International Joint Conference on Theory and Practice of Software Development (TAPSOFT'97), April 14-18, 1997, Lille, France
Springer LNCS 1214, pp. 535-549.

Specifying Complex and Structured Systems with Evolving Algebras

Wolfgang May *

Institut für Informatik, Universität Freiburg
Am Flughafen 17, 79110 Freiburg, Germany
may@informatik.uni-freiburg.de

Abstract. This paper presents an approach for specifying complex, structured systems with Evolving Algebras by means of aggregation and composition. Evolving algebras provide a formal method for *executable* specifications which has been employed for specifying several algorithms and programming languages. With its transition system-like rule-based syntax, the concept is as well very intuitive as well-suited for formal reasoning and verification.

Following the need for structuring capabilities in specification frameworks, the paper proposes a concept for hierarchically structuring Evolving Algebras corresponding to the semantics of the system to be modeled, allowing to build up complex systems from simpler ones by several combinators. The concept can be generalized to arbitrary rule-based state-oriented formalisms.

In such systems, transitions regarded as atomic on the corresponding level are allowed to be specified by computations performed by sub-Evolving-Algebras instead of single rules. The subsystems provide a natural way of encapsulating data and behaviour while a computation is running. Communication is done via distinguished locations accessible to the participating systems.

1 Introduction

Formal specification methods gain increasing interest in system design and validation. Their application to complex tasks, for instance workflow systems, requires structuring capabilities of the formal framework.

Evolving Algebras [Gur91] provide a formal description of operational semantics for algorithms in an easy-to-understanding way, tailored to the natural abstraction level of the algorithm. They have been employed for specifying several algorithms and operational semantics of programming languages. With its formal, transition-system like rule-based syntax, the concept is also well-suited for formal reasoning and verification.

Evolving Algebra specifications are directly executable [GH94, BP95] thus, because of their clear and intuitive concept they are well-suited for prototyping, testing, and simulating systems in the design and development phase.

* Supported by grant no. GRK 184/1-96 of the Deutsche Forschungsgemeinschaft.

On the other side, the flat concept, based on elementary updates, provides no means for specifying encapsulation, communication, or any system structure: In every state, all rules have equal rights, "seeing" all data and communicating implicitly via the whole signature. The "length" of a computation in the sense of rule applications introduces an implicit notion of time. Additional rules for synchronization have to be applied, which impair the clear and intuitive specification. Thus, semantical, higher-level structuring devices for Evolving Algebras seem appropriate for specifying real-world complex systems.

In this paper, a concept for equipping Evolving Algebras with a modular structure allowing to build up complex systems from simpler ones by several combinators is worked out: Transitions regarded as atomic on the corresponding level are allowed to be carried out by computations of sub-Evolving-Algebras running in isolation on an own signature, communicating via distinguished locations of the shared part of the signatures. Thus, subsystems provide a natural way of encapsulating data and internal behaviour. Their behaviour is aggregated to atomic transitions on the upper level.

The paper is structured as follows: In the next section, the classical concept of Evolving Algebras as presented in [Gur94] is reviewed and a motivating example is pointed out. Section 3 relates computations of Evolving Algebras to sequences of first-order interpretations, setting the base for a logical treatment. In Section 4, some combinators for structuring systems of Evolving Algebras are presented. Section 5 formally defines the notion of systems of Evolving Algebras and gives an operational model for structured Evolving Algebras as constructs of simple Evolving Algebras. Section 6 completes the work with an overview of related work and some concluding remarks.

2 Evolving Algebras

Evolving Algebras[2] [Gur88, Gur91, Gur94] are transition systems whose states are *static algebras*, ie first-order interpretations over a *functional* signature Σ. The transition relation is specified by *rules* describing the modifications of the interpretation of the function symbols from one state to another.

For static algebras, the most concepts are taken over from predicate logic: the *signature* Σ of a static algebra is a finite set of function symbols, each with a fixed arity. *Terms* are defined as usual.

A *static algebra* $\mathcal{A} = (A, \mathcal{S})$ over a signature Σ consists of a non-emtpy set $\mathcal{S}$ (*superuniverse*) and an interpretation A of the function symbols, $A(f) : \mathcal{S}^{\mathsf{ord}(f)} \to \mathcal{S}$. As usual, $\mathcal{A}$ can be extended straightforwardly to an evaluation of terms. For an n-ary function symbol $f \in \Sigma$ and $s_1, \ldots, s_n \in \mathcal{S}$, $(f, s_1, \ldots, s_n)$ is a *location* over Σ and $\mathcal{S}$. In order to handle partial functions, Σ includes a constant undef which is interpreted as the element *undef* $\in \mathcal{S}$. Additionally, Σ includes the constants true and false, mapped to the universe $\mathsf{Bool} := \{\mathit{true}, \mathit{false}\} \subset \mathcal{S}$. The only relation in static algebras is the equality relation. With the universe Bool, every relation can be represented by its characteristic function.

[2] since recently aka Gurevich Abstract State Machines

For a static algebra $\mathcal{A} = (A, \mathcal{S})$ and a function symbol, its *domain* is defined as

$$\mathsf{dom}(f) := \{(s_1, \ldots, s_{\mathsf{ord}(f)}) \in \mathcal{S}^{\mathsf{ord}(f)} \mid (A(f))(s_1, \ldots, s_{\mathsf{ord}(f)}) \neq \mathit{undef}\,\} \,.$$

Furthermore, $\mathsf{dom}(\mathcal{A}) := \{(f, s_1, \ldots, s_n) \mid f \in \Sigma, s_i \in \mathcal{S}\}$ is a subset of the set of locations over Σ and $\mathcal{S}$.

An Evolving Algebra is given by an initial state $\mathcal{I}(\mathcal{E})$ (which also gives the interpretation of state-independent function symbols for all states) and a set $\mathcal{R}(\mathcal{E})$ of transition rules describing the change of the interpretation of state-dependent function symbols in a Pascal-like syntax. Signature and superuniverse are constant over all states, so there is a signature $\Sigma(\mathcal{E})$ and a superuniverse $\mathcal{S}(\mathcal{E})$.

Definition 1 *An* elementary update u *is an update of the interpretation of a function symbol at one location:* $u:\ f(t_1, \ldots, t_n) := t_0$ *, where* f *is an* n*-ary function symbol and* t_i *are terms.*

The set of rules is defined by structural induction as follows:
- *If* u *is an elementary update, then* u *is a rule.*
- *If* $r_1, \ldots, r_n$ *are rules, then* $r_1, \ldots, r_n$ *is a rule ("block").*
- *If* $g_1, \ldots, g_k$ *are boolean terms over* Σ *("guards") and* $r_1, \ldots, r_{k+1}$ *are rules, then* $r =$ if g_1 then r_1 elsif g_2 then r_2 elsif $\ldots$ elsif g_k then r_k else r_{k+1} endif *is a rule.*

A program *of an Evolving Algebra is a finite set of rules.*

A rule schema is a rule containing free variables, standing for all its ground instances. As in logic programming, a rule schema is *safe* iff all variables occurring free on the left side of an updates also occur positively in a corresponding guard. Thus, on finite interpretations, execution of safe rule schemas can be done by executing finitely many ground instances.

Definition 2 *An* update *over a signature* Σ *and a superuniverse* $\mathcal{S}$ *is a pair* (ℓ, s)*, where* ℓ *is a location and* $s \in \mathcal{S}$*.*

Definition 3 *Let* $\mathcal{A}$ *be a static algebra.*
- *For* $r \equiv f(t_1, \ldots, t_n) := t_0$*,* $\mathsf{Upd}(r, \mathcal{A}) := \{(f, \mathcal{A}(t_1), \ldots, \mathcal{A}(t_n), \mathcal{A}(t_0))\}$*.*
- *For a block* $r \equiv r_1, \ldots, r_n$*,* $\mathsf{Upd}(r, \mathcal{A}) := \mathsf{Upd}(r_1, \mathcal{A}) \cup \ldots \cup \mathsf{Upd}(r_n, \mathcal{A})$*.*
- *For a rule* $r \equiv$ if g_1 then r_1 elsif g_2 then r_2 $\ldots$ elsif g_k then r_k else r_{k+1} endif *and* $\mathcal{A} \models g_i$ *and* $\mathcal{A} \not\models g_j$ *for all* $j < i$*,* $\mathsf{Upd}(r, \mathcal{A}) := \mathsf{Upd}(r_i, \mathcal{A})$*.*
  If $\mathcal{A} \not\models g_j$ *for all* j*, then* $\mathsf{Upd}(r, \mathcal{A}) := \mathsf{Upd}(r_{k+1}, \mathcal{A})$*.*
- *For a program* $P = \{r_1, \ldots, r_n\}$*,* $\mathsf{Upd}(P, \mathcal{A}) := \mathsf{Upd}(r_1, \mathcal{A}) \cup \ldots \cup \mathsf{Upd}(r_n, \mathcal{A})$*.*
- *A set* U *of updates is consistent if for every location* ℓ*,* $|\{s \mid (\ell, s) \in U\}| \leq 1$*.*

Definition 4 *Let* U *be a consistent set of updates and* $\mathcal{A}$ *a static algebra. Then the state* $\mathcal{B} = (B, \mathcal{S})$ *obtained by executing* U *is given as*

$$(B(f))(s_1, \ldots, s_n)) = \begin{cases} s & \text{if } (f, s_1, \ldots, s_n, s) \in U \,, \\ (A(f))(s_1, \ldots, s_n) & \text{otherwise} \,. \end{cases}$$

If in some state the calculated update set is inconsistent, the system stops.

Definition 5 *The static algebra which is obtained by applying a ground rule $r \in \mathsf{grd}(\mathcal{R}(\mathcal{E}))$ in a static algebra $\mathcal{A}$ (ie executing $\mathsf{Upd}(r)$) is denoted by $r(\mathcal{A})$. For a set $\mathcal{R}$ of rules, $\mathcal{R}(\mathcal{A})$ denotes the static algebra which is obtained by executing $\mathsf{Upd}(\mathcal{R})$ in $\mathcal{A}$. $\mathcal{R}^*(\mathcal{A})$ denotes the static algebra obtained by running the Evolving Algebra $(\mathcal{A}, \mathcal{R})$ until it reaches a fixpoint.*

Example 1 Imagine a drink service: there are two "producers", C produces a glass of champagne, O produces an orange juice. Also there are three "processing units": CS takes a glass of champagne and sells it, OS takes an orange juice and sells it, CO takes a glass of champagne and an orange juice and produces two mixed drinks. The components can only communicate via two locations c and o, each of them offering place for exactly one glass. C and O see only the location which they output to, CS, OS, and CO see both locations. Their visible behaviour can be specified as follows:

C: if $c = empty$ and $order_C$ then $c := glass$
O: if $o = empty$ and $order_O$ then $o := glass$
C: if $c = empty$ and $order_C$ then $c := glass$
O: if $o = empty$ and $order_O$ then $o := glass$
CS: if $c = glass$ and $o = empty$ then $c := empty$
OS: if $c = empty$ and $o = glass$ then $o := empty$
CO: if $c = glass$ and $o = glass$ then $c := empty,\ o := empty$

where $order_S$ and $order_O$ are set by some more rules. On some abstraction level, the internal computations of C and O are irrelevant, and the above rules work well: If one orders champagne and orange, a mix is produced. Now, imagine that C and O stand for more complex processes – the output is a function computed from the input, and should also be specified by rules. Since all rules are united in one set, there is no encapsulation or synchronization. In contrary, the "length" of a computation in the sense of rule applications introduces an implicit, formulation-dependent notion of time. It is very unlikely that $c = glass$ and $o = glass$ at some point of that implicit time. Thus nobody will get a mixed drink, even if he orders both components.

Thus, semantical, formulation-independent higher-level structuring devices for Evolving Algebras seem appropriate for specifying real-world complex systems.

3 Model-Theoretic Characterization

An Evolving Algebra $\mathcal{E}$ with a program $\mathcal{R}$ defines a linear state space, covering the classical notion of (deterministic) algorithms. In the following, let $\mathfrak{R}$ denote the temporal successor relation in this state space: $\mathfrak{R}(\mathcal{A}, \mathcal{B}) \Leftrightarrow \mathcal{B} = \mathcal{R}(\mathcal{A})$. Let $\mathfrak{R}^*$ denote the transitive closure of $\mathfrak{R}$.

A set of updates can also be seen as a partial static algebra over Σ. Then, parallel execution of sets of updates corresponds to taking the union of partial algebras, and application of a set of updates corresponds to overwriting a static algebra with a partial static algebra.

Definition 6 *Let $\mathcal{A}$ be a static algebra and r a ground rule. Then the partial interpretation $r^{part}(\mathcal{A})$ is defined as*

$$((r^{part}(\mathcal{A}))(f))(s_1,\ldots,s_n) := \begin{cases} s & \text{if } (f,s_1,\ldots,s_n,s) \in \mathsf{Upd}(r,\mathcal{A}), \\ undef & \text{otherwise} . \end{cases}$$

Definition 7 *Let $\mathcal{A}$ be a static algebra. For a set $\mathcal{R}$ of rules,* $\mathsf{write}(\mathcal{R},\mathcal{A})$ *denotes the set of locations which are updated:*

- *If $r \equiv f(t_1,\ldots,t_n) := t_0$, then* $\mathsf{write}(r,\mathcal{A}) := \{(f,\mathcal{A}(t_1),\ldots,\mathcal{A}(t_n))\}$.
- *If $r \equiv r_1,\ldots,r_n$ is a block, then* $\mathsf{write}(r,\mathcal{A}) := \mathsf{write}(r_1,\mathcal{A}) \cup \ldots \cup \mathsf{write}(r_n,\mathcal{A})$.
- *If $r \equiv$* $\mathsf{if}\ g_1\ \mathsf{then}\ r_1\ \mathsf{elsif}\ g_2\ \mathsf{then}\ r_2\ \ldots\ \mathsf{elsif}\ g_k\ \mathsf{then}\ r_k\ \mathsf{else}\ r_{k+1}\ \mathsf{endif}$, $\mathcal{A} \models g_i$ *and* $\mathcal{A} \not\models g_j$ *for all $j < i$, then* $\mathsf{write}(r,\mathcal{A}) := \mathsf{write}(r_i,\mathcal{A})$. *If $\mathcal{A} \not\models g_j$ for all j, then* $\mathsf{write}(r,\mathcal{A}) := \mathsf{write}(r_{k+1},\mathcal{A})$.

Definition 8 *Every superuniverse $\mathcal{S}$ can be seen as the flat lattice constructed from $\mathcal{S}$ by adding an element $\top$, and the definitions* $\mathsf{undef} < s < \top$ *for all* $\mathsf{undef} \neq s \in \mathcal{S}$. *The operator $\sqcup$ denotes the least upper bound of two elements of this lattice, ie $\sqcup(s_1,s_2) = \top \Leftrightarrow (s_1,s_2 \neq \mathsf{undef} \wedge s_1 \neq s_2)$.*

For two (partial) static algebras $\mathcal{A} = (A,\mathcal{S})$ and $\mathcal{A}' = (A',\mathcal{S}')$ over signatures Σ resp. Σ', their union *$\mathcal{B} = (B,\mathcal{S}\cup\mathcal{S}') := \mathcal{A} \cup \mathcal{A}'$ over $\Sigma \cup \Sigma'$ is defined as*

$$(B(f))(s_1,\ldots,s_{\mathsf{ord}(f)}) := \sqcup((A(f))(s_1,\ldots,s_{\mathsf{ord}(f)}),(A'(f))(s_1,\ldots,s_{\mathsf{ord}(f)})) .$$

For a static algebra $\mathcal{A} = (A,\mathcal{S})$ over Σ and a set L of locations over Σ and $\mathcal{S}$, the restriction of $\mathcal{A}$ to L, *$\mathcal{A}|_L = (A|_L,\mathcal{S})$, is defined as*

$$(A|_L\ (f))(s_1,\ldots,s_n) := \begin{cases} (A(f))(s_1,\ldots,s_n) & \text{if } (f;s_1,\ldots,s_n) \in L , \\ undef & \text{otherwise} . \end{cases}$$

For two static algebras $\mathcal{A} = (A,\mathcal{S})$ and $\mathcal{A}' = (A',\mathcal{S}')$ over signatures Σ resp. Σ' and a set L of locations over Σ' and $\mathcal{S}'$, the superposition *$\mathcal{B} = (B,\mathcal{S}\cup\mathcal{S}') = \mathcal{A} \uplus_L \mathcal{A}'$ of $\mathcal{A}$ with $\mathcal{A}'$ on L is defined as*

$$(B(f))(s_1,\ldots,s_n) := \begin{cases} (A'(f))(s_1,\ldots,s_n) & \text{if } (f;s_1,\ldots,s_n) \in L , \\ (A(f))(s_1,\ldots,s_n) & \text{otherwise} . \end{cases}$$

As a shorthand, $\mathcal{A} \uplus \mathcal{A}'$ stands for $\mathcal{A} \uplus_{\mathsf{dom}(\mathcal{A}')} \mathcal{A}'$.

The difference $\mathsf{diffs}(\mathcal{A},\mathcal{A}')$ *between two static algebras is a set of locations:*

$$\mathsf{diffs}(\mathcal{A},\mathcal{A}') := \{(f;s_1,\ldots,s_{\mathsf{ord}(f)}) \mid (A(f))(s_1,\ldots,s_{\mathsf{ord}(f)}) \neq (A'(f))(s_1,\ldots,s_{\mathsf{ord}(f)})\}$$

Lemma 1 *For a static algebra $\mathcal{A}$ and a ground rule r,*

$$r^{part}(\mathcal{A}) = (r(\mathcal{A})|_{\mathsf{write}(r,\mathcal{A})}) \quad .$$

Theorem 1 *Let $\mathcal{A}$ a static algebra, and $\mathcal{R}$ a set of rules. Then*

a) $\mathcal{R}$ is consistently applicable in $\mathcal{A}$ iff

$$\mathcal{A}' := \bigcup_{r\in\mathcal{R}} (r(\mathcal{A})|_{\mathsf{write}(r,\mathcal{A})}) = \bigcup_{r\in\mathcal{R}} r^{part}(\mathcal{A})$$

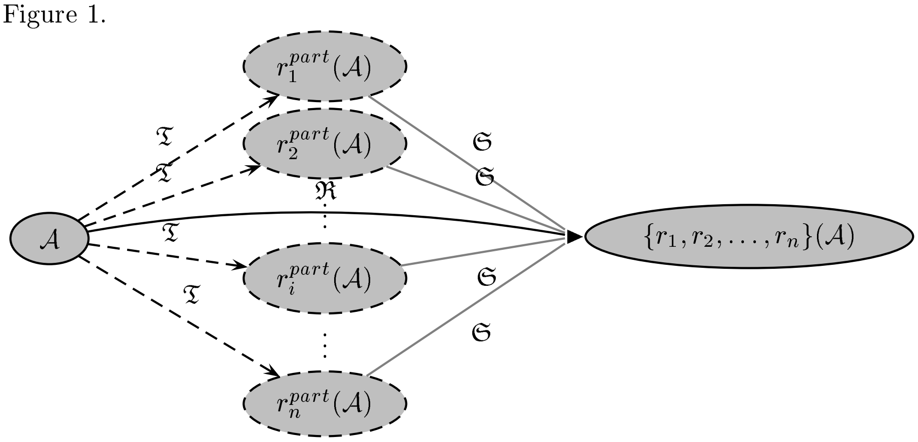
is consistent (ie no locations are evaluated to $\top$).

b) If $\mathcal{R}$ is consistently applicable in $\mathcal{A}$, then

$$\begin{aligned}\mathcal{R}(\mathcal{A}) &= \mathcal{A} \uplus_{(\bigcup_{r\in\mathcal{R}} \mathsf{write}(r,\mathcal{A}))} \bigcup_{r\in\mathcal{R}} (r(\mathcal{A})|_{\mathsf{write}(r,\mathcal{A})}) \\ &= \mathcal{A} \uplus \bigcup_{r\in\mathcal{R}} r^{part}(\mathcal{A})\ .\end{aligned}$$

3.1 Integration of Partial Interpretations into the State Space

The partial static algebras obtained by application of a single ground rule to a static algebra are integrated as auxiliary nodes into the state space as shown in Figure 1.



where ► represents the computation of the union and superposing it to the previous static algebra: the intermediate partial algebras are joined, and the gaps are filled with the values of the (total) algebra representing the previous state.

Fig. 1. Structure with Auxiliary Partial Interpretations

The additional accessibility relations in the augmented structure have the following semantics:

$\xrightarrow{\mathfrak{T}}$: Labeled elementary transition relation to the partial algebras which are obtained by execution of a single rule: $\mathfrak{T}(\mathcal{A}, r, \mathcal{B})$ iff $\mathcal{B} = r^{part}(\mathcal{A})$.

$\xrightarrow{\mathfrak{S}}$: Evaluation of partial static algebras: $\mathfrak{S}(\mathcal{C}, \mathcal{B})$ if $\mathcal{B}$ reads the partial algebra $\mathcal{C}$ as a result of the application of some rule (then, $\mathcal{C} \subset \mathcal{B}$).

Definition 9 *The accessibility relations of an augmented structure are* consistent *if for every $\mathcal{B}, \mathcal{C}$, $\mathfrak{S}(\mathcal{C}, \mathcal{B}) \Leftrightarrow \exists \mathcal{A} : (\mathfrak{R}(\mathcal{A}, \mathcal{B}) \wedge \exists r : \mathfrak{T}(\mathcal{A}, r, \mathcal{C}))$, ie exactly those static algebras which are computed by an application of some rule are accepted as a result.*

Theorem 2 *Thus, if the accessibility relations are consistent, for all $\mathcal{A}, \mathcal{B}$,*

$$\mathfrak{R}(\mathcal{A}, \mathcal{B}) \Leftrightarrow \mathcal{B} = \mathcal{A} \uplus \bigcup \{\mathcal{C} \mid \mathfrak{T}(\mathcal{A}, r, \mathcal{C})\} \quad \textit{and} \quad \mathfrak{R}(\mathcal{A}, \mathcal{B}) \Rightarrow \mathcal{B} = \mathcal{R}(\mathcal{A})\ .$$

This semantics can be axiomatized by a non-monotonic consequence relation as follows: $\mid\!\sim$ is used as an auxiliary relation which represents inheritable information, whereas $\vdash$ represents the non-monotonic consequence relation. In the following, let f be an n-ary function symbol and $s, s_1, \ldots, s_n$ elements of the superuniverse.

$$\frac{\mathfrak{T}(\mathcal{A}, r, \mathcal{C}) \quad , \quad r \text{ applied to } \mathcal{A} \text{ modifies } (f, s_1, \ldots, s_n) \text{ to } s}{\mathcal{C} \vdash f(s_1, \ldots, s_n) = s}$$

$$\frac{\mathfrak{R}(\mathcal{A}, \mathcal{B}) \quad , \quad \mathcal{A} \vdash f(s_1, \ldots, s_n) = s}{\mathcal{B} \mid\!\sim f(s_1, \ldots, s_n) = s}$$

$$\frac{\mathfrak{S}(\mathcal{C}, \mathcal{B}) \quad , \quad \mathcal{C} \vdash f(s_1, \ldots, s_n) = s}{\mathcal{B} \vdash f(s_1, \ldots, s_n) = s}$$

$$\frac{\mathcal{A} \mid\!\sim f(s_1, \ldots, s_n) = s \text{ , not } \mathcal{A} \vdash f(s_1, \ldots, s_n) = v \neq s}{\mathcal{A} \vdash f(s_1, \ldots, s_n) = s}$$

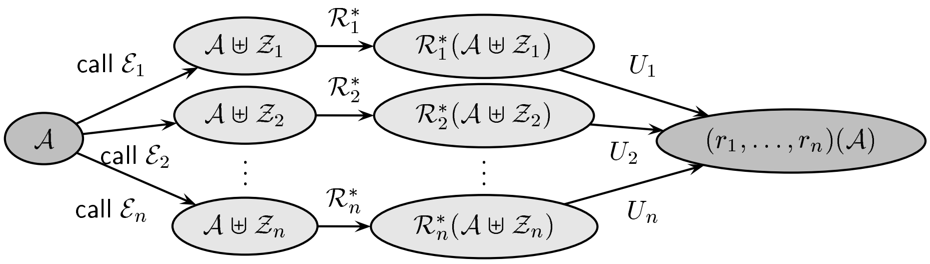
For subsequent steps, where a hierarchically structured state space is introduced, it is preferable to work with static algebras with two qualities of truth instead of partial algebras. In the auxiliary states, is has to be distinguished between "safe" knowledge derived by the updates and frame knowledge taken over from the state where the rule is applied. The relation $\vdash$, defined by the above inference system describes the "safe" knowledge of the states. A second truth relation, $\models \supset \vdash$, then gives the state "as is", including frame knowledge: In the auxiliary states, – in Fig. 1 those $\mathcal{C}$ such that there exists an $\mathcal{A}$ such that $\mathfrak{T}(\mathcal{A}, \_, \mathcal{C})$ – $\models_{\mathcal{C}} := \models_{\mathcal{A}} \uplus \vdash_{\mathcal{C}}$. In the main states, $\models\; = \;\vdash$. For initial states, $\vdash_{\mathcal{Z}} := \;\models\!\restriction_{\mathsf{dom}(\mathcal{Z})}$. With these definitions, there is a homogenous state space where $\models$ is total in every state and $\vdash$ is partial; $\models$ corresponds to the notion of "model" whereas $\vdash$ corresponds to derivability wrt. the current subsystem.

4 Structured Evolving Algebras: Complex Computations Instead of Elementary Rules

Instead of computing the auxiliary states by applying a single rule in a state, they can be computed by a complex computation, ie by running an Evolving Algebra on this state: If $\mathcal{E}$ is an Evolving Algebra, then

$$\mathsf{if}\ g\ \mathsf{then\ call}\ \mathcal{E}$$

is a rule. Rules like this are applied by initializing $\mathcal{E}$ and running $\mathcal{R}(\mathcal{E})$ until a fixpoint is reached. The accumulated net-updates of these subcomputations are given back as one aggregated update as shown in Figure 2. Communication in both directions is done via locations. Logically, a hierarchical structure as shown in Figure 3 is obtained. There is an additional accessibility relation $\mathfrak{Q}$ representing the call of another Evolving Algebra. The substructures (represented by shaded boxes) are not necessarily isolated but can have several states in common. Thus the whole structure can also be seen as a homogenous state space with several accessibility relations $\mathfrak{Q}, \mathfrak{S}, \mathfrak{T}, \mathfrak{R}_1, \mathfrak{R}_2, \ldots$, where each of the accessibility relations is deterministic (but, in general there are $\mathfrak{R}_1(\mathcal{A}, \mathcal{B})$ and $\mathfrak{R}_2(\mathcal{A}, \mathcal{C})$ with $\mathcal{B} \neq \mathcal{C}$).



$\mathcal{E}_i = (\mathcal{Z}_i, \mathcal{R}_i)$; $U_i =$ set of net updates which are executed by running $\mathcal{E}_i$ on $\mathcal{A}$.

Fig. 2. Operational Concept of Hierarchical Evolving Algebras

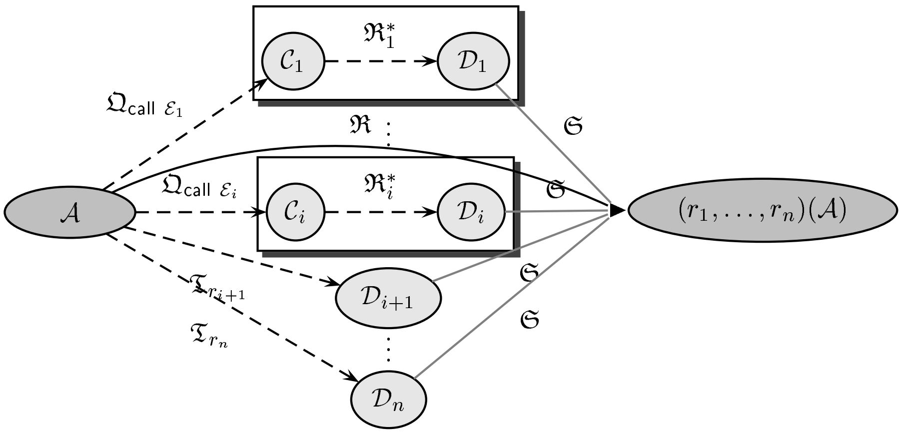


Fig. 3. Hierarchically Structured State Space

Definition 10 *Analogous to Def. 9, there is the following requirement: The accessibility relations of a hierarchical structure are* consistent *if for every $\mathcal{B}, \mathcal{D}$,*

$$\mathfrak{S}(\mathcal{D},\mathcal{B}) \;\Leftrightarrow\; \exists \mathcal{A} : (\,\mathfrak{R}(\mathcal{A},\mathcal{B}) \wedge \exists r : (\mathfrak{T}(\mathcal{A},r,\mathcal{D}) \vee \exists \mathcal{E}_i, \mathcal{C} : (\mathfrak{Q}(\mathcal{A}, \mathsf{call}\ \mathcal{E}_i, \mathcal{C}) \wedge \mathfrak{R}_i^*(\mathcal{D},\mathcal{C}) \wedge \neg\exists \mathcal{X} : \mathfrak{R}_i(\mathcal{C},\mathcal{X})))) \;.$$

The axiomatization is also done extending the ideas of Section 3.1. Apart from the two truth relations $\vdash$ and $\models$, two auxiliary relations $\mathrel{\vdash\!\!\sim}$ and $\approx$ for inheriting $\vdash$- resp. $\models$-information are used:

$$\vdash \;=\; \mathrel{\vdash\!\!\sim} \uplus \text{Updates} \quad , \quad \models \;=\; \approx \uplus \mathrel{\vdash\!\!\sim} \uplus \text{Updates}$$

$$\frac{\mathfrak{R}_i(\mathcal{A},\mathcal{B}) \quad , \quad \mathcal{A} \models f(s_1,\ldots,s_n) = s}{\mathcal{B} \approx f(s_1,\ldots,s_n) = s} \quad , \quad \frac{\mathfrak{R}_i(\mathcal{A},\mathcal{B}) \quad , \quad \mathcal{A} \vdash f(s_1,\ldots,s_n) = s}{\mathcal{B} \mathrel{\vdash\!\!\sim} f(s_1,\ldots,s_n) = s}$$

$$\frac{\mathfrak{Q}(\mathcal{A},r,\mathcal{C}) \;,\; \mathcal{A} \models f(s_1,\ldots,s_n) = s}{\mathcal{C} \approx f(s_1,\ldots,s_n) = s}$$

$$\frac{\mathfrak{Q}(\mathcal{A}, \mathsf{call}\ \mathcal{E}, \mathcal{C}) \;,\; \mathcal{Z}(\mathcal{E}) \vdash f(s_1,\ldots,s_n) = s \neq \mathit{undef}}{\mathcal{C} \vdash f(s_1,\ldots,s_n) = s}$$

$$\frac{\mathfrak{T}(\mathcal{A},r,\mathcal{C}) \quad , \quad \mathcal{A} \models f(s_1,\ldots,s_n) = s}{\mathcal{C} \approx f(s_1,\ldots,s_n) = s}$$

$$\frac{\mathfrak{T}(\mathcal{A},r,\mathcal{C}) \quad , \quad r \text{ applied to } \mathcal{A} \text{ updates } (f,s_1,\ldots,s_n) \text{ to } s}{\mathcal{C} \vdash f(s_1,\ldots,s_n) = s}$$

$$\frac{\mathfrak{S}(\mathcal{D},\mathcal{B}) \quad , \quad \mathcal{D} \vdash f(s_1,\ldots,s_n) = s}{\mathcal{B} \vdash f(s_1,\ldots,s_n) = s} \quad , \quad \frac{\mathcal{B} \vdash f(s_1,\ldots,s_n) = s}{\mathcal{B} \models f(s_1,\ldots,s_n) = s}$$

$$\frac{\mathcal{A} \mathrel{|\!\sim} f(s_1,\ldots,s_n) = s \ , \ \text{not } \mathcal{A} \vdash f(s_1,\ldots,s_n) = t \neq s}{\mathcal{A} \vdash f(s_1,\ldots,s_n) = s}$$

$$\frac{\mathcal{A} \approx f(s_1,\ldots,s_n) = s \ , \ \text{not } \mathcal{A} \vdash f(s_1,\ldots,s_n) = t \neq s}{\mathcal{A} \models f(s_1,\ldots,s_n) = s}$$

Based on this concept, arbitrary possibilities for structuring programs and computations can be provided: A static algebra can be handed over at certain situations to another set of rules.

The *encapsulated parallel* composition has already been introduced in Figure 2: Every visible atomic transition is a complete execution of an Evolving Algebra.

Also, a *joined parallel* composition of Evolving Algebras can be defined. The initialization of a joined parallel system is done when starting the system by joining the initializations of all subsystems. The visible atomic transitions of a joined parallel system are $\mathcal{R}_1 \cup \mathcal{R}_2 \cup \ldots \cup \mathcal{R}_n$, the whole system behaves like $(\mathcal{Z}_1 \cup \ldots \cup \mathcal{Z}_n, \mathcal{R}_1 \cup \ldots \cup \mathcal{R}_n)$.

Also, Evolving Algebras can be executed sequentially:

$$\textsf{if } g \textsf{ then call } name_1 \ ; \ \textsf{call } name_2 \ ; \ldots ; \ \textsf{call } name_n$$

Since every Evolving Algebra is initialized and executed until it reaches a fixpoint and then the resulting state is given to the next Evolving Algebra, this is an *encapsulated sequential* composition (see Figure 4).

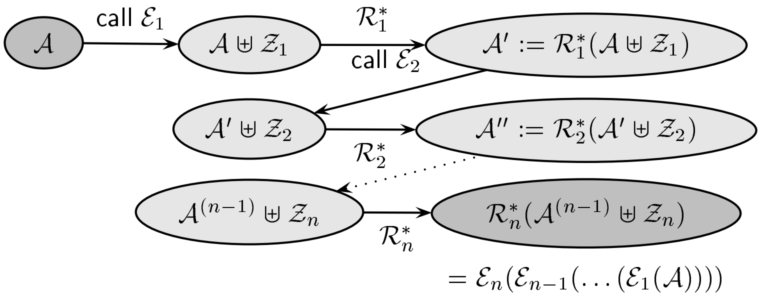


Fig. 4. Operational Concept of Sequential Composition

5 Systems of Evolving Algebras

Motivated by the above-mentioned possibilities for composing Evolving Algebras, a formal theory of complex systems of Evolving Algebras is developed. Regarding Evolving Algebras as atomic units, systems of Evolving Algebras are constructed by several operators. A system is given by a description of its initial state and the set of its visible transitions. For composing Evolving Algebras, also their initializations have to be considered:

Definition 11 *For a static algebra $\mathcal{A} = (A, \mathcal{S})$, $\mathsf{init}(\mathcal{A})$ is the rule*

$$\begin{aligned} &\mathsf{init}(\mathcal{A}) := \mathsf{if\ true\ then}\ u_1, \ldots, u_n \quad , \textit{ where} \\ &\{u_1, \ldots, u_n\} = \{\, f(s_1, \ldots, s_{\mathsf{ord}(f)}) := \mathcal{A}(f(s_1, \ldots, s_{\mathsf{ord}(f)})) : \\ &\qquad\qquad f \in \Sigma, s_1, \ldots, s_{\mathsf{ord}(f)} \in \mathcal{S}, \mathcal{A}(f(s_1, \ldots, s_{\mathsf{ord}(f)})) \neq \mathit{undef}\} \end{aligned}$$

is the set of updates which have to be executed to get $\mathcal{A}$ from the empty static algebra $\mathcal{O}$.

Corollary 1 *For two static algebras $\mathcal{A}$ and $\mathcal{A}'$, $\mathcal{A} \uplus \mathcal{A}' = (\mathsf{init}(\mathcal{A}'))(\mathcal{A})$ and $\mathcal{A} = (\mathsf{init}(\mathcal{A}))(\mathcal{O})$.*

Proof and explanation: With the notation introduced, $(\mathsf{init}(\mathcal{A}'))(\mathcal{A})$ is the state obtained by applying $\mathsf{init}(\mathcal{A}')$ in $\mathcal{A}$.

Definition 12 *In the first step, the set $\mathbb{R}$ of transition expressions, based on single rules, is defined:*
- *If r is a rule, then $\{r\} \in \mathbb{R}$.*
- *If $\mathcal{A}$ is a static algebra, then $\{\mathcal{A}\} \in \mathbb{R}$.*
- *If g is a boolean term and $\mathcal{R} \in \mathbb{R}$, then $\{\mathsf{if}\ g\ \mathsf{then}\ \mathcal{R}\} \in \mathbb{R}$.*
- *If $\mathcal{Q}, \mathcal{R} \in \mathbb{R}$, then $(\mathcal{Q} \cup \mathcal{R})$, $(\mathcal{Q} \circ \mathcal{R})$ and $(\mathcal{R}^*)$ are also elements of $\mathbb{R}$.*

In particular, the classical rule sets $\mathcal{R}$ are elements of $\mathbb{R}$.

Definition 13 *The semantics is given by the transitions induced by applying the elements of $\mathbb{R}$ to a static algebra: the elements of $\mathbb{R}$ define operators on static algebras which can be regarded as complex transitions.*

$$\begin{aligned} &\{r\}(\mathcal{A}) := r(\mathcal{A}) \qquad , \quad \{\mathcal{A}'\}(\mathcal{A}) := \mathcal{A} \uplus_{\mathsf{dom}\mathcal{A}'} \mathcal{A}' \\ &\{\mathsf{if}\ g\ \mathsf{then}\ \mathcal{R}\}(\mathcal{A}) := \begin{cases} \mathcal{R}(\mathcal{A}) & \textit{if } \mathcal{A} \models g \\ \mathcal{A} & \textit{otherwise .} \end{cases} \\ &(\mathcal{Q} \cup \mathcal{R})(\mathcal{A}) := \mathcal{A} \uplus (\mathcal{Q}(\mathcal{A}) \mid_{\mathsf{diffs}(\mathcal{A}, \mathcal{Q}(\mathcal{A}))} \cup \mathcal{R}(\mathcal{A}) \mid_{\mathsf{diffs}(\mathcal{A}, \mathcal{R}(\mathcal{A}))}) \\ &(\mathcal{Q} \circ \mathcal{R})(\mathcal{A}) := \mathcal{Q}(\mathcal{R}(\mathcal{A})) \\ &(\mathcal{R}^n)(\mathcal{A}) := (\mathcal{R} \circ \mathcal{R}^{n-1})(\mathcal{A}) \quad , \quad (\mathcal{R}^*)(\mathcal{A}) := \lim_{n \to \infty} (\mathcal{R}^n)(\mathcal{A}) \end{aligned}$$

The definition of $(\mathcal{Q} \cup \mathcal{R})(\mathcal{A})$ contains the notion of consistency of rule application: two transitions can be executed in parallel only if their updates are not conflicting.

Definition 14 *The set $\mathbb{E}$ of systems of Evolving Algebras is defined as follows:*
- *If $\mathcal{A}$ is a static algebra, then $\mathcal{A}$ is an expression in $\mathbb{E}$.*
- *If r is an Evolving Algebra rule, then $\{r\}$ is an expression in $\mathbb{E}$.*
- *If g is a boolean term and $\mathcal{E} \in \mathbb{E}$, then $\{\mathsf{if}\ g\ \mathsf{then}\ \mathcal{E}\}$ is an expression in $\mathbb{E}$.*
- *If $\mathcal{E}$ and $\mathcal{F}$ are expressions in $\mathbb{E}$, then $(\mathcal{E} \cup \mathcal{F})$, $(\mathcal{F} \circ \mathcal{E})$, $(\mathcal{F} \bullet \mathcal{E})$, $(\mathcal{F} \prec \mathcal{E})$, $(\mathcal{F} \lll \mathcal{E})$, $(\mathcal{E}^+)$, and $(\mathcal{E}^*)$ are expressions in $\mathbb{E}$.*

The underlying ideas are as follows:

$\mathcal{A}$, $\mathcal{R}$**:** Base cases.

$\{\mathsf{if}\ g\ \mathsf{then}\ \mathcal{E}\}$**:** if the guard g is satisfied in the current state, the execution of $\mathcal{E}$ is a visible action.

$\mathcal{E} \cup \mathcal{F}$**:** (union): The initialization results from the initializations of both subsystems. The system uses the rules from both systems.

$\mathcal{F} \circ \mathcal{E}$**:** (sequencing I): There is no initialization. Each visible action consists of initializing and executing $\mathcal{E}$ followed by initialization and executing $\mathcal{F}$.

$\mathcal{F} \bullet \mathcal{E}$**:** (prefixing $\mathcal{F}$ with $\mathcal{E}$): The second argument is completely added to the initialization. The initialization consists of executing $\mathcal{E}$ and initializing $\mathcal{F}$: $\mathcal{F} \bullet \mathcal{E} = \mathcal{F} \bullet \mathcal{E}^*$. The visible actions are the actions of $\mathcal{F}$.

! The construction $\mathcal{O} \bullet \mathcal{E}$ can be used to hide all activities of $\mathcal{E}$ and make only its final state $\mathcal{E}^\infty$ visible. It is often used when joining initializations of subsystems.

$\mathcal{F} \prec \mathcal{E}$**:** (alternation): The initialization consists of initializing both subsystems. Each visible action consists of one step of $\mathcal{E}$ followed by one step of $\mathcal{F}$.

$\mathcal{F} \prec\!\!\prec \mathcal{E}$**:** (sequencing II): The initialization consists of initializing both subsystems. Each visible action consists of first applying the rules of $\mathcal{E}$ until a fixpoint is reached, and then applying the rules of $\mathcal{F}$ until a fixpoint is reached.

$\mathcal{E}^+$**:** (external fixpoint): There is no initialization. Each visible action consists of initializing $\mathcal{E}$ and applying the rules of $\mathcal{E}$ until a fixpoint is reached.

$\mathcal{E}^*$**:** (internal fixpoint): The initialization consists of the initialization of $\mathcal{E}$. Each visible action consists of applying the rules of $\mathcal{E}$ until a fixpoint is reached.

Definition 15 *The formal semantics of expressions is defined in terms of operators $\mathcal{I} : \mathbb{E} \to \mathbb{E}$, $\mathcal{P} : \mathbb{E} \to \mathbb{R}$ and $\mathcal{Q} : \mathbb{E} \to \mathbb{R}$ – corresponding to an initialization and two expressions in $\mathbb{R}$ describing the behavior in parallel resp. sequential contexts. The definition is given in Figure 5. Two systems $\mathcal{E}$ and $\mathcal{F}$ are* equivalent, *$\mathcal{E} \equiv \mathcal{F}$, iff those three operators return the same results on them.*

$\mathcal{S}$	$\mathcal{I}(\mathcal{S})$	$\mathcal{P}(\mathcal{S})$	$\mathcal{Q}(\mathcal{S})$
$\mathcal{A}$	$\mathcal{O}$	$\mathsf{init}(\mathcal{A})$	$\mathsf{init}(\mathcal{A})$
$\{r\}$	$\mathcal{O}$	$\{r\}$	$\{r\}^*$
$\{\mathsf{if}\ g\ \mathsf{then}\ \mathcal{E}\}$	$\mathcal{O}$	$\{\mathsf{if}\ g\ \mathsf{then}\ \mathcal{Q}(\mathcal{E})\}$	$\{\mathsf{if}\ g\ \mathsf{then}\ \mathcal{Q}(\mathcal{E})\}^*$
$\mathcal{E} \cup \mathcal{F}$	$(\mathcal{O} \bullet \mathcal{I}(\mathcal{E})) \cup (\mathcal{O} \bullet \mathcal{I}(\mathcal{F}))$	$\mathcal{P}(\mathcal{E}) \cup \mathcal{P}(\mathcal{F})$	$(\mathcal{P}(\mathcal{E}) \cup \mathcal{P}(\mathcal{F}))^* \circ (\mathcal{Q}(\mathcal{I}(\mathcal{E})) \cup \mathcal{Q}(\mathcal{I}(\mathcal{F})))$
$\mathcal{F} \circ \mathcal{E}$	$\mathcal{O}$	$(\mathcal{P}(\mathcal{F}))^* \circ \mathcal{Q}(\mathcal{I}(\mathcal{F})) \circ (\mathcal{P}(\mathcal{E}))^* \circ \mathcal{Q}(\mathcal{I}(\mathcal{E}))$	$(\mathcal{P}(\mathcal{F}))^* \circ \mathcal{Q}(\mathcal{I}(\mathcal{F})) \circ (\mathcal{P}(\mathcal{E}))^* \circ \mathcal{Q}(\mathcal{I}(\mathcal{E})) = \mathcal{Q}(\mathcal{F}) \circ \mathcal{Q}(\mathcal{E})$
$\mathcal{F} \bullet \mathcal{E}$	$\mathcal{I}(\mathcal{F}) \circ \mathcal{E}$	$\mathcal{P}(\mathcal{F})$	"
$\mathcal{F} \prec \mathcal{E}$	$(\mathcal{O} \bullet \mathcal{I}(\mathcal{E})) \cup (\mathcal{O} \bullet \mathcal{I}(\mathcal{F}))$	$\mathcal{P}(\mathcal{F}) \circ \mathcal{P}(\mathcal{E})$	$(\mathcal{P}(\mathcal{F}) \circ \mathcal{P}(\mathcal{E}))^* \circ (\mathcal{Q}(\mathcal{I}(\mathcal{E})) \cup \mathcal{Q}(\mathcal{I}(\mathcal{F})))$
$\mathcal{F} \prec\!\!\prec \mathcal{E}$	$(\mathcal{O} \bullet \mathcal{I}(\mathcal{E})) \cup (\mathcal{O} \bullet \mathcal{I}(\mathcal{F}))$	$(\mathcal{P}(\mathcal{F}))^* \circ (\mathcal{P}(\mathcal{E}))^*$	$(\mathcal{P}(\mathcal{F}))^* \circ (\mathcal{P}(\mathcal{E}))^* \circ (\mathcal{Q}(\mathcal{I}(\mathcal{E})) \cup \mathcal{Q}(\mathcal{I}(\mathcal{F})))$
$\mathcal{E}^+$	$\mathcal{O}$	$(\mathcal{P}(\mathcal{E}))^* \circ \mathcal{Q}(\mathcal{I}(\mathcal{E}))$	$(\mathcal{P}(\mathcal{E}^+))^*$
$\mathcal{E}^*$	$\mathcal{I}(\mathcal{E})$	$(\mathcal{P}(\mathcal{E}))^*$	$(\mathcal{P}(\mathcal{E}))^* \circ \mathcal{Q}(\mathcal{I}(\mathcal{E}))$

Fig. 5. Semantics of System Expressions

Definition 16 *For a system $\mathcal{E}$, the final state is given by the static Algebra $\mathcal{E}^\infty := (\mathcal{Q}(\mathcal{E}))(\mathcal{O})$. If in case of the iteration operator, there is no fixpoint, the system defines an infinite computation (cf. server processes).*

The mappings $\mathcal{I}$, $\mathcal{P}$, and $\mathcal{Q}$ provide all information needed for composing expressions in a useful way:

$\mathcal{I}$: Reaching the initial configuration: $(\mathcal{I}(\mathcal{E}))^\infty$ is the initial state of $\mathcal{E}$.

$\mathcal{P}$: Description of actions (elementary rules or complex transitions; represented by an expression in $\mathbb{R}$) which can be executed atomically in this system.

$\mathcal{Q}$: The effect of running the system $\mathcal{E}$ in isolation on a given state: starting $\mathcal{E}$ in a state $\mathcal{A}$, it stops in $(\mathcal{Q}(\mathcal{E}))(\mathcal{A})$ or defines an infinite process.

Corollary 2 *An Evolving Algebra $\mathcal{E} = (\mathcal{Z}, \mathcal{R})$ is equivalent to the system $\mathcal{R} \bullet \mathcal{Z}$: $(\mathcal{Z}, \mathcal{R}) \equiv (\mathcal{O}, \mathcal{R}) \bullet (\mathcal{Z}, \emptyset) \equiv (\mathcal{O}, \mathcal{R}) \bullet (\mathcal{O}, \mathsf{init}(\mathcal{Z}))$.*

Proof. $\mathcal{I}(\mathcal{R} \bullet \mathcal{Z}) = \mathcal{I}(\mathcal{R}) \circ \mathcal{Z} = \mathcal{O} \circ \mathcal{Z} = \mathcal{Z}$, $\mathcal{P}(\mathcal{R} \bullet \mathcal{Z}) = \mathcal{P}(\mathcal{R}) = \mathcal{R}$, $\mathcal{Q}(\mathcal{R} \bullet \mathcal{Z}) = \mathcal{Q}(\mathcal{R}) \circ \mathcal{Q}(\mathcal{Z}) = \mathcal{R}^* \circ \mathsf{init}(\mathcal{Z})$, and $(\mathcal{R} \bullet \mathcal{Z})^\infty = (\mathcal{Q}(\mathcal{R} \bullet \mathcal{Z}))(\mathcal{O}) = (\mathcal{R}^* \circ \mathsf{init}(\mathcal{Z}))(\mathcal{O}) = \mathcal{R}^*(\mathsf{init}(\mathcal{Z})(\mathcal{O})) = \mathcal{R}^*(\mathcal{Z})$.

Corollary 3 *The operators possess the following algebraic properties:*

Commutativity: *The operation $\_ \cup \_$ is commutative.*

Idempotency: *The operations $\mathcal{S} \cup \_$, $\mathcal{O} \circ \_$, $\_ \circ \mathcal{O}$, $\mathcal{O} \bullet \_$, $\_ \bullet \mathcal{O}$, and $\_^*$ are idempotent.*

Implicite Closures: $\mathcal{F} \circ \mathcal{E} \equiv \mathcal{F}^* \circ \mathcal{E}^*$, $\quad \mathcal{F} \prec\!\!\prec \mathcal{E} \equiv \mathcal{F}^* \prec\!\!\prec \mathcal{E}^*$, $\quad \mathcal{F} \bullet \mathcal{E} \equiv \mathcal{F} \bullet \mathcal{E}^*$.

Associativity: *The binary operators $\cup$, $\circ$, $\bullet$, $\prec$, and $\prec\!\!\prec$ are associative.*

Special properties of $\mathcal{O} \bullet \_$: $\mathcal{I}(\mathcal{O} \bullet \mathcal{S}) \equiv \mathcal{S}$, $\quad \mathcal{P}(\mathcal{O} \bullet \mathcal{S}) \equiv \emptyset$, $\quad \mathcal{Q}(\mathcal{O} \bullet \mathcal{S}) \equiv \mathcal{Q}(\mathcal{S})$.

Proof. The only interesting proof is that of the associativity of $\cup$ wrt. $\mathcal{I}$, which also sheds light on the use of $\bullet$:
$\mathcal{I}(\mathcal{E} \cup (\mathcal{F} \cup \mathcal{G})) = \mathcal{O} \bullet \mathcal{I}(\mathcal{E}) \cup (\mathcal{O} \bullet \mathcal{I}(\mathcal{F} \cup \mathcal{G})) = \mathcal{O} \bullet \mathcal{I}(\mathcal{E}) \cup (\mathcal{O} \bullet (\mathcal{O} \bullet \mathcal{I}(\mathcal{F}) \cup \mathcal{O} \bullet \mathcal{I}(\mathcal{G})))$.
Due to the special properties of $\mathcal{O} \bullet \_$,
$\mathcal{I}(\mathcal{O} \bullet (\mathcal{O} \bullet \mathcal{I}(\mathcal{F}) \cup \mathcal{O} \bullet \mathcal{I}(\mathcal{G}))) = \mathcal{O} \bullet \mathcal{I}(\mathcal{F}) \cup \mathcal{O} \bullet \mathcal{I}(\mathcal{G})$, and
$\mathcal{I}(\mathcal{O} \bullet \mathcal{I}(\mathcal{F}) \cup \mathcal{O} \bullet \mathcal{I}(\mathcal{G})) = \mathcal{O} \bullet \mathcal{I}(\mathcal{O} \bullet \mathcal{I}(\mathcal{F})) \cup \mathcal{O} \bullet \mathcal{I}(\mathcal{O} \bullet \mathcal{I}(\mathcal{G})) = \mathcal{O} \bullet \mathcal{I}(\mathcal{F}) \cup \mathcal{O} \bullet \mathcal{I}(\mathcal{G})$,
$\mathcal{P}(\mathcal{O} \bullet (\mathcal{O} \bullet \mathcal{I}(\mathcal{F}) \cup \mathcal{O} \bullet \mathcal{I}(\mathcal{G}))) = \emptyset = \mathcal{P}(\mathcal{O} \bullet \mathcal{I}(\mathcal{F}) \cup \mathcal{O} \bullet \mathcal{I}(\mathcal{G}))$,
$\mathcal{Q}(\mathcal{O} \bullet (\mathcal{O} \bullet \mathcal{I}(\mathcal{F}) \cup \mathcal{O} \bullet \mathcal{I}(\mathcal{G}))) = \mathcal{Q}(\mathcal{O} \bullet \mathcal{I}(\mathcal{F}) \cup \mathcal{O} \bullet \mathcal{I}(\mathcal{G}))$,
thus $\mathcal{O} \bullet (\mathcal{O} \bullet \mathcal{I}(\mathcal{F}) \cup \mathcal{O} \bullet \mathcal{I}(\mathcal{G})) \equiv (\mathcal{O} \bullet \mathcal{I}(\mathcal{F})) \cup (\mathcal{O} \bullet \mathcal{I}(\mathcal{G}))$, and $\mathcal{I}(\mathcal{E} \cup (\mathcal{F} \cup \mathcal{G})) \equiv (\mathcal{O} \bullet \mathcal{I}(\mathcal{E})) \cup (\mathcal{O} \bullet \mathcal{I}(\mathcal{F})) \cup (\mathcal{O} \bullet \mathcal{I}(\mathcal{G})) \equiv \mathcal{I}((\mathcal{E} \cup \mathcal{F}) \cup \mathcal{G})$.

For an Evolving Algebra $\mathcal{E}$, $\mathcal{I}(\mathcal{E}^*) = \mathcal{I}(\mathcal{E})$, but $\mathcal{P}(\mathcal{E}^*) = (\mathcal{P}(\mathcal{E}))^* \neq \mathcal{P}(\mathcal{E})$. This difference is of importance when joining systems: in a join with $\mathcal{E}$, the rules of $\mathcal{E}$ are visible whereas in a join with $\mathcal{E}^*$ only the whole effect is visible.

Apart from the above-mentioned kind of union, where the rules of both systems can be applied, an *encapsulated union* can be defined as $\mathcal{E} \uplus \mathcal{F} := \mathcal{E}^* \cup \mathcal{F}^*$.

Example 2 *With this, the starting example can easily be reformulated as $C^* \cup O^* \cup CS^* \cup OS^* \cup CO^*$. The initialization consists of initializing all subsystems, the behaviour on a given state of each subsystem is aggregated to an atomic transition from the point of view of the main system.*

5.1 Operational Model

The operations of $\mathbb{E}$ are implemented by constructions of Evolving Algebras. Systems are different from a classic Evolving Algebra only in the point that their rules are not completely given explicitly in Pascal-style notation but can also be given implicitly by the behavior of other systems. As mentioned, an Evolving Algebra itself is a system which is completely described “by itself”.

In the following, it is shown how every system $\mathcal{S}$ can be described operationally by a system of Evolving Algebras mirroring the above ideas:

Rules:

Classical Evolving Algebra rules "if g then u": If g is satisfied in the current state, then execute the updates u.

Complex rules "if g then $\mathcal{E}$": Operations in a given state $\mathcal{A}$: if g is satisfied in $\mathcal{A}$, then copy $\mathcal{A}$ and execute $\mathcal{E}$ until it stops and obtain a new state $\mathcal{A}'$. Let M be the set of locations which are updated. Then the rule if g then $\mathsf{init}(\mathcal{A}' \mid_M)$ is a rule.

Systems:

For a state sequence $\mathcal{P}$ starting in a state $\mathcal{A}$ and ending in a state $\mathcal{A}'$, let $R(\mathcal{P})$ be the set of locations which are read before they are updated the first time, and $M(\mathcal{P})$ be the set of locations which are updated. Also, when standing as a guard, let a static algebra $\mathcal{A}$ denote the first-order formula

$\bigwedge f(s_1, \ldots, s_n) = t$**:** $f(s_1, \ldots, s_n) \in \mathsf{dom}(\mathcal{A})$ and $\mathcal{A}(f(s_1, \ldots, s_n)) = t \neq \mathit{undef}$.

$\mathcal{S} := \mathcal{E} = (\mathcal{Z}, \mathcal{R})$**:** Initialization: $\mathsf{init}(\mathcal{Z})$. Rules: $\mathcal{R}$.

$\mathcal{S} := \mathcal{E} \cup \mathcal{F}$**:** Initialization: perform the initializations of both subsystems and join the resulting states.
Rules: rules of both subsystems.

$\mathcal{S} := \mathcal{F} \circ \mathcal{E}$**:** According to the given semantics, $\mathcal{S} = \mathcal{F}^* \circ \mathcal{E}^*$ holds.
Initialization: empty.
For a state $\mathcal{A}$, let P be the state sequence starting with $\mathcal{A}$, performing the initialization for $\mathcal{E}$, applying the rules of $\mathcal{E}$ until a fixpoint is reached, then performing the initialization for $\mathcal{F}$ and applying the rules of $\mathcal{F}$ until a fixpoint $\mathcal{A}'$ is reached. Then if $\mathcal{A}|_{R(\mathcal{P})}$ then $\mathsf{init}(\mathcal{A}' \mid_{M(\mathcal{P})})$ is a rule of $\mathcal{S}$.

$\mathcal{S} := \mathcal{F} \bullet \mathcal{E}$**:** Initialization: compute $\mathcal{I}(\mathcal{F}) \circ \mathcal{E}$ by a subsystem. Rules: rules of $\mathcal{F}$.

$\mathcal{S} := \mathcal{F} \prec \mathcal{E}$**:** Initialization: compute $\mathcal{I}(\mathcal{E})$ and $\mathcal{I}(\mathcal{F})$ by subsystems and join the resulting static algebras.
For a state $\mathcal{A}$, let P be the state sequence starting with $\mathcal{A}$, doing one step with the rules of $\mathcal{E}$ and then doing one step with the rules of $\mathcal{F}$, reaching a state $\mathcal{A}'$. Then if $\mathcal{A}|_{R(\mathcal{P})}$ then $\mathsf{init}(\mathcal{A}' \mid_{M(\mathcal{P})})$ is a rule of $\mathcal{S}$.

$\mathcal{S} := \mathcal{F} \lessdot \mathcal{E}$**:** Initialization: compute $\mathcal{I}(\mathcal{E})$ and $\mathcal{I}(\mathcal{F})$ by subsystems and join the resulting static algebras.
For a state $\mathcal{A}$, let P be the state sequence starting with $\mathcal{A}$, applying the rules of $\mathcal{E}$ until a fixpoint is reached, then applying the rules of $\mathcal{F}$ until a fixpoint $\mathcal{A}'$ is reached. Then if $\mathcal{A}|_{R(\mathcal{P})}$ then $\mathsf{init}(\mathcal{A}' \mid_{M(\mathcal{P})})$ is a rule of $\mathcal{S}$.

$\mathcal{S} := \mathcal{E}^+$**:** Initialization: empty.
For a state $\mathcal{A}$, let P be the state sequence starting with $\mathcal{A}$, performing the initialization for $\mathcal{E}$ and applying the rules of $\mathcal{E}$ until a fixpoint $\mathcal{A}'$ is reached. Then if $\mathcal{A}|_{R(\mathcal{P})}$ then $\mathsf{init}(\mathcal{A}' \mid_{M(\mathcal{P})})$ is a rule of $\mathcal{S}$.

$\mathcal{S} := \mathcal{E}^*$**:** Initialization: initialize $\mathcal{E}$.
For a state $\mathcal{A}$, let P be the state sequence starting with $\mathcal{A}$ and applying the rules of $\mathcal{E}$ until a fixpoint $\mathcal{A}'$ is reached. Then if $\mathcal{A}|_{R(\mathcal{P})}$ then $\mathsf{init}(\mathcal{A}' \mid_{M(\mathcal{P})})$ is a rule of $\mathcal{S}$.

Thus, complex operations correspond to execution of subsystems starting with the current state (possibly performing their own initializations) and evolving by their own rules until a fixpoint is reached. Then the performed updates (or a part of this state) are returned as results.

6 Related Work and Conclusion

Fundamentally, in all specification methods there is a need for structuring. Especially methods with an operational flavor, such as Petri Nets, Rewriting Logic [Mes92] or in general rule-based systems take great advantage from features for encapsulating internal data structures and behaviour. In process algebraic frameworks, some structuring capabilities are provided by the term structure. Action refinement for the Pi-calculus [Mil91] is presented in [AH93]. General aspects of process refinement are dealt with in [DG91, DG95]. In [BK94], a concept for defining transactions as sequences of elementary actions in a logic-programming style is defined, parallelism is modeled by interleaving. In [AH96], an abstract framework for reactive modules is presented which permits parallel composition and abstraction from the internal behaviour of modules. There, the focus is on the observable behaviour of communication variables, the transitions performed by composed modules are given in a declarative style, similar to the transition oracle of [BK94].

In this paper, the focus is on dynamic, operational aspects providing as well a formal specification as an operational model. Although it is primarily formulated in Evolving Algebra terms, the ideas of this work can be transferred to other formal specification methods with an underlying state-oriented concept. A similar approach for the state-oriented deductive database language Statelog which also can be used for specification has been presented in [LML96].

From the software engineering point of view, the concept can further be extended with the usual modularization concepts of import, export, visible signature, renaming, and hiding (cf. [BHK90]).

The presented approach complements the method of refining Evolving Algebras by different abstraction levels [BR94]. There, the behaviour of rules performing complex changes on data structures in abstract terms is specified on a lower level in less abstract rules, and the finer specification is proven to be equivalent. For execution, the coarser rule system is *replaced* by the finer one. In contrast, in the hierarchical concept presented here, rules specifying a behaviour on a lower abstraction level are encapsulated as a system which is then *called* by the rules on the above level.

Another approach for composing Evolving Algebras for modeling concurrent computation has been presented in [GR93]. There, the focus is on joining Evolving Algebras with a shared signature to provide a communication mechanism. [BDR94] proposes another model for Occam, based on [GR93], also using shared variables for communication.

Also, in [GdL95], parallel execution of rules is formally examined, based on partial interpretations, using restriction, and overwriting.

Both approaches are concerned only with flat rule sets, thus no sequential composition, iteration, or hierarchical structuring is considered.

The paper adapts the Evolving Algebra concept for specifying complex systems in a modular style, also providing an abstract executable operational semantics for structured systems in general. The model-theoretic characterization also permits formal reasoning about such systems.

Acknowledgements.
The author thanks GEORG LAUSEN and BERTRAM LUDÄSCHER for many fruitful discussions and their help with improving the presentation of this paper.

References

[AH93] L. Aceto and M. Hennessy. Towards Action-Refinement in Process Algebras. *Information and Computation*, 103, 1993.

[AH96] R. Alur and T.A. Henzinger. Reactive Modules. *Proc. 11th Symp. on Logic in Computer Science (LICS)*, 1996

[BDR94] E. Börger, I. Durdanovic, and D. Rosenzweig. Occam: Specification and Compiler Correctness, Part I: The Primary Model, 1994.

[BHK90] J. Bergstra, J. Heering, and P. Klint. Module Algebra. *Journal of the ACM*, 37(2):335–372, 1990.

[BK94] A. J. Bonner and M. Kifer. An overview of transaction logic. *Theoretical Computer Science*, 133(2):205–265, 1994.

[BP95] B. Beckert and J. Posegga. leanEA: A Poor Man's Evolving Algebra Compiler. Technical Report 25, Universität Karlsruhe, Fak. f. Informatik, 1995.

[BR94] E. Börger and D. Rosenzweig. The WAM – Definition and Compiler Correctness. In *Logic Programming: Formal Methods and Practical Applications*, Ch. 1. North Holland, 1994.

[DG91] P. Degano and R. Gorrieri. Atomic refinement in Process Description languages. In *16th Symp. on Mathematical Foundations of Computer Science*, Springer LNCS 520, pp. 121-130, 1991.

[DG95] P. Degano and R. Gorrieri. A Causal Operational Semantics of Action Refinement. *Information and Computation*, 122(1):97–119, 1995.

[GdL95] R. Groenboom and G. R. de Lavalette. A Formalisation of Evolving Algebras. In *Proc. Accolade95*, pages 17–28, 1995.

[GH94] Y. Gurevich and J. Huggins. An Evolving Algebra Interpreter. WWW, 1994.

[GR93] P. Glavan and D. Rosenzweig. *Communicating evolving algebras.* Springer LNCS 702, pp. 182-215. Springer, 1993.

[Gur88] Y. Gurevich. *Logics and the challenge of Computer science*, pp. 1–57. In: Current Trends in Theoretical Computer Science, Comp. Sc. Press, 1988.

[Gur91] Y. Gurevich. An attempt to discover semantics (A Tutorial Introduction). *Bulletin of the EATCS*, 43:264–284, 1991.

[Gur94] Y. Gurevich. *Lipari Guide.* Oxford University Press, 1994.

[LML96] B. Ludäscher, W. May, and G. Lausen. Nested Transactions in a Logical Language for Active Rules. In *Proc. Intl. Workshop on Logic in Databases (LID)*, San Miniato, Italy, Springer LNCS 1154, pp. 197-222, 1996.

[Mes92] J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96:73–155, 1992.

[Mil91] R. Milner. The Polyadic π-calculus: A Tutorial. Technical Report 180, Computer Science Department, University of Edinburgh, 1991.