show that $t_1^{\bullet}$ and t'', the next transition on the way to t_1', are concurrent. If they are concurrent, we can swap them thus moving $t_1^{\bullet}$ to the right towards t_1'. If we follow this approach we will eventually have $t_1^{\bullet}$ immediately to the left of t_1', and we can remove $t_1^{\bullet}; t_1'$ according to Definition 4.8. What remains to be done is to prove that $t_1^{\bullet}$ and t'' are concurrent. There are two cases: b If t'' is forward then we have a situation as in the proof of Proposition 4.8, where we show that the transitions are concurrent. The second case is for t'' being reverse. Assume $t'' \equiv R' \xrightarrow{\underline{\alpha}[n]} R''$. Hence by Proposition 4.1.3 we have $R' \xrightarrow{\underline{\mu}[k]} R$. So we have $R' \xrightarrow{\underline{\alpha}[n]} R''$ and $R' \xrightarrow{\underline{\mu}[k]} R$. This matches the assumptions of Proposition 4.6 (where R' is P). If we show that $n \notin cau(R, k)$ then they are concurrent. The transitions $R \xrightarrow{\underline{\mu}[k]} R' \xrightarrow{\underline{\alpha}[n]} R''$ show that $\mu[k]$ is undone before $\alpha[n]$, so n cannot be one of the causes of k in R. Hence, the transitions $t_1^{\bullet}$ and t'' are concurrent, and the result follows by induction.

One of the consequences of causal consistency for CCB_s concerns reachability: any state that can be reached from a standard process during an arbitrary computation can be reached by computing forwards alone. This property is not valid in the full calculus CCB as can be seen in Chapter 1 and in Example 4.6.

4.4 Expressiveness

A comparison of the expressiveness of CCB to existing calculi is complicated by the non-existence of non-deterministic choice in our calculus, whereas other comparable

calculi contain it. Therefore, we show that non-deterministic choice could easily be added to CCB and was left out only to make it more realistic for our application. CCB with non-deterministic choice has this syntax:

$$P ::= S | S \stackrel{def}{=} P \;\Big|\; (s;b).P \;\Big|\; P|Q \;\Big|\; P+Q \;\Big|\; P\backslash L$$

The + operator is the standard non-deterministic choice operator.

The forward SOS rules are extended by the following rule:

$$\textsf{sum}\ \frac{P \xrightarrow{a[k]} P' \quad \textsf{fsh}[k](Q)}{P+Q \xrightarrow{a[k]} P'+Q} \qquad \textsf{concert sum}\ \frac{P \xrightarrow{a[k]} P' \quad \textsf{fsh}[k](Q)}{P+Q \xrightarrow{a[k]} P'+Q}$$

The reverse SOS rules are extended with following rule:

$$\textsf{sum rev sum}\ \frac{P \xrightarrow{\underline{a[k]}} P' \quad \textsf{std}(Q)}{P+Q \xrightarrow{\underline{a[k]}} P'+Q}$$

We now can compare the expressiveness of this calculus to existing calculi. If we consider CCSK and CCS-R, then it is at least as expressive as CCSK and CCS-R. In addition, in our calculus we have the concerted actions. Whilst other calculi can also have a forward action followed by a reverse action, there is no possibility to prevent other actions to happen in between. This is a new feature in CCB.

Due to our new operator, we can get sequences of transitions not possible in calculi without out-of-causal-order reversibility. If we take the example from Chapter 1, we get the following trace of actions (underlining indicating reversing, as before, and writing the concerted reaction as two transitions): $cdq\underline{cd}$. Here c is done before d and is undone before d as well. In causal order, we would have $cdq\underline{dc}$ as the only possible trace.

The reversible π calculus, as introduced in [66], is also less expressive than CCB, because it is a causally consistent calculus. Similar to CCSK and CCSR, it does not have concerted actions. Compared to the κ-calculus, CCB is not more expressive. In particular, out-of-causal-order reversibility can be modelled in the κ-calculus, as we have seen in Section 3.4.4. As we have seen in Section 3.4.5, P systems can also model out-of-causal-order reversibility, but comparing expressiveness is not directly possible, since P systems does not have transition labels.

4.5 Conclusion

We have given a formal definition of CCB. We have also examined properties of the calculus, showing that the calculus without weak actions is causally consistent. The calculus without weak actions is also a conservative extension of a forward-only calculus. In contrast if we add the weak actions we get a more complex behaviour. In particular, we can reach states by doing forward and reverse transitions which cannot be reached by computing forwards alone.

Chapter 5

The hydration of formaldehyde in water

So far we have seen an informal example for the type of problems we want to tackle in Section 2 and a formal syntax and SOS rules for it in Section 4. In this chapter we examine a more complicated example to establish how our calculus performs and include the subscripts we used in Section 2 in our calculus.

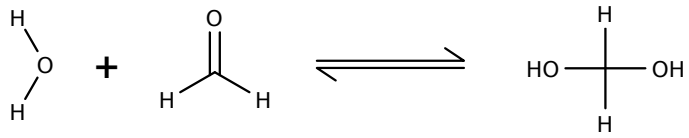

Figure 5.1: Hydration of formaldehyde in water into methanediol.

We describe in detail the hydration of formaldehyde in water.[1] Formaldehyde is a good preservative and is well known for its use in preserving specimen samples[38]. It also serves as an important building block in industrial processes and is therefore produced in large quantities.[2] At room temperature, formaldehyde forms a colourless and smelly gas. Formaldehyde reacts with water molecules to form methanediol. The reaction is shown in Figure 5.1. It is reversible but it is much faster towards the methanediol, so the equilibrium contains mostly methanediol. It should be noted that other reactions are taking place in a solution of formaldehyde in water, however we consider only the interaction of formaldehyde with water molecules. The carbon atom of formaldehyde is not shown in Figure 5.1 in line with a common shorthand formula for organic molecules. It resides at the point where the line, which reflect

[1] Detailed description of the reaction is given in [17]. The main path is on page 143, the base-catalysed reaction is on page 713, and the acid-catalysed reaction in on page 343 of [17].

[2] According to [30] 41 million tonnes of formaldehyde-water solution were produced in 2010, the most important product being various resins.

covalent bonds, from the oxygen atom and the hydrogen atoms meet; we follow this convention in all other reaction diagrams. We use single lines to represent single bonds in all our reactions, and we use double lines for double bonds, where two electron pairs are shared between atoms.

5.1 The most common mechanism of the reaction

A closer look at the reaction in Figure 5.1 shows that the most common sequence of intermediate reactions leading to methanediol is the one given in Figure 5.2. We now describe carefully its steps. The oxygen atom in one of the water molecules attacks the central carbon atom in the formaldehyde to form the intermediate 2 in Figure 5.2. Then one of the hydrogen atoms of the positively charged oxygen atom is taken away by another water molecule in a proton transfer. This gives the intermediate 3. Finally, a proton is donated to the other oxygen atom in the intermediate 3 to give the final configuration 4. As in the description of the autoprotolyis in Section 2.1 curved arrows for electron movements are given for the reaction going from left to right.

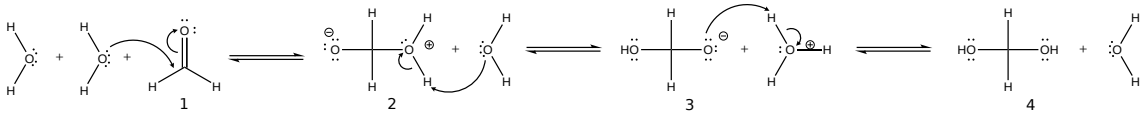

Figure 5.2: The most common path through the hydration of formaldehyde.

In order to model this reaction, we need to understand what it is that makes it happen. The main factor is that the oxygen atom in the water molecule is nucleophilic: it tends to connect to another atomic nucleus, which is electrophile, i.e. tends to connect to nucleophiles. In the formaldehyde the oxygen atom attracts electrons towards itself, so the carbon atom has a positive charge and is an electrophile. Now, the oxygen atom in the water molecule, which also attracts electrons and therefore has negative charge and is a nucleophile, is attracted by this carbon atom and forms a bond to the carbon atom. This bond is formed out of the electrons of one of the electron pairs in the oxygen atom so far not involved in bonding, so-called lone electron pairs. Since carbon cannot form more than four bonds, this reaction is compensated by the double bond in the formaldehyde becoming a single bond and the electrons from the double bond moving to a lone pair on the oxygen atom, which now has three lone pairs. These movements are *concerted*, namely they happen together without a stable intermediate state and cannot be separated. So there is a continuous change between two tetracoordinate species through a pentacoordinate transition state. We have now reached the intermediate 2 in Figure 5.2.

Looking at the intermediate 2, we can see that contains one oxygen atom which is negatively charged, because the oxygen atom has three lone pairs and a surplus of one electron. The other oxygen atom is positively charged, since it has three bonds and only one lone pair and therefore is missing an electron. The intermediate 2 reacts then with a water molecule. The water molecule is nucleophilic and has lone pairs for a bonding. It therefore can abstract one of the hydrogen atoms on the positively charged oxygen atom. This leads to the intermediate 3 and a H_3O^+ molecule, a water molecule with an additional hydrogen atom and a positively charged oxygen atom.

Finally, a hydrogen atom can be re-donated to the negatively charged oxygen atom. Note that the re-donated hydrogen atom may not be the one which was originally attached to the oxygen atom. This transfer is possible since the oxygen atom in the H_3O^+ is electron-deficient and the negatively charged oxygen atom is rich in electrons. We then get the substrate 4 which contains the final product methanediol, and water.

5.2 Other paths through the reaction

There are other paths through the reaction of formaldehyde and water. We assume that we now start with a mixture of three water molecules and one formaldehyde molecule. This shall allow us to explore all other water-formaldehyde interactions. Two water molecules can interact to form H_3O^+ and OH^-. This is known as *autoprotolysis of water* and is described in detail in Chapter 2 and [59]. These two molecules can be considered an acid and a base respectively[3]. Both can interact with formaldehyde and thus produce methanediol by different mechanisms than those in the previous section.

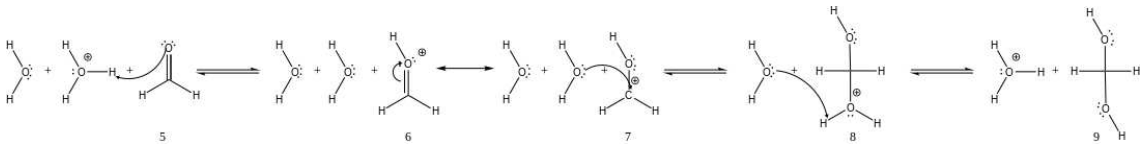

Figure 5.3: Acid-catalysed hydration of formaldehyde in water.

The acid-catalysed reaction is described in Figure 5.3. Here the H_3O molecule, which was formed during the autoprotolysis serves as the acid as it easily donates a proton. This proton can be donated to the oxygen atom in the formaldehyde, since

[3]There are slightly different definitions of acids and bases in the literature, we use the general Bronsted-Lowry acid-base theory, which defines a base as a proton acceptor and an acid as a proton donor.

this is slightly negatively charged, as we have seen. We then get the intermediate 6 in Figure 5.3. The charge on the oxygen atom can "move" to the carbon atom by the electrons forming one of the bonds between the carbon atom and the oxygen atom switching to the oxygen atom as a lone pair. The real charge distribution is somewhere in between. We shall assume that the intermediates 6 and 7 are somewhat different structures, and we shall model the transition from 6 to 8 as a pair of concerted actions. Once the intermediate 8 is reached, one of the protons from the oxygen atom in the intermediate 8 can be abstracted by one of the water molecules, giving H_3O^+ and making the reaction a catalytic process.

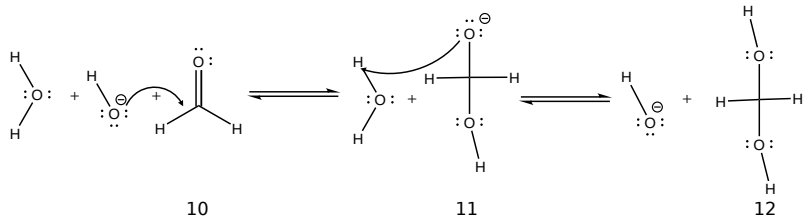

Figure 5.4: Base-catalysed hydration of formaldehyde in water.

The base-catalysed reaction in Figure 5.4 starts with a water and a OH^- molecule, the base, which tends to accept protons very strongly. It can therefore interact with the carbon atom in the formaldehyde, which is similar to the water molecule attacking the carbon atom. One of the bonds of the carbon-oxygen double bond is broken and the oxygen atom becomes negatively charged. This gives the intermediate 11 which is in fact the same as the intermediate 3 in Figure 5.2. Then, one of the protons of the water molecule can be abstracted, and we obtain the methanediol and an HOH^- molecule. Although this is the same structure as the original acid, it is made up from different atoms. The process is considered catalytic since the OH^- molecule is retrieved at the end of the process.

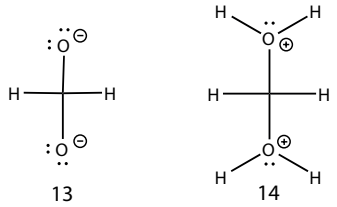

Figure 5.5: Other compounds possible in the reaction of formaldehyde with water.

There are other ways how several molecules of water can react with formaldehyde to produce methanediol and other byproducts. For example the base, OH^-, could directly interact with intermediate 7 to form methanediol and two water molecules. Also, the two compounds given in Figure 5.5 can be created as intermediate products in the reaction.

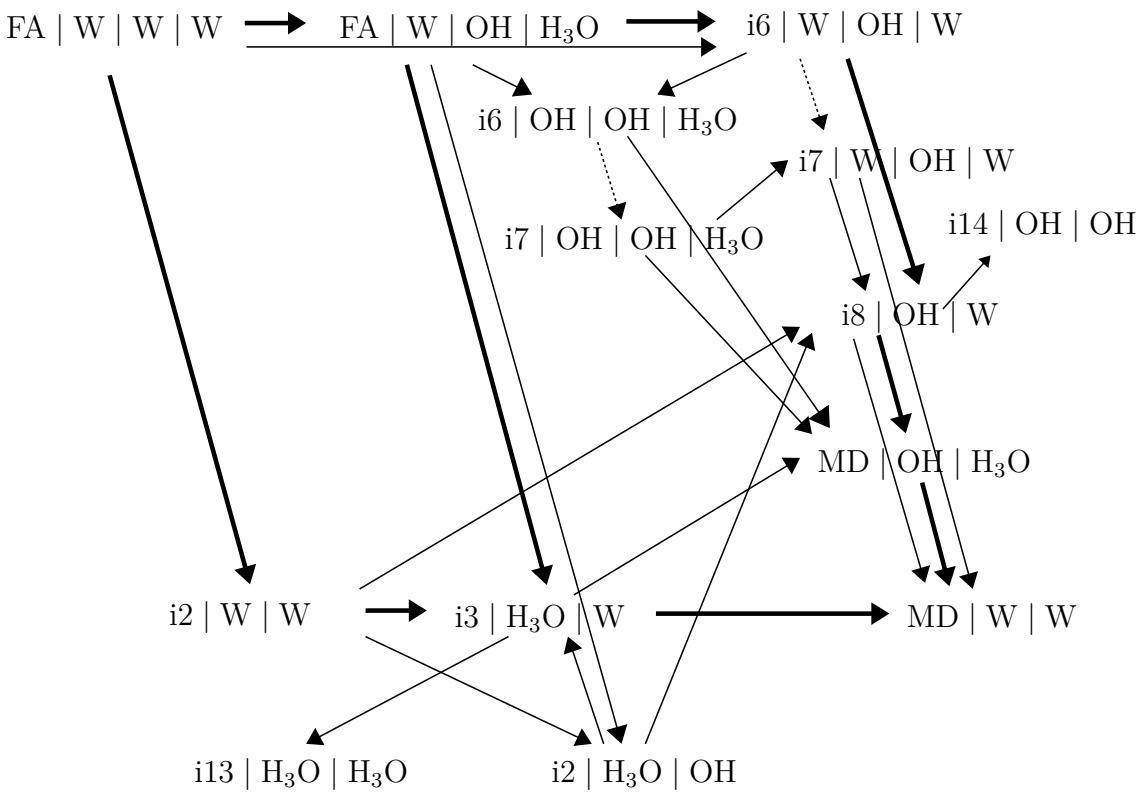

Figure 5.6: All possible reactions in a system of formaldehyde and three water molecules. The reactions displayed in more detail in Figure 5.7 are denoted here by bold arrows.

All possible reactions of a system of one formaldehyde and three water molecules are shown in Figure 5.6. Although the directions of reactions are denoted with arrows, all reactions are reversible; the arrows merely indicate the direction to the final product. We write FA for formaldehyde, W for water, MD for methanediol, OH for hydroxide/OH^-, and H3O for hydronium/H_3O^+. Many of the intermediate compounds are denoted by iX where $X \in \{2, 3, 6, 7, 8, 13, 14\}$. Figure 5.6 includes all intermediates from Figure 5.3 and Figure 5.4. The compounds in Figure 5.5 have only one path leading to them, and we could only move away from them by reverting these actions. This is because these compounds are the "extreme" states where there is either no hydrogen atom on an oxygen atom or both oxygen atoms are fully saturated with two hydrogen atoms. The resonance between the intermediates i6 and i7 in Figure 5.3 is represented by dashed arrows.

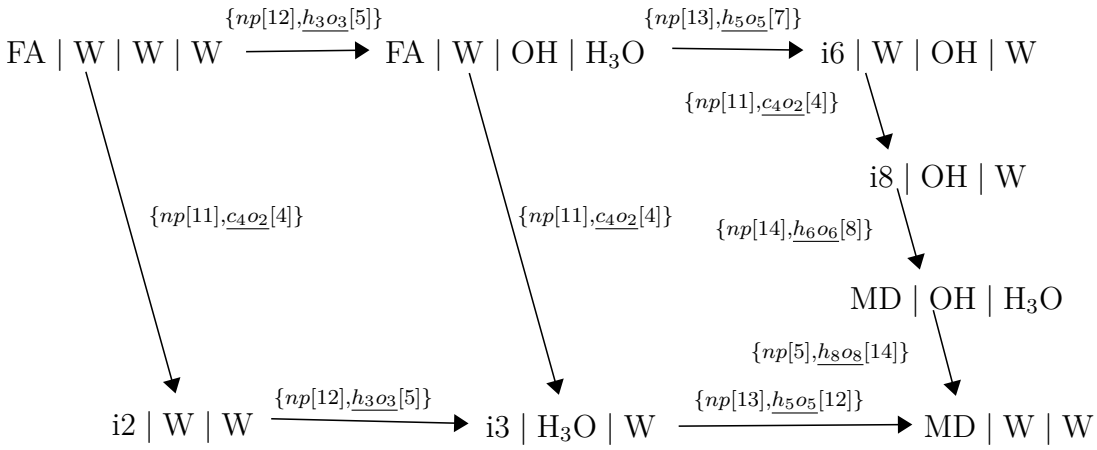

Figure 5.7: The three main reaction paths in the hydration of formaldehyde.

5.3 A CCB model of the hydration of formaldehyde in water

We are now ready to model our reaction in CCB. Figure 5.7 shows in more detail the three main paths through our reaction that we described in the last section; these paths have been highlighted in bold in Figure 5.6.

The initial state of the reaction is modelled by the composition of one formaldehyde FA and three water molecules, written as FA | W | W | W. The final products are methanediol MD and two water molecules, written as MD | W | W. The path via the intermediate molecules 2 and 3 in Figure 5.2 is via the nodes denoted by i2 and i3 in Figure 5.7. The FA | W | OH | H_3O node is the result of an autoprotolysis. The base-catalysed and acid-catalysed reactions are put into the same diagram, again the intermediates i6 and i8 correspond to molecules in Figure 5.3. As we can see in Figure 5.7 the reactions are driven completely by concerted actions.

We model the formaldehyde molecule CH_2O and the three water molecules H_2O as appropriate compositions of their atoms, namely hydrogen, oxygen and carbon. We use our general prefixing operator to define these atoms:

$$\begin{aligned} H &\stackrel{def}{=} (h;p).H' \\ O &\stackrel{def}{=} (o,o,n).O' \\ C &\stackrel{def}{=} (c,c,c,c;p).C' \end{aligned}$$

Carbon has four strong actions c, representing the potential for four covalent bonds, and a weak action p, standing for a positive partial charge. The oxygen atom can have up to three bonds. Normally it has two bonds, however, if a suitable

reaction partner is close, an additional weak bond is available. We use therefore the simple prefixing operator to model it, with one weak action n standing for the potential for a negative partial charge. A partial charge means that a part or parts of a molecule have an electric charge, even though the molecule as a whole is neutrally charged. These uneven charge distributions enable many reactions by allowing another molecule to attack a partially charged part. The hydrogen atom has one strong bond, and we use additionally a weak action p to represent that it can become positively charged. Processes H', O', and C' represent unspecified further behaviour of the respective atoms.

Since our reaction involves multiple copies of H and O we shall adopt a subscript notation to denote distinct copies of the same process. Both action labels and process names will be subscripted. For example, $H_1 \stackrel{def}{=} (h_1; p).H_1'$ and $H_2 \stackrel{def}{=} (h_2; p).H_2'$ are two copies of the hydrogen atom H. We shall abstract away these subscripts in Section 6, where we introduce chemical process equivalence. Also, the copies of actions c of carbon will be subscripted.

The synchronisation function γ is defined on subscripted actions as follows:

$$\begin{array}{lcll}
\gamma(c_i, h_j) & = & c_i h_j & i \in \{1, \ldots, 4\}, j \in \{1, \ldots, 8\} \\
\gamma(h_i, o_j) & = & h_i o_j & i \in \{1, \ldots, 8\}, j \in \{1, \ldots, 6\} \\
\gamma(c_i, o_j) & = & c_i o_j & i \in \{1, \ldots, 4\}, j \in \{1, \ldots, 6\} \\
\gamma(c_i, n) & = & c_i n & i \in \{1, \ldots, 4\} \\
\gamma(h_i, n) & = & h_i n & i \in \{1, \ldots, 8\} \\
\gamma(n, p) & = & np &
\end{array}$$

Now we are ready to model H_2O and CH_2O molecules. A water molecule is modelled simply as a composition of two copies of the hydrogen process with one copy of oxygen atom; see below. The restriction of actions h_i and o_i, for $i \in \{3, 4\}$, ensures that actions such as $h_3[5]$ cannot be done alone but only together with their partners $o_3[5]$. This happens via undoing of $h_3 o_3[5]$ bond. When any of these actions is fresh they can only happen together with their partners (as prescribed by the communication function γ), and not alone.

$$((h_3[5]; p).H_3' \mid (h_4[6]; p).H_4' \mid (o_3[5], o_4[6], n).O_2') \setminus \{h_3, h_4, o_3, o_4\}$$

The formaldehyde molecule is modelled by

$$\begin{array}{l}
(((c_1[1], c_2[2], c_3[3], c_4[4]; p).C' \mid (h_1[1]; p).H_1' \mid (h_2[2]; p).H_2' \mid (o_1[3], o_2[4], n).O_1') \\
\quad \setminus \{c_1, c_2, c_3, c_4, h_1, h_2, o_1, o_2, \underline{c_1 h_1}, \underline{c_2 h_2}\}
\end{array}$$

Again the restriction of actions c_i, for $i \in \{1,2,3,4\}$, o_j, for $j \in \{1,2\}$, and h_k, for $k \in \{1,2\}$, ensures that such actions cannot be done alone but only together with their partners. When any of these actions is fresh they can only happen together with their partners (as prescribed by the communication function γ), and not alone. This also means that we do not allow the creation of new bonds (involving these actions) between different molecules as single acts of synchronisation. Such new bonds will be created via concerted actions and (almost) always via the weak np bonds which will then get promoted to strong bonds.

We also restrict $\underline{c_i h_i}$ for $i \in \{1,2\}$. It prevents breaking any of the bonds between C_1 and its hydrogen atoms H_1 and H_2. This serves two purposes. Firstly, it makes sure that once we have done the p action of the carbon process, we will break one of the bonds between the carbon atom and the oxygen atom. This is justified since in reality it is one of the oxygen bonds which is broken, so this models the reality closely. Secondly, it also prevents O_2 or O_3 from abstracting H_1 or H_2 from the carbon atom. Note that $\underline{h_i}$, $\underline{o_j}$ and $\underline{n}$ are not restricted in FA and in W: this allows us to break bonds involving these actions as a part of concerted actions.

The four molecules of the reaction are now composed in parallel:

$$(\mathrm{CH_2O} \mid \mathrm{H_2O} \mid \mathrm{H_2O} \mid \mathrm{H_2O}) \setminus \{n, p\}$$

We restrict actions n and p, so that they cannot happen separately from each other but only together within this system of processes.

The main path through the reaction require only two copies of water molecules so we start with the following initial state, where keys $1, \ldots, 8$ specify the initially existing bonds of formaldehyde and the two water molecules.

$$\begin{aligned}
&(((c_1[1], c_2[2], c_3[3], c_4[4]; p).C' \mid (h_1[1]; p).H_1' \mid (h_2[2]; p).H_2' \mid (o_1[3], o_2[4], n).O_1') \\
&\qquad \setminus \{c_1, c_2, c_3, c_4, h_1, h_2, o_1, o_2, \underline{c_1 h_1}, \underline{c_2 h_2}\} \\
&\mid ((h_3[5]; p).H_3' \mid (h_4[6]; p).H_4' \mid (o_3[5], o_4[6], n).O_2') \setminus \{h_3, h_4, o_3, o_4\} \\
&\mid ((h_5[7]; p).H_5' \mid (h_6[8]; p).H_6' \mid (o_5[7], o_6[8], n).O_3') \setminus \{h_5, h_6, o_5, o_6\} \\
&\mid (h_7[9]; p).H_7' \mid (h_8[10]; p).H_8' \mid (o_7[9], o_8[10], n).O_4' \setminus \{h_7, h_8, o_7, o_8\}) \setminus \{n, p\}
\end{aligned}$$

As we did in Section 2.1, we move restrictions to the outside, giving the following process:

$$
\begin{aligned}
&((c_1[1], c_2[2], c_3[3], c_4[4]; p).C' \mid (h_1[1]; p).H_1' \mid (h_2[2]; p).H_2' \mid (o_1[3], o_2[4], n).O_1' \\
&\mid (h_3[5]; p).H_3' \mid (h_4[6]; p).H_4' \mid (o_3[5], o_4[6], n).O_2' \\
&\mid (h_5[7]; p).H_5' \mid (h_6[8]; p).H_6' \mid (o_5[7], o_6[8], n).O_3' \\
&\mid (h_7[9]; p).H_7' \mid (h_8[10]; p).H_8' \mid (o_7[9], o_8[10], n).O_4') \\
&\setminus \{c_1, c_2, c_3, c_4, h_1, h_2, o_1, o_2, \underline{c_1 h_1}, \underline{c_2 h_2}\} \setminus \{h_3, h_4, o_3, o_4\} \\
&\setminus \{h_5, h_6, o_5, o_6\} \setminus \{h_7, h_8, o_7, o_8\} \setminus \{n, p\}
\end{aligned}
$$

We leave out restrictions in the following sections to improve readability. The initial process without restrictions is as follows:

$$
\begin{aligned}
&(c_1[1], c_2[2], c_3[3], c_4[4]; p).C' \mid (h_1[1]; p).H_1' \mid (h_2[2]; p).H_2' \mid (o_1[3], o_2[4], n).O_1' \\
&\mid (h_3[5]; p).H_3' \mid (h_4[6]; p).H_4' \mid (o_3[5], o_4[6], n).O_2' \\
&\mid (h_5[7]; p).H_5' \mid (h_6[8]; p).H_6' \mid (o_5[7], o_6[8], n).O_3' \\
&\mid (h_7[9]; p).H_7' \mid (h_8[10]; p).H_8' \mid (o_7[9], o_8[10], n).O_4'
\end{aligned}
$$

remembering that the restrictions are still there.

5.3.1 The main path through the reaction

We first consider the reaction steps in Figure 5.2, following the path via i3 | H_3O | W and i2 | W | W in Figures 5.6 and 5.7. The first step is the n, p reaction between C_1 and O_2 or O_3. Note that there are other n, p reactions that are allowed by our model. We could have O_2 getting a hydrogen atom from O_3, or vice versa, which is the autoprotolysis of water which we have mentioned before. We shall discuss this further later on. Also O_1 could pull one of the hydrogen atoms from one of the water molecules: again, we shall discuss it later.

Returning to the first step of our reaction, we have O_2 bonding with C with the key 11. This is followed immediately by breaking of the bond 3 or 4 by the rule concert. Note that breaking of 1 or 2 is not possible because of the restriction of c_1h_1 and c_2h_2. We break the bond 4 and get a transition by concerted actions: we create the bond $np[11]$ and break the bond $\underline{c_4 o_2}[4]$. Henceforth we shall write the name of the target of a transition below the transition, using the names that appear in Figures 5.6-5.7. Here, for example, the compound resulting from this transition is

i2 | W but since in general we have an extra water molecule W, we write i2 | W | W (changes highlighted in **bold**).

$$
\begin{aligned}
&(c_1[1], c_2[2], c_3[3], c_4[4]; p).C' \mid (h_1[1]; p).H'_1 \mid (h_2[2]; p).H'_2 \mid (o_1[3], o_2[4], n).O'_1 \\
&\mid (h_3[5]; p).H'_3 \mid (h_4[6]; p).H'_4 \mid (o_3[5], o_4[6], n).O'_2 \\
&\mid (h_5[7]; p).H'_5 \mid (h_6[8]; p).H'_6 \mid (o_5[7], o_6[8], n).O'_3 \\
&\mid (h_7[9]; p).H'_7 \mid (h_8[10]; p).H'_8 \mid (o_7[9], o_8[10], n).O'_4 \\
&\xrightarrow{\{np[11], \underline{c_4 o_2}[4]\}} \\
&(c_1[1], c_2[2], c_3[3], \boldsymbol{c_4}; p[\mathbf{11}]).C' \mid (h_1[1]; p).H'_1 \mid (h_2[2]; p).H'_2 \\
&\mid (o_1[3], \boldsymbol{o_2}, n).O'_1 \mid (h_3[5]; p).H'_3 \mid (h_4[6]; p).H'_4 \mid (o_3[5], o_4[6], n[\mathbf{11}]).O'_2 \\
&\mid (h_5[7]; p).H'_5 \mid (h_6[8]; p).H'_6 \mid (o_5[7], o_6[8], n).O'_3 \\
&\mid (h_7[9]; p).H'_7 \mid (h_8[10]; p).H'_8 \mid (o_7[9], o_8[10], n).O'_4 \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{i2} \mid \text{W} \mid \text{W}
\end{aligned}
$$

Now the prom rewrite rule must be applied before we derive the next concerted transition: we promote the weak bond 11 of the carbon atom to a stronger bond on c_4, which has become available:

$$
\begin{aligned}
&(c_1[1], c_2[2], c_3[3], \boldsymbol{c_4}; p[\mathbf{11}]).C' \mid (h_1[1]; p).H'_1 \mid (h_2[2]; p).H'_2 \\
&\mid (o_1[3], \boldsymbol{o_2}, n).O'_1 \mid (h_3[5]; p).H'_3 \mid (h_4[6]; p).H'_4 \mid (o_3[5], o_4[6], n[\mathbf{11}]).O'_2 \\
&\mid (h_5[7]; p).H'_5 \mid (h_6[8]; p).H'_6 \mid (o_5[7], o_6[8], n).O'_3 \\
&\mid (h_7[9]; p).H'_7 \mid (h_8[10]; p).H'_8 \mid (o_7[9], o_8[10], n).O'_4 \\
&\Rightarrow \\
&((c_1[1], c_2[2], c_3[3], c_4[\mathbf{11}]; \boldsymbol{p}).C' \mid (h_1[1]; p).H'_1 \mid (h_2[2]; p).H'_2 \\
&\mid (o_1[3], o_2, n).O'_1 \mid (h_3[5]; p).H'_3 \mid (h_4[6]; p).H'_4 \mid (o_3[5], o_4[6], n[11]).O'_2 \\
&\mid (h_5[7]; p).H'_5 \mid (h_6[8]; p).H'_6 \mid (o_5[7], o_6[8], n).O'_3 \\
&\mid (h_7[9]; p).H'_7 \mid (h_8[10]; p).H'_8 \mid (o_7[9], o_8[10], n).O'_4 \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{i2} \mid \text{W} \mid \text{W}
\end{aligned}
$$

The next step is to form a bond between O_3 and either H_3 or H_4. We bond with H_3 with the key 12 and break the bond 5, producing a pair of concerted actions. We then move a weak bond 11 on n in O_2 to a stronger bond on o_3 (which has become

available) by rewrite rule move-r. We also promote the weak bond 12 in H_3 to a strong bond on h_3 by rewrite rule prom, giving overall this transition:

$$\xrightarrow{\{np[12],\underline{h_3o_3}[5]\}} (c_1[1], c_2[2], c_3[3], c_4[11]; p).C' \mid (h_1[1]; p).H_1' \mid (h_2[2]; p).H_2'$$
$$\mid (o_1[3], o_2, n).O_1' \mid (h_3[\mathbf{12}]; p).H_3' \mid (h_4[6]; p).H_4' \mid (o_3[\mathbf{11}], o_4[6], n).O_2'$$
$$\mid (h_5[7]; p).H_5' \mid (h_6[8]; p).H_6' \mid (o_5[7], o_6[8], n[\mathbf{12}]).O_3'$$
$$\mid (h_7[9]; p).H_7' \mid (h_8[10]; p).H_8' \mid (o_7[9], o_8[10], n).O_4'$$

i3 | H_3O | W

The next step is a proton transfer from O_3 to O_1. We transfer H_5 (but we could have used H_6 or H_3 since they all have the p action ready). Performing the transfer of H_5 from O_3 to O_1 (and breaking the bond 12), we obtain the following:

$$\xrightarrow{\{np[13],\underline{h_5o_5}[7]\}} (c_1[1], c_2[2], c_3[3], c_4[11]; p).C' \mid (h_1[1]; p).H_1' \mid (h_2[2]; p).H_2'$$
$$\mid (o_1[3], o_2, n[\mathbf{13}]).O_1' \mid (h_3[12]; p).H_3' \mid (h_4[6]; p).H_4' \mid (o_3[11], o_4[6], n).O_2'$$
$$\mid (\boldsymbol{h_5}; p[\mathbf{13}]).H_5' \mid (h_6[8]; p).H_6' \mid (\boldsymbol{o_5}, o_6[8], n[12]).O_3'$$
$$\mid (h_7[9]; p).H_7' \mid (h_8[10]; p).H_8' \mid (o_7[9], o_8[10], n).O_4'$$

MD | W | W

we obtain the final product (methanediol $CH_2(OH)_2$) and two water molecules:

$$\Rightarrow\ (c_1[1], c_2[2], c_3[3], c_4[11]; p).C' \mid (h_1[1]; p).H_1' \mid (h_2[2]; p).H_2'$$
$$\mid (o_1[3], o_2[\mathbf{13}], \boldsymbol{n}).O_1' \mid (h_3[12]; p).H_3' \mid (h_4[6]; p).H_4' \mid (o_3[11], o_4[6], n).O_2'$$
$$\mid (h_5[\mathbf{13}]; \boldsymbol{p}).H_5' \mid (h_6[8]; p).H_6' \mid (o_5[\mathbf{12}], o_6[8], \boldsymbol{n}).O_3'$$
$$\mid (h_7[9]; p).H_7' \mid (h_8[10]; p).H_8' \mid (o_7[9], o_8[10], n).O_4'$$

MD | W | W

Note that the n, p actions are ready again and all the existing bonds are on strong actions. So we now can reverse the reaction by O_3 abstracting a hydrogen atom from H_4 or H_5, and all the way to the initial state. Moreover, let us inspect the bonds 4, 5 and 7 which are broken during the reaction. The bonds were formed during the initial bonding of the atoms. They are broken as a result of application of our new general prefixing operator. This operator enables the reaction to work forwards without external control. Indeed, the original order of the formation of the bonds is completely irrelevant for the reaction to work.

5.3.2 The base-catalysed path

We now consider the base-catalysed path described in Figure 5.4. The path is via FA | W | OH | H_3O and i3 | H_3O | W to MD | W | W in Figures 5.6-5.7. The path needs three water molecules. We shall show the application of the promotion or move rules without explaining them in detail. We start with the initial system (remembering that restrictions are not shown, changes highlighted in **bold**):

$(c_1[1], c_2[2], c_3[3], c_4[4]; p).C' \mid (h_1[1]; p).H'_1 \mid (h_2[2]; p).H'_2 \mid (o_1[3], o_2[4], n).O'_1$
$\mid (h_3[5]; p).H'_3 \mid (h_4[6]; p).H'_4 \mid (o_3[5], o_4[6], n).O'_2$
$\mid (h_5[7]; p).H'_5 \mid (h_6[8]; p).H'_6 \mid (o_5[7], o_6[8], n).O'_3$
$\mid (h_7[9]; p).H'_7 \mid (h_8[10]; p).H'_8 \mid (o_7[9], o_8[10], n).O'_4$

FA | W | W | W

$\xrightarrow{\{np[12], \underline{h_3 o_3}[5]\}} (c_1[1], c_2[2], c_3[3], c_4[4]; p).C' \mid (h_1[1]; p).H'_1 \mid (h_2[2]; p).H'_2$
$\mid (o_1[3], o_2[4], n).O'_1 \mid (\boldsymbol{h_3}; p[\mathbf{12}]).H'_3 \mid (h_4[6]; p).H'_4 \mid (\boldsymbol{o_3}, o_4[6], n).O'_2$
$\mid (h_5[7]; p).H'_5 \mid (h_6[8]; p).H'_6 \mid (o_5[7], o_6[8], n[\mathbf{12}]).O'_3$
$\mid (h_7[9]; p).H'_7 \mid (h_8[10]; p).H'_8 \mid (o_7[9], o_8[10], n).O'_4$

$\Rightarrow (c_1[1], c_2[2], c_3[3], c_4[4]; p).C' \mid (h_1[1]; p).H'_1 \mid (h_2[2]; p).H'_2$
$\mid (o_1[3], o_2[4], n).O'_1 \mid (h_3[\mathbf{12}]; \boldsymbol{p}).H'_3 \mid (h_4[6]; p).H'_4 \mid (o_3, o_4[6], n).O'_2$
$\mid (h_5[7]; p).H'_5 \mid (h_6[8]; p).H'_6 \mid (o_5[7], o_6[8], n[12]).O'_3$
$\mid (h_7[9]; p).H'_7 \mid (h_8[10]; p).H'_8 \mid (o_7[9], o_8[10], n).O'_4$

FA | W | OH | H_3O

$\xrightarrow{\{np[11], \underline{c_4 o_2}[4]\}} (c_1[1], c_2[2], c_3[3], \boldsymbol{c_4}; p[\mathbf{11}]).C' \mid (h_1[1]; p).H'_1 \mid (h_2[2]; p).H'_2$
$\mid (o_1[3], \boldsymbol{o_2}, n).O'_1 \mid (h_3[12]; p).H'_3 \mid (h_4[6]; p).H'_4 \mid (o_3, o_4[6], n[\mathbf{11}]).O'_2$
$\mid (h_5[7]; p).H'_5 \mid (h_6[8]; p).H'_6 \mid (o_5[7], o_6[8], n[12]).O'_3$
$\mid (h_7[9]; p).H'_7 \mid (h_8[10]; p).H'_8 \mid (o_7[9], o_8[10], n).O'_4$

$$\Rightarrow (c_1[1], c_2[2], c_3[3], c_4[\mathbf{11}]; \boldsymbol{p}).C' \mid (h_1[1]; p).H_1' \mid (h_2[2]; p).H_2'$$
$$\mid (o_1[3], o_2, n).O_1' \mid (h_3[12]; p).H_3' \mid (h_4[6]; p).H_4' \mid (o_3[\mathbf{11}], o_4[6], \boldsymbol{n}).O_2'$$
$$\mid (h_5[7]; p).H_5' \mid (h_6[8]; p).H_6' \mid (o_5[7], o_6[8], n[12]).O_3'$$
$$\mid (h_7[9]; p).H_7' \mid (h_8[10]; p).H_8' \mid (o_7[9], o_8[10], n).O_4'$$

i3 | H_3O | W

$$\xrightarrow{\{np[13], \underline{h_3 n}[12]\}} (c_1[1], c_2[2], c_3[3], c_4[11]; p).C' \mid (h_1[1]; p).H_1' \mid (h_2[2]; p).H_2'$$
$$\mid (o_1[3], o_2, n[\mathbf{13}]).O_1' \mid (\boldsymbol{h_3}; p[\mathbf{13}]).H_3' \mid (h_4[6]; p).H_4' \mid (o_3[11], o_4[6], n).O_2'$$
$$\mid (h_5[7]; p).H_5' \mid (h_6[8]; p).H_6' \mid (o_5[7], o_6[8], \boldsymbol{n}).O_3'$$
$$\mid (h_7[9]; p).H_7' \mid (h_8[10]; p).H_8' \mid (o_7[9], o_8[10], n).O_4'$$

$$\Rightarrow (c_1[1], c_2[2], c_3[3], c_4[11]; p).C' \mid (h_1[1]; p).H_1' \mid (h_2[2]; p).H_2'$$
$$\mid (o_1[3], o_2[\mathbf{13}], \boldsymbol{n}).O_1' \mid (h_3[\mathbf{13}]; \boldsymbol{p}).H_3' \mid (h_4[6]; p).H_4' \mid (o_3[11], o_4[6], n).O_2'$$
$$\mid (h_5[7]; p).H_5' \mid (h_6[8]; p).H_6' \mid (o_5[7], o_6[8], n).O_3'$$
$$\mid (h_7[9]; p).H_7' \mid (h_8[10]; p).H_8' \mid (o_7[9], o_8[10], n).O_4'$$

MD | W | W

We have reached the final state of the reaction: a methanediol with two water molecules. Notice that this methanediol process is identical (including keys) to the methanediol process in the main path.

5.3.3 The acid-catalysed path

Next we consider the acid-catalysed path described in Figure 5.3. The path is via i6 | W | OH | W and i8 | OH | W in Figures 5.6–5.7. The path uses three water molecules as in the base-catalysed path. We shall apply the promotion or move rules as needed. We start with the initial system:

$$(c_1[1], c_2[2], c_3[3], c_4[4]; p).C' \mid (h_1[1]; p).H_1' \mid (h_2[2]; p).H_2'$$
$$\mid (o_1[3], o_2[4], n).O_1' \mid (h_3[12]; p).H_3' \mid (h_4[6]; p).H_4' \mid (o_3, o_4[6], n).O_2'$$
$$\mid (h_5[7]; p).H_5' \mid (h_6[8]; p).H_6' \mid (o_5[7], o_6[8], n[12]).O_3'$$
$$\mid (h_7[9]; p).H_7' \mid (h_8[10]; p).H_8' \mid (o_7[9], o_8[10], n).O_4'$$

FA | W | OH | H_3O

$$\xrightarrow{\{np[13],\underline{h_5o_5}[7]\}} (c_1[1], c_2[2], c_3[3], c_4[4]; p).C' \mid (h_1[1]; p).H_1' \mid (h_2[2]; p).H_2'$$
$$\mid (o_1[3], o_2[4], n[\mathbf{13}]).O_1' \mid (h_3[12]; p).H_3' \mid (h_4[6]; p).H_4' \mid (o_3, o_4[6], n).O_2'$$
$$\mid (\boldsymbol{h_5}; p[\mathbf{13}]).H_5' \mid (h_6[8]; p).H_6' \mid (\boldsymbol{o_5}, o_6[8], n[12]).O_3'$$
$$\mid (h_7[9]; p).H_7' \mid (h_8[10]; p).H_8' \mid (o_7[9], o_8[10], n).O_4'$$

$$\Rightarrow (c_1[1], c_2[2], c_3[3], c_4[4]; p).C' \mid (h_1[1]; p).H_1' \mid (h_2[2]; p).H_2'$$
$$\mid (o_1[3], o_2[4], n[13]).O_1' \mid (h_3[12]; p).H_3' \mid (h_4[6]; p).H_4' \mid (o_3, o_4[6], n).O_2'$$
$$\mid (h_5[\mathbf{13}]; \boldsymbol{p}).H_5' \mid (h_6[8]; p).H_6' \mid (o_5[12], o_6[8], n).O_3'$$
$$\mid (h_7[9]; p).H_7' \mid (h_8[10]; p).H_8' \mid (o_7[9], o_8[10], n).O_4'$$

i6 | W | OH | W

The last rewrite is by rule move-r. The reaction continues as follows:

$$\xrightarrow{\{np[11],\underline{c_4o_2}[4]\}} (c_1[1], c_2[2], c_3[3], \boldsymbol{c_4}; p[\mathbf{11}]).C' \mid (h_1[1]; p).H_1' \mid (h_2[2]; p).H_2'$$
$$\mid (o_1[3], \boldsymbol{o_2}, n[13]).O_1' \mid (h_3[12]; p).H_3' \mid (h_4[6]; p).H_4' \mid (o_3, o_4[6], n).O_2'$$
$$\mid (h_5[13]; p).H_5' \mid (h_6[8]; p).H_6' \mid (o_5[12], o_6[8], n[\mathbf{11}]).O_3'$$
$$\mid (h_7[9]; p).H_7' \mid (h_8[10]; p).H_8' \mid (o_7[9], o_8[10], n).O_4'$$

$$\Rightarrow (c_1[1], c_2[2], c_3[3], c_4[\mathbf{11}]; \boldsymbol{p}).C' \mid (h_1[1]; p).H_1' \mid (h_2[2]; p).H_2'$$
$$\mid (o_1[3], o_2[\mathbf{13}], \boldsymbol{n}).O_1' \mid (h_3[12]; p).H_3' \mid (h_4[6]; p).H_4' \mid (o_3, o_4[6], n).O_2'$$
$$\mid (h_5[13]; p).H_5' \mid (h_6[8]; p).H_6' \mid (o_5[12], o_6[8], n[11]).O_3'$$
$$\mid (h_7[9]; p).H_7' \mid (h_8[10]; p).H_8' \mid (o_7[9], o_8[10], n).O_4')$$

i8 | OH | W

We continue from i8 | OH | W via MD | OH | H_3O with concerted actions:

$$\xrightarrow{\{np[14],\underline{h_6o_6}[8]\}} (c_1[1], c_2[2], c_3[3], c_4[11]; p).C' \mid (h_1[1]; p).H_1' \mid (h_2[2]; p).H_2'$$
$$\mid (o_1[3], o_2[13], n).O_1' \mid (h_3[12]; p).H_3' \mid (h_4[6]; p).H_4' \mid (o_3, o_4[6], n).O_2'$$
$$\mid (h_5[13]; p).H_5' \mid (\boldsymbol{h_6}; p[\mathbf{14}]).H_6' \mid (o_5[12], \boldsymbol{o_6}, n[11]).O_3'$$
$$\mid (h_7[9]; p).H_7' \mid (h_8[10]; p).H_8' \mid (o_7[9], o_8[10], n[\mathbf{14}]).O_4'$$

$$\Rightarrow (c_1[1], c_2[2], c_3[3], c_4[11]; p).C' \mid (h_1[1]; p).H'_1 \mid (h_2[2]; p).H'_2$$
$$\mid (o_1[3], o_2[13], n).O'_1 \mid (h_3[12]; p).H'_3 \mid (h_4[6]; p).H'_4 \mid (o_3, o_4[6], n).O'_2$$
$$\mid (h_5[13]; p).H'_5 \mid (h_6\mathbf{[14]}; \boldsymbol{p}).H'_6 \mid (o_5[12], o_6\mathbf{[11]}, \boldsymbol{n}).O'_3$$
$$\mid (h_7[9]; p).H'_7 \mid (h_8[10]; p).H'_8 \mid (o_7[9], o_8[10], n[14]).O'_4$$

MD | OH | H_3O

We reach the final state of a methanediol and two water molecules:

$$\xrightarrow{\{np[5], \underline{h_8 o_8}[10]\}} (c_1[1], c_2[2], c_3[3], c_4[11]; p).C' \mid (h_1[1]; p).H'_1 \mid (h_2[2]; p).H'_2$$
$$\mid (o_1[3], o_2[13], n).O'_1 \mid (h_3[12]; p).H'_3 \mid (h_4[6]; p).H'_4 \mid (o_3, o_4[6], n\mathbf{[5]}).O'_2$$
$$\mid (h_5[13]; p).H'_5 \mid (h_6[14]; p).H'_6 \mid (o_5[12], o_6[11], n).O'_3$$
$$\mid (h_7[9]; p).H'_7 \mid (\boldsymbol{h_8}; p\mathbf{[5]}).H'_8 \mid (o_7[9], \boldsymbol{o_8}, n[14]).O'_4$$

$$\Rightarrow (c_1[1], c_2[2], c_3[3], c_4[11]; p).C' \mid (h_1[1]; p).H'_1 \mid (h_2[2]; p).H'_2$$
$$\mid (o_1[3], o_2[13], n).O'_1 \mid (h_3[12]; p).H'_3 \mid (h_4[6]; p).H'_4 \mid (o_3\mathbf{[5]}, o_4[6], \boldsymbol{n}).O'_2$$
$$\mid (h_5[13]; p).H'_5 \mid (h_6[10]; p).H'_6 \mid (o_5[12], o_6[11], n).O'_3$$
$$\mid (h_7[9]; p).H'_7 \mid (h_8\mathbf{[5]}; \boldsymbol{p}]).H'_8 \mid (o_7[9], o_8\mathbf{[14]}, \boldsymbol{n}).O'_4$$ MD | W | W

We return to the transition from intermediate 6 to intermediate 8 in Figure 5.3. Once the hydrogen atom is bound to the oxygen atom in compound 6 we have a so-called *resonance*, where the positive charge can be on the oxygen atom or the carbon atom (or in between) and the structure resonates between the intermediate 6 and 7 (indicated by the dashed arrow in Figure 5.3). We can model this movement between the two intermediates as follows: starting from i6 | W | OH | W we break

bond 4 without forming a new bond at the same time, and perform a move-r rewrite on O_1:

$$
\begin{aligned}
&(c_1[1], c_2[2], c_3[3], c_4[11]; p).C' \mid (h_1[1]; p).H'_1 \mid (h_2[2]; p).H'_2 \\
&\mid (o_1[3], o_2[13], n).O'_1 \mid (h_3[12]; p).H'_3 \mid (h_4[6]; p).H'_4 \mid (o_3[\mathbf{5}], o_4[6], \boldsymbol{n}).O'_2 \\
&\mid (h_5[13]; p).H'_5 \mid (h_6[10]; p).H'_6 \mid (o_5[12], o_6[11], n).O'_3 \\
&\mid (h_7[9]; p).H'_7 \mid (h_8[\mathbf{5}]; \boldsymbol{p}]).H'_8 \mid (o_7[9], o_8[\mathbf{14}], \boldsymbol{n}).O'_4 \\
&\xrightarrow{c_4 o_2[4]} \\
&(c_1[1], c_2[2], c_3[3], \boldsymbol{c_4}; p).C' \mid (h_1[1]; p).H'_1 \mid (h_2[2]; p).H'_2 \\
&\mid (o_1[3], o_2[\mathbf{13}], \boldsymbol{n}).O'_1 \mid (h_3[12]; p).H'_3 \mid (h_4[6]; p).H'_4 \mid (o_3, o_4[6], n).O'_2 \\
&\mid (h_5[13]; p).H'_5 \mid (h_6[8]; p).H'_6 \mid (o_5[12], o_6[8], n).O'_3 \\
&\mid (h_7[9]; p).H'_7 \mid (h_8[10]; p).H'_8 \mid (o_7[9], o_8[10], n).O'_4 \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{i7} \mid \text{W} \mid \text{OH} \mid \text{W}
\end{aligned}
$$

The movement from i7 to i6 is gotten by creating the bond on c_4 and n of O_1 with the key 4:

$$
\begin{aligned}
&\xrightarrow{c_4 n[4]} (c_1[1], c_2[2], c_3[3], c_4[\mathbf{4}]; p).C' \mid (h_1[1]; p).H'_1 \mid (h_2[2]; p).H'_2 \\
&\mid (o_1[3], o_2[13], n[\mathbf{4}]).O'_1 \mid (h_3[12]; p).H'_3 \mid (h_4[6]; p).H'_4 \mid (o_3, o_4[6], n).O'_2 \\
&\mid (h_5[13]; p).H'_5 \mid (h_6[8]; p).H'_6 \mid (o_5[12], o_6[8], n).O'_3 \\
&\mid (h_7[9]; p).H'_7 \mid (h_8[10]; p).H'_8 \mid (o_7[9], o_8[10], n).O'_4 \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{i6} \mid \text{W} \mid \text{OH} \mid \text{W}
\end{aligned}
$$

Note that the last compound is chemically equivalent to i6 | W | OH | W, and it is also identical syntactically to i6 | W | OH | W except the keys 4 and 13 are swapped in O_1.

We have noted before that the main path and the base-catalysed path form a diamond: we can get from FA | W | W | W to i3 | H_3O | W either via i2 | W | W or via FA | W | OH | H_3O. We underline that the resulting methanediol processes are syntactically identical. However, there is no such tight correspondence between methanediol in the main path and methanediol in the acid-catalysed path. In the main path, we bind a water molecule to the carbon atom, and then use another H_2O as a proton shuttle to move a hydrogen atom. The H_2O which is used as a proton shuttle is unchanged. In the acid-catalysed path, we bind a hydrogen atom to the formaldehyde first, bind a water molecule to it and then remove one of these hydrogen atoms and put it back on a water molecule. Hence, the two methanediol

processes are not written identically but they represent the same compound. We shall explain later how this equivalence might be formally defined.

5.3.4 Other paths

We now discuss the remaining reactions in Figure 5.6.

The compounds 13 and 14 in Figure 5.5 are of particular interest since they have only one path leading to and out of them. Firstly, we can get from i3 | H_3O | W to i13 | H_3O | H_3O as follows:

$$
\begin{aligned}
&(c_1[1], c_2[2], c_3[3], c_4[11]; p).C' \mid (h_1[1]; p).H_1' \mid (h_2[2]; p).H_2' \mid (o_1[3], o_2, n).O_1' \\
&\mid (h_3[12]; p).H_3' \mid (h_4[6]; p).H_4' \mid (o_3[11], o_4[6], n).O_2' \\
&\mid (h_5[7]; p).H_5' \mid (h_6[8]; p).H_6' \mid (o_5[7], o_6[8], n[12]).O_3' \\
&\mid (h_7[9]; p).H_7' \mid (h_8[10]; p).H_8' \mid (o_7[9], o_8[10], n).O_4'
\end{aligned}
$$

i3 | H_3O | W

$$
\begin{aligned}
&\xrightarrow{\{np[13], \underline{h_4 o_4}[6]\}} (c_1[1], c_2[2], c_3[3], c_4[11]; p).C' \mid (h_1[1]; p).H_1' \mid (h_2[2]; p).H_2' \\
&\mid (o_1[3], o_2, n).O_1' \mid (h_3[12]; p).H_3' \mid (\boldsymbol{h_4}; p[\mathbf{13}]).H_4' \mid (o_3[11], \boldsymbol{o_4}, n).O_2' \\
&\mid (h_5[7]; p).H_5' \mid (h_6[8]; p).H_6' \mid (o_5[7], o_6[8], n[12]).O_3' \\
&\mid (h_7[9]; p).H_7' \mid (h_8[10]; p).H_8' \mid (o_7[9], o_8[10], n[\mathbf{13}]).O_4'
\end{aligned}
$$

$$
\begin{aligned}
&\Rightarrow (c_1[1], c_2[2], c_3[3], c_4[11]; p).C' \mid (h_1[1]; p).H_1' \mid (h_2[2]; p).H_2' \\
&\mid (o_1[3], o_2, n).O_1' \mid (h_3[12]; p).H_3' \mid (h_4[\mathbf{13}]; \boldsymbol{p}).H_4' \mid (o_3[11], o_4, n).O_2' \\
&\mid (h_5[7]; p).H_5' \mid (h_6[8]; p).H_6' \mid (o_5[7], o_6[8], n[12]).O_3' \\
&\mid (h_7[9]; p).H_7' \mid (h_8[10]; p).H_8' \mid (o_7[9], o_8[10], n[13]).O_4'
\end{aligned}
$$

i13 | H_3O | H_3O

We can also reverse back to i3 | H_3O | W by performing concerted actions followed by a promotion and a move:

$$\xrightarrow{\{np[6],\underline{h_4 n}[13]\}}(c_1[1], c_2[2], c_3[3], c_4[11]; p).C' \mid (h_1[1]; p).H_1' \mid (h_2[2]; p).H_2'$$
$$\mid (o_1[3], o_2, n).O_1' \mid (h_3[12]; p).H_3' \mid (\boldsymbol{h_4}; p[\mathbf{6}]).H_4' \mid (o_3[11], o_4, n[\mathbf{6}]).O_2'$$
$$\mid (h_5[7]; p).H_5' \mid (h_6[8]; p).H_6' \mid (o_5[7], o_6[8], n[12]).O_3'$$
$$\mid (h_7[9]; p).H_7' \mid (h_8[10]; p).H_8' \mid (o_7[9], o_8[10], \boldsymbol{n}).O_4'$$

$$\Rightarrow (c_1[1], c_2[2], c_3[3], c_4[11]; p).C' \mid (h_1[1]; p).H_1' \mid (h_2[2]; p).H_2'$$
$$\mid (o_1[3], o_2, n).O_1' \mid (h_3[12]; p).H_3' \mid (h_4[\mathbf{6}]; \boldsymbol{p}).H_4' \mid (o_3[11], o_4[\mathbf{6}], \boldsymbol{n}).O_2'$$
$$\mid (h_5[7]; p).H_5' \mid (h_6[8]; p).H_6' \mid (o_5[7], o_6[8], n[12]).O_3'$$
$$\mid (h_7[9]; p).H_7' \mid (h_8[10]; p).H_8' \mid (o_3[9], o_8[10], n).O_4'$$

i3 | H_3O | W

Furthermore, we can also get from i8 | OH | W to i14 | OH | OH:

$$(c_1[1], c_2[2], c_3[3], c_4[1]; p]).C' \mid (h_1[1]; p).H_1' \mid (h_2[2]; p).H_2'$$
$$\mid (o_1[3], o_2[13], n).O_1' \mid (h_3[12]; p).H_3' \mid (h_4[6]; p).H_4' \mid (o_3, o_4[6], n).O_2'$$
$$\mid (h_5[13]; p).H_5' \mid (h_6[8]; p).H_6' \mid (o_5[12], o_6[8], n[11]).O_3'$$
$$\mid (h_7[9]; p).H_7' \mid (h_8[10]; p).H_8' \mid (o_7[9], o_8[10], n).O_4'$$

i8 | OH | W

$$\xrightarrow{\{np[14],\underline{h_7 o_7}[9]\}} (c_1[1], c_2[2], c_3[3], c_4[11]; p).C' \mid (h_1[1]; p).H_1' \mid (h_2[2]; p).H_2'$$
$$\mid (o_1[3], o_2[13], n[\mathbf{14}]).O_1' \mid (h_3[12]; p).H_3' \mid (h_4[6]; p).H_4' \mid (o_3, o_4[6], n).O_2'$$
$$\mid (h_5[13]; p).H_5' \mid (h_6[8]; p).H_6' \mid (o_5[12], o_6[8], n[11]).O_3'$$
$$\mid (\boldsymbol{h_7}; p[\mathbf{14}]).H_7' \mid (h_8[10]; p).H_8' \mid (\boldsymbol{o_7}, o_8[10], n).O_4'$$

$$\Rightarrow (c_1[1], c_2[2], c_3[3], c_4[11]; p).C' \mid (h_1[1]; p).H_1' \mid (h_2[2]; p).H_2'$$
$$\mid (o_1[3], o_2[13], n[14]).O_1' \mid (h_3[12]; p).H_3' \mid (h_4[6]; p).H_4' \mid (o_3, o_4[6], n).O_2'$$
$$\mid (h_5[13]; p).H_5' \mid (h_6[8]; p).H_6' \mid (o_5[12], o_6[8], n[11]).O_3'$$
$$\mid (h_7[\mathbf{14}]; \boldsymbol{p}).H_7' \mid (h_8[10]; p).H_8' \mid (o_7, o_8[10], n).O_4'$$

i14 | OH | OH

And we can reverse back to i8 | OH | W in very much the same way as from i13 | H_3O | H_3O to i3 | H_3O | W.

The reaction from i6 | W | OH | W to i6 | OH | OH | H_3O and its inverse is the autoprotolysis of water, and works independently of the rest. The corresponding applies to the reaction from i7 | OH | OH | H_3O to i7 | W | OH | W and its reversal.

The step from i6 | OH | OH | H_3O to i7 | OH | OH | H_3O involves the intermediates i6 and i8 only (as in Figure 5.3) and no other compounds, so the reaction is as described before.

The reaction from i2 | W | W to i2 | OH | H_3OW is also an autoprotolysis of water.

There are several more reactions possible, all via concerted actions. We only describe one of them, namely from FA | W | W | W to i6 | W | OH | W:

$$
\begin{aligned}
&(c_1[1], c_2[2], c_3[3], c_4[4]; p).C' \mid (h_1[1]; p).H'_1 \mid (h_2[2]; p).H'_2 \\
&\mid (o_1[3], o_2[4], n).O'_1 \mid (h_3[5]; p).H'_3 \mid (h_4[6]; p).H'_4 \mid (o_3[5], o_4[6], n).O'_2 \\
&\mid (h_5[7]; p).H'_5 \mid (h_6[8]; p).H'_6 \mid (o_5[7], o_6[8], n).O'_3 \\
&\mid (h_7[9]; p).H'_7 \mid (h_8[10]; p).H'_8 \mid (o_7[9], o_8[10], n).O'_4
\end{aligned}
$$

$$
\begin{aligned}
&\xrightarrow{\{np[13], \underline{h_5 o_5}[7]\}} (c_1[1], c_2[2], c_3[3], c_4[4]; p).C' \mid (h_1[1]; p).H'_1 \mid (h_2[2]; p).H'_2 \\
&\mid (o_1[3], o_2[4], n\mathbf{[13]}).O'_1 \mid (h_3[5]; p).H'_3 \mid (h_4[6]; p).H'_4 \mid (o_3[5], o_4[6], n).O'_2 \\
&\mid (\boldsymbol{h_5}; p\mathbf{[13]}).H'_5 \mid (h_6[8]; p).H'_6 \mid (\boldsymbol{o_5}, o_6[8], n).O'_3 \\
&\mid (h_7[9]; p).H'_7 \mid (h_8[10]; p).H'_8 \mid (o_7[9], o_8[10], n).O'_4
\end{aligned}
$$

$$
\begin{aligned}
&\Rightarrow (c_1[1], c_2[2], c_3[3], c_4[4]; p).C' \mid (h_1[1]; p).H'_1 \mid (h_2[2]; p).H'_2 \\
&\mid (o_1[3], o_2[4], n[13]).O'_1 \mid (h_3[5]; p).H'_3 \mid (h_4[6]; p).H'_4 \mid (o_3[5], o_4[6], n).O'_2 \\
&\mid (h_5\mathbf{[13]}; \boldsymbol{p}).H'_5 \mid (h_6[8]; p).H'_6 \mid (o_5, o_6[8], n).O'_3 \\
&\mid (h_7[9]; p).H'_7 \mid (h_8[10]; p).H'_8 \mid (o_7[9], o_8[10], n).O'_4
\end{aligned}
$$

i6 | W | OH | W

5.4 Conclusion

We have shown so far how CCB can be used with a more complicated example than the autoprotolysis of water used before in Chapter 2. We now evaluate the suitability

of CCB in the modelling of covalent reactions as exemplified by the hydration of formaldehyde in water. We have used the software described in Chapter 8 to perform the evaluation described here. We first consider if all chemically valid interactions between the compounds of the reaction can be represented well in our calculus. We have seen in Figure 5.3 the resonance between the intermediates i6 and i7. We are able to model appropriately the movement between i6 and i7 by transitions that break spontaneously a bond or create a bond. What is less clear is why this spontaneous break should happen at this point and not elsewhere. We are unable to represent the movement from i7 to i8, however, we model appropriately the evolution from i6 to i8 via a concerted actions transition. Apart from the reaction steps described above, all other steps can be represented well in our calculus.

The other way of assessing the suitability of CCB for this type of reactions is to ask if our calculus enables transitions which do not occur in reality. We notice that some bonds can break spontaneously as, for example, in water molecules. Although this does not happen in reality by itself the process (known as water splitting) can happen as part of photosynthesis or in various technical processes. So the spontaneous breaking of water molecules could be treated as harmless. A spontaneous breaking of a bond can also happen in the methanediol MD, where the bonds between carbon atom and oxygen atom could break. This is a consequence of allowing the spontaneous break of the double bond in the compound i6. The difference between these two cases is not properly modelled in our calculus. However, if the carbon-oxygen bond in MD breaks, the carbon atom could bond immediately afterwards to the water molecule, and we would get the compound i8, which is a valid intermediate, so this break of the carbon-oxygen bond does not cause a problem.

Our model also allows the interaction of the methanediol with a water molecule in the final system MD | W | W. Since the carbon process C has the action p ready, and the oxygen process O_3 in the water molecule has the action n ready, and n, p is a part of our synchronisation function, a bond between C and O_3 can be created[4]. The reason why this type of interaction does not occur in reality is that the four groups around the carbon atom shield it from any interaction (as opposed to the three groups on the formaldehyde, of which two are relatively small hydrogen atoms). This is a *steric* effect, the effect due to atoms occupying space and preventing other atoms from moving. Our calculus does not model spatial arrangement of atoms well enough.

[4]We have actually used a similar mechanism of C interacting with O to get from compound 1 to compound 2 in Figure 5.2.

Finally, there is one more case which is possible in our calculus, namely the transfer of a proton from a OH^- molecule to an electronegative atom. Since the p action is still ready on the H in OH^- any other n action can interact. In reality OH^- is no longer polar, because the formal negative charge cancels out any partial charge. We could say that there is no p on the hydrogen process any more.

Overall, our calculus gives a good modelling of the example. We have one transition we cannot properly model (the resonance) and we have two transitions which can happen in our model, but not in reality (the carbon atom with four groups reacting and OH^- losing a proton). Furthermore the spontaneous break of the carbon-oxygen bond in methanediol is problematic. Considering the number of correctly modelled reactions, we can say that we model most reactions possible correctly and the most reactions modelled are correct.

We will informally discuss an extension of our calculus to solve these problems in Chapter 9.

Chapter 6

Chemical process equivalence

In Chapter 5 we have seen that the system of FA (formaldehyde molecule) with three W (water molecules) evolves to MD (methanediol molecule) with two W (water molecules) via different sequences of concerted actions. Some paths end up at the same process, others lead to somewhat syntactically different processes. We have explained that these different processes represent the same compounds. We also have explained that the difference originates as a side effect of using subscripted copies of atoms, where both the action names and process names are uniformly subscripted. Also keys can lead to differences since they can be chosen arbitrarily. We now present how we can formally define the equivalence mentioned here, which is referred to as *chemical process equivalence*.

We have chosen this term to indicate that this is an equivalence on processes, but inspired by a chemical notation. It should be mentioned that "chemical equivalence" is used by chemists to describe atoms inside a molecule which have the same electronic environment and are therefore identical for reactions.[1] We have already seen this: For example in the autoprotolysis of water in Chapter 2 the two hydrogen atoms of a water molecule are chemically equivalent. When we looked at the hydration of formaldehyde in Chapter 5 we implicitly used chemical equivalence when discussing the various paths. It is still possible to explore which path a reaction actually proceeds with the help of isotope labelling. For this a particular atom in a molecule is replaced by one if its isotopes. Since isotopes are electronically identical it will not change the reaction, but the isotope can be identified in the result or

[1] Since spectroscopic methods are used to elucidate chemical structures and are influenced by chemical equivalence the concept is typically discussed in this context, see [54], p. 51–52.

intermediates of the reaction and the atom can be traced in this way through the reaction.[2]

In Chapter 5 we have added the subscripts to actions and process names so that they started with 1 for each action or process name. Whilst this is easy to read it makes handling the subscripts by functions working on the whole process difficult. We will therefore use unique subscripts in all of the processes we deal with. To make understanding easier we adopt the convention to use multiples of 10 as subscripts for process names of subprocesses representing hydrogens (e.g. H_{10}), multiples of 100 for the oxygen processes (e.g. O_{100}) and multiples of 100 plus 1000 for the carbon processes (e.g. C_{1000}). Other processes could be subscripted by multiples of 100 plus 2000 and so on. Subscripts for actions are starting with the process name's identifier plus 1 (e.g. $(h_{11}).H_{10}$). In this way, we have unique subscripts and can easily identify which subprocess an actions belongs to.

6.1 Definition of chemical process equivalence

The first step for developing a definition of equivalence is to define α *conversion* on subscripts and keys in processes. Let $\mathcal{S}$ be a set of subscripts, taken from $\mathbb{N}$. $\mathcal{A}$ and $\mathcal{PI}$ (defined in Section 4.1) are now keys and constants with a subscript, otherwise the syntax of the calculus is as defined in Section 4.1. The notion of α conversion of a key k in P is defined as follows:

Definition 6.1. Assume P is a consistent process, and k and l are keys. We define α conversion of k into l in P by function $\alpha\mathsf{keys}$ in Figure 6.1 if $\mathsf{fsh}[l](P)$, otherwise it is is undefined.

The notion of α conversion of a subscript k in P is defined as follows:

Definition 6.2. Assume P is a consistent process, and k and l are subscripts. We define α conversion of k into l in P by function $\alpha\mathsf{subscripts}$ in Figure 6.2 if $\mathsf{fshsubscript}[l](P)$, otherwise it is undefined.

Note that the function $\alpha\mathsf{subscripts}$ also converts subscripts in restrictions. For α conversion of keys we need the helper function $\alpha\mathsf{k}$, which is given in Figure 6.1. Otherwise all functions and predicates used for α conversion of keys are defined in Chapter 4. For α conversion of subscripts we need helper functions $\alpha\mathsf{s}$, $\alpha\mathsf{sr}$, $\alpha\mathsf{sc}$,

[2]For isotope labelling see [13] p. 752.

$$
\begin{aligned}
&\alpha\mathsf{keys}(\mathbf{0}, k, l) = \mathbf{0}\\
&\alpha\mathsf{keys}(S, k, l) = \alpha\mathsf{keys}(P, l, l) \quad \text{if } S \stackrel{def}{=} P\\
&\alpha\mathsf{keys}((s; b).P, k, l) = (\alpha\mathsf{k}(s, k, l); \alpha\mathsf{k}(b, k, l)).\alpha\mathsf{keys}(P, k, l)\\
&\alpha\mathsf{keys}(P \mid Q, k, l) = \alpha\mathsf{keys}(P, k, l) \mid \alpha\mathsf{keys}(Q, k, l)\\
&\alpha\mathsf{keys}(P\backslash L, k, l) = \alpha\mathsf{keys}(P, k, l)\backslash L
\end{aligned}
$$

$$
\begin{aligned}
&\alpha\mathsf{k} : (\mathcal{A} \cup \mathcal{AK})^* \times \mathcal{K} \times \mathcal{K} \to (\mathcal{A} \cup \mathcal{AK})^*\\
&\alpha\mathsf{k}(\epsilon, k, l) = \epsilon\\
&\alpha\mathsf{k}(\alpha : s, k, l) = \begin{cases} a[l] : \alpha\mathsf{k}(s, k, l) & \text{if } \alpha = a[k], a \in \mathcal{A}\\ \alpha : \alpha\mathsf{k}(s, k, l) & \text{otherwise} \end{cases}
\end{aligned}
$$

Figure 6.1: Function $\alpha\mathsf{keys}$ defining α conversion of key k into key l in CCB and helper function $\alpha\mathsf{k}$, with $k, l \in \mathcal{K}$.

and $\alpha\mathsf{k}$, which are given in Figure 6.2. We also need some new predicates and functions presented in Figure 6.3. Here, similarly to having defined freshness of a key in Figure 4.1, we define freshness of a subscript.

Example 6.1. We consider the first water molecule from Chapter 2. To recall this was modelled as:

$$P \stackrel{def}{=} (((h_{11}[1]; p).H'_{10} \mid (h_{21}[2]; p).H'_{20} \mid (o_{101}[1], o_{102}[2], n).O'_{100}) \setminus \{h_{11}, h_{12}, o_{101}, o_{102}\}$$

We can apply α conversion of keys to P. For example we can change key 1 to 5, which is fresh in P. Performing $\alpha\mathsf{keys}(P, 1, 5)$ results in:

$$(((h_{11}[5]; p).H'_{10} \mid (h_{21}[2]; p).H'_{20} \mid (o_{101}[5], o_{102}[2], n).O'_{100}) \setminus \{h_{11}, h_{12}, o_{101}, o_{102}\}$$

We could also change a subscript. Since 15 is a fresh subscript we could apply $\alpha\mathsf{subscripts}(P, 11, 15)$ resulting in:

$$(((h_{15}[1]; p).H'_{10} \mid (h_{21}[2]; p).H'_{20} \mid (o_{101}[1], o_{102}[2], n).O'_{100}) \setminus \{h_{15}, h_{12}, o_{101}, o_{102}\}$$

Notice the restriction has been converted as well.

Finally, we could change a subscript on a process name by applying $\alpha\mathsf{subscripts}(P, 10, 30)$ and we get:

$$(((h_{11}[1]; p).H'_{30} \mid (h_{21}[2]; p).H'_{20} \mid (o_{101}[1], o_{102}[2], n).O'_{100}) \setminus \{h_{11}, h_{12}, o_{101}, o_{102}\}$$

We note that this breaks our convention about subscripts: The process H'_{30} has an action h_{11}, whereas we originally assigned them so that subscripts for actions are

$$
\begin{aligned}
&\alpha\mathsf{subscripts}(\mathbf{0}, k, l) = \mathbf{0}\\
&\alpha\mathsf{subscripts}(S, k, l) = \alpha\mathsf{sc}(S, k, l)\\
&\alpha\mathsf{subscripts}((s; b).P, k, l) = (\alpha\mathsf{s}(s, k, l); \alpha\mathsf{s}(b, k, l)).\alpha\mathsf{subscripts}(P, k, l)\\
&\alpha\mathsf{subscripts}(P \mid Q, k, l) = \alpha\mathsf{subscripts}(P, k, l) \mid \alpha\mathsf{subscripts}(Q, k, l)\\
&\alpha\mathsf{subscripts}(P\backslash L, k, l) = \alpha\mathsf{subscripts}(P, k, l)\backslash\alpha\mathsf{sr}(L, k, l)
\end{aligned}
$$

$$
\begin{aligned}
&\alpha\mathsf{s} : (\mathcal{A} \cup \mathcal{AK})^* \times \mathcal{S} \times \mathcal{S} \to (\mathcal{A} \cup \mathcal{AK})^*\\
&\alpha\mathsf{s}(\epsilon, k, l) = \epsilon\\
&\alpha\mathsf{s}(\alpha : s, k, l) = \begin{cases} a_l[x] : \alpha\mathsf{s}(s, k, l) & \text{if } \alpha = a_k[x], a \in \mathcal{A}\\ a_l : \alpha\mathsf{s}(s, k, l) & \text{if } \alpha = a_k, \alpha \in \mathcal{A}\\ \alpha : \alpha\mathsf{s}(s, k, l) & \text{otherwise}\end{cases}\\
&\alpha\mathsf{sr} :\subseteq \mathcal{A} \times \mathcal{S} \times \mathcal{S} \to\subseteq \mathcal{A}\\
&\alpha\mathsf{sr}(\epsilon, k, l) = \epsilon\\
&\alpha\mathsf{sr}(\{\alpha\} \cup L, k, l) = \begin{cases} \{a_l\} \cup \alpha\mathsf{sr}(L, k, l) & \text{if } \alpha = a_k, \alpha \in \mathcal{A}\\ \{\alpha\} \cup \alpha\mathsf{sr}(L, k, l) & \text{otherwise}\end{cases}
\end{aligned}
$$

$$
\begin{aligned}
&\alpha\mathsf{sc} : \mathcal{PI} \times \mathcal{S} \times \mathcal{S} \to \mathcal{PI}\\
&\alpha\mathsf{sc}(\alpha, k, l) = \begin{cases} S_l & \text{if } \alpha = S_k, \alpha \in \mathcal{PI}\\ \alpha & \text{otherwise}\end{cases}
\end{aligned}
$$

Figure 6.2: Function αsubscripts defining α conversion of subscript k into subscript l in CCB and helper function αs, αsr, and αsc, with $k, l \in \mathcal{S}$.

starting with the process name's identifier plus 1. This means that process H'_{30} should have action h_{31}. We get the differently numbered action since this rule was a convention only and is not enforced by the α conversion rules.

Having defined α conversion, we can now define swapping keys or subscripts between processes. First we define the function swapkeys for swapping keys in a process P:

Definition 6.3. Assume P is a consistent process, k, l, and m are keys and $l, m \notin \mathsf{keys}(P)$. We define swapping of k and l in P by function $\mathsf{swapkeys} : \mathsf{Proc} \times \mathcal{K} \times \mathcal{K} \to \mathsf{Proc}$, which is given as follows:

$$\mathsf{swapkeys}(P, k, l) \stackrel{def}{=} \alpha\mathsf{keys}(\alpha\mathsf{keys}(\alpha\mathsf{keys}(P, k, m), l, k), m, l)$$

We then define the function swapsubscripts for swapping subscripts in a process:

$$\mathsf{s} : (\mathcal{A} \cup \mathcal{AK})^* \to \subseteq \mathcal{S}$$
$$\mathsf{s}(\epsilon) = \emptyset$$
$$\mathsf{s}(\alpha : s) = \begin{cases} \{l\} \cup \mathsf{s}(s) & \text{if } \alpha = a_l[x] \text{ or } \alpha = a_l, a \in \mathcal{A}, l \in \mathcal{S}, x \in \mathcal{K} \\ \mathsf{s}(s) & \text{otherwise} \end{cases}$$

$$\frac{}{\mathsf{fshsubscript}[m](\mathbf{0})} \qquad \frac{\mathsf{fshsubscript}[m](P)}{\mathsf{fshsubscript}[m](S)} \quad S \stackrel{def}{=} P$$

$$\frac{m \notin \mathsf{s}(s) \quad \mathsf{fshsubscript}[m](P)}{\mathsf{fshsubscript}[m]((s;b).P)} \qquad \frac{m \notin \mathsf{s}(s) \quad m \neq n \quad \mathsf{fshsubscript}[m](P)}{\mathsf{fshsubscript}[m]((s;b[n]).P)}$$

$$\frac{\mathsf{fshsubscript}[m](P) \quad \mathsf{fshsubscript}[m](Q)}{\mathsf{fshsubscript}[m](P \mid Q)} \qquad \frac{\mathsf{fshsubscript}[m](P)}{\mathsf{fshsubscript}[m](P \setminus L)}$$

Figure 6.3: Function s and predicate $\mathsf{freshsubscript}$.

Definition 6.4. Assume P is a consistent process, k, l, and m are subscripts, $k \in \mathsf{s}(P)$, and $l \notin \mathsf{s}(P)$, $m \notin \mathsf{s}(P)$. We define swapping of k and l in P by function $\mathsf{swapsubscripts} : \mathsf{Proc} \times \mathcal{S} \times \mathcal{S} \to \mathsf{Proc}$, which is given as follows:

$$\mathsf{swapsubscripts}(P, k, l) \stackrel{def}{=}$$
$$\mathsf{alphaconversionsub}(\mathsf{alphaconversionsub}(\mathsf{alphaconversionsub}(P, k, m), l, k), m, l)$$

Example 6.2. We again consider the water molecule from Chapter 2:

$$P \stackrel{def}{=} (((h_{11}[1];p).H'_{10} \mid (h_{21}[2];p).H'_{20} \mid (o_{101}[1], o_{102}[2], n).O'_{100}) \setminus \{h_{11}, h_{12}, o_{101}, o_{102}\}$$

We can swap keys 1 and 2 in P. Application of $\mathsf{swapkeys}(P, 1, 2)$ yields

$$(((h_{11}[2];p).H'_{10} \mid (h_{21}[1];p).H'_{20} \mid (o_{101}[2], o_{102}[1], n).O'_{100}) \setminus \{h_{11}, h_{12}, o_{101}, o_{102}\}$$

Since we have made our processes identifiable by using the H_x notation we can see that we swapped the keys between the two hydrogen processes. Considering that the order of the parallel processes within P is irrelevant[3] we could not distinguish the two hydrogen processes otherwise. So what we have effectively achieved is a swap of chemically equivalent atoms.

The same effect could be achieved by applying $\mathsf{swapsubscripts}(P, 11, 21)$ and $\mathsf{swapsubscripts}(P, 10, 20)$, which gives:

$$(((h_{21}[1];p).H'_{20} \mid (h_{11}[2];p).H'_{10} \mid (o_{101}[1], o_{102}[2], n).O'_{100}) \setminus \{h_{11}, h_{12}, o_{101}, o_{102}\}$$

[3]Note rule $\mathsf{red1}$ in Figure 4.7.

If we rearrange the processes H'_{20} and H'_{10} we get the identical process as before, where the H'_{10} process is linked to o_{102} and the H'_{20} process is linked to o_{101}.

We can now define chemical process equivalence:

Definition 6.5. Let P and Q be consistent CCB processes. P is chemically process equivalent to Q, written as $P \approx Q$, if P can be rewritten to Q by the repeated application of swapkeys, swapsubscripts, αsubscripts, and αkeys in some order.

The functions swapkeys, swapsubscripts, αsubscripts, and αkeys have been implemented in the software described in Chapter 8.

6.2 Properties of the equivalence relation

In this section we establish several properties of the functions swapkeys, swapsubscripts, αsubscripts, and αkeys, that are used to define chemical process equivalence $\approx$.

Clearly an α conversion on a key or subscript which is fresh does not have any effect:

Proposition 6.1. *Let P be a consistent CCB process and $k, l \in \mathcal{K}$. If* $\mathsf{fsh}[k](P)$ *then* $\alpha\mathsf{keys}(P, k, l) = P$.

Note that $\mathsf{fsh}[l](P)$ is assumed to be true by Definition 6.1, which makes the proposition obvious.

Proposition 6.2. *Let P be a consistent CCB process and $k, l \in \mathcal{S}$. If* $\mathsf{fshsubscript}[k](P)$ *then* $\alpha\mathsf{subscripts}(P, k, l) = P$.

Note that $\mathsf{fshsubscript}[l](P)$ is assumed to be true by Definition 6.2.

The order of execution of two α conversions is not relevant if either all old and new keys are different or the two swap operations are identical:

Proposition 6.3. *Let P be a consistent CCB process and $k, l \in \mathcal{K}$. If $(k \neq m \wedge l \neq m \wedge n \neq k) \vee (k = m \wedge l = n)$ then* $\alpha\mathsf{keys}(\alpha\mathsf{keys}(P, k, l), m, n) = \alpha\mathsf{keys}(\alpha\mathsf{keys}(P, m, n), k, l)$.

Proof. $\mathsf{fsh}[n](P)$ and $\mathsf{fsh}[l](P)$ are assumed to be true by Definition 6.1. If $k \neq m$ then the subsets of $\mathcal{AK}$ affected by $\alpha\mathsf{keys}(X, k, l)$ and $\alpha\mathsf{keys}(X, m, n)$ operations,

applied to the same process, are disjoint. If $l \neq m \wedge n \neq k$ as well, then none of the actions changed by one operation will be changed by the other operation. Therefore, the two operations are independent and can be swapped. On the other hand, if $k = m$ the two operations affect the same subsets of $\mathcal{AK}$, so only the operation executed first will have an effect. If they change the keys to different keys the result will be different, so only if $l = n$ is true as well the overall result will hold.

The corresponding holds for α conversion of subscripts:

Proposition 6.4. *If* $(k \neq m \wedge l \neq m \wedge n \neq k) \vee (k = m \wedge l = n)$ *then* $\alpha\mathsf{subscripts}(\alpha\mathsf{subscripts}(P, k, l), m, n) = \alpha\mathsf{subscripts}(\alpha\mathsf{subscripts}(P, m, n), k, l)$.

The reasoning is similar to that in the proof of Proposition 6.3, with $\mathsf{fshsubscript}[n](P)$ and $\mathsf{fshsubscript}[l](P)$ assumed to be true by Definition 6.2.

Function $\alpha\mathsf{keys}$ is transitive:

Proposition 6.5. *Let* P *be a consistent CCB process and* $k, l \in \mathcal{K}$. *Then* $\alpha\mathsf{keys}(\alpha\mathsf{keys}(P, k, l), l, m) = \alpha\mathsf{keys}(P, k, m)$

Proof. Since $\mathsf{fsh}[l](P)$, the subset of $\mathcal{AK}$ affected by $\alpha\mathsf{keys}(\alpha\mathsf{keys}(P, k, l), l, m)$ will be the same as that affected by $\alpha\mathsf{keys}(P, k, l)$. So the first operation changes a set of keys from k to l and exactly these will be changed by the second operation from l to m. This is equivalent to changing k to m directly.

The same holds for α conversion of subscripts:

Proposition 6.6. *Let* P *be a consistent CCB process and* $k, l \in \mathcal{K}$. *Then* $\alpha\mathsf{subscripts}(\alpha\mathsf{subscripts}(P, k, l), l, m) = \alpha\mathsf{subscripts}(P, k, m)$

The reasoning is as in the proof of Proposition 6.5.

6.3 Chemical process equivalence in the hydration of formaldehyde

We use the example from Section 5.3 to demonstrate the usefulness of the equivalence. If we consider the path in Figure 5.7 via $FA \mid W \mid HO \mid H_3O$ and $i3 \mid H3O \mid W$, we get the following process:

$$P_1 \stackrel{def}{=} (c_{1001}[1], c_{1002}[2], c_{1003}[3], c_{1004}[11]; p).C'_{1000} \mid (h_{11}[1]; p).H'_{10} \mid (h_{21}[2]; p).H'_{20}$$
$$\mid (o_{101}[3], o_{102}[13], n).O'_{100} \mid (o_{201}[11], o_{202}[6], n).O'_{200} \mid (h_{31}[13]; p).H'_{30} \mid (h_{41}[6]; p).H'_{40}$$
$$\mid (h_{51}[7]; p).H'_{50} \mid (h_{61}[8]; p).H'_{60} \mid (o_{301}[7], o_{302}[8], n).O'_{300}$$
$$\mid (h_{71}[9]; p).H'_{70} \mid (h_{81}[10]; p).H'_{80} \mid (o_{401}[9], o_{402}[10], n).O'_{400}$$

If we consider the path in Figure 5.7 via $FA \mid W \mid HO \mid H_3O$, $i6 \mid W \mid HO \mid W$, $i8 \mid HO \mid W$, and $MD \mid HO \mid H_3O$ respectively, we get the this process:

$$P_2 \stackrel{def}{=} (c_{1001}[1], c_{1002}[2], c_{1003}[3], c_{1004}[11]; p).C'_{1000} \mid (h_{11}[1]; p).H'_{10} \mid (h_{21}[2]; p).H'_{20}$$
$$\mid (o_{101}[3], o_{102}[13], n).O'_{100} \mid (o_{301}[12], o_{302}[11], n).O'_{300} \mid (h_{31}[12]; p).H'_{30} \mid (h_{51}[13]; p).H'_{50}$$
$$\mid (h_{41}[6]; p).H'_{40} \mid (h_{81}[5]; p]).H'_{80} \mid (o_{201}[5], o_{202}[6], n).O'_{200}$$
$$\mid (h_{71}[9]; p).H'_{70} \mid (h_{61}[10]; p).H'_{60} \mid (o_{401}[9], o_{402}[10], n).O'_{400}$$

The processes are shown with the restrictions dropped, the subscripts changed as explained at the beginning of this chapter, and arranged to show similarities as much as possible, by writing the methanediol in the first two lines and the two molecules of water in the following lines. Figure 6.4 shows P_1 and P_2. The matching parts are shaded. The remaining four hydrogen atoms and one oxygen atom and the bonds connected to them are different between the two processes.

If we disregard the naming of the atoms (which was done for convenience only) we get two identical graphs: two water (H_2O) molecules and one methanediol (CH_4O_2). If we employ subscripts to identify copies of the same atom, we can see that in the first case the bonds 6 and 11 are on the same oxygen atom, whereas in the second case they are on different oxygen atoms. As explained in Section 5.3 the difference is not because we did not do the "right" transition, but is systematic - it cannot be avoided by choosing different hydrogen processes for transfer or using other actions where possible. Chemical process equivalence shows that the results are equivalent.

Since the two processes above are identical from a chemical point of view, it should be possible to transform them into one another using (some of) the functions used

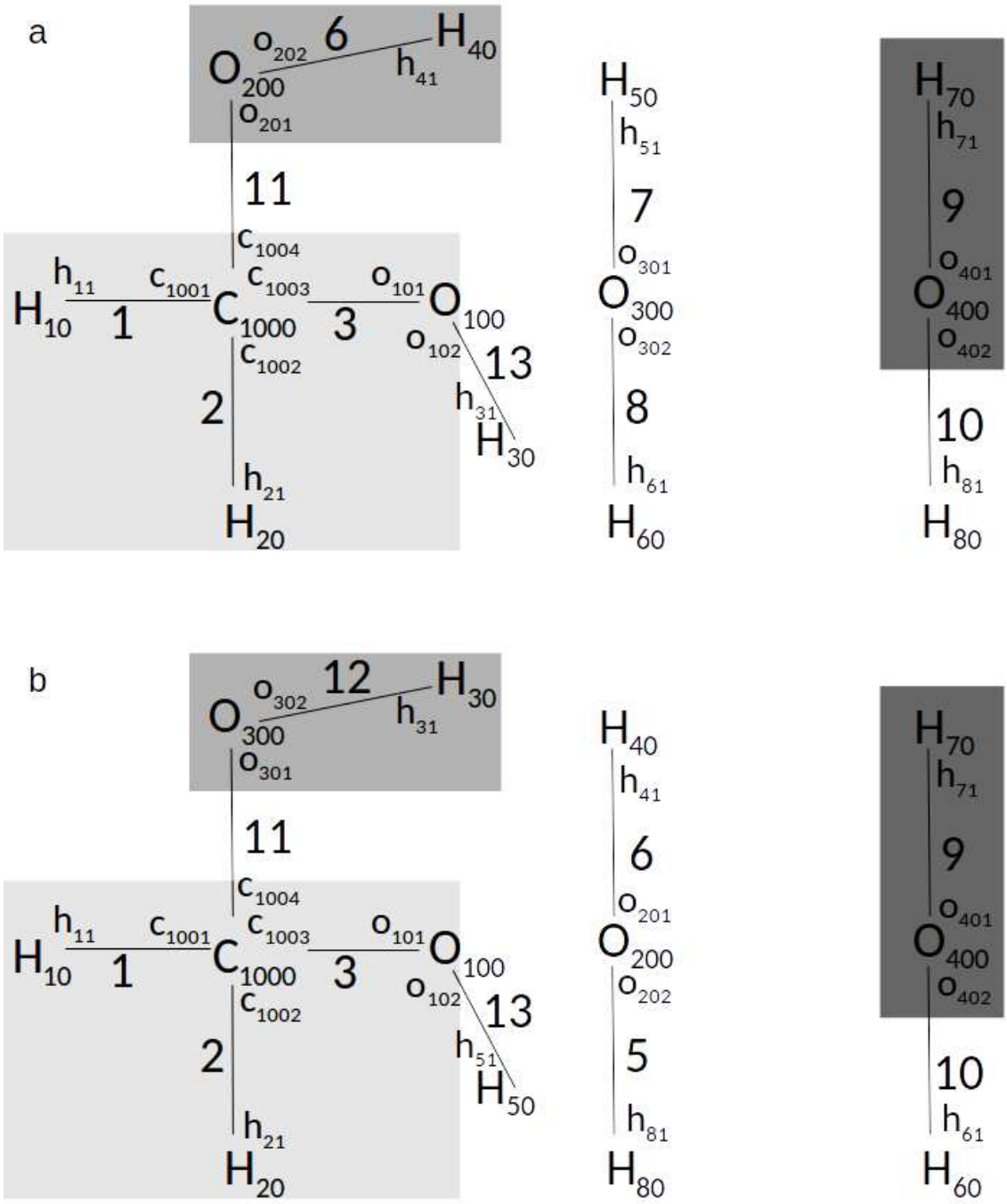

Figure 6.4: A full representation of the two processes P_1 and P_2, resulting from using the paths in Figure 5.7 via $FA \mid W \mid HO \mid H_3O$ and $i3 \mid H3O \mid W$, or via $FA \mid W \mid HO \mid H_3O$, $i6 \mid W \mid HO \mid W$, $i8 \mid HO \mid W$, and $MD \mid HO \mid H_3O$ respectively. The nodes are marked with process names, the actions are put around the process names and the communication keys are represented by lines with the key next to them. For example in P_1 C_{1000} represents the process with the process identifier C_{1000}, with the four actions $c_{1001-1004}$ arranged around it. Action c_{1001} shares key 1 with the action h_{11} on the process H_{10}. Matching parts in the two processes are shaded.

for defining chemical process equivalence. Starting with process P_2 (which we give below for convenience) the transformations are as follows:

$(c_{1001}[1], c_{1002}[2], c_{1003}[3], c_{1004}[11]; p).C'_{1000} \mid (h_{11}[1]; p).H'_{10} \mid (h_{21}[2]; p).H'_{20}$
$\mid (o_{101}[3], o_{102}[13], n).O'_{100} \mid (o_{301}[12], o_{302}[11], n).O'_{300} \mid (h_{31}[12]; p).H'_{30} \mid (h_{51}[13]; p).H'_{50}$
$\mid (h_{41}[6]; p).H'_{40} \mid (h_{81}[5]; p]).H'_{80} \mid (o_{201}[5], o_{202}[6], n).O'_{200}$
$\mid (h_{71}[9]; p).H'_{70} \mid (h_{61}[10]; p).H'_{60} \mid (o_{401}[9], o_{402}[10], n).O'_{400}$

We first swap subscripts 60 with 80 and 61 with 81 (this effectively swaps hydrogen processes H'_{60} and H'_{80}) by applying $\mathsf{swapsubscripts}(P_2, 60, 80)$ and $\mathsf{swapsubscripts}(P_2, 61, 81)$:

$(c_{1001}[1], c_{1002}[2], c_{1003}[3], c_{1004}[11]; p).C'_{1000} \mid (h_{11}[1]; p).H'_{10} \mid (h_{21}[2]; p).H'_{20}$
$\mid (o_{101}[3], o_{102}[13], n).O'_{100} \mid (o_{301}[12], o_{302}[11], n).O'_{300} \mid (h_{31}[12]; p).H'_{30} \mid (h_{51}[13]; p).H'_{50}$
$\mid (h_{41}[6]; p).H'_{40} \mid (h_{61}[5]; p]).H'_{60} \mid (o_{201}[5], o_{202}[6], n).O'_{200}$
$\mid (h_{71}[9]; p).H'_{70} \mid (h_{81}[10]; p).H'_{80} \mid (o_{401}[9], o_{402}[10], n).O'_{400}$

Then we swap subscripts 200 with 300, 201 with 301, and 202 with 302 (this effectively swaps oxygen processes O'_{200} and O'_{300}) by applying $\mathsf{swapsubscripts}(P_3, 200, 300)$, $\mathsf{swapsubscripts}(P_3, 201, 301)$, and $\mathsf{swapsubscripts}(P_3, 202, 302)$:

$(c_{1001}[1], c_{1002}[2], c_{1003}[3], c_{1004}[11]; p).C'_{1000} \mid (h_{11}[1]; p).H'_{10} \mid (h_{21}[2]; p).H'_{20}$
$\mid (o_{101}[3], o_{102}[13], n).O'_{100} \mid (o_{201}[12], o_{202}[11], n).O'_{200} \mid (h_{31}[12]; p).H'_{30} \mid (h_{51}[13]; p).H'_{50}$
$\mid (h_{41}[6]; p).H'_{40} \mid (h_{61}[5]; p]).H'_{60} \mid (o_{301}[5], o_{302}[6], n).O'_{300}$
$\mid (h_{71}[9]; p).H'_{70} \mid (h_{81}[10]; p).H'_{80} \mid (o_{401}[9], o_{402}[10], n).O'_{400}$

Next we swap subscripts 50 with 30 and 51 with 31 (this effectively swaps hydrogen processes H'_{30} and H'_{50}) by applying $\mathsf{swapsubscripts}(P_4, 30, 50)$ and $\mathsf{swapsubscripts}(P_4, 31, 51)$:

$(c_{1001}[1], c_{1002}[2], c_{1003}[3], c_{1004}[11]; p).C'_{1000} \mid (h_{11}[1]; p).H'_{10} \mid (h_{21}[2]; p).H'_{20}$
$\mid (o_{101}[3], o_{102}[13], n).O'_{100} \mid (o_{201}[12], o_{202}[11], n).O'_{200} \mid (h_{51}[12]; p).H'_{50} \mid (h_{31}[13]; p).H'_{30}$
$\mid (h_{41}[6]; p).H'_{40} \mid (h_{61}[5]; p]).H'_{60} \mid (o_{301}[5], o_{302}[6], n).O'_{300}$
$\mid (h_{71}[9]; p).H'_{70} \mid (h_{81}[10]; p).H'_{80} \mid (o_{401}[9], o_{402}[10], n).O'_{400}$

Finally we swap subscripts 40 with 50 and 41 with 51 (this effectively swaps hydrogen processes H'_{40} and H'_{50}) by applying $\mathsf{swapsubscripts}(P_5, 40, 50)$ and $\mathsf{swapsubscripts}(P_5, 41, 51)$:

$$(c_{1001}[1], c_{1002}[2], c_{1003}[3], c_{1004}[11]; p).C'_{1000} \mid (h_{11}[1]; p).H'_{10} \mid (h_{21}[2]; p).H'_{20}$$
$$\mid (o_{101}[3], o_{102}[13], n).O'_{100} \mid (o_{201}[12], o_{202}[11], n).O'_{200} \mid (h_{41}[12]; p).H'_{40} \mid (h_{31}[13]; p).H'_{30}$$
$$\mid (h_{51}[6]; p).H'_{50} \mid (h_{61}[5]; p]).H'_{60} \mid (o_{301}[5], o_{302}[6], n).O'_{300}$$
$$\mid (h_{71}[9]; p).H'_{70} \mid (h_{81}[10]; p).H'_{80} \mid (o_{401}[9], o_{402}[10], n).O'_{400}$$

Now we rearrange parallel processes (using rule $\mathsf{red1}$, namely $P \mid Q \Rightarrow Q \mid P$, which allows swapping parallel processes):

$$(c_{1001}[1], c_{1002}[2], c_{1003}[3], c_{1004}[11]; p).C'_{1000} \mid (h_{11}[1]; p).H'_{10} \mid (h_{21}[2]; p).H'_{20}$$
$$\mid (o_{101}[3], o_{102}[13], n).O'_{100} \mid (o_{201}[12], o_{202}[11], n).O'_{200} \mid (h_{31}[13]; p).H'_{30} \mid (h_{41}[12]; p).H'_{40}$$
$$\mid (h_{51}[6]; p).H'_{50} \mid (h_{61}[5]; p]).H'_{60} \mid (o_{301}[5], o_{302}[6], n).O'_{300}$$
$$\mid (h_{71}[9]; p).H'_{70} \mid (h_{81}[10]; p).H'_{80} \mid (o_{401}[9], o_{402}[10], n).O'_{400}$$

The next step is to change keys: Key 5 becomes 8, key 6 becomes 7, key 12 becomes 6 (by applying $\alpha\mathsf{keys}(\alpha\mathsf{keys}(\alpha\mathsf{keys}(P_7, 5, 8), 6, 7), 12, 6)$:

$$(c_{1001}[1], c_{1002}[2], c_{1003}[3], c_{1004}[11]; p).C'_{1000} \mid (h_{11}[1]; p).H'_{10} \mid (h_{21}[2]; p).H'_{20}$$
$$\mid (o_{101}[3], o_{102}[13], n).O'_{100} \mid (o_{201}[6], o_{202}[11], n).O'_{200} \mid (h_{31}[13]; p).H'_{30} \mid (h_{41}[6]; p).H'_{40}$$
$$\mid (h_{51}[7]; p).H'_{50} \mid (h_{61}[8]; p]).H'_{60} \mid (o_{301}[8], o_{302}[7], n).O'_{300}$$
$$\mid (h_{71}[9]; p).H'_{70} \mid (h_{81}[10]; p).H'_{80} \mid (o_{401}[9], o_{402}[10], n).O'_{400}$$

Finally we need to apply $\mathsf{swapsubscripts}(P_8, 201, 202)$ and $\mathsf{swapsubscripts}(P_8, 301, 302)$ to ensure the bonds inside O'_{200} and O'_{300} are on the same actions as in P_1:

$$(c_{1001}[1], c_{1002}[2], c_{1003}[3], c_{1004}[11]; p).C'_{1000} \mid (h_{11}[1]; p).H'_{10} \mid (h_{21}[2]; p).H'_{20}$$
$$\mid (o_{101}[3], o_{102}[13], n).O'_{100} \mid (o_{201}[11], o_{202}[6], n).O'_{200} \mid (h_{31}[13]; p).H'_{30} \mid (h_{41}[6]; p).H'_{40}$$
$$\mid (h_{51}[7]; p).H'_{50} \mid (h_{61}[8]; p]).H'_{60} \mid (o_{301}[7], o_{302}[8], n).O'_{300}$$
$$\mid (h_{71}[9]; p).H'_{70} \mid (h_{81}[10]; p).H'_{80} \mid (o_{401}[9], o_{402}[10], n).O'_{400}$$

The resulting process is exactly the same as the first process P_1. We have shown the equivalence of the results of following $FA \mid W \mid HO \mid H_3O$ and $i3 \mid H3O \mid W$, or via $FA \mid W \mid HO \mid H_3O$, $i6 \mid W \mid HO \mid W$, $i8 \mid HO \mid W$, and $MD \mid HO \mid H_3O$ in Figure 5.7. We did this by using rules $\alpha\mathsf{keys}$, $\mathsf{swapsubscripts}$, and $\mathsf{red1}$ to convert

P_2 into P_1. The correctness of this conversion has been verified using the software described in Chapter 8.

Remark 6.1. In Figure 6.4 we have used a conversion of processes into graph-like structures. This is possible since our processes are built by parallel composition of processes, each of them representing an atom (e.g. the process used in this section consists of hydrogen and oxygen atoms modelled in Section 2.1). The communications add keys to these, each communication linking two processes (atoms in our modelling). Since we used process names for the subprocesses, we can use process names as nodes and the communication keys as edges in Figure 6.4. In this figure we also give an implicit mapping of nodes and edges, forming an isomorphism between P_1 and P_2, where C_{1000} in P_1 maps to C_{1000} in P_2, O_{200} in P_1 maps to O_{300} in P_2, H_{40} in P_1 maps to H_{30} in P_2 and so forth. We will use these ideas in Chapter 8 to develop an automatic mapping between equivalent processes.

6.4 Behavioural equivalence

Chemical process equivalence, as defined before, is comparing processes. Since processes represent molecules in our application, it compares molecules. If the molecules are results of reactions, we can decide if different reaction paths lead to the same result or different molecules. Similar notions of equivalence are, for example, used in the λ-calculus ([22], p. 95).

In CCS, the most important notions of equivalence are bisimulations.[4] These are *behavioural*, two processes are considered equivalent if they show the same behaviour. This is a useful notation, for example, in software engineering, where it is important to assure that an implementation behaves like a specification, independently of its internal workings. In our examples, such an interpretation is not possible, since reaction steps are labelled with the names of the actions involved. These are a very localized aspect of a larger molecule and they would not normally be considered to indicate identical or equivalent molecules. A relevant chemical concept are for example functional groups ([13], pp. 390/391). A functional group is a specific substituent, which enables a certain reaction. Identical functional groups in different compounds typically enable the same reactions. Our notation is too localized to capture them, though. For example there are various functional groups containing an oxygen atom, where a carbon atom reacts with that oxygen atom. Our modelling

[4]There are different types of bisimulation, in particular the treatment of silent actions, which we do not use in CCB is different (see [79] Chapters 4 and 5 for bisimulations in CCB). For bisimulations in the π-calculus see [80] Chapters 12 and 13.

could only capture this oxygen-carbon interaction and would not have information about the environment, which would be needed for a meaningful comparison of the various functional groups.

We could still use behavioural aspects to define certain chemical properties. An example of this could be solubility ([13], p. 441). Substances may, if put into a solvent, form a homogeneous mixture with the solvent. Common solvents are, for example, water or methanol. Not every substance is soluble in every solvent, though. Various chemical and physical properties play a role here. In our calculus, if we have properly modelled both the solute and the solvent, we could put the molecules in parallel and compare the reactions possible to those within the solute and the solvent only. If they are equivalent, then the substance is insoluble in this solvent. If the combined processes representing molecules are not equivalent to the pure processes/molecules, there is some interaction. This interaction would indicate solubility. More generally speaking, this is an example of chemical context. We simulate the behaviour of a compound in various contexts. If there is no difference, this tells us something about the compound.

6.5 Conclusion

We have added subscripts to actions and process identifiers in Chapter 2. This has allowed us to distinguish actions or processes which could not be distinguished otherwise. If we use actions or processes which only differ by subscript we get syntactically different processes. However they indicate the same chemical structure, if we translate processes back into structures, by doing the reverse of what we did in Section 5.3. We have shown that using α conversion of subscripts and keys, we can convert such syntactically different processes into syntactically identical processes, thus giving a notion of chemical process equivalence in our calculus. Considering the example from Chapter 5, we can show the equivalence of all results of the model to processes which model what are considered different compounds by chemists. This shows that our modelling is realistic for this type of reciting networks. We have also suggested how a behavioural approach similar to bisimilarity in CCS could be used to model chemical concepts.

Chapter 7

Base Excision Repair

7.1 Description of Base Excision Repair

So far we have seen some applications of CCB, which are very close to the original inspiration of the calculus. If the principles behind CCB are of some general relevance, there should be other processes we can model using our calculus. A candidate for these might be biological processes. These are ultimately chemical reactions, but they are often viewed at a much higher level of abstraction then demonstrated so far. Typically, in such studies atoms would no longer play a rôle, but proteins and other macromolecules, consisting of thousands or more atoms are considered as entities. Typical examples are pathways, gene regulation, transcription, or DNA repair. One of the mechanisms for DNA repair is *base excision repair* (BER). Specifically, BER is responsible for repairing small damages in the DNA, where a single base pair is not correct. Such damages can be inflicted by processes in the body or various external factors like radiation. Repair of such damages is important to prevent a degradation of the DNA information in the organism. There are various subtypes of BER, and various proteins involved in it. For our purposes, we look at the case that a uracil base has been incorporated in the DNA. Uracil is normally only found in RNA, whereas DNA consists of the four bases adenine (A), cytosine (C), guanine (G), and thymine (T).

Uracil-DNA glycosylase (UNG or UDG) is the protein responsible for removing uracil from DNA and making the position available for insertion of the correct base. The process has been extensively studied[1] and modelled in [55] and [57]. A

[1] A good overview is given in [106]. We use a highly simplified view here. For example, there is not a single UDG, but a family of UDGs all exhibiting slightly different behaviour.

description of the process on an abstract level is as follows: UDG can bind to any of the deoxyribose/phosphate groups forming the backbone strands of the DNA. From there it can "walk" along the chain to the next deoxyribose/phosphate group (that walk makes it much more likely to find a damage than if UDG would just randomly bind to and get off the DNA strand again). If the base attached to this group is uracil, UDG will bind to it and dissolve the bond from the uracil to the DNA. Uracil can then be released and UDG can either continue the search or get off the DNA strand. The correct base can take the place of the uracil.

7.2 Modelling BER

In order to model BER, we need the following components: deoxyribose/phosphate groups, the UDG, the uracil and the four other bases. The other bases will not take part in the reactions, but include them in our model in order to demonstrate the interaction we. The components are the following:

$$\begin{array}{lcl} DP & \stackrel{def}{=} & (p3, p5, b, d).DP' \\ UDG & \stackrel{def}{=} & (h; f).(e).UDG' \\ U & \stackrel{def}{=} & (b; e).(u).U' \\ A & \stackrel{def}{=} & (b; i).(a).A' \\ T & \stackrel{def}{=} & (b; i).(t).T' \\ G & \stackrel{def}{=} & (b; i).(g).G' \\ C & \stackrel{def}{=} & (b; i).(c).C' \end{array}$$

where processes A, T, G, C, and U model the bases adenine (A), cytosine (C), guanine (G), thymine (T), and uracil (U) respectively. Process UDG represents the Uracil-DNA glycosylase and DP a deoxyribose/phosphate group. Here d, e, i, and f are weak actions, all other actions, namely $p3$, $p5$, h, b, u, a, t, g, and c are strong.

The synchronisation function for our system is as follows:

$$\begin{array}{lcl} \gamma(p3, p5) & = & p \\ \gamma(b, b) & = & bb \\ \gamma(a, t) & = & at \\ \gamma(g, c) & = & gc \\ \gamma(h, d) & = & hd \\ \gamma(f, d) & = & fd \\ \gamma(e, e) & = & ee \end{array}$$

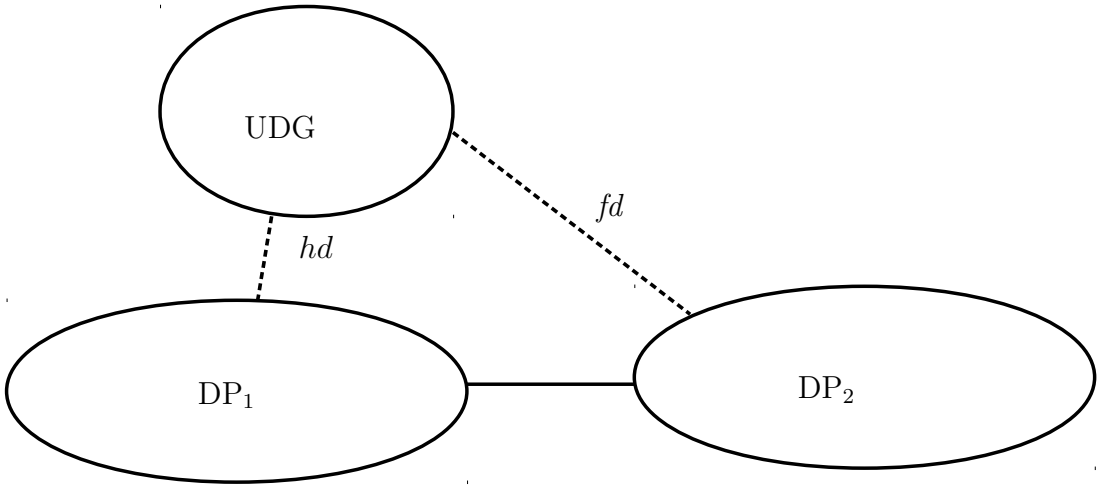

Figure 7.1: A *UDG* unit whilst performing a step along a DNA strand. The *hd* bond is broken together with the new *fd* being formed.

We model the deoxyribose/phosphate groups first. This has two ends, normally called 3' and 5', which we model as $p3$ and $p5$ actions respectively. They help us to build the DNA strands. Also, there is a b action, which enables binding of a base. Then we need a possibility for the *UDG* to bind and to "walk": this is enabled by actions d, h, and f, as we will see. *UDG* is modelled to have the prefix $(h; f)$, which enables the walk, since the strong action h can be bonded to a DP and f can interact with a neighbouring DP. This breaks the hd bond, and the fd bond is then promoted to d, which gives us the UDG being bonded to the neighbouring DP the same way it was bound before to the other DP. Figure 7.1 shows this intermediate situation whilst this step is performed. The five bases (C, G, T, A) all have a b action to bind to a DP. If $A - T$ respectively $C - G$ are opposite each other they can bind and therefore form a correct base pair in the DNA. Uracil (U) is not able to form a base pair in our DNA context. All bases have a weak action in the prefix $(b; x)$. This action x serves for removing the base from the DNA by breaking the b bond. For uracil the action x is e, for other bases it is i. By this UDG can be specific to uracil. In our model no action interacts with i, but of course other proteins not modelled might react with it. Note that i and e actions in the U, A, T, G, and C processes cannot happen if the u, a, t, g respctively c actions are done, so i and e are blocked by u, a, t, g, and c. Since u, a, t, g and c are used to form the base pairs this means that a correct base pair cannot be removed from the DNA in any case. This models the situation we need for the repair mechanism to work. We have used concerted actions in two instances here: Firstly to enable the UDG walk. This is

an instance where backtracking would not work, since we need out-of-causal-order reversibility in this case. We cannot unbond from the old DP first and then choose the next DP, but we must hold the old bond until the new bond is formed. Secondly, we use a concerted action to enable the repair mechanism by making a bonding on the repair action break the bond to DP.

In the synchronisation function, we have the interaction of $p3$ and $p5$ to form the strands, the b-b interaction for binding the bases, the h-d and f-d interaction for the "walk", the a-t and g-c interactions for forming the base pairs, and the e-e interaction for the repair action.

In order to model a strand of DNA, we restrict ourselves to three base pairs. This means we need six DP processes and six bases. We put two "correct" base pairs and one containing a uracil base. An extra C base must be available for replacing U. We also use subscripts to distinguish processes where there is more than one instance of the process. The system is modelled in CCB as follows:

$$(DP_1 \mid DP_2 \mid DP_3 \mid A \mid T \mid G_1 \mid G_2 \mid U \mid C_1 \mid C_2 \mid DP_4 \mid DP_5 \mid DP_6 \mid UDG) \backslash \{p3, p5, d, b, a, t, g, e, u, c, h, f, i\}$$

We leave out the restriction from now on for ease of reading. We number actions using subscripts where there is more than one instance, and set initial bonds as required. We get the following process:

$$(p3_1, p5_1[1], d_1, b_1[5]).DP'_1 \mid (p3_2[1], p5_2[3], d_2, b_2[4]).DP'_2 \mid (p3_3[3], p5_3, d_3, b_3[9]).DP'_3 \mid (b_1[5]; i_1).(a[6]).A' \mid (b_2[7]; i_2).(t[6]).T' \mid (b_3[8]; i_3).(g_1).G'_1 \mid ((b_4[9]; i_4).(g_2[10]).G'_2 \mid (b_5[4]; e_2).(u).U' \mid (b_6[11]; i_6).(c_1[10]).C' \mid (b_7; i_7).(c_2).C' \mid (p3_4, p5_4[12], d_4, b_4[7]).DP'_4 \mid (p3_5[12], p5_5[13], d_5, b_5[8]).DP'_5 \mid (p3_6[13], p5_6, d_6, b_6[11]).DP'_6 \mid (h; f).(e_2).UDG'$$

Initially UDG bonds to DP_1, which in turn is bonded to a correct base pair (note UDG was not bound so far to any other process). The situation is shown in Fig-

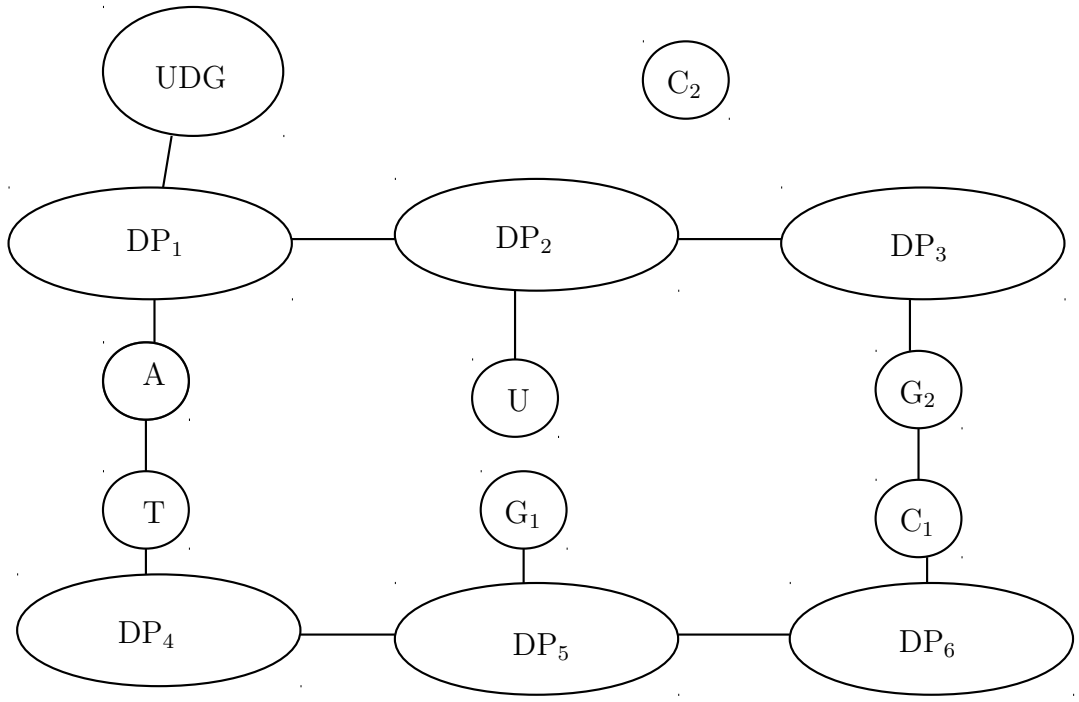

Figure 7.2: A three base pair DNA fragment, with a uracil instead of a cytosine, and a UDG protein attached.

ure 7.2 and is as follows (the bonds created, news keys and action on which keys were removed are shown in **bold**):

$\xrightarrow{hd[2]} (p3_1, p5_1[1], d_1[\mathbf{2}], b_1[5]).DP_1' \mid (p3_2[1], p5_2[3], d_2, b_2[4]).DP_2' \mid$
$(p3_3[3], p5_3, d_3, b_3[9]).DP_3' \mid (b_1[5]; i_1).(a[6]).A' \mid (b_2[7]; i_2).(t[6]).T' \mid (b_3[8]; i_3).(g_1).G_1' \mid$
$((b_4[9]; i_4).(g_2[10]).G_2' \mid (b_5[4]; e_2).(u).U' \mid (b_6[11]; i_6).(c_1[10]).C' \mid (b_7; i_7).(c_2).C' \mid$
$(p3_4, p5_4[12], d_4, b_4[7]).DP_4' \mid (p3_5[12], p5_5[13], d_5, b_5[8]).DP_5' \mid$
$(p3_6[13], p5_6, d_6, b_6[11]).DP_6' \mid (h[\mathbf{2}]; f).(e_2).UDG'$

The UDG can now randomly "walk" along the chain. The $(h; f)$ prefix can appropriately model this, since if the weak f action binds to the neighbour, the bond on h is broken. In our case, action f in UDG can communicate with d_2 in DP_2. This

breaks bond 2 from h in UDG to d_1 in DP_1, thus having performed a "step". We then move key 14 from f to h (via the prom rule from Figure 4.7) and get:

$\xrightarrow{\{fd[14],\underline{hd}[2]\}}\Rightarrow (p3_1, p5_1[1], \boldsymbol{d_1}, b_1[5]).DP_1' \mid (p3_2[1], p5_2[3], \boldsymbol{d_2[14]}, b_2[4]).DP_2' \mid$
$(p3_3[3], p5_3, d_3, b_3[9]).DP_3' \mid (b_1[5]; i_1).(a[6]).A' \mid (b_2[7]; i_2).(t[6]).T' \mid (b_3[8]; i_3).(g_1).G_1' \mid$
$((b_4[9]; i_4).(g_2[10]).G_2' \mid (b_5[4]; e_2).(u).U' \mid (b_6[11]; i_6).(c_1[10]).C' \mid (b_7; i_7).(c_2).C' \mid$
$(p3_4, p5_4[12], d_4, b_4[7]).DP_4' \mid (p3_5[12], p5_5[13], d_5, b_5[8]).DP_5' \mid$
$(p3_6[13], p5_6, d_6, b_6[11]).DP_6' \mid \boldsymbol{(h[14]; f)}.(e_2).UDG'$

UDG could now simply continue its walk, or it can interact via its e action with the uracil. Note that other bases expose the i action, so uracil cannot interact with them. The u, a, t, g, or c actions block e or i, so correct base pairs are not affected by repairs. In our example e_2 on UDG interacts with e_2 on U, breaking bond 4 between b_5 in UDG and b_2 in DP_2. We have achieved the desired repair, since the uracil is removed from the DNA. We model this by the following transition (we use the rewrite rule again):

$\xrightarrow{\{ee[15],\underline{bb}[4]\}}\Rightarrow (p3_1, p5_1[1], d_1, b_1[5]).DP_1' \mid (p3_2[1], p5_2[3], d_2[14], \boldsymbol{b_2}).DP_2' \mid$
$(p3_3[3], p5_3, d_3, b_3[9]).DP_3' \mid (b_1[5]; i_1).(a[6]).A' \mid (b_2[7]; i_2).(t[6]).T' \mid (b_3[8]; i_3).(g_1).G_1' \mid$
$((b_4[9]; i_4).(g_2[10]).G_2' \mid \boldsymbol{(b_5[15]; e_2)}.(u).U' \mid (b_6[11]; i_6).(c_1[10]).C' \mid (b_7; i_7).(c_2).C' \mid$
$(p3_4, p5_4[12], d_4, b_4[7]).DP_4' \mid (p3_5[12], p5_5[13], d_5, b_5[8]).DP_5' \mid$
$(p3_6[13], p5_6, d_6, b_6[11]).DP_6' \mid (h[14]; f).(\boldsymbol{e_2[15]}).UDG'$

The floating C_2 can now take the place of the U by binding to DP_2 and G_1. This is represented by the following two transitions:

$\xrightarrow{bb[16]}\xrightarrow{gc[17]} (p3_1, p5_1[1], d_1, b_1[5]).DP_1' \mid (p3_2[1], p5_2[3], d_2[14], \boldsymbol{b_2[16]}).DP_2' \mid$
$(p3_3[3], p5_3, d_3, b_3[9]).DP_3' \mid (b_1[5]; i_1).(a[6]).A' \mid (b_2[7]; i_2).(t[6]).T' \mid (b_3[8]; i_3).(\boldsymbol{g_1[17]}).G_1' \mid$
$((b_4[9]; i_4).(g_2[10]).G_2' \mid (b_5[15]; e_2).(u).U' \mid (b_6[11]; i_6).(c_1[10]).C' \mid (\boldsymbol{b_7[16]}; i_7).(\boldsymbol{c_2[17]}).C' \mid$
$(p3_4, p5_4[12], d_4, b_4[7]).DP_4' \mid (p3_5[12], p5_5[13], d_5, b_5[8]).DP_5' \mid$
$(p3_6[13], p5_6, d_6, b_6[11]).DP_6' \mid (h[14]; f).(e_2[15]).UDG'$

The resulting process is shown in Figure 7.3.

If uracil would have been bonded on u, the interaction with UDG could not have happened, so the defect is recognized. We now have the uracil broken from the deoxyribose/phosphate group and the b action on the deoxyribose/phosphate group ready to bond to another base. UDG needs to release U and then either to continue

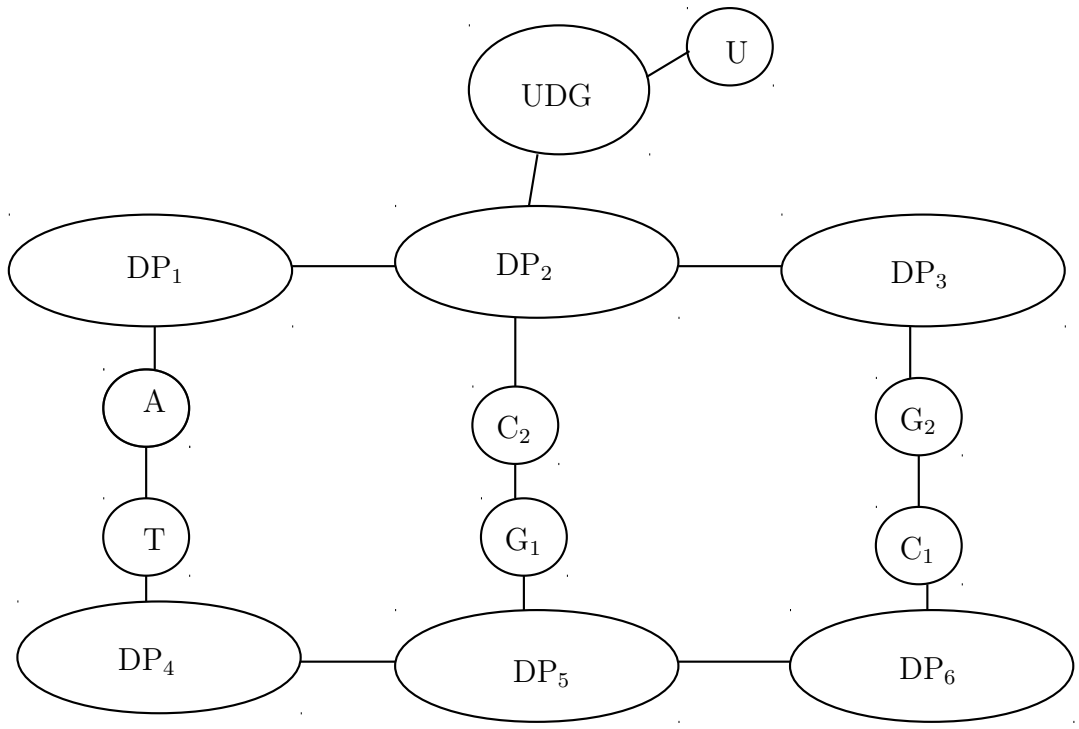

Figure 7.3: The repaired DNA fragment, with the uracil replaced by a cytosine.

its walk or release itself from the DNA. We can again use our new operator for modelling UDG, since this way we achieve that the repair mechanism happens. On the other hand by combining two groups of actions we "block" this repair to happen if the desired second bond is there.

A limitation of our modelling is that it allows the UDG during its "walk" to bind to any DG group, since there is no restriction which d action is used. In reality of course UDG must continue with the nearest DP group. This is a spatial effect our calculus does not model so far. Similarly, the repair by UDG binding to the e action of a uracil (U) is not dependent on the UDG being next to it, whereas in reality it is. Again this is a spatial effect, which we will discuss in Section 9.3.1.

We have used the software from Chapter 8 to test the modelling. The desired path is one of the possibilities, which shows that our modelling is adequate in this respect. We also get some undesired reactions: Apart from UDG binding to any DP group during its walk, there is also the possibility that the unused $p5$ action of a DP, which is at the end of the DNA strand, interacts with an unused $p3$ from the a DP at the other end of the DNA strand. This is not impossible as a such, but prevented in reality by at least two effects we do not model. One is again the spatial arrangement, the other is the fact that the ends of the DNA are protected by special groups, which we do not model here. They prevent reactions at the DNA ends.

7.3 Conclusion

In this chapter we have seen that our calculus is suitable for modelling higher-level processes in biological systems. We have also used the full power of our calculus with nested prefixes like $(s; b).(s'; b').P$, and not only simple processes of the form $(s; b).\mathbf{0}$, as we did before. We have modelled the "walk" of UDG along a strand of DNA by actions in s, and, once a fault is found, the repair mechanism was modelled by the actions in s'. We have seen that our modelling enables some unwanted reactions, but we can explain this by noting that we do not include spatial effects in our modelling.

Chapter 8

Simulation software

In order to ensure that we get a faithful list of the possible transitions of a CCB process we decided to develop a software simulation tool. It is obvious from previous sections that already relatively small systems are difficult to analyse "manually". A simulation tool can help us to comprehensively examine transitions.

8.1 Software architecture

The simulation software has been written in Java and is named CCBsimulation.[1] An object-oriented approach, as it exists in Java, is suitable for our sort of problem. Furthermore the standard libraries are extensive and cover the needs for our basic simulation.

The tool emphasizes the integration of the execution of the calculus with the graphical display. For this reason we have decided against using an existing tool. Some of such tools have good support for the language implementation (e.g. Maude [75] or ProB [99]), but would make a graphical display difficult. The Concurrency Workbench [110] has a graphical interface, but is relatively old and does not support the use of keys. Furthermore the core of our language is small and can easily be implemented in a compact system, whereas more flexible systems allowing different calculi and languages to be integrated possess more complexity. We can therefore implement the language and the graphical display without much overhead in a small system.

[1] The source code for the project is available at `https://github.com/stefhk3/ccbsimulation`.

Some design principles and fundamental decisions were taken for the implementation of the simulation. These are:

- A process object always represents one state of a process. If a transition is executed this changes the process (i.e. adds/removes the key of one or more actions in the process). This means that in order to build a transition network the process needs to be copied into a new object before a transition is applied.
- Recursion is a central principle used throughout. Since the definition of the syntax of the calculus is recursive, our process classes are as well: Apart from the **0** process, which represents the end of the recursion, all processes contain one (or in case of `Parallel` two) other process.
- In object oriented programming methods of a class can implement algorithms working with objects of that class. This means that the classes need to be changed when the calculus is extended or additional features are implemented. For core functionalities this seems acceptable, but in other cases more flexibility might be desired. For this we adopt the visitor pattern ([33], p. 331). We will explain below what this means in our software.

The code of the simulation tool consists of various classes. These are organised in packages according to their purpose. The classes which are needed to represent and execute processes are explained here by package and shown in the class diagram in Figure 8.1:

- The package **org.openscience.ccb.process** contains classes for modelling processes. They follow the syntax given in Chapter 4. So we have a class for `Prefix`, `Restriction` and `Parallel`, plus an extra class for the **0** process. There is an interface `Process` which is implemented by all process classes. Important methods implemented in the classes are:
  - `getActionsReady` lists all actions which are ready in a process to be executed. This is an important function for inferring the transitions.
  - `inferTransitions`: This method gives a list of all possible transitions of this process. It takes the synchronization function as a parameter. `inferTransitions` and `getActionsReady` work by calling the respective method on the sub-process(es), since processes are defined recursively. The **0** process will just return an empty list of actions or transitions, which means the ultimate result is built recursively from the sub-processes.

- `executeTransition`: This executes a transition. The transition has typically been found using `inferTransitions`. Executing a transition changes the state of the process object respectively the relevant subprocesses in it.
- `getKeys`: This method gives all keys from this process. This is used for checking if a key is fresh.
- `toString`: This is a standard Java method for printing classes. In our case it formats the process according to the syntax in Chapter 4.
- `getXML`: This returns an XML representation of the process. XML is used to save and reload processes. There is also a constructor in the process classes, which accepts XML as an argument and builds the process from the XML code.
- `clone`: This provides a deep copy of a `Process`. As mentioned transitions are executed in a process, so this is needed to execute several transitions from one `Process`.

The methods mentioned are part of the interface `Process` and are therefore implemented in all processes. As explained above this means all classes need to be changed if the methods would change. This is justified since these are core methods. For other potential aspects of processes the visitor pattern is employed. For this each process has a method `accept(CCBVisitor visitor)`, where `visitor` is any class implementing the interface `visitor`. This interface has a method `visit(Process process)`, which is implemented by the actual visitor classes and performs the desired operation on the process, depending on the type of process. The process classes implement `accept`. In this method the process class calls the `visit` method of the visitor and calls `accept` of subprocesses if necessary, without knowing about what the visitor actually does. This means that the `visit` method of the visitor is called with all processes as a parameter and the visitor can perform any desired action on them. In this way a new operation, which modifies a process and all its subprocesses, can be added without modifying the code of the processes.

- The classes in **org.openscience.ccb.transition** model transitions. There is an abstract class `Transition` which is implemented by `SimpleTransition` and `Communication`. These classes represent all transitions from Figures 4.2 and 4.4. Wheter an action is forward or reverse is determined by the attribute `keybroken` of the class being 0 (for forward transitions) or a natural number (which must be an existing key). In a similar fashion concerted actions have the

attribute `triggeredTransition` set to give the linked action. This attribute is null in case of other transitions.

- Actions are modelled in the **org.openscience.ccb.action** package. An abstract class `Action` is implemented by `WeakAction` and `StrongAction`. The behaviour of these classes is identical, but in order to distinguish them when inferring the transitions they are modelled as different classes. There is also a class `PastSemicolonAction`, which models the b action in prefixes of the form $(s; b)$. Again this class has no different behaviour. It extends `WeakAction` since b always is a weak action. Any action can have a key or be fresh. We do not model subscripts explicitly, if subscripts are desired the actions names have to be given like $a1$. We assume the action names contain letters only and the subscripts are numbers only, so if necessary they can be separated.

- The package **org.openscience.ccb.reduction** models the move and prom rules from Figure 4.7. We have decided not to implement these directly in the processes, but as separate classes. The decision is explained below.

- The package **org.openscience.ccb.predicate** models the predicates from Section 9.3.1 (distance `ds` and connectivity `conn`). These are implemented as separate classes similar to the reduction rules.

- **org.openscience.ccb.synchronisation** contains only one class, `Synchronize`. This represents the synchronisation function γ between actions of CCB. The important method is `isSychronized`, which takes two action labels and gives the name of their synchronisation, or null if they cannot interact.

- **org.openscience.ccb.parser** contains `CCBParser`, which parses a process term into a `Process` object.

- In **org.openscience.ccb.test** collects unit tests for the package, see Section 8.4.

- Finally **org.openscience.ccb.ui** comprises the user interface classes discussed in Section 8.2.

The software tool uses some external software. This is firstly the Java Runtime Environment. The version required is 1.8 or higher. One major reason is that the graphical user interface described in Section 8.2 uses JavaFX, which is only available from 1.8. JavaFX is superior to older solutions (AWT and Swing) with respect to user experience and look and feel. JGraphT [52] is used for the handling of graphs

together with mxgraph [84] for the layout of graphs. Finally XOM [122] is used for XML handling and parsing (we decided to use XOM instead of the DOM or SAX implementations, which are part of the standard Java library, since it has a cleaner architecture and API[2]). JUnit [53] is used to run unit tests for the package.

[2] [119] explains this well, even it is by the author of XOM.

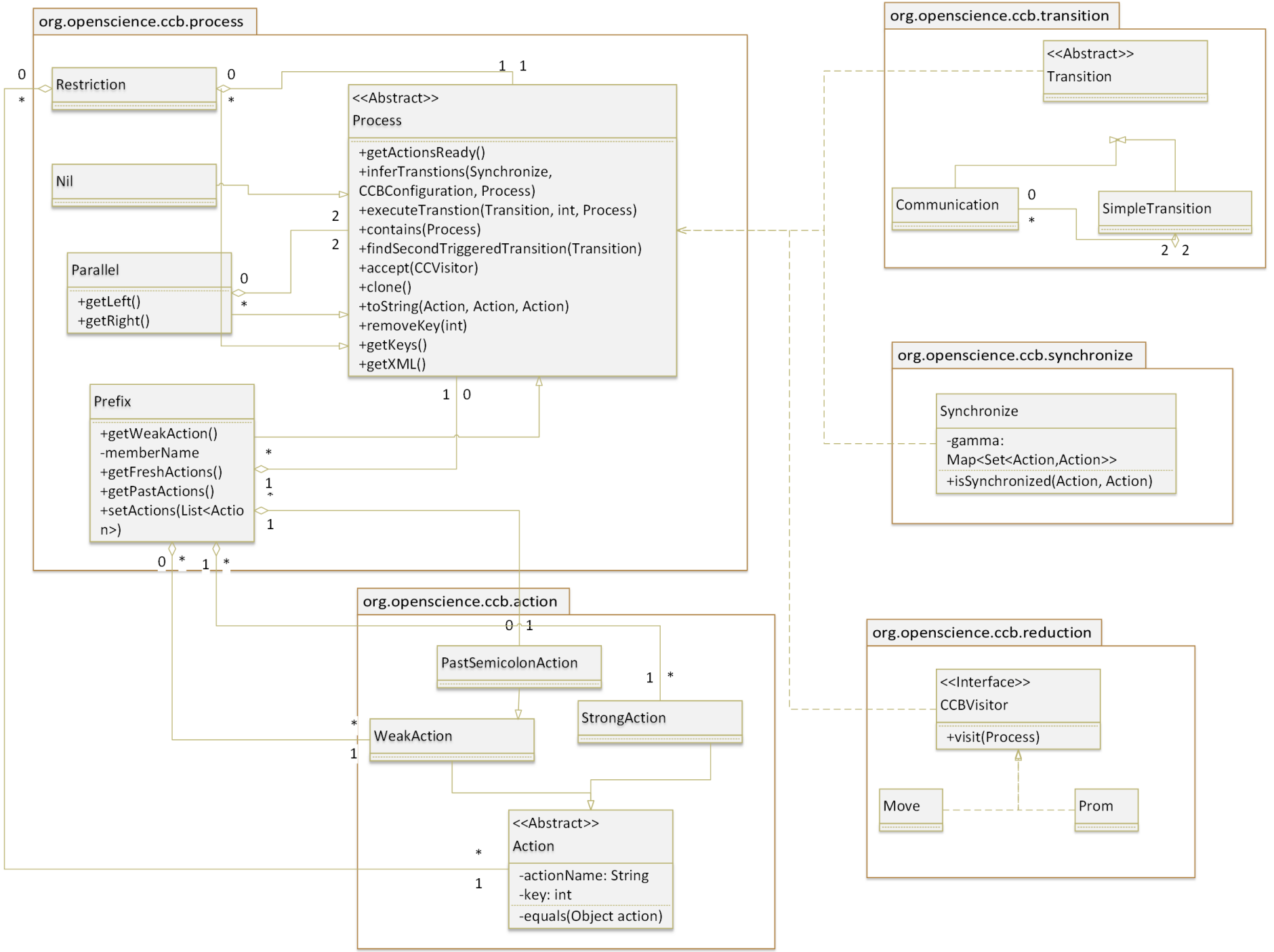

Figure 8.1: The main classes of the software package `CCBsimulation`.

8.2 User interfaces

All of the above mentioned classes are used by two user interface implementations, a command-line and a graphical user interface. The difference is therefore only in the interface, the implementation of the execution of CCB processes is identical.

8.2.1 Command-line interface

We offer a command-line interface, called CCBcommandline, which is usable on any machine, even without a graphical user interface. A record of a typical session in CCBcommandline is shown in Figure 8.2. The user enters a process and defines the synchronisation function and the weak actions. The user is then presented with all possible inital transitions. In the transitions, the actions which will be executed are marked with _, any undoing triggered by the ; operator is marked with #. The user is then offered with the choice to either execute all or some of the transitions. In Figure 8.2 the user started with the system from Section 5.3 and has decided to execute only the first transition. The result is a new set of processes, for which the user again is shown the transitions. The user can continue until either they decide to stop or continue until no new processes can be generated.

There are a couple of settings which influence the behaviour of the software. They are provided in a configuration file. Two of them control if the rules prom and move are applied automatically whenever possible. A third determines if spontaneous undoing of communications is possible. If this is enabled the number of possible transitions increases considerably and it becomes difficult to spot desired transitions. Also in many cases, for example in our chemical modelling, spontaneous breaks do not normally happen. Therefore they can be switched off by default. The user then has the ability to execute individual transitions as needed.

8.2.2 Graphical user interface

As an alternative we also offer a graphical user interface for the CCB simulation, called CCBgui. CCBgui offers the functions of CCBcommandline and, in addition, various visualizations. Figure 8.3 shows a screenshot of CCBgui. The process loaded is the same as in Figure 8.2. After loading the initial process we have executed all possible transitions. These gave us new processes (shown in the upper left section). We can see that now 19 new transitions are possible (centre left section). Similar to

CCBcommandline the user can decide to execute some or all possible transitions and do spontaneous breaks of bonds (the same configuration file as in CCBcommandline is used). In addition, the following features have been implemented:

- A graphical view of the processes dealt with so far. This is the lower left section of the GUI. Hovering over the nodes in this display shows the process term, which is represented by a node. In Figure 8.3 we started with process P_0 and got three transitions to P_1, P_2, and P_3. From P_2 one of the transitions goes to P_3. Since the model loaded was again that from Section 5.3, this shows the transitions from

  $$FA \mid W \mid W \mid W$$

  in Figure 5.6, where

  $$P_0 = FA \mid W \mid W \mid W$$

  $$P_1 = i2 \mid W \mid W$$

  $$P_2 = FA \mid W \mid OH \mid H_3O$$

  $$P_3 = i6 \mid W \mid OH \mid W$$

  The layout of the transitions in CCGgui is done automatically (using mxgraph), so it is different from that in Figure 5.6, which was done manually.

- Processes can be viewed as graphs. The translation is as mentioned in Section 6.3 and is further elaborated on in the next section. In the screenshot P_1 is selected and the molecules modelled by this process are shown. Again mxgraph is used for layout, so the layout does not necessarily follow chemical conventions. Processes can be selected for display either from the list of current processes or from the graphical view of the processes dealt with so far.

- The current state of the overall system can be saved and reloaded. This includes all current processes, the synchronization action and the set of weak actions. From this, the program can start in the current state. The past processes are currently not saved.

Figures 8.4 and 8.5 shows the base excision repair from Chapter 7. The upper image in Figure 8.4 shows the UDG floating freely. Note that the two DNA backbone strands are not shown as parallel, this is again due to the generic layout algorithm. This forms a ring out of the backbone strands and the two aminoacid bridges (compare Figure 7.2). The lower image in Figure 8.4 shows UDG bonded to a deoxyribose/phosphate group. In the upper image in Figure 8.5 UDG has moved

to the deoxyribose/phosphate group where the uracil is bonded. Finally, in the lower image in Figures 8.5, the uracil has been removed from the DNA. In each step only the relevant transition was executed. Other transitions are possible at each stage, but disregarded here.

```
shk3@Heinrich:~/git/ccbsimulation/lib$ java -cp "*" org.openscience.ccb.ui.CCBcommandline
Enter your process:
(((c1[1],c2[2],c3[3],c4[4];p).0 | (h1[1];p).0 | (h2[2];p).0 | (o1[3],o2[4],n).0) \ {c1,c2,c3,c4,h1,h2,o1,o2,c1h1,
   ↪ c2h2} | ((h3[5];p).0 | (h4[6];p).0 | (o3[5],o4[6],n).0) \ {h3,h4,o3,o4} | ((h5[7];p).0 | (h6[8];p).0 | (o5
   ↪ [7],o6[8],n).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0 | (o7[9],o8[10],n).0) \ {h7,h8,o7,o8} ) \ {n,
   ↪ p}
Enter your synchronisation function [format: a,a,b...):
c1,h1,c1h1,c1,h2,c1h2,c1,h3,c1h3,c1,h4,c1h4,c1,h5,c1h5,c1,...
Enter the weak actions [format: a,b,c...):
n,p
0: Ready for execution ((c1[1],c2[2],#c3[3],c4[4];_p).0 | (h1[1];p).0 | (h2[2];p).0 | (n,o1[3],o2[4]).0) \ {c1,c2
   ↪ ,c3,c4,h1,h2,o1,o2,c1h1,c2h2} | ((h3[5];p).0 | (h4[6];p).0 | (_n,o3[5],o4[6]).0) \ {h3,h4,o3,o4} | ((h5[7];
   ↪ p).0 | (h6[8];p).0 | (n,o5[7],o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0 | (n,o7[9],o8[10]).0)
   ↪  \ {h7,h8,o7,o8};np;0
1: Ready for execution ((c1[1],c2[2],c3[3],c4[4];p).0 | (h1[1];p).0 | (h2[2];p).0 | (_n,o1[3],o2[4]).0) \ {c1,c2,
   ↪ c3,c4,h1,h2,o1,o2,c1h1,c2h2} | ((#h3[5];_p).0 | (h4[6];p).0 | (n,o3[5],o4[6]).0) \ {h3,h4,o3,o4} | ((h5[7];
   ↪ p).0 | (h6[8];p).0 | (n,o5[7],o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0 | (n,o7[9],o8[10]).0)
   ↪  \ {h7,h8,o7,o8};np;0
2: Ready for execution ((#h3[5];_p).0 | (h4[6];p).0 | (n,o3[5],o4[6]).0) \ {h3,h4,o3,o4} | ((h5[7];p).0 | (h6[8];
   ↪ p).0 | (_n,o5[7],o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0 | (n,o7[9],o8[10]).0) \ {h7,h8,o7,
   ↪ o8};np;0
3 transitions ready, execute [a]ll or give [e]xclude or [i]nclude list or say [b]reakX
i1
0: Ready for execution ((c1[1],c2[2],#c3[3],c4[4];_p).0 | (h1[1];p).0 | (h2[2];p).0 | (o1[3],o2[4],n[100]).0) \ {
   ↪ c1,c2,c3,c4,h1,h2,o1,o2,c1h1,c2h2} | ((h3[100];p).0 | (h4[6];p).0 | (_n,o3,o4[6]).0) \ {h3,h4,o3,o4} | ((h5
   ↪ [7];p).0 | (h6[8];p).0 | (n,o5[7],o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0 | (n,o7[9],o8
   ↪ [10]).0) \ {h7,h8,o7,o8};np;0
1: Ready for execution ((c1[1],c2[2],#c3[3],c4[4];_p).0 | (h1[1];p).0 | (h2[2];p).0 | (o1[3],o2[4],n[100]).0) ...
2: Ready for execution ((#h3[100];_p).0 | (h4[6];p).0 | (n,o3,o4[6]).0) ...
3: Ready for execution ((h3[100];p).0 | (#h4[6];_p).0 | (n,o3,o4[6]).0) ...
4: Ready for execution ((h3[100];p).0 | (h4[6];p).0 | ...
5: Ready for execution ((#h5[7];_p).0 | (h6[8];p).0 | ...
6 transitions ready, execute [a]ll or give [e]xclude or [i]nclude list or say [b]reakX
```

Figure 8.2: A record of the program `CCBcommandline` with a system of three water and one formaldehyde molecule loaded. One round of transitions has been executed. The synchronization function and some processes have been shortened (indicated by ...) in the figure.

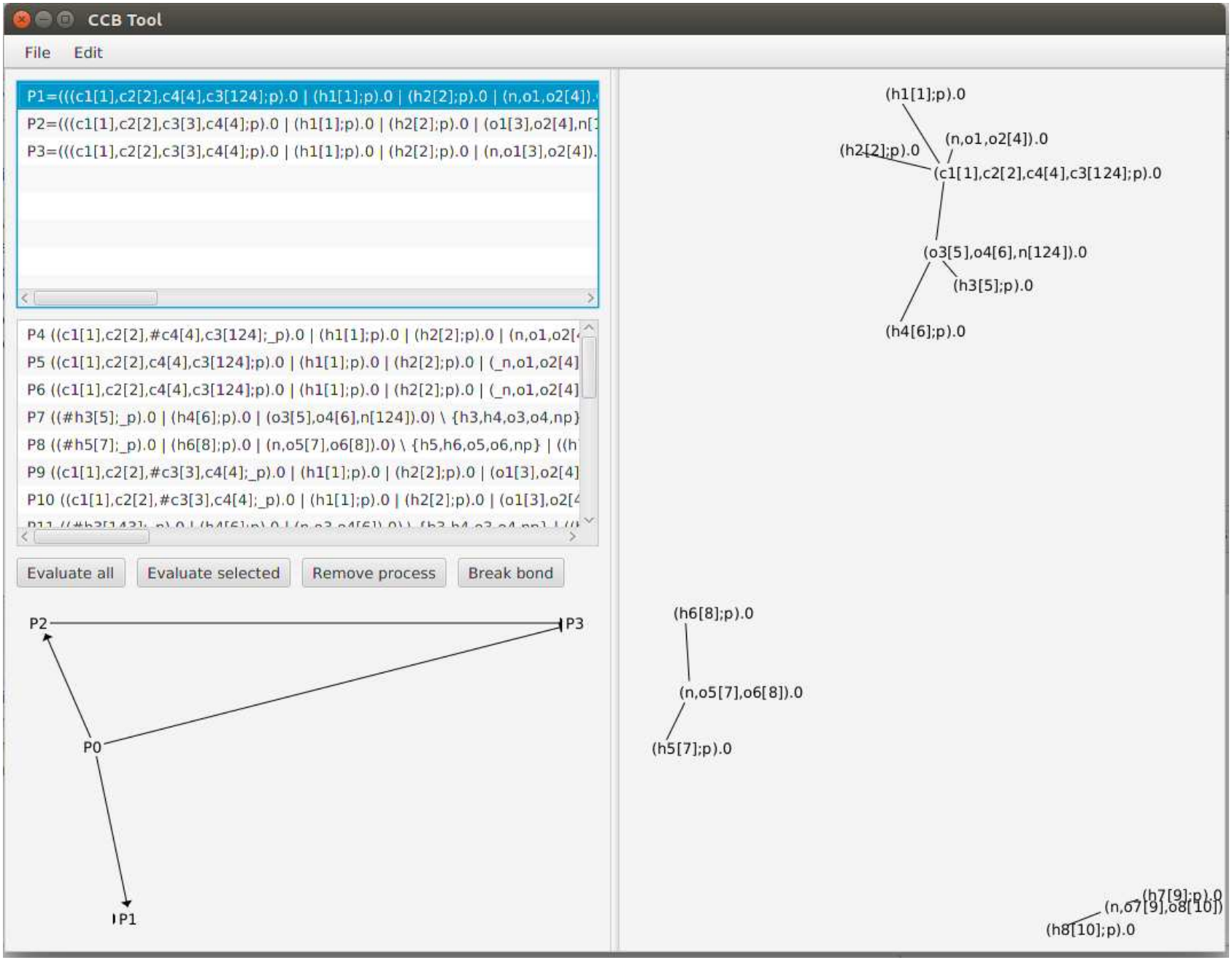

Figure 8.3: A screenshot of the program `CCBgui` with a system of three water and one formaldehyde molecule loaded. One round of transitions has been executed.

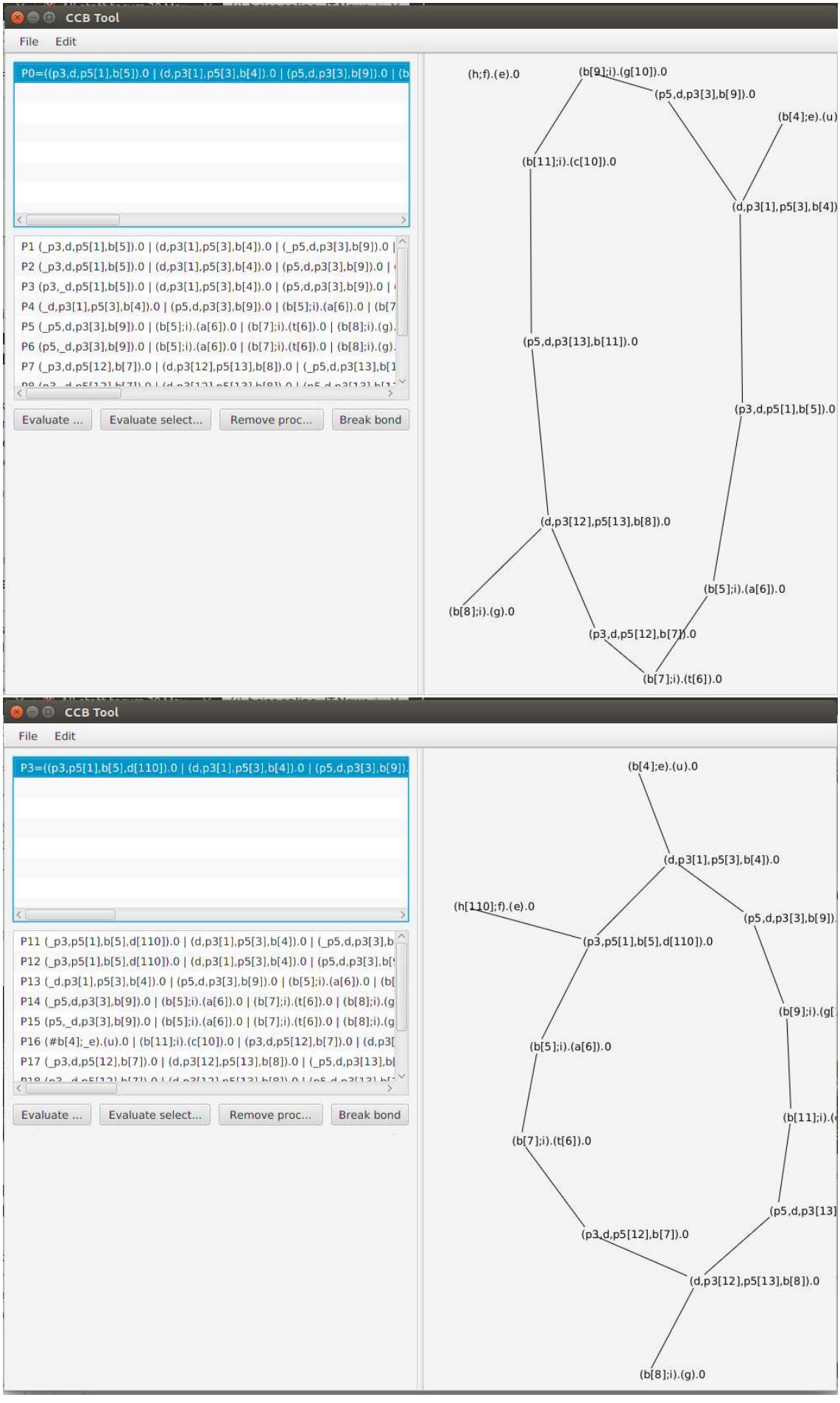

Figure 8.4: A screenshot of the first two steps of the execution of the base excision repair in CCBsimulation. Only the relevant transition is chosen and executed.

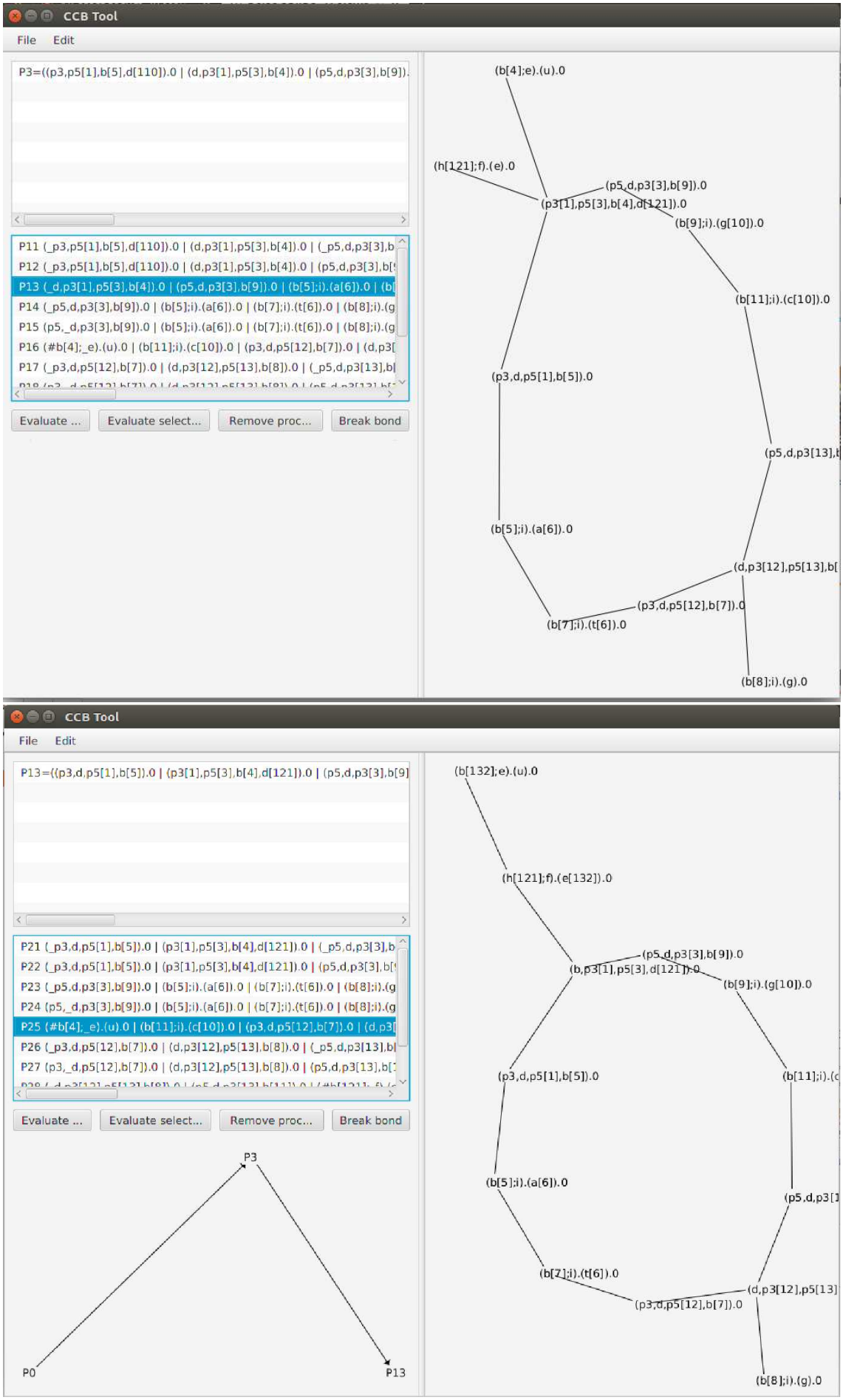

Figure 8.5: A screenshot of the execution of the last two steps of the base excision repair in CCBsimulation. Only the relevant transition is chosen and executed.

8.3 Process equivalence

As we have explained in Remark 6.1 we can convert processes which are chemically process equivalent into each other by using a set of simple rewrite rules. We also explained that using a graph isomorphism as an intermediate step helps with quickly establishing if two graphs can be transformed. We firstly give an overview of this process and then explain how it is implemented in the software. The steps to check if processes P_1 and P_2 are chemically process equivalent are:

- We convert P_1 and P_2 into graphs G_1 and G_2.
- We check if there exists an isomorphism between G_1 and G_2.
- If an isomorphism exists use the algorithm from Figure 8.6 to derive the rewrite rules needed to rewrite P_1 into P_2. If P_1 and P_2 are not isomorphic, then P_1 and P_2 cannot be chemically process equivalent.

The implementation of these steps was done as part of the software and is used by both user interfaces. The first step, the graph conversion, is done as explained shortly in Section 6.3. In more detail, we firstly create a node for each subprocess, which is in parallel with other processes. If two such subprocesses share a key, we create an edge linking them. If there is more than one key shared between two processes, this is represented by the weight of the edge being incremented for each shared key. The set or the semicolon operators are disregarded, the keys of all actions in one subprocess are treated the same.

The class `SimpleGraph` from JGraphT is used for building the graph. This represents an undirected, weighted graph. The processes are used as nodes directly. The edges are built from the keys, as described in Section 6.3. As mentioned there the same conversion is used for displaying the processes in the graphical user interface. The layout is irrelevant for the equivalence check or the inference of transitions, it is only for display purposes.

For the isomorphism check the class `VF2GraphIsomorphismInspector` from JGraphT is used. An isomorphism is defined as a bijection between the vertices in the graphs, so that two edges are adjacent in one graph if and only if they are adjacent in the other graph. Efficient solutions to the problem of finding graph isomorphisms have been proposed in the literature, `VF2GraphIsomorphismInspector` implements the algorithm described in [19], which is a fast and commonly used al-

gorithm for this purpose.[3] The algorithm is implemented in JGraphT. In order to apply it to our processes we need to provide implementations for comparing vertices and edges. These are implemented as part of our software package. Two edges are considered equals if their weights are equal. Two vertices (which are processes in our case) are considered equal if the actions, ignoring subscripts, are equal. This is necessary since the informal process identifiers we used when writing down our processes (for example the H_1' in $(h_1; p).H_1'$) are not used in the software. When entering $(h_1; p).H_1'$) into the software it would become $(h_1; p).\mathbf{0}$) and we have no process identifier to decide of two subprocesses are identical. Using the actions for this purpose is in line with our chemical metaphor since two atoms with the same interaction capability are the same for all practical purposes.

Finally the algorithm described in Figure 8.6 can be applied to transform the process transformed into G_1 to the process transformed into G_2. The algorithm firstly finds a new key and a new subscript, which is larger than all currently used (line 2). It then swaps all keys in G_1, which are not matched in G_2, to the new key and increments the new key (lines 3 to 7). It then swaps all keys in G_1, which are not matches in G_2 (these are the keys changed in the first step), to the matching key in G_2 (lines 8 to 11). It then executes a similar algorithms for all subscripts in actions in lines 12 to 32. In this part, it iterates over all vertices in the graph (which represent all parallel processes). For each vertex, it checks if the subscript of the process identifier and the subscripts of the actions in this process match those in G_2, if not, it sets them to new subscript and increments the new subscript (line 12 to 22). There are separate loops for past actions (lines 17 to 19) and fresh actions (lines 20 to 22). This is necessary because for past actions we need to order them by the keys to make sure actions with the same keys get the same subscript in G_1 and G_2. For fresh actions, there are no key and a lexicographical ordering is enough to ensure that actions with the same name get the same subscripts in G_1 and G_2. It then repeats the process and sets subscripts, which do not match those in G_2 to the matching subscript in G_2 (lines 23 to 32).

If we would swap key X in G_1 directly into the key Y it maps onto in G_2, it could happen that Y is already used in G_2. We could then not distinguish Y keys in G_2, which were swapped to Y, from those who had key Y in the original process. Therefore if, in a later step, we swap Y into a third key, we would also change keys which should not be changed. In an implementation this might not be necessary since the processes might well be identifiable otherwise (e.g. by memory addresses), but we want to keep the algorithm as general as possible.

[3] For an overview of algorithms for finding graph isomorphism see [70].

1 *Precondition*: We have G_1 and G_2, representing processes P_1 and P_2. The conversion is done as in Section 6.3. Both processes have all process identifiers and actions annotated with subscripts, which are unique in the process. All edges have a key $\in \mathcal{K}$. We have a function *getProcess* which returns the subprocess represented by a vertex. We have a graph isomorphism which maps every vertex in G_1 onto a vertex in G_2 and every edge in G_1 onto an edge in G_2. We have a function $getEdge(G_2, e)$, which return the edge in graph G_2 mapped to edge e in G_1 and a function $getVertex(G_2, v)$, which return the vertex in graph G_2 mapped to vertex v in G_1.

2 Find a k_n greater than all keys used in G_1 and an s_n greater than all subscripts used in G_1.

3 *for each* (edge e with key k in G_1)

4 Let k_2 be the key of $getEdge(G_2, e)$;

5 *if* $(k \neq k_2)$

6 $\alpha\mathsf{keys}(P_1, k, k_n)$;

7 Increment k_n;

8 *for each* (edge e with key k in G_1)

9 Let k_2 be the key of $getEdge(G_2, e)$;

10 *if* $(k \neq k_2)$

11 $\alpha\mathsf{keys}(P_1, k, k_2)$;

12 *for each* (vertex v with subscript s in G_1)

13 Let s_2 be the subscript of $getVertex(G_2, v)$;

14 *if* $(s \neq s_2)$

15 $\alpha\mathsf{subscripts}(P_1, s, s_n)$;

16 Increment s_n;

17 *for each* (action a with subscript s_1 in fresh actions of $getProcess(v)$ in lexicographical order)

18 find subscript s_2 for matching action in $getProcess(getVertex(G_2, v))$;

19 *if* $(s \neq s_2)$ $\alpha\mathsf{subscripts}(P_1, s_1, s_n)$; Increment s_n;

20 *for each* (action a with subscript s_1 in past actions of $getProcess(v)$ ordered by key)

21 find subscript s_2 for matching action in $getProcess(getVertex(G_2, v))$;

22 *if* $(s \neq s_2)$ $\alpha\mathsf{subscripts}(P_1, s_1, s_n)$; Increment s_n;

23 *for each* (vertex v with subscript s in G_1)
24 Let s_2 be the subscript of $getVertex(G_2, v)$;
25 *if* $(s \neq s_2)$
26 αsubscripts(P_1, s, s_2);
27 *for each* (action a with subscript s_1 in fresh actions of $getProcess(v)$ in lexicographical order)
28 find subscript s_2 for matching action in $getProcess(getVertex(G_2, v)$;
29 *if* $(s \neq s_2)\alpha$subscripts(P_1, s_1, s_2)
30 *for each* (action a with subscript s_1 in past actions of $getProcess(v)$ ordered by key)
31 find subscript s_2 for matching action in $getProcess(getVertex(G_2, v))$;
32 *if* $(s \neq s_2)\alpha$subscripts(P_1, s_1, s_2)

Figure 8.6: An algorithm to rewrite a CCB process P_1 into a chemically equivalent processes P_2.

In the CCBgui software the user can enter two processes and is presented with the steps needed to transform the first process into the second. Figure 8.7 shows on example of this. The two processes used here are those used in Section 6.3. We notice that there are more steps needed than in Section 6.3. The reason is that in Section 6.3 we move processes and use swapkeys and swapsubscripts to minimize the number of operations needed. The algorithm performs a graph matching and, based on this, systematically uses αsubscripts and αkeys to make the processes identical. An intelligent minimization of changes using moving of processes and swaps of keys and subscripts is not implemented.

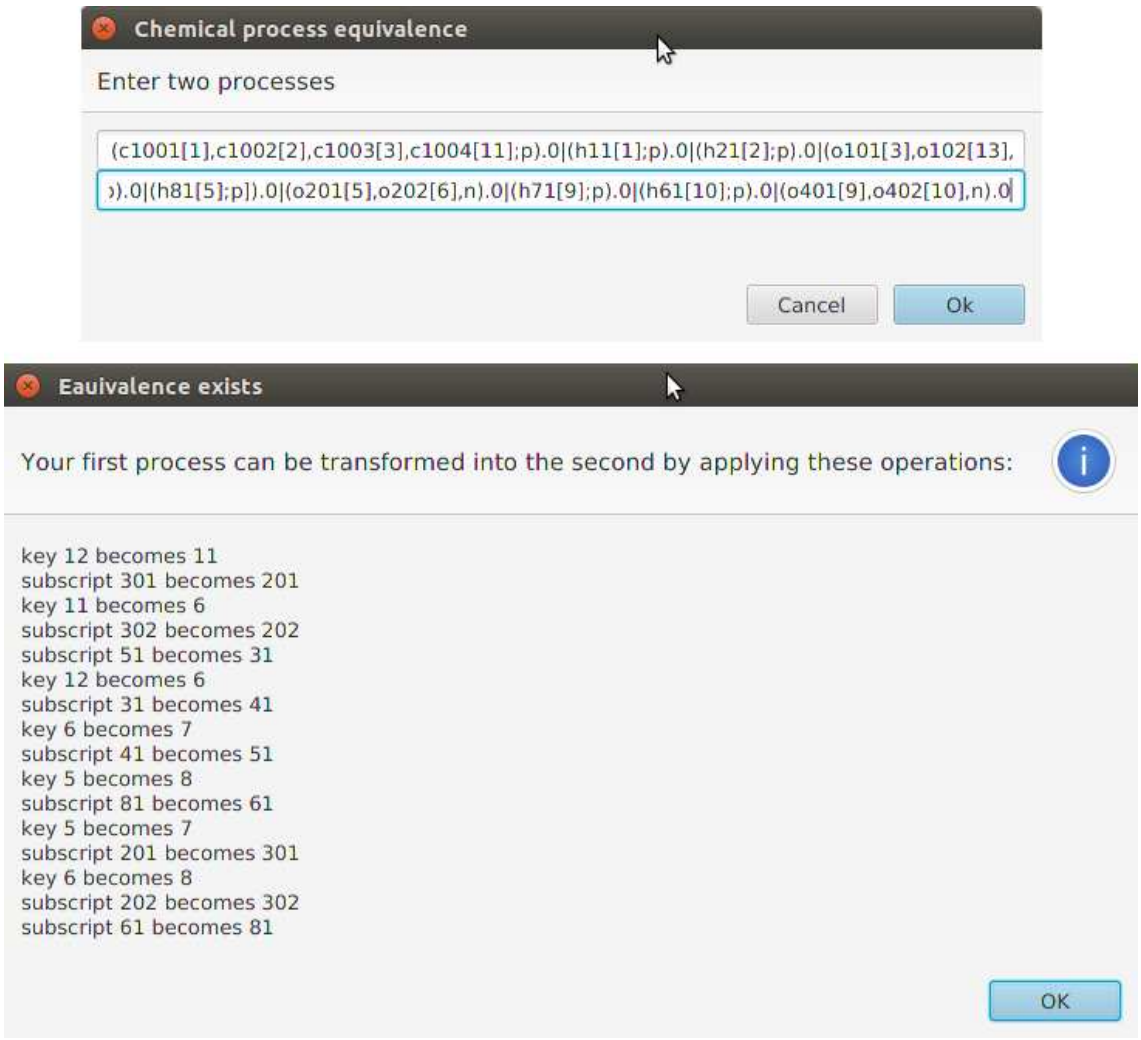

Figure 8.7: CCBgui demonstrating chemical process equivalence of two processes.

8.4 Testing

In order to ensure the quality of the software two testing strategies have been employed.

Firstly, JUnit is used for unit testing (the tests are collected in the package **org.openscience.ccb.test**). They contain a number of small test cases. These have been written to test a range of examples, covering simple standard situations as well as border cases (for example processes with no transition possible, only a weak action possible, only communications possible, no communcations possible, transitions possible in the process, but prevented by a restriction). The tests include all components, namely the parser, the modelling of the processes, actions and transitions, including the inference and execution of transitions, and application of the reduction rules. The tests work directly on the classes representing processes and do not involve the user interfaces. Since the JUnit tests can be executed auto-

matically they also serve for regression testing. We have decided not to do automatic testing for the user interfaces, since this is relatively complicated to do and does not provide much value if (as in our case) no logic is implemented directly in the user interface.

Secondly, we have executed the hydration of formaldehyde in water from Chapter 5, using the command line interface. A record of this is provided in Appendix 9.3.2. The compounds and paths from Figure 5.6 are all derived during the execution. Some other reactions which are possible in the calculus have been excluded by only executing the desired reactions. We have also executed the base excision repair from Chapter 7 and the autoprotolysis of water from Chapter 2 and found them to give the transitions expected.

8.5 Conclusion

We have demonstrated the use of our software package CCBsimulation. We have shown that it can be used to derive the transitions of the examples given in previous chapters. It derives all possible transitions, which helps us to verify that the software works correctly. We have also checked that all transitions that we derived "by hand" in Chapters 7, 2, and 5 can be derived by CCBsimulation. Two user interfaces are available for the user to choose. We have shown that the software architecture mirrors CCB, in particular by the use of recursion. We have demonstrated that chemical process equivalence can be checked in our software.

Chapter 9

Conclusion

This chapter summarizes the thesis presented and evaluates the significance of the work done. It also provides directions for future work based on this thesis.

9.1 Summary

We have introduced a new process calculus called the Calculus of Covalent Bonding (CCB). It is inspired by chemical reactions where covalent bonds can be formed and broken. Forming and breaking typically go hand in hand, driven by small disequilibria in charge distribution. We called this type of reversibility, where undoing is part of a pair of bonds being formed and broken, locally controlled reversibility. We called such pairs of events concerted actions.

In order to model this type of reversibility by means of a calculus we introduced a new general prefixing operator that allows us to model locally controlled reversibility. CCB incorporates this new operator into a CCS-style calculus. The new operator permits us to perform pairs of concerted actions, where the first element of the pair is a creation of a (weak) bond and the second element is breaking one of the existing bonds. The mechanism has purely a local character; there is no need for an extensive memory or global control.

We have given the calculus operational semantics. We have shown that the calculus allows for undoing steps of computation in both causal consistent and out-of-causal order. We have also shown that the calculus without the special prefixing operator is a conservative extension of a forward-only calculus, whereas adding our new operator gives as new behaviour.

We have modelled the autoprotolysis of water and the hydration of formaldehyde in water as examples of a chemical reaction, where our calculus models directly the type of reactions it was inspired by. The Base Excision Repair (BER) is an example where our calculus is applied to model a higher level mechanism, where agents are not necessarily atoms and bonds not always covalent bonds between atoms. Here we also used the full calculus, using nested prefixes like $(s; b).(s'; b').P$, and not only simple processes of the form $(s; b).\mathbf{0}$.

We have established chemical process equivalence of two processes. This allowed us to model reaction networks, where different paths lead to the same molecule, but use different atoms to achieve the reaction.

The software package CCBsimulation allows the simulation of CCB processes and evaluates possible transitions. A command line interface and a graphical user interface can display complex transition networks. The user can understand complex transition networks, which would be difficult to examine manually.

9.2 Evaluation of CCB

We have set out to devise new ways of modelling computations by drawing inspiration from biological and chemical processes. Reversibility was intended to be the focus of our research. We have decided to look at chemical reactions as a manageable field of examples. Modelling these using process calculi, which so far had mainly been used in biological modelling, we aimed for novel ways of doing computations.

We have designed CCB by following this direction. With this we have formally described a calculus which incorporates out-of-causal-order reversibility. To the best of our knowledge this is the first such calculus. This means we can reach in our calculus states which could not be reached by computing forwards alone. Furthermore the calculus has locally controlled reversibility as opposed to a global control. This is relevant because local control is involved in self-organising processes.

When modelling two chemical reactions we have found that they can be represented realistically. Most reactions in our calculus are chemically valid. We had a case (the resonance) which could not be represented properly and some reactions possible in our calculus which are not possible or not realistic. We will look at these in Section 9.3. We have also seen that we can abstract away from faithful modelling of chemical reactions. Base Excision Repair was an example where we can describe a high-level representation of a biological mechanism using the full calculus. An-

other example where this is useful is a model of long standing transactions with compensations in [93].

Our chemical process equivalence allows to deal with reaction networks and the various paths in them realistically. We can show that different paths, which lead to the same compound in reality, can be modelled appropriately in our calculus.

The CCBsimulation allows the simulation of complex transition networks. We have used it successfully to evaluate processes which would have been impossible to evaluate by hand.

9.3 Future work

9.3.1 Improve chemical modelling

As has been mentioned in this thesis there are a number of situations which cannot be modelled well with our calculus. Many of these are related to the spatial aspect of reactions. Some of the cases we have seen are as follows:

1. Intramolecular reactions: In Section 5.4 we have seen that an intramolecular reaction would be possible in our process. Whilst intramolecular reactions exist in such a case the atoms cannot get close enough to react, whereas in larger molecules this could well happen.
2. Steric effects: As we have seen in Section 5.2 the number (and to a degree the size) of the groups around the carbon can prevent a reaction from happening.
3. In our high level example (Section 7) we see a spatial problem similar to the one just mentioned: Interactions are only possible between the neighbouring molecules.

We suggest as a starting point for further research to deal with the mentioned cases by introducing two properties into an extended calculus, called CCB-S for CCB-spatial, namely the distance between processes, and the number of neighbours of a process. We shortly discuss this calculus to give an indication of the direction of future work.

The syntax of CCB-S is the same as that of CCB as given in Section 4.1. This implies that every CCB process is a CCB-S process as well.

In order to deal with spatial issues in our processes we define a metric space. For this we define a function ds which gives the distance for any pair of subprocesses within a process.

Definition 9.1. The distance $\mathsf{ds}(P, Q, S)$ between two subprocesses P and Q of a process $S \stackrel{def}{=} P \mid Q \mid R$ is

$$\mathsf{ds}(P,Q,S) = \begin{cases} 1 & \text{if } \mathsf{keys}(P) \cap \mathsf{keys}(Q) \neq \emptyset \\ 1 + \mathsf{ds}(Q,R,S) & \text{if } \mathsf{keys}(P) \cap \mathsf{keys}(R) \neq \emptyset \text{ and} \\ & \mathsf{ds}(R,Q,S) < \infty \\ \infty & otherwise \end{cases}$$

In this we treat all bonds as having the same length. This is not the case in reality (bond lengths vary roughly from 100 to 300 pm, depending on electronic details of the bonding), but for many purposes a treatment based on a uniform bond length is sufficient.

In order to deal with the steric effects we define the number of neighbours of a process by a function conn.

Definition 9.2. The connectivity $\mathsf{conn}(P, S)$ of a subprocess P of process $S \stackrel{def}{=} P \mid R$ is

$$\mathsf{conn}(P,S) = |\mathsf{keys}(P) \cap \mathsf{keys}(R)|$$

We extend the SOS rules for CCB with side conditions referring to the functions ds and conn. The new SOS rules are given in Figures 9.1-9.3. The predicates std and fsh are as before (Figure 4.1). The only instances where the distance is used is in rules com and concert act. In these rules the thresholds for *con* and *dis* must be chosen by the user as well as the relations R and R' when a system is specified. R can any of $<$, $=$ or, $>$, whereas R' can be $<$or $\leq$. We will see below that this enables the modelling of various examples by varying the conditions. Forward rules not involving a communication are not affected by distance, since they do not involve more than one process. In rule rev com we do not need to stipulate a distance, since the processes that have communicated have the distance 1, and since they share a key.

This calculus could be used to deal with the three cases mentioned above. The intramolecular reactions could be excluded by giving a minimum distance, the steric effect can be modelled by not allowing reactions if an atoms has four or more neigh-

$$\text{act1}\ \frac{\mathsf{std}(X) \quad \mathsf{fsh}[k](s)}{(s,a;b).X \xrightarrow{a[k]} (s,a[k];b).X} \qquad \text{act2}\ \frac{X \xrightarrow{a[k]} X' \quad \mathsf{fsh}[k](t)}{(t;b).X \xrightarrow{a[k]} (t;b).X'}$$

$$\text{par}\ \frac{X \xrightarrow{a[k]} X' \quad \mathsf{fsh}[k](Y)}{X \mid Y \xrightarrow{a[k]} X' \mid Y} \qquad \text{com}\ \frac{X \xrightarrow{a[k]} X' \quad Y \xrightarrow{b[k]} Y'}{X \mid Y \xrightarrow{c[k]} X' \mid Y'}\ (*)$$

$$\text{res}\ \frac{X \xrightarrow{a[k]} X'}{X\backslash L \xrightarrow{a[k]} X'\backslash L}\ a \notin L \qquad \text{con}\ \frac{X \xrightarrow{a[k]} X'}{S \xrightarrow{a[k]} X'}\ S \stackrel{def}{=} X$$

Figure 9.1: Forward SOS rules of CCB-S. The condition (*) is $\gamma(a,d) = c$, $b \in \mathcal{WA}$, $\mathsf{ds}(X,Y)\, R\, dis$ where $dis \in \mathbb{N}$, $\mathsf{conn}(X)\, R'\, con$, and $\mathsf{conn}(Y)\, R'\, con$ where $con \in \mathbb{N}$.

$$\text{rev act1}\ \frac{\mathsf{std}(X) \quad \mathsf{fsh}[k](s)}{(s,a[k];b).X \xrightarrow{\underline{a}[k]} (s,a;b).X} \qquad \text{rev act2}\ \frac{X \xrightarrow{\underline{a}[k]} X' \quad \mathsf{fsh}[k](t)}{(t;b).X \xrightarrow{\underline{a}[k]} (t;b).X'}$$

$$\text{rev par}\ \frac{X \xrightarrow{\underline{a}[k]} X' \quad \mathsf{fsh}[k](Y)}{X \mid Y \xrightarrow{\underline{a}[k]} X' \mid Y} \qquad \text{rev com}\ \frac{X \xrightarrow{\underline{a}[k]} X' \quad Y \xrightarrow{\underline{b}[k]} Y'}{X \mid Y \xrightarrow{\underline{c}[k]} X' \mid Y'}\ (*)$$

$$\text{rev res}\ \frac{X \xrightarrow{\underline{a}[k]} X'}{X\backslash L \xrightarrow{\underline{a}[k]} X'\backslash L}\ a \notin L \qquad \text{rev con}\ \frac{X \xrightarrow{\underline{a}[k]} X'}{X \xrightarrow{\underline{a}[k]} S}\ S \stackrel{def}{=} X'$$

Figure 9.2: Reverse SOS rules of CCB-S. The condition (*) is $\gamma(a,d) = c$, and $b \in \mathcal{WA}$.

bours and for the BER example a maximum distance would be an appropriate condition. We see this as a promising area for further research.

9.3.2 Other future work

In Section 8.3 we presented a way to establish chemical process equivalence using a graph conversion. There are potentially other ways of doing this, e.g. using a term rewriting system. The application of such systems could also offer a better understanding of the rules. For example it would be interesting to see if the minimal number of transformations can be established.

So far we did not take energy into account. Since reactions involve energy, by either releasing or absorbing energy, and the laws of thermodynamics determine if a reaction happens or not, including energy flows is a potentially useful extension.

$$\text{aux1}\ \frac{\mathsf{std}(X) \quad \mathsf{fsh}[k](t)}{(t;b).X \xrightarrow{(b)[k]} (t;b[k]).X} \qquad \text{aux2}\ \frac{X \xrightarrow{(b)[k]} X' \quad \mathsf{fsh}[k](t)}{(t;a).X \xrightarrow{(b)[k]} (t;a).X'}$$

$$\text{concert}\ \frac{X \xrightarrow{(a)[k]} X' \quad X' \xrightarrow{\underline{b}[l]} X'' \qquad Y \xrightarrow{\alpha[k]} Y' \quad Y' \xrightarrow{\underline{d}[l]} Y''}{X \mid Y \xrightarrow{\{e[k],\underline{f}[l]\}} X'' \mid Y''}$$

$$(*)\text{concert act}\ \frac{X \xrightarrow{\{a[k],\underline{b}[l]\}} X' \quad \mathsf{fsh}[k](t)}{(t;a).X \xrightarrow{\{a[k],\underline{b}[l]\}} (t;a).X'}$$

$$\text{concert par}\ \frac{X \xrightarrow{\{a[k],\underline{b}[l]\}} X' \quad \mathsf{fsh}[k](Y)}{X \mid Y \xrightarrow{\{a[k],\underline{b}[l]\}} X' \mid Y} \qquad \text{concert rev}\ \frac{X \xrightarrow{\{a[k],\underline{b}[l]\}} X'}{X\backslash L \xrightarrow{\{a[k],\underline{b}[l]\}} X'\backslash L}\ (**)$$

Figure 9.3: SOS rules for concerted actions of CCB-S. The condition (*) is α is c or (c), $\gamma(b,c) = e$ for some $c \in \mathcal{A}$, $\gamma(a,d) = f$, $\mathsf{ds}(X,Y)\,R\,dis$ where $dis \in \mathbb{N}$, $\mathsf{conn}(X)\,R'\,con$, and $\mathsf{conn}(Y)\,R'\,con$ where $con \in \mathbb{N}$. The condition (**) is $a, \underline{h} \notin L \cup (L)$. Recall that $t \in \mathcal{AK}^*$.

Ideally it would give a more general modelling and exclude some unwanted paths, which so far can be executed in the calculus.

Another area for further work is the interdisciplinary application of the calculus and its possible extension. The principles described may be relevant in many other areas. If these are examined this could result in further extensions to CCB. A specific area here could be models of distributed computing or other complex computational systems where coordination of tasks is needed. Reversibility can help to make such system more resilient. Our calculus could help in finding ways to control reversibility in such systems.

The software tool demonstrated in Chapter 8 could be extended to form the core of a generic process calculus simulation tool. For this it would need to be modularized in order for a user to provide his own rules for parsing, infer transitions, executing processes etc. The use of the visitor pattern is an example of how this could be done. The extension could be done by rules being supplied in textual format, but also by code being provided by the user. With the help of Java techniques like introspection it would be possible to load code at runtime. Also the code for the display of the processes could be modular, so that different layout algorithms could be provided. Some might be optimized for the display of chemical structures, others could render the processes for different application areas.

Appendix: Record of the execution of the hydration of formaldehyde in water using CCBsimulation

Here we give a complete execution of a system of one formaldehyde and three water molecules, as described in Chapter 5. The execution contains all states from Figure 5.6. After each step we have executed those reactions which can realistically happen. Behind each reaction we give an explanation, using the following keys:

- [X +] new processes, which are followed in the next step. X refers to the states in Figure 5.6.
- [X -] leads to an already covered compound. These paths are shown by the program, since they represent a new transition. We do not execute them. These include reverse reactions of previously done transitions.
- [1] not possible because of steric hindrance, see Section 5.2.
- [2] single oxygen not possible, see Section 5.4.
- [3] transition to same state. This can for example happen if a water molecule abstract a hydrogen atom from an H_3O^+ molecule, producing a H_3O^+ and a water molecule again.
- [4] intramolecular reaction, see Section 5.4.

The resonances are not executed here as single steps, but only as concerted actions with the next step. As a consequence, $i7 \mid W \mid OH \mid W$ and $i7 \mid OH \mid OH \mid H3O$ are not reached directly, but are an intermediate state in a reaction.

All **bold** text are comments, added manually and not part of the program output.

```
Enter your process:
(((c1[1],c2[2],c3[3],c4[4];p).0 | (h1[1];p).0 | (h2[2];p).0 | (o1[3],
   ↪ o2[4],n).0) \ {c1,c2,c3,c4,h1,h2,o1,o2,c1h1,c2h2} | ((h3[5];p)
   ↪ .0 | (h4[6];p).0 | (o3[5],o4[6],n).0) \ {h3,h4,o3,o4} | ((h5
   ↪ [7];p).0 | (h6[8];p).0 | (o5[7],o6[8],n).0) \ {h5,h6,o5,o6} |
   ↪ ((h7[9];p).0 | (h8[10];p).0 | (o7[9],o8[10],n).0) \ {h7,h8,o7,
   ↪ o8} ) \ {n,p} [FA W W W +]
Enter your synchronisation function [format: a,a,b...):
c1,h1,c1h1,c1,h2,c1h2,c1,h3,c1h3,c1,h4,c1h4,c1,h5,c1h5,c1,h6,c1h6,c1,
   ↪ h7,c1h7,c1,h8,c1h8,c2,h1,c2h1,c2,h2,c2h2,c2,h3,c2h3,c2,h4,c2h4
   ↪ ,c2,h5,c2h5,c2,h6,c2h6,c2,h7,c2h7,c2,h8,c2h8,c3,h1,c3h1,c3,h2,
   ↪ c3h2,c3,h3,c3h3,c3,h4,c3h4,c3,h5,c3h5,c3,h6,c3h6,c3,h7,c3h7,c3
   ↪ ,h8,c3h8,c4,h1,c4h1,c4,h2,c4h2,c4,h3,c4h3,c4,h4,c4h4,c4,h5,
   ↪ c4h5,c4,h6,c4h6,c4,h7,c4h7,c4,h8,c4h8,o1,h1,o1h1,o1,h2,o1h2,o1
   ↪ ,h3,o1h3,o1,h4,o1h4,o1,h5,o1h5,o1,h6,o1h6,o1,h7,o1h7,o1,h7,
   ↪ o1h7,o2,h1,o2h1,o2,h2,o2h2,o2,h3,o2h3,o2,h4,o2h4,o2,h5,o2h5,o2
   ↪ ,h6,o2h6,o2,h7,o2h7,o2,h8,o2h8,o3,h1,o3h1,o3,h2,o3h2,o3,h3,
   ↪ o3h3,o3,h4,o3h4,o3,h5,o3h5,o3,h6,o3h6,o3,h7,o3h7,o3,h8,o3h8,o4
   ↪ ,h1,o4h1,o4,h2,o4h2,o4,h3,o4h3,o4,h4,o4h4,o4,h5,o4h5,o4,h6,
   ↪ o4h6,o4,h7,o4h7,o4,h8,o4h8,o5,h1,o5h1,o5,h2,o5h2,o4,h3,o5h3,o5
   ↪ ,h4,o5h4,o5,h5,o5h5,o5,h6,o5h6,o5,h7,o5h7,o5,h8,o5h8,o6,h1,
   ↪ o6h1,o6,h2,o6h2,o6,h3,o6h3,o6,h4,o6h4,o4,h5,o6h5,o6,h6,o6h6,o6
   ↪ ,h7,o6h7,o6,h8,o6h8,o7,h1,o7h1,o7,h2,o7h2,o7,h3,o7h3,o7,h4,
   ↪ o7h4,o4,h5,o7h5,o7,h6,o7h6,o7,h7,o7h7,o7,h8,o7h8,o8,h1,o8h1,o8
   ↪ ,h2,o8h2,o4,h3,o8h3,o8,h4,o8h4,o4,h5,o8h5,o8,h6,o8h6,o8,h7,
   ↪ o8h7,o8,h8,o8h8,c1,n,c1n,c2,n,c2n,c3,n,c3n,c4,n,c4n,h1,n,h1n,
   ↪ h2,n,h2n,h3,n,h3n,h4,n,h4n,h5,n,h5n,h6,n,h6n,h7,n,h7n,h8,n,h8n
   ↪ ,n,p,c1,o1,c1o1,c1,o2,c1o2,c1,o3,c1o3,c1,o4,c1o4,c1,o5,c1o5,c1
   ↪ ,o6,c1o6,c1,o7,c1o7,c1,o8,c1o8,c2,o1,c2o1,c2,o2,c2o2,c2,o3,
   ↪ c3o3,c2,o4,c2o4,c2,o5,c2o5,c2,o6,c2o6,c2,o7,c2o7,c2,o8,c2o8,c3
   ↪ ,o1,c3o1,c3,o2,c3o2,c3,o3,c3o3,c3,o4,c3o4,c3,o5,c3o5,c3,o6,
   ↪ c3o6,c3,o7,c3o7,c3,o8,c3o8,c4,o1,c4o1,c4,o2,c4o2,c4,o3,c4o3,c3
   ↪ ,o4,c3o4,c3,o5,c3o5,c3,o6,c3o6,c3,o7,c3o7,c3,o8,c3o8
Enter the weak actions [format: a,b,c...):
n,p

0: Ready for execution P0 ((c1[1],c2[2],#c3[3],c4[4];_p).0 | (h1[1];p
   ↪ ).0 | (h2[2];p).0 | (n,o1[3],o2[4]).0) \ {c1,c2,c3,c4,h1,h2,o1
```

```
   ↪ ,o2,c1h1,c2h2} | ((h3[5];p).0 | (h4[6];p).0 | (_n,o3[5],o4[6])
   ↪ .0) \ {h3,h4,o3,o4} | ((h5[7];p).0 | (h6[8];p).0 | (n,o5[7],o6
   ↪ [8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0 | (n,o7
   ↪ [9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [i2 W W +]
1: Ready for execution P1 ((c1[1],c2[2],c3[3],c4[4];p).0 | (h1[1];p)
   ↪ .0 | (h2[2];p).0 | (_n,o1[3],o2[4]).0) \ {c1,c2,c3,c4,h1,h2,o1
   ↪ ,o2,c1h1,c2h2} | ((#h3[5];_p).0 | (h4[6];p).0 | (n,o3[5],o4
   ↪ [6]).0) \ {h3,h4,o3,o4} | ((h5[7];p).0 | (h6[8];p).0 | (n,o5
   ↪ [7],o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0 |
   ↪ (n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [i6 W OH W +]
2: Ready for execution P2 ((#h3[5];_p).0 | (h4[6];p).0 | (n,o3[5],o4
   ↪ [6]).0) \ {h3,h4,o3,o4} | ((h5[7];p).0 | (h6[8];p).0 | (_n,o5
   ↪ [7],o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0 |
   ↪ (n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [FA W OH H3O +]
3 transitions ready, execute [a]ll or give [e]xclude or [i]nclude
   ↪ list or say [b]reakX
a

Transitions from 0
0: Ready for execution P0 ((c1[1],c2[2],#c4[4],c3[100];_p).0 | (h1
   ↪ [1];p).0 | (h2[2];p).0 | (n,o1,o2[4]).0) \ {c1,c2,c3,c4,h1,h2,
   ↪ o1,o2,c1h1,c2h2} | ((h3[5];p).0 | (h4[6];p).0 | (o3[5],o4[6],n
   ↪ [100]).0) \ {h3,h4,o3,o4} | ((h5[7];p).0 | (h6[8];p).0 | (_n,
   ↪ o5[7],o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0
   ↪ | (n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [1]
1: Ready for execution P1 ((c1[1],c2[2],c4[4],#c3[100];_p).0 | (h1
   ↪ [1];p).0 | (h2[2];p).0 | (n,o1,o2[4]).0) \ {c1,c2,c3,c4,h1,h2,
   ↪ o1,o2,c1h1,c2h2} | ((h3[5];p).0 | (h4[6];p).0 | (o3[5],o4[6],n
   ↪ [100]).0) \ {h3,h4,o3,o4} | ((h5[7];p).0 | (h6[8];p).0 | (_n,
   ↪ o5[7],o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0
   ↪ | (n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [1]
2: Ready for execution P2 ((c1[1],c2[2],c4[4],c3[100];p).0 | (h1[1];p
   ↪ ).0 | (h2[2];p).0 | (_n,o1,o2[4]).0) \ {c1,c2,c3,c4,h1,h2,o1,
   ↪ o2,c1h1,c2h2} | ((#h3[5];_p).0 | (h4[6];p).0 | (o3[5],o4[6],n
   ↪ [100]).0) \ {h3,h4,o3,o4} | ((h5[7];p).0 | (h6[8];p).0 | (n,o5
   ↪ [7],o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0 |
   ↪ (n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [4]
3: Ready for execution P3 ((c1[1],c2[2],c4[4],c3[100];p).0 | (h1[1];p
   ↪ ).0 | (h2[2];p).0 | (_n,o1,o2[4]).0) \ {c1,c2,c3,c4,h1,h2,o1,
```

```
   ↪ o2,c1h1,c2h2} | ((h3[5];p).0 | (h4[6];p).0 | (o3[5],o4[6],n
   ↪ [100]).0) \ {h3,h4,o3,o4} | ((#h5[7];_p).0 | (h6[8];p).0 | (n,
   ↪ o5[7],o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0
   ↪ | (n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [i8 OH W +]
4: Ready for execution P4 ((#h3[5];_p).0 | (h4[6];p).0 | (o3[5],o4
   ↪ [6],n[100]).0) \ {h3,h4,o3,o4} | ((h5[7];p).0 | (h6[8];p).0 |
   ↪ (_n,o5[7],o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p
   ↪ ).0 | (n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [i3 H3O W +]
5: Ready for execution P5 ((#h5[7];_p).0 | (h6[8];p).0 | (n,o5[7],o6
   ↪ [8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0 | (_n,o7
   ↪ [9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [i2 H3O OH +]

Transitions from 1
6: Ready for execution P6 ((c1[1],c2[2],#c3[3],c4[4];_p).0 | (h1[1];p
   ↪ ).0 | (h2[2];p).0 | (o1[3],o2[4],n[101]).0) \ {c1,c2,c3,c4,h1,
   ↪ h2,o1,o2,c1h1,c2h2} | ((h3[101];p).0 | (h4[6];p).0 | (_n,o3,o4
   ↪ [6]).0) \ {h3,h4,o3,o4} | ((h5[7];p).0 | (h6[8];p).0 | (n,o5
   ↪ [7],o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0 |
   ↪ (n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [MD W W +]
7: Ready for execution P7 ((c1[1],c2[2],#c3[3],c4[4];_p).0 | (h1[1];p
   ↪ ).0 | (h2[2];p).0 | (o1[3],o2[4],n[101]).0) \ {c1,c2,c3,c4,h1,
   ↪ h2,o1,o2,c1h1,c2h2} | ((h3[101];p).0 | (h4[6];p).0 | (n,o3,o4
   ↪ [6]).0) \ {h3,h4,o3,o4} | ((h5[7];p).0 | (h6[8];p).0 | (_n,o5
   ↪ [7],o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0 |
   ↪ (n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [i8 OH W -]
8: Ready for execution P8 ((#h3[101];_p).0 | (h4[6];p).0 | (n,o3,o4
   ↪ [6]).0) \ {h3,h4,o3,o4} | ((h5[7];p).0 | (h6[8];p).0 | (_n,o5
   ↪ [7],o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0 |
   ↪ (n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [FA W OH H3O - ]
9: Ready for execution P9 ((h3[101];p).0 | (#h4[6];_p).0 | (n,o3,o4
   ↪ [6]).0) \ {h3,h4,o3,o4} | ((h5[7];p).0 | (h6[8];p).0 | (_n,o5
   ↪ [7],o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0 |
   ↪ (n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [2]
10: Ready for execution P10 ((h3[101];p).0 | (h4[6];p).0 | (_n,o3,o4
   ↪ [6]).0) \ {h3,h4,o3,o4} | ((#h5[7];_p).0 | (h6[8];p).0 | (n,o5
   ↪ [7],o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0 |
   ↪ (n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [3]
```

```
11: Ready for execution P11 ((#h5[7];_p).0 | (h6[8];p).0 | (n,o5[7],
   ↪ o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0 | (_n,
   ↪ o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [i6 OH OH H3O +]

Transitions from 2
12: Ready for execution P12 ((c1[1],c2[2],#c3[3],c4[4];_p).0 | (h1
   ↪ [1];p).0 | (h2[2];p).0 | (n,o1[3],o2[4]).0) \ {c1,c2,c3,c4,h1,
   ↪ h2,o1,o2,c1h1,c2h2} | ((h3[102];p).0 | (h4[6];p).0 | (_n,o3,o4
   ↪ [6]).0) \ {h3,h4,o3,o4} | ((h5[7];p).0 | (h6[8];p).0 | (o5[7],
   ↪ o6[8],n[102]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0
   ↪  | (n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [i3 H3O W -]
13: Ready for execution P13 ((c1[1],c2[2],#c3[3],c4[4];_p).0 | (h1
   ↪ [1];p).0 | (h2[2];p).0 | (n,o1[3],o2[4]).0) \ {c1,c2,c3,c4,h1,
   ↪ h2,o1,o2,c1h1,c2h2} | ((h3[102];p).0 | (h4[6];p).0 | (n,o3,o4
   ↪ [6]).0) \ {h3,h4,o3,o4} | ((h5[7];p).0 | (h6[8];p).0 | (o5[7],
   ↪ o6[8],n[102]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0
   ↪  | (_n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [i2 H3O OH -]
14: Ready for execution P14 ((c1[1],c2[2],c3[3],c4[4];p).0 | (h1[1];p
   ↪ ).0 | (h2[2];p).0 | (_n,o1[3],o2[4]).0) \ {c1,c2,c3,c4,h1,h2,
   ↪ o1,o2,c1h1,c2h2} | ((#h3[102];_p).0 | (h4[6];p).0 | (n,o3,o4
   ↪ [6]).0) \ {h3,h4,o3,o4} | ((h5[7];p).0 | (h6[8];p).0 | (o5[7],
   ↪ o6[8],n[102]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0
   ↪  | (n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [i6 W OH W -]
15: Ready for execution P15 ((c1[1],c2[2],c3[3],c4[4];p).0 | (h1[1];p
   ↪ ).0 | (h2[2];p).0 | (_n,o1[3],o2[4]).0) \ {c1,c2,c3,c4,h1,h2,
   ↪ o1,o2,c1h1,c2h2} | ((h3[102];p).0 | (#h4[6];_p).0 | (n,o3,o4
   ↪ [6]).0) \ {h3,h4,o3,o4} | ((h5[7];p).0 | (h6[8];p).0 | (o5[7],
   ↪ o6[8],n[102]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0
   ↪  | (n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [2]
16: Ready for execution P16 ((c1[1],c2[2],c3[3],c4[4];p).0 | (h1[1];p
   ↪ ).0 | (h2[2];p).0 | (_n,o1[3],o2[4]).0) \ {c1,c2,c3,c4,h1,h2,
   ↪ o1,o2,c1h1,c2h2} | ((h3[102];p).0 | (h4[6];p).0 | (n,o3,o4[6])
   ↪ .0) \ {h3,h4,o3,o4} | ((h5[7];p).0 | (h6[8];p).0 | (o5[7],o6
   ↪ [8],n[102]).0) \ {h5,h6,o5,o6} | ((#h7[9];_p).0 | (h8[10];p).0
   ↪  | (n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [i6 OH OH H3O -]
17: Ready for execution P17 ((#h3[102];_p).0 | (h4[6];p).0 | (n,o3,o4
   ↪ [6]).0) \ {h3,h4,o3,o4} | ((h5[7];p).0 | (h6[8];p).0 | (o5[7],
   ↪ o6[8],n[102]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0
   ↪  | (_n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [3]
```

```
18: Ready for execution P18 ((h3[102];p).0 | (#h4[6];_p).0 | (n,o3,o4
   ↪ [6]).0) \ {h3,h4,o3,o4} | ((h5[7];p).0 | (h6[8];p).0 | (o5[7],
   ↪ o6[8],n[102]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0
   ↪  | (_n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [2]
19: Ready for execution P19 ((h3[102];p).0 | (h4[6];p).0 | (_n,o3,o4
   ↪ [6]).0) \ {h3,h4,o3,o4} | ((#h5[7];_p).0 | (h6[8];p).0 | (o5
   ↪ [7],o6[8],n[102]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];
   ↪ p).0 | (n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [FA W W W -]
20 transitions ready, execute [a]ll or give [e]xclude or [i]nclude
   ↪ list or say [b]reakX
i3,4,5,6,11,12

Transitions from 3
0: Ready for execution P0 ((c1[1],c2[2],#c4[4],c3[100];_p).0 | (h1
   ↪ [1];p).0 | (h2[2];p).0 | (n,o2[4],o1[103]).0) \ {c1,c2,c3,c4,
   ↪ h1,h2,o1,o2,c1h1,c2h2} | ((h3[5];p).0 | (h4[6];p).0 | (o3[5],
   ↪ o4[6],n[100]).0) \ {h3,h4,o3,o4} | ((h5[103];p).0 | (h6[8];p)
   ↪ .0 | (_n,o5,o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8
   ↪ [10];p).0 | (n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [1]
1: Ready for execution P1 ((c1[1],c2[2],#c4[4],c3[100];_p).0 | (h1
   ↪ [1];p).0 | (h2[2];p).0 | (n,o2[4],o1[103]).0) \ {c1,c2,c3,c4,
   ↪ h1,h2,o1,o2,c1h1,c2h2} | ((h3[5];p).0 | (h4[6];p).0 | (o3[5],
   ↪ o4[6],n[100]).0) \ {h3,h4,o3,o4} | ((h5[103];p).0 | (h6[8];p)
   ↪ .0 | (n,o5,o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];
   ↪ p).0 | (_n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [1]
2: Ready for execution P2 ((c1[1],c2[2],c4[4],#c3[100];_p).0 | (h1
   ↪ [1];p).0 | (h2[2];p).0 | (n,o2[4],o1[103]).0) \ {c1,c2,c3,c4,
   ↪ h1,h2,o1,o2,c1h1,c2h2} | ((h3[5];p).0 | (h4[6];p).0 | (o3[5],
   ↪ o4[6],n[100]).0) \ {h3,h4,o3,o4} | ((h5[103];p).0 | (h6[8];p)
   ↪ .0 | (_n,o5,o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8
   ↪ [10];p).0 | (n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [1]
3: Ready for execution P3 ((c1[1],c2[2],c4[4],c3[100];p).0 | (h1[1];p
   ↪ ).0 | (h2[2];p).0 | (_n,o2[4],o1[103]).0) \ {c1,c2,c3,c4,h1,h2
   ↪ ,o1,o2,c1h1,c2h2} | ((h3[5];p).0 | (h4[6];p).0 | (o3[5],o4[6],
   ↪ n[100]).0) \ {h3,h4,o3,o4} | ((h5[103];p).0 | (#h6[8];_p).0 |
   ↪ (n,o5,o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0
   ↪ | (n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [i14 OH OH +]
4: Ready for execution P4 ((#h3[5];_p).0 | (h4[6];p).0 | (o3[5],o4
   ↪ [6],n[100]).0) \ {h3,h4,o3,o4} | ((h5[103];p).0 | (h6[8];p).0
```

```
   ↪ | (n,o5,o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p)
   ↪ .0 | (_n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [MD OH H3O +]
5: Ready for execution P5 ((#h5[103];_p).0 | (h6[8];p).0 | (n,o5,o6
   ↪ [8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0 | (_n,o7
   ↪ [9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [i2 H3O OH -]
6: Ready for execution P6 ((h5[103];p).0 | (#h6[8];_p).0 | (n,o5,o6
   ↪ [8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0 | (_n,o7
   ↪ [9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [2]

Transitions from 5
7: Ready for execution P7 ((c1[1],c2[2],#c4[4],c3[100];_p).0 | (h1
   ↪ [1];p).0 | (h2[2];p).0 | (n,o1,o2[4]).0) \ {c1,c2,c3,c4,h1,h2,
   ↪ o1,o2,c1h1,c2h2} | ((h3[5];p).0 | (h4[6];p).0 | (o3[5],o4[6],n
   ↪ [100]).0) \ {h3,h4,o3,o4} | ((h5[105];p).0 | (h6[8];p).0 | (_n
   ↪ ,o5,o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0 |
   ↪ (o7[9],o8[10],n[105]).0) \ {h7,h8,o7,o8};np;0 [1]
8: Ready for execution P8 ((c1[1],c2[2],c4[4],#c3[100];_p).0 | (h1
   ↪ [1];p).0 | (h2[2];p).0 | (n,o1,o2[4]).0) \ {c1,c2,c3,c4,h1,h2,
   ↪ o1,o2,c1h1,c2h2} | ((h3[5];p).0 | (h4[6];p).0 | (o3[5],o4[6],n
   ↪ [100]).0) \ {h3,h4,o3,o4} | ((h5[105];p).0 | (h6[8];p).0 | (_n
   ↪ ,o5,o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0 |
   ↪ (o7[9],o8[10],n[105]).0) \ {h7,h8,o7,o8};np;0 [1]
9: Ready for execution P9 ((c1[1],c2[2],c4[4],c3[100];p).0 | (h1[1];p
   ↪ ).0 | (h2[2];p).0 | (_n,o1,o2[4]).0) \ {c1,c2,c3,c4,h1,h2,o1,
   ↪ o2,c1h1,c2h2} | ((#h3[5];_p).0 | (h4[6];p).0 | (o3[5],o4[6],n
   ↪ [100]).0) \ {h3,h4,o3,o4} | ((h5[105];p).0 | (h6[8];p).0 | (n,
   ↪ o5,o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0 | (
   ↪ o7[9],o8[10],n[105]).0) \ {h7,h8,o7,o8};np;0 [4]
10: Ready for execution P10 ((c1[1],c2[2],c4[4],c3[100];p).0 | (h1
   ↪ [1];p).0 | (h2[2];p).0 | (_n,o1,o2[4]).0) \ {c1,c2,c3,c4,h1,h2
   ↪ ,o1,o2,c1h1,c2h2} | ((h3[5];p).0 | (h4[6];p).0 | (o3[5],o4[6],
   ↪ n[100]).0) \ {h3,h4,o3,o4} | ((#h5[105];_p).0 | (h6[8];p).0 |
   ↪ (n,o5,o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0
   ↪ | (o7[9],o8[10],n[105]).0) \ {h7,h8,o7,o8};np;0 [i8 W OH -]
11: Ready for execution P11 ((h5[105];p).0 | (h6[8];p).0 | (_n,o5,o6
   ↪ [8]).0) \ {h5,h6,o5,o6} | ((#h7[9];_p).0 | (h8[10];p).0 | (o7
   ↪ [9],o8[10],n[105]).0) \ {h7,h8,o7,o8};np;0 [i2 W W -]

Transitions from 6
```

```
12: Ready for execution P12 ((c1[1],c2[2],#c4[4],c3[106];_p).0 | (h1
   ↪ [1];p).0 | (h2[2];p).0 | (n,o2[4],o1[101]).0) \ {c1,c2,c3,c4,
   ↪ h1,h2,o1,o2,c1h1,c2h2} | ((h3[101];p).0 | (h4[6];p).0 | (_n,o4
   ↪ [6],o3[106]).0) \ {h3,h4,o3,o4} | ((h5[7];p).0 | (h6[8];p).0 |
   ↪  (n,o5[7],o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p
   ↪ ).0 | (n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [1]
13: Ready for execution P13 ((c1[1],c2[2],#c4[4],c3[106];_p).0 | (h1
   ↪ [1];p).0 | (h2[2];p).0 | (n,o2[4],o1[101]).0) \ {c1,c2,c3,c4,
   ↪ h1,h2,o1,o2,c1h1,c2h2} | ((h3[101];p).0 | (h4[6];p).0 | (n,o4
   ↪ [6],o3[106]).0) \ {h3,h4,o3,o4} | ((h5[7];p).0 | (h6[8];p).0 |
   ↪  (_n,o5[7],o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];
   ↪ p).0 | (n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [1]
14: Ready for execution P14 ((c1[1],c2[2],c4[4],#c3[106];_p).0 | (h1
   ↪ [1];p).0 | (h2[2];p).0 | (n,o2[4],o1[101]).0) \ {c1,c2,c3,c4,
   ↪ h1,h2,o1,o2,c1h1,c2h2} | ((h3[101];p).0 | (h4[6];p).0 | (_n,o4
   ↪ [6],o3[106]).0) \ {h3,h4,o3,o4} | ((h5[7];p).0 | (h6[8];p).0 |
   ↪  (n,o5[7],o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p
   ↪ ).0 | (n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [1]
15: Ready for execution P15 ((c1[1],c2[2],c4[4],c3[106];p).0 | (h1
   ↪ [1];p).0 | (h2[2];p).0 | (_n,o2[4],o1[101]).0) \ {c1,c2,c3,c4,
   ↪ h1,h2,o1,o2,c1h1,c2h2} | ((h3[101];p).0 | (#h4[6];_p).0 | (n,
   ↪ o4[6],o3[106]).0) \ {h3,h4,o3,o4} | ((h5[7];p).0 | (h6[8];p).0
   ↪  | (n,o5[7],o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8
   ↪ [10];p).0 | (n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [4]
16: Ready for execution P16 ((#h3[101];_p).0 | (h4[6];p).0 | (n,o4
   ↪ [6],o3[106]).0) \ {h3,h4,o3,o4} | ((h5[7];p).0 | (h6[8];p).0 |
   ↪  (_n,o5[7],o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];
   ↪ p).0 | (n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [i3 H3O W -]
17: Ready for execution P17 ((#h5[7];_p).0 | (h6[8];p).0 | (n,o5[7],
   ↪ o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0 | (_n,
   ↪ o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [MD OH H3O -]

Transitions from 4
18: Ready for execution P18 ((c1[1],c2[2],#c4[4],c3[100];_p).0 | (h1
   ↪ [1];p).0 | (h2[2];p).0 | (n,o1,o2[4]).0) \ {c1,c2,c3,c4,h1,h2,
   ↪ o1,o2,c1h1,c2h2} | ((h3[104];p).0 | (h4[6];p).0 | (_n,o4[6],o3
   ↪ [100]).0) \ {h3,h4,o3,o4} | ((h5[7];p).0 | (h6[8];p).0 | (o5
   ↪ [7],o6[8],n[104]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];
   ↪ p).0 | (n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [1]
```

```
19: Ready for execution P19 ((c1[1],c2[2],#c4[4],c3[100];_p).0 | (h1
   ↪ [1];p).0 | (h2[2];p).0 | (n,o1,o2[4]).0) \ {c1,c2,c3,c4,h1,h2,
   ↪ o1,o2,c1h1,c2h2} | ((h3[104];p).0 | (h4[6];p).0 | (n,o4[6],o3
   ↪ [100]).0) \ {h3,h4,o3,o4} | ((h5[7];p).0 | (h6[8];p).0 | (o5
   ↪ [7],o6[8],n[104]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];
   ↪ p).0 | (_n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [1]
20: Ready for execution P20 ((c1[1],c2[2],c4[4],#c3[100];_p).0 | (h1
   ↪ [1];p).0 | (h2[2];p).0 | (n,o1,o2[4]).0) \ {c1,c2,c3,c4,h1,h2,
   ↪ o1,o2,c1h1,c2h2} | ((h3[104];p).0 | (h4[6];p).0 | (_n,o4[6],o3
   ↪ [100]).0) \ {h3,h4,o3,o4} | ((h5[7];p).0 | (h6[8];p).0 | (o5
   ↪ [7],o6[8],n[104]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];
   ↪ p).0 | (n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [1]
21: Ready for execution P21 ((c1[1],c2[2],c4[4],#c3[100];_p).0 | (h1
   ↪ [1];p).0 | (h2[2];p).0 | (n,o1,o2[4]).0) \ {c1,c2,c3,c4,h1,h2,
   ↪ o1,o2,c1h1,c2h2} | ((h3[104];p).0 | (h4[6];p).0 | (n,o4[6],o3
   ↪ [100]).0) \ {h3,h4,o3,o4} | ((h5[7];p).0 | (h6[8];p).0 | (o5
   ↪ [7],o6[8],n[104]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];
   ↪ p).0 | (_n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [1]
22: Ready for execution P22 ((c1[1],c2[2],c4[4],c3[100];p).0 | (h1
   ↪ [1];p).0 | (h2[2];p).0 | (_n,o1,o2[4]).0) \ {c1,c2,c3,c4,h1,h2
   ↪ ,o1,o2,c1h1,c2h2} | ((#h3[104];_p).0 | (h4[6];p).0 | (n,o4[6],
   ↪ o3[100]).0) \ {h3,h4,o3,o4} | ((h5[7];p).0 | (h6[8];p).0 | (o5
   ↪ [7],o6[8],n[104]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];
   ↪ p).0 | (n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [MD W W -]
23: Ready for execution P23 ((c1[1],c2[2],c4[4],c3[100];p).0 | (h1
   ↪ [1];p).0 | (h2[2];p).0 | (_n,o1,o2[4]).0) \ {c1,c2,c3,c4,h1,h2
   ↪ ,o1,o2,c1h1,c2h2} | ((h3[104];p).0 | (h4[6];p).0 | (n,o4[6],o3
   ↪ [100]).0) \ {h3,h4,o3,o4} | ((h5[7];p).0 | (h6[8];p).0 | (o5
   ↪ [7],o6[8],n[104]).0) \ {h5,h6,o5,o6} | ((#h7[9];_p).0 | (h8
   ↪ [10];p).0 | (n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [MD H3O
   ↪ HO -]
24: Ready for execution P24 ((h3[104];p).0 | (#h4[6];_p).0 | (n,o4
   ↪ [6],o3[100]).0) \ {h3,h4,o3,o4} | ((h5[7];p).0 | (h6[8];p).0 |
   ↪  (o5[7],o6[8],n[104]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8
   ↪ [10];p).0 | (_n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [i13 H3O
   ↪  H3O +]
25: Ready for execution P25 ((h3[104];p).0 | (h4[6];p).0 | (_n,o4[6],
   ↪ o3[100]).0) \ {h3,h4,o3,o4} | ((#h5[7];_p).0 | (h6[8];p).0 | (
   ↪ o5[7],o6[8],n[104]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8
```

```
   ↪ [10];p).0 | (n,o7[9],o8[10]).0) \ {h7,h8,o7,o8};np;0 [i2 W W
   ↪ -]

Transitions from 11
26: Ready for execution P26 ((c1[1],c2[2],#c3[3],c4[4];_p).0 | (h1
   ↪ [1];p).0 | (h2[2];p).0 | (o1[3],o2[4],n[101]).0) \ {c1,c2,c3,
   ↪ c4,h1,h2,o1,o2,c1h1,c2h2} | ((h3[101];p).0 | (h4[6];p).0 | (_n
   ↪ ,o3,o4[6]).0) \ {h3,h4,o3,o4} | ((h5[107];p).0 | (h6[8];p).0 |
   ↪  (n,o5,o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0
   ↪  | (o7[9],o8[10],n[107]).0) \ {h7,h8,o7,o8};np;0 [MD OH H3O +]
27: Ready for execution P27 ((#h3[101];_p).0 | (h4[6];p).0 | (n,o3,o4
   ↪ [6]).0) \ {h3,h4,o3,o4} | ((h5[107];p).0 | (h6[8];p).0 | (_n,
   ↪ o5,o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0 | (
   ↪ o7[9],o8[10],n[107]).0) \ {h7,h8,o7,o8};np;0 [FA W OH H3O -]
28: Ready for execution P28 ((h3[101];p).0 | (#h4[6];_p).0 | (n,o3,o4
   ↪ [6]).0) \ {h3,h4,o3,o4} | ((h5[107];p).0 | (h6[8];p).0 | (_n,
   ↪ o5,o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0 | (
   ↪ o7[9],o8[10],n[107]).0) \ {h7,h8,o7,o8};np;0 (i6 o w h3o) [2]
29: Ready for execution P29 ((h3[101];p).0 | (h4[6];p).0 | (_n,o3,o4
   ↪ [6]).0) \ {h3,h4,o3,o4} | ((#h5[107];_p).0 | (h6[8];p).0 | (n,
   ↪ o5,o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0 | (
   ↪ o7[9],o8[10],n[107]).0) \ {h7,h8,o7,o8};np;0 [i6 W W OH -]
30 transitions ready, execute [a]ll or give [e]xclude or [i]nclude
   ↪ list or say [b]reakX
i4,24

Transitions from 24
0: Ready for execution P0 ((c1[1],c2[2],#c4[4],c3[100];_p).0 | (h1
   ↪ [1];p).0 | (h2[2];p).0 | (n,o1,o2[4]).0) \ {c1,c2,c3,c4,h1,h2,
   ↪ o1,o2,c1h1,c2h2} | ((h3[104];p).0 | (h4[110];p).0 | (_n,o4,o3
   ↪ [100]).0) \ {h3,h4,o3,o4} | ((h5[7];p).0 | (h6[8];p).0 | (o5
   ↪ [7],o6[8],n[104]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];
   ↪ p).0 | (o7[9],o8[10],n[110]).0) \ {h7,h8,o7,o8};np;0 [1]
1: Ready for execution P1 ((c1[1],c2[2],c4[4],#c3[100];_p).0 | (h1
   ↪ [1];p).0 | (h2[2];p).0 | (n,o1,o2[4]).0) \ {c1,c2,c3,c4,h1,h2,
   ↪ o1,o2,c1h1,c2h2} | ((h3[104];p).0 | (h4[110];p).0 | (_n,o4,o3
   ↪ [100]).0) \ {h3,h4,o3,o4} | ((h5[7];p).0 | (h6[8];p).0 | (o5
   ↪ [7],o6[8],n[104]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];
   ↪ p).0 | (o7[9],o8[10],n[110]).0) \ {h7,h8,o7,o8};np;0 [1]
```

```
2: Ready for execution P2 ((c1[1],c2[2],c4[4],c3[100];p).0 | (h1[1];p
   ↪ ).0 | (h2[2];p).0 | (_n,o1,o2[4]).0) \ {c1,c2,c3,c4,h1,h2,o1,
   ↪ o2,c1h1,c2h2} | ((#h3[104];_p).0 | (h4[110];p).0 | (n,o4,o3
   ↪ [100]).0) \ {h3,h4,o3,o4} | ((h5[7];p).0 | (h6[8];p).0 | (o5
   ↪ [7],o6[8],n[104]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];
   ↪ p).0 | (o7[9],o8[10],n[110]).0) \ {h7,h8,o7,o8};np;0 [i3 W H3O
   ↪  -]

Transitions from 4
3: Ready for execution P3 ((c1[1],c2[2],#c4[4],c3[100];_p).0 | (h1
   ↪ [1];p).0 | (h2[2];p).0 | (n,o2[4],o1[103]).0) \ {c1,c2,c3,c4,
   ↪ h1,h2,o1,o2,c1h1,c2h2} | ((h3[109];p).0 | (h4[6];p).0 | (_n,o4
   ↪ [6],o3[100]).0) \ {h3,h4,o3,o4} | ((h5[103];p).0 | (h6[8];p).0
   ↪  | (n,o5,o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p)
   ↪ .0 | (o7[9],o8[10],n[109]).0) \ {h7,h8,o7,o8};np;0 [1]
4: Ready for execution P4 ((c1[1],c2[2],#c4[4],c3[100];_p).0 | (h1
   ↪ [1];p).0 | (h2[2];p).0 | (n,o2[4],o1[103]).0) \ {c1,c2,c3,c4,
   ↪ h1,h2,o1,o2,c1h1,c2h2} | ((h3[109];p).0 | (h4[6];p).0 | (n,o4
   ↪ [6],o3[100]).0) \ {h3,h4,o3,o4} | ((h5[103];p).0 | (h6[8];p).0
   ↪  | (_n,o5,o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p
   ↪ ).0 | (o7[9],o8[10],n[109]).0) \ {h7,h8,o7,o8};np;0 [1]
5: Ready for execution P5 ((c1[1],c2[2],c4[4],c3[100];p).0 | (h1[1];p
   ↪ ).0 | (h2[2];p).0 | (_n,o2[4],o1[103]).0) \ {c1,c2,c3,c4,h1,h2
   ↪ ,o1,o2,c1h1,c2h2} | ((#h3[109];_p).0 | (h4[6];p).0 | (n,o4[6],
   ↪ o3[100]).0) \ {h3,h4,o3,o4} | ((h5[103];p).0 | (h6[8];p).0 | (
   ↪ n,o5,o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0 |
   ↪  (o7[9],o8[10],n[109]).0) \ {h7,h8,o7,o8};np;0 [i8 OH W -]
6: Ready for execution P6 ((c1[1],c2[2],c4[4],c3[100];p).0 | (h1[1];p
   ↪ ).0 | (h2[2];p).0 | (_n,o2[4],o1[103]).0) \ {c1,c2,c3,c4,h1,h2
   ↪ ,o1,o2,c1h1,c2h2} | ((h3[109];p).0 | (#h4[6];_p).0 | (n,o4[6],
   ↪ o3[100]).0) \ {h3,h4,o3,o4} | ((h5[103];p).0 | (h6[8];p).0 | (
   ↪ n,o5,o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0 |
   ↪  (o7[9],o8[10],n[109]).0) \ {h7,h8,o7,o8};np;0 [4]
7: Ready for execution P7 ((c1[1],c2[2],c4[4],c3[100];p).0 | (h1[1];p
   ↪ ).0 | (h2[2];p).0 | (_n,o2[4],o1[103]).0) \ {c1,c2,c3,c4,h1,h2
   ↪ ,o1,o2,c1h1,c2h2} | ((h3[109];p).0 | (h4[6];p).0 | (n,o4[6],o3
   ↪ [100]).0) \ {h3,h4,o3,o4} | ((h5[103];p).0 | (#h6[8];_p).0 | (
   ↪ n,o5,o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0 |
   ↪  (o7[9],o8[10],n[109]).0) \ {h7,h8,o7,o8};np;0 [4]
```

```
8: Ready for execution P8 ((#h3[109];_p).0 | (h4[6];p).0 | (n,o4[6],
   ↪ o3[100]).0) \ {h3,h4,o3,o4} | ((h5[103];p).0 | (h6[8];p).0 | (
   ↪ _n,o5,o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0
   ↪ | (o7[9],o8[10],n[109]).0) \ {h7,h8,o7,o8};np;0 [3]
9: Ready for execution P9 ((h3[109];p).0 | (#h4[6];_p).0 | (n,o4[6],
   ↪ o3[100]).0) \ {h3,h4,o3,o4} | ((h5[103];p).0 | (h6[8];p).0 | (
   ↪ _n,o5,o6[8]).0) \ {h5,h6,o5,o6} | ((h7[9];p).0 | (h8[10];p).0
   ↪ | (o7[9],o8[10],n[109]).0) \ {h7,h8,o7,o8};np;0 [i3 H3O W -]
10 transitions ready, execute [a]ll or give [e]xclude or [i]nclude
   ↪ list or say [b]reakX
```

Bibliography

[1] The Biochemical Abstract Machine BIOCHAM 3, 2017. `https://lifeware.inria.fr/biocham/`. Accessed 3-November-2017.

[2] Holger Bock Axelsen and Robert Glück. What Do Reversible Programs Compute? In Martin Hofmann, editor, *Foundations of Software Science and Computational Structures*, pages 42–56. Springer Berlin Heidelberg, 2011.

[3] Josef C. M. Baeten and W. Peter Weijland. *Process Algebra (Cambridge Tracts in Theoretical Computer Science 18)*. Cambridge University Press, 1990.

[4] Caroline Baroukh, Anthony Rowe, and Yike Guo. Process calculi for systems biology and applications in severe asthma. In *2010 IEEE International Conference on Bioinformatics and Biomedicine Workshops*, pages 217–222. IEEE, 2010.

[5] Kamila Barylska, Maciej Koutny, Łukasz Mikulski, and Marcin Piątkowski. Reversible computation vs. reversibility in Petri nets. *Science of Computer Programming*, 151:48 – 60, 2018. Special issue of the 8th Conference on Reversible Computation (RC2016).

[6] C. H. Bennett. Logical Reversibility of Computation. *IBM J. Res. Dev.*, 17(6):525–532, November 1973.

[7] Gerard Berry and Gerard Boudol. The Chemical Abstract Machine. In *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '90*, pages 81–94. ACM, 1990.

[8] Pierre Boutillier, Jérôme Feret, Jean Krivine, and Lý Kim Quyên. KaSim & KaSa reference manual (release 3.90). `http://dev.executableknowledge.org/docs/KaSim-manual-master/KaSim_manual.htm`. Accessed: 2-September-2016.

[9] Muffy Calder, Stephen Gilmore, and Jane Hillston. *Modelling the Influence of RKIP on the ERK Signalling Pathway Using the Stochastic Process Algebra PEPA*, In Corrado Priami, Anna Ingólfsdóttir, Bud Mishra, and Hanne Riis Nielson, editors, *Transactions on Computational Systems Biology VII*, pages 1–23. Springer, 2006.

[10] Giulio Caravagna and Jane Hillston. Bio-PEPAd: A non-Markovian extension of Bio-PEPA. *Theoretical Computer Science*, 419(Supplement C):26–49, 2012.

[11] Luca Cardelli. Brane Calculi. In Vincent Danos and Vincent Schachter, editors, *Computational Methods in Systems Biology*, volume 3082 of *Lecture Notes in Computer Science*, pages 257–278. Springer, 2005.

[12] Luca Cardelli and Andrew D. Gordon. Mobile Ambients. In *Proceedings of the First International Conference on Foundations of Software Science and Computation Structure*, FOSSACS '98, pages 140–155. Springer, 1998.

[13] Raymond Chang and Ken Goldsby. *General Chemistry. The Essential Concepts*. McGraw-Hill, 2014.

[14] Matteo Cimini, MohammadReza Mousavi, Michel A. Reniers, and Murdoch J. Gabbay. Nominal SOS. *Electronic Notes in Theoretical Computer Science*, 286:103 – 116, 2012. Proceedings of the 28th Conference on the Mathematical Foundations of Programming Semantics (MFPS XXVIII).

[15] Federica Ciocchetta and Jane Hillston. Bio-PEPA: A Framework for the Modelling and Analysis of Biological Systems. *Theor. Comput. Sci.*, 410(33-34):3065–3084, August 2009.

[16] Federica Ciocchetta, Jane Hillston, Martin Kos, and David Tollervey. Modelling co-transcriptional cleavage in the synthesis of yeast pre-rRNA. *Theoretical Computer Science*, 408(1):41–54, 2008.

[17] Jonathan Claydon, Nick Greeves, Stuart Warren, and Peter Wothers. *Organic Chemistry*. Oxford University Press, 2001.

[18] Rance Cleaveland, Gerald Löttgen, and V. Natarajan. Priority in Process Algebra. In Jan A. Bergstra, Alban Ponse, and Scott Allen Smolka, editors, *Handbook of Process Algebra*, chapter 12, pages 711–765. Elsevier Science, 2001.

[19] Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(10):1367–1372, 2004.

[20] Ioana Cristescu, Jean Krivine, and Daniele Varacca. A Compositional Semantics for the Reversible p-Calculus. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 388–397. IEEE Computer Society, 2013.

[21] Ioana Cristescu, Jean Krivine, and Daniele Varacca. *Rigid Families for the Reversible π-Calculus*, In Simon Devitt and Ivan Lanese, editors, *Reversible Computation: 8th International Conference, RC 2016*, volume 9720 of *Lecture Notes in Computer Science*, pages 3–19. Springer, 2016.

[22] Zoltán Csörnyei and Gergely Dévai. *An Introduction to the Lambda Calculus*, In Zoltán Horváth, Rinus Plasmeijer, Anna Soós, and Viktória Zsók, editors, *Central European Functional Programming School: Second Summer School, CEFP 2007, Cluj-Napoca, Romania, June 23-30, 2007, Revised Selected Lectures*, pages 87–111. Springer Berlin Heidelberg, 2008.

[23] Michele Curti, Pierpaolo Degano, and Cosima Tatiana Baldari. Causal π-Calculus for Biochemical Modelling. In Corrado Priami, editor, *Computational Methods in Systems Biology*, volume 2602 of *Lecture Notes in Computer Science*, pages 21–34. Springer, 2003.

[24] Vincent Danos, Jerome Feret, Walter Fontana, Russell Harmer, Jonathan Hayman, Jean Krivine, Chris Thompson-Walsh, and Glynn Winskel. Graphs, Rewriting and Pathway Reconstruction for Rule-Based Models. In Deepak D'Souza, Telikepalli Kavitha, and Jaikumar Radhakrishnan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012)*, volume 18 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 276–288. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2012.

[25] Vincent Danos, Jérôme Feret, Walter Fontana, Russell Harmer, and Jean Krivine. Rule-Based Modelling of Cellular Signalling. In Luís Caires and Vasco T. Vasconcelos, editors, *CONCUR 2007 - Concurrency Theory*, volume 4703 of *Lecture Notes in Computer Science*, pages 17–41. Springer, 2007.

[26] Vincent Danos and Jean Krivine. Reversible Communicating Systems. In Philippa Gardner and Nobuko Yoshida, editors, *CONCUR 2004 - Concurrency Theory*, volume 3170 of *Lecture Notes in Computer Science*, pages 292–307. Springer, 2004.

[27] Vincent Danos and Jean Krivine. Formal Molecular Biology Done in CCS-R. *Electronic Notes in Theoretical Computer Science*, 180(3):31–49, 2007. Proceedings of the First Workshop on Concurrent Models in Molecular Biology (BioConcur 2003).

[28] Vincent Danos and Cosimo Laneve. Formal molecular biology. *Theoretical Computer Science*, 325(1):69–110, 2004.

[29] Vincent Danos and Sylvain Pradalier. *Projective Brane Calculus*, In Vincent Danos and Vincent Schachter, editors, *Computational Methods in Systems Biology: International Conference CMSB 2004, Paris, France, May 26-28, 2004, Revised Selected Papers*, volume 3082 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 2005.

[30] Centre for Industry Education Collaboration Department of Chemistry, University of York. The Essential Chemical Industry - online. Methanal (Formaldehyde), 2016. `http://www.essentialchemicalindustry.org/chemicals/methanal.html`. Accessed 22-January-2018.

[31] Mariangiola Dezani-Ciancaglini and Paola Giannini. Reversible Multiparty Sessions with Checkpoints. In Daniel Gebler and Kirstin Peters, editors, Proceedings Combined 23rd International Workshop on *Expressiveness in Concurrency* and 13th Workshop on *Structural Operational Semantics,* Québec City, Canada, 22nd August 2016, volume 222 of *Electronic Proceedings in Theoretical Computer Science*, pages 60–74. Open Publishing Association, 2016.

[32] Jakob Engblom. A review of reverse debugging. In *Proceedings of the 2012 System, Software, SoC and Silicon Debug Conference*, pages 1–6, Sept 2012.

[33] Ralph Johnson Richard Helm Erich Gamma, John Vlissides. *Design Patterns.* Pearson, 1994.

[34] François Fages and Sylvain Soliman. *Formal Cell Biology in Biocham*, In Marco Bernardo, Pierpaolo Degano, and Gianluigi Zavattaro, editors, *Formal Methods for Computational Systems Biology: 8th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2008, Advanced Lectures*, volume 5016 of *Lecture Notes in Computer Science*, pages 54–80. Springer, 2008.

[35] François Fages, Sylvain Soliman, and Nathalie Chabrier-Rivier. Modelling and querying interaction networks in the biochemical abstract machine biocham. *Journal of Biological Physics and Chemistry*, 4:64–73, 2004.

[36] Wan Fokkink. *Introduction to Process Algebra (Texts in Theoretical Computer Science. An EATCS Series).* Springer, 2000.

[37] W. Fontana and L.W. Buss. The Barrier of Objects: From Dynamical Systems to Bounded Organizations. IIASA working paper, IIASA, Laxenburg, Austria, March 1996.

[38] Cecil H. Fox, Frank B. Johnson, John Whiting, and Peter P. Roller. Formaldehyde fixation. *Journal of Histochemistry and Cytochemistry*, 33(8):845–853, 1985.

[39] Michael P. Frank. *Reversibility for Efficient Computing*. PhD thesis, Cambridge, MA, USA, 1999. AAI0800784.

[40] Murdoch Gabbay and Andrew Pitts. A new approach to abstract syntax involving binders. In *Proceedings. 14th Symposium on Logic in Computer Science (Cat. No. PR00158)*, pages 214–224, July 1999.

[41] Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977.

[42] Maria Luisa Guerriero, Davide Prandi, Corrado Priami, and Paola Quaglia. *Process Calculi Abstractions for Biology*, In Anne Condon, David Harel, Joost N. Kok, Arto Salomaa, and Erik Winfree, editors, *Algorithmic Bioprocesses*, pages 463–486. Springer, 2009.

[43] Tue Haulund, Torben Ægidius Mogensen, and Robert Glück. Implementing Reversible Object-Oriented Language Features on Reversible Machines. In Iain Phillips and Hafizur Rahaman, editors, *Reversible Computation*, pages 66–73. Springer International Publishing, 2017.

[44] Desmond J. Higham. Modeling and Simulating Chemical Reactions. *SIAM Review*, 50(2):347–368, 2008.

[45] Jane Hillston. *A Compositional Approach to Performance Modelling*. Distinguished Dissertations in Computer Science. Cambridge University Press, 1996.

[46] Charles A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall International Series in Computing Science. Prentice-Hall, 1985.

[47] James Hoey, Irek Ulidowski, and Shoji Yuen. Reversing Imperative Parallel Programs. In Kirstin Peters and Simone Tini, editors, Proceedings Combined 24th International Workshop on *Expressiveness in Concurrency* and 14th Workshop on *Structural Operational Semantics,* Berlin, Germany, 4th September 2017, volume 255 of *Electronic Proceedings in Theoretical Computer Science*, pages 51–66. Open Publishing Association, 2017.

[48] James Hoey, Irek Ulidowski, and Shoji Yuen. Reversing Parallel Programs with Blocks and Procedures. In Jorge A. Pérez and Simone Tini, editors, Proceedings Combined 25th International Workshop on *Expressiveness in Con-*

currency and 15th Workshop on *Structural Operational Semantics,* Beijing, China, September 3, 2018, volume 276 of *Electronic Proceedings in Theoretical Computer Science*, pages 69–86. Open Publishing Association, 2018.

[49] Robert Höllering, Johann Gasteiger, Larissa Steinhauer, Klaus-Peter Schulz, and Achim Herwig. Simulation of Organic Reactions: From the Degradation of Chemicals to Combinatorial Synthesis. *Journal of Chemical Information and Computer Sciences*, 40(2):482–494, 2000.

[50] Michael Hucka, Andrew Finney, Herbert Sauro, Hamid Bolouri, John Doyle, Hiroaki Kitano, et al. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531, 2003.

[51] David A. Huffman. Canonical forms for information-lossless finite-state logical machines. *IRE Transactions on Information Theory*, 5(5):41–59, May 1959.

[52] JGraphT. JGraphT website, 2016. `http://www.jgrapht.org`. Accessed 11-July-2016.

[53] JUnit. JUnit website, 2018. `https://junit.org/`. Accessed 12-May-2018.

[54] William Kemp. *NMR in Chemistry. A Multinuclear Introduction.* Macmillan, 1995.

[55] Agnes Köhler, Jean Krivine, and Jakob Vidmar. A Rule-Based Model of Base Excision Repair. In Pedro Mendes, Joseph O. Dada, and Kieran Smallbone, editors, *Computational Methods in Systems Biology: 12th International Conference, CMSB 2014*, pages 173–195. Springer, 2014.

[56] Masaaki Kotera, Yasuo Tabei, Yoshihiro Yamanishi, Toshiaki Tokimatsu, and Susumu Goto. Supervised de novo reconstruction of metabolic pathways from metabolome-scale compound sets. *Bioinformatics*, 29(13):135–144, 2013.

[57] Jean Krivine. Kappa project: DNA repair, 2017. `https://www.irif.fr/~jkrivine/homepage/Teaching_files/TP-DNA.pdf`. Accessed 11-November-2017.

[58] Stefan Kuhn. Simulation of Base Excision Repair in the Calculus of Covalent Bonding. In Jarko Karri and Irek Ulidowski, editors, *Reversible Computation, 10th International Conference, RC 2018*, volume 11106 of *Lecture Notes in Computer Science*, pages 123–129. Springer, 2018.

[59] Stefan Kuhn and Irek Ulidowski. Towards Modelling of Local Reversibility. In Jean Krivine and Jean-Bernard Stefani, editors, *Reversible Computation, 7th International Conference, RC 2015*, volume 9138 of *Lecture Notes in Computer Science*, pages 279–284. Springer, 2015.

[60] Stefan Kuhn and Irek Ulidowski. A Calculus for Local Reversibility. In Simon Devitt and Ivan Lanese, editors, *Reversible Computation, 8th International Conference, RC 2016*, volume 9720 of *Lecture Notes in Computer Science*, pages 20–35. Springer, 2016.

[61] Stefan Kuhn and Irek Ulidowski. Local reversibility in a Calculus of Covalent Bonding. *Science of Computer Programming*, 151(Supplement C):18–47, 2018. Special issue of the 8th Conference on Reversible Computation (RC2016).

[62] Céline Kuttler and Joachim Niehren. Gene Regulation in the Pi Calculus: Simulating Cooperativity at the Lambda Switch. In Corrado Priami, Anna Ingólfsdóttir, Bud Mishra, and Hanne Riis Nielson, editors, *Transactions on Computational Systems Biology VII*, pages 24–55. Springer, 2006.

[63] Rolf Landauer. Irreversibility and Heat Generation in the Computing Process. *IBM Journal of Research and Development*, 5(3):183–191, July 1961.

[64] Ivan Lanese, Claudio Antares Mezzina, Alan Schmitt, and Jean-Bernard Stefani. Controlling Reversibility in Higher-Order Pi. In Joost-Pieter Katoen and Barbara König, editors, *CONCUR 2011 - Concurrency Theory*, volume 6901 of *Lecture Notes in Computer Science*, pages 297–311. Springer, 2011.

[65] Ivan Lanese, Claudio Antares Mezzina, and Jean-Bernard Stefani. Controlled Reversibility and Compensations. In Robert Glück and Tetsuo Yokoyama, editors, *Reversible Computation, 4th International Workshop, RC 2012*, volume 7581 of *Lecture Notes in Computer Science*, pages 233–240. Springer, 2013.

[66] Ivan Lanese, Claudio Antares Mezzina, and Jean-Bernhard Stefani. Reversing Higher-Order Pi. In *Proceedings of the 21st International Conference on Concurrency Theory CONCUR 2010*, volume 6269 of *Lecture Notes in Computer Science*, pages 478–493. Springer, 2010.

[67] Ivan Lanese, Claudio Antares Mezzina, and Francesco Tiezzi. Causal-Consistent Reversibility. *Bulletin of the EATCS*, 114, 2014.

[68] Paolo Lecca, Corrado Priami, P. Quaglia, B. Rossi, C. Laudanna, and G. Constantin. A Stochastic Process Algebra Approach to Simulation of Autoreactive Lymphocyte Recruitment. *SIMULATION*, 80(6):273–288, 2004.

[69] Yves Lecerf. Machines de Turing réeversibles. *Comptes Rendus Hebdomadaires des Séances de l'Académie des Sciences*, 257:2597–26, 1963.

[70] Jinsoo Lee, Wook-Shin Han, Romans Kasperovics, and Jeong-Hoon Lee. An in-depth comparison of subgraph isomorphism algorithms in graph databases. In *Proceedings of the 39th international conference on Very Large Data Bases*, PVLDB'13, pages 133–144. VLDB Endowment, 2013.

[71] George B. Leeman, Jr. A Formal Approach to Undo Operations in Programming Languages. *ACM Trans. Program. Lang. Syst.*, 8(1):50–87, January 1986.

[72] Errol G. Lewars. *Computational Chemistry*. Springer, 2016.

[73] Armando B. Matos. Linear programs in a simple reversible language. *Theoretical Computer Science*, 290(3):2063 – 2074, 2003.

[74] Pawel Kerntopf Claudio Moraga Krzysztof Podlaski Robert Wille Matthias Soeken, Nabila Abdessaied. COST Action IC1405: Reversible Computation - Extending Horizons of Computing. State of the Art Report Working Group 3: Reversible logic synthesis, 2018. `https://github.com/COST-IC1405/wg3-soar-report/blob/master/README.md`. Accessed 22-Nov-2018.

[75] Maude. The Maude System, 2018. `http://maude.cs.illinois.edu/`. Accessed 25-June-2018.

[76] Mylène Maurin, Morgan Magnin, and Olivier Roux. *Modeling of Genetic Regulatory Network in Stochastic π-Calculus*, In Sanguthevar Rajasekaran, editor, *Bioinformatics and Computational Biology: First International Conference, BICoB 2009*, pages 282–294. Springer, 2009.

[77] Chris McCaig, Rachel Norman, and Carron Shankland. From individuals to populations: A mean field semantics for process algebra. *Theoretical Computer Science*, 412(17):1557–1580, 2011.

[78] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980.

[79] Robin Milner. *Communication and Concurrency (Prentice Hall International Series in Computer Science)*. Prentice Hall PTR, 9 1995.

[80] Robin Milner. *Communicating and Mobile Systems: The π-calculus*. Cambridge University Press, 1999.

[81] Daniel Morrison and Irek Ulidowski. Direction-Reversible Self-Timed Cellular Automata for Delay-Insensitive Circuits. *J. Cellular Automata*, 12(1-2):101–120, 2016.

[82] MohammadReza Mousavi, Iain Phillips, Michel A. Reniers, and Irek Ulidowski. Semantics and expressiveness of ordered SOS. *Information and Computation*, 207(2):85–119, 2009.

[83] Madhu Mutyam and Kamala Krithivasan. P Systems with Membrane Creation: Universality and Efficiency. In *Proceedings of the Third International Conference on Machines, Computations, and Universality*, MCU '01, pages 276–287. Springer-Verlag, 2001.

[84] mxGraph. mxGraph website, 2018. `https://jgraph.github.io/mxgraph/`. Accessed 5-May-2018.

[85] Miloslav Nic, Jiri Jirat, and Bedrich Kosata. IUPAC Compendium of Chemical Terminology (Gold Book), Release 2.3.3b, 2017-03-27. `https://goldbook.iupac.org/`. Accessed 12-December-2017.

[86] Andrei Păun and Gheorghe Păun. The power of communication: P systems with symport/antiport. *New Generation Computing*, 20(3):295–305, 2002.

[87] Gheorghe Păun. Computing with Membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.

[88] Michael Pedersen and Gordon D. Plotkin. A Language for Biochemical Systems: Design and Formal Specification. In Corrado Priami, Rainer Breitling, David Gilbert, Monika Heiner, and Adelinde M. Uhrmacher, editors, *Transactions on Computational Systems Biology XII*, pages 77–145. Springer, 2010.

[89] Anna Philippou and Kyriaki Psara. Reversible Computation in Petri Nets. In Jarkko Kari and Irek Ulidowski, editors, *Reversible Computation*, pages 84–101. Springer International Publishing, 2018.

[90] Iain Phillips and Irek Ulidowski. Reversing algebraic process calculi. *The Journal of Logic and Algebraic Programming*, 73(1-2):70–96, 2007.

[91] Iain Phillips and Irek Ulidowski. Reversibility and Asymmetric Conflict in Event Structures. In Pedro R. D'Argenio and Hernán Melgratti, editors, *CONCUR 2013 - Concurrency Theory*, volume 8052 of *Lecture Notes in Computer Science*, pages 303–318. Springer, 2013.

[92] Iain Phillips, Irek Ulidowski, and Shoji Yuen. Modelling of Bonding with Processes and Events. In Gerhard W. Dueck and D. Michael Miller, editors, *Reversible Computation, 5th International Conference, RC 2013*, volume 7948 of *Lecture Notes in Computer Science*, pages 141–154. Springer, 2013.

[93] Iain Phillips, Irek Ulidowski, and Shoji Yuen. A Reversible Process Calculus and the Modelling of the ERK Signalling Pathway. In Robert Glück and Tetsuo Yokoyama, editors, *Reversible Computation, 4th International Workshop, RC 2012*, volume 7581 of *Lecture Notes in Computer Science*, pages 218–232. Springer, 2013.

[94] Ian Phillips and Irek Ulidowski. Reversing algebraic process calculi. In *Proceedings of 9th International Conference on Foundations of Software Science and Computation Structures, FOSSACS 2006*, volume 3921 of *Lecture Notes in Computer Science*, pages 246–260. Springer, 2006.

[95] Gordon D. Plotkin. A structural approach to operational semantics. *The Journal of Logic and Algebraic Programming*, 60–61:17–139, 2004.

[96] Gordon D. Plotkin. *A Calculus of Chemical Systems*, In Val Tannen, Limsoon Wong, Leonid Libkin, Wenfei Fan, Wang-Chiew Tan, and Michael Fourman, editors, *In Search of Elegance in the Theory and Practice of Computation: Essays Dedicated to Peter Buneman*, pages 445–465. Springer, 2013.

[97] Corrado Priami. Stochastic π-Calculus. *The Computer Journal*, 38(7):578–589, 1995.

[98] Corrado Priami, Aviv Regev, Ehud Shapiro, and William Silverman. Application of a stochastic name-passing calculus to representation and simulation of molecular processes. *Information Processing Letters*, 80(1):25–31, 2001.

[99] ProB. The ProB Animator and Model Checker, 2018. `https://www3.hhu.de/stups/prob/index.php/Main_Page`. Accessed 25-June-2018.

[100] Aviv Regev. Representation and simulation of molecular pathways in the stochastic pi-calculus. In Ralph Gauges, C. van Gend, and U. Kummer, editors, *2nd Workshop on Computation of Biochemical Pathways and Genetic Networks*, pages 109–115. Logos, 2001.

[101] Aviv Regev, Ekaterina M. Panina, William Silverman, Luca Cardelli, and Ehud Shapiro. BioAmbients: an abstraction for biological compartments. *Theoretical Computer Science*, 325(1):141–167, 2004.

[102] Aviv Regev and Ehud Shapiro. Cells as computation. *Nature*, 419(6905):343, 2002.

[103] Aviv Regev and Ehud Shapiro. *The π-calculus as an Abstraction for Biomolecular Systems*, In Gabriel Ciobanu and Grzegorz Rozenberg, editors, *Modelling in Molecular Biology*, pages 219–266. Springer, 2004.

[104] Aviv Regev, William Silverman, and Ehud Shapiro. Representation and simulation of biochemical processes using the π-calculus process algebra. In Russ B. Altman, A. Keith Dunker, and Lawrence Hunker, editors, *Pacific Symposium on Biocomputing 2001*, pages 459–470. World Scientific, 2000.

[105] Aviv Regev, William Silverman, and Ehud Shapiro. Representing biomolecular processes with computer process algebra: π-calculus programs of signal transduction pathways, 2000. `http://www.wisdom.weizmann.ac.il/~biospi/pi_draft.ps`. Accessed 5-May-2016.

[106] Norbert Schormann, Robert Ricciardi, and Debasish Chattopadhyay. Uracil-DNA glycosylases-structural and functional perspectives on an essential family of DNA repair enzymes. *Protein Science*, 23(12):1667–1685, Dec 2014.

[107] Ulrik Pagh Schultz. Towards a General-Purpose, Reversible Language for Controlling Self-reconfigurable Robots. In Robert Glück and Tetsuo Yokoyama, editors, *Reversible Computation*, pages 97–111. Springer Berlin Heidelberg, 2013.

[108] Ulrik Pagh Schultz, Johan Sund Laursen, Lars-Peter Ellekilde, and Holger Bock Axelsen. Towards a Domain-Specific Language for Reversible Assembly Sequences. In *Reversible Computation - 7th International Conference, RC 2015, Grenoble, France, July 16-17, 2015, Proceedings*, pages 111–126, 2015.

[109] Lesley Smart. *Chemical Kinetics and Mechanism*. The Molecular World. The Royal Society of Chemistry, 2002.

[110] Perdita Stevens. The Edinburgh Concurrency Workbench, 2018. `http://homepages.inf.ed.ac.uk/perdita/cwb/`. Accessed 25-June-2018.

[111] David J. Sumpter, Guy B. Blanchard, and David S. Broomhead. Ants and agents: a process algebra approach to modelling ant colony behaviour. *Bulletin of Mathematical Biology*, 63(5):951–980, Sep 2001.

[112] David J. Sumpter and David S. Broomhead. Shape and dynamics of thermoregulating honey bee clusters. *Journal of Theoretical Biology*, 204(1):1–14, May 2000.

[113] David J. Sumpter and David S. Broomhead. Relating individual behaviour to population dynamics. *Proceedings of the Royal Society B: Biological Sciences*, 268(1470):925–932, May 2001.

[114] David J. Sumpter and Stephen J. Martin. The dynamics of virus epidemics in Varroa-infested honey bee colonies. *Journal of Animal Ecology*, 73(1):51–63, 2004.

[115] Irek Ulidowski. Equivalences on Observable Processes. In *Proceedings of the 7th Annual IEEE Symposium on Logic in Computer Science*, pages 148–159. IEEE, 1992.

[116] Irek Ulidowski and Iain Phillips. Ordered SOS Process Languages for Branching and Eager Bisimulations. *Information and Computation*, 178(1):180–213, 2002.

[117] Irek Ulidowski, Iain Phillips, and Shoji Yuen. Concurrency and Reversibility. In Shigeru Yamashita and Shin-ichi Minato, editors, *Reversible Computation, 6th International Conference, RC 2014*, volume 8507 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2014.

[118] UndoDB. UndoDB website, 2018. `https://undo.io/`. Accessed 22-Nov-2018.

[119] Bill Venners. The Good, the Bad, and the DOM, 2018. `https://www.artima.com/intv/dom.html`. Accessed 1-July-2018.

[120] Irina Virbitskaite and Nataliya Gribovskaya. Preserving Behavior in Transition Systems from Event Structure Models. In *Proceedings of the 27th International Workshop on Concurrency, Specification and Programming, Berlin, Germany, September 24-26, 2018.*, 2018.

[121] Glynn Winskel. *Event structures*, In Wilfried Brauer, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Petri Nets: Applications and Relationships to Other Models of Concurrency: Advances in Petri Nets 1986, Part II Proceedings of an Advanced Course Bad Honnef, 8.–19. September 1986*, volume 8052 of *Lecture Notes in Computer Science*, pages 325–392. Springer, 1987.

[122] XOM. XOM website, 2018. `https://xom.nu/`. Accessed 5-May-2018.

[123] Tetsuo Yokoyama and Robert Glück. A Reversible Programming Language and Its Invertible Self-interpreter. In *Proceedings of the 2007 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-based Program Manipulation*, PEPM '07, pages 144–153. ACM, 2007.