Algorithms for the Simulation of Surface Processes

John Segers
Algorithms for the Simulation of Surface Processes

John Segers
Algorithms for the Simulation of Surface Processes

PROEFSCHRIFT

TER VERKRIJGING VAN DE GRAAD VAN DOCTOR AAN DE TECHNISCHE UNIVERSITEIT EINDHOVEN, OP GEZAG VAN DE RECTOR MAGNIFICUS, PROF.DR. M. REM, VOOR EEN COMMISSIE AANGEWZEN DOOR HET COLLEGE VAN PROMOTIES IN HET OPENBAAR TE VERDEDIGEN OP MAANDAG 12 APRIL 1999 OM 16.00 UUR

DOOR

JOHN PETER LEON SEGERS

GEBOREN TE HELMOND
Dit proefschrift is goedgekeurd door de promotoren:

prof.dr. P.A.J. Hilbers

en

prof.dr. R.A. van Santen

Copromotor:

dr. J.J. Lukkien

The work described in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics).
## Contents

1 Introduction .................................................. 1
   1.1 Computer simulation in physics ......................... 1
   1.2 Surface processes ..................................... 2
   1.3 Overview and contributions ........................... 4

2 Stochastic Modeling of Surface Reactions .................. 7
   2.1 Introduction .......................................... 7
   2.2 Surface and adsorbed particles ........................ 9
   2.3 Reactions ............................................. 10
      2.3.1 Transformations .................................. 10
      2.3.2 Reaction rates ................................... 13
   2.4 Some examples ....................................... 16
      2.4.1 The ZGB model .................................. 16
      2.4.2 Diffusion with lateral interaction ................ 19
      2.4.3 A more advanced CO oxidation model .............. 21
   2.5 Reaction waiting times ................................ 24
   2.6 Master equation ...................................... 26
   2.7 Discussion ........................................... 27

3 A Taxonomy of Simulation Algorithms ....................... 29
   3.1 Introduction .......................................... 29
      3.1.1 Correctness of simulation algorithms .............. 30
      3.1.2 Chapter overview ................................ 31
   3.2 Basic simulation algorithm ............................ 32
   3.3 Generalization techniques ............................. 34
      3.3.1 Oversampling ...................................... 34
      3.3.2 Approximate representation of the set of enabled events 37
      3.3.3 Selecting reactions in multiple stages ............ 41
   3.4 Complexity analysis .................................. 43
      3.4.1 Simplifying assumptions .......................... 44
### CONTENTS

3.4.2 Complexity of basic operations ........................................... 44  
3.4.3 Complexity of the algorithms ............................................ 51  
3.4.4 Discussion ........................................................................... 59  
3.5 Combining techniques ............................................................. 60  
   3.5.1 Combining WRsel-h and WRsel-a ..................................... 60  
   3.5.2 Combining WRsel-ha and WRsel-o ................................... 65  
   3.5.3 The merits of oversampling ............................................. 66  
3.6 Overview and literature review ................................................ 75  
3.7 A second class of algorithms .................................................... 79  
3.8 Approximate algorithms ........................................................... 88  
3.9 Hybrid approaches .................................................................. 90  

4 Simulation of Systems with Time-Dependent Reaction Rates ........ 93  
   4.1 Applications .......................................................................... 93  
   4.2 Extension of the model ........................................................... 95  
   4.3 The time-dependent WRsel algorithm ................................... 97  
   4.4 Generalizations of tdWRsel .................................................... 101  
      4.4.1 Oversampling ................................................................. 101  
      4.4.2 Approximate representation of the set of enabled reactions  104  
      4.4.3 Selecting reactions in multiple stages ............................... 107  
   4.5 Combining techniques ........................................................... 109  
   4.6 A second class of algorithms .................................................. 113  
   4.7 Literature review and conclusions .......................................... 118  

5 Design and Performance Evaluation of CARLOS ......................... 123  
   5.1 Requirements and design decisions ....................................... 123  
   5.2 Features of the current implementation .................................. 124  
   5.3 Performance evaluation .......................................................... 128  
      5.3.1 The ZGB model ............................................................... 128  
      5.3.2 Diffusion with lateral interaction .................................... 131  
      5.3.3 A more advanced model for CO oxidation ....................... 134  
   5.4 Conclusions ........................................................................... 136  

6 Opportunities for Parallelism ....................................................... 139  
   6.1 Introduction ............................................................................ 139  
   6.2 Parallel simulation of surface processes ................................ 142  
      6.2.1 Replication ....................................................................... 143  
      6.2.2 Control parallelism ............................................................ 143  
      6.2.3 Data parallelism ............................................................... 144  
   6.3 Approximate parallel simulation ............................................. 149  
   6.4 Parallel simulation of the ZGB model .................................... 150
1

Introduction

In the fifties Metropolis et al. (1953) published an article on stochastic simulation of physical systems. This article marks the beginning of a period in which stochastic models and computers have been applied successfully in physics. The technique used to analyze stochastic models by means of computers has become known as Monte Carlo simulation because of the extensive use of (pseudo-)random numbers. In this monograph we investigate algorithms for simulating surface processes, using a stochastic model.

1.1 Computer simulation in physics

The first computers were built in the early 1940's. Ever since then computers have been used for the simulation of physical systems. In fact, one of the first applications of computer technology, during the second world war, was calculating trajectories of ballistic missiles. Soon after the second world war attention shifted to simulation in an entirely different area of physics, namely the simulation of systems composed of individual, microscopic particles such as molecules, atoms and subatomic particles. In the early days, the systems considered comprised dozens to hundreds of particles. Considering the fact that a milliliter of water contains some $3 \cdot 10^{22}$ individual molecules, the question rises whether such simulation are useful.

The answer to this question is that, yes, simulations of very small physical systems are useful. They are useful for a number of reasons. Computer simulations allow investigation of phenomena for which there is no macroscopic description. One of the most difficult tasks in the physical sciences is building macroscopic models of systems consistent with the
processes that occur at the microscopic level, the level of individual particles. Simulation offers a conceptually simple way to link the behavior of individual particles to the collective behavior of many particles. At present, computer technology allows us to simulate the behavior of tens of thousands to millions of particles. Although this number of particles is still many orders of magnitude smaller than the aforementioned $3 \cdot 10^{22}$ molecules in a milliliter of water, it is often sufficiently large to see macroscopic effects occurring. In other cases larger scale simulations are needed. This requires faster computers, more efficient simulation methods, or both.

A second reason for performing computer simulations of physical systems is to validate microscopic models. The laws governing the interaction of small numbers of particles (pairs, triples, quadruples, ... ) are largely known. In principle, the Schrödinger equation can be used to describe physical systems at the microscopic level. But the equations describing the behavior of more than a few particles are as yet too complex to be solved. Laboratory experiments are performed and theories are proposed to describe the behavior of systems without going down to the level of detail of the Schrödinger equation. These theories can be validated by performing computer simulations taking the models as input and comparing the simulated behavior to the behavior observed in laboratory experiments.

Computer simulations can also offer an alternative to laboratory experiments. If an adequate model for a system is available, computer simulation using the model may be quicker and less expensive than laboratory experiments. Moreover, some things that are hard to do in the laboratory, are easily achieved in a computer simulation. For instance, in a computer simulation, observations can be made at the smallest time scales that occur in the system being simulated. In laboratory experiments this is not always possible. Besides allowing observations at length or time scales that cannot be observed in the laboratory, computer simulations allow experiments that are not possible in laboratory experiments. For instance, in a computer simulation it is possible to change the composition of a catalytic surface during a simulation in order to determine which composition is optimal.

One thing that is important to remember is that simulation is never a goal in itself. Computer simulations can serve as a means to increase our understanding of the world around us, but only in conjunction with theory and experiments.

### 1.2 Surface processes

Over 90% of the chemical manufacturing processes in the world use catalysts in some form. According to the traditional definition, a catalyst is a substance that influences a chemical reaction — more specifically, that changes the rate at which equilibrium is reached — without undergoing chemical change itself. In reality, on a catalytic surface may reactions occur in a catalytic cycle. One can only say that on average a catalyst does not change during
1.2 Surface processes

the reaction. The changes that occur in the catalytic cycle form an interesting subject of research. We can distinguish two kinds of catalytic processes, viz. homogeneous and heterogeneous. In homogeneous catalysis, catalyst and reactants are in the same phase, all are fluids, for instance. A disadvantage of homogeneous catalysis is the relatively difficult separation of the catalyst from the end products. If catalyst and reactants are in different phases, we speak of heterogeneous catalysis. Most often, the catalyst is a solid, while the other substances involved are in the liquid or gas phase. Heterogeneous catalysts are of great importance to modern society. They are used in the production of fuels, building materials, synthetic fabrics used in clothing, medicine, and food. Even though heterogeneous catalysts are used at such large scale, there is yet much to be learned about the processes that take place at the interface between the solid catalyst and the reactants in the gas or liquid phase. A better understanding of these surface processes will hopefully lead to a more structured approach to designing new catalysts, improving efficiency of chemical manufacturing plants and reducing the amount of unwanted by-products.

The work presented in this monograph deals with simulation techniques for surface processes at the microscopic level. The goals are to provide efficient methods for specifying the behavior of individual molecules and atoms at surfaces, and to simulate systems consisting of thousands to millions of particles. Efficient simulation methods are needed for several reasons. The first is to be able to simulate systems large enough to exhibit macroscopic behavior. The second reason is that more efficient simulation methods allow more complex, and thus more realistic models to be simulated. A third reason, of an entirely different nature, may be the cost of computing resources. Computer time on supercomputers is much more expensive than on workstations. It is therefore desirable to be able to perform simulations on workstations, without having to resort to supercomputers.

The work described here is part of a larger project. The participants in the project are the Laboratory of Inorganic Chemistry and Catalysis and the Parallel Systems group of Eindhoven University of Technology. The Inorganic Chemistry and Catalysis group participates in the Netherlands Institute for Research in Catalysis (NIOK). One of the research goals is to be able to design new catalytic reactors. In order to make this possible many aspects of catalytic systems have to be understood, ranging from the molecular basis of catalysis to reactor technology. As mentioned above, this monograph deals with aspects of catalysis at the molecular level.

The focus of this thesis is to gain insight in simulation techniques in order to assist the researchers in the Chemistry Department to obtain new physical insights. Several researchers in the Inorganic Chemistry and Catalysis group use the software package described in Chapter 5 as one of their tools.
1.3 Overview and contributions

This monograph contains elements of physics and chemistry, computer science, and probability theory. As the author's background is in computer science, most chapters deal with issues from this field. The physics and probability theory included do not go beyond what is necessary to achieve the main goal: to provide a framework for simulating the kinetics of surface processes.

In order to provide such a framework, a mathematical model of surface processes is needed. The model we use is described in detail in Chapter 2. We devote much attention to the kinetics of the elementary reaction steps as the literature shows that this is often a source of confusion in the simulation of surface processes.

The model we employ is stochastic in nature. The kinetics of processes are often defined by probability distributions for the amount of time that elapses before they take place. In Chapter 2 we assume that these probability distributions are stationary, i.e., that they do not depend on the current time. In physical terms, we assume that the rates of processes are constant over time.

Chapter 3 contains a taxonomy of algorithms for the simulation of such models with constant rates. The algorithms presented are compared by two characteristics: a measure for their time complexity and the amount of memory they use. We explain how existing algorithms fit into our taxonomy and give bibliographic references.

In Chapter 4 we let go of the constraint that rate constants be time-independent. We allow them to vary over time, thus extending the class of systems we can model to include temperature programmed desorption and reaction, as well as electrochemical systems with electrode potentials that vary over time. It turns out that all algorithms of Chapter 3 generalize to the case of time-dependent rate constants. We investigate how their performance is affected by this generalization. Again we provide a discussion of the algorithms found in the literature. It turns out that some published algorithms are erroneous.

Chapter 5 contains a description of the simulation program CARLOS, developed in the Parallel Systems group at Eindhoven University. We show the results of some simulations to see whether our performance model is accurate.

Chapter 6 extends a conference article by the author discussing parallel algorithms for the simulation of surface processes.

Chapter 7 contains a review of the results and their applicability in a wider scope as well as some pointers for future research.

The correctness arguments for the simulation algorithms we present, rely on some properties from probability theory. These properties and their proofs are presented in Appendix A, along with a brief introduction to the probability-theoretic notions used in those proofs.
1.3 Overview and contributions

Appendix B contains an overview of the notation we use throughout this monograph. It also contains an index of algorithms and explains the conventions used for naming algorithms.

The main research contributions presented in this thesis can be formulated as follows.

- A framework is presented for the simulation of the kinetics of surface processes, with correctness criteria for simulation algorithms.

- A taxonomy of simulation algorithms is presented, accompanied by a performance model for the algorithms.

- New algorithms with better performance than the ones found in the literature are introduced, in particular for the simulation of systems with time-dependent rate constants.

- A simulation tool for the simulation of surface processes has been developed.
2

Stochastic Modeling of Surface Reactions

In this chapter we introduce a formal description of surface reaction systems. We aim to model the behavior of individual particles, atoms or molecules, at a catalytic surface. This behavior includes processes such as adsorption of particles onto a surface, desorption from the surface, surface diffusion and chemical reactions.

After a short introduction we address the three ingredients of our description. We start with the catalytic surface and adsorbed particles. This defines the state of systems. Next come the reactions that may take place at the surface. The modeling of the kinetics is treated in a separate section. After having presented our model, we investigate some examples and discuss alternative ways of describing the kinetics.

The material in this chapter is not new. The reasons for paying this much attention to the model are twofold. The first reason is that it makes this monograph more self-contained, thus making it easier to read for readers without a background in physics or chemistry. The second reason is that we aim to set up a uniform framework that is detailed enough to simulation algorithms for surface reaction systems. A detailed description of the model is needed to establish correctness criterion for the simulation algorithms.

2.1 Introduction

Surface reactions can be modeled in a variety of ways. Different modeling techniques highlight different aspects of the systems modeled. For instance, one might be interested
in the flow of reactants in a chemical reactor. A macroscopic description of the conversion that takes place in the reactor could be sufficient in this case. If, on the other hand, one is interested in the interaction of reactants with a catalyst, reactant flow may be less important, while phenomena at the surface have to be described in more detail. In other words, the modeling is determined by the questions one seeks answers to.

In order to place our work, we briefly discuss the two main approaches to the modeling and analysis of surface processes. This discussion is by no means intended to serve as an overview of modeling techniques for surface processes. A good overview of such techniques is given by Lombardo and Bell (1991).

The first approach to modeling surface processes is known as the analytical approach. In this approach, the kinetics of surface processes are modeled by describing the time dependence of, for instance, fractional surface coverages. The resulting set of equations is often too complicated to be solved directly. Various approximations are applied to obtain simpler equations. An aspect of surface processes that is hard to capture in this way of modeling is the fact that adsorbed particles interact with particles in their vicinity. If such interactions are important, then it becomes necessary to extend the model with probabilities of clusters of particles occurring on the surface. This leads to a larger number of equations, and the equations become more complicated. Nevertheless, useful approximations have been developed and the analytical approach has led to many insights into the nature of surface processes.

The second approach is the so-called numerical approach. In this approach, the surface and the individual particles adsorbed at the surface are part of model descriptions, instead of having only fractional coverages. The inclusion of this geometric information allows a more detailed modeling of the lateral interactions. Models in which space is discrete are called lattice-gas models. This is the modeling used in this thesis. The reason is that lattice-gas models are particularly suited for investigating the relationship between microscopic and mesoscopic behavior.

The numerical approach has its disadvantages too. A well-known problem is the fact that the presence of processes with very different time scales in one model complicates the analysis. There have been attempts to solve this problem by means of hybrid modeling, omitting the detail of lattice-gas modeling for the fast processes (Silverberg and Ben-Shaul, 1987, 1988, 1989; Tammaro et al., 1995).

In summary, in this chapter we describe a model based on the numerical approach because this allows a more detailed description of the behavior of individual molecules or atoms than the analytical approach.
2.2 Surface and adsorbed particles

In surface catalysis, reactions take place at the interface between two phases. Usually the catalyst is a solid and the reactants are in the gas phase. There are several types of catalysts. Well-known types are transition metals such as platinum and nickel, metal oxides, and zeolites. In the large-scale production of chemicals, the catalyst is used efficiently if it has a high surface area to volume ratio. To achieve this, small crystals are embedded in a carrier. In contrast, in research into the phenomena that take place at the gas-surface interface, it is more common to use single-crystal surfaces. These allow more detailed observations and lead to simpler models.

One of the advantages of having a so-called single-crystal surface is the fact that such a surface is uniform. This means that it can be described by a spatial repetition of unit cells. The interaction between surface atoms of the catalyst and adsorbed reactants is the same all over the surface. Therefore all unit cells can be treated in the same way.

Particles that adsorb onto a surface, only do so at specific places. These are determined by the energetic interaction between the adsorbing particle and the surface atoms. Once adsorbed, a particle is not completely fixed, but it vibrates around its preferred position as a result of thermal vibration. The probability of finding an adsorbed particle (adparticle) away from a preferred position is small enough to allow the model of the surface to be discrete in most cases. In some cases this assumption cannot be made, for example if adsorbed particles interact too strongly (Asada, 1986; Zhdanov, 1991).

In the sequel we assume that a discrete-space model is sufficient. Then the unit cell of the surface consists of a number of so-called adsorption sites. Each site may contain a particle.

The relative positions of adsorption sites of a unit cell and the unit cell's geometry are important properties. They determine which particles may adsorb where and which adparticles are close enough to interact. Instead of including this information in the model of the surface, we include it in the specification of the reactions (these include adsorption and desorption). These will not only specify the number of particles involved, as a reaction equation does. The reaction specifications will also determine the constraints on relative positions of particles.

At first sight, this may seem a strange decision: why not separate the modeling of the catalyst surface (including the geometry and neighbor relations between sites) from the reactions as much as possible? The reason is that now all geometry information is relegated to the specification of the reactions. This leads to a more flexible modeling, as we will see later in this chapter (in Section 2.4.3 in particular).

As no geometry information is incorporated in the model of the surface, it suffices to have \( \mathcal{L} = [0..L_0) \times [0..L_1) \times [0..L_2) \) as a model. This represents a lattice consisting of \( L_0 \times L_1 \) unit cells with \( L_2 \) sites per cell.
In order to eliminate boundary effects, we assume a periodic boundary for the first two dimensions. This means that the leftmost cell in a row or column and the rightmost one are neighbors. In the site dimension there is no periodic boundary. Addition involving unit cell coordinates is modulo $L_0$ in the first dimension, and modulo $L_1$ in the second. Adding site numbers generally has no meaning as the sites in a unit cell may be numbered in any way.

Every adsorption site takes a value from a finite domain $\mathcal{D}$. This set contains the particle types that occur in the system and, possibly, a value to designate vacant sites. In other words, 'vacant' is considered a particle type. A surface with a certain configuration of adsorbed particles can now be described by mapping from $\mathcal{L}$ to $\mathcal{D}$. The set of all such functions forms the state space. By convention, it is denoted by $\Sigma$. Another convention we introduce is the use of $\sigma$, possibly with subscripts, for states.

Note that the composition of the gas phase is not included in the state space. The underlying assumption is that the gas phase composition is homogeneous over space. Systems for which this assumption is not valid require cannot be described adequately in our model.

### 2.3 Reactions

So far we have only concerned ourselves with the modeling of system states. In this section we describe how reactions are modeled. In the modeling of reactions we do not distinguish between different kinds of reactions in the sense that all processes – adsorption, desorption, surface diffusion, and chemical reactions – are treated the same way. Two aspects have to be addressed in a model for reactions on a surface. First, we have to describe which reactions are possible in a configuration of the surface, and what exactly is the result of occurrence of a reaction. Second, we have to model the dynamics. In other words, for a reaction that is possible we have to describe when it occurs. In the following subsections we further detail these two aspects.

#### 2.3.1 Transformations

In our model the occurrence of a reaction is an instantaneous event. The only result of the occurrence of a reaction is a change in the configuration of the surface. Conversely, the surface configuration changes only by the occurrence of reactions.

The assumption that reactions can be modeled by events without duration is not a restriction. It is still possible to model reactions that are not instantaneous, by splitting them into two parts. The first part designates the start of the reaction, the second its completion.

A reaction is described by a location on the surface and a transformation type. This transformation type specifies the conditions on the surface configuration that must hold for the
2.3 Reactions

transformation to be possible at its location, as well as the state change induced by the occurrence of the reaction. For example, consider the adsorption of a particle $A$ onto a certain site. For adsorption to be possible at that site, the site must be vacant. This requirement is the enabling condition of the reaction. The result of the adsorption taking place is that the surface configuration changes at the site involved: the ‘vacant’ label is replaced by an $A$. All other sites remain unchanged.

For transformation types involving more than one site, the relative positions of the sites have to be specified. This can be accomplished by specifying the relative positions of the unit cells and the site numbers. At this point the geometry of the lattice becomes important. We use a small example to explain how a surface model and reaction specification describe a certain geometry.

Consider a surface of which the atoms are closely packed. Assume that we have two particle types, $B$ and $e$. The absence of adparticles is denoted by the particle type $e$ and $B$ particles adsorb on the threefold hollow sites as shown in Figure 2.1. This figure also shows how the surface structure can be seen as a repetition of unit cells. Every unit cell contains two adsorption sites, numbered 0 and 1. We introduce a transformation type to model the associative desorption of two adjacent $B$’s. Since every site has three neighboring sites, there are three possibilities; see Figure 2.1. Two $B$’s in one unit cell may desorb associatively, as may two $B$’s in two adjacent unit cells. For the latter case there are two possible orientations for the two unit cells involved. The three possible orientations of the two sites involved in the desorption can be given by the following three sets:

$$\{(0, 0/0), (0, 0/1)\}, \{(0, 0/1), (1, 0/0)\}, \{(0, 0/1), (0, 1/0)\}.$$  

In each triple the first two coordinates refer to a unit cell, and the third refers to a site within that cell. The unit cell positions referred to in one set are interpreted as relative positions. The first of the three sets, for example, represents all patterns of the form $\{(x, y/0), (x, y/1)\}$, for any $x$ and $y$. The three patterns are also said to specify the neighbor relation of the surface sites with respect to $B_2$ desorption. The ‘$/$’ that separates the site number from the unit cell coordinates emphasizes the different role of the third dimension.

In order to specify a transformation type, we add two particle types to each site specification. The first particle type specifies the type of particle that must be present for the transformation to be possible. The second particle type specifies the state of the site after the transformation has occurred. For associative $B$ desorption we get the following three transformation types:

$$\{((0, 0/0), B, e),((0, 0/1), B, e)\},$$

$$\{((0, 0/1), B, e),((1, 0/0), B, e)\},$$

and

$$\{((0, 0/1), B, e),((0, 1/0), B, e)\}.$$
The above example teaches us how to define a transformation type in general. A transformation type \( \mathcal{T} \) is a subset of \( \mathcal{L} \times \mathcal{D} \times \mathcal{D} \). Not every subset of \( \mathcal{L} \times \mathcal{D} \times \mathcal{D} \) is a valid transformation type; the following conditions must be satisfied:

1. \( (\exists l, p, q : (l, p, q) \in \mathcal{T} : p \neq q) \),

2. \( (\forall l_0, p_0, q_0, l_1, p_1, q_1 : (l_0, p_0, q_0) \in \mathcal{T} \land (l_1, p_1, q_1) \in \mathcal{T} : l_0 = l_1 \Rightarrow (l_0, p_0, q_0) = (l_1, p_1, q_1)) \).

The first condition expresses that we exclude the pathological case of transformations that do not change the surface configuration. The second condition states that no site occurs in more than one triple in \( \mathcal{T} \).

For a transformation type \( \mathcal{T} \), the set \( \{ l, p, q : (l, p, q) \in \mathcal{T} : l \} \) gives the relative positions of the sites involved. We call this set the pattern of the transformation type and denote it by \( p(\mathcal{T}) \). Furthermore we define the source configuration \( sc(\mathcal{T}) \) as the set

\[ \{ l, p, q : (l, p, q) \in \mathcal{T} : (l, p) \} \]
and the target configuration \( \text{tc}(\cal T) \) as

\[
\{l, p, q : (l, p, q) \in \cal T : (l, q)\}
\]

A transformation is a pair \((\cal T, l)\), where \(\cal T\) is a transformation type and \(l\) designates a unit cell. In order to simplify the definition of what it means for a transformation to be enabled, we use the coordinates of site 0 of the unit cell to identify the unit cell – i.e., \(l\) has the form \((x, y/0)\).

With these definitions we can define formally what it means for a transformation to be enabled. A transformation \((\cal T, l)\) is enabled in state \(\sigma \in \Sigma\) if and only if

\[
(\forall l' \text{, } p : (l', p) \in \text{sc}(\cal T) : \sigma(l + l') = p).
\]

We abbreviate this formula to \(\cal E(\cal T, l, \sigma)\). The state change that reflects the occurrence of a transformation \((\cal T, l)\) can now be defined formally as well. If \((\cal T, l)\) occurs in state \(\sigma\), the resulting state is \(\cal C(\cal T, l, \sigma)\), defined as

\[
\cal C(\cal T, l, \sigma)(l') = \begin{cases} 
q & \text{for } (l' - l, q) \in \text{tc}(\cal T), \\
\sigma(l') & \text{for } l' - l \notin \text{p}(\cal T).
\end{cases}
\]

Taking a more abstract view of the model we have, we see that we are dealing with a discrete event system. The set of states is \(\Sigma\). In state \(\sigma\), the collection of enabled events is \(\{\cal T, l : \cal E(\cal T, l, \sigma) : (\cal T, l)\}\). The transition operator is \(\cal C\): given an event \(e\) and a state \(\sigma\) it gives the state that results if \(e\) occurs in \(\sigma\). Note that this system contains disabling: an enabled event may become disabled by the occurrence of another event. From now on we use ‘event’ and ‘transformation’ interchangeably.

It will turn out to be convenient to have some additional notation. We denote the set of events enabled in state \(\sigma\) by \(E(\sigma)\).

### 2.3.2 Reaction rates

Having described what the effects of reactions are by the introduction of transformations, we now have to model the kinetics, i.e., the time that elapses between the moment that a transformation becomes enabled and the moment that it occurs – if it occurs at all. This amount of time is known to vary for different reactions of the same type. It depends on the energy of the particles involved, which is not part of our model description. We assume that the effect of the energy of particles on reaction times can be incorporated by describing the speed of reactions in a stochastic manner (Fichthorn and Weinberg, 1991; van Kampen, 1992).

The stochastic model for the kinetics is based on the following fundamental assumption.
If a reaction with rate constant $k$ is enabled in a state $\sigma$, then the probability of this reaction occurring in an infinitesimal time interval of size $dt$ is equal to $k \cdot dt$. The probability that more than one reaction occurs in an interval of length $dt$ is $o(dt)$.

Every reaction has a rate constant $k$ associated with it. The above assumption says that in any state and at any time, the probability of occurrence of an enabled reaction in a vanishingly small interval is proportional to its rate, and that the probability of two reactions occurring simultaneously is negligible. We give a more formal interpretation of the kinetics assumption in Section 2.5.

The formulation of the reaction kinetics is stochastic in nature. The state of a system at a certain time $t$ is now a stochastic variable. In other words, we are dealing with a stochastic process. Since the probabilities of state changes depend on the current state, the stochastic process is a so-called birth process (Feller, 1966).

We can now define a reaction type as a pair consisting of a transformation type and a rate constant. For the time being we assume rate constants to be constant indeed. Later, in Chapter 4, we investigate systems in which the rate constants vary over time, e.g. because the temperature increases with time.

A common way to specify the rate constant $k$ of a reaction is by means of an Arrhenius formula:

$$ k = n \exp(-E_{act}/k_B T). $$

The pre-exponential factor $n$ is assumed to be independent of the temperature. It may be interpreted as an attempt frequency. The exponential factor may be interpreted as a success probability. It depends on the the temperature $T$ and the activation energy $E_{act}$; $k_B$ is the Boltzmann constant.

For modeling purposes it turns out to be convenient to have reactions that occur immediately upon becoming enabled. There are two reasons why this can be convenient. The first reason is that complex reactions, involving large patterns, may be split up into a sequence of simpler reactions. The first of these simpler reactions determines the rate in the way we saw above. The subsequent steps follow immediately; the steps cannot be interrupted by any normal reactions.

The second reason for having immediate reactions, is that they provide a way to incorporate reactions that are very fast compared to the reactions of interest, without having to specify how fast they are. An example of this occurs in the ZGB model for CO oxidation, which can be found in the next section of this chapter.

In models that have immediate reactions, it is often the case that occurrence of a reaction enables multiple immediate reactions. Sometimes it is desirable to be able to specify that
2.3 Reactions

some immediate reactions take precedence over others. This is possible by assigning priorities to immediate reaction types. If, in a certain state, multiple immediate reactions are enabled, one with highest priority will occur first. If multiple immediate reactions with the same priority are enabled, while none with higher priority are enabled, then any of these may be the next reaction to take place. The probability is the same for each one.

The kinetics of systems are still described by the fundamental kinetics assumption for those states in which no immediate reactions are enabled. In states in which immediate reactions are enabled, only immediate reactions occur until a state is reached in which only normal reactions are enabled, a so-called stable state.\(^1\) The amount of time systems spend in non-stable, or transient states is by definition equal to zero.

Although immediate reactions are convenient, they do not add to the expressiveness of our modeling. For every model with immediate reactions there is an equivalent model without immediate reactions. Sequences of reactions consisting of normal reaction followed by a number of immediate reactions can be translated into one, possibly very complex, normal reaction. The rate constant of this new reaction is obtained as the product of the rate constant of the normal reaction and the probabilities of the immediate reactions in the sequence. This way of eliminating immediate reactions works only if no infinite sequences of immediate reactions are possible. We restrict the use of immediate reactions to this case.

The notion of a reaction is the last concept we need for modeling surface reaction systems. Our modeling of systems with rate constants that do not vary over time is captured by the following definition.

**Definition 2.1 (System)**

A surface reaction system is a triple \((\mathcal{L}, \mathcal{D}, \mathcal{RT})\).

- \(\mathcal{L} = [0..L_0] \times [0..L_1] \times [0..L_2]\) describes a regular catalyst surface, consisting \(L_0 \times L_1\) unit cells, with \(L_2\) adsorption sites in each unit cell. We apply a periodic boundary condition to the grid of unit cells. Adsorption site \(i\) of unit cell \((x, y)\) is referred to as site \((x \times y/i)\). If unit cells have only one adsorption site, we also write \((x, y)\).

- Every site takes a value in \(\mathcal{D}\), the domain of particle types. \(\mathcal{D}\) is a finite set and generally contains an element used to indicate that a site is vacant. A state or configuration of the system is an element of \(\Sigma = \mathcal{L} \rightarrow \mathcal{D}\), the set of functions from \(\mathcal{L}\) to \(\mathcal{D}\).

- \(\mathcal{RT}\) is a finite set of reaction types. Its elements describe the possible reactions. The reaction types are divided into two classes, the normal and the immediate reaction types. An element \(RT\) of \(\mathcal{RT}\) consists of a transformation \(T\) and a number, \(k\), indicating its rate constant. The transformation \(T\) is a subset of \(\mathcal{L} \times \mathcal{D} \times \mathcal{D}\) – it gives the

\(^1\)Note that this definition of stability has no relation to the physical notion of stability.
conditions under which reactions of type $RT$ may take place and the state transformation associated with their occurrence. If $k$ is positive, then $RT$ is a normal reaction type, with $k$ specifying its rate constant. If $k$ is negative, then $RT$ is an immediate reaction type. In this case $k$ is required to be integer valued; its value indicates the priority of the type. A higher priority, i.e., a priority closer to zero, means that the type has higher precedence.

The collection of reactions enabled in state $\sigma$ is denoted by $E(\sigma)$. The rate constant of a reaction type $RT$ is denoted by $k(RT)$. For a set of enabled reactions $S$, the sum of the rate constants of the elements of $S$ is denoted by $k(S)$. We write $k(e)$ instead of $k([e])$. The sum of the rate constants of all reaction types is denoted by $k(RT)$.

The fact that enabled reactions can be specified by means of reaction types and unit cells at which these are enabled is made possible by the homogeneity of the surface. Because the lattice is invariant under translation over unit cells, transformation types and unit cells suffice to describe the set of enabled transformations. Furthermore, the rate constant of a reaction does not depend on the unit cell at which it is enabled. This allows the enabled reactions to be specified by means of reaction types and enabling locations.

2.4 Some examples

In this section we consider some examples of simple surface reaction systems. The examples should show the reader that the model is expressive enough to capture a large class of systems. The examples presented here are used later in testing and benchmarking simulation algorithms.

2.4.1 The ZGB model

Let us start with a simple example, based on the work by Ziff et al. (1986). We consider a system with species $A$ and $B$. Particles of type $A$ may adsorb onto the hollow sites of a square surface. Particles of type $B$ may do so as well, but in the gas phase $B$ is diatomic. The adsorption of $A$ has rate constant $k_A$. More specifically, if a site is vacant at time $t$, the probability that it contains an $A$ at time $t + dt$ equals $k_A dt$. The dissociative adsorption of $B_2$ has rate constant $k_{B_2}$. $B_2$ adsorbs only at adjacent vacant sites.

Adsorbed $A$ and $B$ particles form $AB$ and desorb, provided that they are located on adjacent adsorption sites. This reaction is assumed to be so fast that it occurs immediately upon becoming enabled. In other words, the system is adsorption limited.
2.4 Some examples

This surface reaction system can be captured by a model \((\mathcal{L}, \mathcal{D}, \mathcal{RT})\) with \(\mathcal{L} = [0..L_0) \times [0..L_1) \times [0..L_2)\) and \(\mathcal{D} = \{e, A, B\}\). The transformation type for \(A\) adsorption contains only one element:

\[
\mathcal{T}_A = \{((0, 0), e, A)\},
\]

specifying that any vacant site may become an \(A\)-covered site. The corresponding reaction type is \(RT_A = (\mathcal{T}_A, k_A)\). For \(B\) there are two reaction types because two orientations are possible when the \(B_2\) molecule dissociates. The reaction types are \(RT^0_{B_2} = (\mathcal{T}^0_{B_2}, k_{B_2})\) and \(RT^1_{B_2} = (\mathcal{T}^1_{B_2}, k_{B_2})\), where

\[
\mathcal{T}^0_{B_2} = \{((0, 0), e, B), ((1, 0), e, B)\}, \quad \text{and} \\
\mathcal{T}^1_{B_2} = \{((0, 0), e, B), ((0, 1), e, B)\}.
\]

Finally, the infinitely fast \(AB\) formation step is modeled by four reaction types, because four orientations are possible. The transformations types are

\[
\mathcal{T}^0_{AB} = \{((0, 0), A, e), ((1, 0), B, e)\}, \\
\mathcal{T}^1_{AB} = \{((0, 0), A, e), ((0, 1), B, e)\}, \\
\mathcal{T}^2_{AB} = \{((0, 0), A, e), ((-1, 0), B, e)\}, \quad \text{and} \\
\mathcal{T}^3_{AB} = \{((0, 0), A, e), ((0, -1), B, e)\}.
\]

The corresponding reaction types are \(RT^i_{AB} = (\mathcal{T}^i_{AB}, -1)\) for \(i = 0, 1, 2, 3\). Recall that rate \(-1\) indicates that a reaction type is immediate.

Reaction types specify geometric information about elementary steps. Therefore it is more illustrative to depict them graphically than textually. Figure 2.2 contains a graphical representation of the reaction types specified above. We will use this format in the remainder of this monograph.

The presence of reaction steps with infinite rate is translated into immediate reaction types in the model, in a nice, concise way. There are two alternatives to this approach. One other way to capture this approximately is to introduce four reaction types, all with rate constant \(k_{AB}\) and with the same transformations as the immediate types above, choosing \(k_{AB}\) so much larger than \(k_A\) or \(k_B\) that an enabled \(AB\) formation occurs before any enabled adsorption step with high probability. The problem with this approach is that the value \(k_{AB}\) needed to achieve this, grows with the dimensions of the lattice. If the lattice size increases, the average number of enabled adsorption reactions increases as well. If an associative desorption reaction becomes enabled, it competes with many adsorptions. So for larger lattices, the probability of its being the next reaction decreases.
The problem of infinitely fast reactions can be circumvented without resorting to immediate reactions by defining different reaction types. Instead of specifying a reaction type for adsorption of $A$, we can specify reaction types for adsorption of $A$ plus the associative desorption of $AB$, if possible. This leads to the reaction types of Figure 2.3. This figure extends the graphical notation. Multiple reaction types with the same rate constant and pattern are represented by one diagram. The reaction types are obtained by taking an element from each set on the left hand side and replacing every ‘#’ on the right hand side by the particle that has been selected for the corresponding site on the left hand side. Something similar can be done for $B$ adsorption combined with $AB$ desorption. Note that the number of reaction types increases rapidly when a reaction type is combined with the infinite-rate reactions it enables. Using immediate reactions leads to a smaller, cleaner model.

The ZGB model has been studied extensively. It was introduced by Ziff, Gulari, and Barshad (1986). In the paper by Ziff et al. a link was made to the catalytic oxidation of CO on a platinum surface. Our particle type $A$ plays the role of CO, while $B_2$ corresponds to oxygen. The rate constants of the reaction types were scaled so their sum equals one. As a result of this scaling, the rate constant for $B_2$ adsorption can be expressed in terms of the rate constant for $A$ adsorption: $k_{B_2} = \frac{1 - k_A}{2}$.

Later studies introduced all kinds of variations of the original ZGB model, using, for example, a finite rate constant for $AB$ formation, different lattice geometries, or introducing diffusion steps (Brosilow et al., 1993; Clément et al., 1994; Grinstein et al., 1989; Jensen et al., 1990; Mai and von Niessen, 1991, 1993; Meakin, 1990; Meakin and Scalapino, 1987; Tambe et al., 1994; Ziff and Brosilow, 1992). The wealth of existing material on this model makes it suitable for benchmarking the simulation algorithms to be introduced later.
2.4 Some examples

Figure 2.3: Reaction types obtained by combining $A$ adsorption with $AB$ formation. One diagram represents multiple reaction types. The ‘#’ symbol, pronounced ‘same’, indicates that a site remains unchanged. The factors $(\times 2)$ and $(\times 4)$ indicate that there are actually 2 or 4 types, equivalent up to rotation around the origin. Altogether there are 124 reaction types for the combined adsorption of $A$ and $AB$ formation.

2.4.2 Diffusion with lateral interaction

The behavior of particles at a surface is often determined by the presence or absence of particles on nearby sites. These may, for instance, influence the activation barrier for diffusion. Although in our approach all reaction types have predefined rate constants, interactions that influence rate constants can be modeled. We give a simple example of a system with this
Figure 2.4: A square surface with particles that may hop to adjacent sites. A site is adjacent to another if the two unit cells have an edge in common.

The system we consider is one consisting of a square surface and only one species, \( A \), of which a fixed number of particles are present on the surface. In physical terms, we consider a system without adsorption, desorption or chemical reaction. The only events are particle hops from sites to adjacent sites, as in Figure 2.4. The hopping rate depends on the local environment of the particle. We apply the usual Arrhenius formula to express the relation between the activation energy for hopping and its rate constant. I.e., we write \( k = \nu \exp(-E_{\text{act}}/k_BT) \). The activation energy \( E_{\text{act}} \) depends on the local environment as \( E_{\text{act}} = E_{\text{act}}^0 + i E_{\text{int}} \), where \( i \) is the number of \( A \)-covered adjacent sites adjacent to the site of the particle we are considering. A positive value for the interaction energy \( E_{\text{int}} \) models an attractive lateral interaction between adjacent \( A \) particles. The hopping rates in this system do not depend on the interaction energy at the sites to which particles hop. This is not very realistic, but the goal of this example is only to illustrate that lateral interactions can be modeled, not to give a realistic model with lateral interactions.

It follows that our model of the system will have reaction types with one of four rate constants, namely \( k_i = \nu \exp(-(E_{\text{act}}^0 + i E_{\text{int}})/k_BT) \), for \( i = 0..3 \). A particle on the surface may have zero to three nearest neighbors occupied by \( A \); if it has four neighbors occupied by \( A \) it is closed in and cannot hop anywhere. Listing all reaction types does not seem useful since there are 32 of them, and they are very similar. They all have patterns of five sites in a ‘+’ shape, with an \( A \) in the middle in the source configuration, This \( A \) has moved to one of the other sites in the target configuration.

Models similar to this one have also been studied in the literature (Bowler and Hood, 1991; Cao, 1994; Farbman et al., 1996; Kang and Weinberg, 1989; Silverberg and Ben-Shaul, 1987).
2.4 Some examples

2.4.3 A more advanced CO oxidation model

The ZGB model was introduced as a rudimentary model for a monomer-dimer reaction system. In this section we describe a more detailed model for such a system, viz. the oxidation of CO on the (100) face of platinum. This model was developed by Geltman et al. (1998). It incorporates phenomena such as the surface transformation of the Pt(100) surface under the influence of CO adsorption.

The top layer of the Pt(100) surface has a hexagonal structure, such as that depicted in Figure 2.1 on page 12, while the bulk structure has a fourfold symmetry. Adsorbates like CO are known to lift the reconstruction of the top layer, so that it becomes the same as the bulk structure (Jackman et al., 1983; Thiel et al., 1983).

In order to include two surface phases in the model, dummy particle types are introduced to label the sites of the lattice with the phase they are in. The lattice consists of unit cells with two sites each. Dummy particle \( h \) is used to label sites in the hexagonal phase and \( s \) is used to label sites in the square phase. The sites of a unit cell are always in the same phase.

If a unit cell is in the hexagonal phase the two sites can be seen to correspond to the threefold hollows sites of the surface. Both sites of each unit cell may contain adsorbates in this case A unit cell in the square phase has only one hollow site. We let site 0 of the unit cell correspond to this. The reaction types of the model are such that sites 1 of square unit cells are not accessible to adsorbates. They are used only to contain the phase labels.

The species that may adsorb are CO and O\(_2\). CO adsorbs on both phases, while O\(_2\) adsorption on the hexagonal phase is so slow that it can be ignored; in the model O\(_2\) only adsorbs on the square phase. Adsorbed CO molecules may desorb again. Since associative desorption of oxygen is very slow, it is left out of the model. CO and O may desorb associatively, forming CO\(_2\). The reaction types for adsorption and desorption are shown in Figure 2.5 together with the reaction types for CO\(_2\) formation.

Note how the specification of the reactions includes the neighbor relations, allowing both the square and hexagonal phases to be included in the model.

In the square phase every adsorption site has four nearest neighbors: sites \((x \pm 1, y/0)\) and \((x, y \pm 1/0)\) are adjacent to \((x, y/0)\). Hence there are two reaction types for O\(_2\) adsorption. In the hexagonal phase every site has three neighbors. CO adsorption is only possible at sites that do not have hexagonal neighbors occupied by CO. Adjacent cells in the square phase do not have to be vacant. CO adsorption is also possible in vacant unit cells in the square phase. When CO desorbs, the correct phase label has to be assigned to the vacated site. Figure 2.5 shows how this is done. Reaction between CO and O is possible within the square phase and across a boundary between the two phases. The square phase is dominant, making the symmetry of the reaction fourfold.

The surface reconstruction process is modeled by additional reaction types. If a nucleus of five CO molecules exists in the hexagonal phase, then this patch may transform into the
Figure 2.5: Reaction types for CO oxidation model of (Gelten et al., 1998). The adsorption and desorption reactions are shown as well as the CO₂ formation steps. Labels $h$ and $s$ are used to determine the surface phase. $P$ is an abbreviation of $\{h, u\}$. 
2.4 Some examples

Figure 2.6: Reaction types for the CO oxidation model of (Gelten et al., 1998). The reaction types that model the surface transformation processes are shown.

square phase. The nucleus size is based on the experimental results of Hopkinson et al. (1993). Hexagonal unit cells containing CO and adjacent to square unit cells containing CO may be trapped into the square phase. Individual vacant unit cells in the square phase may revert back to the hexagonal phase. Figure 2.6 depicts the reaction types that describe these processes. The nucleation reaction may lead to a square unit cell in which CO is adsorbed at site 1. An immediate reaction is included to remove such illegal states.

The modeling of the surface transformation takes two aspects of the physical system into account: the difference in the neighbor relations between sites in the two phases and the difference in adsorption of O$_2$ on the two phases. The difference in site density between the two phases is not taken into account correctly. In reality the site density of the square phase is 16% lower than that of the hexagonal phase, while in the model it is 50% lower.

The complete model consists of 50 reaction types. A variant of the model in which CO adsorbs onto the top sites of the hexagonal phase and in which CO diffusion is included consists of 51 reaction types. Both models give rise to spatio-temporal pattern formation and oscillatory behavior. Because of this they make interesting test cases for the simulation algorithms discussed in Chapter 3.
2.5 Reaction waiting times

This section contains some properties of the kinetics of surface reaction systems. In particular, we investigate some consequences of the kinetics assumption made in Section 2.3.2. We start by investigating individual reactions, and then focus on system evolution.

The first property of the kinetics is one about the time that elapses before an enabled reaction occurs if it is considered in isolation. We exclude immediate reactions from the discussion because, by definition, no time elapses between the enabling and occurrence of immediate reactions. We consider a system \((\mathcal{L}, \mathcal{D}, \mathcal{R}T)\) in a stable state \(\sigma\) at time \(t_0\), and assume that only one reaction, \(e\) is enabled in \(\sigma\). The rate constant of \(e\) is denoted by \(k\), and the time that elapses before it occurs by \(W\). The variable \(W\) is referred to as \(e\)'s waiting time. As the occurrence of reactions is described in a stochastic manner, \(W\) is a random variable. The probability of \(W\) exceeding \(w + dw\), for \(w \geq 0\), is equal to the probability that \(e\) does not occur in \((t_0, t_0 + w)\) and does not occur in \((t_0 + w, t_0 + w + dw]\). According to the kinetics assumption of Section 2.3.2, the probability of \(e\) not occurring in \((t_0 + w, t_0 + w + dw]\) is equal to \(1 - kdw\), irrespective of the amount of time \(e\) is already enabled at \(t_0 + w\). Hence, the probability of \(W\) exceeding \(w + dw\) equals \(\mathbb{P}[W > w](1 - kdw)\). Rewriting and taking the limit \(dw \to 0\), we obtain

\[
\frac{d \mathbb{P}[W > w]}{dw} = -k \mathbb{P}[W > w]
\]

for \(w \geq 0\). Integration yields the result we are after:

\[
\mathbb{P}[W > w] = \exp(-kw) .
\] (2.2)

In words, the reaction’s waiting time has an exponential probability distribution. Furthermore, the waiting time is independent of the current time \(t_0\).

Exponential distributions are memoryless. This means that if a reaction is enabled at some time \(t_0\), its waiting time is distributed according to (2.2), irrespective of the time at which it became enabled. Appendix A contains more information on the exponential distribution.

The next question we ask ourselves is what we can say about the waiting time of reactions if more than one reaction is enabled. Consider a system in some stable state \(\sigma\) at some time \(t_0\) and let \(e \in E(\sigma)\). By \(W(e)\) we denote the time that elapses from \(t_0\) until \(e\) occurs, given that \(e\) remains enabled in all states the system visits between \(t_0\) and \(e\)'s occurrence. Owing to the condition that \(e\) not be disabled between \(t_0\) and its occurrence, the state of the system might as well be \(\sigma\) from \(t_0\) until \(e\) occurs, as far as the distribution of \(W(e)\) is concerned. So with the above definition, the kinetics assumption implies that waiting time for an enabled reaction \(e\) with rate constant \(k\) has an exponential distribution with parameter \(k\). In formula,

\[
\mathbb{P}[W(e) > w] = \exp(-k(e)w) ,
\] (2.3)
for any \( w \geq 0 \). The reverse is true as well. For any discrete event system, in which waiting times for the events have exponential probability distributions, the probability that some enabled event \( e \) occurs in a vanishingly small interval \((t, t + dt)\) is equal to \( k(e) dt \), if \( W(e) \) has an exponential distribution with parameter \( k(e) \).

Now that we know more about the waiting times of individual reactions, we can investigate properties of waiting times for sets of reactions. According to the kinetics assumption, the probability of more than one reaction occurring in an interval \((t, t + dt)\) is \( o(dt) \), if immediate reactions are excluded. This means that for \( dt \to 0 \), the probability of a reaction occurring in \((t, t + dt)\) in state \( \sigma \), is simply \((\Sigma e: e \in E(\sigma) : k(e)) dt\). By definition, this is equal to \( k(E(\sigma)) dt \). We can now calculate the distribution of the waiting time for the next reaction in a stable state \( \sigma \); the calculation is analogous to that of the waiting time for one reaction. The result is

\[
\mathbb{P}(W(E(\sigma)) > w) = \exp(-k(E(\sigma))w),
\]

for \( w \geq 0 \).\(^2\) Note that we do not need the condition here that the reactions in \( E(\sigma) \) are continuously enabled until the next reaction occurs. This condition is vacuously true as the state does not change until the next reaction occurs.

In general, for a set of reactions \( S \subseteq E(\sigma) \), the distribution of \( W(S) \), the waiting time for the next reaction from \( S \) is given by

\[
\mathbb{P}(W(S) > w) = \exp(-k(S)w),
\]

for \( w \geq 0 \).

Having at our disposal the probability distributions of waiting times of reactions, we can calculate the probability that a reaction \( e_0 \) occurs before another reaction \( e_1 \). It is simply the probability that the waiting time of \( e_0 \) is smaller than that of \( e_1 \). As the definition of the waiting time of a reaction includes the condition that it be enabled continuously until its occurrence, \( W(e_0) \) and \( W(e_1) \) are stochastically independent. Property A.1 in Appendix A shows that

\[
\mathbb{P}(W(e_0) < W(e_1)) = \frac{k(e_0)}{k(e_0) + k(e_1)},
\]

for \( e_0, e_1 \in E(\sigma) \). This result generalizes to sets of reactions. One instance is of particular interest, because it is part of another formulation of the kinetics of systems. For a system in stable state \( \sigma \) and for \( e \in E(\sigma) \), the kinetics assumption is equivalent to

\[
\mathbb{P}(W(E(\sigma)) > w) = \exp(-k(E(\sigma))w)
\]

\[
\mathbb{P}(W(e) < W(E(\sigma) - \{e\})) = \frac{k(e)}{k(E(\sigma))}.
\]

\(^2\)The same result follows if we calculate the distribution of the minimum of the waiting times of the individual reactions in \( E(\sigma) \) (see Appendix A).
Above we have already seen that (2.7) follows from the kinetics assumption of Section 2.3.2. The reverse holds as well. Consider a system in a state $\sigma$ at some time $t$ and let $e \in E(\sigma)$. We calculate the probability that $e$ is the next reaction and that it occurs in $(t, t + dt]$, for $dt$ vanishingly small. As $dt$ is small, the probability that the waiting time of the next reaction is at most $dt$ equals $f_{W(E(\sigma))}(t)dt = k(E(\sigma))dt$, where $f_{W(e)}$ denotes the probability density function of random variable $W(e)$. If a reaction occurs, the probability that it is $e$ is $\frac{k(e)}{k(E(\sigma))}$, by (2.7b). The product of these two probabilities is $k(e)dt$. This product gives the probability of $e$ occurring in $(t, t + dt]$ and being the next reaction, provided that the next reaction and the waiting time for the next reaction are stochastically independent. We show that the next reaction and the waiting time of the next reaction are indeed independent. Let $w \geq 0$, then

\[
\mathbb{P}(W(e) \leq w \wedge W(e) < W(E(\sigma) - \{e\})) = \begin{cases} 1 & \text{if } W(e) \text{ and } W(E(\sigma) - \{e\}) \text{ are independent} \\ \int_0^w f_{W(e)}(u) du \int_u^\infty f_{W(E(\sigma) - \{e\})}(v) dv & \text{if } \text{waiting times have exponential distributions} \\ \int_0^w k(e) \exp(-k(e)u) du \int_u^\infty k(E(\sigma) - \{e\}) \exp(-k(E(\sigma) - \{e\})v) dv & \text{by Calculus} \\ \int_0^w k(e) \frac{k(e)}{k(E(\sigma))} (1 - \exp(-k(E(\sigma))w)) & \text{by Calculus} \\ \mathbb{P}(W(e) < W(E(\sigma) - \{e\})) \mathbb{P}(W(E(\sigma)) \leq w) & \text{by (2.7)} \end{cases}
\]

There is one condition left to verify, viz. that the probability that two reactions occur in $(t, t + dt]$ is $o(dt)$. The probability that $e$ and $e'$ both occur in $(t, t + w]$ and $e$ is the next reaction is

\[
\int_0^w f_{W(e')}(u') du' \int_0^u f_{W(e)}(u) du,
\]

for any $w \geq 0$. Taking $w = dt$ and then taking the limit $dt \to 0$, we see that the probability of $e$ and $e'$ both occurring in $(t, t + dt]$ is $o(dt)$.

### 2.6 Master equation

Stochastic models of physical systems are often based on a master equation (van Kampen, 1992). In this section we show that the behavior of the systems that we defined is governed by a master equation as well.
To this end we consider a system \((\mathcal{L}, \mathcal{D}, \mathcal{R})\) without immediate reactions in some state \(\sigma\). We denote the state of the system at time \(t\) by \(\Phi(t)\) (\(\Phi\) is the stochastic process describing the system state). The probability of the system being in state \(\sigma\) at time \(t + \delta t\) can be expressed in terms of the probabilities of finding the system entering \(\sigma\) in the interval \((t, t + \delta t]\) and the system not leaving \(\sigma\) if its state at time \(t\) is \(\sigma\) already. We get the following equation:

\[
\mathbb{P}(\Phi(t + \delta t) = \sigma) = (\Sigma \sigma', e : \sigma' \xrightarrow{e} \sigma : \mathbb{P}(\Phi(t) = \sigma')k(e)\delta t) + \mathbb{P}(\Phi(t) = \sigma)(1 - k(E(\sigma)))\delta t.
\]

Rewriting and taking the limit \(\delta t \to 0\), we obtain the master equation for systems without immediate reactions:

\[
\frac{d \mathbb{P}(\Phi(t) = \sigma)}{dt} = (\Sigma \sigma', e : \sigma' \xrightarrow{e} \sigma : \mathbb{P}(\Phi(t) = \sigma')k(e)) - (\Sigma \sigma', e : \sigma \xrightarrow{e} \sigma' : \mathbb{P}(\Phi(t) = \sigma)k(e)). \tag{2.8}
\]

For systems with immediate reactions the master equation is somewhat more complicated. First of all \(d \mathbb{P}(\Phi(t) = \sigma)/dt\) is not defined for \(\sigma\) a transient state: by definition systems spend zero time in transient states. Hence, only stable states may occur in the master equation. Since we do not allow systems in which infinite sequences of immediate reactions occur, it is possible to eliminate the immediate reactions, as explained in Section 2.3.2. A master equation can be formulated for this system without immediate reactions.

### 2.7 Discussion

We have presented a way of modeling surface processes. In this model reactants may adsorb at discrete surfaces. The processes occurring at the surface are described by reaction types. These specify which state changes may occur and when they occur. The latter aspect is described in a stochastic way. As a result, our description is a stochastic process. This stochastic process is Markovian: given the current state, the future behavior is independent of the past. Furthermore, the stochastic process is stationary. This means that the probability of the system going from a state \(\sigma\) at some time \(t\) to a state \(\sigma'\) at time \(t + u\) \((u \geq 0)\) does not depend on \(t\), but on \(u\) only.\(^3\)

Stationary Markov processes have been studied at length and techniques are available to analyze them (Bhat, 1972; Gamerman, 1997; Isaacson and Madsen, 1976). Most of these techniques aim to compute the equilibrium distribution of processes or properties of this

\(^3\)In Chapter 4, where we discuss systems with time-dependent rate constants, the processes are not stationary anymore.
distribution. In our case this would amount to solving the master equation (2.8). We aim to study the evolution of systems over time more than to obtain explicit solutions of master equations. In other words, we do not only want to know how systems behave at equilibrium, but also how they evolve towards equilibrium.

The occurrence of a reaction is modeled as an instantaneous change in the state of a system. Several ways of describing systems in which transitions are modeled by discrete events have been introduced in the literature (Ajmone Marsan et al., 1995; Zeigler, 1976). We have chosen not to use any of these. The reason is that notions such as reaction types and spatial homogeneity, a central concept and assumption in our model, can only be introduced as derived notions in the formalisms described in the literature.

Surface processes are also described by cellular automata models. Cellular automata (CA) are discrete space models with the assumption that space is homogeneous. The so-called CA rule determines how the state of a cell may change, usually taking into account the state of nearby cells. Generally, the update rule is applied simultaneously to all cells. This makes them unfit to investigate the results of microscopic fluctuations on the global system behavior, even if probabilistic rules are used (Weimar, 1997).

A class of cellular automata that is of particular interest for the modeling of reactive systems is that of the lattice gas automata (LGA), as described by Boon et al. (1996). The goal of the LGA approach is the investigation of reactive systems at the mesoscopic level beyond phenomenological reaction-diffusion equations. Microscopic fluctuations play an important role in this. A difference between the LGA approach and the approach taken here is found in the modeling of diffusion. Diffusion is treated as any other process in our model, but has a special role in the LGA description of reactive systems. Reaction and diffusion steps alternate in LGAs. A second difference lies in the description of time. Time is discretized in the LGA model. The number of steps performed is the only measure of time.
3

A Taxonomy of Simulation Algorithms

In this chapter we investigate algorithms for simulating the evolution of systems over time. Such simulations are known in the literature as dynamic or kinetic Monte Carlo simulations.

It turns out that there are many possible algorithms to perform dynamic Monte Carlo simulations. We present a classification of the possibilities. Many of the algorithms in this classification have been described in the literature. Conversely, all of the algorithms the author encountered in a literature survey, can be placed in the classification presented here. The classification presented in this chapter contains some new algorithms. Besides describing the possible simulation algorithms systematically, we also present a performance model and compare the algorithms.

3.1 Introduction

As mentioned above, we investigate algorithms for dynamic Monte Carlo simulation of surface reaction systems. The qualifier 'dynamic' stresses the fact that time is important in this type of simulation. The idea behind dynamic Monte Carlo simulations is to start with some state at time zero, and to generate a sequence of states and sojourn times according to the correct probability distribution. In other words, in a dynamic Monte Carlo simulation the stochastic process that describes the system evolution is sampled.

Classical Monte Carlo methods ignore the kinetic aspect. They are used to calculate properties of systems at equilibrium. These properties are formulated as integrals over state spaces of high dimension. The computation amounts to sampling the state space and calculating
the contributions of the states visited to the integral. Often the function to be integrated is sharply peaked, and only a small fraction of the states contribute significantly. In order to get a good estimate of the integral, the importance sampling technique is used. Regions of the state space with higher contributions are favored over those with small contributions in the sampling if this technique is applied. In Section 3.3.1 we return briefly to the relation between classical and dynamic Monte Carlo simulation. Good introductions to the use of Monte Carlo methods in physics and chemistry can be found in the books by Allen and Tildesley (1987, Ch. 4) and Binder (1979a).

3.1.1 Correctness of simulation algorithms

Before presenting any simulation algorithms, we have to establish how to verify their correctness. Obviously, only enabled reactions should be performed in a simulation. In the model we made a distinction between normal and the immediate reactions. Both kinds of reactions have to be handled correctly in simulations.

Consider a system in a transient state. If a simulation algorithm is to be correct, the next state should be generated by performing an enabled immediate reaction with highest priority. If there are multiple immediate reactions with highest priority, the model prescribes that a uniform selection has to be made.

The requirements on the selection of reactions in transient states determine to a large extent the structure of simulation algorithms. The set of enabled immediate reactions has to be represented by some variable $I$. If the set $I$ is non-empty, one of its elements with highest priority is selected. The state is updated by applying the transformation associated with the selected reaction. The set $I$ is recalculated so that it again contains every immediate reaction enabled in the current state. Depending on the algorithm there may be additional processing for normal reactions that are enabled or disabled as a result of the state change. This procedure is repeated until $I$ is empty. If $I$ is empty, a normal reaction is performed, possibly leading to a transient state again. How this reaction is selected depends on the algorithm. In summary, a simulation algorithm looks as follows.

1. Initialize. i.e., establish the initial state, set current time to 0 and initialize all variables specific to the algorithm. Compute the set of enabled immediate reactions $I$.

2. While $I \neq \emptyset$, select one of its elements with highest priority, say $e$. Update the state according to $e$’s transformation ($\sigma := \mathcal{C}(e, \sigma)$), and update any variables specific to the algorithm. Update $I$ so it contains all immediate reactions enabled in the new state.

3. Perform a normal reaction according to the algorithm, updating current state and time.
3.1 Introduction

4. Calculate the set of enabled immediate reactions \( I \).

5. Repeat from 2.

Selection of a reaction from \( I \) with highest priority is an operation that is performed very often over the course of a simulation. A suitable representation of the set \( I \) has to be chosen to make this selection fast. A representation in which \( I \) is partitioned in subsets that contain reactions of the same priority allows fast selection of a reaction with highest priority. These subsets can be put in an ordered list, giving fast access to the (non-empty) subset with highest priority. Another question that deserves some attention is how to compute which immediate reactions are enabled in a state. The same question arises for normal reactions. We postpone the discussion to the next sections, where it is addressed for normal reactions.

In the remainder of this chapter we omit any calculation related to immediate reactions, describing only those parts of algorithms that involve simulating normal reactions. In other words, we focus on Step 3 of the above algorithm. The reader may fill in the different ways for simulating normal reactions in Step 3 to obtain the full simulation algorithms.

In a stable state, a next state has to be generated by performing a normal reaction. Furthermore, the current time has to be incremented. These operations have to be performed in such a way that the kinetics assumption of Section 2.3.2 is satisfied. This is the case if and only if the simulated waiting times are mutually independent and have exponential distributions, i.e., if for all \( \sigma \) and for all \( e \in E(\sigma) \) Equation 3.1 holds.

\[
\mathbb{P}(W_{\text{alg}}(e) > w) = \exp(-k(e) \cdot w) \quad (3.1)
\]

This is equivalent to showing that in any state the simulated waiting time for the next reaction and the probability of a specific reaction being the next one have distributions as given by Equation (2.7). In formula, the kinetics are correctly simulated if and only if Equation (3.2) holds for all \( \sigma \).

\[
\begin{align*}
\mathbb{P}(W_{\text{alg}}(E(\sigma)) > w) &= \exp(-k(E(\sigma)) \cdot w) \\
\mathbb{P}(W_{\text{alg}}(e) < W_{\text{alg}}(E(\sigma) - \{e\})) &= \frac{k(e)}{k(E(\sigma))} \quad \text{for any } e \in E(\sigma)
\end{align*}
\quad (3.2)
\]

3.1.2 Chapter overview

In the following sections we present a number of algorithms for simulating normal reactions. When describing algorithms, we leave out as much detail as possible in order to elucidate the algorithmic differences. Each algorithm presented is shown to be correct by verifying that only enabled reactions are performed, and that (3.1) or (3.2) is satisfied. If similar algorithms have been described in the existing literature, these are discussed briefly.
Furthermore, relationships between the different algorithms are pointed out. In the discussion of algorithms, we add some details, regarding, for instance, the data structures used, in order to assess computational cost per reaction and memory use. This analysis of cost helps finding possible improvements.

In the first part of the chapter we study algorithms in which events are selected and occurrence times are generated independently. We start with a straightforward simulation algorithm and present three ways in which this algorithm may be generalized. These generalizations introduce parameterized algorithms. The generalization techniques are independent and they can be combined, giving rise to a plethora of algorithms. Some of these are only of academic interest. For the more promising ones we provide a detailed comparison of some performance characteristics in Section 3.4. This comparison addresses such issues as memory use and the amount of computer time spent per simulated reaction.

Readers familiar with discrete event simulation (DES) may have observed that the standard DES algorithm may be used to simulate surface reaction systems. This algorithm and related ones involve the generation occurrence times for enabled events, and repeatedly select the event with minimum occurrence time. Algorithms of this form are discussed in Section 3.7.

In many publications on dynamic Monte Carlo simulation, algorithms are used that do not satisfy (3.1) or (3.2) exactly, but approximate them by taking, for instance, the mean of the probability distributions involved. We discuss these approximate algorithms in Section 3.8. A final section is devoted to so-called hybrid simulations, simulations in which the analytical and numerical approaches are combined.

### 3.2 Basic simulation algorithm

Correctness criterion (3.1) suggest a way to obtain a simulation algorithm: compute the set of enabled reactions and select waiting times for each one, then perform the reaction with minimum waiting time. This approach is described in Section 3.7.

In view of the correctness criterion (3.2), the most straightforward algorithm is to select a time for the next reaction based on the sum of the rate constants of the enabled reactions and then select an enabled reaction with a probability proportional to its rate constant. The name of the algorithm derives from the fact that a weighted selection is used: we call it WRsel, short for weighted reaction selection. The algorithm reads as follows.

**Algorithm 3.1 (WRsel)**

1. Initialize:
   - Set time to zero \( (t := 0) \);
   - set the state to the initial state, \( \sigma_0 (\sigma := \sigma_0) \);
   - construct a set \( E \) containing the enabled reactions in \( \sigma \) \( (E := E(\sigma)) \).
2. Generate a time increment \( w \) by sampling an exponential distribution with parameter \( k(E(\sigma)) \). Increment \( t \) by \( w \).

3. Select a reaction from \( E \) in such a way that any reaction \( e \in E \) has probability \( \frac{k(e)}{k(E)} \) of being selected. Update the state \( \sigma := C(e, \sigma) \) and recalculate \( E \) so that it again contains all enabled reactions \( E := E(\sigma) \).

Step 1 refines step 1 of the algorithm outline on page 30. The other two steps above refine step 3 of the algorithm outline.

This algorithm is obviously correct. First, reactions are selected from the set of enabled reactions. Second, it satisfies (3.2) by explicitly having the prescribed probabilities in steps 2 and 3. The assignments \( \sigma := C(e, \sigma) \) and \( E := E(\sigma) \) are to be seen only as the specification of the operations to be performed on \( \sigma \) and \( E \). When updating the state, for instance, it is useless to assign new values to sites that are not involved in the reaction performed.

In the literature this algorithm has been described a number of times. A literature survey showed four occurrences of WRsel (Auerbach et al., 1995; Battaile et al., 1997; Gillespie, 1976; Saravanan and Auerbach, 1997). Gillespie uses the algorithm to simulate the time evolution of spatially homogeneous systems, as opposed to our heterogeneous systems. In his terminology WRsel is the ‘direct’ method.

Auerbach et al. (1995) and Saravanan and Auerbach (1997) investigate transport phenomena in zeolites. Zeolites are catalysts with a three-dimensional structure. In both papers, the generation of the time increment according to an exponential distribution is left out. Instead, the mean of the distribution is used, thereby ignoring stochastic fluctuations in reaction waiting times. This simplification is made in many publications on dynamic Monte Carlo simulations.

Battaile et al. (1997) give a detailed description of the algorithm and point out that it is related to the algorithm described in (Bortz et al., 1975). They use the algorithm to simulate chemical vapor deposition at the atomic scale.

Algorithm 3.1 does not describe how to calculate the set of reactions enabled in the current state. In the initialization step there is hardly any choice: the enabledness of all reaction types is checked at every unit cell by matching the source pattern in the initial state. After a reaction is performed, \( E \) has to be recalcualted. This can be done either by checking the enabledness of every reaction type at every site again, like in the initialization step. This is computationally very costly. The second possibility consists of two steps. After a reaction, say \( e \), has been selected, every reaction type is matched at those unit cells that make its pattern overlap with one or more sites that will be modified by \( e \)'s occurrence. The ones that are enabled before \( e \) occurs, will not be enabled anymore after \( e \) has been performed; they are removed from \( E \). Note that \( e \) itself is removed. Then, after \( e \) has been
performed, every reaction type is matched again at the unit cells that make it overlap with the sites that have just been modified. The ones that are enabled, were not enabled before \( e \)'s occurrence. These are added to \( E \). Any immediate reactions that are disabled (enabled) by the occurrence of \( e \), are removed from (added to) \( I \).

When implementing WRsel, one must be able to select the elements from the set of enabled reactions with a probability proportional to their rate constant (step 3 of the algorithm). This can be done in several ways. The easiest way is to represent \( E \) by a list. A faster, but more complicated representation involves a tree structure. We discuss these representations in more detail in Section 3.4. At this point the most important fact is that deletion of disabled reactions requires the entire set \( E \) to be traversed because \( E \) is unordered.

The fact that deletion of disabled reactions from \( E \) is such a time consuming operation makes WRsel unfit for simulating large systems. The size of \( E \) grows linearly with the number of adsorption sites, and for every reaction performed, all elements of \( E \) are visited in every iteration of the algorithm. There are two ways to reduce the cost of deletion of disabled reactions. Both are discussed in the next section.

### 3.3 Generalization techniques

Taking WRsel as a starting point, there are several possibilities for obtaining more general algorithms. In the subsections of this section we explore these possibilities. Each forms a generalization of WRsel. Parameters are introduced by the generalizations, and for suitable choices of the parameters we obtain efficient algorithms.

#### 3.3.1 Oversampling

The first way to generalize the WRsel algorithm is to associate with each reaction a sampling rate. Instead of selecting reactions with probabilities proportional to their rate constants, they are selected with probabilities proportional to their sampling rates. Time increments are based on the sum of the sampling rates of the enabled reactions. If the sampling rates are higher than the actual rate constants of the reactions, the oversampling can be compensated by performing selected reactions conditionally. The algorithm reads as follows.

#### Algorithm 3.2 (WRsel-o)

1. Initialize:
   \[
   t := 0; \quad \sigma := \sigma_0; \quad E := \emptyset; \\
   \text{for } e \in E(\sigma) \rightarrow sr : sr \geq k(e); \quad E := E + \{(e, sr)\}; \quad \text{rof} \\
   R := (\Sigma e, sr : (e, sr) \in E : sr); 
   \]

A Taxonomy of Simulation Algorithms
3.3 Generalization techniques

2. Generate a time increment \( w \) by sampling an exponential distribution with parameter \( R \); increment \( t \) by \( w \).

3. (a) Select an element from \( E \), giving probability \( \frac{w}{R} \) to \((e, sr)\).

(b) With probability \( \frac{k(e)}{sr} \), perform the reaction \( e \), and update \( E \) and \( R \). With probability \( 1 - \frac{k(e)}{sr} \), do nothing.

The initialization step contains the statement '\( sr : sr \geq k(e) \)' Its meaning is that an arbitrary value satisfying the condition \( sr \geq k(e) \) is assigned to \( sr \). We only allow sampling rates at least as large as the rate constants. Otherwise the dynamics of the system are not correctly simulated.

We show that this algorithm is correct by verifying that (3.2) is satisfied. The time until the next reaction can be described by a random variable \( X \) that is the sum of a number of independent, exponentially distributed random variables, all with the same parameter \( R \). The number of summands is itself a random variable \( K \) with a geometric distribution: \( K \) is the number of iterations or trials until the reaction is performed. In each iteration the probability that a reaction is performed is

\[
\left( \sum_{e, sr : (e, sr) \in E : \frac{sr}{R} = \frac{k(e)}{sr}} \frac{k(e)}{sr} \right) = \frac{k(E(\sigma))}{R}.
\]

It is straightforward to show that with this definition, \( X \) has an exponential distribution with parameter \( k(E(\sigma)) \). For completeness' sake, we formulate the property exactly. The proof is included in Appendix A.

**Property 3.1**

Let \( X_1, X_2, \ldots \) be independent random variables, all with probability density function \( f_{X_i}(x) = \lambda \exp(-\lambda x) \), for some \( \lambda > 0 \). Let \( K \) be a random variable, independent of the \( X_i \), with a geometric distribution with success probability \( p \). That is, assume that

\[
\mathbb{P}(K = k) = (1 - p)^{k-1} p,
\]

for integer \( k \geq 1 \). Define \( W = (\Sigma k : 1 \leq k \leq K : X_k) \). Then \( W \) has a \( \text{Exp}(p\lambda) \) distribution.

Our next task is to show that in any state \( \sigma \) and for \( e \in E(\sigma) \), the probability of \( e \) being the next reaction performed is equal to \( \frac{k(e)}{k(E(\sigma))} \). If no reaction is performed in a trial, then for any reaction, the probability of it being selected and performed in the next iteration is unchanged. Therefore, the probability of \( e \in E(\sigma) \) being the next reaction performed
is equal to the conditional probability of \( e \) being performed in an iteration given that \( a \) reaction is performed in that iteration. This conditional probability is easily seen to equal

\[
\frac{sr}{R} \cdot \frac{k(e)}{k(E(\sigma))} = \frac{k(e)}{k(E(\sigma))}.
\]

The numerator of the left hand side is the probability that \( e \) is selected and performed, while the denominator is the probability that a reaction is performed in a trial in state \( \sigma \). This completes the proof that the algorithm is correct.

An algorithm similar to this appears in (Binder, 1979b). By making a suitable choice for the sampling rates, we obtain a well-known algorithm, often referred to as 'the standard MC algorithm.' This algorithm is obtained by taking

\[
sr = (\text{MAX } e : e \in E(\sigma) : k(e))
\]

for all reactions. This has a couple of advantages. First, there is no need to store the sampling rate with every reaction in \( E \). Second, the selection of elements from \( E \) is now a uniform selection, reducing the cost of selection to a constant if \( E \) can be represented by an array. This is the reason why we name it URsel, for uniform reaction selection.

There are some disadvantages as well. The deletion of disabled reactions is still as expensive as in the WRsel algorithm. Furthermore, the maximum of the rate constants of the enabled reactions has to be recorded. This disadvantage can be overcome by increasing the sampling rates to

\[
(\text{MAX } RT : RT \in \mathcal{R} : k(RT))
\]

The cure to the latter problem may be worse than the disease. It brings us to the main disadvantage of the algorithm: the success probability \( \frac{k(e)}{sr} \) may be very small for most reactions. The presence of one reaction with a rate constant that is high compared to the other rate constants, may render the algorithm is virtually useless. This has been observed by others (Binder, 1979b; Bortz et al., 1975).

In spite of the drawbacks, the URsel algorithm is probably still used very often (see, e.g., Fichthorn and Weinberg, 1991; Guo et al., 1997; Meng and Weinberg, 1995; Weinketz, 1994). The reason is that it is strongly related to the algorithm described in the seminal paper by Metropolis et al. (1953). There are two differences between URsel and the Metropolis algorithm, both arising from a difference in purpose: we aim for a dynamic view of the system the algorithm is applied to, while Metropolis and co-workers wanted to estimate values of properties for systems at equilibrium. The first difference is that the Metropolis algorithm does not include time. The second is that in the Metropolis algorithm acceptance probabilities of selected transitions are determined by the change in energy of
3.3 Generalization techniques

the system between the old and new state. If the new state has a lower energy than the old state, the transition is accepted unconditionally. If an increase in energy $\Delta E$ is associated with the transition, it is accepted with probability $\exp(-\Delta E/k_B T)$. The result of this transition acceptance scheme is that equilibrium is reached faster and that computation of the properties of interest is faster. This acceptance scheme and schemes like it (Barker, 1965; Kawasaki, 1966) are examples of importance sampling. Importance sampling does not lead to correct simulation of reaction kinetics because activation energies are ignored. Kang and Weinberg (1995) explain this in detail and present examples of the differences that may occur when different sampling methods are used. In (Kang and Weinberg, 1989) there is more discussion on the dynamical view of Monte Carlo simulation when different acceptance probabilities are used.

In the literature on classical Monte Carlo techniques, algorithms in which the selection of events is non-uniform (such as WRsel and the algorithms introduced later) are known as biased sampling techniques (Frenkel and Smit, 1996).

3.3.2 Approximate representation of the set of enabled events

In the discussion of the WRsel algorithm we already saw that maintaining the set of enabled events is a burden, which presses harder on the performance as the system size increases. As noted in that discussion, the decrease in performance can be attributed mainly to the work involved in removing disabled reactions from $E$, whether $E$ is represented by a list, tree, or some other data structure. In the next section we try to reduce the cost of deletion of disabled reactions by ordering stored reactions by their enabling location. In this section we investigate whether it is possible not to delete any reactions from $E$ when they become disabled and still obtain correct simulations.

Suppose that disabled reactions are not removed from $E$. We investigate which problems this causes, and how these can be overcome. The first problem is that a reaction from $E$ might be selected that is not enabled. Instead of simply performing any selected reaction as in WRsel, we add a check to filter disabled reactions. If a disabled reaction is selected, it is simply removed from $E$. As in the previous section, this leads to an algorithm in which the time increment per iteration is smaller – $k(E)$ exceeds $k(E(\sigma))$ if $E$ contains disabled reactions – and not every iteration is successful. In this case the situation is somewhat more complicated than for the WRsel-o algorithm. Variable $E$ changes even in iterations in which no reaction is performed, so the time until the next reaction is not a sum of identically distributed exponential deviates anymore. We address this complication further in the correctness proof of the algorithm we are designing.

Simply discarding disabled reactions is not enough to obtain a correct algorithm. It ignores the possibility that reactions may become disabled and then enabled again. If this happens, then either $E$ has to be inspected to ensure that no multiple instances of one reaction are
created, or we allow multiple instances of reactions in $E$ and deal with these in some way. The former solution, ensuring $E$ never contains multiple instances of any reaction, is not a viable one. Before a reaction may be added to $E$, all elements of $E$ have to be inspected, making insertion of elements in $E$ as expensive as deletion of disabled reactions was before.

For reasons of computational cost, we allow $E$ to contain multiple instances of reactions. In order to obtain a correct simulation algorithm, we must prevent a reaction that occurs more than once in $E$ from being performed with a higher probability than prescribed by its rate constant. One way of achieving this is to store extra information with every reaction in $E$, allowing multiple instances of one reaction to be distinguished. More specifically, if multiple instances of reactions are labeled by the time at which they became enabled, we know that all but the one most recently enabled are not eligible for execution. The one most recently enabled is eligible for execution, provided that it is enabled in the current state. Still, verifying whether a selected reaction has counterparts in $E$ enabled more recently, requires inspection of all elements of $E$, and has to be rejected for that reason.

A better solution is to add labels to the adsorption sites of the lattice as well. Besides the type of particle present, we store the time of the last modification with each site. A reaction is eligible for execution – i.e., it may be performed if it is selected – if it is enabled in the current state, and none of the sites in its pattern have been modified since it became enabled. In fact, if the latter part of the condition is fulfilled, then so is the former: a reaction only becomes disabled if its source configuration does not match in the current state anymore, implying that some of the sites in the reaction’s pattern have been modified.

The amount of time simulated will be a floating-point variable. Using the amount of time simulated for enabling times of reactions and modification times of lattice sites does not give an ordering that is strict enough. It is possible that time increments are so small that $t$ has the same value in two consecutive iterations of the algorithm. The number of iterations performed increases by one every iteration, giving the required ordering.

The analysis results in the following proposed algorithm. It is called WRsel-a, as it is a generalization of WRsel in which only an approximation of the set of enabled reactions is represented.

Algorithm 3.3 (WRsel-a)

1. Initialize:
   
   $t := 0; \sigma := \sigma_0; E := \emptyset ;$
   
   for $l \in \mathcal{L} \rightarrow lm[l] := 0; \text{rof}$
   
   for $e \in E(\sigma) \rightarrow E := E + \{(e, 0)\}; \text{rof}$
   
   $R := (\Sigma e : e \in E(\sigma) \cdot k(e));$

2. Generate a time increment $w$ by sampling an exponential distribution with parameter $R$; increment $t$ by $w$. 
3.3 Generalization techniques

3. Select and remove an element from $E$ giving probability $\frac{k(e)}{R}$ to $(e, t_{en})$. If $(e, t_{en})$ is eligible for execution, then update $\sigma$ and $lm$; add newly enabled reactions to $E$ with the iteration count for enabling time; update $R$.

The algorithm uses a variable $lm$ to maintain for each lattice site the number of the iteration in which it was last modified. Checking whether a reaction is eligible involves inspection of $lm$ for all sites in the reaction's pattern.

Having already established that in any state only reactions from $E(\sigma)$ are performed, all we need to do to prove the algorithm correct is to verify that the conditions (3.2) are satisfied. For a change, we start with (3.2b). In this calculation we write $\mathcal{E}(e, t_{en}, lm)$ for the condition that a reaction $e$ with enabling time $t_{en}$ is eligible for execution in $lm$. The probability that a reaction is performed in an arbitrary iteration of the algorithm is calculated as follows.

\[
\begin{align*}
(\Sigma e, t_{en} : (e, t_{en}) \in E \land \mathcal{E}(e, t_{en}, lm) : \frac{k(e)}{R}) \\
= \{ (e, t_{en}) \in E \land \mathcal{E}(e, t_{en}, lm) \equiv e \in E(\sigma) \} \\
(\Sigma e : e \in E(\sigma) : \frac{k(e)}{R}) \\
= \text{Definition of } k() \} \\
\frac{k(E(\sigma))}{R}
\end{align*}
\]

The probability that a specific reaction $e \in E(\sigma)$ occurs is $\frac{k(e)}{R}$. Hence, the conditional probability of $e \in E(\sigma)$ occurring given that a reaction occurs, equals

\[
\frac{k(e)}{R} \div \frac{k(E(\sigma))}{R} = \frac{k(e)}{k(E(\sigma))}.
\]

As this probability depends on none of the program variables $E$, $R$, or $lm$, we conclude that the probability of $e$ being the next reaction is $\frac{k(e)}{k(E(\sigma))}$ for any $e \in E(\sigma)$.

Our remaining obligation is to show that the time until the next reaction has an exponential distribution with parameter $k(E(\sigma))$. This requires a generalization of Property 3.1 on page 35. The generalization lies in the fact that in every iteration until the next successful one, both the probability of success and the probability distribution from which the time increment is generated are different. This is caused by the deletion of non-eligible reactions from $E$ and the accompanying decrease of $R$ in every unsuccessful iteration. We only mention the generalized property here, and relegate its proof to Appendix A (see page 174).
Property 3.2

Let $X_1, X_2, \ldots$ be independent random variables, with probability density functions

$$f_{X_i}(x) = \lambda_i \exp(-\lambda_i x)
,$$

with $\lambda_i > 0$ for $i \geq 1$. Let $\lambda > 0$ such that $\lambda \leq \lambda_i$ for all $i \geq 1$. Next let $Z_1, Z_2, \ldots$ be independent random variables, independent of the $X_i$, with

$$\Pr[Z_i = 1] = 1 - \Pr[Z_i = 0] = \frac{\lambda}{\lambda_i}.$$

Define random variables $K$ and $W$ by

$$K = (\text{MIN } i : 1 \leq i \land Z_i = 1 : i),$$

$$W = (\sum k : 1 \leq k \leq K : X_k).$$

Then $W$ has an exponential distribution with parameter $\lambda$ if and only if

$$\lim_{k \to \infty} (\prod i : 1 \leq i \leq k : 1 - \frac{\lambda}{\lambda_i}) = 0. \tag{3.3}$$

In words, $W$ has an exponential distribution if and only if the probability of infinitely many unsuccessful trials is zero.

Taking for $\lambda_i$ the parameter of the distribution from which the time increment in the $i^{th}$ trial is sampled, and taking $k(E(\sigma))$ for $\lambda$, we see that the waiting time for the next reaction has the correct distribution. The condition (3.3) is satisfied for this choice of the $\lambda_i$ and $\lambda$, as $\frac{\lambda}{\lambda_i}$ goes to 1 as more and more non-eligible reactions are removed from $E$. Hence WRsel-a is a correct simulation algorithm.

This way of dealing with disabled reactions was introduced by Lukkien et al. (1998). We will encounter the algorithm described by Lukkien and coworkers later in this chapter, as a combination of WRsel-a with another generalization technique.

Before we go on to the third and final generalization of WRsel, two more remarks about WRsel-a are in order. First, we have a way of reducing the cost of dealing with disabled reactions, but the gain is not yet clear. Under adverse conditions, most of the time might be spent selecting and removing non-eligible reactions from $E$. Since non-eligible reactions are not removed immediately, memory use increases by a potentially large factor. A careful analysis of the benefits is in order, and will be presented in Section 3.4.3.

The second remark to be made is that there is another way of representing $E(\sigma)$ approximately, that does not introduce any of the problems related to multiple instances of reactions. In this alternative solution, $E$ remains the same throughout a simulation. It represents the set of all possible reactions $\{RT, I : RT \in \mathcal{T} \land I \in \mathcal{E} : (RT, I)\}$. At any time it will probably contain many reactions that are not enabled, but there is no cost for updating $E$. A variation of this solution is discussed in Section 3.5.1.
3.3.3 Selecting reactions in multiple stages

The algorithms we have seen so far perform the selection of a reaction in one step. An element from a large set is selected. The cost of a weighted selection from a set \( E \) grows as \( \log_2 |E| \) (see Section 3.4.2 for details) and can become quite high. The URsel algorithm gives a way to remedy this, by transforming the weighted selection into a uniform one, with cost \( \Theta(1) \). If, however, rate constants of reactions vary over a wide range, this does not lead to acceptable performance. In this section we explore an altogether different solution to reduce the cost of selecting reactions. The result is an algorithm that allows us to exploit the one aspect of the modeling that the preceding algorithms ignore, namely the fact that there is a limited number of reaction types. The idea is to group reactions within \( E(\sigma) \) and make the selection of reactions hierarchical. First a group is selected and then a reaction is selected from that group.

Algorithm 3.4 (WRsel-h)

1. Initialize:
   \[ t := 0; \sigma := \sigma_0; R := (\Sigma e : e \in E(\sigma) : k(e)) \]
   Construct a set \( E \), the elements of which are sets of enabled reactions. More specifically, construct sets \( s_0, \ldots, s_{H-1} \) and define \( E = \{s_0, \ldots, s_{H-1}\} \). We require \( (\cup i :: s_i) = E(\sigma) \) and \( s_i \cap s_j = \emptyset \) for \( i \neq j \).

2. Generate a time increment \( w \) by sampling an exponential distribution with parameter \( R \); increment \( t \) by \( w \).

3. Select element \( s \) from \( E \) with probability \( \frac{k(s)}{R} \). Having selected \( s \), select an element \( e \) from it with probability \( \frac{k(e)}{k(s)} \). Update \( \sigma, E, \) and \( R \) according to the occurrence of \( e \).

Obviously, only enabled reactions are performed. The time increment in Step 2 is the same as in WRsel, so (3.2a) is satisfied. The probability that and enabled reaction \( e \) is selected is easy to calculate. Suppose that \( e \in s \) and \( s \in E \). The probability that \( e \) is selected is

\[
\frac{k(s)}{(\Sigma s' : s' \in E : k(s'))} \cdot \frac{k(e)}{k(s)} = \frac{k(e)}{k(E(\sigma))}.
\]

This shows that (3.2b) is satisfied too, and therefore the algorithm is correct.

The idea of grouping reactions and selecting reactions for execution in a hierarchical way has been applied by many authors. The usual choice is to put all reactions of the same type into one group. This leads to a simplification of the selection within the groups. As reactions of the same type have the same rate constants, the selection becomes uniform. The selection of the group becomes weighted. The number of groups is fixed and much
smaller than the number of enabled reactions, thereby reducing the cost of selection. This algorithm we call WTselULsel (weighted type selection, uniform location selection). It has been invented and reinvented a number of times in different contexts (Bortz et al., 1975; Bowler and Hood, 1991; Kohli and Ives, 1972; Nordmeyer and Zacer, 1992).

The issue of generating time increments that give rise to correct dynamical behavior has been discussed in many papers (Bowler and Hood, 1991; Cao, 1994; Fichthorn and Weinberg, 1991; Kang and Weinberg, 1989). These discussions are not all that clear, especially the ones that take average inter-event times as a starting point, without mentioning the stochastic kinetics assumption.

Another way to group reactions is according to the location at which they occur. Then selection of a reaction amounts to two weighted selections: first a location, then a reaction enabled at that location. We call this algorithm WLselWTsel. An advantage of this algorithm is that the structure it imposes on $E$ for the selection of reactions allows deletion of disabled reactions without having to search all of $E$. To this end $E$ has to be represented by a data structure that supports fast indexing of the groups associated with the locations.

A special case of the WLselWTsel algorithm is described by Jensen (1993a,b). He investigates branching annihilating random walk systems (see, e.g., Bramson and Gray, 1985). In such systems particles may hop to one of their nearest neighbor sites, both particles being annihilated if this site is already occupied. Alternatively, a particle may produce $n$ new particles on the closest neighboring sites, where $n$ is a parameter of the model. If a location is occupied, all reaction types are enabled at that location. On the other hand, no reactions are enabled at empty sites. A list is maintained containing the occupied sites. In every iteration a site is selected uniformly from this list, after which one of the reaction types is selected. The rate constants of the reaction types are normalized so their sum is 1. The time increments are generated by taking the average of the exponential distribution with parameter $k(E(\sigma))$. To the author’s knowledge, no other instances of WLselWTsel occur in the literature.

In the version of WLselWTsel as described above, no enabled reactions have to be stored because all reaction types are enabled at the sites containing particles. In general, an arbitrary subset of the reaction types may be enabled at a site. There is, however, a variant of WTselWLsel in which no reactions are stored. Instead of storing the reactions so that one can be selected uniformly after a type has been selected, it is possible to just search the lattice for an enabled reaction of the selected type. In order to be able to select reaction types with the correct probability, the number of enabled reactions of every type is needed. Thus, after performing a reaction it is still necessary to perform all computations to find out which reactions have become enabled or disabled. It is just not necessary to delete reactions from or add them to a large set. This algorithm we dub WTselULsearch.

The biggest benefit of WTselULsearch is that the set of enabled reactions is not stored. As a result its memory use is very low and no expensive deletion operations are necessary. On
the downside, it may take a large number of steps to find an instance of a selected reaction type if only few reactions of that type are enabled.

Algorithm 3.4 and the ones deriving from it select reactions in two steps. It is of course possible to use more steps than two. In the literature reactions are sometimes selected in three steps, in particular in systems that contain diffusion reactions. A selection scheme used for such systems is to first select a reaction type such as adsorption or diffusion, then a location, and in the case of diffusion, a destination site for the moving particle. An example of such an algorithm can be found in Nordmeyer and Zaera (1992). From a conceptual point of view this way of selecting reactions may be nice because it separates different processes. However, it makes the simulation algorithm more complex, without obvious advantages in terms of performance.

3.4 Complexity analysis

In the preceding sections we treated a basic simulation algorithm and three techniques to generalize this algorithm. The goal of the generalizations was to obtain algorithms that are computationally less expensive than the basic WRsel algorithm. In this section we present a more thorough analysis of the complexity of the algorithms described so far. We analyze the computational cost per reaction and the memory use of the algorithms.

Often, in performance analysis of algorithms, the asymptotic behavior is considered. In such a case the quantity of interest (running time, memory use) is not calculated in detail, but its growth rate is given. The so-called big-oh notation is used for growth rates (Knuth, 1976). It is defined by

$$g \in O(f) \equiv (\exists c, m : c > 0 \land m \geq 0 : (\forall n : n \geq m : g(n) < cf(n))),$$

for functions $f$ and $g$ on the natural numbers.

A complexity analysis in terms of growth rate functions is rather coarse. The functions $n \mapsto n$ and $n \mapsto 100n$ are both $O(n)$, for instance, while an algorithm that performs a computation in $n$ steps is often vastly preferable over one that establishes the same result in $100n$ steps. This is the reason why we analyze the performance of simulation algorithms in more detail. The difference between $n$ and $100n$ could be the difference between a simulation that takes an hour and one that takes four days.

Another distinction made in complexity analysis is that between analysis of worst case behavior and expected behavior. The former allows statements about guaranteed performance, while the latter says something about the average over a number of samples of the quantity considered. Our main measure of performance measure is the time per reaction for a simulation. Since many reactions are performed in one simulation, the expected time per reaction is a more meaningful quantity to analyze than the worst case time per reaction.
3.4.1 Simplifying assumptions

The amount of memory used and the cost per reaction depend not only on the simulation algorithm, but also on the system simulated. The dependency on the system size can be analyzed without too much trouble, as system sizes are ordered. Other aspects of systems are less easily ordered. For example, it is hard to predict how many reactions will be enabled on average for a given system, without performing simulations. Therefore we analyze the performance for a model system. In this section we describe the assumptions made about this model system.

Algorithms do not differ in performance with respect to immediate reactions. Therefore it seems reasonable to focus our attention on normal reactions by considering a system $(L, D, RT)$ without immediate reaction types.

As mentioned before, one of our goals is to simulate the dynamical behavior of systems that are not at equilibrium. In our complexity analysis, however, we assume that the number of enabled reactions may be considered a constant; an assumption that is generally only valid over relatively short intervals of time. In particular, in off-equilibrium simulations, the number of enabled reactions cannot be expected to remain constant over entire simulations. However, the number of enabled reactions usually changes only significantly after many reactions have been performed. Many simulations can be considered to consist of a modest number of intervals in each of which the number of enabled reactions is roughly constant. The performance model to be introduced below is valid for each of the individual intervals.

There are two notable exceptions to the assumption that the number of enabled reactions in a simulation does not change in short amounts of time. First, if a normal reaction is followed by a large number of immediate reactions, the number of enabled reactions may change considerably in small periods of simulated time. Also, if the system being simulated is small, then a small change in the absolute number of enabled reactions constitutes a large relative change. In these situations the performance analysis that follows is not valid.

If the number of enabled reactions does not change over time, then on average, every reaction disables as many reactions as it enables. We denote the expected number of reactions disabled (enabled) per reaction by $n$. Furthermore, we denote the sum of the numbers of sites of the patterns of all reaction types by $n_{src}$. Finally, we write $n_{mod}$ for the average number of sites modified by reactions that are performed.

3.4.2 Complexity of basic operations

The algorithms have a number of operations in common, such as sampling an exponential distribution, adding elements to a set or deleting them from it, checking for newly enabled
3.4 Complexity analysis

reactions, and so on. The cost of these operations can be analyzed in isolation, and charac-
terized by a small number of parameters. Each of these parameters gives the cost of a basic
operation on, for example, a floating-point number or a data structure.

All algorithms we discussed have a step in which an exponential distribution is sampled.
On most computers functions are available for sampling a uniform distribution on the unit
interval. By means of the inverse function method (see, e.g., Knuth, 1969) we can trans-
form samples from a uniform distribution into samples from an exponential distribution.
We explain briefly how this method works. Let $X$ be an arbitrary random variable; de-
ote its probability distribution function by $F_X$. Let $U$ be a random variable with uniform
distribution on $(0, 1)$. Then

$$
P[X \leq x] = F_X(x) = P[U \leq F_X(x)].$$

As $F_X$ is a probability distribution function, it is a non-decreasing function of $X$. If it is
strictly increasing it has an inverse and then $P[U \leq F_X(x)] = P[F_X^{-1}(U) \leq x]$. It follows
that we can obtain samples of $X$ by generating a sample $u$ of $U$ and applying $F_X^{-1}$ to it. In
particular, if $X$ has an exponential distribution with parameter $r$, then

$$F_X^{-1}(u) = \frac{\ln(1 - u)}{r}.$$  

If $U$ is uniformly distributed on $(0,1)$ then $1 - U$ is as well. Hence we can also use $-\frac{\ln u}{r}$
to obtain samples of an exponential distribution with parameter $r$. In order to characterize
the cost of generating such a sample, we introduce $\tau_{\text{rand}}$ for the cost of sampling the unit
interval uniformly and $\tau_{\text{fp}}$ for the cost of a floating-point operation. For reasons of simplic-
ity we attribute the same cost to the calculation of the natural logarithm and floating-point
division, obtaining $\tau_{\text{rand}} + 2\tau_{\text{fp}}$ for the cost of generating a time increment. The time incre-
ment is added to the current time, giving rise to another floating-point operation. The total
time spent on updating the time is

$$\tau_{\text{time}} = \tau_{\text{rand}} + 3\tau_{\text{fp}}$$

per iteration, for all algorithms considered.

Most algorithms discussed contain operations on weighted sets. A weighted set is a set of
ordered pairs: a data item (reaction) and a weight (rate constant). The operations performed
on weighted sets are

- random selection with probabilities for the elements proportional to their weight, also
called weighted selection,
- deletion of arbitrary elements, and
• insertion of elements.

The computational cost of these operations depends on the representation used. We discuss
two possible representations and analyze the cost of the operations. The first representation
we discuss is an array representation and the second a tree representation.

For the array representation we assume that we have an array that is large enough to contain
all elements of the largest set we have to represent. This can be established for our simu-
lation algorithms by starting with a small array and doubling its size every time it is filled
completely until the assumed equilibrium is reached. We use a variable to maintain the
number of array positions used or the number of elements in the set. Then a new element
can be inserted by putting it at the first unused array position and updating the number of
positions used. We assume the cost of this to be \( \tau_{fp} \). Given an index, the element at that po-
sition in the array can be deleted by overwriting it with the last element and decrementing
the number of positions used. The cost is the same as that of insertion of an element.

Weighted selection of an element from the array can be done as follows. First the unit
interval is sampled uniformly. The sample is multiplied by the sum of the weights of the
elements in the array, resulting in a number \( w \). The element at position \( i \) is selected, with \( i \)
the unique solution of

\[
(\sum j : 1 \leq j < i : w_j) < w \leq (\sum j : 1 \leq j \leq i : w_j),
\]

where \( w_j \) is the weight of the element at position \( j \) in the array. The expected cost of
finding the index \( i \) satisfying Equation (3.4) is \( \frac{1}{2}|S|\tau_{fp} \) for weighted set \( S \). The sum of the
weights cannot be changed incrementally as elements are inserted in \( S \) or deleted from it
for reasons of loss of accuracy. The cost of computing the sum of the weights is \( |S|\tau_{fp} \).
Adding the cost of sampling the unit interval and the multiplication of the sample by the
sum of the weights, we obtain

\[
\tau_{wset}(|S|) = \tau_{rand} + (\frac{3}{2}|S| + 1)\tau_{fp}
\]

for the cost of weighted selection from \( S \) for this array implementation.

A tree implementation is generally faster than an array implementation except maybe for
small sets. We describe an implementation using complete binary trees, as suggested by
Knuth (1969, Sect. 3.4). The leaves of the tree contain the elements of \( S \) and their rate
constants. The internal nodes contain both the sums of the rate constants of the leaves of
their left subtrees and those of their right subtrees. This scheme for numbering the internal
nodes of the tree was introduced by Wong and Easton (1980). It minimizes the number of
operations for weighted selection of elements.

Weighted selection of an element starts with sampling the unit interval uniformly and mul-
tiplying the sample by the sum of the weights of the elements of \( S \), giving a number \( w \).
This number is repeatedly compared to values stored in internal tree nodes. If it is smaller than the sum of the weights of the left subtree, then the left branch is taken. Otherwise, the right branch is taken and \( w \) is decremented by the node value it was compared with. The process stops when a leaf is reached. Figure 3.1 shows an example.

For an \( l \) level tree, the expected cost of the weighted selection of an element is

\[
\tau_{\text{rand}} + \tau_{\text{fp}} + \frac{1}{2}(l - 1)\tau_{\text{fp}} + (l - 1)\tau_{\text{fp}}.
\]

The first two terms correspond to sampling the unit interval uniformly and the multiplication by the sum of the weights. The third term gives the expected cost of the subtractions performed when a right branch is taken. The last term gives the cost of \( l - 1 \) comparisons. Since complete binary trees are height balanced, the number of levels \( l \) is at most \( \log_2 |S| + 1 \). The expected cost of weighted selection of an element from \( S \) is

\[
\tau_{\text{wsel}}(|S|) = \tau_{\text{rand}} + \left(\frac{3}{2} \log_2 |S| + 1\right)\tau_{\text{fp}}.
\]

If an element is removed from \( S \), the tree representing \( S \) has to be updated. In order to keep the tree’s structure intact, we move the rightmost leaf at depth \( l \) to the position of the element that is removed, as indicated in 3.2. In order to maintain the invariant that the values at every internal node equal the sums of the rate constants of the leaves of its left and right subtrees, internal nodes have to be updated on two root paths. At every internal node of these two paths, one of the values has to be recalculated. The sum of the weights of the leaves of the right subtree is necessary to be able to do the calculation at every node in \( \tau_{\text{fp}} \) time. The cost of deletion of an element from a set with \( N \) elements is

\[
\tau_{\text{del}}(N) = 2 \log_2(N) \tau_{\text{fp}}.
\]

This assumes that we can access the rightmost leaf at maximum depth in negligible time, something that can be accomplished by maintaining a reference to it. The cost of maintaining this reference is negligible in comparison with the cost of moving a leaf.
Figure 3.2: Example of deletion of a leaf from a tree, keeping its structure intact.

Inserting an element into $S$ involves creating a tree leaf and updating internal nodes’ values of one root path. The cost of this is $\tau_{int}(N) = \log_2(N) \tau_{fp}$ if $S$ has $N$ elements.

The use of complete binary trees facilitates fast selection of reactions with the correct probabilities. It has another advantage. Because of the structure of complete binary trees, it is possible to represent them in consecutive memory locations (by means of arrays), with the structure of the tree implicit in the locations of the nodes. A tree represented by an array is called an implicit tree.

The fact that complete binary tree representations of sets allow weighted selection of elements in logarithmic time has been known for decades (Knuth, 1969). However, it seems that this has not been appreciated by researchers in the field of simulation of surface processes. The claim that computer science has something to offer to other disciplines, such as the natural sciences, is substantiated by articles such as (Maksym, 1988) and (Blue et al., 1995). In the former of these a method is introduced for performing weighted selection from a set in time proportional to the square root of the size of the set, by means of a method known as ‘binning.’ The latter is the first paper on simulation of surface processes proposing the use of binary trees.

Matias et al. (1993) describe a representation for weighted sets that allows the three operations to be performed in constant expected time. This representation is rather complicated and it has two disadvantages. The first is that although constant, the cost of the operations appears to be too high to compete with the tree representation we described if the represented set contains fewer than a million elements. The second is that a large lookup table is used in order to attain the expected constant cost. This lookup table is a number of times as large as the set to be represented.¹

The URsel and WTselULsel algorithms contain uniform selections of elements from sets. It is not necessary to maintain weights in this case. The set may be represented by an array. Selecting an element uniformly from an array amounts to sampling the unit interval uniformly, scaling the sample to the right range, $[1..|S|]$ for set $S$, and indexing the array.

¹I estimate it has at least $1.3 \cdot 10^7$ entries for a set with one million elements.
3.4 Complexity analysis

We assume that the cost of indexing a large array is comparable to the cost of a floating-point operation. Thus, the cost of uniform selection from a set is

\[ \tau_{\text{sel}} = \tau_{\text{rand}} + 2\tau_{\text{fp}}. \]

Insertion of an element in an unweighted set \( S \) and deletion of an element from it are done in the same way as in the array representation of weighted sets. Both have cost \( \tau_{\text{fp}} \).

Next, we investigate the cost of adding newly enabled reactions to the set \( E \) and removing disabled reactions from it, respectively. In order to calculate which reactions are enabled if a reaction is performed, each modified site has to be compared with each site in the pattern of each reaction type. If there is a match, then the other sites of the pattern have to be compared with the state of the lattice. Altogether there are \( n_{\text{mod}} n_{\text{src}} \) pattern matches. Denoting the cost of one pattern match by \( \tau_{\text{pm}} \), the cost of the enabling check is \( n_{\text{mod}} n_{\text{src}} \tau_{\text{pm}} \). The cost of adding \( n \) reactions to \( E \) is \( n \tau_{\text{enab}}(|E|) \).

Checking which reactions become disabled can be done in a similar fashion. The only difference is that the patterns overlapping modified sites have to be compared against the lattice state before the update. If they match before the update, they do not match after the update, but have been disabled. The cost of the disabling check is \( \tau_{\text{enab}} \) as well. Having calculated the sets of enabled and disabled reactions, the set \( E \) has to be updated, for most of the algorithms. Deleting disabled reactions is an expensive operation: the \( n \) disabled reactions have to be found in the tree representing \( E \). One of these disabled reactions is the reaction that has just been performed. This one is easily found. Finding the other \( n - 1 \) gives rise to an expected \( \frac{1}{2}(n - 1)|E| \) reaction comparisons. A reaction comparison consists of a reaction type and location comparison. We assume that this can be done in \( \tau_{\text{fp}} \). Including the time of deleting \( n \) reactions, we see that the total time spent on dealing with disabled reactions is

\[ n_{\text{mod}} n_{\text{src}} \tau_{\text{pm}} + \frac{1}{2}(n - 1)|E| \tau_{\text{fp}} + n \tau_{\text{del}}(|E|) \]

if this approach is followed. An alternative is to traverse \( E \), and check for every element whether it is still enabled. Then \(|E|\) pattern matches are performed and \( n \) reactions are deleted, the total cost of which is

\[ |E| \tau_{\text{pm}} + n \tau_{\text{del}}(|E|) \]

It is hard to say which is the more efficient way of dealing with disabled reactions without more knowledge about the values of \( \tau_{\text{fp}}, \tau_{\text{pm}}, n_{\text{mod}}, \) and \( n_{\text{src}} \). More important than this is the fact that the complexity of both is \( \mathcal{O}(|E|) \). Rather arbitrarily we choose the second way of dealing with disabled reactions.
<table>
<thead>
<tr>
<th>Basic operation</th>
<th>Cost Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sampling $U(0, 1)$</td>
<td>$\tau_{\text{rand}}$</td>
</tr>
<tr>
<td>Floating-point operation</td>
<td>$\tau_{\text{fp}}$</td>
</tr>
<tr>
<td>Matching reaction pattern</td>
<td>$\tau_{\text{pm}}$</td>
</tr>
<tr>
<td>Incrementing time</td>
<td>$\tau_{\text{time}} = \tau_{\text{rand}} + 3\tau_{\text{fp}}$</td>
</tr>
</tbody>
</table>
| Weighted selection from $S$                         | $\tau_{wset}(|S|) = \begin{cases} 
\tau_{\text{rand}} + \left(\frac{3}{2}\log_2 |S| + 1\right)\tau_{\text{fp}} & \text{tree} \\
\tau_{\text{rand}} + \left(\frac{3}{2}|S| + 1\right)\tau_{\text{fp}} & \text{list} 
\end{cases}$ |
| Insertion in weighted set $S$                       | $\tau_{\text{ins}}(|S|) = \begin{cases} 
\log_2 |S|\tau_{\text{fp}} & \text{tree} \\
\tau_{\text{fp}} & \text{list} 
\end{cases}$ |
| Deletion from weighted set $S$                      | $\tau_{\text{del}}(|S|) = \begin{cases} 
2\log_2 |S|\tau_{\text{fp}} & \text{tree} \\
\tau_{\text{fp}} & \text{list} 
\end{cases}$ |
| Uniform selection from $S$                          | $\tau_{\text{usel}} = \tau_{\text{rand}} + 2\tau_{\text{fp}}$ |
| Insertion in unweighted set $S$                     | $\tau_{\text{ins}} = \tau_{\text{fp}}$ |
| Deletion from unweighted set $S$                    | $\tau_{\text{del}} = \tau_{\text{fp}}$ |
| Updating state                                      | $\tau_{\text{react}} = \tau_{\text{pm}}$ |
| Checking for enabled reactions                     | $n_{\text{mod}}n_{\text{src}}\tau_{\text{pm}}$ |

Table 3.1: Overview of the cost of basic operations

The occurrence of a reaction is reflected by a state change. The cost of updating $\sigma$ is assumed to be equal to $\tau_{\text{pm}}$. Actually, it is somewhat smaller than $\tau_{\text{pm}}$ if not all particles in reaction patterns change. For the sake of simplicity we ignore this.

Table 3.1 gives an overview of the cost of the operations discussed in this section.

In the comparison of the performance of simulation algorithms it would help if we knew which of $\tau_{\text{rand}}, \tau_{\text{fp}}$, and $\tau_{\text{pm}}$ is the largest, which the smallest, and how large the differences are. This is hard to say in general. The cost of generating random numbers is highly dependent on the random number generator used. $\tau_{\text{rand}}$ may be as small as $3\tau_{\text{fp}}$, but it may also be as large as $100\tau_{\text{fp}}$. Similarly, the cost of a pattern match may vary greatly for different cache sizes. The rationale behind the introduction of the three parameters is that they allow the cost of operations that depend in similar ways on hardware or auxiliary algorithms to be expressed by one parameter.

The memory requirements of the simulation algorithms are determined mainly by the memory used for storing the lattice and set of enabled reactions $E$; we ignore storage used by the reaction types. One lattice site requires $\left\lceil \frac{\log_2 |L|}{8} \right\rceil$ bytes of storage. This amount of storage is denoted by $m_{\text{site}}$. The amount of storage needed for a lattice configuration is $|\mathcal{L}|m_{\text{site}} = L_0L_1L_2m_{\text{site}}$ bytes.

The amount of memory used for storing $E$ differs from algorithm to algorithm. Depending
3.4 Complexity analysis

on the operations implemented and the way in which $E$ is structured, more or less storage is needed. The assumptions that we make about storage requirements of $E$ are the following.

- We assume 4-byte memory addresses.
- We assume that $L_0L_1L_2 < 2^{32}$, and that lattice indexes are 4 bytes large.
- We assume that (cumulative) rate constants are stored as double precision floating-point numbers, taking 8 bytes of storage.
- We assume that time stamps, as used in the WRsel-a algorithm are 32-bit integers.

3.4.3 Complexity of the algorithms

We analyze the complexity of the algorithms introduced in this chapter, in terms of the parameters $\tau_{fp}$, $\tau_{rand}$, and $\tau_{pm}$ with respect to the computational cost per reaction performed. We also give expressions for the amount of memory used by the algorithms.

The basic algorithm, WRsel, is easy to analyze using the results of the previous section. One iteration of the algorithm consists of selecting a time increment and incrementing the time, selecting a reaction, updating the state and updating $E$. This means that we have

$$
\tau_{WRsel} = \tau_{time} + \tau_{wsel}(|E(\sigma)|) + \tau_{react} + \tau_{enab} + \tau_{disab}.
$$

Expressions for $\tau_{time}$, $\tau_{wsel}$ and $\tau_{react}$ can be found in Table 3.1. Since $E$ is unstructured and we use a tree representation, we have

$$
\tau_{enab} = n_{mod}n_{src}\tau_{pm} + n \log_2 |E(\sigma)|\tau_{fp},
$$

and

$$
\tau_{disab} = |E|\tau_{pm} + 2n \log_2 |E(\sigma)|\tau_{fp}.
$$

The total cost per reaction for WRsel is given by

$$
\tau_{WRsel} = 2\tau_{rand} + (|E(\sigma)| + n_{mod}n_{src} + 1)\tau_{pm} + (3(n + \frac{1}{2}) \log_2 |E(\sigma)| + 4)\tau_{fp}.
$$

(3.5)

The amount of memory used to store the lattice is $L_0L_1L_2m_{site}$. Enabled reactions are identified by their type and a unit cell. A reference to a reaction type takes 4 bytes of memory. Unit cells can also be identified by using 4 bytes of memory. At the internal nodes of the tree representing $E$, two cumulative rate constants are stored as explained in
Section 3.4.2, each taking 8 bytes of memory. If we allocate 16 bytes for the leaves as well, we can use an implicit tree. The total amount of memory used by the tree is some \(32|E|\) bytes. Altogether, the memory use of WRsel is given by (3.6).

\[
m_{\text{WRsel}} = L_0 L_1 L_2 m_{\text{slice}} + 32|E(\sigma)|
\]  

(3.6)

Next we consider WRsel-o. In this algorithm it may take a number of iterations before a reaction is performed. In state \(\sigma\) and with \(R = (\Sigma e, sr : (e, sr) \in E : sr)\), the probability of a trial being successful is \(\frac{k(E(\sigma))}{k(E(\sigma))}\). The expected number of iterations until a successful trial is the inverse, \(\frac{R}{k(E(\sigma))}\). In every unsuccessful iteration the time is incremented, a reaction is selected, and it is rejected. In a successful iteration, the operations performed in an unsuccessful one are performed, and the state is updated. Furthermore, disabled reactions are removed and newly enabled reactions are added to \(E\). This means that the expected cost of a reaction is

\[
\tau_{\text{WRsel-o}} = \frac{R}{k(E(\sigma))} (\tau_{\text{time}} + \tau_{\text{wsel}}(|E(\sigma)|) + \tau_{\text{acc}}) + \tau_{\text{react}} + \tau_{\text{enab}} + \tau_{\text{disab}}.
\]

The decision to accept or reject requires sampling the unit interval uniformly, and a comparison of two floating-point numbers, provided that the quotients \(\frac{k(e)}{sr}\) are stored instead of the sampling rates. This means that

\[
\tau_{\text{acc}} = \tau_{\text{rand}} + \tau_{\text{fp}}.
\]

We assume the cost of deciding on the sampling rate of the newly enabled reactions to be negligible. The cost of dealing with newly enabled reactions is slightly higher than for WRsel, because the quotients of rate constants and sampling rates are computed. The contributions of the other operations are the same as for WRsel, so the expected cost of a reaction is given by

\[
\tau_{\text{WRsel-o}} = 3 \frac{R}{k(E(\sigma))} \tau_{\text{rand}} + (|E(\sigma)| + n_{\text{mod}}n_{\text{src}} + 1) \tau_{\text{pm}}
\]

\[
+ ((\frac{1}{2}\log_2 |E(\sigma)| + 5) \frac{R}{k(E(\sigma))} + n(\log_2 |E(\sigma)| + 1)) \tau_{\text{fp}}.
\]  

(3.7)

For each reaction in \(E\), the quotient of the rate constant and the sampling rate has to be stored, requiring an extra four bytes of storage per reaction, for a total of 12 bytes per reaction. Having 16 bytes at our disposal in the implicit tree, the actual amount of memory used for storing the enabled reactions is the same as for WRsel.

The complexity of URsel is more interesting than that of WRsel-o: URsel does not use weighted selections but uniform selection of reactions. As a result, the selection of reactions from \(E\) is cheaper. The expected number of iterations until a reaction is performed
is $|E| k_{\text{max}}(\sigma) \over \bar{k}(\bar{E}(\sigma))$, with $k_{\text{max}}(\sigma)$ denoting the maximum of the rate constants of the reactions enabled in state $\sigma$. The time per reaction is

$$\tau_{\text{URsel}} = \frac{|E| k_{\text{max}}(\sigma)}{\bar{k}(\bar{E}(\sigma))} \left( \tau_{\text{time}} + \tau_{\text{usrsel}} + \tau_{\text{acc}} + \tau_{\text{react}} + \tau_{\text{enab}} + \tau_{\text{disab}} \right).$$

The sampling rate of all enabled reactions is the same, $k_{\text{max}}(\sigma)$. As this depends on $\sigma$, the computation to decide whether a reaction $e$ is accepted requires a random number in the range $[0, k_{\text{max}}(\sigma))$ instead of on the unit interval. Hence,

$$\tau_{\text{acc}} = \tau_{\text{rand}} + 2 \tau_{\text{fp}}.$$

The cost of dealing with disabled and newly enabled reactions is lower for URsel than for WRsel-o, because no weights are used:

$$\tau_{\text{disab}} = |E| \tau_{\text{pm}} + n \tau_{\text{fp}},$$

and

$$\tau_{\text{enab}} = n_{\text{mod}} n_{\text{src}} \tau_{\text{pm}} + n \tau_{\text{fp}}.$$

The expected total cost per reaction is therefore

$$\tau_{\text{URsel}} = \frac{3 |E(\sigma)| k_{\text{max}}(\sigma)}{\bar{k}(\bar{E}(\sigma))} \tau_{\text{rand}} + (|E(\sigma)| + n_{\text{mod}} n_{\text{src}} + 1) \tau_{\text{pm}} + (6 |E(\sigma)| k_{\text{max}}(\sigma) \over \bar{k}(\bar{E}(\sigma)) + 2n) \tau_{\text{fp}}. \tag{3.8}$$

Comparing Equations (3.5) and (3.8) we see that the coefficients of the $\tau_{\text{pm}}$ terms are the same: oversampling has no effect on the amount of pattern matching per reaction performed. The number of random numbers generated per reaction performed is higher for URsel than for WRsel. The coefficients of the $\tau_{\text{fp}}$ terms are less easily compared. If the quotient $|E(\sigma)| k_{\text{max}}(\sigma) \over \bar{k}(\bar{E}(\sigma))$ is small, then fewer floating-point operations are performed in the URsel algorithm. If the quotient is large, then the reverse will be the case. There are too many unknowns in the formulas to give a meaningful expression for the crossover point.

The memory use of URsel is more favorable than that of WRsel as the set $E$ is represented more efficiently. Each element of $E$ still takes eight bytes of storage, but there is no overhead resulting from the internal tree nodes anymore. We get

$$m_{\text{URsel}} = L_0 L_1 L_2 m_{\text{site}} + 8 \ |E(\sigma)| . \tag{3.9}$$

The analysis of WRsel-a is more involved: $E$ contains more than just the enabled reactions, and we have to analyze how much larger it is. Before investigating this, we analyze the cost per simulated reaction. As in WRsel-o it may take multiple iterations of the algorithm before a reaction is performed. If $|E(\sigma)|$ is constant on average, while performing a reaction
leads to \( n \) reactions being enabled, it should on average take \( n \) iterations of the WRsel-a algorithm until a reaction eligible for execution is selected. In all of these iterations, a time increment is generated and the current time is updated, a reaction is selected and removed from \( E \), and it is checked whether this reaction is eligible for execution. If an eligible reaction has been selected, the state and site modification times are updated and any newly enabled reactions are added to \( E \). This means that \( \tau_{\text{WRsel-a}} \) can be broken down as follows:

\[
\tau_{\text{WRsel-a}} = n(\tau_{\text{time}} + \tau_{\text{wsel}}(|E|) + \tau_{\text{del}}(|E|) + \tau_{\text{acc}}) + \tau_{\text{react}} + \tau_{\text{enab}}.
\]

Checking whether a reaction is eligible for execution requires the same amount of work as matching its source pattern against the lattice state. The only difference between the two is that times are compared instead of particles. Thus, the time spent on determining whether a selected reaction may be performed is \( \tau_{\text{acc}} = \tau_{\text{pm}} \). Since both \( \sigma \) and \( \text{lm} \) are updated if a reaction is performed, the contribution of \( \tau_{\text{react}} \) to the time per reaction is \( 2\tau_{\text{pm}} \). The total cost per simulated reaction is

\[
\tau_{\text{WRsel-a}} = 2n\tau_{\text{rand}} + (n_{\text{mod}} n_{\text{src}} + n + 2)\tau_{\text{pm}} + \left(\frac{3}{2} n \log_2 |E| + 4n\right)\tau_{\text{fp}}.
\]

(3.10)

This formula shows that the effect of allowing the set \( E \) to be 'polluted' with ineligible reactions is that the number of pattern matches per reaction performed decreases, at the cost of an increase in the number of random numbers to be generated and the number of floating-point operations performed.

The memory use of WRsel-a is higher than that of the algorithms we have seen so far. This is caused by three factors: the enabling time of each reaction is stored, the last modification time of each lattice site is stored, and \( E \) may contain more reactions than \( E(\sigma) \). Site modification times and enabling times can be stored in four bytes, so a reaction takes twelve bytes of storage and the lattice \( S L_0 L_1 L_2 \). The only unknown part is the size of \( E \). We know that, by assumption, the probability of selecting an eligible reaction is \( \frac{1}{n} \). This can be related to the size of \( E \) by observing that the probability of a trial being successful is also equal to \( \frac{k(E(\sigma))}{k(E)} \), so

\[
\frac{\sum e : e \in E(\sigma) : k(e)}{\sum e, t_{\text{en}} : (e, t_{\text{en}}) \in E : k(e)} = \frac{1}{n}.
\]

(3.11)

Unfortunately it is not possible to give a closed expression for \( |E| \) in terms of \( n \) and \( |E(\sigma)| \). Equation (3.11) does show, however, that \( E \) may contain \( n \) times as many elements as \( E(\sigma) \). With each reaction in \( E \), an enabling time is stored, besides the usual enabling location and reference to the reaction type. The enabling time is the iteration sequence number and can be stored in four bytes. The amount of memory needed per reaction is twelve bytes. As before, internal nodes of the tree representing \( E \) take 16 bytes of memory. Thus, using an implicit tree, the amount of memory used for \( E \) is still \( 32|E| \). Sites require more storage than before: besides the particle present, the time of the last modification is stored. This
3.4 Complexity analysis

takes four bytes of memory. We get the following expression for the memory use of WRsel-a:

\[ m_{\text{WRsel-a}} = L_0 L_1 L_2 (m_{\text{site}} + 4) + 32|E| \cdot \] (3.12)

The WRsel-h algorithm is again somewhat easier to analyze: as in WRsel, a reaction is performed in every iteration of the algorithm. The only difference between the WRsel-h and WRsel algorithms is the way in which enabled reactions are selected. In the WRsel-h algorithm \( E \) is a set of sets. This set can still be represented by a binary tree, but now each leaf has to represent a set of reactions instead of a reaction. Naturally, such a set of reactions can be represented by a tree with the reactions at the leaves. So we have, in fact, a binary tree with the reactions at the leaves, just as in WRsel. The difference is that in the case of WRsel-h the tree may not be balanced, and hence its height may exceed \( \log_2 |E| + 1 \). As a result, the cost of selecting a reaction from \( E \) is higher than it is for WRsel. Furthermore, we cannot use an implicit tree for \( E \) anymore, but have to use a pointer implementation, at the cost of memory overhead for storing the pointers.

For suitably chosen groupings of the reactions the disadvantages of WRsel-h vanish and, moreover, turn into advantages. Consider the WTselULsel algorithm. In this algorithm it is not necessary to represent the groups of reactions by trees, as selection from those groups is uniform. The reactions in a group are all of the same type. For each group we need the sum of the rate constants of the enabled reactions, the number of enabled reactions in the group, a reference to the reaction type and a reference to an array of unit cells at which the reactions are enabled. The representation of \( E \) consists of a binary tree with \( |\mathcal{R}T| \) leaves, each of which requires 20 bytes of storage. Each internal node contains two floating-point numbers representing cumulative rate constants, requiring 16 bytes of storage. Allocating the same amount of memory for every tree node, 20 bytes, the total amount of memory used for the tree is \( 40|\mathcal{R}T| \) bytes. The locations of the enabled reactions require 4 bytes of storage each, so the total amount of memory used is

\[ m_{\text{WTselULsel}} = L_0 L_1 L_2 m_{\text{site}} + 40|\mathcal{R}T| + 4|E(\sigma)| \cdot \] (3.13)

Usually the number of enabled reactions is orders of magnitude larger than the number of reaction types, making the last term dominate the second.

Selection of a reaction in the WTselULsel algorithm involves a weighted selection from a set of size \( |\mathcal{R}T| \) and a uniform selection from a set of reactions of some type. This means that the time per reaction can be written as

\[ \tau_{\text{WTselULsel}} = \tau_{\text{time}} + \tau_{\text{wset}}(|E(\sigma)|) + \tau_{\text{std}} + \tau_{\text{tran}} + \tau_{\text{enab}} \cdot \]

Deleting a reaction from \( E \) is somewhat cheaper for WTselULsel than for WRsel because only one root path of the tree needs to be adapted, viz. the one from the leaf involved.
Moreover, this path only has length \( \log_2 |RT| \). The cost of updating the array in which the reaction is stored, is extra, but is only \( \tau_{fp} \). We find

\[
\tau_{disab} = |E(\sigma)| \tau_{pm} + n(\log_2 |RT| \tau_{fp} + \tau_{fp}).
\]

The cost of adding a newly enabled reaction is the same as that of removing a reaction, and therefore

\[
\tau_{enab} = n_{modn_{src}} \tau_{pm} + n(\log_2 |RT| \tau_{fp} + \tau_{fp}).
\]

Combining these results we obtain the total cost per reaction:

\[
\begin{align*}
\tau_{WTselULsel} &= 3\tau_{rand} + (|E(\sigma)| + n_{mod}n_{src} + 1) \tau_{pm} \\
&\quad + ((2n + \frac{3}{2}) \log_2 |RT| + 2n + 6) \tau_{fp}.
\end{align*}
\]

(3.14)

When we compare this expression for the cost of a reaction in the WTselULsel algorithm to that for the cost of a reaction in the WRsel algorithm as given by Equation (3.5), we see that the amount of pattern matching is not influenced by the hierarchical selection of reactions. However, at the cost of generating one extra random number, the number of floating-point operations is reduced significantly. The \( \Theta(n \log_2 |E(\sigma)|) \) term of WRsel is replaced by a \( \Theta(n \log_2 |RT|) \) term. Moreover, the constant of proportionality is smaller too. This is reduction in the number of floating-point operations per reaction is indeed significant: for WTselULsel, the number of floating-point operations per reaction is independent of the system size.

As mentioned in Section 3.3.3, it is possible not to store the enabled reactions at all, but search the lattice for enabled reactions of a given type. In this case we need to be able to select a reaction type with a probability proportional to the sum of the rate constants of the enabled instances. A tree can be used to perform this selection. It differs from the tree used in the WTselULsel algorithm in that no references to the enabled reactions are stored in the leaves. Hence, the memory requirements of WTselULsearch are given by

\[
m_{WTselULsearch} = L_0 L_1 L_2 m_{site} + 32 |RT|.
\]

(3.15)

There are various ways to search the lattice for an enabled reaction of a given type. The easiest is to repeatedly try randomly chosen locations and check whether a reaction of the right type is enabled. The expected number of attempts is \( \frac{L_0 L_1}{m} \) if the reaction type is enabled at \( m \) locations. In state \( \sigma \), the expected number of attempts to find an enabled reaction is

\[
\left( \sum_{RT \in RT} \frac{m_{RT}(E(\sigma))k(RT)}{k(E(\sigma))} \cdot \frac{L_0 L_1}{m_{RT}(E(\sigma))} \right) = \frac{L_0 L_1 k(\mathcal{RT})}{k(E(\sigma))}.
\]
where $m_{RT}(E(\sigma))$ is the number of reactions of type $RT$ in $E(\sigma)$. In each of these attempts a random number is generated, scaled to the right range for the uniform selection, and the lattice is inspected to check whether the reaction is enabled. Hence we have

\[
\tau_{WselULsearch} = \tau_{time} + \tau_{wsel}(|RT|) + \frac{L_0L_1k(RT)}{k(E(\sigma))} (\tau_{use}\tau + \tau_{acc}) + \tau_{react} + \tau_{disab} + \tau_{enab},
\]

\[
\tau_{use} = \tau_{rand} + \tau_{fp},
\]

and

\[
\tau_{acc} = \tau_{pm}.
\]

Updating the tree, so that it represents the number of enabled reactions of every type in the new state, amounts to checking which reactions have become disabled, which have become enabled, and updating at most $2n$ root paths in the tree:

\[
\tau_{disab} = n_{mod\Delta src} \tau_{pm} + n \log_2 |RT|,
\]

and

\[
\tau_{enab} = n_{mod\Delta src} \tau_{pm} + n \log_2 |RT|.
\]

Adding the cost of incrementing the time and updating the state, we obtain the following expression for the cost of a reaction:

\[
\tau_{WselULsearch} = \left(\frac{L_0L_1k(RT)}{k(E(\sigma))} + 2\right)\tau_{rand} + \left(2n_{mod\Delta src} + \frac{L_0L_1}{k(E(\sigma))} + 1\right)\tau_{pm} + ((2n + \frac{3}{2}) \log_2 |RT| + \frac{L_0L_1}{k(E(\sigma))} + 4)\tau_{fp}.
\] (3.16)

The WTselULsearch algorithm, like WTselULsel, has the property that the number of floating-point operations per reaction is independent of the system size — in the term $\frac{L_0L_1}{k(E(\sigma))}$ both the numerator and the denominator depend linearly on the number of lattice sites, making the quotient independent of the system size. In fact, for the WTselULsearch algorithm, the time per reaction is independent of the system size. So, in terms of growth behavior, it is the best algorithm we have seen so far. The constants of proportionality, in particular the term $\frac{L_0L_1k(RT)}{k(E(\sigma))}$, may be so large that other algorithms perform better for system sizes feasible in practice.

We have one algorithm left to analyze, WLselWTsel. In this algorithm reactions are grouped according to their enabling location. Selection of a reaction consists of two successive weighted selections: a location followed by a reaction type. At most $|RT|$ reactions are enabled at a site. The expected number of enabled reactions per site depends on the system. We settle for taking the maximum, obtaining

\[
\tau_{WLselWTsel} = \tau_{time} + \tau_{wsel}(L_0L_1) + \tau_{wsel}(|RT|) + \tau_{react} + \tau_{disab} + \tau_{enab}
\]
as a bound on the expected time per reaction. The cost of the weighted selection of a location is \( \tau_{\text{rand}} + (\frac{3}{4} \log_2(L_0L_1) + 1) \tau_{\text{fp}} \). The cost of the weighted selection of a reaction from a group is at most \( \tau_{\text{rand}} + (\frac{3}{4} \log_2 |RT| + 1) \tau_{\text{fp}} \). Usually, only a small fraction of all reactions is enabled per location, so this bound is not very good. In order to reduce the cost of deleting disabled reactions from \( E \), we introduce a two-dimensional array of size \( L_0 \times L_1 \). Each element \((l_0, l_1)\) contains a reference to the group of reactions enabled at unit cell \((l_0, l_1)\). By doing so, the worst case cost of finding a disabled reaction in \( E \) is \((1 + |RT|) \tau_{\text{fp}}\): the array is indexed, and at most \( RT \) reactions are inspected before the right one is found. Deleting a reaction amounts to moving the rightmost leaf the group’s tree, changing the values on two root paths in the group’s tree, and changing the values on one root path in the tree of groups. The cost of dealing with disabled reactions is

\[
\tau_{\text{disab}} = n_{\text{mod}} n_{\text{src}} \tau_{\text{pm}} + n(1 + |RT|) \tau_{\text{fp}} + n(2 \log_2 |RT| + \log_2(L_0L_1)) \tau_{\text{fp}}
\]

Adding a reaction is slightly cheaper because in the group’s tree only one root path needs to be updated. The cost of updating time and state and that of checking which reactions become disabled or enabled is the same as for the previous algorithm. Thus, we get

\[
\tau_{\text{WLselWTsel}} = 3 \tau_{\text{rand}} + (2n_{\text{mod}} n_{\text{src}} + 1) \tau_{\text{pm}} \\
+ ((3n + \frac{3}{2}) \log_2 |RT|) \tau_{\text{fp}} \\
+ ((2n + \frac{3}{2}) \log_2(L_0L_1) + n(|RT| + 1) + 5) \tau_{\text{fp}}
\]

(3.17)

for the cost of one reaction. Recall that this is not an exact expression for the expected cost of a reaction, as it assumes that in every unit cell \(|RT|\) reactions are enabled. The most noteworthy property of the time per reaction for the WLselWTsel algorithm is that the number of pattern matches per reaction is independent of the system size. It shares this property with the WRsel-a algorithm.

The memory use of WLselWTsel is higher than that of the other instantiations of WRsel-a. The lattice takes the same amount of memory as before, \( L_0L_1L_2m_{\text{site}} \) bytes. The tree of reaction groups has \( L_0L_1 \) leaves. Each leaf contains an identification of the unit cell associated with that leaf (four bytes), a reference to the reactions enabled at that unit cell (four bytes), and the sum of their rate constants (eight bytes). Every internal node contains two cumulative rate constants, using 16 bytes. Hence the tree of reaction groups requires \( 32L_0L_1 \) bytes of storage. The reactions in one group are represented by a tree as well. The internal nodes contain two cumulative rate constants each and the leaves contain references to reaction types. The amount of memory used for the trees of the reaction groups is bounded by \( 32|E(\sigma)| \). Finally, there is the array of references to reaction groups. This takes \( 4L_0L_1 \) bytes of memory. The total memory requirements of the algorithm are given by

\[
m_{\text{WLselWTsel}} = 36L_0L_1 + L_0L_1L_2m_{\text{site}} + 32|E(\sigma)|
\]

(3.18)
3.4 Complexity analysis

3.4.4 Discussion

Having analyzed the time per reaction and memory use of the algorithms, we can compare their merits and drawbacks. The insight gained helps in deciding which combinations of techniques can lead to better algorithms. The results of the previous section are summarized in Table 3.2.

\[
\tau_{\text{WRsel}} = 2\tau_{\text{rand}} + (|E(\sigma)| + n_{\text{mod}}n_{\text{src}} + 1)\tau_{\text{pm}} \\
+ (3(n + \frac{1}{2}) \log_2 |E(\sigma)| + 4)\tau_{\text{fp}} \\
\tau_{\text{URsel}} = 3\frac{|E(\sigma)|k_{\text{max}}(\sigma)}{k(E(\sigma))}\tau_{\text{rand}} \\
+ (|E(\sigma)| + n_{\text{mod}}n_{\text{src}} + 1)\tau_{\text{pm}} + (6\frac{|E(\sigma)|k_{\text{max}}(\sigma)}{k(E(\sigma))} + 2n)\tau_{\text{fp}} \\
\tau_{\text{WRsel-a}} = 2n\tau_{\text{rand}} + (n_{\text{mod}}n_{\text{src}} + n + 2)\tau_{\text{pm}} + (\frac{9}{2}n \log_2 |E| + 4n)\tau_{\text{fp}} \\
\tau_{\text{WTselULsel}} = 3\tau_{\text{rand}} + (|E(\sigma)| + n_{\text{mod}}n_{\text{src}} + 1)\tau_{\text{pm}} \\
+ (2(n + \frac{3}{2}) \log_2 |\mathcal{R}\mathcal{T}| + 2n + 6)\tau_{\text{fp}} \\
\tau_{\text{WTselULsearch}} = \left(\frac{L_0L_1k(\mathcal{R}\mathcal{T})}{k(E(\sigma))}\right)^2 + 2)\tau_{\text{rand}} + (2n_{\text{mod}}n_{\text{src}} + \frac{L_0L_1}{k(E(\sigma))} + 1)\tau_{\text{pm}} \\
+ (2(n + 1) \log_2 |\mathcal{R}\mathcal{T}| + \frac{L_0L_1}{k(E(\sigma))} + 4)\tau_{\text{fp}} \\
\tau_{\text{WLselWTsel}} = 3\tau_{\text{rand}} + (2n_{\text{mod}}n_{\text{src}} + 1)\tau_{\text{pm}} \\
+ (3(n + \frac{3}{2}) \log_2 |\mathcal{R}\mathcal{T}|)\tau_{\text{fp}} \\
+ (2(n + \frac{3}{2}) \log_2 (L_0L_1) + n(|\mathcal{R}\mathcal{T}| + 1) + 5)\tau_{\text{fp}}
\]

Table 3.2: Time per reaction for six simulation algorithms.

Three algorithms have a term $|E(\sigma)|\tau_{\text{pm}}$: WRsel, URsel, and WTselULsel. This term greatly influences the performance as system sizes grow. URsel exhibits better performance than WRsel if the rate constants of the reactions do not vary over a wide range, and for most systems WTselULsel performs better than WRsel. Still, the presence of the term $|E(\sigma)|\tau_{\text{pm}}$ shows that the improvements are not enough.

The two algorithms WRsel-a and WLselWTsel do not suffer from this problem. They show two entirely different ways of removing the linear growth of the time per reaction with $|E(\sigma)|$. As such, they are promising. This promise comes at a cost: both algorithms have higher memory requirements. The memory use of WRsel-a is dominated by the number of elements of $E$, while that of WLselWTsel is dominated by the size of the lattice. Although the memory use is higher than that of the other algorithms, WRsel-a and WLselWTsel are generally still preferable over those. First, the memory use still grows linearly with the size
of the system. Second, in current day computers, large memories become more and more common. In situations where no large memory is available, WTselULsearch may be the algorithm of choice.

The WTselULsearch algorithm has another property, unique to the algorithms we have seen so far, that is very desirable in a simulation algorithm. The time spent per reaction is independent of the size of the system. The only dependence on the system size is through the factors \( \frac{L \cdot L_1}{k(E(\sigma))} \). But if the system's behavior does not depend on the size, this factor is constant. For systems with few reaction types, WTselULsearch may well be the fastest algorithm. For systems with larger numbers of reaction types, it is more probable that of each type, fewer reactions are enabled. This increases the time per reaction for WTselULsearch.

It would be nice to have an algorithm that has all the advantages of WTselULsearch, WRsel-a, and WLselWTsel combined. More specifically, we would like an algorithm for which the time per reaction does not grow with the system size, that has modest memory requirements, and that performs well for most systems.

The fact that we aim for an algorithm that performs well for a large class of systems, excludes algorithms in which the set of enabled reactions is not represented. Such algorithms cannot perform well for systems in which few reactions are enabled of each type. What about the requirement of the time per reaction not growing with the system size? For WRsel-a, the time per reaction grows logarithmically with the system size, because of the term \( \log_2 |E| \). This term, which comes from the weighted selection from set \( E \), can be reduced by selecting reactions hierarchically: in WTselULsel it has been replaced by a term logarithmic in \( |\mathcal{R}\sigma\mathcal{T}| \). The combination of the techniques that led to these two algorithms is one we have to investigate.

### 3.5 Combining techniques

We continue investigating simulation algorithms, by combining techniques introduced in Sections 3.3.1 through 3.3.3. The analysis of the algorithms we have seen so far indicated that the combination of WRsel-a and WRsel-h may lead to a good simulation algorithm. Therefore we start with combining these two, giving us WRsel-ha. Again there are several possibilities for grouping reactions, leading to a number of algorithms. Next, we add oversampling to WRsel-ha, resulting in an even more general algorithm, WRsel-hao. All algorithms discussed so far are special cases of this one.

#### 3.5.1 Combining WRsel-h and WRsel-a

Starting from WRsel-h, it is easy to see how to obtain WRsel-ha: simply add enabling times to all reactions, omit the deletion of disabled reactions, and add an eligibility check
before performing reactions.

**Algorithm 3.5 (WRsel-ha)**

1. Initialize:
   \[ t := 0; \sigma := \sigma_0; R := (\Sigma e : e \in E(\sigma) : k(e)); \]
   \[ \text{for } l \in \mathcal{L} \rightarrow \text{lm}[l] := 0; \text{ref} \]
   Construct a set \( E \), the elements of which are sets of pairs, each consisting of a reaction and an enabling time, 0: \( E = \{s_0, \ldots, s_{H-1}\} \). We require that every reaction enabled in \( E(\sigma) \) occurs in exactly one of the \( s_i \).

2. Generate a time increment \( w \) by sampling an exponential distribution with parameter \( R \); increment \( t \) by \( w \).

3. Select element \( s \) from \( E \) with probability \( \frac{k(e)}{K} \). Having selected \( s \), select an element \( (e, t_{en}) \) from it with probability \( \frac{k(e)}{k(s)} \). Subtract \( k(e) \) from \( R \). If \( (e, t_{en}) \) is eligible for execution, then update \( \sigma \) and \( \text{lm} \); add newly enabled reactions to \( E \), using the iteration count as the enabling time.

Having already established the correctness of WRsel-h and WRsel-a, it is easily verified that this algorithm is correct. The probability of \( e \) being selected is \( \frac{k(e)}{K} \), as in WRsel-a. The probability of an eligible reaction being selected and the distribution used in generating time increments have not changed either. Therefore, the correctness proof is the same as that of WRsel-a.

Again, there are several ways of grouping reactions. The most obvious one is to group reactions according to their type. The resulting algorithm was first described in the literature by Lukkien et al. (1998). In the paper in question the algorithm is called VSSM-b, version b of the Variable Step Size Method. In our systematic naming it is called WTselULsel-a.

Before investigating other special cases of WRsel-ha, we analyze the complexity of the WTselULsel-a algorithm. The analysis of the time per reaction is to a large extent the same as the analysis for WRsel-a. The only difference is in the cost of operations on \( E \). The cost of selection of a reaction from \( E \), deletion of a reaction from \( E \), and insertion of a reaction in \( E \) are all the same as in WTselULsel. The resulting time per reaction is given by

\[
\tau_{\text{WTselULsel-a}} = 3n\tau_{\text{rand}} + (n_{\text{mod}}n_{\text{src}} + n + 2)\tau_{\text{pm}} + \left(\frac{7}{2}\log_2 |\mathcal{R}| + 8\right)n\tau_{\text{fp}}.
\]

(3.19)

We see in this formula that the time per reaction is independent of the system size for WTselULsel-a. The cost of selection of reactions in two steps has is independent of the system size, and leaving disabled reactions in \( E \) removes the system size dependence of the number of pattern matches per reaction.
The memory use of WTselULSel-a is also found by combining partial results of WTselULSel and WRsel-a. The result is

\[ m_{WTselULSel-a} = L_0 L_1 L_2 (m_{site} + 4) + 8|E| + 40|RT| . \]  

(3.20)

The memory use of WTselULSel-a lies between that of WTselULSel and that of WRsel-a. It uses more memory than WTselULSel because disabled reactions are left in \( E \). As rate constants are stored only with the types, not with every individual reaction in \( E \), the memory requirements of WTselULSel-a are lower than those of WRsel-a.

A second possibility for grouping reactions is by enabling location. The resulting algorithm, WLselWTsel-a, strongly resembles WTselULSel-a. It differs only in the order in which reaction type and location are selected. This difference, however, has an important impact on the time per reaction. Because the location selection is weighted, there is a logarithmic dependency on the system size in the time per reaction. Furthermore, in WLselWTsel-a it is still possible to remove all disabled reactions from \( E \), but it is not necessary. Since WLselWTsel-a does not have any advantages over WTselULSel-a or WLselWTsel, we do not investigate its performance any further.

In Section 3.3.2 we already mentioned that it is possible to enlarge \( E \) to contain even more reactions, so that, in fact, it contains all possible reactions, whether enabled or not. Grouping these reactions by their type, we obtain \( |RT| \) groups, each containing all unit cells. Hence, there is no need to represent these groups anymore. After having selected a reaction type, a location is selected uniformly. If the reaction thus selected is enabled, it can be performed. Note that it is not necessary to maintain lattice modification times or reaction enabling times. These were introduced in WRsel-a only to be able to distinguish multiple instances of a single reaction. In this algorithm these do not occur. This algorithm is akin to WTselULSearch. The difference is that per iteration only one location is tried, instead of the lattice being searched for a location at which the selected type is enabled. As a result, it is not guaranteed that a reaction occurs in every iteration of the algorithm. It may seem that the time per reaction is worse for this algorithm than it is for WTselULSearch. However, it is not necessary to maintain the numbers of enabled reactions of each type anymore, reducing the number of pattern matches, and simplifying the implementation. In order to make a better comparison possible, algorithm 3.6 contains the details of the new method.

**Algorithm 3.6 (WTselULtrial)**

1. Initialize:
   \[ t := 0; \sigma := \sigma_0; \ R := (\Sigma RT : RT \in RT : k(RT)); \]

2. Generate a time increment \( w \) by sampling an exponential distribution with parameter \( L_0 L_1 R \); increment \( t \) by \( w \).
3. Select reaction type $RT$ with probability

$$\frac{L_0L_1k(RT)}{\left(\sum RT': \quad L_0L_1k(RT')\right)} = \frac{k(RT)}{R};$$

select a unit cell uniformly. If a reaction of type $RT$ is enabled at the selected unit cell, then update $\sigma$.

As this algorithm is a special case of WRsel-ha, its correctness follows from the correctness of WRsel-ha. In the correctness argument of WRsel-ha, Property 3.2 is used. For the special case of WTselULtrial, Property 3.1 is already sufficient.

The WTselULtrial algorithm as described here was first presented by Lukkien et al. (1998). It is used by more people, but then adapted to specific systems. For example, the algorithmic description of the ZGB model in (Ziff et al., 1986) is an instance of the WTselULtrial algorithm. Another example of this is found in a paper by Dickman and Burschka (1988) where a single-species adsorption-desorption model is described algorithmically.

In order to calculate the time per reaction for WTselULtrial, we observe that the expected number of iterations for a reaction is $\frac{L_0L_1k(RT)}{k(E(\sigma))}$. As no reactions are stored, we have

$$\tau_{WTselULtrial} = \frac{L_0L_1k(RT)}{k(E(\sigma))}(\tau_{time} + \tau_{wset}(|R'|) + \tau_{use} + \tau_{acc} + \tau_{react}).$$

The cost of the uniform selection of a site is $\tau_{rand} + \tau_{fp}$, as in WTselULsearch. A selected reaction is accepted if it is enabled, so $\tau_{acc} = \tau_{pm}$. Substituting this and expressions for the other terms from Table 3.1 in the above equation, we obtain

$$\tau_{WTselULtrial} = 3\frac{L_0L_1k(RT)}{k(E(\sigma))} \tau_{rand} + \left(\frac{L_0L_1k(RT)}{k(E(\sigma))} + 1\right) \tau_{pm}$$
$$+ \left(\frac{3}{2} \log_2 |R'| + 5\right) \frac{L_0L_1k(RT)}{k(E(\sigma))} \tau_{fp}. \quad (3.21)$$

Since no set of enabled reactions is maintained, the memory use of the algorithm is very low. Storage is required for the lattice and for the tree that is used in selecting reaction types. The following expression gives the amount of memory used:

$$m_{WTselULtrial} = L_0L_1L_2m_{site} + 16 \log_2 |R'|. \quad (3.22)$$

Comparing equations (3.16) and (3.21), we see that fewer pattern matches are performed in the WTselULtrial algorithm, but that more random numbers are used and reactions may involve more floating-point operations. Since the probabilities for reactions to be selected remain constant over the course of a simulation, some optimizations are possible in WTselULtrial.
The first observation leading to an improvement in performance is that the distribution from which time increments are generated is the same one in every iteration. Since the sum of independent random variables with the same exponential distribution has a Gamma distribution, we can reduce the number of random deviates used. If in the entire simulation \( j \) iterations are performed, the total elapsed time can be generated by sampling the Gamma\((j, R)\) distribution (see Appendix A). In many practical cases the stochastic fluctuations in the time increments can be ignored, and taking the average \( \frac{j}{R} \) suffices. If the average is not good enough, techniques like those described by Devroye (1986) can be used to sample the Gamma distribution.

It is instructive to point out the relationship to the so-called Monte Carlo step or MCS one often encounters in the literature. One MCS is defined as \( L_0L_1L_2 \) iterations of the algorithm. It is now clear that the elapsed time for one MCS is distributed according to the Gamma\((L_0L_1L_2, R)\) distribution.

The second observation about WTselULtrial is that the tree used for selecting reaction types does not change. Under these conditions it is possible to perform the selection of a reaction type in constant time, at the cost of extra memory use (Devroye, 1986; Walker, 1974, 1977). After initializing two tables of size \( N \), with \( N \) the smallest power of two at least \( |\mathcal{R}| \), the cost of selection is only \( 2T_{\mathcal{R}} \).

If \( k(E(\sigma)) \) is much smaller than \( k(|\mathcal{R}|) \), then WTselULtrial performs badly. The performance can be improved by grouping certain reaction types. Consider a system in which two reaction types \( RT_0 \) and \( RT_1 \) have the same rate constant, and cannot be enabled at the same time at the same unit cell. These two types can be replaced by one combined type in the simulation. This new type gets the same rate constant as \( RT_0 \) or \( RT_1 \). If it is selected, and a reaction of type \( RT_0 \) is enabled at the selected site, then it is performed. If a reaction of type \( RT_1 \) is enabled, then this is performed. Because these two possibilities exclude each other, any enabled reaction of type \( RT \in \mathcal{R} \) is selected with probability

\[
\frac{1}{L_0L_1} \frac{RT}{\sum_{RT' \neq RT_0, RT_1} : k(RT')}.
\]

The time increment is now generated by sampling the exponential distribution with parameter \( \sum_{RT' \neq RT_0, RT_1} : k(RT') \). It follows that the algorithm is still correct.

The key to the performance improvement over standard WTselULtrial is the higher success probability per trial. The number of pattern matches per trial increases, as one or two patterns are matched if the combined type is selected. Because of the higher success probability, the number of pattern matches per reaction does not increase.

If the types \( RT_0 \) and \( RT_1 \) have different rate constants, they may be combined as well. Suppose that \( k(RT_0) < k(RT_1) \). Then \( RT_0 \) can be replaced by a combined type with rate constant \( k(RT_0) \) while the rate constant of \( RT_1 \) is lowered to \( k(RT_1) - k(RT_0) \).
3.5 Combining techniques

3.5.2 Combining WRsel-ha and WRsel-o

The WRsel-ha algorithm is already a fairly general simulation algorithm. It allows choice in the selection scheme for reactions and in dealing with disabled reactions. It can be generalized even further by adding the possibility of oversampling to it. This can be done at two points in the algorithm. Sampling rates can be associated with individual reactions and with reaction groups. Algorithm 3.7 is the result.

Algorithm 3.7 (WRsel-hao)

1. Initialize:
   \[ t := 0; \sigma := \sigma_0; \]
   \[ \text{for } l \in L \rightarrow lm[l] := 0; \text{ rof} \]
   Construct a set \( E \), the elements of which are pairs, each pair consisting of a set and a sampling rate. Each of the sets itself contains triples of a reaction, a sampling rate, and an enabling time. We require all reactions enabled in \( \sigma_0 \) to be represented exactly once in \( E \), and, conversely, that every reaction in \( E \) is enabled in \( \sigma_0 \). Furthermore, we impose restrictions on the sampling rates. For \( (s, sr) \in E \), we require \( sr \geq (\Sigma e, sr', t_{en} : (e, sr', t_{en}) \in s : sr') \), and for all \( (e, sr', t_{en}) \in s \) we require \( sr' \geq k(e) \).
   \[ R := (\Sigma s, sr : (s, sr) \in E : sr); \]

2. Generate a time increment \( w \) by sampling an exponential distribution with parameter \( R \); increment \( t \) by \( w \).

3. (a) Select an element from \( E \), giving probability \( \frac{sr}{R} \) to \( (s, sr) \). Continue at 3b with probability
   \[ (\Sigma e, sr', t_{en} : (e, sr', t_{en}) \in s : sr') \]
   otherwise the iteration ends.
   
   (b) Having selected \( (s, sr) \), select an element from \( s \), giving probability
   \[ sr' \]
   \[ (\Sigma e', sr'', t_{en'} : (e', sr'', t_{en'}) \in s : sr'') \]
   to \( (e, sr', t_{en}) \). Continue at 3c with probability \( \frac{k(e)}{sr'} \); otherwise the iteration ends.
   
   (c) Remove \( (e, sr', t_{en}) \) from \( s \); subtract \( sr' \) from \( sr \) and from \( R \). If \( s \) has become empty, then remove \( (s, sr) \) from \( E \) and subtract \( sr \) from \( R \). If \( (e, t_{en}) \) is eligible for execution, then change \( \sigma \) and \( lm \) according to the occurrence of \( e \). Add newly enabled reactions to \( E \), also updating \( R \) and any group sampling rates affected. Use the current iteration count as the enabling time for new reactions.
Showing that this algorithm is correct is a matter of combing the ingredients of the correctness arguments of WRsel-ha and WRsel-oo. We do not present all the details of the proof, but focus on one of its ingredients, viz. the probability of an eligible reaction \((e, sr', t_{en})\) being selected and performed. Suppose that \(s\) is the group containing \((e, sr', t_{en})\) and \(sr\) its sampling rate. Then the probability of \(e\) being performed in an iteration in state \(\sigma\) is

\[
\frac{sr}{R} \cdot \left( \frac{sr''}{\sum e', sr'', t_{en} : (e', sr'', t_{en}) \in s : sr''} \right) \cdot \frac{k(e)}{sr'} = \frac{k(e)}{R}.
\]

Summing over all eligible reactions, we see that the probability of a trial in state \(\sigma\) with total sampling rate \(R\) resulting in a reaction being performed is \(\frac{k(E(\sigma))}{R}\). The rest of the proof follows exactly the correctness proof of WRsel-a.

The algorithm as described above is more restrictive than necessary. Removal of ineligible reactions may be left out, but there is nothing against removing some reactions when they become disabled. If no disabled reactions are allowed in \(E\), we obtain a combination of hierarchical selection with oversampling, WRsel-ho. The WRsel-ho algorithm was already described by Binder (1979b). Binder’s description is less concrete than ours, in the sense that he writes about objects in an abstract system and actions these objects may undergo. The objects he describes are our groups of reactions, while the actions, in turn, correspond to reactions.

There are many possible ways of grouping reactions and choosing sampling rates for the groups thus obtained and for the individual reactions. All algorithms shown before WRsel-haoho may be obtained in this way, but the possibilities are not limited to these. We continue by exploring some of the more interesting possibilities, and by briefly discussing how some algorithms found in the literature may be put into our framework.

### 3.5.3 The merits of oversampling

The time per reaction for the WTselULsel-a algorithm is independent of the system size. For this reason, it is one of the most promising simulation algorithms. A disadvantage of WTselULsel-a is the strong dependence of the time per reaction on \(n\), the number of reactions disabled by occurrence of one reaction. In systems that contain lateral interactions this is a severe limitation: in those systems \(n\) is usually very large, as is \(n_{srC}\). Algorithms such as WTselULsearch and WTselULtrial do not perform well either for systems with many or large patterns. The cost of finding an enabled reaction is too high in this case. These performance problems are alleviated by using oversampling. We explain this by considering again the example of Section 2.4.2.

The example of Section 2.4.2 contains lateral interactions. The rate at which particles hop, depends on their environment. Figure 3.3 shows the reaction types for a particle hopping
3.5 Combining techniques

Figure 3.3: Graphical representation of the reaction types for diffusion to the right in the system of Section 2.4.2. The hopping rate constant for a particle surrounded by \( i \) others is \( k_i = v \exp\left(-\frac{E_{\text{act}}^{0} + iE_{\text{int}}}{k_B T}\right) \).

to the right. All patterns are the same in this figure. The only differences between the reaction types shown are their rate constants and the occupation of the sites above, below, and to the left of the central particle. The same holds for reaction types specifying hopping in the other three directions: there are eight types that are very similar, leading to a model with 32 reaction types. Algorithms like WTselULtrial and WTselULsearch do not show satisfactory performance for models with this number of reaction types. Simulation speed using WTselULsel-a is not satisfactory either: occurrence of one reaction disables and enables too many others in models with lateral interaction.

Consider what happens when using WTselULsel-a or WTselULsel for simulating this system. Suppose that a reaction is performed in which a particle moves from an environment in which it has three nearest neighbors to one where it has none, as in Figure 3.4. The particle's source location becomes empty, enabling three jumps: one for each of the particles adjacent to it. These jumps were not already enabled. Furthermore, any jumps of these particles enabled before the reaction took place, are not enabled anymore. To be more pre-
cise, all reactions enabled at the locations adjacent to the moving particle’s source location, are not enabled anymore after the particle has moved away. Different reactions become enabled at these locations. Note that the number of jumps and the direction of the jumps at these locations have not changed, only the rate constants have. Adding these newly enabled reactions and removing the disabled ones is in a sense useless work. Leaving the disabled reactions in $E$ saves some work as $E$ does not have to be searched for them, but $E$’s size increases and extra work associated with handling ineligible reactions has to be done.

The simulation algorithm deals with reaction types. The example suggests, however, that the enabled jumps are the more central concept. Even though there are 32 reaction types, there are really only four types of transformations. A particle may jump to an empty, adjacent site, of which there are at most four. Only the rate constant of the jump depends on the environment. Therefore we investigate a variant of the URsel algorithm in which jumps are stored instead of reactions. The sampling rate for every jump is set to the maximum rate constant, either $v \exp(-E_{\text{act}}^0/k_B T)$ or $v \exp(-(E_{\text{act}}^0 + 3E_{\text{int}})/k_B T)$, depending on the sign of $E_{\text{int}}$. Upon selection of a jump, the lattice is inspected to see which reaction type is enabled at the selected location. Knowing the reaction type, the selected jump can be accepted with the correct probability. If the jump is accepted, $\sigma$ is updated, and it is checked which jumps have become disabled or enabled. Both the number of patterns and their sizes are reduced because jumps are considered. There are only four patterns of two locations each, as opposed to 32 patterns of five locations when reactions are considered. Since disabled jumps have to be removed from $E$, the time per reaction is still proportional to the number of enabled reactions or jumps.

This particular way of oversampling was first described by Gilmer and Bennema (1972). These authors study crystal growth in the presence of surface diffusion. The algorithm used by Gilmer and Bennema can be classified as a ULselWTsel-o algorithm. In their model all reaction types (adsorption, desorption, and diffusion to all nearest neighbors) are enabled at every unit cell. Hence, it is not necessary to keep track of the set of enabled reactions.

A variation of the technique described above can be applied to other algorithms. We start with our most general algorithm so far, WRsel-hao. Reactions are grouped as follows. The
set of reaction types $\mathcal{RT}$ is partitioned into so-called supertypes. Two reaction types belong to the same supertype if their transformations induce the same state change. Formally, we define an equivalence relation $\sim$ on reaction types. To this end we define the effect of a transformation $\mathcal{T}$ by

$$e(\mathcal{T}) = \{s, p, q : (s, p, q) \in \mathcal{T} \land p \neq q : (s, p, q)\}.$$  

Stretching the language, we speak of the effect of a reaction type when referring to the effect of its transformation. Note that the effect of a transformation is itself a valid transformation, so we can speak about effects being enabled or not.

Two reaction types are defined to be equivalent if their effects are equal:

$$(\mathcal{T}, k) \sim (\mathcal{T}' , k') \equiv e(\mathcal{T}) = e(\mathcal{T}') ,$$

for $(\mathcal{T}, k), (\mathcal{T}', k') \in \mathcal{RT}$. The equivalence classes of $\sim$ are the supertypes. As all reaction types in a supertype have the same effect, we can conveniently speak about the effect of a supertype without introducing confusion. The set of supertypes of a systems is denoted by $\Delta\mathcal{RT}$.

In the simulation algorithm, reactions are grouped by supertype instead of by type. For every reaction group, the supertype and the types it comprises are stored, as well as the unit cells at which at least one reaction of the supertype is enabled. After selection of the supertype, a unit cell has to be selected in order to identify a reaction that can be performed. The selection can be done by selecting unit cells with probabilities proportional to the sum of the rate constants of the reactions of the supertype enabled there. It is not necessary to select a specific reaction type from those enabled at the selected unit cell, as these reactions have the same effect.

Still, having to maintain the rate constant of every enabled reaction effect as a sum of rate constants of enabled reactions, we have not gained much in performance. Again oversampling brings a solution. We associate with each supertype a sampling rate. The sampling rate is set to the maximum sum of rate constants of reactions of that supertype enabled at one unit cell at the same time. Now unit cell selection from a reaction group becomes a uniform selection. The actual sum of rate constants of reactions of the selected supertype enabled at the selected unit cell is calculated by inspecting the lattice. This way reactions can be accepted with the correct probabilities. The selection of a supertype, which is the first step in selecting a reaction effect, is weighted. The weight of a reaction group is simply the number of elements it contains times the sampling rate of the supertype.

If we do not remove reactions from $E$ when they become disabled, then we have to be able to decide which reactions in $E$ are eligible. Now we do not have to deal with multiple instances of reactions anymore, but with multiple instances of effects only. If we distinguish eligible effects from ineligible ones, then no ineligible reactions are performed. The
algorithm thus obtained is a variant of WTselULsel-ao. The first step in selecting a reaction is used for selecting a supertype instead of a reaction type. The second step still consists of selection of a unit cell. Oversampling is used to make this a uniform selection. We call the algorithm WSTselULsel-a, short for weighted supertype selection followed by uniform location selection, with approximate representation of $E(\sigma)$. In the algorithm we use some new notation: we write $m_{SRT}(S)$ for the number of reactions of supertype $SRT$ in set of reactions $S$.

Algorithm 3.8 (WSTselULsel-a)

1. Initialize:
   \[ t := 0; \sigma := \sigma_0; \]
   \[ \text{for } l \in \mathcal{L} \rightarrow Im[l] := 0; \text{rof} \]
   Compute the set of supertypes from $\mathcal{RT}$. For each supertype $SRT$ the sampling rate, $k_{SRT}$, by taking the maximum sum of rate constants of reactions of supertype $SRT$ that can be enabled at the same time.
   Construct a set $E$, of which each element contains the locations of the enabled reactions (in $\sigma_0$) of a supertype. Store an enabling time, zero, with every location.
   \[ R := (\Sigma SRT : : m_{SRT}(E(\sigma))k_{SRT}); \]

2. Generate a time increment $w$ by sampling an exponential distribution with parameter $R$. Increment $t$ by $w$.

3. (a) Select a supertype from $E$, supertype $SRT$ with a probability proportional to the product of its sampling rate and the number of elements in the corresponding group of $E$. Having selected $SRT$, select a location, say $l$, in the corresponding group in $E$ uniformly.
    
    (b) If the selected effect $(SRT, l)$ is not eligible, then remove it from $E$ and subtract $k_{SRT}$ from $R$.
    
    (c) If the effect $(SRT, l)$ is eligible for execution, compute its rate constant $k$ in the current state, $\sigma$, by summing the rate constants of the reactions in $SRT$ enabled at $l$. With probability \[ \frac{k}{k_{SRT}} \], remove $(SRT, l)$ from $E$, update $\sigma$ and $Im$, add newly enabled effects to $E$, using the iteration count as the enabling time and update $R$.

In Step 3 of this algorithm, the set $E$ is updated so that it contains all newly enabled reaction effects. This is sufficient to have all enabled reactions accounted for in later iterations of the algorithm.

It is worth noting that a reaction effect may become enabled without a reaction becoming enabled. The reaction effect is added to $E$ anyway. If, later, it is selected, and still no reactions with that effect are enabled, this will be reflected by a zero acceptance probability.
The performance analysis of WSTselULsel-a is rather involved. The number of iterations between successful ones depends both on the number of iterations between consecutive selections of eligible reaction effects and on the acceptance probabilities of eligible reaction effects. The average number of iterations between consecutive selections of eligible reaction effects is not the same as the average number of iterations between consecutive selections of eligible reactions in, for example, WTselULsel-a. On the one hand, the number of reaction types may greatly exceed the number of supertypes, and in the presence of only few supertypes, only few can become disabled and enabled per reaction performed. On the other hand, we have already argued that reaction effects may become enabled when no reactions do.

The conclusion is that the average number of reaction effects enabled and disabled per reaction performed, may either be greater or smaller than \( n \), the average number of reactions enabled and disabled per reaction performed. We introduce a new parameter, \( n^* \), for it in the performance analysis.

First we analyze the expected time per iteration for the WSTselULsel-a algorithm; then we calculate the expected time per reaction performed. In every trial a time increment is generated and the time is incremented, at cost \( \tau_{time} = \tau_{rand} + 3\tau_{fp} \). Next a supertype is selected, with expected cost \( \tau_{wsel}(|\delta RT|) \). Recall that \( \delta RT \) denotes the set of supertypes. For systems with lateral interactions, \( \delta RT \) is much smaller than \( RT \). After having selected a supertype, a location is selected uniformly from a list; the cost of this selection is \( \tau_{use} = \tau_{rand} + 2\tau_{fp} \).

The expected cost of checking the eligibility of the selected reaction effect is bounded from above by \( \tau_{pm} \). Again, the cost is substantially less than this bound for systems with lateral interactions. On average, out of every \( n^* \) iterations, ineligible reaction effects are selected \( n^* - 1 \) times. They are removed from \( E \) at cost \( \tau_{del}(|\delta RT|) + \tau_{fp} = (\log_2 |\delta RT| + 1)\tau_{fp} \).

Once in every \( n^* \) iterations, the selected reaction effect is eligible and an actual rate constant and acceptance probability are computed. The expected cost of this is \( |\delta RT| \cdot \tau_{pm} + \tau_{fp} \), assuming that the actual rate constant is computed by checking which of the source patterns of transformations in the supertype match in the current lattice state.

Accepting or rejecting the selected reaction effect costs \( \tau_{rand} + \tau_{fp} \). If it is accepted, then it is removed from \( E \), \( \sigma \) and \( lm \) are updated, and \( E \) and \( R \) are updated to account for newly enabled reaction effects. Since reaction effects have smaller patterns than reactions, the number of pattern matches performed when checking for newly enabled reaction effects is only \( n^2_{mod}|\delta RT| \cdot (n_{mod}|\delta RT| \cdot m_{SRT}(E(\sigma))k_{SRT}) \).

As the acceptance probability is equal to

\[
p_{acc}(\sigma) = \frac{k(E(\sigma))}{(\Sigma SRT: SRT \in \delta RT)(m_{SRT}(E(\sigma))k_{SRT})},
\]
the expected cost of a trial is bounded from above by

\[
(3 + \frac{1}{n^*})\tau_{\text{rand}} \\
+ (1 + \frac{1}{n^*}(\frac{|\mathcal{RT}|}{|\delta\mathcal{RT}|} + p_{\text{acc}}(\sigma)(n^2_{\text{mod}}|\delta\mathcal{RT}| + 2)))\tau_{\text{pm}} \\
+ ((\frac{5}{2} + \frac{n^* - 1}{n^*}) \log_2 |\delta\mathcal{RT}| + 6)\tau_{\text{fp}} \\
+ \frac{1}{n^*}(2 + p_{\text{acc}}(\sigma)(n^* + 1)(\log_2 |\delta\mathcal{RT}| + 1))\tau_{\text{fp}}.
\]

The expected number of iterations until a successful one is

\[
n^* \cdot \frac{1}{p_{\text{acc}}(\sigma)}
\]

so the expected cost per reaction performed is at most

\[
\tau_{\text{WTselULsel-a}} = \frac{(3n^* + 1)}{p_{\text{acc}}(\sigma)}\tau_{\text{rand}} \\
+ \left(\frac{n^* + |\mathcal{RT}|}{|\delta\mathcal{RT}|} \cdot \frac{1}{p_{\text{acc}}(\sigma)} + n^2_{\text{mod}}|\delta\mathcal{RT}| + 2\right)\tau_{\text{pm}} \\
+ \left((\frac{5}{2}n^* - 1) \log_2 |\delta\mathcal{RT}| + 7n^* + 1\right)\tau_{\text{fp}} \\
+ (n^* + 1)(\log_2 |\delta\mathcal{RT}| + 1)\tau_{\text{fp}}.
\]

Should the set of reaction type be such that no two types are equivalent, then the oversampling disappears. For the time per reaction this has two effects: all occurrences of $\delta\mathcal{RT}$ may be replaced by $\mathcal{RT}$, and the factors $\frac{1}{p_{\text{acc}}(\sigma)}$ disappear. The expression that results is almost the same as that in Equation (3.19). The differences have two causes.

The first difference stems from the fact that in WSTselULsel-a some extra work is performed for computing acceptance probabilities. These computations lead to extra random numbers being used and extra floating-point operations being performed, although acceptance probabilities are either zero or one.

The second difference between WTselULsel-a and this special case of WSTselULsel-a lies in the number of pattern matches performed. In the latter algorithm, only the effects of reaction types are taken into consideration when the lattice is checked for newly enabled reactions. Because of this it is possible that reactions are added to $E$ that have not become enabled at all. If the number of disabled reactions in $E$ is higher, the number of unsuccessful trials increases as well. The enabling check based on reaction effects, however, is
cheaper than that based on reactions, partly compensating this loss of efficiency. It is unclear which is the better solution for models in which no or hardly any reaction types are equivalent.

In systems with lateral interactions, $\delta R_T$ is generally much smaller than $R_T$. Furthermore, in such systems reaction effect patterns are usually much smaller than reaction patterns. The differences become larger as the range of interaction increases. Even for the diffusion system of Section 2.4.2 with its limited lateral interactions, the differences are already considerable. In that system the reaction patterns have size five, while the reaction effect patterns consist of only two sites; the number of reactions types is 32, while there are only four supertypes. We analyze the performance of the WSTselULsel-a algorithm on this system in more detail and compare it to that of WTselULsel-a.

A more detailed comparison of the performance of WTselULsel-a WTselULsel-a requires us to estimate the values of $n^*$ and $n$. This is not easily done for any surface coverage. If the coverage is 50%, and the interaction between nearest neighbors is repulsive, the equilibrium surface configuration is an ordered one, as in Figure 3.5, if the temperature is low. In the case that the surface overlayer has such an ordering, the majority of the reactions that occur are hops from an environment without any nearest neighbor particles to one with three, and back (see also Figure 3.5). The values of $n$ and $n^*$ are determined to a great extent by these kinds of events. Therefore, we can estimate $n$ and $n^*$ by considering the numbers of reactions that become disabled and enabled as a result of a particle hopping from an empty environment to one where it is has three neighbors, and back.

A hop from an empty environment to one with three neighboring particles disables itself and the three hops which the migrating particle could make in the other directions. Furthermore, the environments of the particles surrounding the destination site change, disabling all their enabled reactions. Altogether, 16 reactions are disabled. In the resulting state, the particle that has just moved, can only move back to the site it came from. Each of the three particles surrounding it can hop in three directions. So 10 reactions become enabled. If the particle does hop back to its original position, then these 10 reactions are disabled again, and the 16 that where disabled before, become enabled again. The average number of
reactions disabled for these two hops is 13, as is the average number of reactions enabled.

A particle hopping form an empty environment to one in which it is surrounded by three particles disables seven reaction effects. First of all, the hop itself and the hops the particle could make in the three other reactions are disabled. Second, one hop is disabled for each of its new nearest neighbors. Only one reaction effect becomes enabled, viz. the hop of the particle back to its previous location. If the particle does move back to the site it came from, one reaction effect is disabled and four are enabled. On average, four reaction effects are disabled and four are enabled.

We conclude that \( n \approx 13 \) and \( n^* \approx 4 \) for the system of Section 2.4.2 if the surface coverage is 50% and the temperature is low. The number of modified sites per reaction is two for this model and the sum of the sizes of the reaction patterns is \( 32 \cdot 5 = 160 \). Substitutions of these values for \( n, n^*, n_{\text{mod}}, \) and \( n_{\text{src}} \) into Equations (3.19) and (3.23) yields

\[
\tau_{\text{WTselULsel-a}} \approx 39 \tau_{\text{rand}} + 335 \tau_{\text{pm}} + 331 \tau_{\text{fp}},
\]

and,

\[
\tau_{\text{WTselULsel-a}} \approx \frac{13}{p_{\text{acc}}(\sigma)} \tau_{\text{rand}} + \left( \frac{12}{p_{\text{acc}}(\sigma)} + 18 \right) \tau_{\text{pm}} + \left( \frac{47}{p_{\text{acc}}(\sigma)} + 15 \right) \tau_{\text{fp}}.
\]

A pattern match involves the inspection of five sites in the WTselULsel-a algorithm, and either two or five in the WSTselULsel-a algorithm (a more detailed analysis shows that there are approximately \( \frac{4}{p_{\text{acc}}(\sigma)} + 18 \) pattern matches involving two sites and \( \frac{8}{p_{\text{acc}}(\sigma)} \) involving five sites). Therefore, it is fair to assume the cost of pattern matches is higher than that of generating a random number or performing a floating-point operation. Assuming that \( \tau_{\text{rand}} = \tau_{\text{fp}} = \frac{1}{2} \tau_{\text{pm}} \), WSTselULsel-a is faster than WTselULsel-a if \( p_{\text{acc}}(\sigma) \geq 0.085 \) (0.079 if we assume that matching a reaction effect has half the cost of matching a reaction).

The acceptance probability \( p_{\text{acc}}(\sigma) \) depends on the rate constants of the reaction types and the sampling rates of the supertypes. Since we assumed the interaction to be repulsive, the sampling rate of all four supertypes is \( \nu \exp(-\frac{E_{\text{act}} + 3E_{\text{int}}}{k_B T}) \). The smallest rate constant in the system, for hopping away from an empty environment, is \( \nu \exp(-\frac{E_{\text{act}}}{k_B T}) \). Thus, the acceptance probability is at least

\[
\frac{\nu \exp(-\frac{E_{\text{act}}}{k_B T})}{\nu \exp(-\frac{E_{\text{act}} + 3E_{\text{int}}}{k_B T})} = \exp(\frac{3E_{\text{int}}}{k_B T}).
\]

So, if \( \exp(\frac{3E_{\text{int}}}{k_B T}) \geq 0.085 \), then WSTselULsel-a is faster than WTselULsel-a on this model. This means that the magnitude of the interaction has to be smaller than 0.82\( k_B T \). This, in turn, means that at \( T = 350K \), the repulsion between nearest neighbors has to be less than 2.4 kJ/mol, a rather weak interaction.
3.6 Overview and literature review

In order to get the above numbers, we made many assumptions, regarding the state of the system, the relative values of \( \tau_{\text{rand}} \), \( \tau_{\text{pm}} \), and \( \tau_{\text{fp}} \), et cetera. From the formulas it is not obvious how the two algorithms compare for other systems. Some experimentation is required with a number of systems in order to make a better comparison.

It is possible that the acceptance probabilities of the WSTselULsel-a algorithm are too low in practical cases. If this is so, then the algorithm may be changed to remedy this. Instead of putting all reaction types with the same effect in one supertype, they may be divided over a number of supertypes in such a way that the acceptance probabilities are higher. The simulation of systems with interactions over longer ranges may benefit from this. Generally, nearest neighbor interactions are stronger than interactions of second and higher order neighbors. In this case, reaction patterns that differ only in the higher order neighbor occupations may put put into one group, while reactions that differ in the first order neighbors are put into different groups.

If interaction between second and higher order neighbors is taken into account, the number of reaction types may grow to many thousands or even millions. In the presence of this many reaction types, an approach based on supertypes may be the only viable one, even if acceptance probabilities are low.

Finally, we note that it is possible to avoid storing reaction effects altogether. The result is the WSTselULtrial algorithm. In every iteration of this algorithm a supertype is selected and a unit cell is selected. Then the sum of the rate constants of the reactions of this supertype enabled at the selected unit cell is used to calculate the acceptance probability. The reaction effect is performed with this probability. The optimizations we discussed for WTselULtrial are also applicable to this algorithm.

3.6 Overview and literature review

This section contains an overview of the simulation algorithms presented. In order to clarify the relationships between the different algorithms, they are depicted in the diagram of Figure 3.6. An algorithm at the head of an arrow is a special case of the algorithm at the tail of that arrow.

Figure 3.6 contains some algorithms we have not discussed before. These are found at the right hand side of the diagram. They are based on the selection of reaction effects, hence their names: WREsel(-a) and UREsel(-a). The performance of these algorithms does not match that of WSTselULsel-a. The time per reaction depends on the system size because the time it takes to select a reaction effect in one step depends on the number of reaction effects in \( E \).

The main contribution of the work presented in the preceding sections is that the algorithms are formulated within one framework. We have formulated a concise correctness criterion
Figure 3.6: Overview of simulation algorithms
for simulation algorithms which led to a correct, albeit inefficient, algorithm in a natural way. Identifying three independent ways to improve on this algorithm allowed a systematic exploration of simulation algorithms. The three techniques we identified are sufficient to describe all algorithms found in the literature in our framework, and find new algorithms as well. In particular the WSTselULsel-a algorithm is worth mentioning. Formulating algorithms in a general framework also allowed us to perform a fairly detailed performance analysis. The performance analysis facilitates the comparison of the algorithms discussed. In principle, it should be possible, given a system to be simulated, to select the algorithm that performs best for that system. The number of parameters to take into account when determining which algorithm will be optimal is too large to make the choice straightforward. Making an optimal choice requires knowledge of the outcome of the simulations because of the occurrence of \( n \), the average number of reactions enabled per reaction performed, in the formulas for the time per reaction.

There are, however, some conclusions that can be drawn from the performance analysis. We expect algorithms that do not rely no recording the exact set of enabled reactions to perform better than algorithms that do. Consider, for instance, Equations (3.14) and (3.19) for the times per reaction of WTSselULsel and WTSselULsel-a, respectively. If the set of enabled reactions is recorded, as in WTSselULsel, more pattern matching occurs. These equations show another difference: for some algorithms the time per reaction does not depend on the system size, while for others it does. This becomes particularly important when simulations of large systems are required. A third important aspect in which algorithms differ, is the sensitivity of the time per reaction to the range of rate constants that occur in the model to be simulated. Algorithms such as WTSselULtrial are very sensitive to this, much more than, e.g., WTSselULsel-a.

Based on the results of the performance analysis, we expect WTSselULsel-a to be the algorithm that generally has the lowest time per reaction. For models that do not contain reactions with large patterns, and in which the rate constants do not differ widely, WTSselULtrial is an attractive alternative. This algorithm can be expected to have a low time per reaction for such models, and has very low memory requirements because no set of reactions is stored.

The algorithms in the diagram all have the same structure. In every iteration a time increment is selected, and independent of this a reaction is selected. The simulated time is updated, and the selected reaction is performed if it does not turn out to be invalid. There is a second class of algorithms in which the selection of time and reaction are more closely coupled. This class is discussed in the next section.

We cannot expect Figure 3.6 to contain all possible simulation algorithms with the aforementioned structure. For specific systems, other combinations of the techniques presented in this chapter may lead to efficient simulation algorithms. And we cannot possibly foresee the systems that may be considered for simulation. Some conclusions may be drawn, how-
ever, from our investigation. First, there is no best algorithm. Any algorithm will perform badly for some systems. Because of this, a general purpose simulation package should implement multiple algorithms. This allows good performance for a large class of systems. Second, our investigation resulted in a number of algorithms for which the time per reaction is independent of the system size. All algorithms with this property select reactions in two stages. Algorithms in which reactions are selected in one step have a logarithmic dependence on the number of enabled reactions in their performance. Third, all algorithms encountered in a literature survey are present in our classification or closely related to one of the algorithms in it. We continue this section with a discussion of the algorithms found in the literature.

Abraham and White (1970) wrote one of the earliest papers on the simulation of surface processes. In this paper Abraham and White study vapor deposition on two-dimensional lattices. They describe the simulation algorithm they use in detail. Their algorithm is closely related to ULselWTsel-o. First a unit cell is selected uniformly. If the selected unit cell is empty, then adsorption is tried; if it is occupied, desorption or migration is tried. If the rate of change for an empty site exceeds the rate of change for that site if it were occupied (the sum of the desorption and migration rates if it contained a particle), then an adsorption attempt at that site is accepted unconditionally. Desorption and migration attempts are accepted conditionally in this case. If, on the other hand, the adsorption rate is smaller than the sum of the desorption and migration rates, adsorption attempts are accepted conditionally while desorption and migration attempts are accepted unconditionally. The conditional acceptance of events ensures that they locally have the correct probability of being performed. Globally the kinetics are not correct. Because of the uniform site selection, a particle that has a very low migration rate has the same probability of moving as a particle with a high migration rate in the case that the migration rates exceed the adsorption rates at those sites.

Bortz et al. (1975) are generally credited with the invention of WTselULsel. They were the first to apply the algorithm to Ising spin systems, but the algorithm was used earlier than this in the simulation of crystal growth (Kohli and Ives, 1972). Neither Bortz et al. nor Kohli and Ives keep track of the simulated time. In 1991, the WTselULsel algorithm was reinvented by Bowler and Hood. These authors do not increment the time variable in every iteration of the algorithm, but calculate the elapsed time when the simulation has finished. The result of their calculation for the elapsed time differs from that obtained in our framework. Cao (1994) presents an alternative way to calculate the elapsed time a posteriori. The formula derived can be interpreted as the first moment of the random variable describing the elapsed time for the case that the number of enabled reactions remains constant throughout the simulation for every reaction type. Finally, Nordmeyer and Zaera (1992) reinvented WTselULsel. These authors present the algorithm as a generalization of Gillespie's 'direct method' (Gillespie, 1976), or WRsel.

Other instantiations of WRsel-h occur in the literature long before 1972, when Kohli and
A second class of algorithms

Ives described the algorithm in the context of crystal growth simulations. The earliest occurrence found in our literature survey is that by Dismuke (1956). Dismuke applies an instantiation of WRsel-h to neutron scattering and collision. In this instantiation the time of the next event is determined by sampling the appropriate exponential distribution. Then the event type is determined by means of a weighted selection. In the final step a particular event selected. This does not involve the selection of a location as in our framework, but, for instance, the direction of scattering.

We already mentioned that a number of algorithms used for specific models are really instantiations of WTSelULtrial. Usually, in these algorithms the elapsed time is measured in Monte Carlo steps. In the previous section (page 64) we saw that this gives enough information to calculate the elapsed time for simulations. Examples of this use of WTSelULtrial abound (Aukrust et al., 1990; Dickman and Burschka, 1988; Fichthorn et al., 1989a,b; Ziff et al., 1986). Farbman et al. (1996) describe an algorithm in which the type selection is uniform instead of weighted, and the selected reactions are performed conditionally. This algorithm, UTselULsel, does not occur in Figure 3.6, but it is clear how it can be obtained from WRsel-ho.

3.7 A second class of algorithms

In the first section of this chapter we saw that a simulation algorithm is correct if in any state only enabled reactions are performed, and Equation (3.2) is satisfied. This led directly to the WRsel algorithm of Section 3.2. An alternative to (3.2) to guarantee correct simulation of the kinetics is (3.1), repeated here:

\[
\Pr(W_{alg}(e) > w) = \exp(-k(e)w), \quad (3.1)
\]

for \( w \geq 0 \) and enabled reaction \( e \).

There is a straightforward way to guarantee that (3.1) is satisfied. It is shown in Algorithm 3.9.

Algorithm 3.9 (Rsched)

1. Initialize:
   \[ t := 0; \sigma := \sigma_0; E := \emptyset; \]
   for \( e \in E(\sigma) \)
   \[ \rightarrow \text{sample } w \text{ from } \text{Exp}(k(e)); \]
   \[ E := E + \{(e, w)\}; \]

   return
2. Select the element \((e, t_{occ})\) from \(E\) with minimum occurrence time \(t_{occ}\). Update the state, set the time \(t\) to \(t_{occ}\), and remove all disabled reactions from \(E\). Generate a waiting time \(w\) for every newly enabled reaction \(e\). Add \((e, t + w)\) to \(E\).

In the literature on discrete event simulation, the set \(E\) is called the set of scheduled or pending events. In our case the events are reactions, so we refer to this algorithm as the reaction scheduling algorithm or Rsched.

It is clear that (3.1) is satisfied by the Rsched algorithm, and that only enabled reactions are performed. Gillespie (1976) proposes this algorithm for the simulation of homogeneous chemical reactions, but the algorithm dates back further than that. The correctness of Rsched does not depend on the fact that the waiting times have exponential distributions. The algorithm can be applied to any discrete event system, and it is known as the standard discrete event simulation algorithm or DES (see, e.g., Banks and Carson, 1984; Fishman, 1973; Mitran, 1982). It is used, for instance, for the simulation of queuing systems, population dynamics, digital circuits et cetera.

In the simulation of surface processes the Rsched algorithm is hardly ever used. Ray and Baetzold (1990) use a similar algorithm to simulate surface diffusion. They do not schedule all enabled reactions, but schedule at most one reaction per particle. If a particle may hop in multiple directions, waiting times are generated for all directions. Only the jump with minimum waiting time is scheduled. This is correct because all jumps of a particle are disabled if the scheduled jump is disabled. An extended version of Rsched is used in the simulation of temperature programmed desorption (Jansen, 1995), as we will see in Chapter 4.

The operations performed on \(E\) are the following: insertion of elements, deletion of the element with minimum occurrence time, and deletion of arbitrary elements. A data structure permitting the insertion of elements and deletion of the element with minimum occurrence time is the priority queue (see, e.g., Cormen et al., 1990; Knuth, 1973). It can be implemented in such a way that for both types of operation the time grows logarithmically with the number of elements in the set. An operation for the deletion of arbitrary elements can be added. The cost of such an operation is linear in the size of the set because there is no ordering on reactions. So again the deletion of disabled reactions limits the performance.

In the Rsched algorithm more random numbers are used per reaction than in WRsel. A waiting time is generated for every reaction that becomes enabled, even if it is not performed, when Rsched is used. In WRsel one time increment is generated for every reaction performed.

The three generalization techniques for WRsel presented in Section 3.3 can also be applied to Rsched. The oversampling technique offers a way of turning weighted selections into uniform ones. The Rsched algorithm does not contain any steps involving weighted
3.7 A second class of algorithms

selection, so by itself this technique has no application to Rsched. However, if we schedule reaction effects instead of reactions, we are using oversampling. The reaction effect scheduling algorithm reads as follows.

Algorithm 3.10 (REsched)

1. Initialize:
   \( t := 0; \sigma := \sigma_0; \)
   Compute the set of supertypes \( \delta RT \) from \( RT \). Compute for each supertype \( SRT \) the sampling rate, \( k_{SRT} \), by taking the maximum sum of rate constants of reactions of supertype \( SRT \) that can be enabled at one unit cell at the same time.
   Generate occurrence times for all reaction effects enabled in state \( \sigma_0 \), and store the reaction effects together with these occurrence times in \( E \).

2. (a) Select the reaction effect \((SRT, l, t_{occ})\) from \( E \) with minimum occurrence time \( t_{occ} \). Set \( t \) to \( t_{occ} \). Compute the rate constant \( k \) of \((SRT, l)\) in state \( \sigma \). Accept \((SRT, L)\) with probability \( \frac{k}{k_{SRT}} \) and reject it with probability \( 1 - \frac{k}{k_{SRT}} \).
   (b) If \((SRT, l)\) has been accepted, then update \( \sigma \), schedule all newly enabled reaction effects and add them to \( E \), and delete all disabled reaction effects from \( E \).
   (c) If \((SRT, l)\) has been rejected, then remove it from \( E \). Reschedule it with rate constant \( k_{SRT} \) and insert it in \( E \) again.

When a reaction has been performed and the time has been updated, the waiting times for newly enabled reaction effects have exponential distributions. Since exponential distributions are memoryless (see Appendix A), the (residual) waiting times of reaction effects that were already enabled also have exponential distributions. The amount by which the time is incremented in an iteration of the algorithm is the minimum of the waiting times of the reaction effects in \( E \). This minimum also has an exponential distribution (see appendix A), with parameter \( \Sigma_{SRT} : SRT \in \delta RT : m_{SRT}(E(\sigma))k_{SRT} \). The probability of reaction effect \((SRT, l)\) having the smallest occurrence time is

\[
\frac{k_{SRT}}{\Sigma_{SRT'} : m_{SRT'}(\sigma)k_{SRT'}}
\]

(compare to Equation (2.6)). The probability of it being accepted is equal to

\[
\frac{(\Sigma RT : RT \in SRT \land (RT, l) \in E(\sigma) : k(\sigma))}{k_{SRT}}
\]
Hence the probability of an iteration being successful is

\[
\frac{\sum_{RT, \text{SRT}, l : (\text{SRT}, l) \in E \land (RT, l) \in E(\sigma) \mid k(RT))}{\sum_{\text{SRT}'} : \text{m}_{\text{SRT}'}(\sigma)k_{\text{SRT}'}} = \frac{k(E(\sigma))}{\sum_{\text{SRT}'} : \text{m}_{\text{SRT}'}(\sigma)k_{\text{SRT}'}}. \quad (3.26)
\]

Property 3.1 applies and shows that the waiting time for the next reaction in state \( \sigma \) has an exponential distribution with parameter \( k(E(\sigma)) \). With this expression for the probability of specific reaction effects being performed and the probability of a trial being successful it is easily verified that the probability of reaction effect (\( \text{SRT}, l \)) being the next one to be performed in state \( \sigma \) is

\[
\frac{\sum_{RT : RT \in \text{SRT} \land (RT, l) \in E(\sigma) \mid k(RT))}{k(E(\sigma))}.
\]

As this equals the probability of one of the enabled reactions of supertype \( \text{SRT} \) at unit cell \( l \) being the next reaction, we conclude that the algorithm is correct.

The performance of the RESched algorithm is still limited by the amount of time spent on the deletion of disabled reaction effects from \( E \). This limitation can be overcome by maintaining enabling times for reactions in \( E \) and times of last modification for lattice sites. Instead of removing reactions from \( E \) when they become disabled, their eligibility is checked when they are selected. Ineligible reactions are simply discarded; not even the time variable \( t \) needs to be incremented. This algorithm is called Rsched-a. This improvement over the standard Rsched algorithm was introduced by Jansen (1995) in his paper on the simulation of temperature programmed desorption.

The third generalization consists of allowing groups of reactions to be scheduled. If a group is selected that has more than one element, then one of these is selected with a probability proportional to its rate constant. Algorithm 3.11 contains the details.

**Algorithm 3.11** (Gsched)

1. Initialize:
   \( t := 0; \sigma := \sigma_0; E := \emptyset; \)
   Compute the reactions enabled in \( \sigma_0 \) and put them into groups, say \( s_0, \ldots, s_{H-1} \);
   \( \text{for } i \in [0..H) \)
   \( \quad \rightarrow \text{sample } w \text{ from } \text{Exp}(k(s_i)); \)
   \( \quad E := E + \{(s_i, w)\}; \)
   \( \text{rof} \)

2. Select the element \( (s, t_{occ}) \) from \( E \) with minimum occurrence time. Select a reaction from \( s \), giving probability \( \frac{k(e)}{k(s)} \) to reaction \( e \). Set the time to \( t_{occ} \), and update \( \sigma \). Delete
disabled reactions from $E$ and add newly enabled reactions to it. This may involve adding elements to existing groups, adding new groups, removing elements from existing groups, and removing groups (because all elements in a group may become disabled).

Generate waiting times for the new groups, compute occurrence times and add them to $E$. Delete groups that changed from $E$, reschedule them, and add them to $E$ again.

Because of the memorylessness of the exponential distribution, the (residual) waiting times for the reaction groups all have exponential distributions, the distribution for a group $s$ having parameter $k(s)$. The waiting time for the next reaction is the minimum of the groups' waiting times, and hence has an exponential distribution itself. The parameter of this distribution is $(\Sigma i : k(s_i)) = k(E(\sigma))$, in accordance with Equation (3.2a). The probability of an enabled reaction $e$ being the next reaction, with $e$ being an element of a group $s$, is equal to the probability of $s$ having the minimum waiting time in $E$ and $e$ being selected from $s$. Therefore, $e$ is the next reaction with probability $k(s) / k(E(\sigma))$.

Equation (3.2b) is satisfied as well, and we conclude that Gsched is a correct simulation algorithm.

There are several ways to group reactions. Two of them we have already explored in the discussion of WRsel-h. Grouping reactions by their type is the first. It leads to an algorithm in which types are scheduled (Tsched) and has the benefit that selection of reactions from groups is uniform. The second way of grouping reactions is by their enabling location (Lsched). This choice permits the cost of deletion of disabled reactions to be reduced. A third possibility is to put all enabled reactions into one group. The resulting algorithm is a familiar one: WRsel. Actually, Step 2 allows the selection of reactions from groups to be done hierarchically. WTSelULsel, WTSelULSearch and WLselWTSel can be seen as special cases of this algorithm, Gsched-h.

Gsched may be generalized further by introducing oversampling in it and not requiring the immediate deletion of disabled reactions. The following algorithm is the result.

**Algorithm 3.12** (Gsched-a0)

1. Initialize:
   $\sigma := \sigma_0$; $t := 0$;
   for $l \in L \rightarrow \text{lm}[l] := 0$; rof
   Compute the reactions enabled in $\sigma_0$; put every enabled reaction in a reaction group, together with a sampling rate and with enabling time zero, ensuring that rate constants do not exceed sampling rates. Assign sampling rates to reaction groups, in such a way that the sum of the sampling rates of the reactions in a group does not exceed the group's sampling rate. Schedule the reaction groups according to their sampling rates.
2. (a) Select the reaction group \((s, sr, t_{occ})\) with minimum occurrence time \(t_{occ}\). Set \(t\) to \(t_{occ}\). Accept \(s\) with probability

\[
\frac{(\sum e, sr', t_{en} : (e, sr', t_{en}) \in s : sr')}{sr'}
\]

If \(s\) is accepted, then continue at 2b, otherwise the iteration ends.

(b) Select an element from \(s\), giving probability

\[
\frac{(\sum e', sr'', t_{en}' : (e', sr'', t_{en}') \in s : sr'')}{sr'}
\]

to \((e, sr', t_{en}')\). Accept \(e\) with probability \(\frac{k(e)}{sr'}\). If \(e\) is accepted, then continue at 2c, otherwise the iteration ends.

(c) Remove the selected triple \((e, sr', t_{en}')\) from \(s\). If \((e, sr', t_{en})\) is eligible, then update \(\sigma\) and \(lm\). Compute the newly enabled reactions and add them to existing reaction groups or put them into new ones, together with a sampling rate and enabling time. Reschedule all existing groups that changed and schedule all new groups.

The ingredients of the correctness arguments of WRsel-hao and Gsched may be combined to show that Gsched-ao is a correct simulation algorithm. The eligibility check ensures that multiple instances of the same reaction are properly dealt with. Step 2 is the same as Step 3b of WRsel-hao on page 65 with respect to the probability of reactions being selected and accepted. Therefore Gsched-ao satisfies Equation (3.2b) and the probability of an iteration being successful is \(\frac{F(\sigma)}{R}\), where \(R\) is the sum of the groups’ sampling rates. The time increment per iteration of the algorithm is the minimum of independent, exponentially distributed variables. It is therefore exponentially distributed as well, with parameter \(R\). Property 3.2 is applicable and shows that Equation (3.2a) is satisfied.

Gsched-ao may also be generalized by making the reaction selection within groups hierarchical. The WRsel-hao algorithm is a special case of the resulting Gsched-hao algorithm. The algorithms discussed in Sections 3.2, 3.3, and 3.5 are all special cases of WRsel-hao, so all algorithms discussed can be obtained from Gsched-hao.

Instances of the Gsched-hao algorithm in which more than one reaction group is used, such as TschedULsel-a and LschedWTsel, perform worse than algorithms that use only one reaction group, such as WTselULsel-a and WLselWTsel. We consider the performance of TschedULsel-a in more detail and compare it to the performance of WTselULsel-a.

The time per reaction for the TschedULsel-a algorithm can be expressed in terms of the cost of the basic operations introduced before and the cost of operations on priority queues. In an unsuccessful iteration of the TschedULsel-a algorithm, a reaction is selected, the time is incremented, the reaction’s eligibility is checked, it is removed from \(E\), and the
3.7 A second class of algorithms

type to which it belongs is rescheduled. Selection of a reaction involves finding the type with minimum scheduled time and selecting a location uniformly. We write $\tau_{pqfindmin}$ for the cost of finding the type with minimum scheduled time. Incrementing the time amounts to setting the time to the scheduled time of the selected type, an operation with cost $\tau_{fp}$. Deleting an element from $E$ amounts to deleting an element from an array and moving the last array element to fill the hole. Rescheduling a priority queue element can be done by deleting it and reinserting it with a new scheduled time. We denote the cost of the rescheduling operation by $\tau_{resched}$. The cost of an unsuccessful iteration is

$$\tau_{pqfindmin}(|E|) + \tau_{usel} + \tau_{fp} + \tau_{acc} + 2\tau_{fp} + \tau_{resched}(|E|).$$

In a successful iteration of the algorithm the operations performed in an unsuccessful iteration are performed, and a reaction occurs. This means that the state and the lattice modification times are updated and that newly enabled reactions are checked for and inserted into $E$. Insertion of a reaction into $E$ amounts to adding an element to an array and rescheduling a reaction type. Assuming, as usual, that one reaction leads to $n$ reactions becoming enabled on average, we obtain the following expression for the extra cost of a successful trial:

$$2\tau_{pm} + n_{mod}n_{src}\tau_{pm} + n(2\tau_{fp} + \tau_{resched}(|RT|)).$$

As it takes, on average $n$ iterations for a reaction to occur, the cost per reaction is

$$\tau_{TschedULse} = n(\tau_{pqfindmin}(|RT|) + 2\tau_{resched}(|RT|)) + n(\tau_{usel} + \tau_{acc} + 5\tau_{fp}) + (n_{mod}n_{src} + 2)\tau_{pm}. \quad (3.27)$$

The minimum element of a priority queue can be found in constant time. We attribute cost $\tau_{fp}$ to it. Rescheduling an element involves deleting and reinserting it. Both operations are logarithmic in the size of the priority queue. We denote their cost by $\tau_{pqins}$ and $\tau_{pqdel}$, respectively. The rescheduling operation also involves generating a new scheduled time. This amounts to generating a random number ($\tau_{rand}$), calculating a waiting time ($2\tau_{fp}$), and adding it to the current time ($\tau_{fp}$). We substitute this in Equation (3.27) to obtain

$$\tau_{TschedULse} = 3n\tau_{rand} + (n_{mod}n_{src} + n + 2)\tau_{pm} + 13n\tau_{fp} + 2n(\tau_{pqins}(|RT|)) + \tau_{pqdel}(|RT|). \quad (3.28)$$

Even if the total cost of priority queue deletion and insertion is only $2 \log_2(|RT|)\tau_{fp}$, the cost of a reaction is higher for the TschedULse-a algorithm than it is for the WTselULse-a algorithm.

In Sections 3.3.2 and 3.5.1 we saw that it is not necessary to record any enabled reactions. The resulting algorithms, WTselULsearch and WTselULtrial have very low memory requirements. In the same way we can obtain algorithms with low memory requirements.
from the Gsched-a algorithm. In these algorithms types are scheduled. A random site is tried to find a reaction (TschedULtrial), or the lattice is searched (TschedULsearch) for an enabled reaction of the type that has the smallest occurrence time.

There is one degree of freedom in Gsched-hao we have not exploited yet. If there are multiple reaction groups, then the selection within these groups may be performed using different selection mechanisms for different groups. We may, for instance, divide the reaction types into three groups. Reaction types that have small patterns and are enabled at a large fraction of the unit cells can be put into a group. The selection of reactions from this group can then be done by using the selection scheme of WTselULtrial. Reaction types that are part of large supertypes may be put into another group. It is more advantageous to use the WSTselULsel-a selection mechanism on this group. Any remaining reaction types may be put into a third group. Reactions from this group can be selected using the selection scheme of WTselULsel-a. In this approach different reaction types can be simulated according to the algorithm best suited for them, thus improving performance. Whether this improvement is significant is hard to judge from the formulas for the time per reaction for the various algorithms. It requires more detailed knowledge of the parameters that occur in the formulas. In Chapter 5 we perform some experiments, using WTselULsel-a and WTselULtrial in one simulation.

The algorithms discussed in this section are not used very often in the simulation of surface processes. Apart from the occurrences of Rsched (Jansen, 1995; Ray and Baetzold, 1990), our literature survey showed only one article, describing an algorithm in the class of Gsched-hao based algorithms (Dumont and Dufour, 1986). In this article an algorithm is described that can be seen as an instance of Gsched-h with two reaction groups. Within one group a selection based on Rsched is used. The authors do not schedule reactions but 'particle futures.' They do not properly handle particles for which more than one reaction is possible. In particular, in a situation like in Figure 3.7 only one associative $B_2$ desorption

![Figure 3.7: Example of associative $B_2$ desorption from (Dumont and Dufour, 1986). According to the program text in the article it is possible that only the encircled reaction is scheduled, although four others are possible too.](image)

is scheduled, although more are enabled. Within the other group, selection is performed according to WTselULtrial.
Figure 3.8: Overview of the simulation algorithms based on scheduling (groups of) reactions.
All algorithms in this section except Rsched are new. They are depicted in the second overview noindent of this chapter in Figure 3.8. The most important contribution, however, is not found in the individual algorithms but in the observation that Gsched-hao allows the algorithms described in the preceding sections to be combined.

### 3.8 Approximate algorithms

In the discussion of the WTselULtrial algorithm we already mentioned that sometimes the generation of time increments is simplified. Instead of sampling a probability distribution, the mean of the distribution is taken. The result is an algorithm that approximates the correct time evolution of systems. Stochastic fluctuations in the time increments are eliminated by working with averages.

If the stochastic fluctuations in the waiting times for the next events are not of interest, the performance of the algorithms discussed in Section 3.2, 3.3, and 3.5 may be increased by removing the sampling in the time increment step. This cannot be done in the algorithms of Section 3.7. In those algorithms the waiting times do not only determine the time increment, but they also determine the order in which reactions occur. If the fluctuations in the waiting times are removed, anomalous results will be obtained from the simulations.

In the algorithms discussed, the time increments depend on the state of the system, and on its rate of change in particular. The algorithms are event-based: time is incremented with the amount of time that elapses from one event (or attempt) to the next. Another possibility is to take a fixed time increment and calculate for each step the probability of an event occurring in that step. Algorithms based on this approach are called interval-driven algorithms in the literature on discrete event simulation.

The variant of WTselULtrial in which the average of the $\text{Exp}(L_0L_1R)$ distribution is used for the time increment has a fixed time step, of size $\frac{1}{L_0L_1R}$. Therefore it can be viewed as an interval-driven algorithm. The probability of an iteration of the WTselULtrial being successful is

$$\left( \Sigma RT, l : (RT, l) \in E(\sigma) : \frac{1}{L_0L_1} k(RT) \right) = \frac{k(E(\sigma))}{L_0L_1R}.$$ 

According to Equation (2.4), the probability of the next reaction occurring in the interval $(t, t + \frac{1}{L_0L_1R})$ is $1 - \exp(-\frac{k(E(\sigma))}{L_0L_1R})$. This means that the fixed time step version of WTselULtrial gives a good approximation to the correct time evolution only if $\frac{k(E(\sigma))}{L_0L_1R}$ is close to zero for all states. In most cases this condition will be satisfied.

In general, a fixed time step algorithm gives a good approximation of the correct time evolution if the time step $\Delta t$ is so small that in any state, $k(E(\sigma))\Delta t$ is a good approximation
of \(1 - \exp(-k(E(\sigma))\Delta t)\). In practice this means that the probability of a reaction occurring in a time step has to be small in order to get accurate simulations. In the literature this accuracy is often sacrificed for the benefit of faster simulations. Time steps are chosen so large that a reaction occurs in every step. In the simulation of systems with only diffusion reactions, all particles are allowed to move in one step (Bitar et al., 1994; Dawnkaski et al., 1995; Lombardo and Bell, 1988; Lutsevich et al., 1991; Sales and Zgrablich, 1987a; Tringides and Gomer, 1984). If in such an approach a particle moves to a position where its jump rate is very high, particles that have not yet moved in the current step have a large probability of moving earlier, although their jump rates are lower. A careful validation of the simulation results is called for when this approach is used.

Fixed time step algorithms do not perform well for systems in which the rate of change varies over a wide range for different states. In order to get accurate simulation results, the probability of a reaction occurring in a step must be small in all states, also in those with a high rate of change. If the rate of change decreases, for example because the system gets closer to equilibrium, then the probability of a reaction occurring in a step becomes smaller. This may lead to unacceptably long simulation times. Bitar et al. (1994) propose a solution to this problem. The solution consists of adapting the step size so that larger time steps are made when the rate of change is small. Another solution is to use event-based algorithms for such systems. These usually show better performance.

Dawnkaski et al. (1995) do not only describe a fixed time step algorithm, but they discuss other approaches found in the literature too. The algorithm they propose is one in which all sites are visited in every simulation step (it is not clear whether the sites are visited in some fixed or in random order). For each site the probability of reaction is computed and then a reaction is performed. The authors choose the time step in such a way that the probability for the fastest reaction is about 0.5. As we have argued above, the dynamics of such a simulation do in general not reflect the reaction kinetics accurately. Even if they do, all fluctuations in the waiting times are ignored.

Dawnkaski et al., however, consider their algorithm superior to Rsched as described by Ray and Baetzold (1990), and to WTselULsel as described by Kang and Weinberg (1989) and Bowler and Hood (1991). Their first complaint about those descriptions is that the relationship between simulated time and real time is unclear. This may be the case for the description given by Bowler and Hood (1991), but furthermore the Dawnkaski et al. seem to think that these algorithms are incorrect and do not allow slow reactions to occur before fast ones. In reality, Rsched and WTs elULsel do allow reactions with low rate constants to occur before faster ones, with the correct probability. Owing to the large step size this is not the case for the algorithm proposed by Dawnkaski et al. Again the source of confusion about the relation between simulated and real time seems to stem the fact that these authors consider only the mean waiting times for reactions instead of the appropriate probability distributions.
3.9 Hybrid approaches

A conceptual advantage of our approach to simulating surface processes is the uniform treatment of all processes. No distinction is made between reaction and diffusion in the model nor in the simulations. This approach has a drawback as well. If, for instance, diffusion is orders of magnitude faster than any chemical reaction in a system, then most simulation time will be spent on diffusion steps. The only effect of performing many diffusion steps is that the adsorbed overlayer equilibrates between consecutive reaction steps.

In general, if the reaction types in a system can be partitioned into two classes, the rate constants of which differ by orders of magnitude, the state is brought to equilibrium by the fast processes between consecutive slow reactions. Several authors have observed that simulating the fast processes at their real rate constants is infeasible because of large computation times, and sometimes unnecessary because equilibrium is also reached if the rate constants of the fast processes are lowered (Koper et al., 1998; Weinketz, 1994; Weinketz and Cabrera, 1997). A point worth mentioning in this context is that the rate constant for diffusion needed to obtain equilibrium decreases if the lattice size decreases.

A solution to the problem of fast and slow processes may be found in using a combination of dynamic simulation for the slow processes and classical Monte Carlo simulation of the fast processes. This approach is taken by Meng and Weinberg (1994, 1995, 1997); Sales and Zgrablich (1987a,b), who study systems in which diffusion is much faster than adsorption and desorption. After every adsorption or desorption step, the adparticles are redistributed by employing the standard Metropolis algorithm (Metropolis et al., 1953). As explained in Section 3.3.1, this is a Monte Carlo technique in which the acceptance probability of selected reactions depends on the change in the system's energy. Reactions that bring the system into a more favorable state are accepted unconditionally. Equilibrium is reached faster this way.

Another solution to solving the problem of analyzing systems with vastly differing rate constants is proposed by Silverberg and Ben-Shaul (1987, 1988, 1989); Tammaro et al. (1995). These authors study in which there is one highly mobile species. The particles of the mobile species are not assigned to lattice sites like the particles of the stationary species. Instead, only the number of mobile particles is recorded. Based on this number and on the configuration of immobile particles, probabilities are computed for finding mobile particles at empty sites. Instead of having reaction types of the form \( A + B \rightarrow e + e \), for dissociative \( AB \) desorption there are reaction types of the form \( e + B \rightarrow e + e \). The rate constant of each such reaction is the product of the rate constant of the original reaction type and the probability of the \( e \) site being occupied by \( A \). If a reaction occurs, the number of mobile particles is decremented and a new equilibrium distribution over the empty sites is computed. As the range of lateral adsorbate-adsorbate interaction increases, the advantages
of this hybrid approach over a uniform simulation approach disappear (Silverberg and Ben-Shaul, 1989). A problem in this approach is that it effectively allows mobile particles to move between topologically disconnected regions of empty sites.

As the hybrid approaches still use dynamic Monte Carlo simulation for some of the processes, the techniques discussed in this chapter may be employed to speed up the dynamic part of the simulation.
Simulation of Systems with Time-Dependent Reaction Rates

The model for surface processes described in Chapter 2 deals with systems in which rate constants are constant over time. Sometimes this is too restrictive. It precludes, for example, the description of temperature programmed desorption (TPD) and reaction (TPR) experiments, and voltammetric scans in electrochemistry.

In this chapter we start with a brief explanation of these applications. Next we extend the model so we can deal with arbitrary time-dependent rate constants. We then investigate which simulation algorithms can be used for this more general class of models and analyze their performance. Surprisingly, it turns out that algorithms based on scheduling reaction types perform best. We conclude the chapter with a literature review.

The algorithms be to discussed in this chapter bear a close resemblance to the algorithms discussed in Chapter 3. For this reason, we do not discuss all aspects of the algorithms as extensively as we did in Chapter 3. We focus on the new aspects, those related directly to the time dependence of the rate constants. As a result, this chapter is not easily read without having read Chapter 3 first.

4.1 Applications

As announced in the introductory remarks to this chapter, we begin the discussion of systems with time-dependent rate constants with two applications, viz. TPD and TPR systems, and electrochemistry.
A well-known way of studying the behavior of adsorbates at a surface is by means of a technique called temperature programmed desorption or TPD. At the beginning of a TPD experiment, particles are allowed to adsorb at low, constant temperature. In the second phase of the experiment, the temperature of the system is raised at some fixed rate and the desorption rate of adsorbed particles is measured. The measurements can be used to calculate adsorption energies, interaction between adsorbates, etcetera (Yates, 1985). It is possible that the adsorbed particles react before desorbing, in which case we speak of temperature programmed reaction.

We assume that an Arrhenius expression can be used for the rate constants and that the temperature is raised at rate $\alpha$. If the initial temperature is $T_0$, then we obtain time-dependent rate constants of the form

$$k(t) = v \exp \left( -\frac{E_{\text{act}}}{k_B(T_0 + \alpha t)} \right).$$

(4.1)

More recently, dynamic MC simulations have raised interest in the field of electrochemistry. A tutorial discussion of Monte Carlo methods for equilibrium and non-equilibrium problems in interfacial electrochemistry can be found in (Brown et al., in press). Reactions take place at a metal-electrolyte interface. Their rate constants are influenced by a voltage that is applied between the electrolyte and the metal surface. In a so-called cyclic voltammetric scan the potential is raised at a fixed rate for some time, and then lowered. Adsorption and desorption of particles change with the potential. By measuring the current that results from charged particles adsorbing and desorbing, adsorption and interaction energies can be calculated. The time-dependent rate constants have the following form:

$$k(t) = v \exp \left( -\frac{ae_0\phi(t)}{k_BT} \right).$$

(4.2)

In this equation, $\alpha$ is the so-called transfer coefficient and $e_0$ the charge of an electron. The potential at time $t$ is denoted by $\phi(t)$. It is usually of the form $\phi(t) = \phi_0 + \beta t$. In (Brown et al., in press) these systems are simulated by performing a number of simulations, all with fixed potential. The output configuration of one simulation forms the input for the next. In this way a simulation with time-dependent rates is approximated.

The approach to simulating a system with time-dependent rates by means of a sequence of simulations is not exact. In fact, it may be difficult to decide how long the individual simulations should be and how large the rate increments should be. If the increments are too large, which is equivalent to the individual simulations in the sequence being too long, incorrect results may be obtained. In the literature review at the end of this chapter we return to the subject of approximate algorithms for the simulation of systems with time-dependent rates.

The formulas above show two ways in which reaction rates may vary with time. In the ensuing section we investigate how, in general, the time-dependence of the reaction rates influences the distribution of the reaction waiting times.
4.2 Extension of the model

The only extension of the model is that the reaction rates are now functions on the nonnegative real numbers. We restrict our attention to rate functions that are continuous. Under this restriction the kinetics assumption from Section 2.3.2 carries over virtually unchanged. The only difference is that the current time now occurs in it:

If a reaction \( e \) with rate function \( k(e) \) is enabled in state \( \sigma \) at time \( t \), then the probability of this reaction occurring in the infinitesimal interval \( (t, t + dt] \) is equal to \( k(e, t)dt \). The probability that more than one reaction occurs in \( (t, t + dt] \) is \( o(dt) \).

Starting from this assumption, we can again write a differential equation for the waiting time of a normal (i.e., non-immediate) reaction \( e \) enabled at some time \( t \). We denote this waiting time by \( W(e, t) \). The differential equation is obtained in the same way as in the constant-rate case, and its shape is the same too:

\[
\frac{d \mathbb{P}[W(e, t) > w]}{dt} = -k(e, t) \mathbb{P}[W(e, t) > w],
\]

for \( w \geq 0 \). Integration yields

\[
\mathbb{P}[W(e, t) > w] = \exp(-\int_t^{t+w} k(e, u)du). \tag{4.3}
\]

We call this a time-dependent exponential distribution with parameter \( k(e) \), and write tdExp(\( k(E) \)) as an abbreviation. As explained in Appendix A, there may be a nonzero probability of \( W(e, t) \) being infinite. This corresponds to \( e \) not occurring in finite time. The probability of \( e \)'s waiting time being infinite is

\[
\mathbb{P}[W(e, t) = \infty] = \lim_{w \to \infty} \exp(-\int_t^{t+w} k(e, u)du).
\]

Conversely, if the waiting time of some event \( e \) has a tdExp(\( k(e) \)) distribution, then the probability of \( e \) occurring in a vanishingly small interval \( (t, t + dt] \) equals \( k(e, t)dt \). This means that the descriptions of the kinetics in terms of reaction waiting times and in terms of reaction probabilities in infinitesimal intervals are equivalent.

When dealing with the correctness of simulation algorithms, it is convenient to have a formulation of the system kinetics in which only the waiting time of the next reaction and the probability of a particular reaction being the next one occur. Equations (2.7a) and (2.7b) give such a formulation for systems with constant rates. Unfortunately, if rates are time-dependent, the next reaction and the waiting time of the next reaction are not independent. Therefore, we do not calculate the probability of an enabled reaction \( e \) being the next reaction, but we calculate the probability that \( e \) is the next reaction and occurs before \( w \) time elapses (\( w < \infty \)):

\[
\mathbb{P}[W(e, t) > w] = \exp(-\int_t^{t+w} k(e, u)du),
\]

for \( w \geq 0 \). Integration yields

\[
\mathbb{P}[W(e, t) > w] = \exp(-\int_t^{t+w} k(e, u)du). \tag{4.3}
\]

We call this a time-dependent exponential distribution with parameter \( k(e) \), and write tdExp(\( k(E) \)) as an abbreviation. As explained in Appendix A, there may be a nonzero probability of \( W(e, t) \) being infinite. This corresponds to \( e \) not occurring in finite time. The probability of \( e \)'s waiting time being infinite is

\[
\mathbb{P}[W(e, t) = \infty] = \lim_{w \to \infty} \exp(-\int_t^{t+w} k(e, u)du).
\]

Conversely, if the waiting time of some event \( e \) has a tdExp(\( k(e) \)) distribution, then the probability of \( e \) occurring in a vanishingly small interval \( (t, t + dt] \) equals \( k(e, t)dt \). This means that the descriptions of the kinetics in terms of reaction waiting times and in terms of reaction probabilities in infinitesimal intervals are equivalent.

When dealing with the correctness of simulation algorithms, it is convenient to have a formulation of the system kinetics in which only the waiting time of the next reaction and the probability of a particular reaction being the next one occur. Equations (2.7a) and (2.7b) give such a formulation for systems with constant rates. Unfortunately, if rates are time-dependent, the next reaction and the waiting time of the next reaction are not independent. Therefore, we do not calculate the probability of an enabled reaction \( e \) being the next reaction, but we calculate the probability that \( e \) is the next reaction and occurs before \( w \) time elapses (\( w < \infty \)):
\[ \mathbb{P}[W(e, t) \leq w \wedge W(e, t) < W(E(\sigma) - \{e\}, t)] = \int_0^w f_{W(e)}(t, u)du \int_u^\infty f_{W(E(\sigma) - \{e\})}(t, u)du' + \mathbb{P}[W(e, t) \leq w] \mathbb{P}[W(E(\sigma) - \{e\}, t) = \infty] . \]

We continue with the first term of the last expression of this derivation, using the fact that the density function of a random variable \( X \) with \( \text{tdExp}(k) \) distribution is given by

\[ f_X(t, x) = k(t + x) \exp(-\int_t^{t+x} k(u)du) \]

for \( t, x \in [0, \infty) \). We derive

\[ \int_0^w k(e, t + u) \exp(-\int_t^{t+u} k(e, v)dv)du \cdot \int_u^\infty k(E(\sigma) - \{e\}, t + u') \exp(-\int_{t+u'}^{t+u} k(E(\sigma) - \{e\}, v)dv)du' = \{ \text{Calculus} \} \int_0^w k(e, t + u) \exp(-\int_t^{t+u} k(e, v)dv) \cdot \left( \exp(-\int_t^{t+u} k(E(\sigma) - \{e\}, v)dv) - \mathbb{P}[W(E(\sigma) - \{e\}, t) = \infty] \right)du \]

\[ = \{ \text{Calculus; } k(e, v) + k(E(\sigma) - \{e\}, v) = k(E(\sigma), v) \} \int_0^w k(e, t + u) \exp(-\int_t^{t+u} k(E(\sigma), v)dv)du - \mathbb{P}[W(e, t) \leq w] \mathbb{P}[W(E(\sigma), t) = \infty] . \]

Combining the results of these two derivations, we obtain

\[ \mathbb{P}[W(e, t) \leq w \wedge W(e, t) < W(E(\sigma) - \{e\}, t)] = \int_0^w k(e, t + u) \exp(-\int_t^{t+u} k(E(\sigma), v)dv)du . \]

We rewrite this once more, using the fact that \( e \) is an enabled reaction, to obtain the following equation:

\[ \mathbb{P}[W(E(\sigma), t) \leq w \wedge W(e, t) < W(E(\sigma) - \{e\}, t)] = \int_0^w k(e, t + u) \exp(-\int_t^{t+u} k(E(\sigma), v)dv)du . \quad (4.4) \]
4.3 The time-dependent WRsel algorithm

The probability of $e$ being the only reaction to occur in $(t, t+w)$ is $k(e, t)w$ for $w \to 0$ if $e$'s waiting time is defined by (4.4). Furthermore, Equation (4.4) implies that the probability of two reactions occurring in a vanishingly small interval $(t, t+w)$ is $o(w)$. Hence, (4.4) is equivalent to the kinetics assumption for systems with time-dependent rates.

Before we go on to discuss simulation algorithms for systems with time-dependent rates, we point out that if we take $k(e, t)$ to be independent of $t$, then Equation (4.4) reduces to (2.7) (see page 25).

4.3 The time-dependent WRsel algorithm

The first algorithm that we discussed for systems with constant rates followed directly from the description of the kinetics in terms of waiting times. For time-dependent systems, this is not the case. We simply start with generalizing the WRsel algorithm, and later show that it is correct.

Algorithm 4.1 (tdWRsel)

1. Initialize: $t := 0$; $\sigma := \sigma_0$; $E := E(\sigma)$;

2. Generate a time increment $w$ by sampling the $\text{tdExp}(k(E, t))$ distribution. Stop if $w = \infty$. Increment $t$ by $w$.

3. Select a reaction from $E$, reaction $e$ with probability $\frac{k(e, t)}{k(E, t)}$. Update the state ($\sigma := \mathcal{C}(e, \sigma)$) and recalculate $E$ so that it contains all reactions enabled in the new state ($E := E(\sigma)$).

We prove this algorithm correct by verifying that Equation (4.4) is satisfied for the simulated waiting times $W_{\text{alg}}$. Let $e$ be a reaction enabled in state $\sigma$ at time $t$ and let $w \in [0, \infty)$. We calculate as follows:

\[
P\{W_{\text{alg}}(E(\sigma), t) \leq w \wedge W_{\text{alg}}(e, t) < W_{\text{alg}}(E(\sigma) - \{e\}, t) \}
= \{ E = E(\sigma) \}
= P\{W_{\text{alg}}(E, t) \leq w \wedge W_{\text{alg}}(e, t) < W_{\text{alg}}(E - \{e\}, t) \}
= \int_0^\infty P\{W_{\text{alg}}(E, t) \leq w \wedge W_{\text{alg}}(e, t) < W_{\text{alg}}(E - \{e\}, t) | W_{\text{alg}}(E, t) = u \}
\cdot f_{W_{\text{alg}}(E)}(t, u) \, du
+ P\{W_{\text{alg}}(E, t) \leq w \wedge W_{\text{alg}}(e, t) < W_{\text{alg}}(E - \{e\}, t) | W_{\text{alg}}(E, t) = \infty \}
\cdot P\{W_{\text{alg}}(E, t) = \infty \}
\]
Simulation of Systems with Time-Dependent Reaction Rates

\[ \mathbb{P}(X \leq x | X = y) \text{ is 0 for } y > x, 1 \text{ for } y \leq x \]
\[ \int_0^w \mathbb{P}(W_{alg}(e, t) < W_{alg}(E - \{ e \}, t)|W_{alg}(E, t) = u) f_{W_{alg}}(E)(t, u) du \]
\[ = \{ \text{Prob. of selection of } e \text{ at the new current time, } t + u, \text{ is } \frac{k(e, t + u)}{k(E, t + u)} \} \]
\[ \int_0^w \frac{k(e, t + u)}{k(E, t + u)} f_{W_{alg}}(E)(t, u) du \]
\[ = \{ W_{alg}(E) \text{ has a tdExp}(k(E)) \text{ distribution}; E = E(\sigma) \} \]
\[ \int_0^w k(e, t + u) \exp(- \int_t^{t+u} k(E(\sigma), v) dv) du . \]

This shows that the algorithm is correct.

The tdWRsel algorithm was first described by Jansen (1995); he calls it the variable step size method. Jansen describes how to adapt both WRsel and Rsched to simulate TPD and TPR experiments and analyzes the time per reaction for tdWRsel and tdRsched-a. He concludes that the latter is more efficient. We discuss the time-dependent variant of the Rsched algorithm in Section 4.6.

Prados et al. (1997) also described the tdWRsel algorithm, as a way to explain the tdWTSEL algorithm. This latter algorithm, which we discuss shortly, was used to simulate one-dimensional Ising systems with decreasing temperatures. Prados et al. present a general scheme for generating time increments (step 2 of the algorithm) for the case that no simple analytical expression exists for the primitive of \( k(E(\sigma), t) \) w.r.t. \( t \).

The tdWRsel and WRsel algorithms have the same structure. A time for the next reaction is generated, and then a reaction is selected and performed. The implementation of the operations is different, however, because rates are time-dependent in tdWRsel. We look at steps 2 and 3 in more detail. Along with a description of how these steps are implemented, we analyze the cost per reaction for tdWRsel. As in Chapter 3 we assume that each reaction disables \( n \) reactions and enables \( n \) in this analysis. The cost of a reaction is made up of five terms:

\[ \tau_{tdWRsel} = \tau_{time} + \tau_{wsel}(|E(\sigma)|) + \tau_{react} + \tau_{disab} + \tau_{enab} . \]

First we consider step 3 of tdWRsel. It says that reaction \( e \) should have a probability \( \frac{k(e, t)}{k(E, t)} \) of being selected from \( E \) if the current time is \( t \). The time changes in every iteration of the algorithm, so the rate constants change from one iteration to the next as well. Therefore, in every iteration, the rate constants of all reactions in \( E \) have to be recalculated. The cost of evaluating a rate constant is denoted by \( \tau_{rate} \).

As rate constants have to be recalculated in every iteration, the use of a tree does not offer any advantages over a list representation of \( E \). The values at all nodes of the tree would have to be recalculated before every selection from \( E \), keeping the cost of selection linear
in the size of $E$. So we use a list representation for $E$. The cost of selection of a reaction is bounded from above by
\[
\tau_{wsel}(|E(\sigma)|) = |E(\sigma)|\tau_{rate} + |E(\sigma)|\tau_{fp}
\]
\[
+ \tau_{rand} + \tau_{fp}
\]
\[
+ |E(\sigma)|\tau_{fp}.
\]

The first two terms of the right hand side give the cost of recalculating rate constants and their partial sums. The third and fourth terms give the cost of generating a random number in the right range, and the last term that of selecting the reaction from the list corresponding to this number.

The cost of finding and deleting $n$ disabled reactions is lower because deletion from a list is less costly than deletion from a tree. Deleting an element from a list can be done at cost $\tau_{fp}$, so we have
\[
\tau_{disab} = |E(\sigma)|\tau_{pm} + n\tau_{fp}.
\]

The cost of updating the state is $\tau_{pm}$. Finally, in step 3 of the algorithm, it is checked which reactions have become enabled, and the enabled ones are inserted in $E$. As the cost of list insertion is $\tau_{fp}$, we have
\[
\tau_{enab} = n_{modnr}\tau_{pm} + n\tau_{fp}.
\]

Second, we investigate how to generate time increments. Using the inverse function method (see Section 3.4.2), we obtain to solve $w$ from
\[
\int_1^{t+w} k(E, v)dv = -\ln u, \quad (4.5)
\]
where $u$ is obtained by sampling the unit interval uniformly. It is possible that
\[
\int_1^{\infty} k(E, v)dv < -\ln u \quad (4.6)
\]
because $W(E(\sigma))$ may have value $\infty$. Selecting an infinite time increment corresponds to Equation (4.6) being true. If (4.6) does not hold, then (4.5) has a unique solution. It is possible, however, that it cannot be solved analytically. If this is the case, a numerical scheme such as that presented in (Prados et al., 1997) or any other numerical solution method may be used. Let us assume that $k(E, t)$ can be integrated analytically, i.e., we assume that for every reaction $e$ there is a function $K(e)$ such that $\frac{dK(e,t)}{dt} = k(e,t)$, and write $K(E, t) = (\Sigma e : e \in E : K(e, t))$. Then $w$ has to be solved from
\[
K(E, t + w) - K(E, t) = -\ln u. \quad (4.7)
\]
For the case that the rate of $e$ is of the form (4.1), as in TPD experiments, we get

$$K(e, t) = \frac{v}{\alpha} (T_0 + \alpha t) E_2 \left( \frac{E_{sat}}{k_B (T_0 + \alpha t)} \right),$$

and

$$E_2(x) = \int_1^\infty \frac{\exp(-x u)}{u^2} du$$

(see Jansen, 1995). No analytical method is known to solve Equation (4.7) for this $K$.

For electrochemical systems with rate constants of the form (4.2), the expression is somewhat simpler (see Koper et al., 1998):

$$K(e, t) = \frac{vk_B T}{\alpha e_0 \beta} \exp \left( \frac{\alpha e_0 \phi(t)}{k_B T} \right),$$

and

$$\phi(t) = \phi_0 + \beta t.$$

Even if analytical solutions to Equation (4.7) exist, the cost of calculating $w$ is at least linear in $|E|$ because the primitives of the rate constants of all enabled reactions have to be evaluated at least once in this calculation. The cost of step 2 in the tdWRsel algorithm is

$$\tau_{time} = \tau_{rand} + \tau_{incr}(|E(\sigma)|) + \tau_{fp}.$$

The cost of solving Equation (4.5) is represented by the term $\tau_{incr}(|E(\sigma)|)$.

The time per reaction for tdWRsel is given by

$$\tau_{tdWRsel} = 2\tau_{rand} + (|E(\sigma)| + n_{mod} n_{src} + 1) \tau_{pm} + 2(|E(\sigma)| + n + 1) \tau_{fp} + \tau_{incr}(|E(\sigma)|) + |E(\sigma)| \tau_{rate}. \quad (4.8)$$

For specific choices of the rate constants, both $\tau_{rate}$ and $\tau_{incr}(|E(\sigma)|)$ may be expressed in terms of $\tau_{fp}$. Without any knowledge of the rate constants this is not possible. The occurrences of $|E(\sigma)|$ in Equation 4.8 show that the time per reaction depends on the system size for tdWRsel.

Comparing Equation (4.8) to Equation (3.5), which gives the time per reaction for WRsel, we see that the coefficient of $\tau_{fp}$ is smaller for the time-dependent algorithm. On the other hand, the expression for tdWRsel contains two extra terms, associated with the cost of calculating time increments and evaluating rate constants. The former of these two dominates all other terms in the equation.
4.4 Generalizations of tdWRsel

In Chapter 3 we investigated three ways to generalize the WRsel algorithm, viz. oversampling, using an approximation of the set of enabled reactions, and selecting reactions hierarchically. The latter of these can be used for simulation of systems with time-dependent rate constants as well for systems with constant rates. The correctness of the former two depended on a property of the exponential distribution. We investigate if these techniques can be applied to systems with time-dependent rate constants too.

4.4.1 Oversampling

The idea behind oversampling is to use artificially high sampling rates for reactions and to accept selected reactions conditionally. For the case that rate constants are time-dependent, we allow the sampling rates to be time-dependent as well. The algorithm reads as follows.

Algorithm 4.2 (tdWRsel-o)

1. Initialize:
   \( t := 0; \sigma := \sigma_0; E := \emptyset; \)
   \text{for } e \in E(\sigma) \rightarrow sr : sr \geq k(e); E := E + \{(e, sr)\};\text{rof}

2. Generate a time increment \( w \) by sampling the \( \text{tdExp}(\Sigma e, sr : (e, sr) \in E : sr) \) distribution. Stop if \( w = \infty \). Increment \( t \) by \( w \).

3. (a) Select an element from \( E \), giving probability \( \frac{sr(t)}{R(t)} \) to \( (e, sr) \), where
   \( R = (\Sigma e, sr : (e, sr) \in E : sr) \).
   (b) With probability \( \frac{k(e, t)}{sr(t)} \), perform the selected reaction \( e \), and update \( E \) and \( R \).
   With probability \( 1 - \frac{k(e, t)}{sr(t)} \), do nothing.

The condition \( 'sr \geq k(e)' \) in the initialization step should be read to mean that \( sr(t) \geq k(e, t) \) for all \( t \geq 0 \). Note how the acceptance probability in step 3b now depends on the value of \( t \), the simulated time.

We verify the correctness of this algorithm by showing that Equation (4.4) holds for the simulated waiting times. The waiting time for the next reaction is the sum of a number of \( \text{tdExp}(R) \)-distributed variables, with \( R = (\Sigma e, sr : (e, sr) \in E : sr) \). The number of terms in this sum is itself stochastic. It is determined by the acceptance probabilities of the reactions. The probability that a reaction is accepted in some iteration of the algorithm in state \( \sigma \) at time \( t \) is

\[
\frac{k(E(\sigma), t)}{R(t)}.
\]
We investigate how to express the proposition

\[ W_{\text{alg}}(E, t) = w \land W_{\text{alg}}(e, t) < W_{\text{alg}}(E - \{e\}, t) \]

in terms of time increments and success probabilities of successive iterations of the algorithm.

Denote the time increment of the iteration of the algorithm that starts at time \( t \) by \( X_1 \), a random variable with a \( \text{tdExp}(R) \) distribution. Stochastic variable \( Z_1(S) \) is defined to take the value 1 if a reaction from \( S \subseteq E(\sigma) \) is selected and accepted in this iteration; it is 0 otherwise. This means that if the time increment \( X_1 \) equals \( x_1 \), then

\[
\mathbb{P}[Z_1(S, t + x_1) = 1] = 1 - \mathbb{P}[Z_1(S, t + x_1) = 0] = \frac{k(S, t + x_1)}{R(t + x_1)}.
\]

Similarly, we define \( X_2, X_3, \ldots \) and \( Z_2(S), Z_3(S), \ldots \) for the time increments, and for selection and acceptance of reactions from \( S \) for the second and third iterations, and so on. With these definitions we can rephrase \( W_{\text{alg}}(E, t) = w \land W_{\text{alg}}(e, t) < W_{\text{alg}}(E - \{e\}, t) \) as follows:

\( e \) is selected and accepted in some iteration \( j \) of the algorithm, and in iterations 1 through \( j - 1 \), no reaction was accepted. Moreover, the sum of the time increments of these \( j \) iterations equals \( w \).

If we want to express this condition in a formula, we have to condition on the time increments associated with each of the iterations. The following formula results:

\[
(\exists k : 1 \leq k
\quad :
(\exists x_1, \ldots, x_k : (\forall i : 1 \leq i \leq k : 0 \leq x_i)
\quad \land (\Sigma i : 1 \leq i \leq k : x_i) = w
\quad :
(\forall i : 1 \leq i < k
\quad :
X_i(t + x_1 + \cdots + x_{i-1}) = x_i
\quad \land Z_i(E(\sigma), t + x_1 + \cdots + x_i) = 0
\quad )
\quad )
\quad \land X_k(t + x_1 + \cdots + x_{k-1}) = x_k
\quad \land Z_k(e, t + x_1 + \cdots + x_k) = 1
\quad )
\).
\]
This formula contains several occurrences of sums of $x$'s. It can be simplified by changing dummies: we take $y_i = x_1 + \cdots + x_i$ as dummies. The condition

$$\left( \sum i : 1 \leq i \leq k : x_i \right) = w$$

is then replaced by $y_k = w$ (so $y_k$ is not a dummy, it is just a synonym for $w$). We define $y_0$ to be equal to 0. Thus, $e$ is the next reaction and has waiting time $w$ if and only if

$$\left( \exists k : 1 \leq k \right)$$

$$\left( \exists y_1, \ldots, y_{k-1} : (\forall i : 1 \leq i \leq k : y_{i-1} \leq y_i) \right)$$

$$\left( \forall i : 1 \leq i < k \right)$$

$$X_i(t + y_{i-1}) = y_i - y_{i-1}$$

$$\land Z_i(E(\sigma), t + y_i) = 0$$

$$\land X_k(t + y_{k-1}) = w - y_{k-1}$$

$$\land Z_k(e, t + w) = 1$$

(4.9)

We now have to show that the probability density of $e$ being the next reaction and having waiting time $w$ is

$$k(e, t + w) \exp(-\int_t^{t+w} k(E, v)dv).$$

(4.10)

According to Property 4.1 below, the waiting time for the next reaction ($e$ or any other reaction) has density

$$k(E, t + w) \exp(-\int_t^{t+w} k(E, v)dv).$$

**Property 4.1**

Let $X_1, X_2, \ldots$ be independent random variables, $X_i$ having a tdExp($\Lambda$) distribution. Let $\lambda$ be a function such that $\lambda(x) \leq \Lambda(x)$ for all $x \geq 0$. Let $Z_1, Z_2, \ldots$ be independent random variables, independent of the $X_i$, with

$$\mathbb{P}[Z_i(t) = 1] = 1 - \mathbb{P}[Z_i(t) = 0] = \frac{\lambda(t)}{\Lambda(t)},$$

for any $t \geq 0$. Define $W$ as follows: $W(t) = w$ if and only if

$$\left( \exists k : 1 \leq k \right)$$

$$\left( \exists y_1, \ldots, y_{k-1} : (\forall i : 1 \leq i \leq k : y_{i-1} \leq y_i) \right)$$

$$\left( \forall i : 1 \leq i < k \right)$$

$$X_i(t + y_{i-1}) = y_i - y_{i-1} \land Z_i(t + y_i) = 0$$

$$\land X_k(t + y_{k-1}) = w - y_{k-1}$$

$$\land Z_k(t + w) = 1$$

).
where $y_0 = 0$ and $y_k = w$. Then $W$ has a \( \text{tdExp}(\lambda) \) distribution.

**Proof:** See Appendix A.

A straightforward generalization of this property shows that the probability density of \( e \) being the next reaction and having waiting time \( w \) is indeed given by Equation (4.10).

The technique of oversampling has not been applied to the simulation of surface reaction systems with time-dependent rate constants before. Boardman et al. (1968) used a limited but useful form of oversampling in the simulation of electron scattering in semiconductors. They do not associate a sampling rate with every reaction, but their approach is based on the introduction of a *dummy* reaction, one that does not change the state of the system. The rate constant of this dummy reaction is set to \( \Lambda = k(E(\sigma), t) \), for some constant \( \Lambda \) exceeding the rate of change in all states and at all time instants. The new total rate of change is \( \Lambda \) in all states and at all instants in time. Generation of the waiting time for the next reaction is the same as in the case of time-independent rate constants. The cost of reaction selection is not affected by the presence of a dummy reaction. The probability of a (non-dummy) reaction being performed in state \( \sigma \) at time \( t \) is \( \frac{k(E(\sigma), t)}{\Lambda} \). If the rate of change varies over a wide range over the course of a simulation, this particular way of oversampling is not very effective.

Having shown that oversampling can be used for systems with time-dependent rate constants, we can now generalize algorithms such as URsel, WRESel, and URESel to the simulation of systems with time-dependent rates. The tdWRESel algorithm still suffers from the same shortcomings as tdWRsel: the cost of both generation of time increments and reaction selection grows (at least) linearly with \( |E(\sigma)| \).

The tdURsel and tdUREsel algorithms inherit the disadvantage of possibly low success probabilities from their time-independent counterparts. In order to maximize the success probability it is necessary to keep track of the maximum of the rates of the enabled reactions or reaction effects. As rate constants change over time, this involves recalculating the rate constants of all enabled reactions in every iteration of the algorithm. If there is a reaction type \( RT \) that has a higher rate constant than any other reaction type, at every instant in time, then \( k(RT) \) can be used as the sampling rate for each reaction. In this case the success probability is below optimal, but only two rate constants have to be evaluated in every iteration: the rate constant of the selected reaction and that of \( RT \).

### 4.4.2 Approximate representation of the set of enabled reactions

In the tdWRsel algorithm, a significant part of the amount of work per reaction consists of removing reactions from \( E \) when they become disabled. It can be advantageous not to remove reactions when they become disabled, but to leave them in \( E \). Section 3.3.2
contains an explanation of the adaptations that had to be made to WRsel to distinguish eligible reaction from ineligible ones if this path is followed. These adaptations can also be made to tdWRsel. The question is whether Equation (4.4) still holds if this is done. It turns out that it does. The proof requires a generalization of Property 4.1, as we will see below.

Algorithm 4.3 (tdWRsel-a)

1. Initialize:
   \( t := 0; \sigma := \sigma_0; E := \emptyset; \)
   \( \text{for } l \in \mathcal{L} \rightarrow l\text{m}[l] := 0; \text{rof} \)
   \( \text{for } e \in E(\sigma) \rightarrow E := E + \{(e, 0)\}; \text{rof} \)

2. Generate a time increment \( w \) by sampling the \( \text{tdExp}(k(E)) \) distribution. Stop if \( w = \infty \). Increment \( t \) by \( w \).

3. Select and remove an element from \( E \), giving probability \( \frac{k(e, t)}{k(E, t)} \) to \((e, t_{en})\). If the selected reaction is eligible for execution, then update \( \sigma \) and \( \text{lm} \); add any newly enabled reactions to \( E \), using the number of iterations performed as the enabling time.

As in the previous section, the waiting time for the next reaction is the sum of a number of \( \text{tdExp} \)-distributed variables. These variables have identical distributions for the tdWRsel-o algorithm because \( E \) does not change in unsuccessful iterations of tdWRsel-o. Now, however, the set \( E \) changes in every iteration. This means that the waiting time for the next reaction is the sum of a number of \( \text{tdExp} \)-distributed variables with different parameters. Similarly, the success probability changes as ineligible reactions are removed from \( E \). The proposition that \( e \) is the next reaction and has waiting time \( w \) is still equivalent to (4.9), but the random variables have different distributions. We consider the sequence of values that \( E \) takes between successful iterations, denoted by \( E^1, E^2, \ldots \). The random variable describing the time increment in the \( i \)\textsuperscript{th} iteration, \( X_i \), has a \( \text{tdExp}(k(E^i)) \) distribution. For any \( S \subseteq E(\sigma) \) and \( t \geq 0 \) the distribution of \( Z_i \) is given by

\[
P[Z_i(S, t) = 1] = 1 - P[Z_i(S, t) = 0] = \frac{k(S, t)}{k(E^i, t)}.
\]

Property 4.2 shows that the waiting time for the next reaction has a \( \text{tdExp}(k(E(\sigma))) \) distribution. It is easily adapted to show that the probability of \( e \) being the next reaction and having waiting time \( w \) is given by (4.10).

Property 4.2

Let \( X_1, X_2, \ldots \) be independent random variables, \( X_i \) having a \( \text{tdExp}(\lambda_i) \) distribution. Let
\( \lambda \) be a function such that \( \lambda(x) \leq \lambda_i(x) \) for all \( i \) and for all \( x \geq 0 \). Let \( Z_1, Z_2, \ldots \) be independent random variables, independent of the \( X_i \), with

\[
\mathbb{P}(Z_i(t) = 1) = 1 - \mathbb{P}(Z_i(t) = 0) = \frac{\lambda(t)}{\lambda_i(t)},
\]

for any \( t \geq 0 \). Define \( W \) as follows: \( W(t) = w \) if and only if

\[
(\exists k : 1 \leq k \\
\quad : (\exists y_1, \ldots, y_{k-1} : (\forall i : 1 \leq i \leq k : y_{i-1} \leq y_i) \\
\quad \quad : (\forall i : 1 \leq i < k \\
\quad \quad \quad : X_i(t + y_{i-1}) = y_i - y_{i-1} \land Z_i(t + y_i) = 0) \\
\quad \quad \quad \land X_k(t + y_{k-1}) = w - y_{k-1} \\
\quad \quad ) \land Z_k(t + w) = 1
\]

where \( y_0 = 0 \) and \( y_k = w \). If there are \( l \geq 0 \) and \( \Lambda \) such that \( \lambda_i = \Lambda \) for all \( i \geq l \), then \( W \) has a \( \text{tdExp}(\lambda) \) distribution.

**Proof:** See Appendix A. \( \square \)

The time-dependent version of the \( \text{WRsel-a} \) algorithm has not been described before. Its greatest benefit is a reduction in the cost of deletion of disabled reactions. The generation of time increments and selection of reactions are still done in the same way as in \( \text{tdWRsel} \). Under the assumption that a reaction enables and disables \( n \) reactions on average, we get the following expression for the time per reaction:

\[
\tau_{\text{tdWRsel-a}} = n(\tau_{\text{time}}(|E|) + \tau_{\text{wset}}(|E|) + \tau_{\text{acc}}) + \tau_{\text{react}} + \tau_{\text{enab}}.
\]

The term \( \tau_{\text{acc}} \) represents the cost of checking whether the selected reaction is eligible for execution and deleting it from \( E \), so

\[
\tau_{\text{acc}} = \tau_{\text{pm}} + \tau_{\text{fp}}.
\]

The way the other terms are broken down into the cost of more elementary operations is the same as in Section 4.3. The result is

\[
\tau_{\text{tdWRsel-a}} = 2n\tau_{\text{rand}} + (n + n_{\text{mod}}n_{\text{src}} + 2)\tau_{\text{pm}} + 2n(|E| + 2)\tau_{\text{fp}}
\]

\[
+ n\tau_{\text{incr}}(|E|) + n|E|\tau_{\text{rate}}.
\]

(4.11)

The structure of this formula is as expected: the amount of pattern matching per reaction performed has decreased, at the cost of extra floating-point operations, extra rate-constant evaluations, and extra time increments.
4.4.3 Selecting reactions in multiple stages

Using the fact that reactions of the same type have the same rate constants, we can reduce the cost of generating time increments and selection of reactions. The selection cost is reduced by selection of reactions in two stages: first a reaction type is selected, then a reaction of that type is selected. The former selection is weighted, and the latter uniform. The tdWTselULsel algorithm reads as follows.

Algorithm 4.4 (tdWTselULsel)

1. Initialize:
   \[ t := 0; \sigma := \sigma_0; \]
   Construct a set \( E \), the elements of which are sets. Each of those sets contains the enabled reactions of a reaction type: \( E = \{RT : RT \in \mathcal{R}T : s(RT)\} \).

2. Generate a time increment \( w \) by sampling the tdExp(\( k(E) \)) distribution. Stop if \( w = \infty \). Add \( w \) to \( t \).

3. Select a reaction type – i.e., select an element from \( E \), giving probability
   \[
   \frac{|s(RT)|k(RT, t)}{k(E, t)}
   \]
   to type \( RT \). Having selected \( RT \), select one of its elements uniformly and delete this element from \( s(RT) \). Perform the selected reaction, delete any disabled reactions and add newly enabled reactions to \( E \).

The only difference between tdWRsel and tdWTselULsel is that in the latter reactions are selected in two stages. The probability of a particular reaction being selected has not changed, however, so the algorithm is correct.

As mentioned in Section 4.3, this algorithm was described by Prados et al. (1997) for the simulation of Ising spin systems. Kurosawa (1966) already applied the tdWTselULsel algorithm to the simulation of electron transport in semiconductors. Kurosawa does not explain why the method is correct.

The selection of a reaction type is weighted. As the weights of the types change in every iteration of the algorithm, we do not use a tree representation for \( E \). Instead, \( E \) is represented by an array of length \( |\mathcal{R}T| \). Each of the array elements contains a reference to an array that contains the enabled reactions of one type. Furthermore, the number of enabled reactions of each type is recorded. We analyze the time per reaction of the tdWTselULsel algorithm for this representation. The time per reaction is made up of five terms:

\[
\tau_{tdWTselULsel} = \tau_{time} + \tau_{wsel}(|\mathcal{R}T|) + \tau_{usel} + \tau_{react} + \tau_{disab} + \tau_{enab}.
\]
Generating a time increment requires solving \( w \) from

\[
(\Sigma_{RT} : RT \in \mathcal{RT} : m_{RT}(E(\sigma))K(RT, t + w)) =
(\Sigma_{RT} : RT \in \mathcal{RT} : m_{RT}(E(\sigma))K(RT, t)) - \ln u,
\]

with \( K \) defined as in Section 4.3. Evaluation of \( K(\sigma, t) \) is now an operation linear in \( |\mathcal{RT}| \), whereas it was linear in \( |E(\sigma)| \) for \( \text{tdWRSel} \). The cost of generating a time increment and updating the current time is

\[
\tau_{\text{time}} = \tau_{\text{rand}} + \tau_{\text{incr}}(|\mathcal{RT}|) + \tau_{fp}.
\]

The weighted selection of a reaction type from involves computing the rate constants of all reaction types and multiplying them by \( m_{RT}(E(\sigma)) \). Cumulative rate constants are computed and a type is selected using a random number. The cost of these operations is

\[
\tau_{\text{wsel}}(|\mathcal{RT}|) = \tau_{\text{rand}} + |\mathcal{RT}|\tau_{\text{rate}} + (3|\mathcal{RT}| + 1)\tau_{fp}.
\]

The uniform selection of a reaction has cost \( \tau_{\text{usel}} = \tau_{\text{rand}} + 2\tau_{fp} \) and the cost of updating the state is \( \tau_{pm} \), as before. Finding disabled reactions is also done as before. Deleting a reaction of some type \( RT \) amounts to moving an array element and decrementing the variable in which the number of enabled reactions of type \( RT \) is stored, so

\[
\tau_{\text{disab}} = |E(\sigma)|\tau_{pm} + 2n\tau_{fp}.
\]

The cost of adding a newly enabled reaction to \( E \) is \( 2n\tau_{fp} \) as well, while checking which reactions have become enabled requires \( n_{mod}n_{src} \) pattern matches. Therefore,

\[
\tau_{\text{enab}} = n_{mod}n_{src}\tau_{pm} + 2n\tau_{fp}.
\]

Adding the contributions of all these operations, we obtain

\[
\tau_{\text{tdWRSelULsel}} = 3\tau_{\text{rand}} + (|E(\sigma)| + n_{mod}n_{src} + 1)\tau_{pm} + (5n + 2|\mathcal{RT}| + 4)\tau_{fp} + \tau_{\text{incr}}(|\mathcal{RT}|) + |\mathcal{RT}|\tau_{\text{rate}}.
\]  

(4.12)

Only the term \(|E(\sigma)|\tau_{pm}\) depends on the system size. This term represents the cost of deletion of disabled reactions from \( E \). The contributions of generating time increments and reaction selection to the time per reaction are independent of the system size.

Reactions may also be grouped in other ways, for instance by their enabling location. In order to keep the cost of generating time increments independent of the size of the system, the number of enabled reactions of every type has to be recorded no matter how reactions are grouped.
4.5 Combining techniques

Having established that the techniques discussed in Chapter 3 are applicable to the simulation of systems with time-dependent rate constants, the same combinations of techniques can be used to obtain other simulation algorithms. In this section some of the algorithms thus obtained are analyzed with respect to the time per reaction.

Applying the technique of leaving disabled reactions in $E$ to the tdWTselULsel algorithm, we can eradicate the last factor that depends on the system size from the time per reaction.

Algorithm 4.5 (tdWTselULsel-a)

1. Initialize:
   $t := 0; \sigma := \sigma_0$;
   $\text{for } l \in \mathcal{L} \rightarrow \text{lm}[l] := 0; \text{rof}$
   Construct a set $E$ that contains sets of pairs: $E = \{RT : RT \in \mathcal{RT} : s(RT)\}$. Each element $s(RT)$ contains the enabled reactions of type $RT$, and an enabling time for each reaction.

2. Generate a time increment $w$ by sampling the $tdExp(k(E))$ distribution. Stop if $w = \infty$. Increment $t$ by $w$.

3. Select a reaction type. I.e., select an element from $E$, giving probability
   \[
   \frac{|s(RT)|k(RT, t)}{k(E, t)}
   \]
   to type $RT$. Having selected $RT$, select one of the elements $(l, t_{en})$ of $s(RT)$ uniformly. Remove $(l, t_{en})$ from $s(RT)$. If $(RT, l, t_{en})$ is eligible, then update $\sigma$ and $\text{lm}$; add newly enabled reactions to $E$, using the number of iterations performed as the enabling time.

With respect to the correctness of this algorithm we note that Property 4.2 applies, and omit further details. The algorithm is new and is the first algorithm for the simulation of systems with time-dependent rate constants for which the time per reaction is independent of the system size. We analyze the time per reaction in detail to substantiate this claim.

The expected number of iterations per reaction is $n$. In each of these iterations the time is updated, a reaction is selected by successive selections of a reaction type and a location, and the reaction’s eligibility is checked. Once in $n$ iterations, the reaction is performed and newly enabled reactions are added to $E$. Therefore the time per reaction for tdWTselULsel-a can be written as follows:

$$
\tau_{tdWTselULsel-a} = n(\tau_{time} + \tau_{wsel(|\mathcal{RT}|)} + \tau_{sel} + \tau_{acc} + \tau_{react} + \tau_{enab}).
$$ (4.13)
The performance analyses of tdWRsel-a and tdWTselULsel already contain the information needed to express the right hand side of Equation (4.13) in terms of the cost of our basic operations. A point worth noting is that the cost of dealing with disabled reactions is captured by the term \( \tau_{\text{acc}} \), which denotes the cost of checking whether a selected reaction is eligible. We list the expressions for the terms on the right hand side of Equation (4.13) and the resulting expression for \( \tau_{\text{tdWTselULsel-a}} \):

\[
\tau_{\text{time}} = \tau_{\text{rand}} + \tau_{\text{incr}}(|\mathcal{RT}|) + \tau_{fp},
\]
\[
\tau_{\text{wset}}(|\mathcal{RT}|) = \tau_{\text{rand}} + |\mathcal{RT}| \tau_{\text{rate}} + (3|\mathcal{RT}| + 1) \tau_{fp},
\]
\[
\tau_{\text{use}} = \tau_{\text{rand}} + 2\tau_{fp},
\]
\[
\tau_{\text{acc}} = \tau_{pm} + \tau_{fp},
\]
\[
\tau_{\text{react}} = 2\tau_{pm},
\]
\[
\tau_{\text{enab}} = n_{\text{mod}} n_{\text{src}} \tau_{pm} + 2n \tau_{fp},
\]

and

\[
\tau_{\text{tdWTselULsel-a}} = 3n \tau_{\text{rand}} + (n + n_{\text{mod}} n_{\text{src}} + 2) \tau_{pm} + (3|\mathcal{RT}| + 7)n \tau_{fp}
\]
\[
+ n|\mathcal{RT}| \tau_{\text{rate}} + n \tau_{\text{incr}}(|\mathcal{RT}|). \tag{4.14}
\]

Equation (4.14) indeed shows that the time per reaction is independent of the system size. It also shows that the performance of tdWTselULsel-a degrades severely if the simulated model becomes more complex. As models grow, in terms of the size of the reaction patterns and the number of reaction types, \( n \) increases. The cost of computing time increments and rate constants grows with the product of \( n \) and \(|\mathcal{RT}|\). These two terms limit the use of the algorithm to models of moderate complexity.

In the second part of this section we investigate whether the time-dependent version of WSTselULsel-a performs better in this respect.

Algorithm 4.6 (tdWSTselULsel-a)

1. Initialize:
   \[ t := 0; \sigma := \sigma_0; \]
   \[ \text{for } l \in \mathcal{L} \rightarrow \text{l}m[l] := 0; \text{rof} \]
   Compute the set of supertypes from \( \mathcal{RT} \). Compute for each supertype \( SRT \) the sampling rate function, \( k_{SRT} \), by taking the maximum sum of rates of reactions of supertype \( SRT \) that can be enabled at the same time at a unit cell (see below).
   Construct a set \( E \), of which each element contains the locations of the enabled reactions (in \( \sigma_0 \)) of a supertype. Store an enabling time, zero, with each of the locations.

2. Generate a time increment \( w \) by sampling the
   \[ \text{tdExp}(\Sigma SRT : SRT \in \mathcal{RT} : m_{SRT}(E)k_{SRT}) \]
4.5 Combining techniques

distribution. Stop if \( w = \infty \). Increment \( t \) by \( w \).

3. (a) Select a supertype from \( E \), supertype \( SRT \) with a probability proportional to the product of its sampling rate and the number of elements in the corresponding group of \( E \). Having selected \( SRT \), select a unit cell, say \( l \), in the corresponding group of \( E \) uniformly.

(b) If the selected reaction effect is not eligible, then remove it from \( E \).

(c) If the selected reaction effect is eligible for execution, compute its rate constant \( k \) in the current state and at the current time by summing the rates of the reactions in \( SRT \) currently enabled at at unit cell \( l \). With probability \( \frac{k}{k_{SRT}(t)} \), remove \((SRT, l)\) from \( E \), update \( \sigma \) and \( lm \), and add newly enabled reaction effects to \( E \), using the iteration count as the enabling time.

In general, finding a sampling rate for a supertype is a difficult problem. It involves computing, for every supertype \( SRT \), a function \( k_{SRT} \) such that for all \( t \), \( \sigma \) and \( l \), \( k_{SRT}(t) \) exceeds the sum of the rate constants of the reactions of supertype \( SRT \) enabled at location \( l \) in \( \sigma \) at time \( t \). In practice, it will often be the case that only one reaction of a supertype can be enabled at a location, and that the rate constants all have the same time dependence. In this case, every supertype \( SRT \) contains a reaction type \( RT \) of which the rate constant \( k(RT, t) \) is higher than that of the other elements of \( SRT \) for all \( t \).

The correctness proof of the tdWTselULsel-a algorithm relies, once more, on Property 4.2. We do not elaborate on the proof. The performance analysis of the algorithm is similar to that of the WSTselULsel-a algorithm, but has some extra complications. Recall that the time per trial for WSTselULsel-a depends on the expected acceptance probability for eligible reactions. This acceptance probability depended on \( \sigma \) for WSTselULsel-a. When rate constants are time-dependent, the acceptance probabilities depend on the current time as well and the expected value cannot be calculated without knowledge of the time dependence of the rate constants. For reasons of simplicity, we assume that the dependence of acceptance probabilities on \( t \) can be ignored in the performance analysis.

Under this assumption, the expected number of trials between successive reactions is \( \frac{n^*}{p_{acc}(\sigma)} \). As before, \( n^* \) denotes the number of reaction effects enabled by performing a reaction, and \( p_{acc}(\sigma) \) gives the expected value of the probability that an eligible reaction is accepted upon selection. The time per reaction is equal to \( \frac{n^*}{p_{acc}(\sigma)} \) times the time per trial.

In every trial the current time is updated. This involves generating a time increment and a floating-point addition. Generating a time increment is cheaper for tdWSTselULsel-a than for tdWTselULsel-a if the number of supertypes is smaller than the number of reaction types. The cost of updating the time is \( \tau_{rand} + \tau_{incv}(|SRT|) + \tau_{fp} \).
A reaction effect is selected and checked for eligibility in every trial as well. The selection consists of the weighted selection of a supertype and the uniform selection of a unit cell from an array. Thus, the cost of selection of a reaction effect is

$$ \tau_{\text{rand}} + |\delta \mathcal{RT}| \tau_{\text{rate}} + (3|\delta \mathcal{RT}| + 1) \tau_{fp} + \tau_{\text{rand}} + 2 \tau_{fp} . $$

As usual, the cost of the eligibility check we take to be $\tau_{pm}$.

On average, an ineligible reaction effect is selected in $n^* - 1$ out of every $n^*$ trials. If this is the case, the reaction effect is deleted from $E$. This amounts to deleting it from an array and updating the number of reaction effects in the array, at cost $2\tau_{fp}$.

Once in every $n^*$ trials, an eligible reaction effect is selected. In this case, the actual rate constant is computed and an acceptance probability is computed. After having computed the actual rate constant of the selected reaction effect, at cost

$$ \frac{|\mathcal{RT}|}{|\delta \mathcal{RT}|} (\tau_{pm} + \tau_{rate}) ,$$

the acceptance probability is computed as the quotient of the actual rate constant and the rate constant of the supertype, at cost $\tau_{fp}$. Accepting or rejecting the selected reaction effect is done in the same way as in the case that rate constants are time-independent. The cost is $\tau_{\text{rand}} + \tau_{fp}$.

If the reaction effect is accepted, then $\sigma$ and $lm$ are updated, the reaction effect is deleted from $E$, and newly enabled reaction effects are inserted in $E$. The cost of these operations is

$$ 2\tau_{pm} + 2\tau_{fp} + n_{\text{mod}}^2 |\delta \mathcal{RT}| \tau_{pm} + 2n^* \tau_{fp} .$$

The expected time per trial is found by adding the contributions of these operations, each contribution multiplied by the expected probability of the operation being performed in a trial. The time per reaction is obtained by multiplying the time per trial by $\frac{n^*}{\tau_{\text{acc}(\sigma)}}$. The resulting expression is

$$ \tau_{\text{dWTSelULSel-a}} = \frac{3n^* + 1}{\tau_{\text{rand}}} (n^* + \tau_{\text{rate}}\frac{|\mathcal{RT}|}{|\delta \mathcal{RT}|} \frac{1}{\tau_{\text{acc}(\sigma)}} + n_{\text{mod}}^2 |\delta \mathcal{RT}| + 2) \tau_{pm}$$

$$ + (3(|\delta \mathcal{RT}| + 2) \frac{n^*}{\tau_{\text{acc}(\sigma)}} + 2(n^* + 1)) \tau_{fp}$$

$$ + \frac{n^*}{\tau_{\text{acc}(\sigma)}} \tau_{\text{incr}}(|\delta \mathcal{RT}|)$$

$$ + (n^* |\delta \mathcal{RT}| + \frac{|\mathcal{RT}|}{|\delta \mathcal{RT}|} \frac{1}{\tau_{\text{acc}(\sigma)}} \tau_{\text{rate}} .$$

(4.15)
4.6 A second class of algorithms

Again, comparison of the performance of tdWTselULsel-a and tdWSTselULsel-a is not easy. One thing is apparent from Equation (4.15): the contributions of the terms associated with incrementing the time and evaluating rate constants have a better growth behavior. They are $O(n^*|\mathcal{R}|)$, as opposed to the $O(n|R|)$ terms that occur in the time per reaction for tdWTselULsel-a. We should not get our hopes up too high, however, keeping in mind the possibly low acceptance probabilities of selected reaction effects as seen in Section 3.5.3 where we discussed the time-independent version of this algorithm.

Both algorithms discussed in this section may be adapted so no reaction effects are stored in a data structure, leading to the time-dependent variants of the WTSelULtrial and WSTselULtrial algorithms.

4.6 A second class of algorithms

The class of algorithms based on scheduling (groups of) reactions or reaction effects can also be generalized to the case of time-dependent rate constants. This class was rejected in Chapter 3 as the algorithms in it are generally slower than algorithms in which reactions and occurrence times are selected independently, if rate constants are time-independent. If, however, rate constants are time-dependent, calculating the occurrence time of the next reaction gives rise to a significant increase in the time per reaction. We investigate what the effects of the time dependence are on the performance of algorithms based on scheduling reactions.

The first algorithm we discuss, is tdRsched, the time-dependent version of the reaction scheduling algorithm.

Algorithm 4.7 (tdRsched)

1. Initialize:
   \[ t := 0; \sigma := \sigma_0; E := \emptyset; \]
   for \( e \in E(\sigma) \)
   \[ \rightarrow \text{sample } w \text{ from } \text{Exp}(k(e)); \]
   \[ E := E + \{(e, w)\}; \]
   \[ \text{rof} \]

2. Select the element \((e, t_{occ})\) from \(E\) with minimum occurrence time \(t_{occ}\). Update the state, set the time to \(t_{occ}\), and remove all disabled reactions from \(E\). Generate a waiting time \(w\) for every newly enabled reaction \(e\). Add \((e, t + w)\) to \(E\).
The only difference between this algorithm and the Rsched algorithm is that here occurrence times for reactions are generated by sampling a time-dependent exponential distribution, instead of one with a constant rate. The correctness of Rsched does not depend on the probability distribution of the time increments, so the time-dependent version is correct too.

The time per reaction for the Rsched algorithm is

\[
\tau_{tdRsched} = \tau_{pqdelmin}(|E(\sigma)|) + \tau_{pm} + \tau_{fp} \\
+ |E(\sigma)| \tau_{pm} + (n - 1) \tau_{pqdel}(|E(\sigma)|) \\
+ n_{mod} n_{src} \tau_{pm} + n(\tau_{rand} + \tau_{incr}(1) + \tau_{fp} + \tau_{pqins}(|E(\sigma)|)).
\]

This equation contains a new parameter, \(\tau_{pqdelmin}\), which denotes the cost of finding and deleting the minimum element of a priority queue. Comparing Equations (4.8) and (4.16), we notice three things. First, both equations contain an \(O(|E(\sigma)|)\) term. Second, \(\tau_{tdRsched}\) contains no \(\tau_{rate}\) terms, while \(\tau_{tdWRsel}\) contains a term \(|E(\sigma)|\tau_{rate}\). Third, there is a difference in the computations for generating time increments. In the tdRsched algorithm this is done by generating waiting times for individual reactions, whereas a waiting time for the next reaction is generated in the tdWRsel. This has an important consequence. Instead of having to solve \(w\) from Equation (4.5) (see page 99), the simpler

\[
\int_{t}^{t+w} k(e, v) dv = -\ln u
\]

is used. Efficient numerical procedures exist for the case that rate constants have the form (4.1) (see, e.g., Press et al., 1993, Sec. 6.3). If rate constants have the form (4.2), then a closed expression for \(w\) exists. The amount of time spent on computing time increments is only proportional to \(n\) for tdRsched, while it grows at least linearly with \(|E(\sigma)|\) for tdWRsel. As the worst-case cost of the operations on the priority queue is \(O(\log_2 |E(\sigma)|)\), we conclude that tdRsched is more efficient a simulation algorithm than tdWRsel. This seems sufficient reason to discuss the time-dependent variants of the TschedULsel-a and STschedULsel-a algorithms compare their performance to that of tdWTselULsel-a and tdWSTselULsel-a.

The tdTschedULsel-a algorithm reads as follows.

**Algorithm 4.8 (tdTschedULsel-a)**

1. Initialize:
   \[ t := 0; \sigma := \sigma_0; E := \emptyset; \]
   \[ \text{for } l \in \mathcal{L} \rightarrow \text{lm}[l] := 0; \text{rof} \]

   For each reaction type \(RT\) compute the enabled reactions and store them, with enabling time zero, in a set \(s(RT)\). Sample \(w\) from the tdExp\((s(RT))|k(RT))\) distribution, and add \((s(RT), w)\) to \(E\).
2. Select the elements \((s(RT), t_{occ})\) from \(E\) with minimum occurrence time. Set \(i\) to \(t_{occ}\). Select an element \((l, t_{en})\) from \(s(RT)\) uniformly and delete it from \(s(RT)\). If the selected reaction is eligible, then update \(\sigma\) and \(lm\), compute the newly enabled reactions and insert them into the appropriate subsets of \(E\). Reschedule the subsets of \(E\) that have changed.

The correctness of this simulation algorithm relies on the lack of memory of the tdExp family of distributions and on the fact that the minimum of independent tdExp distributions is again a tdExp distribution. We omit the details.

Under the usual assumption that one out of every \(n\) iterations is successful, we get the following expression for the time per reaction:

\[
\tau_{tdTschedULsel-a} = n(\tau_{pqfindmin} + \tau_{use1} + \tau_{fp} + \tau_{acc} + 2\tau_{fp} + \tau_{resched(|RT|)}) + 2\tau_{pm} + n_{mod}n_{src}\tau_{pm} + n(2\tau_{fp} + \tau_{resched(|RT|)}).
\]

The first line gives the contributions of selecting a reaction, updating the time, checking the reaction’s eligibility, deleting it from \(E\) and rescheduling the subset of \(E\) associated with the type of the selected reaction. The second line give the contributions of the operations performed if the selected reaction is eligible: \(\sigma\) and \(lm\) are updated, newly enabled reactions are checked and \(n\) reactions are inserted into \(E\), giving rise to (at most) \(n\) rescheduling operations on the priority queue.

The cost of finding the reaction type with minimum occurrence time in the priority queue is constant; we assume it to be equal to the cost of a floating-point operation. Rescheduling an element of a priority queue may be done by deleting it, computing a new priority (occurrence time), and inserting it again. The cost of both deletion and insertion of elements is logarithmic in the size of the priority queue. Computing a new occurrence time has cost \(\tau_{rand} + \tau_{incr}(1) + \tau_{fp}\). We obtain

\[
\tau_{tdTschedULsel-a} = 3n\tau_{rand} + 2n\tau_{incr}(1) + (n_{mod}n_{src} + n + 2)\tau_{pm} + 10n\tau_{fp} + 2n(\tau_{pqdet(|RT|)} + \tau_{pqins(|RT|)})
\]

for the time per reaction. We compare this to the time per reaction for tdWTselULsel-a, as given by Equation (4.14) on page 110. The first two terms are the same, and tdTschedULsel-a scores better on the third if there is more than one reaction type. Next we see that in the tdTschedULsel-a algorithm, the amount of time spent on generating time increments does not have the dependency on \(|RT|\) that it has in tdWTselULsel-a. Finally, the \(n|RT|\tau_{rate}\) term of tdWTselULsel-a has been replaced by the term that accounts for 2n logarithmic operations on a priority queue of size \(|RT|\). This means that the tdTschedULsel-a algorithm can be expected to perform significantly better than tdWTselULsel-a. It also performs significantly better than the tdRschd algorithm, the performance of which depends on \(|E(\sigma)|\).
The tdTschedULsel-a algorithm has the property that the time per reaction performed is independent of the system size. No algorithm for the simulation of systems with time-dependent rate constants in the literature has this property. The tdTschedULsel-a algorithm is arguably the most important new algorithm described in this thesis.

The TschedULtrial algorithm can also be generalized to deal with time-dependent rate constants. Its advantages over tdWTselULtrial are the same as those of tdTschedULsel-a over tdWTselULsel-a.

The last simulation algorithm we discuss, apart from the ones discussed in the literature review of Section 4.7, is the time-dependent variant of the STschedULsel-a algorithm. It reads as follows.

**Algorithm 4.9** (tdSTschedULsel-a)

1. Initialize:
   \[ \sigma := \sigma_0; \ t := 0; \ E := \emptyset; \]
   \[ \text{for } l \in L \rightarrow lm[l] := 0; \ 	ext{rof} \]
   Compute the set of supertypes from \( RT \). Compute for each supertype \( SRT \) the sampling rate, \( k_{SRT} \), by taking the maximum sum of rate constants of reactions of that supertype that can be enabled at the same time at a unit cell.
   Compute for each supertype \( SRT \) the enabled reaction effects and store them, with enabling time zero, in a set \( s(SRT) \). Sample \( w \) from the \( \text{tdExp}(m_{SRT}(E)k_{SRT}) \) distribution and add \( (s(SRT), w) \) to \( E \).

2. (a) Select the element \((s(SRT), t_{occ})\) from \( E \) with minimum occurrence time. Set \( t \) to \( t_{occ} \). Select an element \((l, t_{en})\) from \( s(SRT) \) uniformly.
   (b) If the selected reaction effect is not eligible, then remove it from \( E \) and reschedule supertype \( SRT \).
   (c) If the selected reaction effect is eligible, then compute its rate constant \( k \) in the current state and at the current time. With probability \( \frac{k}{k_{SRT}(E)} \), remove the selected reaction from \( E \), update \( \sigma \) and \( lm \), add newly enabled reaction effects to the appropriate subsets of \( E \) using the iteration count as enabling time, and reschedule supertypes for which the number of reactions in \( E \) has changed.

The correctness argument for this algorithm is almost the same as that for Gsched-ao. The most notable difference is that Property 4.2 is used instead of Property 3.2.

We again assume that the average acceptance probability in Step 2c does not depend on \( t \). The average number of iterations per reaction is \( \frac{R^*}{\mu_{occ}(\sigma)} \). We calculate the computational cost per iteration for the algorithm, and then the time per reaction.
4.6 A second class of algorithms

In every iteration, the minimum element of a priority queue is accessed, the time is updated, a reaction is selected uniformly and it is checked whether the selected reaction is eligible. The cost of these operations is

\[ \tau_{pqadm} + \tau_{usel} + \tau_{fp} + \tau_{pm} = \tau_{rand} + \tau_{pm} + 4\tau_{fp} \cdot \]

The selected reaction is ineligible with probability \( \frac{n^*-1}{n^*} \) and eligible with probability \( \frac{1}{n^*} \). If it is ineligible, it is removed from \( E \) and the supertype it belongs to is rescheduled, at cost

\[ 2\tau_{fp} + \tau_{resched}(\mid \delta RT \mid) \]
\[ = \tau_{rand} + 3\tau_{fp} + \tau_{incr}(1) + \tau_{pqins}(\mid \delta RT \mid) + \tau_{pqdel}(\mid \delta RT \mid) \cdot \]

After selection of an eligible reaction, a rate constant and acceptance probability are computed. Computed the reaction effect's rate has expected cost

\[ \frac{|RT|}{|\delta RT|}(\tau_{pm} + \tau_{rate}) \]

and computing the acceptance probability adds a floating-point operation. The cost of the decision to accept or reject the selected reaction effect is \( \tau_{rand} + \tau_{fp} \).

Upon acceptance, the iteration continues by updating the state and the lattice modification times at cost \( 2\tau_{pm} \). The reaction effect is deleted from \( E \) at cost \( 2\tau_{fp} \) and its supertype is rescheduled, the latter operation adding \( \tau_{resched}(\mid \delta RT \mid) \) to the cost of the iteration. The lattice is inspected for newly enabled reaction effects, requiring \( n^* \mid \delta RT \mid \) pattern matches. By assumption, \( n^* \) newly enabled reaction effects are found. These are added to \( E \) and the supertypes to which reaction effects are added rescheduled. Adding \( n^* \) effects to \( E \) and rescheduling \( n^* \) supertypes has cost

\[ n^*\tau_{rand} + 3n^*\tau_{fp} + n^*\tau_{incr}(1) + n^*\tau_{pqins}(\mid \delta RT \mid) + n^*\tau_{pqdel}(\mid \delta RT \mid) \cdot \]

The expected amount of time spent on one iteration of the algorithm is

\[ (2 + \frac{n^*+1}{n^*} \cdot \frac{p_{acc}(\sigma)}{n^*})\tau_{rand} \]
\[ +(1 + \frac{1}{n^*} \cdot |RT| + \frac{p_{acc}(\sigma)}{n^*} \cdot (n^* |\delta RT| + 2))\tau_{pm} \]
\[ +(7 - \frac{1}{n^*} + 3p_{acc}(\sigma)(1 + \frac{1}{n^*})\tau_{fp} \]
\[ +(1 - \frac{1}{n^*} + p_{acc}(\sigma))(\tau_{incr}(1) + \tau_{pqins}(\mid \delta RT \mid) + \tau_{pqdel}(\mid \delta RT \mid)) \]
\[ + \frac{1}{n^*} \frac{|RT|}{|\delta RT|} \tau_{rate}, \]
which, when multiplied by \( \frac{n^*}{p_{acc}(\sigma)} \), gives the following expression for the expected time per reaction:

\[
\tau_{tdStschduLsel-a} = (2 \frac{n^*}{p_{acc}(\sigma)} + n^* + 1)\tau_{rand} \\
+ (n^* + \frac{|\Delta RT|}{|\Delta RT|} \frac{1}{p_{acc}(\sigma)} + n_{mod}^2|\Delta RT| + 2)\tau_{pm} \\
+ (\frac{7n^* - 1}{p_{acc}(\sigma)} + 3(n^* + 1))\tau_{fp} \\
+ (\frac{n^* - 1}{p_{acc}(\sigma)} + n^*)(\tau_{incr}(1) + \tau_{pqins}|\Delta RT| + \tau_{pqdel}|\Delta RT|) \\
+ \frac{1}{p_{acc}(\sigma)} \frac{|\Delta RT|}{|\Delta RT|}\tau_{rate} .
\]

(4.18)

The formula is too complicated to make an easy comparison to other algorithms possible. Two aspects are noteworthy. The first is that the formula does not contain any terms that depend on the system size. Second, in contrast to Equation (4.17), which gives the time per reaction for tdTschedULsel-a, this formula contains a term \( \tau_{rate} \). Rate computations being computationally expensive, the occurrence of this term shows that the performance that may be gained by dealing with supertypes instead of types is at least partly canceled by the additional rate-constant computations.

### 4.7 Literature review and conclusions

We investigated the problem of simulating surface processes with time-dependent rate constants in this chapter. This problem has been investigated before, and simulations techniques have been proposed by various researchers. Until recently, the only application found in the literature was the simulation of temperature-programmed desorption and reaction systems. Recently, interest for time-dependent simulation has risen in the field of electrochemistry. In this section we discuss some of the algorithms found in the literature for the simulation of time-dependent surface reactions.

The first paper describing an algorithm for dynamic Monte Carlo simulation of TPD experiments is one by Sales and Zgrablich (1987a). The algorithm proposed by these authors is a fixed time step algorithm, or, because the temperature rises linearly with time, a fixed temperature step algorithm. At every temperature, all particles present on the surface are visited. For each particle the probability of desorption in the current time interval is computed. In this computation it is assumed that the rate constant for desorption does not change over the course of one simulation step. This assumption avoids having to integrate the rate function. The particle desorbs with the calculated probability. After all particles have been visited, the temperature is increased by some fixed amount.
We saw in Section 3.8 that the accuracy of fixed step algorithms depends strongly on the step size used. The criteria for obtaining accurate simulation results formulated there, can be generalized to systems with time-dependent rate constants. The time step $\Delta t$ has to be small enough, so that at every time instant $t$ during the simulation

$$\int_{t}^{t+\Delta t} k(E(\sigma), u)du$$

is a good approximation of

$$1 - \exp(- \int_{t}^{t+\Delta t} k(E(\sigma), u)du).$$

If this constraint is not satisfied, anomalous results may be obtained. These anomalies may occur in the order in which reactions take place and in the times at which reactions occur. The latter anomaly occurs in some of the simulations reported in (Sales and Zgrablich, 1987a), as we explain below.

Consider an adsorbed particle somewhere on the surface and denote the rate constant for desorption at time $t$ by $k_d(t)$. When this particle is visited in the simulation algorithm as proposed by Sales and Zgrablich, it desorbs with probability $k_d(t)\Delta t$, if the time increment is $\Delta t$. Since the rate constant for desorption increases in $(t, t + \Delta t]$, the probability used by Sales and Zgrablich is too small. The desorption probability in the interval $(t, t + \Delta t]$ is actually $\int_{t}^{t+\Delta t} k_d(u)du$. In the first figure of the paper this leads to incorrect desorption spectra: the temperature at which desorption starts is shifted to the right by about 5 K because of the desorption probabilities being too small. Similar remarks can be made about the other figures in the article.

Lombardo and Bell (1988) describe an algorithm similar to that introduced by Sales and Zgrablich. The difference is that not all particles are visited in every step, but $m$ random sites are visited in every step, where $m$ denotes the number of particles on the surface. If a site is selected where a particle is present, it desorbs with probability $k_d(t)\Delta t$. This means that the probability for a particle to desorb in the interval $(t, t + \Delta t]$ is $p_{sel}(m)k_d(t)\Delta t$, writing $p_{sel}(m)$ for the probability of the particle’s site being selected with $m$ particles present on the surface. If the lattice has $N$ sites, then

$$p_{sel}(m) = 1 - (1 - \frac{1}{N})^m.$$ 

If $\Delta t$ is small enough for $k_d(t)\Delta t$ to be an accurate approximation of the desorption probability in $(t, t + \Delta t]$, then the probability used by Lombardo and Bell is still off by a factor $p_{sel}(m)$. This is a serious error because this factor is highly dependent on the surface coverage.

Lutsevich et al. (1991) also use the fixed step approach to simulating temperature programmed desorption. They too assume that the increase of rate constants over the course of
one time step may be ignored, and use $k_d(t) \Delta t$ for the probability of desorption of a particle in the interval $(t, t + \Delta t]$. They choose $\Delta t$ so large that for some particles the product $k_d(t) \Delta t$ exceeds one, for the benefit of faster simulations. Besides this being a rather crude approach, the algorithm contains an error similar to that of Lombardo and Bell: not every particle is visited in every simulation step. Instead of selecting all particles in random order, $N$ uniform site selections are done. This means that the probability of a particle being selected in a simulation step is equal to

$$1 - (1 - \frac{1}{N})^N \approx 1 - \exp(-1),$$

so even for small $\Delta t$ the desorption probability used in the algorithm is off by a constant factor.

Meng and Weinberg (1994) use an algorithm that can be viewed as a simplification of our tdURSel algorithm. The simplification lies in the fact that time increments are not generated by sampling the tdExp($E(\sigma)$) distribution, but a fixed increment is used. Meng and Weinberg use an increment of $\frac{1}{k(E(\sigma), t)}$ if the state is $\sigma$ and $t$ is the current time. The algorithm is more accurate than the ones discussed earlier in this section, but the authors do not check the influence of the simplification in the time increments on the simulation results.

Jansen (1995) gave the first exact algorithms for simulating TPD and TPR systems. Jansen compares two algorithms, the ones called tdWRsel and tdRsched-a in this monograph. His conclusion is that tdRsched-a is preferable because its time per reaction grows as $O(\log_2 |E(\sigma)|)$, while the growth behavior of tdWRsel is $O(|E(\sigma)|)$. Our own investigation of these algorithms yielded the same results.

Finally we mention again the work of Brown et al. (in press). In their approach to the simulation of voltammetric scans, a scan is simulated by a sequence of simulations, all with time-independent rates. The output configuration of one simulation forms the input of the next. The electrode potential is incremented after every simulation. The WtSelULSel algorithm is used for the individual simulations. This approach resembles the fixed step algorithms discussed above in the sense that the rates are assumed to be constant over intervals of predetermined length. Brown et al. do not explain how to determine this interval length so that reliable results are obtained.

The algorithms presented by Meng and Weinberg (1994) and Jansen (1995) are still the state of the art, the former being used more often (Meng and Weinberg, 1997; Weinketz and Cabrera, 1997). The algorithms introduced earlier in this chapter offer ample improvement over the algorithms found in the literature. In particular the tdTSchedULSel-a algorithm is worth mentioning. It is the first exact algorithm for the simulation of systems with time-dependent rate constants for which the time per reaction is independent of the system size.
4.7 Literature review and conclusions

A surprising result is that the algorithms based on scheduling (groups of) reactions show better performance than those in which the time of the next reaction is computed independently of the reaction selected. If rate constants are not time-dependent, the opposite is true. This is caused by the fact that for systems with time-dependent rate constants, generating a time increment for the next reaction is a complex, and expensive operation. If rate constantss are time-independent, generating time increments is much easier.
5

Design and Performance Evaluation of CARLOS

The previous two chapters contain a number of algorithms for simulating surface processes. Some of these have been implemented in a simulation tool called CARLOS. In this chapter we discuss the design and the features of this tool. The performance of the implemented algorithms is evaluated for the three example models of Section 2.4.

There are several advantages to having a general-purpose tool for the simulation of surface processes over designing special-purpose simulation programs for different models. First and foremost, it allows the user to focus on the model instead of on the simulation technique. Time spent on designing a simulation program cannot be spent on analyzing the model being investigated. Second, it allows optimizations in simulation technique to be performed at the level of the simulator. Any optimization can immediately be used for all models. Third, although a general-purpose simulator is larger than a special-purpose program for one model, the amount of code to maintain does not grow with the number of models studied. Therefore, the software maintenance task is less of a burden.

5.1 Requirements and design decisions

The goals of the simulation tool are the following.

1. The user of the tool must be able to specify the model that he wants to simulate in a straightforward way.
2. The user must be able to monitor the evolution of the simulated system (surface coverages, system state, reaction occurrences, et cetera).

3. Simulation runs must be reproducible. This is a desirable feature for two reasons. It helps in testing the tool itself and it helps the user in testing and analyzing the simulated model.

4. Although the tool takes models as a parameter, its performance must compete with the performance of programs written specifically for the simulation of one model.

5. The tool must be portable. As we expect the users to be working in a scientific environment where Unix workstations are available, we focus on portability on Unix machines.

The requirements stated above lead to a number of design decisions.

1. Models are presented to the program by means of text files. This reduces the programming effort in comparison to a graphical dialog interface. The program produces text files containing simulation results and has the possibility of producing graphical output for viewing system states.

2. The requirement that simulation runs be reproducible in combination with the portability requirement, forces us to encapsulate a pseudo-random number generator in the program. It must be possible for the user to seed the generator in order to reproduce simulation runs.

3. In Chapter 3 we already noted that there is no single best simulation algorithm. Therefore, we decide to implement multiple algorithms in order to attain good performance.

5.2 Features of the current implementation

Most of the work on the simulation tool has been done in 1996 and 1997. After that the functionality has not been extended significantly.

When the simulation kernel was written, some of the algorithms described in Chapters 3 and 4 had not been discovered. Most notably, the only algorithms available for simulating systems with time-dependent rate constants were tdRsched(-a) and tdWTselULsel(-a). WSTselULsel-a and the other algorithms based on supertypes had not been discovered either. The current version of CARLOS implements three algorithms, viz. (td)Rsched, WTselULtrial, and WTselULsel-a. The tdRsched algorithm was included because of its performance advantage over tdWTselULsel and because it is easier to implement. The
5.2 Features of the current implementation

program contains numerical routines to efficiently generate waiting times for TPD simulations and voltammetric simulations with linear dependencies of temperature and electrode potential on time, respectively.

The two other algorithms, WTSelULsel-a and WTSelULtrial, complement each other well for the simulation of systems with constant rates. In fact, CARLOS implements a restricted version of the Gsched-ha algorithm (see page 86). The user can assign reaction types to one of two groups. The WTSelULsel-a algorithm is used for one group and the WTSelULtrial algorithm for the other.

Reaction types can be combined as explained on page 64. Furthermore, source configurations of transformation types are allowed to contain sets of particles instead of particles, as in the example of Figure 2.3. The user may specify the rotational symmetries of the lattice and of the reaction types. These two features make specification of models less laborious and less error-prone.

The program has facilities for displaying lattice states on a graphical terminal and writing them to GIF files. The latter feature facilitates making movies of simulations. It is also possible to step through simulations reaction by reaction, making detailed observations feasible. Simulation logs contain information about elapsed time, fractional surface coverages, numbers of reactions performed, and enabling efficiency ($\frac{1}{\tau}$). The logs may be written to files or piped to user defined shell commands. The latter feature can be used to discard logging information before it is written to disk, thus reducing resource use.

Performance measurements show that much time is spent on pattern matching. Furthermore, in the tdRsched implementation, much time is spent on priority queue operations. Some experimentation with several priority queue implementations showed that the so-called pagoda (Gonnet and Baeza-Yates, 1991) is the most efficient. In contrast to other applications (Jones, 1986), in CARLOS, the pagoda consistently outperforms implementations such as the implicit heap, leaf tree, splay tree, skew heap, and two-list priority queue.

In Chapter 3 we described a straightforward way for computing the set of reactions enabled by the occurrence of a reaction. When a reaction is performed, all reaction types are matched at all locations for which they have a site in common with the modified sites of the reaction performed. For most models, many reactions can be ruled out in advance. Take the ZGB model, for instance. Adsorption of $A$ does not enable another adsorption reaction, of $A$ or $B_2$, because it does not create any empty sites. In the naive approach, the pattern matches for $A$ and $B_2$ adsorption are performed nevertheless, wasting CPU time.

Computing the newly enabled reactions can be done more efficiently if some preprocessing is performed to prevent matching patterns of reactions that are guaranteed not to be enabled. In the preprocessing step, it is checked for every modified site of every reaction which reactions are potentially enabled. These are stored in a list. An example will clarify this.
Consider the ZGB model, and the effect of a $B_2$ adsorption in particular. In view of the reaction types as depicted in Figure 2.2, at most six reactions are enabled by a $B_2$ adsorption. Figure 5.1 shows which reactions are potentially enabled. Further reduction of this set requires information about the states of sites other than those modified by the $B_2$ adsorption, and can therefore not be achieved in a preprocessing step.

This optimization has been implemented in CARLOS. The preprocessing step adds a list of reaction types and offsets to each modified site of each reaction type. The pairs of reaction type and offset describe the reactions that are potentially enabled. In the implementation care has been taken to avoid reactions that overlap multiple modified sites being included in more than one list.

Further optimization of the computation of newly enabled reactions is possible. Consider a collection of reactions, $S_{pen}$, potentially enabled by a site modification. The patterns of these reactions may have sites in common. If the patterns of the reactions in $S_{pen}$ are matched one after another, lattice sites may be inspected more than once in the computation of the newly enabled reactions. Extra preprocessing can decrease the number of lattice site inspections.

In the preprocessing step the union of the patterns of the potentially enabled reactions is computed. We call this large pattern the enabling environment. With each site $l$ of the enabling environment an array $a$ is stored. The index type of this array is $D$, the set of particle types. Array element $a[p]$ gives the reactions from $S_{pen}$ that are not enabled if a particle $p$ is found at $l$. The enabling check now amounts to inspecting the sites in the enabling environment, and, for each site, marking the elements of $S_{pen}$ that are not enabled.
5.2 Features of the current implementation

When all sites have been inspected, the unmarked elements of $S_{pen}$ constitute the set of newly enabled reactions. This optimization is also available in CARLOS.

In the WTselULsel-a algorithm ineligible reactions are deleted from $E$ in unsuccessful iterations of the algorithm. Deletion of an ineligible reaction involves generation of a time increment, a weighted type selection, a uniform location, and, finally, an eligibility check. The average amount of work per reaction can be reduced by performing garbage collection on $E$. In a garbage collection operation, $E$ is scanned and ineligible reactions are removed from it. The total amount of work involved in a garbage collection step is roughly $|E| \tau_{pm}$.

If a large fraction $f$ of the reactions in $E$ is ineligible, then the cost per ineligible reaction for garbage collection, roughly $\frac{\tau_{pm}}{f}$, may be lower than the cost per ineligible reaction for the standard way of dealing with ineligible reactions. After a garbage collection operation has been performed, $E$ contains only eligible reactions. Hence, the average number of trials per reaction is temporarily lower than the equilibrium value $n$. Frequent garbage collection may thus increase the efficiency of simulations.

Garbage collection is possible in CARLOS. The program allows the amount of memory available for $E$ to be determined by the user. Garbage collection is performed when the size of $E$ reaches the user-specified limit.

Before we investigate the performance of the implemented algorithms and of the improvements in performance offered by the optimizations suggested above, we briefly discuss correctness of the program. The program has been designed in a fairly modular way. This allows us, for instance, to use different priority queue implementations without much effort. If different priority queue implementations lead to exactly the same sequences of reactions being performed when Rsched-a is used, it is safe to say that they are free of errors. The optimizations in computing newly enabled reactions have been tested in the same way: by comparing simulations in which no optimization is used with simulations in which the optimizations are used.

Other functionality is less easily tested. How do we know whether the simulation kernel is correct? We performed a number of tests, gathering empirical data that suggest that the simulation kernel does not contain programming errors. The first test was to simulate models that can be solved analytically, and to compare the simulation results to the analytical results. A second type of test is found in the simulation of models in which no adsorption and desorption occur. A model which includes only diffusion steps, for example, has the property that the surface coverage remains the same throughout a simulation. Simulation of such a model revealed an error in the initialization routines in CARLOS. As a third type of test, we simulated some models described in the literature. The results produced by CARLOS are consistent with those found in the literature.
5.3 Performance evaluation

In this section we present some performance figures of CARLOS. We compare the performance of the three algorithms implemented, as well as the effect of the optimizations in computing newly enabled reactions. The models introduced in Section 2.4 are used here to evaluate CARLOS’ performance. The three examples highlight different aspects of the influence that models and system sizes have on performance.

Some of the simulations to be presented involve large systems, requiring considerable computing resources. All simulations were performed on a Sun Enterprise 3000 server with three 248 MHz UltraSPARC-2 processors and 2 Gbytes of main memory. The simulation program is a sequential program, so it does not take advantage of the availability of multiple CPUs. However, the load of the machine was small enough to have one processor available solely for the simulations.

5.3.1 The ZGB model

The ZGB model was originally described by Ziff et al. (1986) in an algorithmic fashion. The following steps were given.

1. Select a particle type; $A$ is selected with probability $y_A$, and $B$ with probability $y_B = 1 - y_A$.

2. If $A$ was selected, then select a site uniformly. If the selected site is empty, then
   
   (a) $A$ adsorbs, and
   
   (b) its four neighbor sites are checked in random order for presence of $B$; if $B$ is found, the $A$–$B$ pair desorbs associatively.

3. If $B_2$ was selected, then select a pair of adjacent sites uniformly. If both selected sites are empty, then
   
   (a) $B_2$ adsorbs dissociatively, and
   
   (b) the six neighbors of the two selected sites are checked in random order for presence of $A$; if $A$–$B$ pairs are found, they desorb associatively.

This algorithmic description can be translated into the reaction types given in Figure 2.2 on page 18. The rate constant $k_A$ for $A$ adsorption is equal to $y_A$. The rate constant $k_{B_2}$ for $B_2$ adsorption is equal to $\frac{1-y_A}{2}$ because of the two possible orientations of the pair of sites.

Figure 5.2 shows equilibrium surface coverages and $AB$ production as a function of $y_A$. The $AB$ production is given as the average number of $AB$s produced per attempted adsorption.
5.3 Performance evaluation

![Equilibrium coverages and AB production for the ZGB model as a function of $y_A$.](image)

Depending on $y_A$, the system goes to one of three equilibrium phases. If $y_A \leq y_1 = 0.389 \pm 0.005$, then at equilibrium the surface is poisoned with $B$ (Ziff et al., 1986); no empty sites are available for $A$ adsorption and no reactions are possible. If, on the other hand, $y_A \geq y_2 = 0.52560 \pm 0.00001$, then $A$ poisoning occurs (Ziff and Brosilow, 1992). The interval $(0.389, 0.5256)$ constitutes the reactive phase.

Figure 5.3 shows typical equilibrium states for three values of $y_A$, viz. 0.400, 0.500, and 0.525. The pictures were obtained by cutting a square of 64 x 64 sites from a 256 x 256 lattice. Figures 5.2 and 5.3 show that in the reactive phase the average number of empty sites increases with $y_A$. Since reactions are enabled only at empty sites, the rate of change $k(E(\sigma))$ increases with $y_A$ as well.

We performed simulations for the three values of $y_A$ mentioned above, using the three implemented algorithms in turn for each value. The performance results are plotted in Figure 5.4.

As expected, the time per reaction increases significantly with the lattice size for the Rsched algorithm. It is almost independent of the lattice size for the other two algorithms. For this reason, we restrict the discussion of the performance to WTselULsel-a and WTselULtrial.

Figure 5.4(b) shows that the performance of the WTselULtrial algorithm depends strongly on $y_A$. The origin of this dependence is found in the factors $\frac{L(u) L(k(RT))}{k(E(\sigma))}$ that occur in WTselULtrial's time per reaction (see Equation (3.21) on page 63). In the ZGB model the total rate of change decreases with $y_A$, as argued above, reducing the efficiency of the
Figure 5.3: Snapshots of equilibrium states of ZGB model: (a) $y_A = 0.400$, (b) $y_A = 0.500$, (c) $y_A = 0.525$. $A$ is white, $B$ is grey, and empty sites are black.

Figure 5.4: Performance of three simulation algorithms on the ZGB model: (a) Rsched, and (b) WTselULsel-a and WTselULtrial, marked 's' and 't', respectively.

WTselULtrial.

The performance of WTselULsel-a also depends on $y_A$, but to a lesser extent. The dependence is anti-monotonic for WTselULsel-a: higher values of $y_A$ lead to lower performance of the algorithm. The reason is that $n$, the number of reactions enabled per reaction performed, grows for increasing $y_A$.

The example of the ZGB model shows that WTselULsel-a is generally the best of the three implemented algorithms. Its performance is less sensitive to system sizes and model parameters than that of the other two algorithms. In some cases WTselULtrial shows better performance than WTselULsel-a; we expect Rsched to perform worse than WTselULsel-a on any model. A possible advantage of the WTselULtrial algorithm, not shown in the graphs, is its low memory use. For example, when simulating a $256 \times 256$ lattice, for $y_A = 0.500$, WTselULsel-a requires some $3.0 \cdot 10^6$ bytes of memory, while WTselULtrial
only uses $1.9 \cdot 10^5$ bytes.

### 5.3.2 Diffusion with lateral interaction

The ZGB model is a very simple model: it consists of only seven reaction types and the reaction patterns contain only one or two sites. In order to study the effects of the different ways to compute newly enabled reactions on the performance, we consider a model with more and larger reaction types, the diffusion model of Section 2.4.2. This model comprises 32 reaction types, all with patterns that are five sites large. A repulsive pair interaction of 4 kJ/mol is used in the simulations we perform.

<table>
<thead>
<tr>
<th>$\theta$</th>
<th>WTselULsel-a, standard</th>
<th>WTselULsel-a, restricted set</th>
<th>WTselULsel-a, optimized enabling</th>
<th>WTselULtrial</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.25</td>
<td>$1.6 \cdot 10^{-4}$</td>
<td>$7.1 \cdot 10^{-5}$</td>
<td>$5.4 \cdot 10^{-5}$</td>
<td>$5.6 \cdot 10^{-4}$</td>
</tr>
<tr>
<td>0.50</td>
<td>$1.7 \cdot 10^{-4}$</td>
<td>$9.5 \cdot 10^{-5}$</td>
<td>$7.2 \cdot 10^{-5}$</td>
<td>$1.1 \cdot 10^{-4}$</td>
</tr>
<tr>
<td>0.75</td>
<td>$1.5 \cdot 10^{-4}$</td>
<td>$7.8 \cdot 10^{-5}$</td>
<td>$5.8 \cdot 10^{-5}$</td>
<td>$2.8 \cdot 10^{-5}$</td>
</tr>
</tbody>
</table>

Table 5.1: Simulation performance for diffusion model. The time per reaction (in s) is reported for every surface coverage and every algorithm.

Table 5.1 reports results for simulations of a $128 \times 128$ lattice at 300 K for both the WTselULsel-a and WTselULtrial algorithms. The first column gives the time per reaction for the naive method of computing newly enabled reactions: all reaction types are matched at all locations such that they contain a modified site. The second column gives the results for the case that only the potentially enabled reactions are tried. The third column gives the results for the optimized procedure for computing which potentially enabled reactions are actually enabled. The time per reaction is seen to be reduced by more than 50% by the optimizations for WTselULsel-a. Finally, the fourth column gives the time per reaction for the WTselULtrial algorithm.

This example again shows that sometimes WTselULtrial exhibits remarkable performance. In the current example, the algorithm’s performance depends critically on the surface coverage. The dependence is even much larger than for the ZGB model. This can be understood by considering the rate constants of the reaction types. Table 5.2 gives the normalized rates occurring in the system for $T = 300$ K and a repulsive, pairwise interaction of 4 kJ/mol.

The table shows that the rate constant for hopping of a particle with three nearest neighbors is the highest. This means that in the WTselULtrial algorithms reactions of this type are selected in the majority of iterations. At high surface coverage there will be more particles
Number of neighbors of moving particle & Normalized rate constant \\
0 & 1 \\
1 & 4.97 \\
2 & 24.7 \\
3 & 122 \\

Table 5.2: Normalized rate constants for the diffusion model at $T = 300$ K with repulsive, pairwise interaction of 4 kJ/mol.

surrounded by three others than at low coverages. As the interaction is repulsive, hardly any particles with three nearest neighbors will be presented on the surface if there are enough empty sites.

Comparing the performance of WTSelULSel-a on the ZGB model and the current model, we see that the time per reaction is much smaller for the former. This is caused by two factors: the computation of newly enabled reactions takes longer for the diffusion model and on average more reactions are disabled and enabled when a reaction is performed. This latter factor has the result that the number of unsuccessful iterations per reaction is rather high for the diffusion model: the simulation logs indicate that only one out of every 12 iterations is successful for the case that $\theta = 0.50$. It also means that $E$ contains many ineligible reactions.

In the evaluation of the performance of this model we look at the effects of garbage collection. Recall that the user of CARLOS may specify a limit on the amount of memory available for $E$. If $E$ grows beyond this limit, garbage collection is invoked. Figure 5.5(a) shows the number of reactions in $E$ as a function of the simulated time for four limits on the size of $E$. The graph shows that the amount of memory can be reduced from its natural size of 16 million bytes to a million bytes. Figure 5.5(b) shows that a limit of one million bytes results in much improved performance too: $5.0 \cdot 10^{-5}$ s per reaction instead of $7.5 \cdot 10^{-5}$ s, a reduction of over 30%. The reduction in the time per reaction is of the same order for the case the surface coverage is 0.25 or 0.75 (27% for both).

The current strategy of performing garbage collection when $E$ reaches a certain size is not optimal. In more complicated models it is likely that the ineligible reactions are not evenly divided over the reaction types. Systems in which reactions with low rate constants are enabled by reactions with high rate constants are likely to exhibit such an uneven distribution. The reactions with high rate constants enable other reactions. These have high probability of being disabled because their waiting times are relatively long. Many ineligible reactions of the types with low rate constants will occur in $E$. In that case it would be better to
Figure 5.5: Graphs showing (a) number of reactions in $E$ vs. simulated time for $\theta = 0.50$, and (b) simulation performance vs. limit on size of $E$.

perform garbage collection only on types with large fractions of ineligible reactions.

The use of the Gsched-hao algorithm, combining WTselULsel-a and WTselULtrial in one simulation, offers another way to deal with relatively slow reactions enabled by fast ones. If the WTselULtrial is used for slow reactions, these are not stored at all. As a consequence there are fewer ineligible reactions in $E$. Moreover, the cost of computing newly enabled reactions is reduced in comparison with a pure WTselULsel-a simulation. Pattern matching is performed only for potentially enabled reactions of types for which the WTselULsel-a algorithm is used, not for those of types for which WTselULtrial is used.

We apply the Gsched-hao algorithm to our diffusion model. The question is which algorithm to use for which reaction types. Above we argued that WTselULsel-a should be used for fast reactions and WTselULtrial for slow ones. It seems reasonable to use WTselULsel-a for hopping of particles with three nearest neighbors and WTselULtrial for particles without nearest neighbors. We experiment with two possibilities: (a) use WTselULsel-a only for particles with three nearest neighbors, and (b) use WTselULsel-a for particles with two or three nearest neighbors. Table 5.3 shows performance results for these two possibilities, and for ease of reference, those for simulations using the WTselULsel-a algorithm for all reaction types. For each surface coverage and each choice of algorithm for the various reaction types, the right hand side table gives the time per reaction and optimum amount of memory for the set $E$ of enabled reactions.

The table clearly shows that the time per reaction can be improved substantially by careful selection of the algorithm to be used for each reaction type. As a rule of thumb, the WT-
<table>
<thead>
<tr>
<th>Number of neighbors of moving particle</th>
<th>(a)</th>
<th>(b)</th>
<th>(c)</th>
<th>( \theta )</th>
<th>(a)</th>
<th>(b)</th>
<th>(c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>trial</td>
<td>trial</td>
<td>sel</td>
<td>0.25</td>
<td>1.4 ( \cdot ) 10^{-4}</td>
<td>3.6 ( \cdot ) 10^{-5}</td>
<td>3.9 ( \cdot ) 10^{-5}</td>
</tr>
<tr>
<td>1</td>
<td>trial</td>
<td>trial</td>
<td>sel</td>
<td>0.50</td>
<td>5.0 ( \cdot ) 10^{2}</td>
<td>7.5 ( \cdot ) 10^{3}</td>
<td>5.0 ( \cdot ) 10^{5}</td>
</tr>
<tr>
<td>2</td>
<td>trial</td>
<td>sel</td>
<td>sel</td>
<td>0.50</td>
<td>3.5 ( \cdot ) 10^{-5}</td>
<td>2.6 ( \cdot ) 10^{-5}</td>
<td>5.0 ( \cdot ) 10^{-5}</td>
</tr>
<tr>
<td>3</td>
<td>sel</td>
<td>sel</td>
<td>sel</td>
<td>0.75</td>
<td>1.2 ( \cdot ) 10^{4}</td>
<td>1.0 ( \cdot ) 10^{5}</td>
<td>1.1 ( \cdot ) 10^{4}</td>
</tr>
<tr>
<td></td>
<td>2.0 ( \cdot ) 10^{-5}</td>
<td>2.7 ( \cdot ) 10^{-5}</td>
<td>4.3 ( \cdot ) 10^{-5}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3.0 ( \cdot ) 10^{5}</td>
<td>4.5 ( \cdot ) 10^{5}</td>
<td>7.0 ( \cdot ) 10^{5}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.3: Performance comparison of simulations of the diffusion model. The left hand table lists the methods used for the reaction types; the right hand table gives the time per reaction and memory use for each simulation.

selULtrial algorithm is best used for the slower reaction types, while WTselULsel-a is the more suitable choice for the faster reaction types. The rate constants are not the only quantities to be considered, as columns (a) and (b) in the right hand side table show. A reaction type of which, on average, few instances are enabled, is sometimes better simulated with WTselULsel-a, despite its rate constant being low.

### 5.3.3 A more advanced model for CO oxidation

The final model that we discuss is the CO oxidation model analyzed by Gelten et al. (1998). The reaction types of this model are described in Section 2.4.3 (in Figures 2.5 and 2.6, to be specific). We focus on performance aspects of the simulations. The model is interesting in this respect because it is more complex than the models discussed in the preceding sections. One of the interesting phenomena observed in simulations of the model is oscillatory behavior. This means that the number of enabled reactions does not become constant. The question arises whether our heuristic for choosing algorithms for specific reaction types is still valid.

Here we report the results of simulations of 256 \( \times \) 256 lattices at a temperature of 480 K. We performed simulations of the model both with CO diffusion and without it. First we discuss the case that no diffusion is included.

The use of WTselULtrial leads to abominable performance: the time per reaction is some \( 10^{-2} \) s. WTselULsel-a does much better, with a time per reaction of \( 1.2 \cdot 10^{-4} \) s. Using the optimized way of computing newly enabled reactions reduces the time per reaction to
2.5 \cdot 10^{-5} \text{ s}. The time per reaction can be further reduced by restricting the amount of memory available for the set of enabled reactions $E$ from its natural size of 16 million bytes to 2 million bytes.

At 480 K, the rate constant of the CO$_2$ production reactions exceeds all other rate constants. The CO$_2$ production reaction types have rate constant 8.96 s$^{-1}$, and the rate constant of the next fastest reaction type, oxygen adsorption, is 1.0 s$^{-1}$. CO$_2$ production reactions are enabled at relatively few locations. Because of this, the use of WTselULtrial leads to such low performance. The occurrence of a CO$_2$ reaction enables, among others, CO adsorption at the empty sites created. Hence, if WTselULsel-a is used, CO adsorption reactions are matched in the computation of newly enabled reactions after every occurrence of a CO$_2$ production reaction. As the rate constant of CO adsorption is a factor 20 lower than that of CO$_2$ production, the average waiting time for a CO adsorption is relatively high, and many CO adsorption reactions become disabled before they are selected to be performed. This problem of frequent matching together with a large number of disablings can be alleviated by using the WTselULtrial algorithm for CO adsorption. When this algorithm is used for CO adsorption, reactions are matched much less frequently.

The rate constants of the nucleation and trapping reactions are a factor 25 lower still than that of CO adsorption. If WTselULsel-a is used, these reactions are matched as often as CO adsorption occurs. So, using WTselULtrial for the reaction types that model the surface transformation will probably improve the performance of the simulation.

This leaves us to decide which algorithm to use for oxygen adsorption. Since its rate constant is rather high compared to the rates of CO adsorption and the nucleation and trapping reactions, we use WTselULsel-a for it. Because we use WTselULtrial for the majority of the reaction types, the memory requirements of the simulation go down significantly. The optimal bound on the size of $E$ is a mere 75,000 bytes, resulting in a time per reaction of 1.1 \cdot 10^{-5} \text{ s}. As it turns out, this particular choice of algorithms for the various reaction types is optimal.

In order to model the CO oxidation on platinum more adequately, diffusion of adsorbed CO must be introduced in the model. In the system considered, CO diffusion is very fast. Simulating diffusion at its true rate is not feasible: because diffusion is so much faster than the other reactions, only very few non-diffusion reactions would be performed. Gelten (1999) has performed simulations with diffusion rate constants of 50 s$^{-1}$. In these simulations, 98.4% of the reactions are diffusion reactions. Thus, if the time per reaction is the same as in the case there is no diffusion, simulating the same amount of time takes more than 60 times as long.

Unfortunately, the introduction of diffusion in the model increases the time per reaction. This has to do with the frequent enabling and disabling of diffusion reactions as other diffusion reactions are performed. To make matters worse, one of the effects of diffusion is that the size of the spatio-temporal patterns on the lattice increases (Gelten, 1999). Hence, large
systems have to be simulated for the patterns to fit on the lattice. Simulations performed by Gelten routinely contain more than 30 billion reactions for 2048 x 2048 lattices.

In Table 5.4 we have included the performance effects of slow diffusion (rate constant 5.0 s\(^{-1}\)) and fast diffusion (rate constant 10 s\(^{-1}\)) on simulations.

<table>
<thead>
<tr>
<th></th>
<th>No diffusion</th>
<th>Slow diffusion</th>
<th>Fast diffusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>WTselULtrial</td>
<td>1.0 \cdot 10^{-2}</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>WTselULsel-a,</td>
<td>1.2 \cdot 10^{-4}</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>standard</td>
<td>1.6 \cdot 10^{7}</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>WTselULsel-a,</td>
<td>2.1 \cdot 10^{-5}</td>
<td>4.3 \cdot 10^{-5}</td>
<td>4.4 \cdot 10^{-5}</td>
</tr>
<tr>
<td>optimized</td>
<td>2.0 \cdot 10^{6}</td>
<td>4 \cdot 10^{6}</td>
<td>4 \cdot 10^{6}</td>
</tr>
<tr>
<td>WTselULsel-a +</td>
<td>1.1 \cdot 10^{-5}</td>
<td>2.6 \cdot 10^{-5}</td>
<td>2.6 \cdot 10^{-5}</td>
</tr>
<tr>
<td>WTselULtrial</td>
<td>7.5 \cdot 10^{4}</td>
<td>1.1 \cdot 10^{6}</td>
<td>1.1 \cdot 10^{6}</td>
</tr>
</tbody>
</table>

Table 5.4: Overview of the performance results of simulations of Gelten’s CO oxidation model. For every simulation, the time per reaction (s) and amount of memory for E (bytes) are reported.

### 5.4 Conclusions

The performance results reported in this chapter show many things. We give a brief review and some conclusions.

Simulation of the ZGB model shows that the three implemented algorithms behave as predicted by our performance model. For Rsched-a the time per reaction increases with the system size; for WTselULtrial and WTselULsel-a it is almost independent of the system size. Also, the performance of WTselULtrial depends strongly on the activity of the system. It performs badly for systems in which relatively few reactions are enabled. The same behavior is observed in simulations of the diffusion model described in Section 5.3.2. This example shows even more clearly that a reaction type with a high rate constant and few enabled instances dramatically decreases the performance of WTselULtrial.

Another conclusion that we may draw from the results described, is that the various optimizations pay off. If the computation to find newly enabled reactions is optimized, performance is greatly increased. Garbage collection can also result in a significant performance
increase, and at the same time reduces the amount of memory needed by the simulation program. Garbage collection is invoked when $E$ grows beyond a user-specified limit. This is a rather primitive way to optimize the amount of memory used. It requires the user to experiment and determine the optimum. Unfortunately, we have not been able to give equations that allow the optimum to be calculated. One of the reasons for this is that we have only considered limits on the size of $E$ as a whole, although it is possible and even likely that different reaction types have different influences. Therefore we recommend that an approach be investigated in which different reaction types are considered separately. Ideally, the program should compute an optimum size for each reaction type.

Results also show that the combination of algorithms in one simulation is beneficial to the performance. It is important to choose the right algorithm for every reaction type. Currently, only two algorithms can be used in a simulation, viz. WTselULsel-a and WTselULtrial. The Gsched-hao algorithm, on which this combination of algorithms is based, allows more algorithms to be combined. If more algorithms, such as WSTselULsel-a, are implemented, the choice of algorithm for the reaction types of a model becomes more difficult. Ideally, the simulation program should make the choice, instead of putting the burden on the user. For this to be possible, a better understanding is needed of the influence that reaction types have on simulation performance. For instance, we have no satisfactory explanation for the increase in the time per trial in simulations of Gelten’s CO oxidation model when diffusion is added.

Finally, we note that no attention has been paid to the performance of simulations of systems with time-dependent rate constants. The reason is that only one algorithm is available in CARLOS for such simulations, tdRsched-a. The dependence of time per trial on the system size is the same for tdRsched-a as for Rsched-a, so it is of little use to show simulation results for this aspect of the performance. The same can be said about the influence of the optimizations discussed above, with the exception of combining algorithms. Since only one algorithm is available, no combinations can be investigated.

The results for systems with time-independent rate constants indicate that it is desirable to have at least two algorithms available, one in which the locations of enabled reactions are stored and one in which reactions are tried randomly. This suggests that both tdTschedULsel-a and tdTschedULtrial be implemented.
6

Opportunities for Parallelism

So far we have concentrated on developing fast sequential algorithms for the simulation of surface processes. The question we address in this chapter is whether parallel algorithms may be used to obtain faster simulations. We start with a brief introduction to parallel computing. After that, we investigate the possibilities for the application of parallelism to the simulation of surface processes. We then discuss a case study performed with a parallel simulation program for the ZGB model (see Section 2.4.1).

6.1 Introduction

A traditional computer contains one processing unit or CPU, which is connected to one memory element (MEM). The memory stores a program and data. The CPU repeatedly fetches program instructions from the memory and executes them. From a system perspective, this structure can be generalized in several ways. The first generalization consists of connecting multiple CPUs to one memory element, resulting in a shared memory parallel computer. Multiple programs may be stored in the memory, and the CPUs execute instructions from each of the programs. It is possible to have data generated by one program be used by another. In this way a large computation may be split up over several programs, executed by a collection of CPUs. A next generalization is to connect multiple CPUs and multiple memory elements. The advantage of this architecture is that it allows more concurrency in communication. These two architectures are examples of tightly coupled parallel computers. In so-called loosely coupled parallel machines, a number of traditional computers — i.e., computers with one CPU and one memory element each — are connected
by means of some connection network. The connection network can be, for instance, a simple ethernet, a high-speed crossbar switch, or a two-dimensional grid. This approach to building parallel computers has the advantage that it scales better than a tightly coupled approach. This means that very large machines are feasible, both in total memory size and in number of processors. Figure 6.1 shows diagrams of these different architectures.

![Diagrams of parallel machine architectures](image)

(a) (b) (c) (d)

**Figure 6.1:** Some possible architectures of parallel machines: (a) and (b) are tightly coupled machines; (c) and (d) are loosely coupled machines.

Portability is an important requirement in the development of software. When developing parallel programs, this requirement is particularly important because of the variety of hardware architectures for parallel computers. A general solution is to use a software layer between the application program and the interface to the hardware, offering a standard collection of primitives (see Figure 6.2). The main advantage of using an intermediate software layer is that the portability problem is reduced to reimplementing of this layer instead of all application algorithms. There are two de facto standards for parallel libraries,
6.1 Introduction

![Diagram of the use of a library to create a standard interface to parallel computers.](image)

Figure 6.2: Schematic overview of the use of a library to create a standard interface to parallel computers.

viz. the message passing interface or MPI (Message Passing Interface Forum, 1994), and the parallel virtual machine or PVM (Geist et al., 1994).

A parallel programming interface offers the user a virtual parallel machine. The question how such a virtual machine should look, has not yet been resolved entirely. The two standards mentioned both offer a distributed memory programming model. A user program consists of a number of processes, all with a private memory space. The processes communicate via channels. Approaches based on the shared memory paradigm are used too, but these are losing ground to the distributed memory model. One of the reasons is that modern parallel machines have physically distributed memories. An intermediate software layer can present this as one large memory to the programmer, but leads to a substantial increase in the cost of memory references. A distributed memory interface to a machine that offers a shared memory is often less costly, because the underlying machine has physically distributed memory. Another benefit of using a distributed machine model is portability. The effort of porting a program to another machine and obtaining good performance is smaller if a distributed memory model is used.

The performance of parallel computations is often expressed by the speedup. The speedup is the quotient of the execution time of the parallel computation and the best sequential computation for the same problem. When \( n \) processors are used in a parallel computation, the speedup is at most \( n \). This upper bound is attained only when the sequential computation can be divided evenly over the \( n \) processors, without introducing any overhead. Overhead consists, for instance, of time spent on communication between processors and time spent waiting for intermediate results from other processors.

At the algorithm design level, two types of parallelism can be distinguished, viz. control
parallelism and data parallelism (see, e.g., Quinn, 1994). An example of control parallelism is found in pipelined computations. Data flows through a number of pipeline stages, processing occurring at each of the stages. In general, the process graph may be more complex than just a line.

Control parallelism is based on identifying a number of tasks in a computational problem that may be performed in parallel. The number of such tasks may be limited, so that the maximum speedup is limited too. If $n$ tasks are identified, then at most $n$ processors can be used, yielding a speedup of at most $n$. Note that the maximum speedup does not scale with the problem size. If more data is to be processed by the same algorithm, no extra processors can be employed to speed up the computation.

Control parallelism is based on decomposing the computation to be performed into a number of subcomputations. In the data parallelism approach, it is the data set that is decomposed. In matrix multiplication, for example, every element of the product is an inner product of a matrix row and a column. The inner product computations can be performed in a parallel. The processing elements all perform the same computation, albeit on different data, whereas the processing elements in a data flow program perform different computations.

Data-parallel programs usually exhibit a higher degree of parallelism than programs based on control parallelism. Moreover, a data-parallel program may benefit from using more processors as the problem size increases. The highest speedup that can be attained with $P$ processors is $P$. In most practical cases the speedup will be lower as a result of communication overhead.

Another classification that is often used in the field of parallel computing is that introduced by Flynn (1972). Flynn distinguishes four types of computations. The first is the so-called Single Instruction Single Data (SISD) type. This is a sequential computation. The other three are SIMD, MISD, and MIMD, where 'M' stands for multiple. The SIMD type is used in vectorization: the processing elements operate in lockstep, all performing the same operation at the same time. The MIMD model is better suited for loosely coupled machines. The processing elements, consisting of a CPU and memory, each store a program and execute this, using locally stored data. Data is exchanged between processing elements as prescribed by the programs they execute.

### 6.2 Parallel simulation of surface processes

We investigate how parallelism may be used in the simulation of surface processes. The goal is to see whether simulations that are too large to run on a sequential computer may be run on a parallel computer. There are two dimensions along which a simulation problem may be too large. The lattice may be too large, so that the simulation takes too long or
6.2 Parallel simulation of surface processes

does not fit into the available memory. Second, there may be so many reaction types that the processing time per reaction is prohibitively large. A large number of reaction types usually results in a large number of reactions being disabled and enabled for every reaction performed, increasing the time per reaction and the memory use of most of the algorithms described in Chapter 3.

In summary, we are looking for parallel simulation algorithms that have the following properties.

- They are correct. In particular, the simulated kinetics are correct.
- They allow large systems to be simulated.
- They allow faster simulation of complex systems (systems for which the time per reaction is large).

6.2.1 Replication

A trivial and sometimes very useful way of introducing parallelism is by running multiple, independent simulations in parallel. This is useful if many simulations have to be performed in order to improve statistics or when scanning a range of parameter values. The advantage is that the parallelism introduces no overhead. However, this approach neither helps in improving the speed of individual simulations, nor allows larger systems to be simulated.

6.2.2 Control parallelism

Over the course of a sequential simulation, different tasks are performed repeatedly. Some of these tasks can be performed more or less independently. We distinguish, as an example, three such tasks, viz. generation of random numbers, operations on the set of enabled reactions, and operations on the lattice.

A process may be dedicated to generating random numbers from the uniform probability distribution on the unit interval. This process can generate numbers in advance, and store them in a list. Other processes that need random numbers can obtain them from this list by communicating with the random number generating process.

Similarly, a process may be dedicated to recording the set of enabled reactions. Other processes can communicate with it to request elements to be added to the set or deleted from it. Depending on the simulation algorithm, different operations are required. In the case that a scheduling type of algorithm is used, it has to be possible to delete the element
with the smallest occurrence time. If, on the other hand, reactions are not scheduled, a (weighted or uniform) random selection has to be possible.

The three processes that we identified may be mapped onto different CPUs. This has two potential benefits. First, if the CPUs have their own memory, larger systems can be simulated because the lattice and the set of enabled reactions are stored in separate memories. Second, the performance may increase because, for instance, operations on the lattice may overlap in time with the generation of random numbers.

In this approach, parallelism is introduced within iterations of sequential simulation algorithms. As a result, communication between processes occurs in every iteration of the parallel algorithm, whether a reaction is performed in this iteration or not. In modern parallel computers, inter processor communication latencies commonly range from 10 to 100 \( \mu s \). The performance results of Chapter 5 show that in sequential algorithms we commonly obtain a time per reaction of 50 \( \mu s \). Hence, we cannot afford multiple communications per reaction if we want to reduce the time per reaction, unless successive iterations of the algorithm may overlap in time. The algorithms that we developed do not allow enough overlap to make further investigation of control parallelism worthwhile.

**6.2.3 Data parallelism**

A third approach to introducing parallelism in simulations is by decomposing the data. The most obvious decomposition is a decomposition of the lattice into chunks of equal size. The idea is to perform sequential simulations on these chunks. There is the possibility that reactions occur across the boundary between adjacent chunks. We have to see whether the processes performing the simulations for the different chunks can be synchronized in such a way that the kinetics are simulated correctly. Without synchronization, the local simulated times of the different processes may diverge, and so-called causality errors may occur. A causality error occurs when an event causes something to happen in the past instead of the future. For instance, a reaction that is performed near the boundary of a chunk at a certain time \( t_0 \), may lead to a reaction becoming enabled at the other side of the boundary. Say that this reaction occurs at time \( t_1 \). If the local times at the two chunks are not equal, then \( t_1 \) may be smaller than \( t_0 \).

Before we investigate how to approach the problem of obtaining correctly simulated kinetics for this type of parallel simulation, we look at its potential performance gain. The lattice is divided into chunks. Each process performing a simulation on a chunk may record the set of reactions enabled on its chunk. Thus, the amount of memory used per process is decreased by dividing the lattice into more chunks. Simulation of systems too large for a sequential machine may become feasible on parallel machines, provided that enough processors are available.
It is questionable whether simulations of smaller systems in which the time per reaction is high can be speeded up much by applying the proposed parallelization method. The reason is the following. Whenever a reaction occurs across a boundary between two chunks or so close to it that reactions become enabled at an adjacent chunk, communication operations have to be performed between the chunks involved. These operations, which do not occur in sequential simulations, increase the average time per reaction. The frequency at which communication between adjacent chunks occurs depends on the number of sites close to a boundary. This number, in turn, depends on the size of the reaction patterns and the size of the lattice chunks. The ratio of the boundary length and the number of sites per chunk is proportional to \( \frac{1}{\sqrt{M}} \), where \( M \) stands for the number of sites in the chunk.\(^1\) Because of the \( \frac{1}{\sqrt{M}} \) ratio between the boundary length and the number sites in a chunk, we expect the time per reaction to have the following form:

\[
a + \frac{b}{\sqrt{M}}.
\]

In this formula the first term can be interpreted as the amount of computation per reaction. The second term can be interpreted as the amount of communication: communication cost of size \( b \) is incurred for a fraction \( \frac{1}{\sqrt{M}} \) of the reactions. The larger reaction patterns are, the further a reaction's origin must be situated from a boundary for it not to affect enabledness of reactions at adjacent chunks. This means that the overhead of a parallel simulation will be large for small systems with large reaction patterns. Decomposing the lattice is more attractive for simulating large systems.

The correctness problem that we face — avoiding causality errors — has been investigated extensively by researchers in the field of parallel discrete event simulation (PDES). Fujimoto (1990) gives a good overview of possible solutions. We discuss these solutions and their applicability in our context, but first we investigate how our model for surface reactions can be cast in the form of a PDES model.

PDES models consist of so-called logical processes. Logical processes mimic the behavior of physical processes by executing events. A result of an event being executed may be that new events become enabled both locally and at other logical processes. This is signaled by means of timestamped event messages sent along links between logical processes. The timestamp of an event message contains the time at which the event is due to occur. The only communication between logical processes is by means of timestamped messages. If each logical process executes events in non-decreasing time stamp order, then no causality errors occur.

In a surface reaction system, the physical processes are the reactions that may occur. We can introduce a logical process for every unit cell and have this process simulate reactions

\(^1\)This is the ratio for two-dimensional systems; for three-dimensional systems the ratio is proportional to \( \frac{1}{\sqrt{M}} \).
with their origin at that unit cell. This approach introduces a high degree of parallelism and the state information is highly distributed. It has an important disadvantage. In general, reaction patterns consist of more than one site. Hence, most reactions will influence the state of multiple logical processes. Every lattice site may be changed by a reaction with its origin at another unit cell. So essentially all state information is shared.

Most PDES techniques forbid direct access to shared state information. There are two ways to deal with shared data. First, shared data may be copied, giving all processes that need access to it their own local copy. A protocol is needed to ensure coherence among the various copies, giving rise to considerable overhead. Because of the large overhead in both space and time, and because of the complexity of this approach (Wieland and Jefferson, 1989), we focus on the second way to handle shared data. In this second approach access to shared data is allowed only by means of dummy events. These dummy events do not model any physical activity. They only serve to exchange state information between processes. This is a simpler way to deal with shared state information, but it also introduces much overhead. Because of the amount of shared data if there is a logical process for every unit cell, we can expect many dummy events.

The problem of the high overhead introduced by dummy events can be alleviated by dividing the lattice into fewer parts. If the number of unit cells per chunk is larger, the amount of shared data is reduced. Only state information near the boundary between chunks is shared. By choosing the chunk size equal to one, we can still obtain the previous decomposition.

Two approaches to parallel discrete event simulation are distinguished: the conservative and the optimistic approaches, respectively. In order to explain the difference between the two, the concepts of safe and unsafe events are used. An event with occurrence time \( t_0 \) is safe for being performed by a logical process if there are no events locally with an occurrence time smaller than \( t_0 \) and the process can determine that no events with smaller occurrence times will be generated as a result of incoming messages. Hence, a safe event can be performed without violating causality.

Conservative PDES algorithms are based on performing only safe events, thus avoiding causality errors. Optimistic algorithms, in contrast, allow unsafe events to be performed and rely on a detection and recovery mechanism to ensure correctness of simulations. Below we discuss these two types of PDES algorithms in some more detail.

**Conservative PDES algorithms**

The first parallel simulation algorithms were conservative in nature. If, in a conservative algorithm, a process contains no safe events, than that process must wait idly. It must wait until it contains safe events. This may lead to deadlock situations. We discuss one way to deal with this problem, viz. that introduced independently by Bryant (1977) and by Chandy and Misra (1979).
6.2 Parallel simulation of surface processes

In this algorithm, it must be statically known which processes may communicate with each other. A process has an incoming link for each process it receives messages from. An incoming link has a FIFO buffer associated with it to store unprocessed messages. The algorithm has the property that the timestamps of the sequence of messages transmitted over a link are non-decreasing. Hence, the timestamp of a message gives a lower bound on the timestamps of messages to be received on that link in the future. The timestamp of an incoming link is defined as follows. If all messages received on the link have been processed, then it is equal to the time stamp of the last message received on the link. If not all messages received on a link have been processed, then the link’s timestamp is equal to that of the oldest of these unprocessed messages. Initially, when no messages have been received on any link, the link timestamps should be set to zero: no message has a timestamp smaller than zero, and a message with timestamp zero may arrive.

An event is safe if none of the locally enabled events nor any of the links have smaller timestamps. We can now define exactly when a process does not contain any safe events. This is the case if there is an incoming link without any unprocessed messages and with a link timestamp smaller than that of the other incoming links and smaller than the timestamps of locally enabled events.

There is a potential for deadlock is this algorithm: there may be cycles of logical processes in which each logical process contains only unsafe events. All processes in the cycle then have to wait for a message to arrive, but none of them sends a message. The problem can be alleviated by having processes send null messages. These are messages the only purpose of which is to increment the timestamps of links. The use of null messages does not rule out deadlocks for all systems. In particular, deadlock is still possible in systems in which a cycle of processes exists on which events with a timestamp \( t \) may generate messages with timestamp \( t \). If all processes in the cycle have the same timestamp on their incoming links, they are in a deadlock situation.

This means we cannot use this algorithm for simulating surface reaction models containing immediate reactions. Consider two logical processes, say \( A \) and \( B \), in a model with immediate reactions, controlling adjacent chunks. Suppose that initially, before any reactions have been performed, no immediate reactions are enabled. Then both processes have to wait for a message from the other: if an immediate reaction were enabled in \( A \) (or \( B \)) at time zero, then this could lead to a message from \( A \) to \( B \) (from \( B \) to \( A \)) with timestamp zero. Since \( A \) and \( B \) do not know about each other’s enabled reactions, they wait for each other.

The introduction of messages indicating whether any immediate reactions are enabled removes the danger of deadlock in lattice states in which no immediate reactions are enabled on \( A \)’s or on \( B \)’s chunk. However, deadlock is still possible in states in which immediate reactions are enabled on both chunks. In this case neither can proceed because the other may send a messages containing an immediate reaction with higher priority than any locally enabled immediate reactions.
Conservative PDES algorithms have been developed that can handle systems with cycles of zero timestamp increments, but at the cost of substantial overhead during the simulation. We have to rule them out for the simulation of surface processes, because performance is particularly bad in the presence of cycles of zero timestamp increments (see, e.g., Fujimoto, 1989a; Reed et al., 1988).

**Optimistic PDES algorithms**

Optimistic algorithms for PDES perform both safe and unsafe events. They include mechanisms for detection and recovery from causality errors. The main advantage over conservative algorithms is that it is possible to perform events that might give rise to causality errors, but do not always cause causality errors. The best known mechanism for detecting and recovering from causality errors is the time warp mechanism (Jefferson, 1985; Jefferson and Sowizral, 1982). A causality error is detected when a process receives a message that contains an event with a timestamp smaller than that of the last event executed by that process. Upon detection of a causality error a rollback mechanism is invoked. The event causing the rollback is called a straggler.

In order to recover, two things have to be done. The effects on the state of execution of events with timestamps larger than that of the straggler have to be undone, and the effects of messages sent as a result of executing these events have to be undone. Effects of messages are undone by sending so-called anti-messages. If the original message has not yet been processed when an anti-message arrives, then the two annihilate each other. If the original message has already been processed, the rollback mechanism is invoked, possibly leading to more anti-messages being sent. The procedure of repairing the local state and sending anti-messages is repeated by as many processes as necessary to completely undo the effects of prematurely executed events.

Reverting to a state visited in the past requires old states to be saved or a list to be maintained containing events that occurred in the past. This list can be used to reconstruct the desired state from the present one. No event with a timestamp smaller than the smallest timestamp of any unprocessed event in the system will ever have to be undone. So the smallest timestamp of the unprocessed events, also known as global virtual time or GVT, gives a bound on the amount of information to be maintained about the past. Algorithms for calculating and propagating GVT are discussed by Bellenot (1990).

Many variations on the time warp mechanism have been proposed and evaluated (Ferscha, 1995; Fujimoto, 1989b; Gafni, 1988; Panesar and Fujimoto, 1997; Reiher et al., 1990; Sokol and Stucky, 1990; Tay et al., 1997). The common goal of all these adaptations is to reduce the amount of time spent on undoing erroneous computations. Most result in improved performance for certain classes of models, while degrading the performance for others.
6.3 Approximate parallel simulation

We finish this section on PDES techniques with a discussion of the problems in the simulation of our model for surface processes by means of the time warp technique, focusing on the overhead introduced by the rollback mechanism.

There are at least two aspects in which our application differs from those traditionally considered in PDES: the amount of state information is large and the number of enabled events is large. Because of this, storing complete old states to facilitate fast rollback is not feasible. We can only store events performed in the past, and use these to restore the lattice to a previous state. This means that the cost of rollback is proportional to the number of events to undo, which, in turn, is proportional to the amount of simulated time over which the system is rolled back. Furthermore, in the case of a rollback, events have to be deleted from reaction sets, an operation which we know to be very costly (see Chapter 3). By tagging newly enabled events with the number of rollbacks that has been performed and tagging lattice sites with the rollback number when they are modified by a rollback, event deletion may be avoided. However, reactions may involve sites on multiple lattice chunks, requiring the rollback identification to be global. This too leads to significant overhead. A rollback over \( t \) time units may well take more time than simulating \( t \) time units, something that may lead to rollback thrashing (Fujimoto, 1990).

A high cost for rollbacks does not need to be a problem if the rollback mechanism is hardly ever invoked. Given the danger of thrashing, it seems doubtful that only few rollbacks will occur in the simulation of surface processes. There is another reason to suspect that the frequency of rollback invocations will be considerable. Consider a reaction \( e \) that involves two lattice chunks. Assume that the reaction's enabling location is situated at chunk \( A \), and that performing the reaction involves updating sites of chunk \( B \) as well. Denote the local time at \( A \) when \( e \) is selected to be performed by \( t_A \), and the local time at \( B \) by \( t_B \). If the simulated system is well-behaved, then the probability that \( t_A < t_B \) is \( \frac{1}{2} \), so the reaction causes a rollback with probability \( \frac{1}{2} \). If \( t_A > t_B \), then it is still not ruled out that \( e \) will cause a rollback. A timestamped message is sent from \( A \) to \( B \). It is possible that reactions at \( B \) modify sites of \( e \)'s pattern before this message is processed. In this case \( e \) and any reactions performed at \( A \) after \( e \) have to be undone.

6.3 Approximate parallel simulation

It seems that parallelizing simulations will not lead to the performance increase we are looking for if we insist on simulating the kinetics as given by our model correctly. The option we have left is to investigate parallel simulation algorithms that approximate the correct kinetics, thus trading correctness for performance.

We investigate an approach based on decomposition of the lattice into chunks, since this seems to be the only way to obtain a scalable algorithm. In fact, we investigate an algo-
Algorithm resembling the optimistic approach to PDES. Instead of detecting causality errors and invoking a rollback mechanism, we allow causality errors to occur.

When a reaction is selected that involves two lattice chunks, the processes controlling those chunks exchange information about the lattice state. It is likely that their local simulated times differ. Without further synchronization, there is no bound on the possible difference. We superimpose a global synchronization mechanism on the computation to prevent the local simulated times from differing too much. Maintaining a bound on the allowed time differences is especially important if reaction rates are time-dependent, such as in temperature-programmed desorption and reaction simulations. The situation must be avoided that some parts of the lattice are already in the high temperature regime, while others are still at low temperature. Similarly, if the rates are constant and the simulated system exhibits oscillatory behavior, then the different lattice chunks must be in the same phase of the oscillation for the simulation to give meaningful results. The frequency at which global synchronizations are performed must be larger than the frequency of the oscillations to establish this. Island formation too may lead to some problems. When islands are growing, different parts of the lattice exhibit different behavior. Reactivity at some areas may be substantially lower than at others. Using the proposed parallel simulation approach, the simulation may proceed much faster at some chunks than at others. A mechanism to synchronize the processes at regular intervals is required to ensure consistency of the lattice state.

It is unclear how large a time difference can be allowed between chunks. If it turns out that global synchronization is necessary after every distributed reaction, then apparently the suggested approach is not viable. If, on the other hand, no frequent global synchronizations have to be performed, we expect the algorithm to scale well to large numbers of processors because most communication is very local in nature. The amount of time that may be simulated between consecutive global synchronization operations will of course depend on the model being simulated. Above we already saw some situations in which large time differences may lead to incorrect simulation results. Below we describe the results obtained from parallel simulation of the ZGB model.

6.4 Parallel simulation of the ZGB model

In this section we describe in some detail how the ideas of the preceding section can be applied to the simulation of the ZGB model on distributed memory parallel machines. The results presented here extend those reported by Segers et al. (1996).

The reasons for experimenting with the ZGB model are twofold. The first reason is that the model has been studied at length (Jensen et al., 1990; Meakin and Scalapino, 1987; Ziff and Brosilow, 1992; Ziff et al., 1986). This allows us to compare our simulation results to
those reported in the literature. The second reason for experimenting with the ZGB model is that it is very simple. There are only two species, only seven reaction types, and the largest reaction patterns consist of only two sites. Despite its simplicity, the model gives rise to island formation and it includes immediate reactions. These two aspects make it interesting to study the effects of causality errors on the simulation results.

Chapter 5 contains a performance comparison of three sequential algorithms for the ZGB model, viz. WTselULsel-a, WTselULtrial, and Rsched-a. We develop a parallel version of one these three algorithms, specifically for the ZGB model. For reasons of simplicity we choose the WTselULtrial algorithm. In this algorithm no reactions are stored, but reactions are performed upon selection, if they are enabled. This relieves us of having to define when reactions are eligible, something which is not obvious for distributed reactions.

Before going into the details of the algorithm, a few more words on the programming and machine model that we use are in order. We assume that we have a (virtual) parallel machine in which each processor has its own, private memory. Multiple processes may be executed by every processor. The memory of a processor is shared between the processes it executes. We assume that inter processor communication is possible by message passing and by means of remote process calls. The former mechanism allows data to be exchanged between processes. The latter allows a process to start a new process on a different or on the same processor and inspect or manipulate variables on that processor. This abstract machine model and these communication facilities are offered by ECL (Lukkien and Hilbers, 1998), a communication library developed at Eindhoven University. ECL was designed with the SPMD (single program, multiple data) programming model in mind, a programming model that is convenient for our problem. In the SPMD programming model all processors execute the same program, on different data.

We assume that we have to simulate a square lattice of size $N \times N$. This lattice we divide into chunks. We choose chunks to be square too, of dimension $M \times M$, for $M$ a divisor of $N$. The reason for using square chunks is that they have a favorable ratio in the number of interior sites to boundary sites. Because the reaction patterns of the ZGB model have at most two sites, reactions involve at most two adjacent chunks. For models with larger reaction patterns a division of the lattice into square chunks may lead to more chunks being involved in one reaction; in this case the development of a parallel simulation algorithm can be simplified by dividing the lattice into stripes. If the lattice is divided into stripes, reactions with patterns smaller than the width of the stripes do not involve more than two chunks.

Each of the lattice chunks is assigned to a (virtual) processor. A simulation process is associated with every chunk, running on the processor that stores the chunk. These processes perform trials on their chunks according to the WTselULsel algorithm; every iteration of the complete WTselULtrial algorithm, including the handling of immediate reactions, constitutes a trial. The processes engage in a global synchronization operation at regular
intervals. These global synchronizations divide the simulation into a number of steps of fixed duration. The step length bounds the difference that may occur in local simulated times.

With the lattice divided into blocks, there are reactions in the ZGB model that involve two chunks. The origin of an oxygen adsorption or CO₂ production reaction may lie on one chunk and the other site of the pattern may lie on an adjacent chunk. If a process selects a distributed reaction, then it needs exclusive access to the sites involved in order to determine whether the reaction is enabled, and if so, to update the states of those sites. Exclusive access to lattice sites is provided by associating binary semaphores with the boundary sites.

Claiming a remote site involves communication between processors. The simplest solution is to use the remote process call mechanism. This mechanism gives us a 'first come first served' policy without having to implement a queuing mechanism. Figure 6.3 shows how a remote site is claimed using the remote process call mechanism. The simulation process that wants exclusive access to a remote site, performs an ECL-call operation. The ECL mechanism then takes care of starting a process on the destination processor. This process performs a P-operation on the semaphore associated with the site to be claimed. When the P-operation terminates, the claim process communicates the state of the claimed site to the simulation process that needs it (this is not depicted in the figure).

It is possible that two processes on adjacent processors want to claim the same pair of sites at the same time. As processes claim sites in a pair one by one, deadlock may occur. Consider, for example, the situation depicted in Figure 6.4. In this figure two processes, A and B, want to claim the pair of dark sites. If A has claimed the left one and B the right one, neither will be able to claim both and the processes are in a deadlock situation.

The danger of deadlock can be avoided by prescribing in which order the sites of a pair have to be claimed. The lexicographic order on site coordinates is sufficient. If this order is used, then process B does not claim the right site before having claimed the left site of the pair.
Figure 6.4: Example of deadlock in simulation of distributed reactions.

6.5 Simulation results and performance

The program for the parallel simulation of the ZGB model was written in C, using the communication primitives offered by ECL. As ECL runs on Inmos transputer networks (INMOS, 1988b), Parsytec PowerXplorers (PARSYTEC, 1995), and SUN Solaris workstations, the simulation program runs on these architectures as well. The SUN workstations are connected by an Ethernet and ECL allows multiple workstations to be used for one parallel program. The Transputer and PowerXplorer networks have grid topologies. Hence they are well-suited for running the simulation program, which is based on a grid decomposition of the lattice.

At the time the simulation experiments were performed, the ECL implementation for SUN workstations was not yet available and the PowerXplorer of the Parallel Systems Group at Eindhoven University consisted of only four nodes. A transputer system of 50 nodes was available. Most simulations have therefore been performed on the transputer system. Figure 6.5 shows the simulation results for a 256 x 256 lattice on a network of 7 x 7 transputers. A global synchronization was performed after every processor had performed $M^2$ trials on its $M \times M$ chunk. Instead of sampling an exponential distribution to obtain time increments, the average of the distribution was taken. This means that one step in the simulation corresponds to one Monte Carlo step.

The transition between the reactive phase and the oxygen-poisoned phase (see Section 5.3.1 for an explanation of the three phases observed in simulations of the ZGB model) occurs at $y_1 = 0.3903 \pm 0.0001$, while the transition between the reactive phase and the CO-poisoned phase occurs at $y_2 = 0.5266 \pm 0.0001$. When a 200 x 200 system is simulated on a 2 x 2 transputer or PowerXplorer network, the results are the same within the error mar-
Figure 6.5: Equilibrium surface coverage (vertical) vs. $y_{CO}$ (horizontal) for simulations of $256 \times 256$ systems on a $7 \times 7$ processor network.

Evaluating $y_1 = 0.3902 \pm 0.0001$ and $y_2 = 0.5267 \pm 0.0001$. These values are in agreement with the results of Ziff et al. (1986), who found $y_1 = 0.389 \pm 0.005$ and $y_2 = 0.525 \pm 0.001$. Ziff and Broslow (1992) report a different value for $y_2$ ($0.52560 \pm 0.00001$), but they use a different method to calculate this value. Ziff and Broslow (1992) explain that the value of $y_2$ increases with the system size. In view of this information it is not surprising that Meakin and Scalapino (1987) find $y_2 = 0.5277 \pm 0.0002$ for a $512 \times 512$ system.

Choosing the chunk dimensions smaller than $25 \times 25$ leads to a shift upwards in the value of $y_1$ and a shift downwards in the value of $y_2$. These phenomena should not just be attributed to the introduction of parallelism into the simulation algorithm. They are also observed in sequential simulations (Ziff and Broslow, 1992). No quantitative data is available on the extent to which the system size influences the transition points in sequential simulations. Therefore we cannot isolate the influence of the parallelism on the results.

Publications on the ZGB model all focus on equilibrium aspects and ignore dynamic aspects such as the evolution of island sizes. We also ignore these aspects in favor of a detailed performance analysis.

We know from the preceding chapter that for the WTSeULtrial algorithm, the time per
6.5 Simulation results and performance

![Graph showing performance of parallel simulation of the ZGB model on 2 x 2 PowerXplorer.](image)

Figure 6.6: Performance of parallel simulation of the ZGB model on 2 x 2 PowerXplorer.

reaction depends critically on the parameters of the model being simulated. Figure 5.4 shows how it rises sharply when the parameter \( y_{\text{CO}} \) of the ZGB model is decreased from 0.5 to 0.4. In an attempt to isolate the performance effects of \( y_{\text{CO}} \) and the overhead of parallelism, we express the performance in terms of the time per trial. As mentioned before, we define one trial as an iteration of the complete algorithm, including the computational effort for handling immediate reactions. The time per trial still depends on the reactivity of the system, but the effect is much smaller than on the time per reaction.

Figure 6.6 shows performance data obtained on the 2 x 2 PowerXplorer. We see that for large systems, the time per trial does not vary strongly with \( y_{\text{CO}} \). For small systems the dependence is much more pronounced. This can be attributed to the overhead of the parallel algorithm. If chunks are small, then the fraction of reactions that involve multiple chunks is relatively large. As the reactivity increases with rising \( y_{\text{CO}} \), the average number of communications per trial increases with \( y_{\text{CO}} \) as well.

The dependence of the performance on \( y_{\text{CO}} \) is hard to fit to a curve; the performance is more easily expressed as a function of \( N \), the linear dimension of the lattice, and the CO\(_2\) production rate (see Figure 6.5 for the dependence of the CO\(_2\) production rate on \( y_{\text{CO}} \)). The
surface of Figure 6.6 fits excellently to

\[
\tau(N, k_{CO_2}) = 2.267 \cdot 10^{-6} + 1.046 \cdot 10^{-5}k_{CO_2} \\
+ \frac{1}{N}(7.032 \cdot 10^{-5} + 6.272 \cdot 10^{-3}k_{CO_2}) \tag{6.1}
\]

For \(k_{CO_2}\) fixed, we obtain a curve of the form \(a + \frac{b}{N}\), as predicted in Section 6.2.3. In that curve \(a\) can be seen the average computation cost of a trial, while \(\frac{b}{N}\) can be interpreted as the average communication overhead per trial. The performance of the sequential algorithm, expressed in time per trial, does not depend on \(N\). At \(y_{CO} = 0.45\), the time per trial is 4.78 \(\mu s\) for the sequential algorithm run on one PowerXplorer node. The time per trial for the parallel algorithm on four nodes is 5.53 \(\mu s\) for \(N = 384\). The maximum observed speedup is less than 1. This confirms our suspicion that the overhead of the parallel algorithm is high.

As the maximum number of nodes available on the PowerXplorer machine is four, we analyze the scaling behavior of the algorithm on the transputer network, where 50 nodes are available. In order to investigate the scaling behavior of the algorithm, we plot a different performance curve. In Figure 6.7 the time per trial per processor is plotted as a function of the linear chunk dimension \(M\) and the linear dimension of the processor network \(Q\). Performance of the program is shown for simulations with \(y_{CO} = 0.500\).

The performance curve for the transputer network is less smooth than that for the PowerXplorer network. We do not have a detailed explanation for the roughness of the curve, but it can be partly attributed to the tranputer memory configuration.\(^2\)

The performance as a function of \(Q\), with \(M\) fixed is interesting to study: it shows the scaling behavior of the program with linear work scaling. Figure 6.8 shows the performance as a function of \(Q\) for a number of grid sizes.

For larger chunk sizes, the lines are almost horizontal, meaning that the algorithm scales well. This was to be expected, as most communication is between adjacent chunks. On the transputer network, these can be mapped onto adjacent processors. The cost of the synchronization phase increases with \(Q\). This has more influence for small \(M\) than for large, as the network was synchronized after every processor had performed \(M^2\) trials. In Figure 6.8 this is reflected by the higher slope of the graphs for smaller \(M\).

The sequential algorithm run on one transputer takes 63 \(\mu s\) per trial at \(y_{CO} = 0.525\). The parallel algorithm’s time per trial is 4.8 \(\mu s\) on 49 nodes with \(N = 600\), rising to 18 \(\mu s\) per trial for \(N = 84\). A 4 node transputer network takes 50\(\mu s\) per trial for \(N = 256\), increasing to 71 \(\mu s\) for \(N = 84\). For \(N \geq 128\) the parallel algorithm on four transputers outperforms the sequential algorithm. From these results we conclude that the introduction of parallelism may enable simulation of larger systems. It is doubtful whether the introduction of

\(^2\)In the T805 transputer used, the first 4 kbytes of memory have lower access time then the rest of the memory (INMOS, 1988a).
parallelism is a good way to speed up simulations that can also be run on one processor; the gain does not appear to be very large.

6.6 Conclusions

We analyzed the possibilities for introducing parallelism into simulation algorithms for surface processes. Parallel algorithms have two potential benefits: an increase in speed over sequential algorithms and the possibility to simulate larger systems.

Our investigation shows that the overhead of parallel algorithms is considerable. A functional decomposition of sequential algorithms is not viable because of the high communication latency of modern parallel computers. A data decomposition is not viable either. The overhead of preventing or breaking deadlock of conservative PDES approaches and that of rollback in optimistic approaches is too large to expect a speed increase over sequential simulations that is worth the effort of developing a parallel simulation system.

If we settle for simulations in which correctness of the simulated kinetics cannot be guaranteed, the overhead can be reduced. We proposed an approach in which parts of the lattice,
so-called chunks, are assigned to processors. In order to guarantee that the differences in simulated times between the processors do not grow beyond bound, global synchronization operations are performed regularly. The frequency at which global synchronization operations are performed gives a parameter to increase simulation speed at the cost of a higher probability of deviations between the simulated and actual kinetics. When reactions are performed that involve multiple lattice chunks, state information is exchanged. Although correctness of the kinetics is not guaranteed, consistency of the lattice state is.

Haider et al. (1995) use a similar technique to simulate epitaxial growth systems. The most notable difference with our approach is the introduction of ghost sites. A ghost site is a copy of a lattice site of a chunk residing on a different processor. When access is needed to a remote site, the local copy is used instead of performing communications. State information of boundary sites is copied to the corresponding ghost sites asynchronously instead. The result may be that the lattice state becomes inconsistent.

Von Hanxleden and Scott (1992) describe a parallel Monte Carlo simulation technique for a predator-prey system. They reject the optimistic PDES approach for performance reasons. They also reject our approach, saying that the overhead of distributed reactions is too high. Their own solution is to change the model they simulate into one that is easier to simulate by means of a parallel algorithm. More specifically, they allow multiple animals to be present
at one lattice site. This ensures that an attempt to move an animal to a nearest neighbor site will always succeed. In this approach the question of correctness of the simulation is translated into one about the equivalence of models. However, it is in general not clear how to transform models into ones that are easier to simulate by parallel algorithms and exhibit the same behavior.

We developed a parallel algorithm for simulation of the ZGB model to investigate the performance and correctness of our approach. It turns out that the overhead of the parallel algorithm is considerable, but that it scales well on a transputer network. This means that it is better suited to perform large simulations than to speed up smaller ones. Most communications performed during a simulation are of a local nature and small amounts of data are communicated. Many such communications occur, making a low communication latency important, not only in absolute numbers but also relative to the clock frequency of the processors. This is seen by comparing the performance of the algorithm on a $2 \times 2$ PowerXplorer network to that on a $2 \times 2$ transputer network. The communication latency of the PowerXplorer is slightly higher than that of the transputer, while its clock frequency is triple that of the transputer. The maximum observed speedup on the $2 \times 2$ PowerXplorer network is less than one, while on a $2 \times 2$ transputer network it is greater than one for lattices of size $128 \times 128$ or larger.

The transputer, and to a lesser extent the PowerXplorer, are outdated processors. They have clock frequencies of 25 MHz and 80 MHz, respectively, and they have small memories. Therefore, they are not suited for large scale simulations. More recent parallel machines, such as multiprocessor workstations, have an entirely different architecture. On these machines, improvements in communication latency do not nearly match improvements in processor speed over, for instance, the transputer. Also, the overhead of process creation is much larger than on the transputer. Hence, a parallel simulation program will have to have a different architecture for it to run efficiently on these machines. Instead of using the remote process call mechanism extensively, message passing should be used. Furthermore, the number of communications should be minimized. This can be done by using larger chunks so that the number of sites near a boundary decreases. In order to keep the exclusion protocol for distributed reactions simple for models with larger reactions patterns, the lattice may be decomposed into stripes. Reactions involve at most two stripes.

More experimentation is required to form a better judgment of our parallel simulation technique. In particular, we have not investigated any time-dependent quantities, such as island sizes. As the dynamics of systems is one of the main reasons for performing these simulations, dynamical aspects of the parallel algorithm have to be investigated.
Conclusion

In this chapter we review the results we achieved in the preceding chapters and identify some directions for further research. After this we discuss the applicability of our results to the simulation of discrete event systems in general.

7.1 Review of results

The goal of our research is efficient simulation of surface processes. More specifically, we set out to simulate the collective behavior of systems of many particles at single-crystal surfaces from a description of the behavior of individual particles. In Chapter 2 we gave a formal description of surfaces processes. This mathematical model contains three essential ingredients: descriptions of the surface, the particle types and the processes that may occur at the surface. The surface is modeled by a regular grid or lattice. Particles may adsorb at the lattice points or sites of the grid. Each process or reaction type is described by a transformation type and a rate constant. The transformation type defines how the process changes the assignment of particles to the surface and the rate constant defines the time scale at which transformations occur if they are possible.

The model presented in Chapter 2 allows a large collection of surface reactions to be described. It can be adapted in a straightforward way to allow the description of even more reaction systems. A generalization that may be of interest is modeling three-dimensional catalysts, such as zeolites. Systems with extensive lateral interactions are less easily captured. Formally they form no problem, in practice they do. If the rate constant of, e.g. a diffusion step, depends on the presence or absence of nearest and next nearest neighbors
for the moving particle, many reaction types have to be introduced: one for every possible environment.

The kinetics of systems are described in a stochastic way. From the assumption that a reaction with rate constant $k$ has probability $k dt$ of occurring in an infinitesimal interval of length $dt$, we derived expressions for the waiting times of reactions, and for the probability of an arbitrary reaction occurring before any other reaction. These expressions give two ways to describe the kinetics of systems. First, the reaction kinetics can be described by the probability distribution of the waiting times of the individual reactions. Second, the kinetics are described by the combination of the waiting time distribution of the next reaction and probabilities for each individual reaction being the next to occur.

Both descriptions of the kinetics lead to simulation algorithms in a straightforward way. In Chapter 3 we presented three techniques that can be used to transform the basic simulation algorithms into more efficient ones: oversampling, maintaining a superset of the enabled reactions, and selecting elements of the set of enabled reactions hierarchically. None of these techniques are new. Previous authors, however, have not discussed the techniques within one framework. The benefits of having a framework to describe the algorithms are twofold: it is clear how to show that proposed algorithms are correct and it allows their performance to be compared.

Having at our disposal three ways to improve the performance of simulation algorithms, it is natural to ask whether they can be combined to obtain further improvements in performance. As it turns out there are many possibilities. We discussed some possibilities and the resulting algorithms in detail. Others we included mainly because they show how algorithms published in the literature can be placed in our framework. The systematic exploration of the possible combinations of techniques led to two important results: an algorithm that allows more extensive lateral interactions to be incorporated in simulations (WSTselULsel-a) and an algorithm that allows different simulation techniques to be used for different reaction types within one simulation (Gsched-hao).

The idea behind the WSTselULsel-a algorithm is to improve the efficiency of simulation of systems with lateral interactions by concentrating on reaction effects. The rate constant of a diffusion step, for example, may depend on the particles by which the moving particle is surrounded. But no matter what particles are in its environment, the effect is the same: the particle moves to an nearby site. Only the rate constant of the step depends on the environment. In WSTselULsel-a, reactions that have the same effect are replaced by one new reaction that describes exactly that effect. When this new reaction is selected during the simulation, its rate constant is calculated, requiring the environment to be inspected. One way to do this is to compute which reactions with the desired effect are enabled and add their rate constants. Although this is the easiest solution, it is not necessarily the best, especially if many reactions have the same effect. It would be better to replace the pattern matching by an algorithmic specification. This is easily done if the interaction is pairwise
7.1 Review of results

additive, but further investigation is needed for other forms of interaction.

The WSTselULsel-a algorithm is not the best algorithm for every reaction system. If there are no reaction types with the same effect, then it is better to use another algorithm. It is also possible that is advantageous to combine only some reaction types with the same effect, and to use a different algorithm for other reaction types. The Gsched-hao algorithm shows that it is indeed possible to combine different algorithms in one simulation.

Chapter 4 focuses on a generalization of the model, by allowing rate constants of reactions to be time-dependent. We were able to derive probability distributions for waiting times again to obtain a correctness criterion for simulation algorithms. Some rather involved calculation showed that the techniques for improving performance discussed in Chapter 3 also apply to the simulation of systems with time-dependent rate constants. The oversampling technique has not been used before for the case that rate constants are time-dependent. Combining hierarchical selection with approximate representation of the enabled reactions has not been done before either. These improvements over the algorithms used in the literature are significant: they allow algorithms for which the expected time per reaction performed does not depend on the lattice size. A surprising result in Chapter 4 is that the most efficient algorithms for simulating systems with time-independent rate constant, do not result in the most efficient algorithms when generalized to the case that rate constants are time-dependent.

The current version of the simulation tool CARLOS implements three of the simulation algorithms described in this monograph, viz. WTselULtria, WTselULsel-a, and tdRsched-a. The last of the three can also be used for time-independent systems, but for those it performs worse than the implementation of WTselULsel-a, as our performance model predicts. In the future, CARLOS should be extended so that the performance of algorithms such as WSTselULsel-a can be evaluated. Moreover, it should be made possible to use WSTselULsel-a in conjunction with WTselULtria and WTselULsel-a in one simulation, by implementing the Gsched-hao algorithm (in the current implementation WTselULtria and WTselULsel-a can be combined already). The performance evaluation described in Chapter 5 also showed the importance of some optimization techniques. Two kinds of optimizations are discussed: improvements in the computation of newly enabled reactions and placing limits on the size of $E$, the set containing the enabled reactions. The former optimization decreases the time per reaction, at the cost of some extra memory. The latter optimization leads to a reduction in memory use, and, in many cases also to a decrease in the time per reaction.

In view of the rising interest in time-dependent simulations from the field of interfacial electrochemistry and the performance analysis results of Chapter 4, the tdTschedULsel-a and tdTschedULtria algorithms should be implemented as well. The tdTschedULsel-a algorithm by itself already promises a substantial improvement over the algorithm used currently because it exploits the fact that the number of reaction types is generally much
smaller than the number of enabled reactions. The results presented in Chapter 5 for systems with time-independent rate constants indicate that the combined use of tdt_schedULsel and tdt_schedULtrial will offer another substantial improvement.

A further source of performance improvement might be found in the application of parallel simulation algorithms. Chapter 6 describes the results of a pilot study, indicating that domain decomposition techniques offers performance benefits, without influencing the simulation results, on the ZGB model for CO oxidation. Whether this holds for other models as well is as yet an open question, one that deserves attention in the future.

7.2 Applicability of the results

Our attention has been focused exclusively on the simulation of the time evolution of surface reaction systems. This section is devoted to the question whether the techniques we developed are applicable to other simulation problems as well.

In Chapter 3 we briefly mentioned classical Monte Carlo simulations as a technique for calculating properties of systems at equilibrium. This requires sampling the state space of the system at hand in such a way that a state \( \sigma \) gets weight \( \frac{\mathcal{H}(\sigma)}{Z} \), where \( \mathcal{H} \) is the Hamiltonian describing the system and \( Z = (\Sigma \sigma : \sigma \in \Sigma : \mathcal{H}(\sigma)) \) its partition function. In practice, samples are not taken independently. It is computationally more efficient to generate a sequence of states, where each new state is obtained by making a small change to the current state. If the probabilities of going from a state \( \sigma \) to a state \( \sigma' \) and vice versa satisfy the detailed balance condition given by

\[
\mathcal{H}(\sigma)w(\sigma \rightarrow \sigma') = \mathcal{H}(\sigma')w(\sigma' \rightarrow \sigma),
\]

then all states are visited with the correct relative frequencies. If the state space \( \Sigma \) is discrete, as we already implicitly assumed by writing \( Z \) as a sum over states, then the transition probabilities \( w(\sigma \rightarrow \sigma') \) can be used as rate constants and our simulation techniques apply directly. In areas such as the simulation of spin systems (see, e.g., Binder, 1979a), the state is described by a lattice of spin values. Techniques for reducing the amount of work spent on computing newly enabled transitions as mentioned in Chapter 5 then apply too. Moreover, it is possible to define reaction types as sets of transitions that have the same transition probability, an observation that led Bortz et al. (1975) to their so-called ‘n-fold way’ algorithm.

As noted in Chapter 2, the model we use for surface processes is an example of a discrete event system with continuous time. The question rises whether the simulation algorithms that we developed are applicable to other discrete event systems. The answer to this question is affirmative. Our algorithms apply to the simulation of any discrete event system that
7.2 Applicability of the results

has a discrete state and in which event waiting times are either zero (corresponding to our immediate reactions) or have (td)Exp distributions.

A more interesting question is whether discrete event systems of this type are used and whether performance is an issue in simulation of those systems. One area in which discrete event systems with exponentially distributed event waiting times are used is in the performance analysis of computer systems and networks. The components of such systems are typically described in Petri net based formalisms. A number of modeling and analysis tools have been developed, the functionality of which includes (timed) event simulation of the models created by the users. These tools includes GreatSPN (Balbo and Chiola, 1989; Chiola, 1991; Chiola et al., 1995), TimeNet (German et al., 1995), and UltraSAN (Sanders and Freire, 1993), to name but a few. In all these tools, either the standard discrete event simulation algorithm (Rsched) or WRsel is used. The authors have only sought to improve the efficiency of the simulations by means of improving the event list management. The techniques we used to improve efficiency of the basic algorithms may be applied here as well.

Most modeling and simulation tools provide ways to construct hierarchical models of systems. In particular, they allow components of systems to be made up of subcomponents and they have replication operators to facilitate, for instance, modeling of networks consisting of many identical components. This hierarchical way of modeling allows a more efficient way of simulation than the standard discrete event simulation, using hierarchical event selection. Systems consisting of many identical components may also benefit from using an algorithm in which no event list is maintained, as in the WTselULtrial algorithm.
A

Probability Theory

This thesis deals with a stochastic model for surface reactions. In the description of the model and the simulation algorithms, some concepts and properties from probability theory are used. This appendix contains all the concepts and properties used. The intention is to make the thesis more self-contained by including this appendix, without distracting the attention from the ideas and results to be communicated in the main text.

The appendix starts with the informal introduction of some well-known concepts from probability theory and some notations for them. Readers interested in a more thorough treatment are referred to the textbooks on this subject, such those by Feller (1966) or Galambos (1995). The concepts introduced are then used to formulate some properties. Most of these can be found in textbooks on probability theory, but some of them are, apparently, less common. Their proofs are included.

A.1 Stochastic variables and distribution functions

The reader is assumed to be familiar with the notion of random variables and probability distributions. Probability distributions define the probability of random variables taking certain values. The distribution function of random variable (r.v.) $X$ is denoted by $F_X$. The probability of $X$'s value being at most $x$ equals $F_X(x)$, also written as $\mathbb{P}(X \leq x)$.

A random variable is called discrete if it takes values in a denumerable subset of $\mathbb{R}$. For discrete r.v. $X$ the probability function $f_X$ is defined by $f_X(x) = \mathbb{P}(X = x)$ for $x$ in the value set of $X$. Many common continuous random variables can be characterized by the
continuous equivalent of the probability function, the so-called probability density function or pdf. The relationship between the distribution function and pdf of r.v. X is the following:

$$F_X(x) = \int_{-\infty}^{x} f_X(u)du,$$

for $x \in \mathbb{R}$.

Probability distributions over $\mathbb{R}^2$ are defined using pairs of random variables. For random variables $X$ and $Y$ and for $x, y \in \mathbb{R}$, the joint distribution function $F_{X,Y}(x, y)$ defines the probability that $X \leq x$ and $Y \leq y$, also written as $\mathbb{P}[X \leq x \land Y \leq y]$. The distribution functions $F_X$ and $F_Y$ are called the marginal distributions of $X$ and $Y$ in this context. The two components $X$ and $Y$ are said to be independent if $F_{X,Y}(x, y) = F_X(x)F_Y(y)$ for all $x$ and $y$.

Let $X$ and $Y$ be discrete random variables, and let $x$ and $y$ be values in their respective value sets. Assume that $\mathbb{P}[Y = y] \neq 0$. The conditional probability of $X$ being equal to $x$ given that $Y = y$ is defined by

$$\mathbb{P}[X = x|Y = y] = \frac{\mathbb{P}[X = x \land Y = y]}{\mathbb{P}[Y = y]}.$$

As a consequence of this definition we have

$$f_X(x) = (\Sigma y : y \in S : f_X(x|Y = y)f_Y(y)),$$

where $f_X(x|Y = y)$ is an alternative notation for $\mathbb{P}[X = x|Y = y]$ and $S$ is $Y$'s value set. Conditional distributions can also be defined on pairs of continuous r.v.'s or pairs consisting of a discrete and a continuous r.v. For instance, for the case that $X$ and $Y$ are both continuous we have

$$f_X(x|Y = y) = \frac{f_X(x)}{f_Y(y)}$$

for $y$ such that $f_Y(y) \neq 0$, and

$$f_X(x) = \int_{-\infty}^{\infty} f_X(x|Y = y)f_Y(y)dy.$$

We consider two examples of discrete random variables. First, imagine an experiment with two possible outcomes: success or failure. The probability of success is $p$, $0 \leq p \leq 1$; the probability of failure is $1 - p$. Now we define a random variable that has value 0 if the experiment fails and value 1 if it succeeds. This means that $f_X(0) = 1 - p$, $f_X(1) = p$, and
A.1 Stochastic variables and distribution functions

\( f_X(x) = 0 \) for any other \( x \in \mathbb{R} \). The distribution defined by \( f_X \) is known as the Bernoulli distribution with parameter \( p \).

Next let \( X_1, X_2, \ldots \) be independent r.v.'s, all with a Bernoulli(\( p \)) distribution. Consider the minimum index \( i \) for which \( X_i = 1 \). This is itself a random variable, say \( N \). The value set of \( N \) is \( \mathbb{N} - \{0\} \), and the probability that \( N \) equals \( n \) is \( (1 - p)^{n-1} p \) for \( n \) in this value set. The distribution of \( N \) is known as the geometric distribution with parameter \( p \). We refer to it as the Geom(\( p \)) distribution. Geom(\( p \))-distributed variable \( N \) has the following property:

\[
P[N > n + m | N > n] = P[N > m].
\]

This property is known as lack of memory.

An important continuous distribution is the exponential distribution. A r.v. \( X \) has an Exp(\( \lambda \)) distribution if \( F_X(x) = \exp(-\lambda x) \) for \( x \geq 0 \) and \( F_X(x) = 0 \) for \( x < 0 \). Only positive values are allowed for \( \lambda \). The pdf of Exp(\( \lambda \)) distributed r.v. \( X \) is

\[
f_X(x) = \begin{cases} 
0 & \text{for } x < 0, \\
\lambda \exp(-\lambda x) & \text{for } x \geq 0.
\end{cases}
\]

In Chapter 3 we used the following property of exponential distributions.

Property A.1

Let \( X_0 \) and \( X_1 \) be independent r.v.'s with Exp(\( \lambda_0 \)) and Exp(\( \lambda_1 \)) distributions, respectively. Then

\[
P[X_0 < X_1] = \frac{\lambda_0}{\lambda_0 + \lambda_1},
\]

for \( t \geq 0 \).

Proof:

We calculate as follows:

\[
P[X_0 < X_1] = \int_{x_0 < x_1} \lambda_0 \exp(-\lambda_0 x_0) \lambda_1 \exp(-\lambda_1 x_0) dx_0 dx_1
\]

\[
= \lambda_0 \lambda_1 \int_0^\infty \exp(-\lambda_0 x_0) dx_0 \int_{x_0}^\infty \exp(-\lambda_1 x_1) dx_1
\]

\[
= \lambda_0 \int_0^\infty \exp(-(\lambda_0 + \lambda_1) x_0) dx_0
\]

\[
= \frac{\lambda_0}{\lambda_0 + \lambda_1}.
\]
In Chapter 4 we encountered a generalization of the exponential distribution in the description of the waiting time of reactions with time-dependent rates. The generalization lies in the fact that the parameter of the distribution, $\lambda$, is not constant, but a positive valued function. In this view $\lambda$ can be interpreted to describe a time-dependent parameter of the distribution. We get the following distribution function (see also Equation (4.3) on page 95).

$$F_X(t, x) = 1 - \exp(-\int_t^{t+x} \lambda(u)du),$$

for $t \geq 0$ and $x \geq 0$. Note that $X$ is now a random function of $t$. With this definition, $F_X(t)$ may be a defective distribution, meaning that it does not satisfy the normalization property

$$\lim_{x \to \infty} F_X(t, x) = 1.$$

In order to remedy this defectiveness, we extend the value range of $X(t)$ with a value $\infty$. We define $\infty$ to be larger than any real number and define $F_X(t)$ by

$$F_X(t, x) = 0 \quad \text{if } x \in (\infty, 0),$$

$$F_X(t, x) = \exp(-\int_t^{t+x} \lambda(u)du) \quad \text{if } x \in [0, \infty),$$

$$F_X(t, \infty) = \lim_{x \to \infty} F_X(t, x).$$

A random variable with this distribution is said to have a time-dependent exponential distribution\footnote{There does not seem to be a widely accepted name for this distribution; it is dubbed a time-dependent exponential distribution here because it is related to the (standard) exponential distribution.} with parameter $\lambda$, or a tdExp($\lambda$) distribution for short. The tdExp distributions are memoryless. I.e. if $X$ has a tdExp($\lambda$) distribution, then

$$\mathbb{P}[X(t) > x + y | X(t) > x] = \mathbb{P}[X(t + x) > y],$$

for $t, x, y \geq 0$ such that $\mathbb{P}[X(t) > x] > 0$.

In Chapters 3 and 4 we used the following property of the exponential distribution.

**Property A.2**

Let $X_0$ and $X_1$ be independent with tdExp($\lambda_0$) and tdExp($\lambda_1$) distributions, respectively. Define $Y = \text{MIN}\{X_0, X_1\}$. Then $Y$ has a tdExp($\lambda_0 + \lambda_1$) distribution.

**Proof:**

First we calculate the probability that $Y(t)$ equals $\infty$, for $t \geq 0$:

$$\mathbb{P}[Y(t) = \infty]$$
A.1 Stochastic variables and distribution functions

\[
\begin{align*}
= & \quad \{ \text{Definition of } Y \} \\
\mathbb{P}(X_0(t) = \infty \land X_1(t) = \infty) = & \quad \{ X_0 \text{ and } X_1 \text{ are independent} \} \\
\mathbb{P}(X_0(t) = \infty) \mathbb{P}(X_1(t) = \infty) = & \quad \{ X_0 \text{ and } X_1 \text{ have tdExp distributions} \} \\
\lim_{x \to \infty} \exp(- \int_0^x \lambda_0(u)du) \lim_{x \to \infty} \exp(- \int_0^x \lambda_1(u)du) = & \quad \{ \text{Both limits are finite} \} \\
\lim_{x \to \infty} \exp(- \int_0^x (\lambda_0(u) + \lambda_1(u))du) = & .
\end{align*}
\]

Next, for \( y \in \mathbb{R} \) and nonnegative, we calculate as follows:

\[
\begin{align*}
\mathbb{P}(Y(t) > y) = & \quad \{ \text{Definition of } Y; \text{ independence} \} \\
\mathbb{P}(X_0(t) > y) \mathbb{P}(X_1(t) > y) = & \quad \{ X_0 \text{ and } X_1 \text{ have tdExp distributions} \} \\
\exp(- \int_t^{t+y} \lambda_0(u)du) \exp(- \int_t^{t+y} \lambda_1(u)du) = & \quad \{ \text{Calculus} \} \\
\exp(- \int_t^{t+y} (\lambda_0(u) + \lambda_1(u))du) = .
\end{align*}
\]

Since \( X_0 \) and \( X_1 \) only take nonnegative values, the probability of \( Y \) being negative is zero. \( \square \)

The Gamma distribution is a distribution related to the exponential distribution. A random variable \( X \) has a Gamma(\( r, \alpha \)) distribution if it has a pdf \( f_X \) satisfying

\[
f_X(x) = \begin{cases} 
\frac{1}{\Gamma(r)} \alpha^r x^{r-1} \exp(-\alpha x) & \text{if } x \geq 0, \\
0 & \text{if } x < 0,
\end{cases}
\]

for \( r > 0 \) and \( \alpha > 0 \). The Gamma function \( \Gamma \) is defined by

\[
\Gamma(r) = \int_0^{\infty} x^{r-1} \exp(-x) dx ,
\]

for \( r \in \mathbb{R} \). If \( X_1, \ldots, X_n \) are independent r.v.'s, all with \( \text{Exp}(\lambda) \) distributions, then their sum has a Gamma(\( n, \lambda \)) distribution. This is easily verified by induction over \( n \).
A.2 Moments, probability generating functions and the Laplace transform

Let $X$ be a discrete random variable with probability function $f_X$ and value set $S$. The $k^{th}$ moment of $X$, written $\mathbb{E}(X^k)$, is defined as

$$\mathbb{E}(X^k) = \left( \sum x : x \in S : x^k f_X(x) \right),$$

provided that the sum on the right hand sides converges. For continuous $X$, the $k^{th}$ moment is defined as

$$\mathbb{E}(X^k) = \int_{-\infty}^{+\infty} x^k f_X(x) dx,$$

provided that the integral converges.

The first moment of a random variable is also known as its moment or its mean. First moments have some nice properties, of which we mention three.

1. For random variables $X$ and $Y$ and arbitrary constant $a$ and $b$, the first moment of $aX + bY$ equals $a \mathbb{E}(X) + b \mathbb{E}(Y)$.

2. For continuous r.v. $X$ and function $g$ defined on the value set of $X$, the moment of random variable $g(X)$ equals

$$\int_{-\infty}^{+\infty} g(x) f_X(x) dx.$$

A similar property holds for discrete $X$.

3. If $X$ and $Y$ are independent, then $\mathbb{E}(XY) = \mathbb{E}(X) \mathbb{E}(Y)$.

If $X$ is a r.v. for which the first and second moments are defined, then its variance is defined as

$$\text{var}(X) = \mathbb{E}((X - \mathbb{E}(X))^2).$$

In the previous section we saw how conditional distributions are defined. These are used to define conditional moments. The conditional moment of $g(X)$ given that $Y = y$ is the moment of the conditional distribution of $g(X)$ given that $Y = y$:

$$\mathbb{E}(g(X) \mid Y = y) = \int_{-\infty}^{+\infty} g(x) f_X(x \mid Y = y) dx,$$
for the case that $X$ is continuous. If $Y$ is discrete with value set $S$, then $\mathbb{E}(g(X))$ can be calculated from the conditional moments $\mathbb{E}(g(X) \mid Y = y)$ by using the identity

$$\mathbb{E}(g(X)) = (\sum y : y \in S : \mathbb{E}(g(X) \mid Y = y) \mathbb{P}(Y = y)) .$$

Similar identities hold for the cases that $Y$ is continuous, and for $X$ discrete and $Y$ either discrete or continuous.

The probability generating function $P_X$ of discrete r.v. $X$ with values in $\mathbb{N}$ is defined by $P_X(z) = \mathbb{E}(z^X)$. The Laplace transform $\phi_X$ of continuous r.v. with $f_X(x) = 0$ for $x < 0$ is defined by $\phi_X(s) = \mathbb{E}(\exp(-sX))$ for $0 \leq s$. Without proof we mention that two discrete (continuous) r.v.'s that have the same probability generating functions (Laplace transforms), also have the same distribution functions.

As an example we calculate the Laplace transform of random variable $X$ with an Exp($\lambda$) distribution:

\[
\phi_X(s) = \int_0^\infty \exp(-sx)f_X(x)dx
\]

\[
= \begin{cases} \text{Definition of } \phi ; f_X(x) = 0 \text{ for } x < 0 \\ \int_0^\infty \exp(-sx)f_X(x)dx \end{cases}
\]

\[
= \begin{cases} X \text{ has an Exp}(\lambda) \text{ distribution} \\ \lambda \int_0^\infty \exp(-s(-x))dx \end{cases}
\]

\[
= \begin{cases} \text{Calculus} \\ \frac{\lambda}{s} \end{cases}
\]

The following property we use in the proof of Property 3.2.

Property A.3

Let $K$ be a discrete r.v. with values in $\mathbb{N} - \{0\}$ and let $X_1, X_2, \ldots$ be continuous r.v.'s. Assume that $K$ and the $X_k$ are independent and that $f_{X_k}(x) = 0$ for $x < 0$. Define $W$ as

$$W = (\Sigma k : 1 \leq k \leq K : X_k) .$$

Then

$$\phi_W(s) = (\Sigma k : 1 \leq k : (\Pi i : 1 \leq i \leq k : \phi_{X_i}(s)) \mathbb{P}(K = k)) .$$

Proof:

We derive
\[ \phi(s) \]

\[ = \begin{cases} \text{Definition of } \phi \end{cases} \]

\[ \mathbb{E}(\exp(-s W)) \]

\[ = \begin{cases} \text{Condition to } K \end{cases} \]

\[ (\sum_{k : 1 \leq k} : \mathbb{E}(\exp(-s W) \mid K = k) \mathbb{P}(K = k)) \]

\[ = \begin{cases} \mathbb{E}(\exp(-s W) \mid K = k) = \mathbb{E}(\exp(-s(X_1 + \cdots + X_k))) \end{cases} \]

\[ (\sum_{k : 1 \leq k} : \mathbb{E}(\exp(-s(X_1 + \cdots + X_k))) \mathbb{P}(K = k)) \]

\[ = \begin{cases} \text{Independence of the } X_i \end{cases} \]

\[ (\sum_{k : 1 \leq k} : (\prod_{i : 1 \leq i \leq k} : \mathbb{E}(\exp(-s X_i))) \mathbb{P}(K = k)) \]

\[ = \begin{cases} \text{Definition of } \phi \end{cases} \]

\[ (\sum_{k : 1 \leq k} : (\prod_{i : 1 \leq i \leq k} : \phi_{X_i}(s)) \mathbb{P}(K = k)) . \]

If, in the above property, all \( X_i \) are identically distributed as some \( X \), then the expression simplifies considerably.

**Corollary A.4**

Let \( K \) be a discrete r.v. with values in \( \mathbb{N} - \{0\} \) and let \( X_1, X_2, \ldots \) be continuous r.v.'s. Assume that \( K \) and the \( X_k \) are independent, and that all \( X_k \) have the same distribution as some r.v. \( X \). Assume that \( f_X(x) = 0 \) for \( x < 0 \). Define \( W = (\sum_{k : 1 \leq k \leq K} : X_k) \). Then

\[ \phi_W(s) = (\sum_{k : 1 \leq k} : (\phi_X(s))^k \mathbb{P}(K = k)) \]

\[ = P_K(\phi_X(s)) . \]

**A.3 Proofs of Properties 3.1, 3.2, 4.1, and 4.2**

The material of the preceding sections enables us to prove the various properties used in Chapters 3 and 4.

**Property 3.1**

Let \( X_1, X_2, \ldots \) be independent random variables, all with probability density function
A.3 Proofs of Properties 3.1, 3.2, 4.1, and 4.2

\[ f_{X_i}(x) = \lambda \exp(-\lambda x), \text{ for some } \lambda > 0. \text{ Let } K \text{ be a random variable, independent of the } X_i, \text{ with a geometric distribution with success probability } p. \text{ I.e.,} \]

\[ \mathbb{P}[K = k] = (1 - p)^{k-1} p, \]

for integer \( k \geq 1. \) Define \( W = (\sum k : 1 \leq k \leq K : X_k). \) Then \( W \) has an \( \text{Exp}(p\lambda) \) distribution.

Proof:
According to Corollary A.4 we have

\[ \phi_W(s) = P_K(\phi_X(s)). \]

The example on page 173 shows that the Laplace transform of \( X \) equals \( \frac{\lambda}{\lambda + s}. \) We expand the definition of \( P_K \) and substitute this result into it:

\[
\begin{align*}
\phi_W(s) &= (\sum k : 1 \leq k : \left(\frac{1}{\lambda + s}\right)^k (1 - p)^{k-1} p) \\
&= \left\{ \text{Calculus} \right\} \\
&= \frac{\rho \lambda}{\lambda + s} (\sum k : 0 \leq k : \left(\frac{(1-p)\lambda}{\lambda + s}\right)^k) \\
&= \left\{ 0 \leq \frac{(1-p)\lambda}{\lambda + s} < 1 \right\} \\
&= \frac{\rho \lambda}{\rho \lambda + s} \\
&= \frac{\rho \lambda}{\rho \lambda + s}.
\end{align*}
\]

This is the Laplace transform of the \( \text{Exp}(p\lambda) \) distribution. \( \square \)

Property 3.2 is less well-known.

Property 3.2

Let \( X_1, X_2, \ldots \) be independent random variables, with probability density functions

\[ f_{X_i}(x) = \lambda_i \exp(-\lambda_i x), \]

with \( \lambda_i > 0 \) for \( i \geq 1. \) Let \( \lambda > 0 \) such that \( \lambda \leq \lambda_i \) for all \( i \geq 1. \) Next let \( Z_1, Z_2, \ldots \) be independent random variables, independent of the \( X_i, \) with

\[ \mathbb{P}[Z_i = 1] = 1 - \mathbb{P}[Z_i = 0] = \frac{\lambda}{\lambda_i}. \]
Define random variables $K$ and $W$ by

$$
K = (\text{MIN } i : 1 \leq i \land Z_i = 1 : i), \text{ and }
W = (\Sigma k : 1 \leq k \leq K : X_k).
$$

Then $W$ has an exponential distribution with parameter $\lambda$ if and only if

$$
\lim_{k \to \infty} (\Pi i : 1 \leq i \leq k : 1 - \frac{1}{\lambda_i}) = 0 \tag{3.3}
$$

In words, $W$ has an exponential distribution if and only if the probability of infinitely many unsuccessful trials is zero.

Before giving a formal proof, deriving condition (3.3), we give an intuitive argument. The $X_i$ represent waiting times of events and the $Z_i$ represent acceptance probabilities of events. For $w \geq 0$ and $\delta w$ small, the probability that the waiting time of the next accepted event $W$ is in $[w, w + \delta w)$ equals $\frac{\lambda_i}{\lambda_i + 1} \delta w = \lambda \delta w$ for some $i \geq 1$. The factor $\frac{1}{\lambda_i}$ gives the probability of the $i^{th}$ event being accepted ($Z_i = 1$), and $\lambda_i \delta w$ is the probability of the $i^{th}$ event occurring in an interval of size $\delta w$. Having established that the probability of the next accepted event occurring in an interval of size $\delta w$ is $\lambda \delta w$, we can now conclude that $W$ has an $\text{Exp}(\lambda)$ distribution.

The argument above is not very rigorous. We have, for instance, ignored the possibility that no event is accepted. For this reason we present a more detailed proof, as found by Steutel and Brands (1998).

**Proof:**

The probability function of $K$ is given by

$$
f_K(k) = (\Pi i : 1 \leq i < k : 1 - \frac{1}{\lambda_i}) \frac{\lambda_i}{\lambda_i + 1}, \tag{A.1}
$$

for $1 \leq k$. We use this in calculating $W$'s Laplace transform.

$$
\phi_W(s)
= \{ \text{Property A.3} \}
= (\Sigma k : 1 \leq k : (\Pi i : 1 \leq i \leq k : \phi_{X_i}(s)) \mathbb{P}(K = k))
= \{ (A.1), X_i \text{ is } \text{tdExp}(\lambda_i) \text{ distributed} \}
= (\Sigma k : 1 \leq k : (\Pi i : 1 \leq i < k : (1 - \frac{1}{\lambda_i}) \frac{\lambda_i}{\lambda_i + 1}) \frac{\lambda_i}{\lambda_i + 1 + s})
= \{ \text{Define } a_i = \lambda_i - \lambda \text{ for } 1 \leq i \}
= (\Sigma k : 1 \leq k : (\Pi i : 1 \leq i < k : \frac{a_i}{a_i + \lambda_i + s}) \frac{\lambda}{a_i + \lambda_i + s})
= \{ \text{Define } b_1 = \frac{1}{a_1 + \lambda + s} \text{ and } b_{k+1} = \frac{a_k}{a_{k+1} + \lambda + s} b_k \text{ for } 1 \leq k \}
= \lambda (\Sigma k : 1 \leq k : b_k).
We rewrite the definition of $b_{k+1}$ to obtain

$$(\lambda + s)b_{k+1} = a_kb_k - a_{k+1}b_{k+1}.$$  

Using this expression, we calculate the partial sums of the sequence $b$:

$$(\lambda + s)(\sum i \leq i \leq k \ : \ b_k) = (\lambda + s)b_1 + a_1b_1 - a_kb_k$$  
$$= 1 - a_kb_k.$$  

This result we substitute in the expression we derived for $\phi_W(s)$ to get Equation (A.2).

$$\phi_W(s) = \frac{\lambda}{\lambda + s} \lim_{k \to \infty} (1 - a_kb_k)$$  \hspace{1cm} (A.2)  

From this equation we conclude that $W$ has an $\text{Exp}(\lambda)$ distribution if and only if the limit of $a_kb_k$ equals zero for $k \to \infty$ for all $s \geq 0$. We simplify this condition:

$$(\forall s \ : \ 0 \leq s \ : \ \lim_{k \to \infty} a_kb_k = 0)$$

$$= \quad \{ \text{Definitions of } a \text{ and } b \}$$

$$(\forall s \ : \ 0 \leq s \ : \ \lim_{k \to \infty} (\Pi i \ : \ 1 \leq i \leq k \ : \ \frac{\lambda_i - \lambda}{\lambda_i + s}) = 0)$$

$$= \quad \{ \text{Predicate calculus} \}$$

$$\lim_{k \to \infty} (\Pi i \ : \ 1 \leq i \leq k \ : \ \frac{\lambda_i - \lambda}{\lambda_i}) = 0$$

$$= \quad \{ \}$$

$$\lim_{k \to \infty} (\Pi i \ : \ 1 \leq i \leq k \ : \ 1 - \frac{\lambda}{\lambda_i}) = 0,$$

finishing the proof.


Next we focus on the properties used in Chapter 4. We start with Property 4.2. Property 4.1 is a special case of this, and its proof follows from that of Property 4.2.

**Property 4.2**

Let $X_1, X_2, \ldots$ be independent random variables, $X_i$ having a $\text{tdExp}(\lambda_i)$ distribution. Let $\lambda$ be a function such that $\lambda(x) \leq \lambda_i(x)$ for all $i$ and for all $x \leq 0$. Let $Z_1, Z_2, \ldots$ be independent random variables, independent of the $X_i$, with

$$\mathbb{P}[Z_i(t) = 1] = 1 - \mathbb{P}[Z_i(t) = 0] = \frac{\lambda_i(t)}{\lambda_i(t)}.$$
for any \( t \geq 0 \). Define \( W \) as follows: \( W(t) = w \) if and only if

\[
(\exists k : 1 \leq j \\
: (\exists y_1, \ldots, y_{k-1} : (\forall i : 1 \leq i \leq k : y_{i-1} \leq y_i) \\
: (\forall i : 1 \leq i < k \\
\quad : X_i(t + y_{i-1}) = y_i - y_{i-1} \land Z_i(t + y_i) = 0) \\
\land X_k(t + y_{k-1}) = w - y_{k-1} \\
) \land Z_k(t + w) = 1 
),
\]

where \( y_0 = 0 \). If there are \( l \geq 0 \) and \( \Lambda \) such that \( \lambda_i = \Lambda \) for all \( i \geq l \), then \( W \) has a \( \text{tdExp}(\lambda) \) distribution.

**Proof:**

Let \( w \geq 0 \). We derive

\[
f_W(t, w) \\
= \{ \text{Definition of } W \} \\
(\Sigma k : 1 \leq k \\
: \int \cdots \int dy_1 \ldots dy_{k-1} \\
0 \leq y_1 \leq \ldots \leq y_{k-1} \leq w \\
\quad \cdot (\Pi i : 1 \leq i < k : f_{X_i}(t + y_{i-1}, y_i - y_{i-1})) \\
\quad \cdot (\Pi i : 1 \leq i < k : \mathbb{P}[Z_i(t + y_i) = 0]) \\
\quad \cdot f_{X_k}(t + y_{k-1} - y_k, w - y_{k-1}) \mathbb{P}[Z_k(t + w) = 1] 
) \\
= \{ \text{Definitions of } Z_i \text{ and } X_i \} \\
(\Sigma k : 1 \leq k \\
: \int \cdots \int dy_1 \ldots dy_{k-1} \\
0 \leq y_1 \leq \ldots \leq y_{k-1} \leq w \\
\quad \cdot (\Pi i : 1 \leq i < k : \lambda_i(t + y_i) \exp(- \int_{t + y_i}^{t + y_{i-1}} \lambda_i(v)dv)) \\
\quad \cdot (\Pi i : 1 \leq i < k : 1 - \frac{\lambda_i(t + y_i)}{\lambda_i(t + y_{i-1})}) \\
\quad \cdot \lambda_k(t + w) \exp(- \int_{t + y_{k-1}}^{t + w} \lambda_k(v)dv) \frac{\lambda(t + w)}{\lambda_k(t + w)} 
) \\
= \{ \} \]
(Σ k : 1 ≤ k

: ∫ ... ∫

0≤y₁≤...≤yₖ₋₁≤w

dy₁...dyₖ₋₁

· (Π i : 1 ≤ i < k

: (λᵢ(t + yᵢ) − λ(t + yᵢ)) exp(−∫ₜ+ᵧᵢ λᵢ(v)dv))

· λ(t + w) exp(−∫ₜ+w λ_k(v)dv)

)

= { Definition of g, see below }

(Σ k : 1 ≤ k : g_k(t, w))

where function g_k is defined as

g_k(t, u)

= ∫ ... ∫

0≤y₁≤...≤yₖ₋₁≤u

dy₁...dyₖ₋₁

· (Π i : 1 ≤ i < k

: (λᵢ(t + yᵢ) − λ(t + yᵢ)) exp(−∫ₜ+ᵧᵢ λᵢ(v)dv))

· λ(t + u) exp(−∫ₜ+u λ_k(v)dv)

for integer k ≥ 1 and for t, u ≥ 0.

Now let k ≥ l, then

g_k(t, w)

= { Definition of g, using λᵢ = Λ for i ≥ l }

∫ ... ∫

0≤y₁≤...≤y_l₋₁≤w

dy₁...dy_l₋₁

· (Π i : 1 ≤ i < l

: (λᵢ(t + yᵢ) − λ(t + yᵢ)) exp(−∫ₜ+ᵧᵢ λᵢ(v)dv))

· ∫ ... ∫

y_l₋₁≤y_l≤...≤yₖ₋₁≤w

dy_l...dyₖ₋₁

· (Π i : l ≤ i < k

: (Λ(t + yᵢ) − λ(t + yᵢ)) exp(−∫ₜ+ᵧᵢ Λ(v)dv))

· λ(t + w) exp(−∫ₜ+w Λ(v)dv)

= { Calculus }
\[
\int \cdots \int_{y_1 \leq y_2 \leq \cdots \leq y_{l-1} \leq w} dy_1 \ldots dy_{l-1} \\
\cdot (\Pi i : 1 \leq i < l) \\
\quad : (\lambda_i(t + y_i) - \lambda(t + y_i)) \exp(-\int_{t+y_i}^{t+y} \lambda_i(v)dv)) \\
\cdot \lambda(t + w) \exp(-\int_{t+y}^{t+w} \Lambda(v)dv) \\
\cdot \int \cdots \int_{y_1 \leq y_2 \leq \cdots \leq y_{k-1} \leq w} dy_1 \ldots dy_{k-1} (\Pi i : 1 \leq i < k : \Lambda(t + y_i) - \lambda(t + y_i)) \\
= \{ \text{Calculus} \} \\
\int \cdots \int_{y_1 \leq y_2 \leq \cdots \leq y_{l-1} \leq w} dy_1 \ldots dy_{l-1} \\
\cdot (\Pi i : 1 \leq i < l) \\
\quad : (\lambda_i(t + y_i) - \lambda(t + y_i)) \exp(-\int_{t+y_i}^{t+y} \lambda_i(v)dv)) \\
\cdot \lambda(t + w) \exp(-\int_{t+y}^{t+w} \Lambda(v)dv) \\
\cdot \frac{1}{(k-l)!} \left( \int_{y_{l-1}}^{w} dy(\Lambda(t + y) - \lambda(t + y)) \right)^{k-l}.
\]

To summarize, we have established

\[
f_w(t, w) = (\Sigma k : 1 \leq k : g_k(t, w)) \quad \text{(A.4)}
\]

and, for \( k \geq l \),

\[
g_k(t, w) = \int \cdots \int_{y_1 \leq y_2 \leq \cdots \leq y_{l-1} \leq w} dy_1 \ldots dy_{l-1} \\
\cdot (\Pi i : 1 \leq i < l) \\
\quad : (\lambda_i(t + y_i) - \lambda(t + y_i)) \exp(-\int_{t+y_i}^{t+y} \lambda_i(v)dv)) \quad \text{(A.5)} \\
\cdot \lambda(t + w) \exp(-\int_{t+y}^{t+w} \Lambda(v)dv) \\
\cdot \frac{1}{(k-l)!} \left( \int_{y_{l-1}}^{w} dy(\Lambda(t + y) - \lambda(t + y)) \right)^{k-l}.
\]

We continue by calculating the sum of the \( g_k \) for \( k \geq l \):

\[
(\Sigma k : l \leq k : g_k(t, w)) \\
= \{ \text{(A.5)} \}
\]
\[
\left( \sum_{k : l \leq k} \right) \int \cdots \int_{0 \leq y_1 \leq \cdots \leq y_{l-1} \leq w} dy_1 \cdots dy_{l-1} \\
\cdot (\prod_{i = 1}^{l} \leq i < l) \\
\cdot (\lambda_i(t + y_i) - \lambda(t + y_i)) \exp(- \int_{t + y_{l-1}}^{t + y_i} \lambda_i(v)dv) \\
\cdot \lambda(t + u) \exp(- \int_{t + y_{l-1}}^{t + w} \Lambda(v)dv) \\
\cdot \frac{1}{(k-l)!} \left( \int_{y_{l-1}}^{w} dy(\Lambda(t + y) - \lambda(t + y)) \right)^{k-l}
\]

\[
= \{ \text{Distribution of } \cdot \text{ over } + \} \\
\int \cdots \int_{0 \leq y_1 \leq \cdots \leq y_{l-1} \leq w} dy_1 \cdots dy_{l-1} \\
\cdot (\prod_{i = 1}^{l} \leq i < l) \\
\cdot (\lambda_i(t + y_i) - \lambda(t + y_i)) \exp(- \int_{t + y_{l-1}}^{t + y_i} \lambda_i(v)dv) \\
\cdot \lambda(t + u) \exp(- \int_{t + y_{l-1}}^{t + w} \Lambda(v)dv) \\
\cdot \left( \sum_{k : l \leq k} \frac{1}{(k-l)!} \left( \int_{y_{l-1}}^{w} dy(\Lambda(t + y) - \lambda(t + y)) \right)^{k-l} \right)
\]

\[
= \{ \text{Dummy change } k := k + l; \text{ Taylor series of } \exp \} \\
\int \cdots \int_{0 \leq y_1 \leq \cdots \leq y_{l-1} \leq w} dy_1 \cdots dy_{l-1} \\
\cdot (\prod_{i = 1}^{l} \leq i < l) \\
\cdot (\lambda_i(t + y_i) - \lambda(t + y_i)) \exp(- \int_{t + y_{l-1}}^{t + y_i} \lambda_i(v)dv) \\
\cdot \lambda(t + u) \exp(- \int_{t + y_{l-1}}^{t + w} \Lambda(v)dv) \exp(\int_{t + y_{l-1}}^{t + w} (\Lambda(v) - \lambda(v))dv)
\]

\[
= \{ \text{Definition of } h, \text{ see below } \} \\
h_l(t, w).
\]

For \( k \geq 1 \), function \( h_k \) is defined by
\[
h_k(t, u) = \int \cdots \int_{0 \leq y_1 \leq \cdots \leq y_{k-1} \leq u} dy_1 \cdots dy_{k-1} \\
\cdot (\prod_{i = 1}^{k} \leq i < k) \\
\cdot (\lambda_i(t + y_i) - \lambda(t + y_i)) \exp(- \int_{t + y_{k-1}}^{t + y_i} \lambda_i(v)dv) \\
\cdot \lambda(t + u) \exp(- \int_{t + y_{k-1}}^{t + u} \lambda(v)dv)
\]

for \( t, u \geq 0 \).
Substitution of the above result into Equation (A.4) yields the following expression for $f_w$:

$$f_w(t, w) = (\Sigma k : 1 \leq k < l : g_k(t, w)) + h_l(t, w). \quad (A.7)$$

We continue by calculating with $h_k(t, w)$. Consider the integrand of Equation (A.6) and assume that $k > 1$:

$$\begin{align*}
(\Pi i : 1 \leq i < k) \\
: (\lambda_i(t + y_i) - \lambda(t + y_i)) \exp(- \int_{t + y_i}^{t + y_l} \lambda_i(v)dv) \\
\cdot \lambda(t + w) \exp(- \int_{t + y_{k-1}}^{t + w} \lambda(v)dv) \\
= \quad \{ \text{Range split} \} \\
\lambda(t + w)(\Pi i : 1 \leq i < k - 1) \\
: (\lambda_i(t + y_i) - \lambda(t + y_i)) \exp(- \int_{t + y_i}^{t + y_l} \lambda_i(v)dv) \\
\cdot (\lambda_{k-1}(t + y_{k-1}) - \lambda(t + y_{k-1})) \exp(- \int_{t + y_{k-2}}^{t + y_{k-1}} \lambda_{k-1}(v)dv) \exp(- \int_{t + y_{k-1}}^{t + w} \lambda(v)dv).
\end{align*}$$

The last three factors of this expression we can integrate analytically:

$$\begin{align*}
\int_{y_{k-2}}^{w} dy_{k-1} (\lambda_{k-1}(t + y_{k-1}) - \lambda(t + y_{k-1})) \\
\cdot \exp(- \int_{t + y_{k-2}}^{t + y_{k-1}} \lambda_{k-1}(v)dv) \exp(- \int_{t + y_{k-1}}^{t + w} \lambda(v)dv) \\
= \quad \{ \text{Calculus} \} \\
\exp(- \int_{t + y_{k-2}}^{t + y_{k-1}} \lambda_{k-1}(v)dv) \exp(- \int_{t + y_{k-1}}^{t + w} \lambda(v)dv) \bigg|_{y_{k-1}=y_{k-2}}^{\substack{y_{k-1}=y_{k-2} \cr y_{k-1}=w}} \\
= \quad \{ \} \\
\exp(- \int_{t + y_{k-2}}^{t + y_{k-1}} \lambda(v)dv) - \exp(- \int_{t + y_{k-2}}^{t + y_{k-1}} \lambda_{k-1}(v)dv).
\end{align*}$$

So $h_k$ can be simplified by performing the integration over $y_{k-1}$. The result is the following recurrence relation for $h$:

$$h_k(t, w) = h_{k-1}(t, w) - g_{k-1}(t, w).$$

Unfolding $k - 2$ times we obtain

$$h_k(t, w) = h_1(t, w) - (\Sigma i : 1 \leq i < k : g_i(t, w)). \quad (A.8)$$

We can now finish the proof that $W$ has a $\text{tdExp}(\lambda)$ distribution:

$$\begin{align*}
f_w(t, w) \\
= \quad \{ (A.7) \} \\
(\Sigma k : 1 \leq k < l : g_k(t, w)) + h_l(t, w)
\end{align*}$$
(A.3) Proofs of Properties 3.1, 3.2, 4.1, and 4.2

\[
\begin{align*}
\quad &= \quad \text{[A.8 for } k = l \text{]} \\
&\quad (\sum k : 1 \leq k < l : g_k(t, w)) + h_1(t, w) = (\sum i : 1 \leq i < k : g_i(t, w)) \\
&\quad = \quad \{ \} \\
&\quad h_1(t, w) \\
&\quad = \quad \{ \text{Definition (A.6) with } k = 1; y_0 = 0 \} \\
&\quad \lambda(t + w) \exp(- \int_t^{t+w} \lambda(v)dv).
\end{align*}
\]
This appendix gives an overview of the notation used in this monograph. We start with a list of symbols, explain our notational conventions for quantified expressions and proofs, and address the naming scheme for the algorithms discussed in the main text.

### B.1 List of symbols

This section contains a list of symbols. Every entry in the list is accompanied by a short description of its meaning. Where applicable, a reference to the page containing the definition is given. Also included is an index of algorithms.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>∨</td>
<td>Logical or</td>
<td></td>
</tr>
<tr>
<td>∧</td>
<td>Logical and</td>
<td></td>
</tr>
<tr>
<td>⇒</td>
<td>Logical implication</td>
<td></td>
</tr>
<tr>
<td>∀</td>
<td>Universal quantification</td>
<td></td>
</tr>
<tr>
<td>∃</td>
<td>Existential quantification</td>
<td></td>
</tr>
<tr>
<td>Σ</td>
<td>Sum</td>
<td></td>
</tr>
<tr>
<td>Π</td>
<td>Product</td>
<td></td>
</tr>
<tr>
<td>$F_X$</td>
<td>Distribution function (of r.v. $X$); $P(X = x) = F_X(x)$</td>
<td>p. 167</td>
</tr>
<tr>
<td>$f_X$</td>
<td>Probability density (of random variable $X$)</td>
<td>p. 168</td>
</tr>
</tbody>
</table>

*continued on next page*
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbb{E}(X^k)$</td>
<td>$k^{th}$ moment (of random variable $X$)</td>
<td>p. 172</td>
</tr>
<tr>
<td>$\text{Exp}(\lambda)$</td>
<td>Exponential distribution with rate $\lambda$</td>
<td>p. 169</td>
</tr>
<tr>
<td>$\text{tdExp}(\lambda)$</td>
<td>Same, but time-dependent</td>
<td>p. 170</td>
</tr>
<tr>
<td>$\mathcal{L}$</td>
<td>Lattice representing catalytic surface</td>
<td>p. 9</td>
</tr>
<tr>
<td>$\mathcal{D}$</td>
<td>Domain of particle types</td>
<td>p. 10</td>
</tr>
<tr>
<td>$\Sigma$</td>
<td>Surface states, $\Sigma = \mathcal{L} \rightarrow \mathcal{D}$</td>
<td>p. 10</td>
</tr>
<tr>
<td>$\mathcal{T}$</td>
<td>Transformation type, $\mathcal{T} \subseteq \mathcal{L} \times \mathcal{D} \times \mathcal{D}$</td>
<td>p. 12</td>
</tr>
<tr>
<td>$p$</td>
<td>Pattern of a transformation</td>
<td>p. 12</td>
</tr>
<tr>
<td>$sc$</td>
<td>Source configuration of a transformation</td>
<td>p. 12</td>
</tr>
<tr>
<td>$tc$</td>
<td>Target configuration of a transformation</td>
<td>p. 13</td>
</tr>
<tr>
<td>$\varepsilon$</td>
<td>Enabledness condition</td>
<td>p. 13</td>
</tr>
<tr>
<td>$\mathcal{C}$</td>
<td>State transformation operator</td>
<td>p. 13</td>
</tr>
<tr>
<td>$E(\sigma)$</td>
<td>The set of reactions enabled in state $\sigma$</td>
<td>p. 13</td>
</tr>
<tr>
<td>$k$</td>
<td>Rate constant</td>
<td>p. 14, 95</td>
</tr>
<tr>
<td>$k(S)$</td>
<td>Sum of the rates of the reactions in set $S$</td>
<td>p. 57</td>
</tr>
<tr>
<td>$\mathcal{RT}$</td>
<td>Collection of reaction types of a system</td>
<td>p. 15</td>
</tr>
<tr>
<td>$RT$</td>
<td>Reaction type, includes transformation and reaction rate</td>
<td></td>
</tr>
<tr>
<td>$m_{RT}(S)$</td>
<td>Number of reactions of type $RT$ in set $S$</td>
<td>p. 57</td>
</tr>
<tr>
<td>$\delta\mathcal{RT}$</td>
<td>Collection of supertypes of a system</td>
<td>p. 69</td>
</tr>
<tr>
<td>$SRT$</td>
<td>Supertype</td>
<td>p. 69</td>
</tr>
<tr>
<td>$W(e)$</td>
<td>Waiting time for reaction $e$</td>
<td>p. 24</td>
</tr>
<tr>
<td>$W(S)$</td>
<td>Waiting time for next reaction from $S$</td>
<td>p. 25</td>
</tr>
<tr>
<td>$n$</td>
<td>The average number of reactions disabled (enabled) by a reaction</td>
<td>p. 44</td>
</tr>
<tr>
<td>$n^*$</td>
<td>The average number of reaction effects disabled (enabled) by a reaction</td>
<td>p. 71</td>
</tr>
<tr>
<td>$n_{src}$</td>
<td>The sum of the sizes of the reaction patterns</td>
<td>p. 44</td>
</tr>
<tr>
<td>$n_{mod}$</td>
<td>The average number of sites modified by a reaction</td>
<td>p. 44</td>
</tr>
</tbody>
</table>
B.2 Notational conventions

This section contains a summary of some notational conventions used, in particular those for writing quantified expressions and proofs.

When using quantified expressions, we mention the bound variables explicitly. The format we use for quantifications is \((Q \ l \ : \ de(l) \ : \ t(l))\). \(Q\) is a quantor, such as \(\forall\) or \(\Sigma\); it is required to be a commutative, associative operator with a unit element. The bound variables are represented by \(a\); \(l\) may be a single variable or a list of variables. The domain expression \(de(l)\) tells us over which values the bound variables may range. \(t(l)\), finally, is the term of the expression. There is one quantifier which we write differently, the set constructor. We write \([l \ : \ de(l) \ : \ t(l)]\) instead of \((\bigcup \ l \ : \ de(l) \ : \ {t(l)})\). The domain expression of the quantification is omitted when it is obvious from the context.

Proofs we write in a format based on (Dijkstra and Scholten, 1990). When we have to prove a proposition such as \(P \leq R\), we can do this, for instance, by proving \(P = Q\) and \(Q \leq R\). We write such a proof as follows.

\[
P
= \{ \text{Hint why } P = Q \} \\
Q \\
\leq \{ \text{Hint why } Q \leq R \} \\
R
\]

B.3 Algorithm naming

The table at the end of this section contains a list of all algorithms discussed in Chapters 3 and 4. The names of the algorithms give information about their structure. They identify what type of selection is used — U(niform) or W(eighted) — for choosing reactions (R), reaction effects (RE), reaction types (T), supertypes (ST), or locations (L).

Selections are made by picking an element from a set (sel), searching the entire lattice (search), or trying one random location on the lattice (trial). Another way to select is by scheduling occurrence times (sched) of reactions, reaction effects, et cetera. If only an approximation of the set of enabled or scheduled reactions is maintained, the algorithm name is extended with the letter a. Similarly, letters o and h are used if oversampling is used and if the selection is hierarchical, respectively. These suffixes are separated from the name of the algorithm from which they stem by a hyphen. The time-dependent versions of algorithms are named in the same way, except that their names start with ‘td’ to emphasize the fact that the rate constants of the reactions are time-dependent.
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRsel (3.1)</td>
<td>32</td>
</tr>
<tr>
<td>WRsel-o (3.2)</td>
<td>34</td>
</tr>
<tr>
<td>WRsel-a (3.3)</td>
<td>38</td>
</tr>
<tr>
<td>WRsel-h (3.4)</td>
<td>41</td>
</tr>
<tr>
<td>WRsel-ha (3.5)</td>
<td>61</td>
</tr>
<tr>
<td>WTselULtrial (3.6)</td>
<td>62</td>
</tr>
<tr>
<td>WRsel-hao (3.7)</td>
<td>65</td>
</tr>
<tr>
<td>WSTselULsel-a (3.8)</td>
<td>70</td>
</tr>
<tr>
<td>Rsched (3.9)</td>
<td>79</td>
</tr>
<tr>
<td>REsched (3.10)</td>
<td>81</td>
</tr>
<tr>
<td>Gsched (3.11)</td>
<td>82</td>
</tr>
<tr>
<td>Gsched-ao (3.12)</td>
<td>83</td>
</tr>
<tr>
<td>tdWRsel (4.1)</td>
<td>97</td>
</tr>
<tr>
<td>tdWRsel-o (4.2)</td>
<td>101</td>
</tr>
<tr>
<td>tdWRsel-a (4.3)</td>
<td>104</td>
</tr>
<tr>
<td>tdWTselULsel (4.4)</td>
<td>107</td>
</tr>
<tr>
<td>tdWTselULsel-a (4.5)</td>
<td>109</td>
</tr>
<tr>
<td>tdWSTselULsel-a (4.6)</td>
<td>110</td>
</tr>
<tr>
<td>tdRsched (4.7)</td>
<td>113</td>
</tr>
<tr>
<td>tdTschedULsel-a (4.8)</td>
<td>114</td>
</tr>
<tr>
<td>tdSTschedULsel-a (4.9)</td>
<td>116</td>
</tr>
</tbody>
</table>
Bibliography


Bibliography


Bibliography


Samenvatting

Dit proefschrift gaat over het simuleren van reacties aan oppervlakken. Het doel van zulke simulaties is bij te dragen aan het ontwikkelen van kennis over katalytische reacties.

Katalytische reacties worden op grote schaal gebruikt in de chemisch industrie. Een vergroting van de kennis van de verschijnselen die zich voordoen in katalytische processen op het niveau van elementaire deeltjes, zoals atomen en moleculen, kan leiden tot efficiëntere productieprocessen met minder ongewenste bijproducten.

Het uitgangspunt in dit proefschrift wordt gevormd door een beschrijving van een katalytisch oppervlak, van de reactiestappen die kunnen plaatshebben aan zo’n oppervlak en van de snelheden van de reactiestappen. We nemen aan dat de oppervlakken die we beschouwen regelmatig en discreet zijn. De reacties die we beschouwen zijn processen als adsorptie van atomen of moleculen aan het oppervlak, reacties van deeltjes die geadsorbeerde zijn en desorptie van deeltjes. De snelheden van reactiestappen worden gegeven door reactiesnelheidsconstanten. In dit proefschrift beschouwen we twee gevallen. In het eerste geval nemen we aan dat de reactiesnelheidsconstanten niet afhankelijk zijn van de tijd; het zijn echte constanten. In het tweede, ingewikkeldere geval laten we toe dat de reactiesnelheidsconstanten afhankelijk zijn van de tijd, bijvoorbeeld doordat de temperatuur niet constant is.

We ontwikkelen technieken om, uitgaande van zo’n beschrijving, het gedrag van systemen na te bootsen waarin zich duizenden tot miljoenen deeltjes bevinden. We richten ons daarbij op zogenaamde dynamische Monte Carlo technieken. Dat wil zeggen dat het tijdsgedrag van het bestudeerde systeem beschreven wordt met behulp van een stochastisch model dat de waarschijnlijkheden beschermt waarmee het systeem zich van een willekeurige toestand σ naar een toestand σ’ begeeft over infinitesimale tijdsintervallen. Het nabootsen van het gedrag van het systeem vergt het genereren van een rij van toestanden en tijdstippen dat reacties optreden volgens de voorgeschreven waarschijnlijkheden.

In de systemen die we in dit proefschrift bestuderen, is tijd een belangrijk aspect. Hierin verschilt ons simulatieprobleem van klassieke Monte Carlo simulaties. Bij klassieke Monte Carlo technieken worden eveneens rijen van toestanden gegenereerd volgens bepaalde
overgangswaarschijnlijkheden, maar deze rijen representeren geen evolutiepaden van het systeem in de tijd.

De overgangswaarschijnlijkheden in ons model van katalytische processen worden gegeven door kansverdelingen voor de tijd die het duurt tot reacties die mogelijk zijn, ook daadwerkelijk optreden. Met deze kansverdelingen kunnen op eenvoudige wijze twee simulatie-algoritmen worden opgeschreven. In hoofdstuk 3 worden deze twee algoritmen besproken voor het geval dat de reactiesnelheidsconstanten niet afhankelijk zijn van de tijd; in hoofdstuk 4 bespreken we het geval dat er wel een tijdsafhankelijkheid is.

Door het toepassen van drie generalisatietechnieken en door deze technieken te combineren kan een heel scala van simulatie-algoritmen worden ontwikkeld. Doordat we de algoritmen allemaal afleiden van twee basisalgoritmen door het toepassen generalisatietechnieken, verkrijgen we een systematische classificatie van algoritmen. De algoritmen die in de literatuur zijn beschreven, komen allemaal we allemaal tegen in deze classificatie en kunnen daardoor met elkaar worden vergeleken. Dit is een van de belangrijke bijdragen van het proefschrift. Een andere bijdrage van het creëren van een raamwerk waarbinnen algoritmen worden geclassificeerd, is de mogelijkheid om een prestatiemodel op te stellen. Bij besproken algoritmen worden uitdrukkingen voor het gebruikte geheugen en voor de rekentijd per gesimuleerde reactie gegeven.

De classificatie van simulatie-algoritmen bevat ook nieuwe algoritmen. Met name wordt in hoofdstuk 4 een belangrijk nieuw algoritme beschreven voor het simuleren van systemen met tijdsafhankelijke reactiesnelheidsconstanten. Dit is het eerste algoritme voor het simuleren van systemen met tijdsafhankelijke reactiesnelheidsconstanten met de eigenschap dat de hoeveelheid rekenwerk per gesimuleerde reactie onafhankelijk is van de grootte van het gesimuleerde katalysatoroppervlak. Een ander resultaat van hoofdstuk 4 is de constatering dat een aantal in de literatuur gepubliceerde algoritmen niet correct is. In het bijzonder is de tijdsevolutie niet correct.

Een relatief klein aantal simulatie-algoritmen is geïmplementeerd in een simulatieprogramma, genaamd CARLOS. Hoofdstuk 5 bevat een beschrijving van CARLOS en een vergelijking van de prestaties van de geïmplementeerde algoritmen. Uit de prestatie-analyse blijkt dat het prestatiemodel zoals gepresenteerd bij de algoritmen een zinnelijke vergelijking van algoritmen mogelijk maakt. Het prestatiemodel is echter nog niet gedetailleerd genoeg om voor een willekeurig katalytisch systeem te kunnen voorspellen welk algoritme de beste prestaties zal leveren.

Een manier om computerberekeningen te versnellen, is het verdelen van het rekenwerk over meerdere processoren. In hoofdstuk 6 wordt een analyse gegeven van de mogelijkheden voor de toepassing van parallelisme in het simuleren van oppervlakte-reacties. Een conclusie van de analyse is dat parallelle simulatie-algoritmen met correcte tijdsevolutie geen goede snelheidsverbetering zullen leveren. Om te garanderen dat de gesimuleerde tijdsevolutie correct is, moeten teveel synchronisatiestappen worden uitgevoerd om goede
prestaties te verkrijgen. Een mogelijkheid om de prestaties te verbeteren is toe te laten dat met kleine kans incorrecte tijdsevoluties worden gesimuleerd. Om te onderzoeken of deze aanpak levensvatbaar is, is een parallel simulatie-algoritme geïmplementeerd dat is toege
neden op een specifiek systeem (Ziff et al., 1986). De conclusie is dat voor dat specifieke systeem de aanpak werkt en tot goede resultaten leidt: de prestaties zijn goed en de simulatieresultaten wijken niet meetbaar af van resultaten gepubliceerd in de literatuur. Of dit ook het geval is voor andere, ingewikkeldere systemen zou een startpunt kunnen vormen voor een vervolg op dit onderzoek.
Acknowledgments

This thesis would not have been the same (or should I say ‘would not have been’?) without the support of many others. I would like to take the opportunity to express my gratitude to these people.

The first two people I want to thank are my parents. They may not have contributed directly to this thesis, but if it weren’t for their support, I would not have come this far.

Peter Hilbers I thank for having me as his first Ph.D. student and for always driving me to improve my work. Without his critical questions, the text of this thesis would have been less clear and it would have contained more loose ends (of course, any unclarities and loose ends present are my own responsibility).

Rutger van Santen is acknowledged for his enthusiasm and for the many discussions in the ‘MC club.’ Furthermore, I am grateful to him for his comments on the parts of the thesis dealing with issues from physics and chemistry.

Johan Lukkien’s influence on this thesis is more profound than even I know. He has contributed to my work almost on a daily basis, for instance, through his work on CARLOS and his detailed comments on draft versions of this monograph. Besides that, he has taught me many valuable things about science and research outside the scope of this thesis.

The other two members of the reading committee, Tonek Jansen and Emile Aarts, are also acknowledged for their comments on my work. Tonek Jansen I also thank for the stimulating discussion in the meetings of the MC club.

The other members of the MC club are also acknowledged for the fruitful discussions. Ronald Gelten I want to mention in particular. I thank him for the input he provided to help develop CARLOS and for letting me use his CO oxidation model as an example in my thesis.

Finally, I thank Esther for her love and her support. Her most tangible contribution to this thesis is found in the pages she typed when I was not able to type myself, but she means much more than that. Thank you, Esther.
Curriculum Vitae

Jun. 20, 1969  Born in Helmond, The Netherlands
Sep. 1987 – Aug. 1993  University: Computing Science at Eindhoven University of Technology (M.Sc., with honors)
Sep. 1993 – Jan. 1994  Research assistant at University of Waterloo, Ontario, Canada
Nov. 1998 – System Designer in Forward Looking Work group at Lucent Technologies

Current address:
Lucent Technologies
Forward Looking Work
P.O. Box 18
1270 AA Huizen
Email: jsegers@acm.org
Titles in the IPA Dissertation Series

The State Operator in Process Algebra
J. O. Blanco
Faculty of Mathematics and Computing Science, TUE, 1996-1

Transformational Development of Data-Parallel Algorithms
A. M. Geerling
Faculty of Mathematics and Computer Science, KUN, 1996-2

Interactive Functional Programs: Models, Methods, and Implementation
P. M. Achten
Faculty of Mathematics and Computer Science, KUN, 1996-3

Parallel Local Search
M. G. A. Verhoeven
Faculty of Mathematics and Computing Science, TUE, 1996-4

The Implementation of Functional Languages on Parallel Machines with Distrib. Memory
M. H. G. K. Kesseler
Faculty of Mathematics and Computer Science, KUN, 1996-5

Distributed Algorithms for Hard Real-Time Systems
D. Alstein
Faculty of Mathematics and Computing Science, TUE, 1996-6

Communication, Synchronization, and Fault-Tolerance
J. H. Hoepman
Faculty of Mathematics and Computer Science, UvA, 1996-7
Reductivity Arguments and Program Construction
H. Doornbos
Faculty of Mathematics and Computing Science, TUE, 1996-8

Functorial Operational Semantics and its Denotational Dual
D. Turi
Faculty of Mathematics and Computer Science, VUA, 1996-9

Single-Rail Handshake Circuits
A. M. G. Peeters
Faculty of Mathematics and Computing Science, TUE, 1996-10

A Systems Engineering Specification Formalism
N. W. A. Arends
Faculty of Mechanical Engineering, TUE, 1996-11

Normalisation in Lambda Calculus and its Relation to Type Inference
P. Severi de Santiago
Faculty of Mathematics and Computing Science, TUE, 1996-12

Abstract Interpretation and Partition Refinement for Model Checking
D. R. Dams
Faculty of Mathematics and Computing Science, TUE, 1996-13

Topological Dualities in Semantics
M. M. Bonsangue
Faculty of Mathematics and Computer Science, VUA, 1996-14

Algorithms for Graphs of Small Treewidth
B. L. E. de Fluiter
Faculty of Mathematics and Computer Science, UU, 1997-01

Process-algebraic Transformations in Context
W. T. M. Kars
Faculty of Computer Science, UT, 1997-02

A Generic Theory of Data Types
P. F. Hoogendijk
Faculty of Mathematics and Computing Science, TUE, 1997-03
The Evolution of Type Theory in Logic and Mathematics
T. D. L. Laan
Faculty of Mathematics and Computing Science, TUE, 1997-04

Preservation of Termination for Explicit Substitution
C. J. Bloo
Faculty of Mathematics and Computing Science, TUE, 1997-05

Discrete-Time Process Algebra
J. J. Vereijken
Faculty of Mathematics and Computing Science, TUE, 1997-06

A Functional Approach to Syntax and Typing
F. A. M. van den Beucken
Faculty of Mathematics and Informatics, KUN, 1997-07

Ins and Outs in Refusal Testing
A.W. Heerink
Faculty of Computer Science, UT, 1998-01

A Discrete-Event Simulator for Systems Engineering
G. Naumoski and W. Alberts
Faculty of Mechanical Engineering, TUE, 1998-02

Scheduling with Communication for Multiprocessor Computation
J. Verriet
Faculty of Mathematics and Computer Science, UU, 1998-03

An Asynchronous Low-Power 80C51 Microcontroller
J. S. H. van Gageldonk
Faculty of Mathematics and Computing Science, TUE, 1998-04

In Terms of Nets: System Design with Petri Nets and Process Algebra
A. A. Basten
Faculty of Mathematics and Computing Science, TUE, 1998-05

Inductive Datatypes with Laws and Subtyping – A Relational Model
E. Voermans
Faculty of Mathematics and Computing Science, TUE, 1999-01
Towards Probabilistic Unification-based Parsing

Hugo ter Doest
Faculty of Computer Science, UT, 1999-02

Algorithms for the Simulation of Surface Processes

J.P.L. Segers
Faculty of Mathematics and Computing Science, TUE, 1999-03
Stellingen
behorende bij het proefschrift

Algorithms for the Simulation
of Surface Processes

van

John Segers

1. Het achterwege laten van het formuleren van correctheidseisen heeft geleid tot
de publicatie van een aantal incorrecte algoritmen voor dynamische Monte Carlo
simulaties van oppervlakteprocessen met tijdsafhankelijke reactiesnelheidscon-
stanten.


2. Algoritmen voor dynamische Monte Carlo simulaties van systemen met tijdson-
afhankelijke reactiesnelheidsconstanten, kunnen worden omgezet in algoritmen
voor het simuleren van systemen met tijdsafhankelijke reactiesnelheidsconstan-
ten.


zijn ze fout omdat in een niet-lege porie bestaande uit N sites de minimale be-
zetting $\frac{1}{N}$ is.

and Catalytic Reaction in Single-File Systems. J. Catalysis,

4. Het is mogelijk up-down counters te ontwerpen met constante responstijd.

Counters. Science of Computer Programming, 27:185–204,
1996.

5. Moderne parallelle computers zijn minder geschikt voor het uitvoeren van fijnkor-
relige, parallelle algoritmen dan oudere parallelle computers. Het is te verwach-
ten dat de trend naar grofkorreliger parallelisme zich voortzet.

6. Het grote aantal organisaties dat standaarden ontwikkelt voor de telecommuni-
catiesector belemmert een snelle ontwikkeling van wereldwijde standaarden en
daarmee de belangen van gebruikers.
7. Het niet vermelden van titels van artikelen in literatuurverwijzingen leidt weliswaar tot kortere bibliografieën, maar het maakt het werk minder toegankelijk.

8. Niet elke IT’er is een informaticus en niet elke informaticus is een IT’er.

9. De televisieuitzendingen van de nationale wetenschapsquiz leveren een negatieve bijdrage aan het imago van de wetenschap.

10. Het gebruik van de benaming ‘openbaar vervoer’ voor vervoersdiensten aangeboden door particuliere ondernemingen is misleidend.