Published in: Fiadeiro, J.L.; Schobbens, P.Y. (eds.): Recent Trends in Algebraic Development Techniques, 18th International Workshop, WADT 2006, La Roche en Ardenne, Belgium, June 1-3, 2006, Revised Selected Papers. Lecture Notes in Computer Science, Springer, Berlin Heidelberg New York, USA, 2007

Autonomous Units and Their Semantics - the Parallel Case*

Hans-Jörg Kreowski, Sabine Kuske

University of Bremen, Department of Computer Science
P.O.Box 330440, D-28334 Bremen, Germany
(kreo,kuske)@informatik.uni-bremen.de

Abstract

Communities of autonomous units are rule-based and graph-transformational devices to model data-processing systems that may consist of distributed and mobile components. The components may communicate and interact with each other, they may link up to ad-hoc networks. In this paper, we introduce and investigate the parallel-process semantics of communities of autonomous units.

1 Introduction

Communities of autonomous units are introduced in [1] as rule-based and graph-transformational devices to model processes that run interactively, but independently of each other in a common environment. The main goal of this approach is to cover new programming and modeling paradigms like communication networks, multiagent systems, swarm intelligence, ubiquitous, wearable and mobile computing in a common and systematic way with rigorous formal semantics. While the sequential process semantics is considered in [1], we introduce and start to investigate the parallel-process semantics of communities of autonomous units in this paper.

An autonomous unit consists of a goal, a set of rules, and a control condition. The rules can be applied to environments which are assumed to be graphs. Rule application is usually quite nondeterministic because many rules may be applicable to an environment and even a single rule may be applicable to various parts of the environment. The control condition can cut down this nondeterminism by dividing all possible rule applications into the "good" and the "bad" ones. The control condition gives the unit autonomy in the sense that the unit can decide for one of the good rule applications to be performed. The goal describes the environments the unit wants to reach. A set of autonomous units forms a community which is additionally provided with a description of initial environments (where computational processes can start) and with an overall goal. A process then consists of a finite or infinite sequence of rule applications.

* Research partially supported by the Collaborative Research Centre 637 (Autonomous Cooperating Logistic Processes: A Paradigm Shift and Its Limitations) funded by the German Research Foundation (DFG).

To cover parallelism, we assume that not only single rules but multisets of rules can be applied to environment graphs. This means that in each step many rules can be applied and single rules multiple times. As the rules may belong to different units, the autonomous units act in parallel. The units in a community are not directly aware of each other, but they may notice the outcome of the activities of their co-units because some of their rules may become applicable and others may loose this possibility. In this way, autonomous units can communicate and interact. To cover these phenomena in the process semantics of a single unit, we assume a change relation on environments that makes the environment dynamic. Then a parallel process of a single autonomous unit can be described as a sequence of application of multisets of rules of this unit in parallel with changes of the environment. These notions are introduced in Sections 3 and 4 and illustrated by the running example of a community of two autonomous units that work together and compute shortest paths. The basic ingredients including graphs, rules, rule applications, graphs class expressions to describe goals, and control conditions are defined by means of the notion of graph transformation approaches in Sect. 2. All components are quite generic so that they can be instantiated in various ways according to need or taste. Graph transformation approaches play a somewhat similar role for graph transformation than institutions for algebraic specification.

To shed some first light on the significance and usefulness of communities of autonomous units with parallel-process semantics, we compare our concepts with the parallelism provided by other well-known frameworks. In Sect. 5, we translate place/transition systems into communities of autonomous units and show that firing sequences of multisets of transitions correspond to parallel processes of the associated community. Similarly, cellular automata can be considered as communities of autonomous units as shown in Sect. 6. Cellular automata are particularly interesting as all their cells change states simultaneously so that the mode of computation is massively parallel. In Sect. 7, we discuss the relationship between communities of autonomous units and multiagent systems. As the latter are defined in an axiomatic way, the former can be seen as rule-based models providing an operational semantics for multiagent systems independent of the implementation of agents.

The introduction and investigation of autonomous units is mainly motivated by the Collaborative Research Centre 637 *Autonomous Cooperating Logistic Processes*. This interdisciplinary project focuses on the question whether logistic processes with autonomous control may be more advantageous than those with central control, especially regarding time, costs and robustness. The guiding principle of autonomous units is the integration of autonomous control into rule-based models of processes. The aims are

1. to describe algorithmic and particularly logistic processes in a very general and uniform way, based on a well-founded semantic framework,
2. to provide a range of applications that reaches from classical process chain models like the ones by Kuhn (see, e.g., [2]) or Scheer (see, e.g., [3]) and the

well-known Petri nets (see, e.g., [4, 5]) to agent systems see, e.g., [6]) and swarm intelligence (see, e.g., [7]),

3. to comprise the foundation of the dynamics of processes by means of rules where rule applications define process, transformation, and computation steps yielding local changes.

Archetypes of a rule-based approach to data processing are grammatical systems of all kinds (see, e.g., [8]) and term rewriting systems (see, e.g., [9]) as well as the domain of graph transformation (see, e.g., [10–12]) and DNA computing (see, e.g., [13]). The rule-based approach is meant to ensure an operational semantics as well as to lay the foundation for formal verification.

In [1] we have shown that autonomous units with sequential process semantics generalize our former modeling concept of graph transformation units (see, e.g., [14]). While the latter apply their rules without any interference from the outside, an autonomous unit works in a dynamic environment which may change because of the activities of other units in the community. This makes quite a difference because the running of the system is no longer controlled by a central entity. Clearly, this applies to the parallel case, too, because it generalizes the sequential case, obviously.

2 Parallel Graph Transformation Approaches

Graph transformation (see e.g. [10, 15]) constitutes a formal specification technique that supports the modeling of the rule-based transformation of graph-like, diagrammatic, and visual structures in an intuitive and direct way. The ingredients of graph transformation are provided by so-called graph transformation approaches. In this section, we recall the notion of a graph transformation approach as introduced in [14] but modified with respect to the purposes of this paper.

Two basic components of every graph transformation approach are a class of graphs and a class of rules that can be applied to these graphs. In many cases, rule application is highly nondeterministic – a property that is not always desirable. Hence, graph transformation approaches can also provide a class of control conditions so that the degree of nondeterminism of rule application can be reduced. Moreover, graph class expressions can be used in order to specify for example sets of initial and terminal graphs of graph transformation processes.

The basic idea of parallelism in a rule-based framework is the application of many rules simultaneously and also the multiple application of a single rule. To achieve these possibilities, we assume that multisets of rules can be applied to graphs rather than single rules.

Given some basic domain D, the set of all multisets D_* over D with finite carriers consists of all mappings $m\colon D \to \mathbb{N}$ such that the carrier $car(m) = \{d \in D \mid m(d) \neq 0\}$ is finite. For $d \in D, m(d)$ is called the multiplicity of d in m. The union or sum of multisets can be defined by adding corresponding multiplicities. D_* with this sum is the free commutative monoid over D where the multiset with empty carrier is the null element, i.e. $null\colon D \to \mathbb{N}$ with $null(D) = 0$. Note that the elements of D correspond to singleton multisets, i.e. for $d \in D, \hat{d}: D \to \mathbb{N}$ with $\hat{d}(d) = 1$ and $\hat{d}(d') = 0$ for $d' \neq d$. If $\mathcal{R}$ is a set of rules, $r \in R_*$ comprises a selection of rules each with some multiplicity. Therefore, an application of r to a graph yielding a graph models the parallel and multiple application of several rules.

Formally, a parallel graph transformation approach is a system $\mathcal{A} = (\mathcal{G}, \mathcal{R}, \mathcal{X}, \mathcal{C})$ the components of which are defined as follows.

– $\mathcal{G}$ is a class of *graphs.*
– $\mathcal{R}$ is a class of *graph transformation rules* such that every $r \in \mathcal{R}_*$ specifies a binary relation on graphs $SEM(r) \subseteq \mathcal{G} \times \mathcal{G}$.
– $\mathcal{X}$ is a class of *graph class expressions* such that each $x \in \mathcal{X}$ specifies a set of graphs $SEM(x) \subseteq \mathcal{G}$.
– $\mathcal{C}$ is a class of *control conditions* such that each $c \in \mathcal{C}$ specifies a set of sequences $SEM_{Change}(c) \subseteq SEQ(\mathcal{G})$ where $Change \subseteq \mathcal{G} \times \mathcal{G}$.[1] As we will see later the relation *Change* defines the changes that can occur in the environment of an autonomous unit. Hence, control conditions have a loose semantics which depends on the changes of the environment given by *Change*.

For technical simplicity we assume in the following that $\mathcal{A} = (\mathcal{G}, \mathcal{R}, \mathcal{X}, \mathcal{C})$ is an arbitrary but fixed parallel graph transformation approach. The multisets of rules in $\mathcal{R}_*$ are called parallel rules. A pair of graphs $(G, G') \in SEM(r)$ for some $r \in R_*$ is an application of the parallel rule r to G with the result G'. It may be also called a direct parallel derivation or a parallel derivation step.

Sometimes it is meaningful to parameterize the semantics of a graph class expression x by the class of graphs. i.e. $SEM_G(x) \subseteq \mathcal{G}$ for all $G \in \mathcal{G}$. This allows one to describe relations and functions between graphs rather than just sets of graphs. An example of this kind can be found in Sect. 4.

Examples

In the following we present some instances of the components of parallel graph transformation approaches. These will be used in the following Sections. Further examples of graph transformation approaches can be found in e.g. [10].

Graphs. A well-known instance for the class $\mathcal{G}$ is the class of all directed edge-labeled graphs. Such a graph is a system $G = (V, E, s, t, l)$ where V is a set of nodes, E is a set of edges, $s, t: E \to V$ assign to every edge its source $s(e)$ and its target $t(e)$, and the mapping l assigns a label to every edge in E. The components of G are also denoted by V_G, E_G, etc. As usual, a graph M is a subgraph of G, denoted by $M \subseteq G$ if $V_M \subseteq V_G$, $E_M \subseteq E_G$, and s_M, t_M, and l_M are the restrictions of s_G, t_G, and l_G to E_M. A graph morphism $g: L \to G$ from a graph L to a graph G consists of two mappings $g_V: V_L \to V_G$, $g_E: E_L \to E_G$ such that sources, targets and labels are preserved, i.e. for all $e \in E_L$, $g_V(s_L(e)) =$

[1] For a set A 2^A denotes its powerset and $SEQ(A)$ the set of finite and infinite sequences over A.

$s_G(g_E(e))$, $g_V(t_L(e)) = t_G(g_E(e))$, and $l_G(g_E(e)) = l_L(e)$. In the following we omit the subscript V or E of g if it can be derived from the context.

Other classes of graphs are trees, forests, Petri nets, undirected graphs, hypergraphs, etc.

Rules. As a concrete example of rules we consider the so-called dpo-rules each of which consists of a triple $r = (L, K, R)$ of graphs such that $L \supseteq K \subseteq R$. The application of a rule to a graph G yields a graph G', if one proceeds according to the following steps: (1) Choose a graph morphism $g: L \to G$ so that for all items x, y (nodes or edges) of L $g(x) = g(y)$ implies that x and y are in K. (2) Delete all items of $g(L) - g(K)$ provided that this does not produce dangling edges. (In the case of dangling edges the morphism g cannot be used.) (3) Add R to the resulting graph D, and (4) glue D and R by identifying the nodes and edges of K in R with their images under g. The conditions of (1) and (2) concerning g are called gluing condition.

Graph transformation rules can be depicted in several forms. In the following they are either shown in the form $L \supseteq K \subseteq R$ or by drawing only its left-hand side L and its right-hand side R together with an arrow pointing from L to R. The different nodes of K are distinguished by different fill-styles.

A graph transformation rule (L, K, R) with positive context is a quadruple (PC, L, K, R) such that $L \subseteq PC$. It can be applied to G by applying (L, K, R) to G as described provided that there is a morphism $g': PC \to G$ such that the restriction of g' to L equals g. A graph transformation rule with negative context is defined as (NC, L, K, R) where (L, K, R) is a rule and $L \subseteq NC$. It can only be applied to G if the negative context of L is not in G, i.e. if the morphism $g: L \to G$ cannot be extended to some morphism $g': NC \to G$ of which g is the restriction to L (cf. also [16]).

Given two rules $r_i = (L_i, K_i, R_i)$ $(i = 1, 2)$ their parallel composition yields the rule $r_1 + r_2 = (L_1 + L_2, K_1 + K_2, R_1 + R_2)$ where $+$ denotes the disjoint union of graphs. In the same way one can construct a parallel rule from any multiset $r \in \mathcal{R}_*$. For every pair $(G, G') \in SEM(r_1 + r_2)$ there exist graphs M_1 and M_2 such that (G, M_1) and (M_2, G') are in $SEM(r_1)$ and (G, M_2) and (M_1, G') are in $SEM(r_2)$. This means that the graph G' can also be obtained from G by applying the rules r_1 and r_2 sequentially and in any order. Moreover, let r_i $(i = 1, 2)$ be two (parallel) rules and let $g_i: L_i \to G$ be two morphisms that satisfy the gluing condition described in steps (1) and (2) of a rule application. Then r_1 and r_2 are independent w.r.t. g_i if the the following independence condition is satisfied:

$$g_1(L_1) \cap g_2(L_2) \subseteq g_1(K_1) \cap g_2(K_2).$$

In this case both rules can be applied to G in parallel via the application of $r_1 + r_2$ using the graph morphism $\langle g_1 + g_2 \rangle: L_1 + L_2 \to G$ such that $\langle g_1 + g_2 \rangle(x) = g_i(x)$ if x is an element of L_i (see, e.g., [17] for more details).

Graph class expressions. Every subset $M \subseteq \mathcal{G}$ is a graph class expression that specifies itself, i.e. $SEM(M) = M$. Moreover, every set $\mathcal{L}$ of labels specifies the class of all graphs in $\mathcal{G}$ the labels of which are elements of $\mathcal{L}$. Every set $P \subseteq \mathcal{R}_*$ of (parallel) graph transformation rules can also be used as a graph class expression specifying the set of all graphs that are reduced w.r.t. P where a graph is said to be reduced w.r.t. P if no rules of P can be applied to the graph. The least restrictive graph class expression is the term *all* specifying the class $\mathcal{G}$.

Control conditions. The least restrictive control condition is the term *free* that allows all parallel graph transformations, i.e. $\mathit{SEM}_{\mathit{Change}}(\mathit{free}) = \mathit{SEQ}(\mathcal{G})$ for all $\mathit{Change} \subseteq \mathcal{G} \times \mathcal{G}$. Another useful control condition is $\mathit{alap}(P)$ where $P \subseteq \mathcal{R}_*$. It applies P as long as possible. More precisely, for every $\mathit{Change} \subseteq \mathcal{G} \times \mathcal{G}$ $\mathit{SEM}_{\mathit{Change}}(\mathit{alap}(P))$ consists of all finite sequences $(G_0, \ldots G_n) \in \mathit{SEQ}(\mathcal{G})$ for which there is an $i \in \{0, \ldots, n\}$ such that no rule in P can be applied to the graphs in $(G_i, \ldots, G_n)$. The condition $\mathit{alap}(P)$ can also be used to specify infinite sequences, a more complicated case that is not needed here.

3 Autonomous Units

Autonomous units act within or interact on a common environment which is modeled as a graph. An autonomous unit consists of a set of graph transformation rules, a control condition, and a goal. The graph transformation rules contained in an autonomous unit *aut* specify all transformations the unit *aut* can perform. Such a transformation comprises for example a movement of the autonomous unit within the current environment, the exchange of information with other units via the environment, or local changes of the environment. The control condition regulates the application process. For example, it may require that a sequence of rules be applied as long as possible or infinitely often. The goal of a unit is a graph class expresson determining how the transformed graphs should look like.

Definition 1 (Autonomous unit). An *autonomous unit* is a system $\mathit{aut} = (g, P, c)$ where $g \in \mathcal{X}$ is the *goal*, $P \subseteq \mathcal{R}$ is a set of graph transformation rules, and $c \in \mathcal{C}$ is a control condition. The components of *aut* are also denoted by g_{aut}, P_{aut}, and c_{aut}, respectively.

An autonomous unit modifies an underlying environment while striving for its goal. Its semantics consists of a set of transformation processes being finite or infinite sequences of environment transformations. An environment transformation comprises the parallel application of local rules or environment changes typically performed by other autonomous units that are working in the same environment. These environment changes are given as a binary relation of environments. Because the parallel-process semantics is meant to describe the simultaneous activities of autonomous units, the environment changes must be possible while a single autonomous unit applies its rules. To achieve this, we assume that there are some rules, called metarules, the application of which defines environment changes. Consequently, environment changes and ordinary rules can be applied in parallel. Hence, in this parallel approach a transformation process of an autonomous unit consists of a sequence of parallel rule applications which combine local rule applications with environment changes specified by other components. Every autonomous unit has exactly one thread of control. Autonomous units regulate their transformation processes by choosing in every step only those rules that are allowed by its control condition. A finite transformation process is called successful if its last environment satisfies the unit goal. Every infinite transformation process is successful if it contains infinitely many environments that satisfy the goal.

Definition 2 (Parallel semantics).

1. Let $aut = (g, P, c)$ be an autonomous unit and let $Change \subseteq \mathcal{G} \times \mathcal{G}$. Let $\mathcal{MR} \subseteq \mathcal{R}_*$ be a set of parallel rules, called metarules, such that $SEM(\mathcal{MR}) = \bigcup_{r \in \mathcal{MR}} SEM(r) = Change$. Let $s = (G_0, G_1, G_2, \cdots) \in SEQ(\mathcal{G})$.
   Then $s \in PAR_{Change}(aut)$ if
   - for $i = 0, \cdots, |s|$ if s is finite[2] and for $i \in \mathbb{N}$ if s is infinite, $(G_{i-1}, G_i) \in SEM(r + r')$ for some $r \in P_*$ and $r' \in \mathcal{MR}$,
   - $s \in SEM_{Change}(c)$.
2. The sequence s is called a *successful transformation process* if s is finite and $G_{|s|} \in SEM(g)$ or there is an infinite monotone sequence $i_0 < i_1 < i_2 < \cdots$ with $G_{i_j} \in SEM(g)$ for all $j \in \mathbb{N}$.

The elements of $PAR_{Change}(aut)$ are sequences of applications of parallel rules which may be called the parallel processes of *aut*. Every single step of these processes applies a parallel rule of the form $r + r'$ where r is a parallel rule of the unit *aut* and r' is a metarule. Therefore, while the autonomous unit acts on the environment graph, the environment may change in addition. But as r and r' may be the null rule and $r + null = r$ as well as $null + r' = r'$, a step can also be an exclusive activity of *aut* or a change of the environment only.

Examples

As examples of autonomous units consider the units *minimum* and *sum* depicted in Fig. 1. The underlying graphs are labeled with natural numbers representing distances. The graph class expression for the goals of both units is *all* meaning that both units do not have any particular goal. The rule of *minimum* deletes the longer one out of two parallel edges labeled with natural numbers. The control condition of the unit *minimum* requires that the rule be applied as long as possible. In other words, a *minimum* process can only stop if no parallel edges are around. Two rule applications are independent if they delete different edges. This means that the rule can be applied k times in parallel, if the corresponding parallel rule application deletes k edges. In particular, one can transform each graph in a simple one without parallel edges in a single step.

[2] For a finite sequence s its number of elements is denoted by $|s|$.

The rule of the second unit *sum* can be applied to a path e_1, e_2 provided that e_1 and e_2 are labeled with natural numbers x and y and that there exists no edge from the source of e_1 to the target of e_2 that is labeled with a number $z \leq x+y$. The last requirement is expressed by the dashed edge which represents negative context. The rule inserts a new edge from the source of e_1 to the target of e_2 and labels it with $x+y$. This rule must also be applied as long as possible. Moreover, the rule can only be applied if the graph morphism from the left-hand side of the rule to the current graph is injective. This means that the rule can only be applied to loop-free paths. This condition ensures that no loops are produced in the computation of the sums of the edge labels. Each two applications of the *sum*-rule are independent because nothing is deleted. Consequently, the *sum*-rule can be applied k times in parallel for every $k \in \mathbb{N}$ as long as there are loop-free paths of length 2 satisfying the negative application condition. In particular, a parallel rule can be applied so that afterwards each loop-free path of length 2 and distance $x + y$ has got a parallel edge of distance $z \leq x + y$.

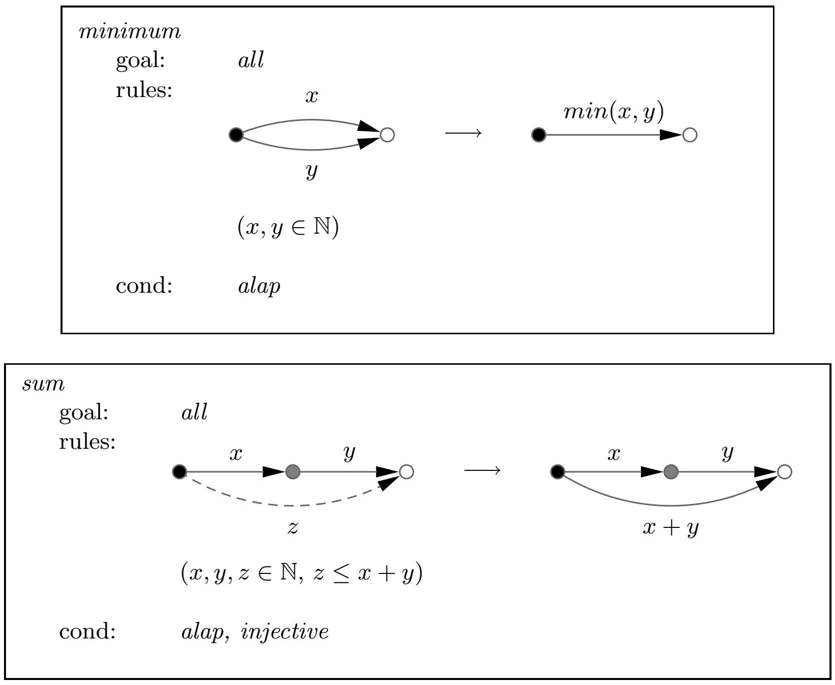


Fig. 1. Two autonomous units.

4 Communities of Autonomous Units

Autonomous units are meant to work within a community of autonomous units that modify the common environment together. In the parallel case these modifications take place in an interleaving manner. Every community is composed of an overall goal that should be achieved, an environment specification that specifies the set of initial environments the community may start working with, and a set of autonomous units. The overall goal may be closely related to the goals of the autonomous units in the community. Typical examples are the goals admitting only graphs that satisfy the goals of one or all autonomous units in the community.

Definition 3 (Community). A *community* is a triple $COM = (Goal, Init, Aut)$, where $Goal, Init \in \mathcal{X}$ are graph class expressions called the *overall goal* and the *initial environment specification*, respectively, and Aut is a set of autonomous units.

In a community all units work on the common environment in a self-controlled way by applying their rules. The change relation integrated in the semantics of autonomous units makes it possible to define a parallel semantics of a community in which every autonomous unit may perform its transformation processes. From the point of view of a single autonomous unit, the changes of the environment that are not caused by itself must be activities of the other units in the community. This is reflected in the following definition.

Definition 4 (Change relation). Let $COM = (Goal, Init, Aut)$ be a community. Then for each $aut \in Aut$ the *change relation* w.r.t. $aut Change(aut)$ is given by the parallel rules composed of rules of the autonomous units in COM other than aut as metarules, i.e. $Change(aut) = \bigcup\limits_{aut' \in Aut - \{aut\}} SEM(P_{aut'}{}^{*})$.

Every transformation process of a community must start with a graph specified as an initial environment of the community. Moreover, it must be in the parallel semantics of every autonomous unit participating in the community. Analogously to successful transformation processes of autonomous units, a finite transformation process of a community is successful if its last environment satisfies the overall goal. Every infinite transformation process of a community is successful if it meets infinitely many environments that satisfy the overall goal.

Definition 5 (Parallel community semantics).

1. Let $COM = (Goal, Init, Aut)$. Then the *parallel community semantics* of COM consists of all finite or infinite sequences $s = (G_0, G_1, \ldots) \in SEQ(\mathcal{G})$ such that $G_0 \in SEM(Init)$ and $s \in PAR_{Change(aut)}(aut)$ for all $aut \in aut$.
2. The sequence s is called a *successful transformation process* if s is finite and $G_{|s|} \in SEM(Goal)$ or there is an infinite monotone sequence $i_0 < i_1 < \cdots$ such that $G_{i_j} \in SEM(Goal)$ for all $j \in \mathbb{N}$.

3. The parallel community semantics is denoted by $PAR(COM)$. Its elements are called parallel processes of COM.

As the definition of the community semantics shows, there is a strong connection between the semantics of a community $COM = (Goal, Init, Aut)$ and the semantics of an autonomous unit $aut \in Aut$. The parallel semantics of COM is a subset of the semantics of aut with respect to the change relation $Change(aut)$. Conversely, one may take the intersection of the parallel semantics of all autonomous units with respect to their own change relation each and restrict this to the sequences starting in an initial environment. Then one gets the parallel semantics of the community. This reflects the autonomy because no unit can be forced to do anything that is not admitted by its own control.

Example

In following we shortly illustrate how communities of autonomous units can be used to find shortest paths by working in parallel. The presented community $CAU(spath)$ is a parallel variant of the famous shortest-path algorithm of Floyd [18].

As initial environments $CAU(spath)$ admits all directed edge-labeled graphs so that every edge from v to v' is labeled with a number representing the distance from v to v'. The set of autonomous units of $CAU(spath)$ consists of the two units *minimum* and *sum* presented in Fig. 1 above.

The goal of the community $CAU(spath)$ is twofold: (1) Whenever there is an edge e and a path p from a node v to a node v' the distance of e has to be less or equal to the distance of p (i.e. the distances edges are the shortest). (2) Whenever there is an edge from a node v to a node v', there is a shortest path from v to v' in the initial environment with the same distance. (This guarantees that the computed edges yield the distances of the shortest paths in the initial graphs.)

The parallel semantics of $CAU(spath)$ is equal to the parallel semantics of the unit *minimum* if the change relation is given by all (parallel) transformations of *sum* and if the transformation processes start with an initial environment of $CAU(spath)$. The analogous property holds for the unit *sum*. More formally, this means

$$\begin{aligned} PAR(CAU(spath)) &= PAR_{PAR(minimum)}(sum)|SEM(Init_{CAU(spath)}) \\ &= PAR_{PAR(sum)}(minimum)|SEM(Init_{CAU(spath)}). \end{aligned}$$

Moreover it can be shown that the community $CAU(spath)$ works correctly, i.e. the parallel semantics of $CAU(spath)$ contains only finite sequences $(G_0, \ldots, G_n)$ such that for every two nodes v and v' there is an edge e with distance x in G_n from v to v' if and only if the shortest path in G_0 from v to v' has distance x (cf. [19] for a correctness proof concerning the sequential variant of this algorithm).

5 Petri Nets

The area of Petri nets (see, e.g., [4, 5]) is established as one of the oldest, well-known, and best studied frameworks in which parallelism is precisely introduced and investigated. Hence it is meaningful to relate Petri nets with the parallel semantics of communities of autonomous units and to shed some light on the significance of the latter in this way. It turns out for instance that place/transition nets, which are the most frequently used variants of Petri nets, can be seen as a special case of communities of autonomous units where the transitions play the role of the units.

A place/transition system $S = (P, T, F, m_0)$ consists of a set P of places, a set T of transitions, a flow relation $F \subseteq (P \times T) \cup (T \times P)$, and an initial marking $m_0 : P \to \mathbb{N}$, i.e. $m_0 \in P_*$. The sets P and T are assumed to be disjoint so that $N = (P \cup T, F)$ is a bipartite graph (with the projections as source and target maps respectively).

The transitions can fire if they are enabled meaning that they transform markings being multisets of places. This is formally defined as follows.

A multiset $m \in P_*$ is called a marking. A transition $t \in T$ is enabled w.r.t. m if ${}^\bullet t \leq m$ where ${}^\bullet t : P \to \mathbb{N}$ describes the input places of t that flow into t, i.e. ${}^\bullet t(p) = 1$ if $(p, t) \in F$ and ${}^\bullet t(p) = 0$ otherwise. The order ${}^\bullet t \leq m$ is defined place-wise, i.e. ${}^\bullet t(p) \leq m(p)$ for all $p \in P$ or, in other words, $m(p) \neq 0$ if $(p, t) \in F$. If t is enabled w.r.t. m, it can fire resulting in a marking which is obtained by subtracting ${}^\bullet t$ from m and by adding $t^\bullet$ given by $t^\bullet(p) = 1$ if $(t, p) \in F$ and $t^\bullet(p) = 0$ otherwise. Such a firing is denoted by $m\,[t\rangle\, m - {}^\bullet t + t^\bullet$. If one interprets $m(p)$ as the number of tokens on the place p, then the firing of t removes one token from each input place of t and puts a new token on each of the output places of t.

Analogously, a multiset of transitions $\tau \in T_*$ can be fired in parallel by summing up all input places and all output places:

$$m[\tau > m - {}^\bullet\tau + \tau^\bullet \text{ provided that } \; {}^\bullet\tau \leq m.$$

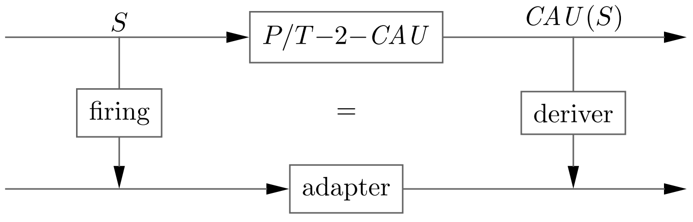
Here ${}^\bullet\tau$ and $\tau^\bullet$ are defined by ${}^\bullet\tau(p) = \sum_{t \in T} \tau(t) * {}^\bullet t(p)$ and $\tau^\bullet(p) = \sum_{t \in T} \tau(t) * t^\bullet(p)$ for all $p \in P$, and the order ${}^\bullet\tau \leq m$ is again place-wise defined, i.e. ${}^\bullet\tau(p) \leq m(p)$ for all $p \in P$.

Now one may consider the underlying net, which is the bipartite graph N, together with a marking as an environment. This is represented by the marking because the net is kept invariant. The transitions can be seen as rules and the firing of multisets of transitions as parallel rule application. As environment class expressions, we need single markings describing themselves as initial markings and the constant *all* accepting all environments. The only control condition needed is the constant *free* allowing a unit the free choice of rules. Then these components form a graph transformation approach, and a place/transition system $S = (P, T, F, m_0)$ can be translated into a community of autonomous units

$$CAU(S) = (all, m_0, \{aut(t) \mid t \in T\})$$

With $aut(t) = (all, \{t\}, free)$.

It is not difficult to prove that there is a one-to-one correspondence between the firing sequences of the place/transition system S that start in the initial marking and the parallel processes of the related community of autonomous units $CAU(S)$ if one removes the firing symbol from the firing sequences. The following figure depicts the relation.



The adapter transforms a firing sequence into a sequence of markings by removing the firing symbol between each two successive markings.

6 Cellular Automata

Cellular automata (see, e.g., [20]) are well-known computational devices that exhibit massive parallelism. A cellular automaton consists of a network of cells each of which being in a particular state. In a computational step, all cells change their states in parallel depending on the states of their neighbours. To simplify technicalities, one may assume that the neighbourhoods of all cells are regular meaning that they have the same number of neighbours and that the state transition of all cells is based on the same finite-state automaton. This leads to the following formal definition.

A cellular automaton is a system $CA = (G, A, init)$ where

- $G = (V, E, s, t, l)$ is a regular graph of type k subject to the condition: for each $v \in V$, there is a sequence of edges $e(v)_1, \cdots, e(v)_k$ with $s(e(v)_i) = v$ and $l(e(v)_i) = i$ for all $i = 1, \ldots, k$,
- $A = (Q, Q^k, d)$ is a finite-state automaton, i.e. Q is a finite set of states, Q^k is the input set and $d \subseteq Q \times Q^k \times Q$ is the state transition with k-tuples of states as inputs, and
- $init\colon V \to Q$ is the initial configuration.

If the graph G is infinite, one assumes a sleeping state $q_0 \in Q$ in addition such that $d(q_0, q_0^k) = \{q_0\}$ and $active(init) = \{v \in V \mid init(v) \neq q_0\}$ is finite.

The latter means that only a finite number of nodes is not sleeping initially and that the sleeping state can only wake up if not all inputs are sleeping. The edge sequence $e(v)_1 \cdots e(v)_k$ yields the neighbours of v as targets, i.e. $t(e(v_1)) \cdots t(e(v)_k))$.

A configuration is a mapping $con: V \to Q$ that assigns each node (which represent cells) an actual state. Configurations can be updated by state transitions of all actual states using the states of the neighbours as input.

Let $con: V \to Q$ be a configuration. Then $con': V \to Q$ is a directly derived configuration, denoted by $con \longmapsto con'$, if the following holds for every $v \in V$:

$$con'(v) \in d(con(v), con(t(e(v)_1)) \cdots con(t(e(v)_k))).$$

The semantics of a cellular automaton CA is given by all configurations that can be derived from the initial configuration:

$$L(CA) = \{con \mid init \overset{*}{\longmapsto} con\}$$

It is worth noting and easy to prove that all configurations derivable from the initial configuration have a finite number of nodes with non-sleeping states. Typical examples of regular graphs underlying cellular automata are the following: The set of nodes is the set of all points in the plane with integer coordinates, i.e. $\mathbb{Z} \times \mathbb{Z}$. Then there are various choices for the neighbourhood of a node $(x, y) \in \mathbb{Z} \times \mathbb{Z}$. that establish the set of edges with sources and targets. Typical ones are:

1. the four nearest nodes (to the north, east, south and west): $(x, y+1), (x+1, y), (x, y-1), (x-1, y)$,
2. the eight nearest nodes: $(x, y+1), (, x+1, y+1), (x+1, y), (x+1, y-1), (x, y-1), (x-1, y-1), (x-1, y), (x-1, x+1)$,
3. only the neighbours to the south and the west: $(x, y-1), (x-1, y)$.

The edges connecting a node with a neighbour may be numbered in the given order.

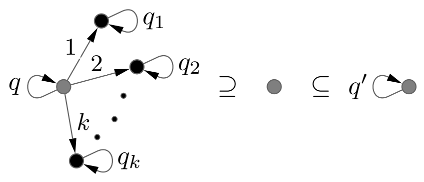
Cellular automata can be translated into communities of autonomous units where each cell is transformed into a unit.

$$\begin{tikzcd} {} \arrow[r, "CA"] & \boxed{celaut{-}2{-}CAU} \arrow[r, "CAU(CA)"] & {} \end{tikzcd}$$

The environments are given by the configurations. To get a graph representation of a configuration con, the underlying regular graph G is extended by a loop at each node v which is labeled with $con(v)$, i.e. $(G, con) = (V, E+V, \overline{s}, \overline{t}, \overline{l})$, such that G is a subgraph and $\overline{s}(v) = \overline{t}(v) = v$ and $\overline{l}(v) = con(v)$ for all $v \in V \subseteq E+V$.

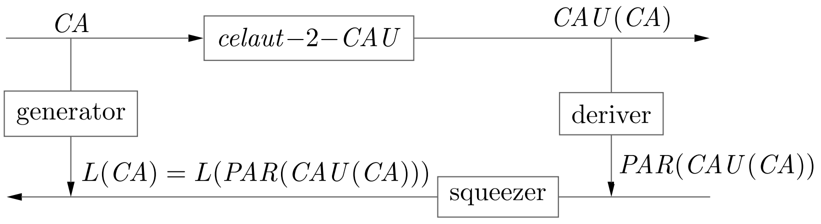
The community of autonomous units $CAU(CA)$ associated with a cellular automaton $CA = (G, A, init)$ gets $(G, init)$ as initial environment and an autonomous unit $aut(v)$ for each $v \in V$.

Each of these units has the same rules with positive context which reflect the state transition:



provided that $q' \in d(q, q_1, \cdots, q_k)$ and not all their states are sleeping. Moreover each unit $aut(v)$ has got a control condition requiring that the central node must be mapped to v. This means that the matching of the left-hand side of each rule is fixed and no search for it is needed. Moreover, the matchings of rules of different units are not overlapping so that the rules can be applied in parallel. If a node is sleeping and all its neighbours are sleeping too, then no rule can be applied. A parallel rule is maximal if all other nodes are matched. It is easy to see that the application of such a parallel rule corresponds exactly to a derivation step on the respective configurations.

In other words, the semantics of a cellular automaton CA and the parallel semantics $PAR(CAU(CA))$ of the community of autonomous units $CAU(CA)$ are nicely related to each other if one applies maximal parallel rules only. Let $L(PAR(CAU(CA)))$ be the set of configurations con such that a parallel process $(G, init) \cdots (G, con) \in PAR(CAU(CA))$ exists. Then $L(PAR(CAU(CA)))$ equals $L(CA)$. This correctness result is depicted by the following figure.



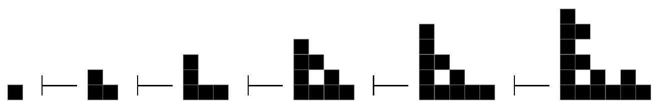
A finite-state automaton fitting the third neighbourhood is

$$SIER = (\{b, w\}, \{b, w\}^2, d)$$

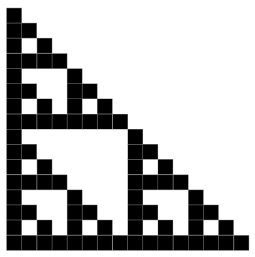
with $d(b, x, y) = b$ for all $x, y \in \{b, w\}$, $d(w, b, w) = d(w, w, b) = b$, and $d(w, b, b) = d(w, w, w) = w$. The state w is sleeping.

The initial configuration may map the node $(0, 0)$ to b and all others to w.

There is a very nice pictorial interpretation of this cellular automaton. Each node (x, y) is represented by the square spanned by the points $(x, y), (x, y + 1), (x + 1, y + 1), (x + 1, y)$. If a configuration con assigns b to (x, y), the square gets the color black and white otherwise. The initial configuration consists of a single black square. Because the automaton is dertemінistic, there is exactly one derivation for each length, where the shorter derivations are initial sections of the longer ones. The first five steps are



After 15 steps the picture looks as follows:



And all derived configurations can be seen as approximations of the Sierpinski triangle, a famous fractal. (see, e.g., [21]).

7 Multiagent Systems

Multiagent systems are modelling and programming devices well-known in artificial intelligence (see, e.g., Wooldridge et al. [6]). A multiagent system provides a set of agents and an initial environment state. Starting at this state, the agents change environment states step by step where they act together in parallel in each step. Each agent can perceive the current environment state at least partly. Based on this perception and its own intention, the agent chooses an action to be performed next. Therefore, a process in a multiagent system *MAS* is a sequence

$$es_0\ es_1\ es_2\ \cdots$$

of environment states es_i for all i where es_0 is initial. Each environment state es_{i+1} is given by the state transition τ of *MAS* depending on the previous state es_i and the action $act(ag)_i$ chosen by every agent ag of *MAS*. The choice of such an action is made according to the function do_{ag} each agent ag is provided with. The *do*-function yields an action depending on the agent's perception $perceive_{ag}(es_i)$ of the current state and the agent's intention $intend_{ag}$. The global state transition τ and the functions $do_{ag}, perceive_{ag}$ and $intend_{ag}$ which are individually assigned to each agent ag of *MAS* are assumed to satisfy some consistency properties (cf. [6] for details). Altogether, multiagent systems form a logical and axiomatic approach to model distributed information proccesses that interact on common environment states. It should be noted that all functions of *MAS* are allowed to be nondeterministic so that chosen actions as well as the next state may not be uniquely determined.

Communities of autonomous units are nicely related to multiagent systems as may be not too surprising from the description above. Actually, a community of autonomous units $CAU = (Goal, Init, Aut)$ turns out to be a particular rule-based model of multiagent systems. The environment states are the environment graphs. The agents are the units. The initial graphs are explicitly given. The rules – or the parallel rules likewise – of a unit are the actions of the agent embodied by the unit. The control condition plays the role of the *do*-function because it identifies the rules that are allowed to be applied next. As the control condition can take into account the current environment graph, the perception of the agent is also reflected. The most important aspect of the correspondence between agents and units is the transition function that is made operational by means of parallel rule application. The parallel rule to be applied in each step is just the sum of all rules chosen by the various units according to their control. If one considers the parallel rules of a unit as actions, the parallel processes of the community and the processes of the corresponding multiagent system coincide. If only the rules are actions, the multiagent system is not parallel with respect to single agents. That all agents must act in parallel in each step is a minor difference to community processes because a multiagent system may provide void actions without effect to the environment.

The relation between communities of autonomous units and multiagent systems is only sketched because a full formal treatment is beyond the scope of the paper. But even on this informal level, it should be clear that both concepts fit nicely together and may profit from each other. Communities of autonomous units represent explicit models of multiagent systems on one abstract, implementation-independent level with a precise, rule-based operational semantics. The *perceive-do* mechanism of multiagent systems to choose next actions provides a wealthy supply of control conditions that can be employed in modeling by means of autonomous units.

8 Conclusion

In this paper, we have supplemented the sequential-process semantics of autonomous units in [1] by a parallel-process semantics which allows the units of a community to act and interact simultaneously in a common environment. Moreover, we have studied the relationship of autonomous units to three other modeling frameworks that provide notions of parallelism: Petri nets, cellular automata, and multiagent systems. While the first two have been correctly transformed into autonomous units, autonomous units have turned out to be models of multiagent systems in that the environments are instantiated as graphs, the actions of agents as rules, and the environment transformation as parallel rule application. This is the very first step of the investigation of autonomous units in a parallel setting. The future study may include the following topics:

1. Besides Petri nets, the theory of concurrency offers a wide spectrum of notions of processes like communicating sequential processes, calculus of com-

municating systems, traces, and bigraphs. A detailed comparison of them with autonomous units can lead to interesting insights.

2. The basic idea of autonomous units is that each of them decides for itself which rule is to be applied next. They are independent of each other and the parts of the environment graphs where their rules apply may be far away from each other. Hence a sequential behaviour of the community (like in many card and board games) will be rarely adequate. But also the parallel behaviour does not always reflect the actual situations to be modeled because a parallel step provides a graph before and a graph after the step whereas there may be activities of units that cannot be related to each other with respect to time. A proper concurrent semantics of autonomous units may fix this problem.
3. In all explicit examples, we have made use of the fact that independent rule applications can be applied in parallel. This holds in the DPO approach (as well as in the SPO approach) together with several other properties and constructions that relate parallel and sequential processes yielding true concurrency for example. It seems to be meaningful to extend these considerations to the framework of autonomous units.

References

1. Hölscher, K., Kreowski, H., Kuske, S.: Autonomous units and their semantics — the sequential case. In: Proc. International Conference of Graph Transformation. Volume 4178 of Lecture Notes in Computer Science. (2006) 245–259
2. Kuhn, A.: Prozessketten – Ein Modell für die Logistk. In Wiesendahl, H.P., ed.: Erfolgsfaktor Logistikqualität. Springer Verlag (2002) 58–72
3. Scheer., A.W.: Vom Geschäftsprozeß zum Anwendungssystem. Springer Verlag (2002)
4. Reisig, W.: Elements of Distributed Algorithms: Modeling and Analysis With Petri Nets. Springer Verlag (1998)
5. Bause, F., Kritzinger, P.: Stochastic Petri Nets - An Introduction to the Theory (Second Edition). Vieweg & Sohn Verlag Braunschweig/Wiesbaden (Germany) (2002)
6. Wooldridge, M., Jennings, N.R.: Intelligent agents: Theory and practice. The Knowledge Engineering Review **10**(2) (1995)
7. Kennedy, J., Eberhart, R.C.: Swarm Intelligence. Morgan Kaufmann (2001)
8. Rozenberg, G., Salomaa, A., eds.: Handbook of Formal Languages, Vol. 1–3. Springer Verlag (1997)
9. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press (1998)
10. Rozenberg, G., ed.: Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 1: Foundations. World Scientific, Singapore (1997)
11. Ehrig, H., Engels, G., Kreowski, H.J., Rozenberg, G., eds.: Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 2: Applications, Languages and Tools. World Scientific, Singapore (1999)
12. Ehrig, H., Kreowski, H.J., Montanari, U., Rozenberg, G., eds.: Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 3: Concurrency, Parallelism, and Distribution. World Scientific, Singapore (1999)

13. Păun, G., Rozenberg, G., Salomaa, A.: DNA Computing — New Computing Paradigms. Springer Verlag (1998)
14. Kreowski, H.J., Kuske, S.: Graph transformation units with interleaving semantics. Formal Aspects of Computing **11**(6) (1999) 690–723
15. Andries, M., Engels, G., Habel, A., Hoffmann, B., Kreowski, H.J., Kuske, S., Plump, D., Schürr, A., Taentzer, G.: Graph transformation for specification and programming. Science of Computer Programming **34**(1) (1999) 1–54
16. Habel, A., Heckel, R., Taentzer, G.: Graph grammars with negative application conditions. Fundamenta Informaticae **26**(3,4) (1996) 287–313
17. Corradini, A., Ehrig, H., Heckel, R., Löwe, M., Montanari, U., Rossi, F.: Algebraic approaches to graph transformation part I: Basic concepts and double pushout approach. [10] 163–245
18. Even, S.: Graph Algorithms. Computer Science Press (1979)
19. Kreowski, H.J., Kuske, S.: Graph transformation units and modules. In Ehrig, H., Engels, G., Kreowski, H.J., Rozenberg, G., eds.: Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 2: Applications, Languages and Tools. World Scientific, Singapore (1999) 607–638
20. Wolfram, S.: A New Kind of Science. Wolfram Media, Inc. (2002)
21. Peitgen, H., Jürgens, H., Saupe, D.: Chaos and Fractals. Springer, Berlin (2004)