Computer Physics Communications 185 (2014) 2138–2150

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

kmos: A lattice kinetic Monte Carlo framework☆

Max J. Hoffmann*, Sebastian Matera, Karsten Reuter

Chair of Theoretical Chemistry and Catalysis Research Center, Technische Universität München, Lichtenbergstr. 4, D-85747 Garching, Germany

ARTICLE INFO

Article history:
Received 3 January 2014
Received in revised form
1 April 2014
Accepted 3 April 2014
Available online 13 April 2014

Keywords:
Lattice kinetic Monte Carlo
Microkinetic modeling
First-principles multi-scale modeling
Heterogeneous catalysis
Graphical user interface
Python
Fortran90
Open source

ABSTRACT

Kinetic Monte Carlo (kMC) simulations have emerged as a key tool for microkinetic modeling in heterogeneous catalysis and other materials applications. Systems, where site-specificity of all elementary reactions allows a mapping onto a lattice of discrete active sites, can be addressed within the particularly efficient lattice kMC approach. To this end we describe the versatile kmos software package, which offers a most user-friendly implementation, execution, and evaluation of lattice kMC models of arbitrary complexity in one- to three-dimensional lattice systems, involving multiple active sites in periodic or aperiodic arrangements, as well as site-resolved pairwise and higher-order lateral interactions. Conceptually, kmos achieves a maximum runtime performance which is essentially independent of lattice size by generating code for the efficiency-determining local update of available events that is optimized for a defined kMC model. For this model definition and the control of all runtime and evaluation aspects kmos offers a high-level application programming interface. Usage proceeds interactively, via scripts, or a graphical user interface, which visualizes the model geometry, the lattice occupations and rates of selected elementary reactions, while allowing on-the-fly changes of simulation parameters. We demonstrate the performance and scaling of kmos with the application to kMC models for surface catalytic processes, where for given operation conditions (temperature and partial pressures of all reactants) central simulation outcomes are catalytic activity and selectivities, surface composition, and mechanistic insight into the occurrence of individual elementary processes in the reaction network.

Program summary

Program title: kmos

Catalogue identifier: AESU_v1_0

Program summary URL: http://cpc.cs.qub.ac.uk/summaries/AESU_v1_0.html

Program obtainable from: CPC Program Library, Queen's University, Belfast, N. Ireland

Licensing provisions: GNU General Public License, version 3

No. of lines in distributed program, including test data, etc.: 27 450

No. of bytes in distributed program, including test data, etc.: 2 777 387

Distribution format: tar.gz

Programming language: Python 16.4%, fortran90: 83.6%.

Computer: PC, Mac.

Operating system: Linux, Mac, Windows.

RAM: 100 MB+

Classification: 7.8.

External routines: ASE, Numpy, f2py, python-lxml

Nature of problem:
Microkinetic simulations of complex reaction networks with all elementary processes occurring at active sites of a static lattice.

☆ This paper and its associated computer program are available via the Computer Physics Communication homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding author.
E-mail addresses: mjhoffmann@gmail.com, max.hoffmann@ch.tum.de (M.J. Hoffmann), karsten.reuter@ch.tum.de (K. Reuter).

http://dx.doi.org/10.1016/j.cpc.2014.04.003

Solution method:
Efficient lattice kinetic Monte Carlo solution of the Markovian master equation underlying the reaction network.

Unusual features:
The framework implements a Fortran90 code generator

Running time:
From 10 s to 10 h

1. Introduction

The pressing demands for ever more energy- and resource-efficient processing reinforce the long-standing quest towards a detailed mechanistic understanding of heterogeneous catalysis. At best down to the atomic level, such understanding would pave the way for a rational design of improved catalysts, which ultimately will be tailored to the nanoscale. Quantitative theory increasingly contributes to this quest with refined kinetic models that meanwhile allow to accurately predict the activity of model catalysts of increasing complexity (from single-crystal surfaces up to nanoparticles at planar supports) without any recourse to experimental data [1–8]. Such models have to span a range of scales in length and time, starting with the making and breaking of the individual chemical bonds at the electronic structure level, over the mesoscopic interplay of the various elementary reactions in the reaction network, to the heat and mass transport at the macroscopic (reactor) scale [9–15].

To achieve this, state-of-the-art multi-scale models resort to a hierarchical combination of different methodology. The current framework for the mesoscopic level are microkinetic approaches evaluating a (Markovian) master equation (*vide infra*) [9,16,17]. Using as input kinetic parameters for all elementary reactions (e.g. provided from first-principles electronic structure theory calculations), such microkinetic models determine for given operation conditions at the surface (e.g. temperature T and partial pressures $\{p_i\}$ of all reactants i) not only the catalytic activity (typically measured as turn-over frequency, TOF, in units of products per active site and time) but also other important information such as surface composition, the occurrence of individual reaction steps in the network, or in particular the presence of a dominant reaction mechanism as well as rate-determining steps therein [18,19]. Averaged over a sufficiently large catalyst surface area the TOF output can then for example be used as input for macroscale simulations of heat and mass transport in a given reactor geometry [20–27].

The traditional and still prevalent microkinetic approach employs a mean-field approximation to solve the master equation, and then only accounts for average surface coverages of the different reaction intermediates at the active surface. In case of heterogeneous arrangement of active sites, strong lateral interactions among the adsorbed species, or diffusion limitations, this approximation is known to break down and lead to qualitatively wrong results [19,28,29]. This has contributed to the recent rise of alternative kinetic Monte Carlo (kMC) simulations, which do not need to rely on the mean-field approximation and therefore provide a faithful account of the detailed spatial distributions of species at the catalyst surface, as well as their correlations and fluctuations [30–32]. In contrast to effective rate equation based models for which a definition of some abstract active site (type) is often sufficient, kMC thus needs as input detailed information about the microscopic arrangement of the true active sites of the crystal surface. In return, it then provides comprehensive information about the (time-resolved) arrangement of chemicals at all these active sites during catalyst operation. Apart from a wealth of mechanistic information, e.g. about correlations in the occupation of neighboring sites at the surface, this allows to obtain proper (not erroneous mean-field) mesoscopic averages of quantities like TOFs that ultimately are required for reactor level modeling.

Due to the inherent methodological simplicity of kMC, seminal works in the surface catalysis context typically relied on specialized code written from scratch [1–3,33–35]. Even though kMC models are used in the field with increasing frequency this practice has largely prevailed and only few general kMC packages have been put forward to date [36–39]. This stands in stark contrast to the manifold of established and powerful software packages employed in the multi-scale framework for either the underlying electronic structure calculations [40] or the continuum mechanics reactor scale simulations [41]. Noting this as an obstacle to a further, wide-spread use of the kMC approach to surface catalysis has been the motivation for the here presented `kmos` package, which as its core objective aims at a most user-friendly and efficient implementation, execution, and evaluation of increasingly complex lattice kMC models in the surface catalysis context.

2. Theoretical background

2.1. (Surface) chemistry on a lattice

In terms of microkinetic modeling, the atomistic evolution proceeding during surface catalytic reactions is quite representative for a wider class of problems including crystal growth, initial corrosion, diffusion in crystalline (battery) materials or surface self-assembly. These problems feature a range of common characteristics, which motivate a so-called lattice approach to kMC that also underlies the `kmos` package. In the following we use a survey over these characteristics to briefly introduce this lattice approach to kMC and clearly define terminology henceforth employed. For a more detailed account of general kMC methodology we refer to existing reviews [30–32]. Even though the following introduction is done within the surface catalysis context, the generalization to the other problems mentioned is self-evident.

Site-specific adsorption and lattice mapping. The first defining characteristic is that the surface adsorption of all reactants and reaction intermediates is assumed to be site-specific, i.e. it always occurs in well-defined so-called active *sites* offered by the crystalline surface. Due to the periodicity of the latter this generally leads to a lattice with each lattice point representing one such site [42]. The actual kMC simulations only consider this lattice, which allows to encompass a wide range of system geometries within this framework. Most straightforward are extended low-index single-crystal surfaces, where the lattice is simply composed of multiple identical surface unit-cells and then continued through the use of periodic boundary conditions. Fig. 1 illustrates [43] this for a fcc(100) model catalyst surface exhibiting two types of active sites. More

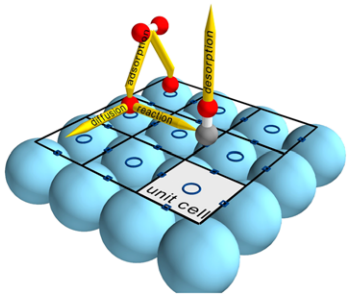

Fig. 1. (Color online) Schematic representation of a typical model catalyst surface, showing the top layer atoms of a metal(100) facet as large spheres. The periodic surface consists of repeating unit cells, each containing one or more active sites (here indicated by circles and squares for hollow and bridge adsorption sites, respectively). Possible elementary reactions in the context of CO oxidation (dissociative adsorption, diffusion, reaction and desorption) are indicated by yellow arrows.

complex geometries like entire nanoparticles are accessed through the thoughtful use of non-primitive unit-cells and/or pseudo reaction intermediates (e.g. declaring different active sites, effective species and kinetic parameters for every facet).

At each site an integer *occupation* value represents one of several possible *reaction intermediates* binding to this site (e.g. 1 for adsorbed O, 2 for adsorbed CO in the prominent CO oxidation context), including a reaction intermediate `empty` (e.g. occupation value 0) and also including the possibility that a (larger) reaction intermediate extends over more than one site (in the lattice context simply achieved by additional constraints linking the occupation of neighboring sites). One specific set of occupation values on the entire lattice is called a *configuration* (denoted by small Latin letters $u, v, \ldots$), and a transition from one configuration to another proceeds through the occurrence of an *event* (denoted by small Greek letters $\alpha, \beta, \ldots$). An event thus changes the occupation of one or more sites.

Rare-event dynamics and Markovian master equation. The second defining characteristic is that the time evolution is characterized by a so-called rare-event dynamics [44]. Due to activation barriers well exceeding thermal energies, the reaction intermediates reside most of the time in their adsorption (lattice) sites, and the events in form of the actual *elementary reactions* (adsorption, diffusion, reaction, desorption) happen comparably fast in between. Exploiting this separation of time scales, prevalent microkinetic theory [16,45] generally assumes that any such event occurs independent of all preceding ones, i.e. it applies a *Markov* approximation. The time evolution of the system (in this case the transitions from configuration to configuration through the consecutive occurrence of events) is then described by a Markovian master equation

$$\dot{\rho}_u(t) = \sum_v \left(w_{uv}\rho_v(t) - w_{vu}\rho_u(t)\right), \tag{1}$$

where $\rho_u(t)$ is the probability for the system to be in configuration u at time t, and w_{vu} is the transition rate (in units of time^{-1}) at which configuration u changes to configuration v.

Locality of elementary reactions. The third defining characteristic of the systems mentioned initially is that changes in configuration due to an event are typically geometrically narrowly confined to as few as ∼1–10 sites. Due to this locality it is possible and convenient to uniquely define any elementary reaction a in terms of the local *educt* E_a lattice configuration before and the local *product* P_a lattice configuration after the event, as well as the concomitant rate constant k_a,

$$a : E_a \xrightarrow{k_a} P_a. \tag{2}$$

Obviously, these local lattice configurations have to extend at least over all sites that actually change occupation due to the occurring elementary step. For a simple unimolecular CO adsorption step the local lattice configuration must e.g. contain the very site involved that changes its occupation from 0 (`empty`) to 2 (`CO`). For a dissociative adsorption step of O_2 the minimum local lattice configuration must in turn extend over the two neighboring sites that change their occupation from 0 (`empty`) to 1 (`O`), and for more complex reactions involving reaction intermediates covering multiple sites the minimum local lattice configurations span even larger lattice areas. In cases the local lattice configurations may need to include further nearby lattice sites, which do not change their actual occupation from educt to product configuration, but occupation value of which is a necessary information to determine the elementary reaction. This is prominently the case in the presence of lateral interactions. In order to properly capture such interactions the local lattice configuration needs to include all lattice sites within the interaction radius to uniquely define the local adsorbate environment. Imagine for the case of the aforementioned unimolecular CO adsorption that this depends on whether or not a site neighboring the actual adsorption site is also occupied with CO. In this situation the local educt and product lattice configuration need to include the actual adsorption site i (which changes its occupation) and the neighboring site j to uniquely define two distinct elementary reactions:

$$a_1 : \texttt{empty@}i;\ \texttt{empty@}j \xrightarrow{k_{a_1}} \texttt{CO@}i$$

and

$$a_2 : \texttt{empty@}i;\ \texttt{CO@}j \xrightarrow{k_{a_2}} \texttt{CO@}i.$$

In the presence of periodicity in the employed lattice there can be a large number of events that in fact all represent the same elementary reaction, just occurring at different lattice sites. The definition through the local lattice configurations allows to efficiently achieve this classification by first checking if local educt and product lattice configurations can be transformed into each other through a lateral lattice translation vector. Since an elementary reaction is not affected by any lattice configuration difference outside the local educt and product configuration this grouping correctly includes many events which only differ by the (non-changing) occupations outside these local configurations. To illustrate this consider again CO adsorption on an empty periodic surface featuring one type of active site. Given that adsorption into any of these sites is equivalent, adsorption events on sites i and j are different events in terms of the overall lattice configuration, yet they would both be grouped to the same elementary reaction by their identical local educt and product lattice configurations. Similarly, adsorption on site i with another adsorbate present on site k is again a different event, but falls still into the same elementary reaction class if there are no lateral interactions between sites i and k, and k is correspondingly outside the local lattice configuration. Notwithstanding, it is important to realize that identical local educt and product lattice configurations are only a necessary, but not a sufficient condition for the same elementary reaction. In the surface catalysis context, this is notably exemplified by Eley–Rideal type reaction events, where an adsorbed reaction intermediate is reacted off in a gas-phase scattering reaction. The local educt and product lattice configurations for such an event are identical to those describing a mere desorption process of the reaction intermediate. Yet, these are two distinct elementary reactions, which in the lattice framework is accounted for through two different rate constants.

Size and structure of the transition matrix. The considerations about locality provide important insight into the structure of the overall transition matrix $\boldsymbol{w}$ in Eq. (1). First, it can be decomposed into a

sum of elementary reaction matrices as

$$w_{vu} = \sum_a w^a_{vu}, \tag{3}$$

where

$$w^a_{vu} = \begin{cases} k_a & (u \to v) \in a \\ 0 & \text{else} \end{cases} \tag{4}$$

and k_a is the reaction rate constant of elementary reaction a. The total number of different matrix elements is thus given by the number of inequivalent elementary reaction steps in the model.

Second, with respect to the structure of $\boldsymbol{w}$ it is useful to define the set of *available events* $\boldsymbol{\sigma}_u$ for any configuration u as the set of all events α_{vu} that lead from configuration u to any other configuration v

$$\boldsymbol{\sigma}_u = \{\alpha_{vu} | w_{vu} \neq 0\}, \tag{5}$$

i.e. $\boldsymbol{\sigma}_u$ is formed by the non-zero elements in the uth column of $\boldsymbol{w}$. The locality of the elementary reactions implies that there are no events that connect largely differing lattice configurations. As such, $\boldsymbol{\sigma}_u$ is much smaller than the total size of $\boldsymbol{w}$, i.e. the transition matrix is sparse. For the later kMC efficiency discussion we further note that given an event α_{vu}, all events in $\boldsymbol{\sigma}_v \setminus \boldsymbol{\sigma}_u$ are said to be *enabled* by α_{vu}, while all events in $\boldsymbol{\sigma}_u \setminus \boldsymbol{\sigma}_v$ are said to be *disabled* by α_{vu}. As any event is local and thus affects much less sites than the total number of sites in the lattice, for every event α_{vu} the number of enabled or disabled events is again much smaller than the number of available events in both u and v, or

$$|\boldsymbol{\sigma}_u \cap \boldsymbol{\sigma}_v| \gg |(\boldsymbol{\sigma}_u \cup \boldsymbol{\sigma}_v) \setminus (\boldsymbol{\sigma}_u \cap \boldsymbol{\sigma}_v)|. \tag{6}$$

This difference in size will become the more pronounced the larger the lattice that is to be simulated.

These insights into the structure of the transition matrix are important as the formal simplicity of the master equation (1) easily disguises the complexity in solving it in practice in the surface catalytic context. This is due to the sheer size of the space of all possible lattice configurations. To illustrate this, let us assume a simple surface system that exhibits only one type of active site per unit cell and allows for a minimum number of two different reaction intermediates at this site (again in the context of CO oxidation this could be adsorbed O and CO). Together with the possibility of an active site being empty, this yields three possible occupations of every site. In order to properly capture the ensemble characteristics of the system like the average TOF we typically need to explicitly simulate at least a surface area of (10×10) surface unit-cells that is then continued by periodic boundary conditions. The total number of configurations possible on this lattice is $3^{100} \approx 10^{47}$, and the $(3^{100} \times 3^{100})$ transition matrix $\boldsymbol{w}$ features $(3^{100})^2 \approx 10^{95}$ matrix entries w_{vu}. As discussed this matrix is sparse though, as there are no events that connect largely differing lattice configurations, and $\boldsymbol{\sigma}_u$ for every configuration u will be much smaller than 3^{100}. Nevertheless, $\boldsymbol{w}$ still contains a total number of non-zero elements that is too large to be stored directly. On the other hand, due to the translational symmetry of the lattice the total number of inequivalent matrix entries w_{vu} is typically rather small and determined by the total number of inequivalent elementary reactions a in the reaction network, cf. Eq. (4). For a CO oxidation model in the described simple surface system this total number can be as low as seven: dissociative adsorption of O_2, associative desorption of two adsorbed O, CO adsorption, CO desorption, O diffusion, CO diffusion, and CO+O reaction.

2.2. Kinetic Monte Carlo

It is clearly hopeless to explicitly solve such a high-dimensional master equation directly or even just aim to store the entire probability density in a lattice representation. The idea behind kinetic Monte Carlo (kMC) simulations is instead to achieve a numerical solution by generating an ensemble of trajectories of the underlying Markov process, where each trajectory propagates the system correctly from configuration to configuration in the sense that the average over the entire ensemble of trajectories yields the probability densities $\rho_u(t)$ of Eq. (1) [30–32,46–48]. Analysis of any single (stochastic) kMC trajectory is correspondingly meaningless, unless a stationary system state (in the catalysis context: steady-state operation) allows to replace the ensemble average by a time average over one trajectory. The actual objective for a kMC computer algorithm (and in turn for a software package like `kmos`) is therefore to generate such kMC trajectories. For this, the kMC code generally only needs to store the (evolving) occupation values on the lattice, as well as the inequivalent rate constants of all elementary reactions. On-the-fly it then generates and focuses on those transition rates w_{vu} that are actually required to propagate the trajectory. More specifically `kmos` aims to optimize the execution of previously defined models. While advanced model generation schemes (like self-learning kMC) could be implemented on top of the `kmos` framework, they are currently out of the scope of general first-principles methodology [9,32] and not included in this manuscript.

Differences between kMC solvers arise in the way how the event sequence is chosen and the concomitant way how the elapsed system time is determined. For the latter, one generally exploits that waiting times for uncorrelated events are Poisson distributed [46–48]. This means that given a rate constant k for an event, the probability that n events occur in an interval Δt is

$$p_n(k, \Delta t) = (k\Delta t)^n \mathrm{e}^{-k\Delta t}/n!. \tag{7}$$

The waiting time between two events is then simply given by the case that no events occur

$$p_0(k, \Delta t) = \mathrm{e}^{-k\Delta t},$$

for which a suitably distributed random number can be directly computed from a uniformly distributed random number $r \in]0, 1]$ [49] as

$$\Delta t = \frac{-\ln(r)}{k}. \tag{8}$$

Lukkien et al. [50] proposed a unified scheme consisting of three categories that classify existing kMC solvers: The first reaction method (FRM), the variable step-size method (VSSM), and the random selection method (RSM). We now briefly describe the essential features of each category, primarily to contrast the conceptual differences. The equivalence of all three approaches has also been shown by Lukkien et al. [50], such that a preference for one or the other emerges only out of efficiency considerations as discussed in the next section.

FRM. At every kMC step the FRM updates the sequence of available events $\boldsymbol{\sigma}_u$ and their corresponding rate constants $\boldsymbol{k}_u$. From this it calculates a sequence of time increments $\boldsymbol{\tau} = -\ln(\boldsymbol{r})/\boldsymbol{k}_u$, where $\boldsymbol{r} \in]0, 1]$ is a sequence of uniformly distributed random numbers. The smallest element of $\boldsymbol{\tau}$ is selected, the elapsed time is advanced by the corresponding time increment, and the corresponding event is executed by updating the system configuration accordingly.

VSSM. At every kMC step the VSSM updates the sequence of available events $\boldsymbol{\sigma}_u$ and calculates the total rate $k_{\mathrm{tot},u} = \sum_{v \in \sigma_u} k_{vu}$. The time is advanced by $-\ln(r)/k_{\mathrm{tot},u}$, where $r \in]0, 1]$ is a uniformly distributed random number, and one of the available events is selected for execution with a probability weighted by its rate constant. Since VSSM is the algorithm underlying `kmos`, Fig. 2 further illustrates these steps in form of a flow chart.

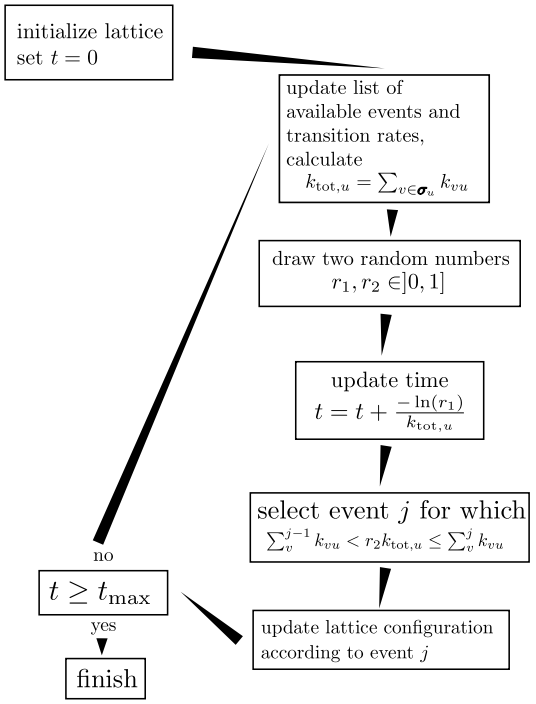

Fig. 2. The basic steps of a VSSM kinetic Monte Carlo algorithm.

RSM. At every kMC step the RSM calculates $K = \sum_a k_a$, where the sum runs over all elementary reactions a. One elementary reaction is chosen with a probability weighted by its rate constant, and one of the N_{sites} sites in the lattice is randomly selected. The time is increased by $-\ln(r)/N_{\mathrm{sites}}K$, where $r \in]0, 1]$ is a uniformly distributed random number. The lattice is updated if the event is available at the selected site.

2.3. Efficient lattice kMC

For small lattice sizes and simple kMC models containing a limited number of elementary reactions efficiency of the kMC code is generally not an issue. In particular in the context of multi-scale modeling approaches the computational costs of kMC simulations are completely negligible compared to typical costs of first-principles electronic structure theory calculations. With increasing lattice sizes and complexity of the kMC model this situation changes, in particular when considering that likely larger numbers of kMC simulations have to be run to cover different gas-phase operation conditions or when performing sensitivity analyses. In this situation, efficiency of the lattice kMC simulations becomes paramount. For the RSM algorithm a major limitation to efficiency might arise out of a diminishing probability for successful events. In the surface catalysis context this commonly arises in situations of almost fully occupied lattices and the presence of a very fast diffusion process. Such events are then predominantly chosen, but essentially never successful. For the rejection-free FRM and VSSM algorithms the main bottleneck arises instead out of the necessity to update the sequence of available events at every kMC step. A naïve approach that is straightforward to implement (and accordingly chosen in many 'from scratch' programs) is to determine $\boldsymbol{\sigma}_u$ through iteration over all lattice sites. With the number of lattice sites possibly going up to tens of thousands for entire nanoparticle simulations any retraction to such a sequential operation on the full lattice ($O(N_{\mathrm{sites}})$) will then drastically impede the overall performance.

Lukkien et al. [50] have systematically analyzed the efficiency of the three kMC approaches and concluded on VSSM as most promising method. In line with this analysis, VSSM has also been chosen as basic algorithm underlying `kmos`. In contrast to the FRM algorithm VSSM requires only two random numbers per kMC step, cf. Fig. 2. As such its main computational burden lies in the repeating update of the set of available events and total reaction rate $k_{\mathrm{tot},u}$. In contrast to the naïve $O(N_{\mathrm{sites}})$ approach, exploitation of the locality of the elementary reactions allows to largely reduce the scaling of both these calculation steps. With respect to the set of available events this is achieved through local update procedures, thereby taking into account Eq. (6). Rather than building this set anew at every kMC step, these local updates merely determine a new $\boldsymbol{\sigma}_v$ from the previous set of available events $\boldsymbol{\sigma}_u$ by strictly removing all disabled events ($\boldsymbol{\sigma}_u \backslash \boldsymbol{\sigma}_v$) and adding all enabled events ($\boldsymbol{\sigma}_v \setminus \boldsymbol{\sigma}_u$), or formally

$$\boldsymbol{\sigma}_v = (\boldsymbol{\sigma}_u \setminus (\boldsymbol{\sigma}_u \setminus \boldsymbol{\sigma}_v)) \cup (\boldsymbol{\sigma}_v \setminus \boldsymbol{\sigma}_u).$$

From the new set of available events $\boldsymbol{\sigma}_v$ its corresponding total rate constant $k_{\mathrm{tot},v} = \sum_{w\in\sigma_v} k_{wv}$ can also be calculated without iterating over the full size of the set (which would generally also scale as $O(N_{\mathrm{sites}})$). For this, not only the contained events directly, but also the number of events $N_{a,v}^{\mathrm{avail}}$ that belong to the same elementary reaction a are stored. This way, if an event α_{wv} belonging to an elementary reaction a is added to the set of available events, the corresponding counter $N_{a,v}^{\mathrm{avail}}$ is simply increased by 1, whereas if α_{wv} was removed, the counter is decreased. As a result one can quickly calculate $k_{\mathrm{tot},v}$ as

$$k_{\mathrm{tot},v} = \sum_a k_a N_{a,v}^{\mathrm{avail}}, \tag{9}$$

where the sum does not iterate over the elements in $\boldsymbol{\sigma}_v$, but only over the much smaller set of elementary reactions ($O(N_{\mathrm{react}})$). Further, if the previous summation is carried out by filling an array of accumulated rates

$$k_{1,v}^{\mathrm{acc}} = 0 \tag{10}$$

$$k_{a,v}^{\mathrm{acc}} = k_{a-1,v}^{\mathrm{acc}} + k_a N_{a,v}^{\mathrm{avail}}, \tag{11}$$

the next event α_{wv} can also be selected without retracting to $O(N_{\mathrm{sites}})$ operations by using a random number $r \in]0, 1]$ and selecting the elementary reaction b for which

$$k_{b-1,v}^{\mathrm{acc}} < r k_{\mathrm{tot},v} \leq k_{b,v}^{\mathrm{acc}}, \tag{12}$$

through a binary search ($O(\log(N_{\mathrm{react}}))$), and then selecting randomly one of the $N_{b,v}^{\mathrm{avail}}$ available events belonging to elementary reaction b ($O(1)$).

As this analysis shows every required task of a VSSM lattice kMC solver can thus be carried out with a computational effort that is independent of the number of sites in the system.

2.4. Sampling of reaction rates

A central capability of kMC simulations in the context of heterogeneous catalysis is the calculation of reaction rates. Normalized to active site or surface area, corresponding TOFs yield the occurrence of any elementary reaction per time. If this elementary reaction yields a final product, then its TOF measures the overall catalytic activity with respect to this product. If there are several elementary reactions leading to different products, then the ratios of their TOFs additionally provide the selectivities.

In the context of kMC simulations a straightforward definition of the TOF per active site of any elementary reaction a at any time t is

$$\mathrm{TOF}^a(t) = \frac{\langle N^a(t)\rangle}{N_{\mathrm{sites}}}, \tag{13}$$

where $N^a(t)$ is the number of times that reaction a has occurred at time t, and the average $\langle\,\rangle$ is over a sufficiently large ensemble

of kMC trajectories. Realizing that the actual occurrence of an event is given by the probability for the system to actually be in a configuration where the event is enabled times its rate constant, an equivalent definition is

$$\mathrm{TOF}^a(t) = \frac{\sum_u \rho_u(t) \sum_v w^a_{vu}}{N_{\mathrm{sites}}}, \tag{14}$$

where the sums run over all configurations u and v, $\rho_u(t)$ is the probability for the system to be in configuration u at time t, and w^a_{vu} are as defined in Eq. (4) the transition rates of all events α_{vu} that correspond to the elementary reaction a.

In catalytic applications the primary focus is typically on steady-state operation. Even if time-dependent operation conditions and consequently time-dependent TOFs are of interest, the changes generally occur over at least mesoscopic times, and can therefore be captured through appropriate binning in constant-condition time windows. In a corresponding stationary situation the ensemble averages in Eqs. (13) and (14) can be replaced by time averages. For the first definition this leads numerically to

$$\mathrm{TOF}^a \approx \frac{\int_0^{t_{\mathrm{final}}} N^a(t)\mathrm{d}t}{N_{\mathrm{sites}}\, t_{\mathrm{final}}} = \frac{N^a_{t_{\mathrm{final}}}}{N_{\mathrm{sites}}}, \tag{15}$$

where $N^a_{t_{\mathrm{final}}}$ is the total number of times elementary reaction a took place in a time span t_{final}. A corresponding straightforward counting to determine TOFs is what is primarily implemented in simple 'from scratch' kMC codes. This approach becomes highly inefficient though, if small TOFs are to be measured. Due to the irregular and rare occurrence of the corresponding elementary reaction long time spans need to be simulated to sufficiently converge the TOF. In this situation it is advantageous to resort to the second TOF definition in Eq. (14),

$$\begin{aligned}\mathrm{TOF}^a &= \frac{\sum_u \bar{\rho}_u \sum_v w^a_{vu}}{N_{\mathrm{sites}}} \\ &\approx \frac{\sum_{i=1}^{N_{\mathrm{final}}} \sum_v w^a_{vu_i} \Delta t_i}{N_{\mathrm{sites}}\, t_{\mathrm{final}}} \\ &= \frac{\sum_{i=1}^{N_{\mathrm{final}}} k_a N^{\mathrm{avail}}_{a,u_i} \Delta t_i}{N_{\mathrm{sites}}\, t_{\mathrm{final}}},\end{aligned}$$

where N_{final} are the number of kMC steps in the time span t_{final}, u_i is the configuration occupied at the beginning of kMC step i and Δt_i is its duration, that is the time until the simulation jumps out of u_i. The second equality demonstrates the efficiency of this approach, which adds to the convergence of the TOF with every kMC step even if k_a is very small, and which furthermore comes at negligible overhead as $N^{\mathrm{avail}}_{a,u_i}$ is calculated at every kMC step i anyway. For the determination of low TOFs this approach can therefore significantly reduce the time required for a converged sampling and is correspondingly implemented in `kmos` by default.

3. The `kmos` framework

The essential idea of the `kmos` approach to kMC modeling is to use a code generator to produce highly efficient code from an abstract definition of a kMC model. As further detailed below `kmos` thus avoids a static and in full generality cumbersome hard-coding of the complex conditional dependences between arbitrary events. Instead it custom tailors the code on the basis of a defined model, which in particular allows for most efficient local updates of enabled and disabled events. The general flow of information in the `kmos` framework is illustrated in Fig. 3. The following three subsections consecutively describe the three main parts apparent from this scheme: The specification of the kMC model, the code generation from the model, and how the generated code implements the VSSM kMC algorithm.

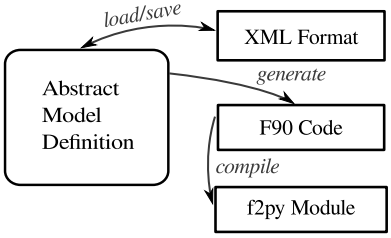

Fig. 3. Scheme for the flow of information in the `kmos` framework. An abstract kMC model is defined by using the `kmos` API. Thus, either the Python code using the `kmos` API itself can serve as definitive specification of the kMC model, or the model can be stored in an XML scheme for archiving and exchange. From the abstract model definition `kmos` generates tailored Fortran90 source code that performs the actual kMC simulations. This code can be compiled and exposed to Python again using f2py.

3.1. kMC model definition

Since this part of the `kmos` framework is Python based, this subsection will borrow a subset of object-oriented terminology to describe its structure: Essentially, a kMC model is a hierarchy of objects with attributes.

The information necessary to define a kMC model generally falls into two related, but distinct categories. On the one hand, there is the information required for the actual kMC simulations. This is information on the sites and lattice structure, on the reaction intermediates (code-internally called species), on general parameters like temperature or partial pressures that can be used to internally compute the rate constants, as well as all possible elementary reactions. On the other hand, there is additional information required for the analysis, in particular for an atomistic visualization of the generated kMC trajectories. This is prominently any explicit geometric information (size and shape of unit-cell, Cartesian coordinates of sites within the unit-cell, representation of substrate and reaction intermediates). In summary, this leads to the following schematic structure of the model definition:

- model
  - lattice (geometry): unit cell, [sites]
  - sites: name, position
  - reaction intermediates (species): name, representation
  - parameters: name, value
  - elementary reactions: name, [conditions], [actions], rate constant

Within this basic skeleton the user has to define the model specific parts. For this, one could envision some configuration file-like format. However, in particular with respect to the sequence of elementary reactions it turns out that one would have to type many very similar statements. For instance, if the same elementary reaction can be executed in several different directions due to the symmetry of the lattice. `kmos` therefore offers an application programming interface (API) that allows to create each object in the model by one constructor call (in terms of object-oriented programming). This has the benefit of offering a fairly straightforward syntax, while at the same time allowing for all the flexibility and expressiveness of a high-level programming language and its control constructs such as for-loops and if-statements.

Lattice definition. The system is represented as a finite lattice with periodic boundary conditions. At present `kmos` only supports one global Bravais lattice; a limitation that we intend to overcome in

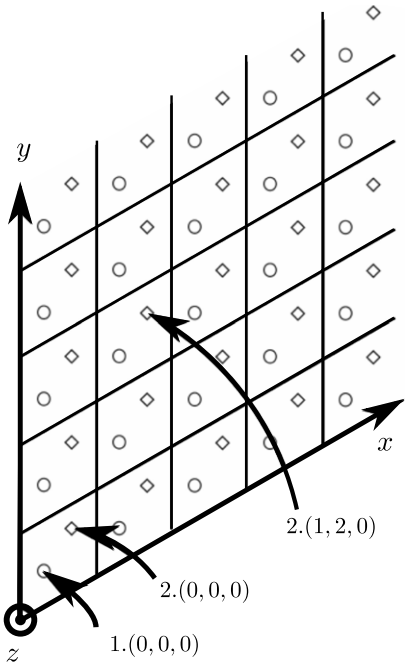

Fig. 4. Illustration of the lattice representation using the four-tuple `n.(x, y, z)`, where `x, y, z` are the integer coordinates of the unit cell, and `n` goes over the different active sites within the unit cell.

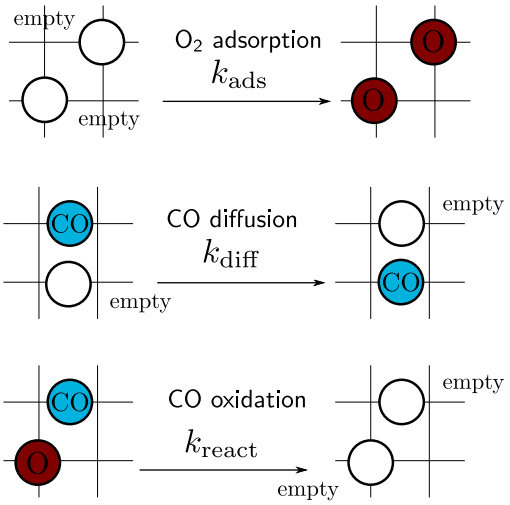

Fig. 5. Graphical representations illustrating the defining characteristics of elementary reactions: Dissociative O_2 adsorption (top panel), CO diffusion (middle panel) and CO oxidation (bottom panel)..

future work. Multiple active sites within the unit cell are accounted for through a basis. Each site is defined through a unique name. Internally every lattice point can thus be represented with a four-tuple `n.(x, y, z)`, where `x, y, z` are the integer coordinates of the unit cell, and `n` goes over the different active sites within the unit cell as illustrated in Fig. 4. Naturally this scheme can describe one-, two-, or three-dimensional lattices by setting 2, 1, or 0 entries to zero respectively. By default `kmos` enforces periodic boundary conditions by internally expanding the lattice by one unit cell along each lattice axis. When interested in modeling a finite lattice, this feature can be blocked by defining an inactive dummy reaction intermediate and initializing the edges of the simulated geometry with it. For visualization the shape and size of the unit cell can be specified, as well as the fractional Cartesian coordinates of all active sites within the unit cell.

Reaction intermediate definition. Reaction intermediates are specified through a unique name, and internally get assigned an integer value. A species `empty` needs to be explicitly defined. Site blocking, e.g. in multidentate adsorption or to mimic infinitely repulsive lateral interactions, can be achieved through the definition of dummy species (say `A_blocked` as additional dummy for a reaction intermediate `A` covering multiple sites). In the specification of the elementary reaction, the blocked sites are then occupied with the dummy, which thus prevents them from being `empty` for other elementary reactions. Special boundary conditions such as a source or a drain that continuously inserts or removes surface intermediates at the edges of the lattice can similarly be modeled by using such special intermediates and corresponding elementary reactions. For the purpose of visualization it is possible to enter a string in the atoms-object-constructor form as understood by the Atomic Simulation Environment (ASE) [51].

Elementary reaction definition. The elementary reactions are defined in terms of the occupations in the local educt and local product lattice configuration, as well as the corresponding rate constant. Some sample definitions are depicted in Fig. 5. For elementary reactions involving more than one site, other involved sites are specified by relative vectors in the four-tuple representation. If the central site used to define the elementary reaction is e.g. a `bridge` site and another `bridge` site in the unit cell in the positive direction of the first lattice vector is involved, then this additional site is referred to as `bridge.(1,0,0)`. `kmos` makes no effort to recognize symmetries in the lattice (e.g. to deduce that a diffusion from `bridge` to `bridge.(1,0,0)` implies the possible equivalent diffusion to bridge.$(-1, 0, 0)$). However, the `kmos` API allows to select pairs of sites based on type and geometrical distance making the inclusion of such equivalences straightforward. Since each element (site and occupation) of the local educt lattice configuration acts as a requirement that the elementary step can be executed, it is coined `Condition`. The definition of an elementary reaction can in principle contain an arbitrary number of such `Conditions`, though there are limits on the number that the compiler can process as discussed in the next section. Nevertheless, this allows to describe fairly complex elementary reactions involving lateral interactions, concerted processes, bystander adsorbates, multidentate adsorption, and even some reconstruction of the underlying lattice structure. The provided primitives (`Conditions` and rate-constant expressions) allow in principle for any sophisticated level of included lateral interaction in the sense of a surface cluster expansion [52,53]. Yet, the analytical dependence of the rate constant on these lateral interactions has to be explicitly provided by the user. Each element (site and occupation) of the local product lattice configuration describes a change induced by the elementary reaction and is thus coined `Action`. Internally, `Condition` and `Action` are identical data types, but for sake of clarity different class names are used.

Rate constant expressions and parameters. `kmos` accepts hard-coded values for the rate constants of the individual elementary reactions. However, in the context of surface catalysis the rate constants are often calculated using expressions such as

$$k = \frac{k_B T}{h} \exp(-\beta \Delta G),$$

for activated surface processes or

$$k = \frac{p_i A}{\sqrt{2\pi m_i k_B T}}$$

for adsorption processes of ideal-gas particles (see e.g. Ref. [54], which also includes the definition of the various parameters appearing in these expressions). Since it is convenient to quickly iterate external parameters (like temperature T and partial

pressures p_i) or directly change activation barriers ΔG for example in a sensitivity analysis study, `kmos` also allows to directly enter such mathematical expressions for the rate constants as strings, which are later evaluated at runtime for the parameters currently present. Since these evaluations are quite expensive, they are only updated if any of the parameters change though.

3.2. Code generator

As discussed in Section 2.3 the main efficiency driver of a VSSM-based kMC code is the local update procedure, with its concomitant determination of disabled and enabled events. This local update procedure is also the only heavily model-dependent part of any kMC program, whereas as detailed in the next section all other parts of the actual kMC algorithm can be written in a generic way. Complicating matters, practical kMC work typically involves frequent changes of the kMC model (refinement through addition of new elementary reaction processes, consideration of further sites and reaction intermediates etc.). These changes require modifications of the code in typically as many locations as there are elementary reactions, since each new reaction might be affected by all existing elementary reactions while it can also affect every existing elementary reaction (*vide infra*). Doing these modifications by hand (as in the early 'from scratch' codes) is therefore not only highly cumbersome, but also extremely prone to human error. The modifications concern furthermore precisely that part of the code that determines the overall efficiency and should therefore not be implemented in an unoptimized way. `kmos`' answer to this situation is to fully automatize this aspect of the work by outsourcing it to a secondary code generator program. On the basis of a defined kMC model this code generator writes the required update procedure in a compilable program language (Fortran90), which consecutively can be included in the remaining kMC program. This way, one gets the best of two worlds: A flexible high-level interface for defining kMC models and at the same time an optimized low-level implementation of the model.

The key to design an efficient code generator is to carefully reflect every logical dependence in the model to infer as many decisions as possible during the generation step and minimize the number of computational steps at runtime. This is quite different from traditional programming approaches as reflected in the following explanation, since the two levels of algorithmic description are inevitably entangled. As stated above every elementary reaction is defined in terms of `Conditions` and `Actions`, and in turn each of these is defined by a relative site coordinate and a concomitant species occupation. All `Conditions` need to be satisfied for an event representing the elementary reaction to become enabled and only one of the `Conditions` needs to be dissatisfied for an event to be disabled.

In order to implement the required updates of the set of available events $\boldsymbol{\sigma}_v$ after an event α_{vu} has been selected, some events have to be added and some have to be removed. Removing events is conceptually and computationally simpler than adding, since removing does not require any inspection of the actual lattice configuration or evaluation if *all* `Conditions` are satisfied. Instead it can be based on evaluating if any `Action` of α_{vu} dissatisfies a `Condition` of an available event in $\boldsymbol{\sigma}_u$. One therefore iterates over the `Actions` (that is configuration changes) due to α_{vu}. For each `Action` *i*, which is defined by a *species* and a *site*, one iterates over the sequence of elementary reactions. For each elementary reaction *b* the first check is if *b* contains at least one `Condition` *j* on the same *site* as in `Action` *i*. If this is the case, then such a reaction *b* could potentially have been affected by the occurrence of event α_{vu}. We correspondingly then also check if the *species* of `Action` *i* *does not* match with the *species* of `Condition` *j*. If this is also the case then an event β_{wv} corresponding to elementary reaction *b* at the lattice site where α_{vu} has occurred, has become disabled. Thus compilable code is generated which removes β_{wv} from the available events $\boldsymbol{\sigma}_v$, if it was enabled in $\boldsymbol{\sigma}_u$.

After the lattice configuration itself is updated (by generating corresponding compilable code to change the occupation entries) the newly enabled events can be added. Again one iterates over all `Actions` of α_{vu}. For each `Action` *i* again defined by a *species* and *site* one iterates over all elementary reactions. For each elementary reaction *b*, the first check is again if it contains at least one `Condition` *j* on the same *site* as `Action` *i*. If in addition the *species* of `Action` *i* *does* match with the species of `Condition` *j*, the corresponding event β_{wv} of elementary reaction *b* might have been enabled by the occurrence of event α_{vu}. Other than in the disabling procedure we now have to iterate over all other `Conditions` of β_{wv} though, which in fact involves inspection of the occupation of all sites contained in the local educt lattice configuration of β_{wv}. Only if *all* `Conditions` are satisfied, β_{wv} has indeed become enabled through the occurrence of event α_{vu}. Thus compilable code is generated which iterates over all `Conditions` of event β_{wv} at the corresponding location in the lattice and checks whether in fact the *species* of all `Conditions` of *b* match with the *species* present at the relative site. If all `Conditions` are satisfied, β_{wv} is added to the available events $\boldsymbol{\sigma}_v$.

In pseudo-code the combined algorithm developed above can be concisely written as follows. Code executing before compile-time (code generation) is set in roman type, while code executed at runtime is set in `monospaced` type (*vide infra*). Variable names are set in *italics*. The **for** statement borrows on the Python style syntax (**for** *i* **in** $x \hookrightarrow$ *block*), which instructs to execute *block* on every element of *x* and the element will be named *i* inside *block*.

```
# Update available events for
# elementary reaction a

#Disable events
for i = (species, site) in actions of a
   for b in elementary reactions involving site
      for  j in  conditions of b
         if i contradicts j
            disable β_wv if enabled

Update lattice configuration

#Enable events
for i = (species, site) in actions of a
   for b in  elementary reactions involving site
      for j in conditions of b
         if i fulfills j
            if all conditions of β_wv are met
              enable β_wv
```

A crucial feature of this general update algorithm is hereby that even though it requires four nested loops, the outcome of the `Conditions` checked in the outermost loops is uniquely determined by the given kMC model. Rather then evaluating these conditions during the actual runtime of the kMC simulation over and over again, the `kmos` code generator evaluates them beforehand and builds the outcome directly into the generated code. The parts that need to be executed at runtime consist therefore at most of two nested loops and are by construction optimized for the defined kMC model. In terms of the generated code, the most tricky part is hereby the implementation of the check, if *all* `Conditions` of a possibly enabled event are satisfied. Checking such interdependences between different `Conditions` typically involves many memory reads over all sites of the local educt lattice configuration

and thus affects the performance. So, the question arises whether there is an optimal way how to arrange the corresponding queries. The corresponding problem of constructing an optimal binary decision tree has generally been shown to be NP-complete [55]. In the kMC context the corresponding intricacy is given in loose terms by the fact that frequent local lattice configuration motifs, which would be the basis for an optimal construction algorithm, are unknown beforehand and precisely the outcome of a kMC simulation. Accordingly, only two heuristic approaches are presently implemented in `kmos` and can be selected from the command line in the code generation step (`kmos export -b < code-generator >`).

In the first approach the generated code is arranged in such a way that the average number of memory accesses are likely to be minimal for the case that every outcome is equally probable. To this end, during the code generation phase, all required read accesses are collected and sorted by decreasing frequency. All possible outcomes are grouped by the result of the most common read access and accordingly written into different conditional branches. Within each branch this process is recursively repeated. This approach shows exceptional performance at runtime for kMC models without lateral interactions and few species. Though, for more complex models involving more than three different species or considerably far-ranging lateral interactions, it often produces an exceedingly large code tree (on the order of 100 MB) and accordingly long compilation time (on the order of hours).

The second approach correspondingly aims at a moderate size of the generated code and for this assumes that the primary source for the existence of multiple `Conditions` is the existence of lateral interactions extending over many lattice sites. All elementary reactions are then automatically grouped into sets of identical `Actions`. That is each group contains elementary reactions that are identical in the sites and species that are changed in the execution, and only differ by their `Conditions` that are not changed by the elementary reaction. The rationale behind this is that models involving lateral interactions contain only a few of these sets. Within each set the present lateral interactions can be determined by as few read accesses as there are lateral interactions, since one specific lateral interaction educt excludes all others within the set. The code resulting from this approach proves to be much shorter even for models involving as much as five species and up to 40 `Conditions` (on the order of few MB), and the compilation time stays typically on the order of minutes.

3.3. Kinetic Monte Carlo solver

The generated code is combined with other generic parts to form a VSSM lattice kMC solver that follows the general flow chart shown in Fig. 2. To realize the efficiency considerations summarized in Section 2.3 this solver operates on a well designed data structure. The base of this data structure is a bijective mapping from the four-tuple `n.(x,y,z)` lattice representation to a one-dimensional representation, which simply enumerates all lattice points. This mapping can be cached in 1D and 4D arrays which makes it very efficient. Any of the frequently executed core parts are then performed on the 1D representation, and only if explicit inspection of the lattice configuration is required is the trivial inverse mapping applied. As shown below the largest arrays then have a size ($N_{\text{react}} \times N_{\text{sites}}$), where N_{react} is the total number of elementary reactions and N_{sites} is the total number of sites. Even for very large lattices such arrays do not represent any notable memory requirements. `kmos` correspondingly uses fixed array sizes and avoids dynamic data types which would require continuous memory allocation and deallocation. On this data structure the fundamental data operations to (a) determine the next event and (b) add and delete events to and from the set of available events can be executed independent of the lattice size.

Data structures.

The deployed kMC solver operates on these 6 arrays, cf. Fig. 6:

- The array $\boldsymbol{L} = \boldsymbol{L}(N_{\text{sites}})$ stores the current configuration of the system, i.e. the integer value of L_x represents the occupation at the xth site.
- The array $\boldsymbol{k} = \boldsymbol{k}(N_{\text{react}})$ stores the rate constants for all elementary processes.
- The array $\boldsymbol{N}^{\text{avail}} = \boldsymbol{N}^{\text{avail}}(N_{\text{react}})$ stores the number of available sites for all elementary reactions, cf. Eq. (9).
- The array $\boldsymbol{k}^{\text{acc}} = \boldsymbol{k}^{\text{acc}}(N_{\text{react}})$ stores the accumulated rate constants, cf. Eq. (11).
- The array $\boldsymbol{A} = \boldsymbol{A}(N_{\text{react}}, N_{\text{sites}})$ stores the available events. Each row of $\boldsymbol{A}$ represents one elementary reaction and is filled from the left, i.e. an element $A_{ai} = x > 0$ tells that site x is currently available for elementary reaction a.
- The array $\boldsymbol{I} = \boldsymbol{I}(N_{\text{react}}, N_{\text{sites}})$ allows to retrieve the available events in array $\boldsymbol{A}$. For this, if site x is currently available for elementary reaction a and the corresponding event is stored in element A_{ai}, then $I_{ax} = i$. If site x is currently not available, then $I_{ax} = 0$.

Determination of the next event. In every kMC step the solver determines the next event and therewith the concomitant elementary reaction and site as illustrated in Fig. 6. First, the array of accumulated rate constants $\boldsymbol{k}^{\text{acc}}$ is updated according to the current set of available events. This includes the calculation of the total rate constant as last element $k^{\text{acc}}(N_{\text{react}})$. The elapsed time is updated as $-\ln(r_1)/k_{\text{tot}}$, where $r_1 \in]0, 1]$. Using another uniform random number $r_2 \in]0, 1]$ and a binary search [56] an elementary reaction a is determined for which $k_a^{\text{acc}} < k_{\text{tot}} r_2 \leq k_{a+1}^{\text{acc}}$ by performing a binary search on $\boldsymbol{k}^{\text{acc}}$. Using a third uniformly distributed random number $r_3 \in]0, 1]$ the concomitant site for the selected event is determined from array $\boldsymbol{A}$ as the value of element A_{ai}, where $i = \lfloor r_3 N_a^{\text{avail}} \rfloor$.

Update of the set of available events. After having selected the event, that is elementary reaction and site, the code-generated part takes over to call the required additions and deletions to the set of available events, as well as the update of the lattice configuration. The prior two operations are straightforward but critical primitives of the generated local update code. In terms of the relevant data structures enabling site x for elementary reaction a consists of the following steps:

1. Increase number of available events:
   $N_a^{\text{avail}} := N_a^{\text{avail}} + 1$
2. Store site x: $A_{aN_a^{\text{avail}}} := x$
3. Assign address for site x: $I_{ax} := N_a^{\text{avail}}$.

Similarly, the deletion of a disabled elementary reaction a at site x proceeds as:

1. Overwrite site x with last site enabled for a: $A_{aI_{ax}} := A_{aN_a^{\text{avail}}}$
2. Empty last site $A_{aN_a^{\text{avail}}} := 0$
3. Reassign address of moved site:
   $I_{aA_{aI_{ax}}} := I_{ax}$
4. Empty address of deleted site: $I_{ax} := 0$
5. Decrease available events: $N_a^{\text{avail}} := N_a^{\text{avail}} - 1$

As one can see an *enabling* or *disabling* operation requires three or five memory transactions, respectively, and thus does not depend on the total system size or complexity. Only the search time for the next elementary reaction grows logarithmically with the number of elementary reactions N_{react} due to the binary search involved. However, this is not expected to become a bottleneck as this number is generally much smaller than the total number of events that have to be enabled or disabled.

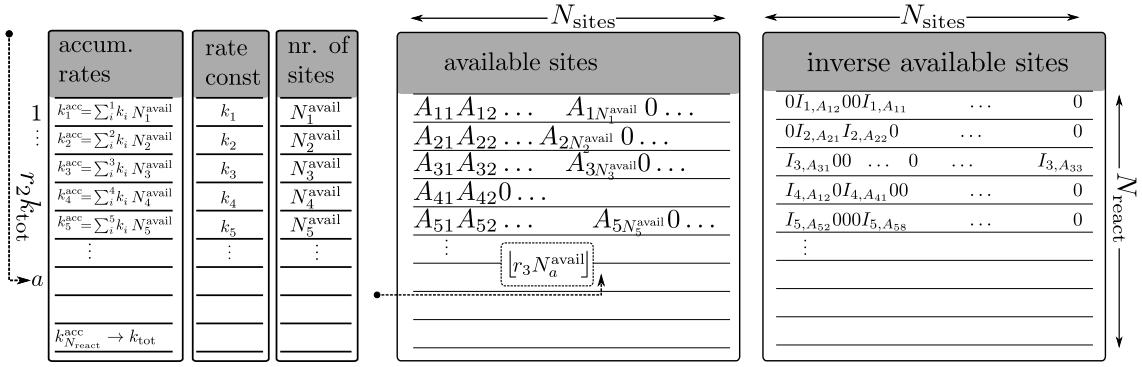

Fig. 6. Main data structures and event selection process of the `kmos` VSSM-kMC solver. The array of rate constants $\boldsymbol{k}$ is usually unchanged during a kMC simulation. $\boldsymbol{N}^{\text{avail}}$ reflects the number of sites available for each elementary reaction. Using one random number r_2 an elementary reaction a is selected using a binary search on the accumulated rate constants $\boldsymbol{k}^{\text{acc}}$. For this elementary reaction a one of the available events is selected from $\boldsymbol{A}$ using the product of a random number r_3 and the number of available sites N_a^{avail} as an index. Note that for each row a in $\boldsymbol{A}$ the first N_a^{avail} elements are non-zero after which all entries are zero. The array $\boldsymbol{I}$ stores the position under which the available events are stored in $\boldsymbol{A}$ so that all necessary updates can be executed on $\boldsymbol{A}$ without traversing it.

This concludes all required algorithmic work in one kMC step and the next step can follow.

Random numbers. As indicated above the kMC solver requires three uniformly distributed random numbers per kMC step. `kmos` relies on the pseudo random number generator (PRNG) provided by the Fortran compiler. Sometimes kMC practitioners are concerned whether such a source of randomness introduces non-physical bias to the generated kMC trajectory. We have not observed any such bias in a kMC simulation so far. In case doubt arises this can be easily tested by changing the PRNG seed conveniently in the configuration file of the compiled kMC model. Furthermore, the currently specified PRNG periods of the most commonly used compilers typically exceed the maximum number of kMC steps during one simulation by several orders of magnitude.

Overall code layout. Having specified the kMC solver independently of any lattice geometry or specifics of elementary processes means one can reuse this part of the algorithm for virtually any lattice kMC model. This is also reflected in the structure of the overall Fortran90 code: It is subdivided into the modules `base`, `lattice`, and `proclist`, of which `base` contains the model-independent parts of the VSSM loop that has been described in this Subsection. The module `lattice` replicates the `base` API in terms of lattice coordinates and implements corresponding information about the model (number of lattice dimensions, numbers of sites per unit cell, and names of sites) for visualization, as well as the central VSSM loop. The third module `proclist` is the one produced by the code-generator and implements how the set of available events is updated after an event has been selected in a kMC step, cf. Section 3.2.

3.4. Simulation front-end

The complete kMC model is stored in an XML text file by using the elementtree XML library. This also allows for easy archiving and exchange of models. A basic graphical user interface (GUI) is provided to visually inspect all aspects of the model definition including the elementary reactions. The generated Fortran90 code is compiled and exposed as a Python module with the f2py [57] interface generator. `kmos` offers a concise API which allows to control all runtime aspects of a compiled model including setup and evaluation, as a script or interactively using IPython [58] and numpy [59], as well as a GUI which visualizes the model geometry using ASE [51] and coverages and turnover frequencies using matplotlib [60], while allowing to visually change parameters during the simulation.

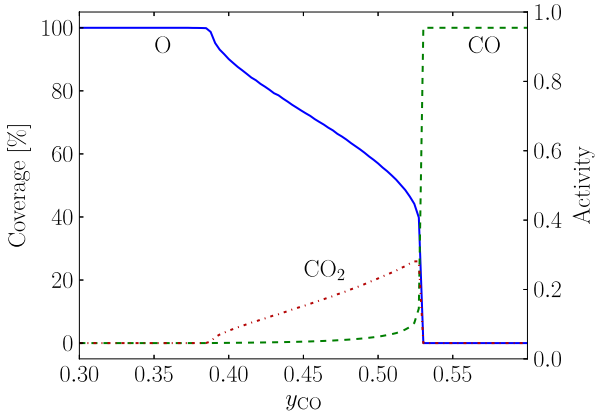

Fig. 7. Coverage dependence and catalytic activity of the ZGB model as implemented using `kmos`. In the idealized ZGB model catalytic activity is defined as the number of CO_2 molecules produced per reactant impingement [61].

4. Performance and scaling in practice

4.1. ZGB model

We demonstrate the performance and scaling behavior of `kmos` using a range of kMC models, and start with the seminal model by Ziff, Gulari, and Barshad (ZGB) [61], that has evolved into an influential reference for the development of stochastic approaches to surface catalytic processes. The original ZGB model generically considers CO oxidation at a simple cubic lattice, featuring one active site and only three elementary reactions: irreversible unimolecular adsorption of CO with rate constant y_{CO}, irreversible dissociative adsorption of O_2 at two neighboring sites with rate constant $1 - y_{\text{CO}}$, and instantaneous CO oxidation reaction of directly neighboring adsorbed CO and O. The only free parameter of the model is thus y_{CO}, which is varied in the range $[0, 1]$ a.u. In the context of numerical kMC simulations we realize this model by approximating the instantaneous CO oxidation reaction with an exceeding rate constant of 10^{15} a.u., and adding unimolecular CO and associative oxygen desorption reactions with negligible rate constants of 10^{-13} a.u. to mimic the irreversible adsorption. Especially the latter is necessary to prevent the system from getting trapped in completely oxygen or CO poisoned configurations, but we validated that neither the obtained results nor runtime performance depends on the particular choice of the finite rate constants chosen for these processes. Fig. 7 shows the resulting lattice occupations and CO_2

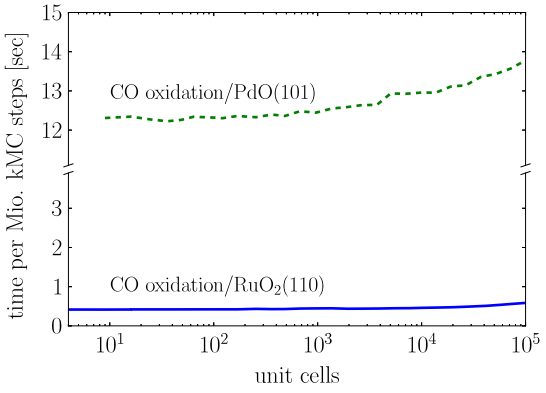

Fig. 8. Single-core CPU times required to execute one million kMC steps for the CO oxidation at $RuO_2(110)$ model (solid line) and the CO oxidation at the PdO(101) film model (dashed line) as a function of the simulated lattice size (in numbers of unit cells). For both models the `kmos` performance is essentially independent of the lattice size in the size range relevant for catalytic applications. The simulation time is instead primarily determined by the model complexity. The benchmarks were carried out on a 3.4 GHz Intel Core i7 processor with 16 GB RAM.

TOF in the relevant range of y_{CO}, perfectly reproducing the two critical y_{CO} values of $y_1 = 0.389$ and $y_2 = 0.527$ that delimit the O and CO coexistence at the surface and the concomitant catalytic activity [61]. The simulations were performed on a lattice containing (200 × 200) sites, and for this benchmark system `kmos` executed 2.15 million kMC steps per second on a 3.4 GHz Intel Core i7 processor with 16 GB RAM. Given that the simulated elapsed time per kMC step varies with every configuration, the corresponding CPU time per million kMC steps (0.47 s) is the only transferable benchmark property across implementations and somewhat even across models of similar complexity (*vide infra*).

4.2. Literature first-principles kMC models

As representative examples for modern first-principles based kMC models we consider the CO oxidation model at $RuO_2(110)$ as put forward by Reuter and Scheffler [3,33] and the CO oxidation model at a thin PdO(101) film on top of Pd(100) as put forward by Rogal, Reuter and Scheffler [34,35]. The prior model does not include lateral interactions, while the latter model does include pairwise nearest-neighbor lateral interactions at an otherwise comparable number of inequivalent elementary reactions. The comparison of the two models therefore provides first insight into the performance dependence of `kmos` on the number of `Conditions`. Specifically, the CO oxidation at $RuO_2(110)$ model includes two different active sites per surface unit cell, and a total of 26 inequivalent elementary processes (unimolecular CO adsorption and desorption, dissociative adsorption and associative desorption of O_2, CO and O diffusion, as well as CO oxidation and CO_2 decomposition) [3,33]. The PdO(101) model includes the same types of elementary reactions and also two different active sites per unit cell. In addition, it accounts for nearest-neighbor lateral interactions that modify the rate constants of all diffusion, desorption and reaction steps.

Fig. 8 shows the CPU time required to execute 1 million kMC steps for both models, again calculated on the 3.4 GHz Intel Core i7 with 16 GB RAM benchmark system. Summarized is the scaling up to a maximum system size comprising 10^5 lattice sites, which is already much larger than the 10^2–10^3 lattice sites on which these models were reliably evaluated in the original publications. In both cases the runtime is practically independent of the lattice size, confirming the scaling considerations made in Section 2.3. The moderate increase is presumably due to a less efficient utilization of the processor cache. Memory limitations eventually also determine the maximum system sizes that `kmos` can currently handle (outside the size range shown). The runtime is instead critically determined by the system complexity, and in particular by the number of `Conditions` implied by the model. Even though the $RuO_2(110)$ model contains a larger number of elementary reactions than the ZGB model, the CPU time per million kMC steps is thus almost the same (0.5 s). In contrast, the pairwise lateral interactions in the PdO(101) model and the concomitant number of `Conditions` increase this CPU time by a factor of ∼25.

4.3. Random models

To further investigate the performance dependence on the model complexity we finally consider random models with varying number of active sites per unit cell, number of possible reaction intermediates (species), number of `Conditions` per elementary reaction, and number of elementary reactions using the moderate-code-size generator. That is, first N_{sites} sites are initialized (`site1`, `site2`, …). Next, N_{species} are initialized (`species1`, `species2`,..). Using these ingredients we construct N_{react} times a pair of elementary reactions: A forward reaction which consists of $N_{\texttt{Condition}}$ `Conditions` with the default species `empty` on $N_{\texttt{Condition}}$s random sites within a finite cut-off radius and corresponding `Actions` on these same sites with random species. The corresponding backward reaction uses the `Actions` of the forward reaction as `Conditions` and uses `empty` on these same sites as `Conditions`. By creating all elementary reactions in such pairs we automatically prevent dead-lock configurations in which no events are available. All elementary reactions have the same constant rate constant, and in all cases, the simulated lattice size was (20 × 20) unit cells, as the preceding sections have shown that the performance scaling with model complexity is independent of the system size.

Using each combination of $N_{\text{sites}} \in [1, 5, 10]$, $N_{\text{species}} \in [2, 5, 10]$, $N_{\texttt{Condition}} \in [1, 2, \ldots, 10]$, and $N_{\text{react}} \in [1, 5, 10]$, we evaluate the single CPU time to execute 1 million kMC steps as in the preceding subsections. Fig. 9 compiles the obtained results, i.e. the dependence on each model dimension. To further analyze the obtained dependences the obtained runtimes are fitted to

$$t \propto (N_{\text{sites}})^a \times (N_{\text{species}})^b \times (N_{\text{react}})^c \times (N_{\texttt{Condition}})^d,$$

yielding $a \approx -0.99$, $b \approx -0.07$, $c \approx 1.24$, and $d \approx 2.00$. This shows empirically that the runtime depends approximately quadratically on the number of `Conditions` per elementary step. Furthermore it demonstrates that the runtime is basically independent of N_{species} and slightly above linear with N_{react}, confirming the observations made above with the first-principles kMC models. Last, it reveals the seemingly paradoxical result that the runtime decreases with N_{sites}. This can be rationalized by the fact that for a fixed number of elementary reactions N_{react} the probability that different events enable or disable each other shrink with increasing content in the unit-cell. This leads on average to fewer add or delete operations to the set of available events and concomitantly to decreasing runtimes. We stress though that this dependence is of little relevance for physically motivated kMC models, since there the number of elementary reactions N_{react} is expected to grow at least linearly with the number of different active sites N_{sites}. In practice, kMC models will also exhibit a different number of `Conditions` for each elementary reaction. As such, the benchmark results obtained for the random models should not be taken too literally. Nevertheless, they should convey a useful rough orientation for the to-be-expected runtimes of real kMC models featuring corresponding numbers of sites, species, and elementary reactions, as well as average number of `Conditions` per reaction.

Most centrally, the results obtained with the random models underscore that the number of `Conditions` is the most critical property in terms of runtime. This is not critical for model

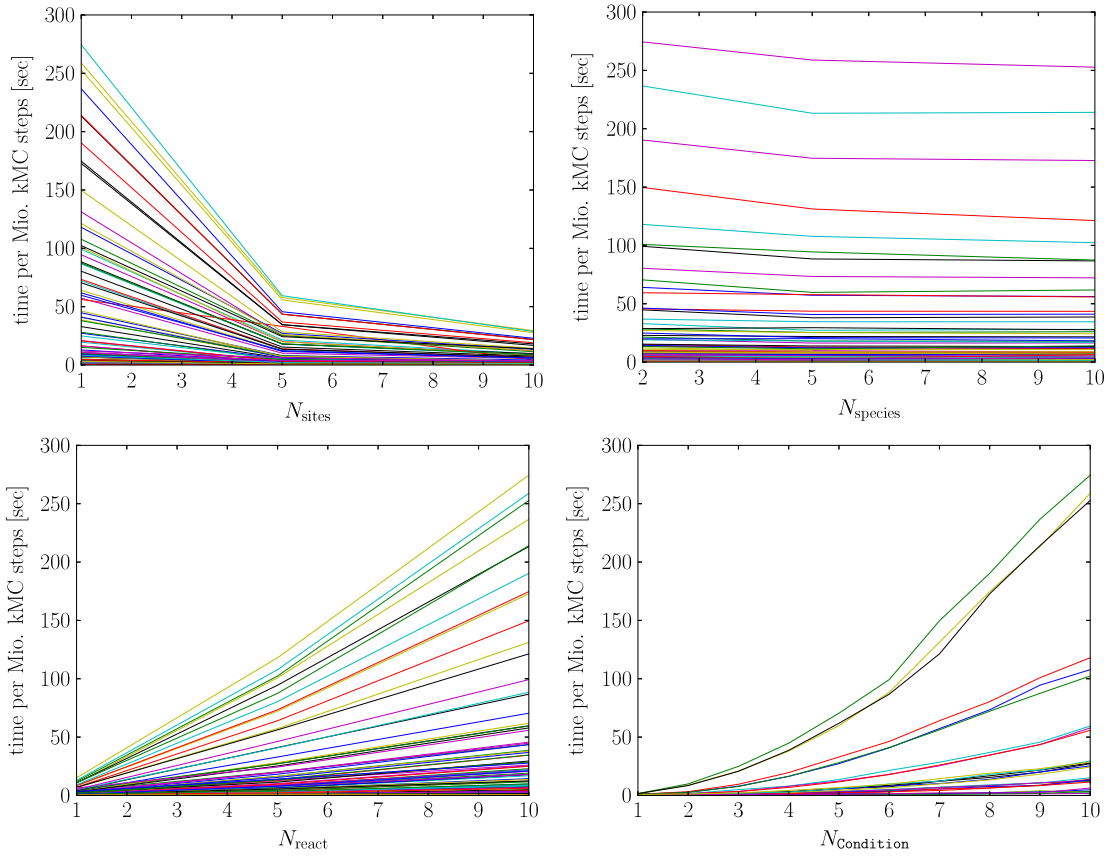

Fig. 9. Single-core CPU times required to execute one million kMC steps for various random models on the standard 3.4 GHz Intel Core i7 benchmark processor with 16 GB RAM. Each panel shows the dependence along one model parameter (N_{sites}, N_{species}, $N_{\texttt{Condition}}$ and N_{react}). Continuous lines connect simulation results obtained for random models in which all other parameters are identical, i.e. in the N_{react} panel each line represents the runtime dependence on N_{react} for a constant set of N_{sites}, N_{species} and $N_{\texttt{Condition}}$.

complexities currently addressed, in particular in the context of first-principles kMC simulations of surface catalysis. Notwithstanding, if eventually more than 5–6 reaction intermediates over multiple active sites and with extensive lateral interactions need to be handled, this will change—and the current moderate_code_size code generating algorithm might also reach the capabilities of current compilers. Long-term systematic improvements of `kmos` and its efficiency are therefore best spent on this aspect and in particular the binary decision tree to group the queries checking on the interdependences between different `Conditions`.

5. Summary

We have presented the open source [62] package `kmos`, which offers a versatile software framework for efficient lattice kMC simulations, in particular in surface catalysis. `kmos` can handle site-specific reaction networks of arbitrary complexity in one- to three-dimensional lattice systems, involving multiple active sites in periodic or aperiodic arrangements, as well as site-resolved pairwise and higher-order lateral interactions. For the kMC model definition `kmos` offers an extended application programming interface. On the basis of this model definition, a code generator creates an optimized low-level implementation of the main efficiency driver of a VSSM-based kMC code, the local update procedure that determines the disabled and enabled events after the execution of each kMC step. Together with a well designed data structure, this leads to an efficient kMC solver the runtime performance of which is essentially independent of the lattice size. Instead, the runtime sensitively depends on the model complexity and there in particular on the number of `Conditions` implied by the elementary reactions. For the complexity of reaction networks currently perceivable in the surface catalytic context this is not critical. Should higher efficiency eventually be required, improvements to this end and the code generation algorithm either through improved binary decision trees or parallelization strategies could become of interest.

Next to the efficiency, `kmos` other core objective is a most user-friendly implementation, execution, and evaluation of lattice kMC simulations. For this the API allows to control all runtime aspects interactively, through scripts or via a basic graphical user interface. Enhancing the reproducibility and reusability of the kMC models through a standard file format, `kmos` is thus hoped to contribute to a further, wide-spread use of the kMC approach by an extending user community.

Acknowledgments

MJH would like to thank Jörg Meyer and Sergey Levchenko for stimulating discussions. We gratefully acknowledge support from the German Research Council (DFG Grant RE 1509/11-1).

References

[1] E.W. Hansen, M. Neurock, Surf. Sci. 464 (2000) 91–107. URL: http://www.sciencedirect.com/science/article/pii/S0039602800005987, http://dx.doi.org/10.1016/S0039-6028(00)00598-7.
[2] E.W. Hansen, M. Neurock, J. Catalysis 196 (2000) 241–252. URL: http://www.sciencedirect.com/science/article/pii/S0021951700930185, http://dx.doi.org/10.1006/jcat.2000.3018.
[3] K. Reuter, D. Frenkel, M. Scheffler, Phys. Rev. Lett. 93 (2004) 116105. URL: http://link.aps.org/doi/10.1103/PhysRevLett.93.116105, http://dx.doi.org/10.1103/PhysRevLett.93.116105.

[4] K. Honkala, A. Hellman, I.N. Remediakis, A. Logadottir, A. Carlsson, S. Dahl, C.H. Christensen, J.K. Nørskov, Science 307 (2005) 555–558. URL: http://www.sciencemag.org/content/307/5709/555, http://dx.doi.org/10.1126/science.1106435, PMID: 15681379.

[5] M. Saeys, M.-F. Reyniers, M. Neurock, G.B. Marin, J. Phys. Chem. B 109 (2005) 2064–2073. URL: http://dx.doi.org/10.1021/jp049421j.

[6] O.R. Inderwildi, S.J. Jenkins, D.A. King, J. Phys. Chem. C 112 (2008) 1305–1307. URL: http://dx.doi.org/10.1021/jp710674q.

[7] A.A. Gokhale, J.A. Dumesic, M. Mavrikakis, J. Amer. Chem. Soc. 130 (2008) 1402–1414. URL: http://dx.doi.org/10.1021/ja0768237.

[8] R.A. van Santen, A.J. Markvoort, I. A.W. Filot, M.M. Ghouri, E. J.M. Hensen, Phys. Chem. Chem. Phys. 15 (2013) 17038. URL: http://xlink.rsc.org/?DOI=c3cp52506f, http://dx.doi.org/10.1039/c3cp52506f.

[9] M.K. Sabbe, M.-F. Reyniers, K. Reuter, Catal. Sci. Technol. 2 (2012) 2010. URL: http://pubs.rsc.org/en/content/articlehtml/2012/cy/c2cy20261a, http://dx.doi.org/10.1039/c2cy20261a.

[10] M. Salciccioli, M. Stamatakis, S. Caratzoulas, D. Vlachos, Chem. Eng. Sci. 66 (2011) 4319–4355. URL: http://www.sciencedirect.com/science/article/pii/S000925091100368X, http://dx.doi.org/10.1016/j.ces.2011.05.050.

[11] J.K. Nørskov, F. Abild-Pedersen, F. Studt, T. Bligaard, Proc. Natl. Acad. Sci. 108 (2011) 937–943. URL: http://www.pnas.org/content/108/3/937, http://dx.doi.org/10.1073/pnas.1006652108, PMID: 21220337.

[12] K. Honkala, Z. odziana, I.N. Remediakis, N. Lopez, Top. Catalysis (2014) 1–11. URL: http://link.springer.com/article/10.1007/s11244-013-0158-3, http://dx.doi.org/10.1007/s11244-013-0158-3.

[13] M. Neurock, Ind. Eng. Chem. Res. 49 (2010) 10183–10199. URL: http://dx.doi.org/10.1021/ie101300c, http://dx.doi.org/10.1021/ie101300c.

[14] P. Sautet, F. Delbecq, Chem. Rev. 110 (2010) 1788–1806. PMID: 19873968 http://dx.doi.org/10.1021/cr900295b.

[15] F.J. Keil, in: B. Kirchner, J. Vrabec (Eds.), Multiscale Molecular Methods in Applied Chemistry, in: Topics in Current Chemistry, vol. 307, Springer, Berlin, Heidelberg, 2012, pp. 69–107. URL: http://link.springer.com/chapter/10.1007/128_2011_128.

[16] C. Gardiner, Handbook of Stochastic Methods: for Physics, Chemistry and the Natural Sciences, third ed., Springer, 2004.

[17] I. Chorkendorff, J.W. Niemantsverdriet, Concepts of Modern Catalysis and Kinetics, John Wiley & Sons, 2006.

[18] H. Meskine, S. Matera, M. Scheffler, K. Reuter, H. Metiu, Surf. Sci. 603 (2009) 1724–1730. URL: http://www.sciencedirect.com/science/article/B6TVX-4VCH6T7-M/2/13208a32b93e687c21ce6aa40dac7ac7, http://dx.doi.org/10.1016/j.susc.2008.08.036.

[19] B. Temel, H. Meskine, K. Reuter, M. Scheffler, H. Metiu, J. Chem. Phys. 126 (2007) 204711. URL: http://www.ncbi.nlm.nih.gov/pubmed/17552793, http://dx.doi.org/10.1063/1.2741556, PMID: 17552793.

[20] D.G. Vlachos, AIChE J. 43 (1997) 3031–3041. URL: http://onlinelibrary.wiley.com/doi/10.1002/aic.690431115/abstract, http://dx.doi.org/10.1002/aic.690431115.

[21] R. Kissel-Osterrieder, F. Behrendt, J. Warnatz, Proc. Combust. Inst. 28 (2000) 1323–1330. URL: http://www.sciencedirect.com/science/article/pii/S008207840080346X, http://dx.doi.org/10.1016/S0082-0784(00)80346-X.

[22] D. Majumder, L.J. Broadbelt, AIChE J. 52 (2006) 4214–4228. URL: http://onlinelibrary.wiley.com/doi/10.1002/aic.11030/abstract, http://dx.doi.org/10.1002/aic.11030.

[23] S. Matera, K. Reuter, Catal. Lett. 133 (2009) 156–159. URL: http://www.springerlink.com/content/2527g41g2357742j/, http://dx.doi.org/10.1007/s10562-009-0168-8.

[24] S. Matera, K. Reuter, Phys. Rev. B 82 (2010) 085446. URL: http://link.aps.org/doi/10.1103/PhysRevB.82.085446, http://dx.doi.org/10.1103/PhysRevB.82.085446.

[25] D. Mei, G. Lin, Catal. Today 165 (2011) 56–63. URL: http://www.sciencedirect.com/science/article/pii/S0920586110007820, http://dx.doi.org/10.1016/j.cattod.2010.11.041.

[26] S. Matera, K. Reuter, J. Catalysis 295 (2012) 261–268. URL: http://www.sciencedirect.com/science/article/pii/S0021951712002771, http://dx.doi.org/10.1016/j.jcat.2012.08.020.

[27] C. Schaefer, A. P.J. Jansen, J. Chem. Phys. 138 (2013) 054102–054102–9. URL: http://jcp.aip.org/resource/1/jcpsa6/v138/i5/p054102_s1, http://dx.doi.org/doi:10.1063/1.4789419.

[28] S. Matera, H. Meskine, K. Reuter, J. Chem. Phys. 134 (2011) 064713. URL: http://link.aip.org/link/JCPSA6/v134/i6/p064713/s1&Agg=doi, http://dx.doi.org/10.1063/1.3553258.

[29] C. Wu, D. Schmidt, C. Wolverton, W. Schneider, J. Catalysis 286 (2012) 88–94. URL: http://www.sciencedirect.com/science/article/pii/S0021951711003551, http://dx.doi.org/10.1016/j.jcat.2011.10.020.

[30] A. Chatterjee, D.G. Vlachos, J. Comput.-Aided Mater. Des. 14 (2007) 253–308. URL: http://apps.isiknowledge.com/full_record.do?product=UA&search_mode=Refine&qid=8&SID=V2Dj8EMnbkBelc6NJG3&page=1&doc=10&colname=WOS, http://dx.doi.org/10.1007/s10820-006-9042-9.

[31] A.F. Voter, Radiation Effects in Solids, 2007, pp. 1–23. URL: http://dx.doi.org/10.1007/978-1-4020-5295-8_1.

[32] K. Reuter, in: Deutschmann, Olaf (Ed.), Wiley-VCH, Weinheim, 2012.

[33] K. Reuter, M. Scheffler, Phys. Rev. B 73 (2006) 045433. URL: http://link.aps.org/doi/10.1103/PhysRevB.73.045433, http://dx.doi.org/10.1103/PhysRevB.73.045433.

[34] J. Rogal, K. Reuter, M. Scheffler, Phys. Rev. Lett. 98 (2007) 046101. URL: http://link.aps.org/doi/10.1103/PhysRevLett.98.046101, http://dx.doi.org/10.1103/PhysRevLett.98.046101.

[35] J. Rogal, K. Reuter, M. Scheffler, Phys. Rev. B 77 (2008) 155410–12. URL: http://link.aps.org/abstract/PRB/v77/e155410, http://dx.doi.org/10.1103/PhysRevB.77.155410.

[36] D.J. Dooling, L.J. Broadbelt, Ind. Eng. Chem. Res. 40 (2001) 522–529. URL: http://dx.doi.org/10.1021/ie000310q, http://dx.doi.org/10.1021/ie000310q.

[37] L. Kunz, Entwicklung eines Computerprogramms zur kinetischen Monte Carlo Simulation von Oberflächenreaktionen auf Nanopartikeln, Diploma Thesis, Universität Karlsruhe, Karlsruhe, 2007. URL: http://www.itcp.kit.edu/deutschmann/img/content/06_LKunz_DA_UniKA.pdf.

[38] C. Garcia Cardona, V. Tikare, S.J. Plimpton, Int. J. Comput. Mater. Sci. Surf. Eng. 4 (2011) 37–54. URL: http://dx.doi.org/10.1504/IJCMSSE.2011.037351, http://dx.doi.org/10.1504/IJCMSSE.2011.037351.

[39] M. Stamatakis, D.G. Vlachos, J. Chem. Phys. 134 (2011) 214115. URL: http://scitation.aip.org/content/aip/journal/jcp/134/21/10.1063/1.3596751, http://dx.doi.org/10.1063/1.3596751.

[40] List of quantum chemistry and solid-state physics software, 2013. URL: http://en.wikipedia.org/w/index.php?title=List_of_quantum_chemistry_and_solid-state_physics_software&oldid=577618719, page Version ID: 577618719.

[41] CFD online, Codes—CFD-Wiki, the free CFD reference, 2013. URL: http://www.cfd-online.com/Wiki/Codes.

[42] N.W. Ashcroft, N.D. Mermin, Solid State Physics, first ed., Brooks Cole, 1976.

[43] P. Ramachandran, G. Varoquaux, Comput. Sci. Eng. 13 (2011) 40–51. http://dx.doi.org/10.1109/MCSE.2011.35.

[44] A.F. Voter, F. Montalenti, T.C. Germann, Annu. Rev. Mater. Res. 32 (2002) 321–346. URL: http://cat.inist.fr/?aModele=afficheN&cpsidt=13860930.

[45] M. Boudart, Catal. Lett. 65 (2000) 1–3. URL: http://link.springer.com/article/10.1023/A3A1019057002970, http://dx.doi.org/10.1023/A:1019057002970.

[46] K.A. Fichthorn, W.H. Weinberg, J. Chem. Phys. 95 (1991) 1090–1096. URL: http://link.aip.org/link/?JCP/95/1090/1, http://dx.doi.org/10.1063/1.461138.

[47] A.B. Bortz, M.H. Kalos, J.L. Lebowitz, J. Comput. Phys. 17 (1975) 10–18. URL: http://www.sciencedirect.com/science/article/B6WHY-4DDR37H-9W/2/c5e217e509b6a99a10c20c683754ce68, http://dx.doi.org/10.1016/0021-9991(75)90060-1.

[48] D.T. Gillespie, J. Comput. Phys. 22 (1976) 403–434. URL: http://www.sciencedirect.com/science/article/pii/0021999176900413, http://dx.doi.org/10.1016/0021-9991(76)90041-3.

[49] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, Numerical Recipes 3rd Edition: The Art of Scientific Computing, third ed., Cambridge University Press, 2007.

[50] J.J. Lukkien, J. P.L. Segers, P. A.J. Hilbers, R.J. Gelten, A. P.J. Jansen, Phys. Rev. E 58 (1998) 2598–2610. URL: http://link.aps.org/doi/10.1103/PhysRevE.58.2598, http://dx.doi.org/10.1103/PhysRevE.58.2598.

[51] S.R. Bahn, K.W. Jacobsen, Comput. Sci. Eng. 4 (2002) 56–66.

[52] S. Müller, J. Phys.: Condens. Matter. 15 (2003) R1429–R1500.

[53] Y. Zhang, J. Rogal, K. Reuter, Phys. Rev. B 74 (2006) 125414. URL: http://link.aps.org/doi/10.1103/PhysRevB.74.125414, http://dx.doi.org/10.1103/PhysRevB.74.125414.

[54] K. Reuter, First-principles kinetic monte carlo simulations for heterogeneous catalysis: Concepts, status and frontiers, 2009. URL: http://edoc.mpg.de/397332.

[55] L. Hyafil, R. Rivest, Inform. Process. Lett. 5 (1976) 15–17.

[56] T.H. Cormen, Introduction to Algorithms, The MIT Press, Cambridge, Masachusetts; London, 2009.

[57] P. Peterson, Int. J. Comput. Sci. Eng. 4 (2009) 296. URL: http://www.inderscience.com/info/inarticle.php?artid=29165, http://dx.doi.org/10.1504/IJCSE.2009.029165.

[58] F. Perez, B.E. Granger, Comput. Sci. Eng. 9 (2007) 21–29. URL: http://ipython.org/citing.html, http://dx.doi.org/10.1109/MCSE.2007.53.

[59] T.E. Oliphant, Comput. Sci. Eng. 9 (2007) 10–20. URL: http://link.aip.org/link/?CSX/9/10/1.

[60] J. Hunter, Comput. Sci. Eng. 9 (2007) 90–95. http://dx.doi.org/10.1109/MCSE.2007.55.

[61] R.M. Ziff, E. Gulari, Y. Barshad, Phys. Rev. Lett. 56 (1986) 2553. URL: http://link.aps.org/doi/10.1103/PhysRevLett.56.2553, http://dx.doi.org/10.1103/PhysRevLett.56.2553.

[62] gnu.org, 2007. URL: http://www.gnu.org/licenses/gpl.html.