Modeling Rule-Based and Structured Systems with Evolving Algebras

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

Technical Report

June, 1996

Abstract

This report presents an approach for modeling rule-based and hierarchically structured systems with Evolving Algebras. Since Evolving Algebras are given by a set of rules, they are a quite natural formalism for describing rule-based systems. Concerning concurrent application of rules, some modifications to the "classical" Evolving Algebra concept presented in [Gur91] and [Gur94] are worked out. In this approach, rules of the Evolving Algebra correspond to action schemata in the application domain. In general, several rules representing different *alternative* actions will be applicable in an exclusive-or-style. Thus, in every step, a consistently applicable subset of rules is executed, introducing a new kind of nondeterminism into the Evolving Algebra concept. A model-theoretic characterization is presented, regarding the computations of an Evolving Algebra as a branching tree-like structure with additional auxiliary states and two qualities of truth, representing safe knowledge and frame knowledge. Based on this, a concept for hierarchically structured Evolving Algebras is developed, allowing atomic transitions being specified by complex computations performed by sub-Evolving-Algebras instead of single rules.

1 Introduction

Evolving Algebras were introduced in [Gur88], [Gur91] and [Gur94] to meet "the challenge of computer science" by providing a formal description of operational semantics for algorithms in an easy-to-understand way, tailored to the natural abstraction level of the algorithm. Most of the applications presented so far dealt with deterministic algorithms or programming languages.

In artificial intelligence, expert systems, and active databases, often rule-based, and by providing alternative actions, nondeterministic algorithms are employed. Since Evolving Algebras are given by a set of rules, they are a quite natural formalism to describe rule-based systems. Also, their clear and intuitive concept is well-suited for prototyping and testing changes of the rule set. E.g. they can be useful for simulating rule-based systems in the design and development phase.

Since the treatment of sets of rules which are applicable differs between deterministic (where all rules have to be applied) and nondeterministic (where some subset can be selected) algorithms, in this paper, the handling of concurrency when applying rules is tailored to the nondeterministic case.

The concept also directly leads to a concept for hierarchically structured systems of Evolving Algebras, further leading to a theory of composing Evolving Algebras.

The paper is structured as follows: In the next section, the classical concept of Evolving Algebras as presented in [Gur91] and [Gur94] is reviewed, casting a special glance on the treatment of concurrent application of rules. Section 3 states demands on parallelism for a concept which is suitable for rule-based systems and presents an intuitive solution which conforms with the basic ideas of Evolving Algebras. Apart from the intuitive description, a model-theoretic characterization is given, leading to a tree-like structure with auxiliary states. In Section 4, the concept is pursued further to a hierarchical structuring of Evolving Algebras: Atomic transitions can also be specified by sub-Evolving-Algebras instead of single rules. Section 5 formally defines the notion of systems of Evolving Algebras and gives an operational model for hierarchical Evolving Algebras as constructs of simple Evolving Algebras. In the Appendix, an operational model in the classical flat Evolving Algebra style is given, using additional control functions and changing active rule sets. Section 6 completes the work with an overview of related work and some concluding remarks.

2 Evolving Algebras

Evolving Algebras as introduced in [Gur88], [Gur91], and [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. The notions of *terms* resp. *closed 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. For every function symbol f, $\mathcal{A}$ defines a mapping $A(f) : \mathcal{S}^{\mathsf{ord}(f)} \to \mathcal{S}$. Thus, $\mathcal{A}$ can be extended straightforwardly to an evaluation of closed 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 $\mathcal{S}$. The set of static algebras over a fixed signature Σ is denoted by $\mathbb{A}_\Sigma$.
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 $\mathsf{Bool} := \{true, false\} \subset \mathcal{S}$. The only relation symbol in static algebras is the equality relation. By the interpretation

$$\mathcal{A}(a = b) = true \;\Leftrightarrow\; (\mathcal{A}(a), \mathcal{A}(b)) \in A(=) \;\Leftrightarrow\; \mathcal{A}(a) = \mathcal{A}(b) \;,$$

"=" can be understood as a function, making the expression $a = b$ a term. With the universe Bool, every relation can be represented by its characteristic function.
In a static algebra $\mathcal{A} = (A, \mathcal{S})$, for every function symbol, its *domain* is defined as

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

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 $\mathcal{Z}(\mathcal{E}) \in \mathbb{A}_\Sigma$ (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 interpretations of state-dependent function symbols in a Pascal-like syntax. Signature and superuniverse are constant over all states, so we can talk about 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:*

$$u : \quad f(t_1, \ldots, t_n) := t_0$$

where f is an n-ary function symbol and t_i are closed 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*

$$r = \begin{array}{l} \mathsf{if}\ g_1\ \mathsf{then}\ r_1 \\ \mathsf{elsif}\ g_2\ \mathsf{then}\ r_2 \\ \vdots \\ \mathsf{elsif}\ g_k\ \mathsf{then}\ r_k \\ \mathsf{else}\ r_{k+1} \\ \mathsf{endif} \end{array}$$

  is a rule.

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

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

Definition 4 *Let $P = \{r_1, \ldots, r_n\}$ be a program and $\mathcal{A}$ a static algebra. Then* $\mathsf{Updates}(P, \mathcal{A}) := \mathsf{Updates}(r_1, \mathcal{A}) \cup \ldots \cup \mathsf{Updates}(r_n, \mathcal{A})$.

Definition 5 *A set U of updates is consistent if for every location ℓ, $|\{s \mid (\ell, s) \in U\}| \leq 1$.*

Definition 6 *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*

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

In order to handle inconsistent sets of updates, there are different definitions:

[Gur91] (Tutorial): "in case of conflict at any location ℓ, choose the new value for ℓ nondeterministically among all candidate values. It could be called *local* nondeterminism."

This way seems to be critical wrt. the aspect of semantically associated updates since the nondeterminism is too much local.

[Gur94] (Lipari Guide), [CDG95]: "To fire an inconsistent update set means to do nothing."

According to Definition 4, in both [Gur91] and [Gur94], for a program P and a static algebra $\mathcal{A}$, the complete set $\mathsf{Updates}(P, \mathcal{A})$ is considered to be the *intended* set of updates. This is a very rigid demand: every executable rule has to be executed. For rule-based systems where possible rules often represent several alternatives, this demand will be weakened in the following section.

Following the definition of [Gur94], an Evolving Algebra defines a linear state space which covers the classical notion of deterministic algorithms.

Proposition 1 *For every rule r there is an equivalent sequence r' (ie for all $\mathcal{A}$,* $\mathsf{Updates}(r, \mathcal{A}) = \mathsf{Updates}(r', \mathcal{A})$) *of guarded updates of the form (cf. [BS90]):*

$$
\begin{aligned}
r' = \;& \mathsf{if}\ g_0\ \mathsf{then}\ U_0 \\
& \mathsf{elsif}\ g_1\ \mathsf{then}\ U_1 \\
& \vdots \\
& \mathsf{elsif}\ g_k\ \mathsf{then}\ U_k \\
& \mathsf{else}\ U_{k+1} \\
& \mathsf{endif}\quad ,
\end{aligned}
$$

where the U_i are sets of elementary updates *and the g_i are disjoint.*

Since the guards in the above normal form are disjoint, the rule can also be splitted into several rules, such that every rule is of the form

$$r:\ \ \mathsf{if}\ ?r\ \mathsf{then}\ !r$$

where $?r$ is a boolean expression and $!r$ is a sequence of updates.

If a rule r contains free variables – ie r is of the form $r(X_1, \ldots, X_n)$ – then it denotes the set $\{r(s_1, \ldots, s_n) \mid s_i \in \mathcal{S}\}$ of rules.

3 Rule-Based Systems and Parallelism

3.1 Conceptional Issues

When modeling real-world processes with (nondeterministic) rule-based systems, it often happens that the guards of several rules are satisfied which cause *contradicting* updates (see e.g. protocols, [Hug94]).

Scenarios like this – including the treatment in [Hug94] – are in discrepancy with [Gur91], [Gur94], and the Evolving Algebra Abstract Machine [CDG95]:

In those papers, all updates whose guards are satisfied have to be executed, only the treatment of inconsistent update sets differs as mentioned.

A remedy for dealing with rule-based systems is a *grained nondeterminism.* Its conceptual basis is provided by the Tutorial [Gur91] with the phrasing "imagine a demon who evaluates all relevant terms in a given state and then makes the necessary changes". For asynchronous distributed computations, it is proposed that "Instead of one demon, you may have a multitude of demons responsible for different rules. Whenever conditions are right, demons perform the required changes ...":

In [Gur94] (Lipari Guide), the concept of *agents* is set up: "The program of an agent may be any program described above". This allows distributed computations, but there is no direct communication or cooperation between agents if those have to work cooperatively for performing some piece of work.

Thus, a one-to-one assignment of demons (of the Evolving Algebra) to agents (of the application domain) seems inappropriate. Instead, agents of the application domain are modeled as elements of the superuniverse while the notion of "demon" is dedicated as an *abstract concept* to the individual rules of the Evolving Algebra (corresponding to action schemata of the application domain): Every demon "guides" the activities of several agents of the application domain. This is especially useful if in the application domain there are no agents but only reactions (cf. [BB92]). In terms of cognition, demons represent authorities which are *superordinate* to the individual agents and have access to the accumulated knowledge of *all* agents. In the context of rule-based systems, a demon is associated to every (instance of a) rule:

▷ Every rule has to be executed atomically in an "all-or-nothing"-style. If in some step several rules are applicable, ie their guards are satisfied, a consistent subset of them is selected for application. The successor state is then obtained by executing this set.

This definition already has been used in [May95].

By associating a demon with every rule, a set of rules corresponds to a team of demons: Every demon checks the applicability of his rule and computes the corresponding set of updates. In order to select a set of rules which is executable without conflicts – ie a consistent set of updates –, the demons sit at a round table and negotiate a result. Every demon realizes either all his updates or none of them as illustrated in the following example:

Example 1
Consider the following situation and updates:

$$\begin{array}{ll} \mathcal{A}: & f(1)=a,\ f(2)=b,\ f(3)=c \\ r_1: & f(1):=d \\ r_2: & f(2):=e, f(3):=f \\ r_3: & f(2):=g \end{array}$$

The "round table" interpretation is shown in Figure 1.

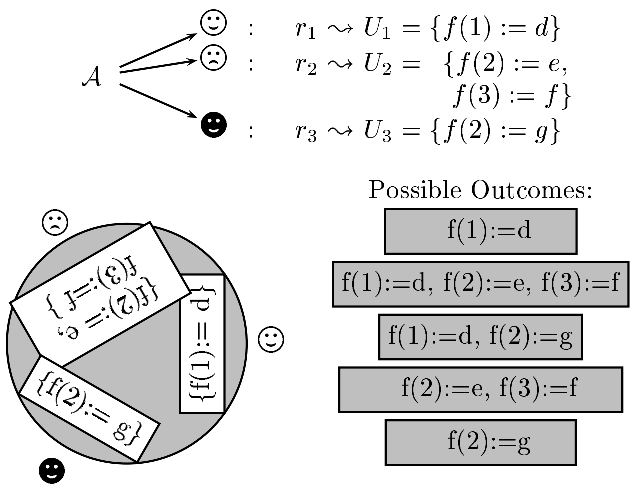


Figure 1: Round Table Negotiation

According to this, Definitions 3 and 4 are slightly modified:

Definition 7 *For a set $\mathcal{R} := \{r_1, \ldots, r_n\}$ of ground instances of rules selected for execution and a static algebra $\mathcal{A}$, define*

$$\mathsf{Updates}(\mathcal{R}, \mathcal{A}) := \mathsf{Updates}(r_1, \mathcal{A}) \cup \ldots \cup \mathsf{Updates}(r_n, \mathcal{A}) \ .$$

A subset $\mathcal{R}' \subseteq \mathcal{R}$ *of a rule set is called* consistently executable *in* $\mathcal{A}$ *if* $\mathsf{Updates}(\mathcal{R}', \mathcal{A})$ *is consistent. For a program* $P := \{r_1, \ldots, r_n\}$ *and a static algebra* $\mathcal{A}$,

$$\begin{array}{l}\mathsf{Poss}(P, \mathcal{A}) := \\ \{\mathcal{R}' \mid \mathcal{R}' \subseteq \mathsf{ground}(P) \textit{ and } \mathsf{Updates}(\mathcal{R}', \mathcal{A}) \textit{ is consistent}\}\end{array}$$

is the set of sets of rules which are consistently executable.

Figure 2 shows how possible update sets are constructed from update sets of single rules.

$$\begin{array}{ll}\mathcal{A}: & f(1) = a,\ f(2) = b,\ f(3) = c \\ r_1: & f(1) := d \\ r_2: & f(2) := e, f(3) := f \\ r_3: & f(2) := g\end{array}$$

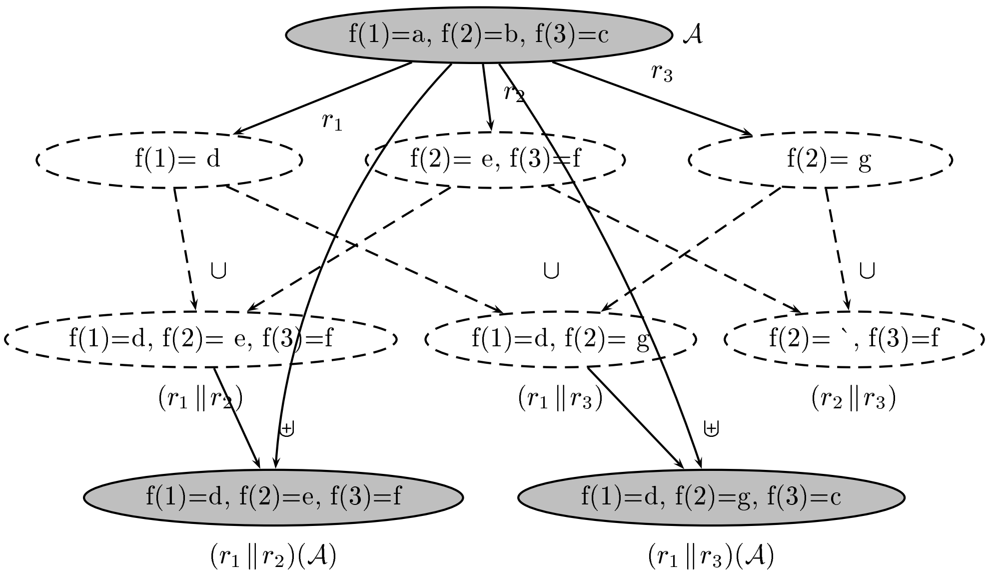


Figure 2: Model-Theoretic Construction of Update Sets

Thus, in general for a program P and a static algebra $\mathcal{A}$ there are several possible successor states. In this setting, an Evolving Algebra $\mathcal{E} = (\mathcal{Z}(\mathcal{E}), \mathcal{R}(\mathcal{E}))$ induces a branching tree structure (see Fig. 3).

Definition 8 *The static algebra which is obtained by applying a ground rule* $r \in \mathsf{ground}(\mathcal{R}(\mathcal{E}))$ *in a static algebra* $\mathcal{A}$ *is denoted by* $r(\mathcal{A})$. *If in a step several ground rules* $\{r_1, r_2, \ldots, r_n\} =: \mathcal{R} \in \mathsf{Poss}(\mathcal{R}(\mathcal{E}), \mathcal{A})$ *are applied, the resulting static algebra is denoted by* $\mathcal{R}(\mathcal{A})$.

In presence of nondeterministic rules, $r(\mathcal{A})$ can be set-valued. The following definitions are given for both cases.

Definition 9 *A sequence* $p = (\mathcal{A}_0 {=} \mathcal{Z}(\mathcal{E}), \mathcal{A}_1, \mathcal{A}_2, \ldots)$ *is a* computation *in* $\mathcal{E}$ *if for every* n *there is a finite set* $\mathcal{R}_n \in \mathsf{Poss}(\mathcal{R}(\mathcal{E}), \mathcal{A})$ *of (ground) rules with* $\mathcal{A}_n = \mathcal{R}_n(\mathcal{A}_{n-1})$ *(nondet.:* $\mathcal{A}_n \in \mathcal{R}_n(\mathcal{A}_{n-1})$*).*

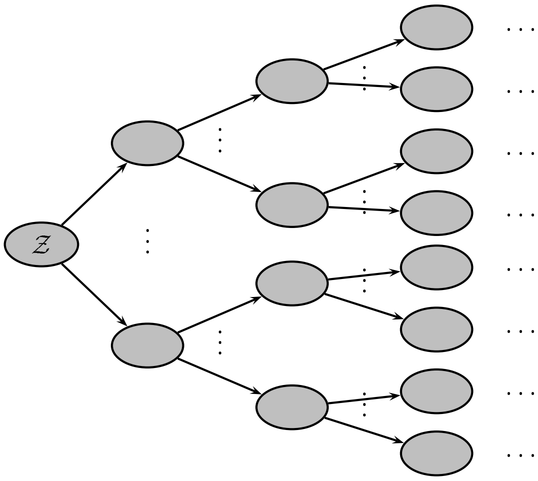


Figure 3: Branching Tree Structure of an Evolving Algebra

Definition 10 *The set $\mathcal{G}(\mathcal{E}) \subseteq \mathbb{A}_\Sigma$ of states reachable from $\mathcal{Z}(\mathcal{E})$ is given by*

$$\begin{array}{l}
\mathcal{G}^0_\mathcal{E} := \{\mathcal{Z}(\mathcal{E})\} \\
\mathcal{G}^{n+1}_\mathcal{E} := \{\mathcal{R}(\mathcal{A}) \mid \mathcal{R} \in \mathsf{Poss}(\mathcal{R}(\mathcal{E}), \mathcal{A}), \mathcal{A} \in \mathcal{G}^n_\mathcal{E}\} \quad \textit{resp.} \\
\mathcal{G}^{n+1}_\mathcal{E} := \bigcup_{\mathcal{R} \in \mathsf{Poss}(\mathcal{R}(\mathcal{E}),\mathcal{A}), \mathcal{A} \in \mathcal{G}^n_\mathcal{E}} \mathcal{R}(\mathcal{A}) \\
\mathcal{G}(\mathcal{E}) := \bigcup_{n \geq 0} \mathcal{G}^n_\mathcal{E} \quad .
\end{array}$$

The final states, ie states in which either no rule is applicable or application of any rule does not change anything, are:

$$\begin{array}{l}
\mathcal{G}^\omega(\mathcal{E}) := \{\mathcal{A} \in \mathcal{G}(\mathcal{E}) \mid \\
\quad \neg\exists \mathcal{A}' \neq \mathcal{A} \in \mathcal{G}(\mathcal{E}), r \in \mathsf{ground}(\mathcal{R}(\mathcal{E})) : \mathcal{A}' = r(\mathcal{A})\}
\end{array}$$

Definition 11 *An Evolving Algebra $\mathcal{E}$ is called* confluent *if $|\mathcal{G}^\omega(\mathcal{E})| = 1$.*

Deterministic processes are always modeled by confluent Evolving Algebras. In many cases, also nondeterministic processes result in confluent Evolving Algebras, such as eg. fault tolerant and fault-correcting protocols.

3.2 Model-Theoretic Characterization

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

$$\begin{array}{l}
((r^{part}(\mathcal{A}))(f))(s_1, \ldots, s_n) := \\
\quad \left\{ \begin{array}{ll} s & \textit{if } (f, s_1, \ldots, s_n, s) \in \mathsf{Updates}(r, \mathcal{A}), \\ \textit{undef} & \textit{otherwise} \ . \end{array} \right.
\end{array}$$

Definition 13 *Let $\mathcal{A}$ be a static algebra. For a set $\mathcal{R}$ of rules selected for execution in $\mathcal{A}$,* $\mathsf{write}(\mathcal{R}, \mathcal{A})$ *denotes the set of locations which are updated:*

- *If* $r \equiv f(t_1,\ldots,t_n) := t_0$ *is an elementary update, 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 14 *For a set $\mathcal{S}$ (universe), let $\mathcal{S}^+$ denote the lattice constructed from $\mathcal{S}$ by adding the elements $\bot$ and $\top$, and the definitions $\bot < s < \top$ for all $s \in \mathcal{S}$. The operator $\sqcup$ denotes the least upper bound of two elements of $\mathcal{S}^+$.*
For two 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*

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

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*

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

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*

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

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 defined as a set of locations:*

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

Lemma 1 *For a static algebra $\mathcal{A}$ and a ground rule r,*
$r^{part}(\mathcal{A}) = (r(\mathcal{A})|_{\mathsf{write}(r,\mathcal{A})})$.

Theorem 1 *Let $\mathcal{E}$ be an Evolving Algebra, $\mathcal{A}$ a static algebra, and $\mathcal{R}\subseteq\mathcal{R}(\mathcal{E})$ a set of ground rules. Then*

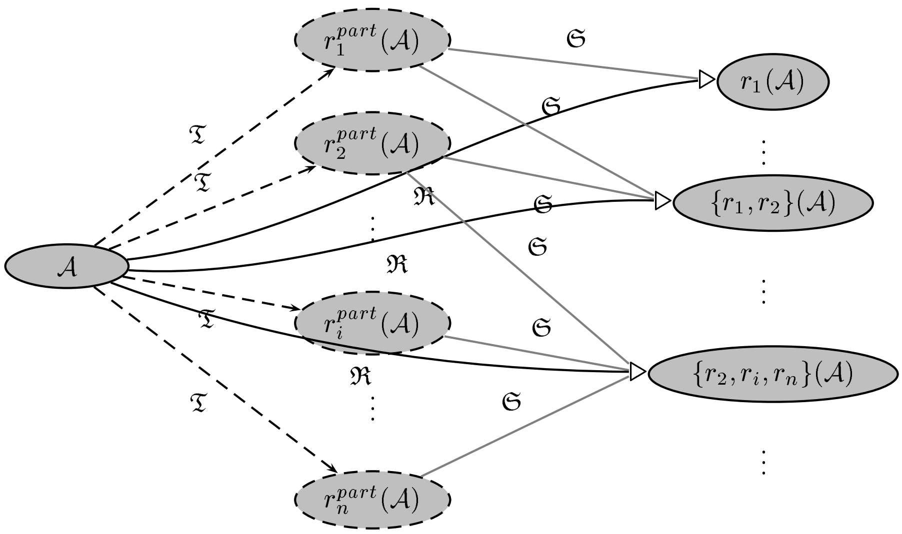
a) $\mathcal{R}$ is consistently applicable in $\mathcal{A}$ iff

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

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

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

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



$\triangleright$ represents the computation of the union and superposing it to the previous static algebra: the static algebras on the $\mathfrak{S}$-inputs are joined, and the gaps are filled with the values of the (total) algebra on the $\mathfrak{R}$-input.

Figure 4: Tree Structure with Auxiliary Partial Interpretations

3.3 Integration of Partial Interpretations into the Tree Structure

The partial static algebras obtained by application of a single ground rule in a static algebra are integrated as auxiliary nodes into the tree structure as shown in Figure 4.

The accessibility relations in the augmented tree structure are marked with some labels and have the following semantics:

$\xrightarrow{\mathfrak{T}}$: Elementary transition relation to the partial algebras which are obtained by execution of a single rule: $\mathfrak{T}(\mathcal{A}, r, \mathcal{B})$ if $\mathcal{B} = r^{part}(\mathcal{A})$ (thus, forall r: $(\exists \mathcal{B} : \mathfrak{T}(\mathcal{A}, r, \mathcal{B})) \;\Leftrightarrow\; \mathcal{A} \models ?r$)

$\xrightarrow{\mathfrak{R}}$: Temporal transition relation to successor states: $\mathfrak{R}(\mathcal{A}, \mathcal{R}, \mathcal{B})$ if $\mathcal{B} \in \mathcal{R}(\mathcal{A})$.

$\xrightarrow{\mathfrak{S}}$: Evaluation of partial static algebras: $\mathfrak{S}(\mathcal{C}, \mathcal{A}, r, \mathcal{B})$ if $\mathcal{B}$ uses the partial algebra $\mathcal{C}$ as result of the application of r in $\mathcal{A}$.

Definition 15 *The accessibility relations of a tree structure are consistent if*

1. $\mathfrak{S}(\mathcal{C}, \mathcal{A}, r, \_)$ *only if* $\mathfrak{T}(\mathcal{A}, r, \mathcal{C})$, *ie only static algebras which are computed by application of the corresponding rule are accepted as its result,*
2. *for every pair* $\mathcal{A}, \mathcal{B}$ *of states and rule* r, *there is at most one* $\mathcal{C}$ *such that* $\mathfrak{S}(\mathcal{C}, \mathcal{A}, r, \mathcal{B})$, *ie for every applied rule at most one result is evaluated,*
3. *for every* $\mathcal{A}, \mathcal{R}, \mathcal{B}$, $\mathfrak{R}(\mathcal{A}, \mathcal{R}, \mathcal{B}) \;\Leftrightarrow\; \mathcal{R} = \{r \mid \exists \mathcal{C} : \mathfrak{S}(\mathcal{C}, \mathcal{A}, r, \mathcal{B})\}$, *ie for exactly those rules which are applied in the corresponding transition, a result is evaluated.*

Definition 16 *The (interpretations of) states are* consistent *with the transition relations iff*

$$\mathfrak{R}(\mathcal{A}, \mathcal{R}, \mathcal{B}) \;\Leftrightarrow\; \mathcal{B} = \mathcal{A} \uplus \bigcup \{\mathcal{C} \mid \mathfrak{S}(\mathcal{C}, \mathcal{A}, r, \mathcal{B}) \wedge r \in \mathcal{R}\} \;.$$

Lemma 2 *If all rules are deterministic, this is equivalent to* $\mathfrak{R}(\mathcal{A}, \mathcal{R}, \mathcal{B}) \;\Leftrightarrow\; \mathcal{B} = \mathcal{A} \uplus \bigcup \{\mathcal{C} \mid \mathfrak{T}(\mathcal{A}, r, \mathcal{C}) \wedge r \in \mathcal{R}\}$.

Theorem 2 (Correctness) *If transition relations and states are consistent, then*

$$\mathfrak{R}(\mathcal{A},\mathcal{R},\mathcal{B}) \;\Rightarrow\; \mathcal{B} \in \mathcal{R}(\mathcal{A}) \;.$$

This semantics can be axiomatized by a non-monotonic consequence relation as follows: $\mathrel{|\!\sim}$ 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.

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

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

$$\frac{\begin{array}{c}\mathfrak{R}(\mathcal{A},R,\mathcal{B}) \quad , \quad r \in R \quad , \quad \mathfrak{S}(\mathcal{C},\mathcal{A},r,\mathcal{B}) \quad , \\ \mathcal{C} \vdash f(s_1, \ldots, s_n) = s\end{array}}{\mathcal{B} \vdash f(s_1, \ldots, s_n) = s}$$

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

For subsequent steps, 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 – ie exactly the states $\mathcal{B}$ such that there exists an $\mathcal{A}$ such that $\mathfrak{T}(\mathcal{A},\_,\mathcal{B})$ –, $\models_{\mathcal{B}} := \models_{\mathcal{A}} \uplus \vdash_{\mathcal{B}}$. In the main states, $\models\, = \,\vdash$. With these definitions, there is a homogenous state space where $\models$ is total in every state and $\vdash$ is partial.

4 Hierarchical 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:

$$r_i : \ \mathsf{if}\ G\ \mathsf{then\ call}\ \mathcal{R}_i$$

In this setting, the demons associated with the rule set of the currently active Evolving Algebra do not compute the updates by themselves, but give an order (ie the current state) to another team – corresponding to another rule set. This team returns a set of updates to the demon which gave the order, who then takes them to the round table as shown in Figure 5 where $\mathcal{R}, \mathcal{R}_1, \mathcal{R}_2, \ldots$ are sets of rules – resp. teams of demons.

Thus, a hierarchical structure as shown in Figure 6 is obtained. There is an additional accessibility relation $\mathfrak{Q}$ representing the call of another rule system. The substructures (represented by shaded boxes) are not necessarily isolated but can have several states in common. In general, they are internally branching. Thus the whole structure can also be seen as a homogenous state space with several temporal accessibility relations $\mathfrak{R}_1, \mathfrak{R}_2, \ldots$.

Definition 17
As a refinement of Definition 15 (1) there are the following requirements on the accessibility relations:

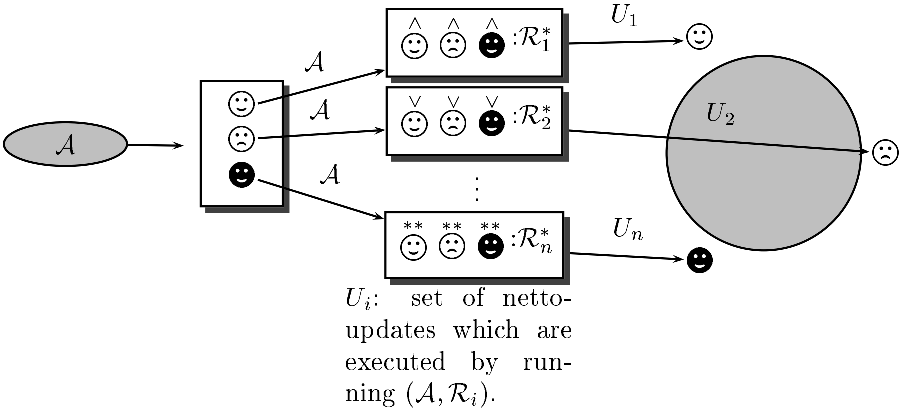


Figure 5: Operational Concept of Hierarchical Evolving Algebras

1a. for classical Evolving Algebra rules r:
$\mathfrak{S}(\mathcal{C}, \mathcal{A}, r, \_)$ *only if* $\mathfrak{T}(\mathcal{A}, r, \mathcal{C})$

1b. for calls of subsystems $r \equiv$ call $\mathcal{R}_i$ *where* $\mathcal{R}_i$ *is a set of rules:*
$\mathfrak{S}(\mathcal{D}, \mathcal{A}, r, \_)$ *only if there is a* $\mathcal{C}$ *such that* $\mathfrak{Q}(\mathcal{A}, r, \mathcal{C})$, $\mathfrak{R}_i^*(\mathcal{C}, \_, \mathcal{D})$ *and application of* $\mathcal{R}_i$ *does not cause any more changes to* $\mathcal{D}$, *ie* $\neg\exists \mathcal{R}' \subset \mathcal{R}_i, \mathcal{E} \neq \mathcal{D} : \mathfrak{R}(\mathcal{D}, \mathcal{R}', \mathcal{E})$.

The axiomatization is also done extending the ideas of Section 3.3. Apart from the two truth relations $\vdash$ and $\models$, two auxiliary relations $\mathrel{|\!\sim}$ and $\mathrel{|\!\bullet}$ for inheriting $\vdash$- or $\models$-information are used:

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

$$\frac{\mathfrak{R}(\mathcal{A}, \_, \mathcal{B}) \quad , \quad \mathcal{A} \models f(s_1, \ldots, s_n) = s}{\mathcal{B} \mathrel{|\!\bullet} f(s_1, \ldots, s_n) = s}$$

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

$$\frac{\mathfrak{Q}(\mathcal{A}, \_, \mathcal{B}) \quad , \quad \mathcal{A} \models f(s_1, \ldots, s_n) = s}{\mathcal{B} \mathrel{|\!\bullet} f(s_1, \ldots, s_n) = s}$$

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

$$\frac{\begin{array}{c}\mathfrak{R}(\mathcal{A}, R, \mathcal{B}) \quad , \quad r \in R \quad , \quad \mathfrak{S}(\mathcal{C}, \mathcal{A}, r, \mathcal{B}) \quad , \\ \mathcal{C} \vdash f(s_1, \ldots, s_n) = s\end{array}}{\mathcal{B} \vdash f(s_1, \ldots, s_n) = s}$$

$$\frac{\mathcal{A} \vdash f(s_1, \ldots, s_n) = s}{\mathcal{A} \models f(s_1, \ldots, s_n) = s}$$

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

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

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

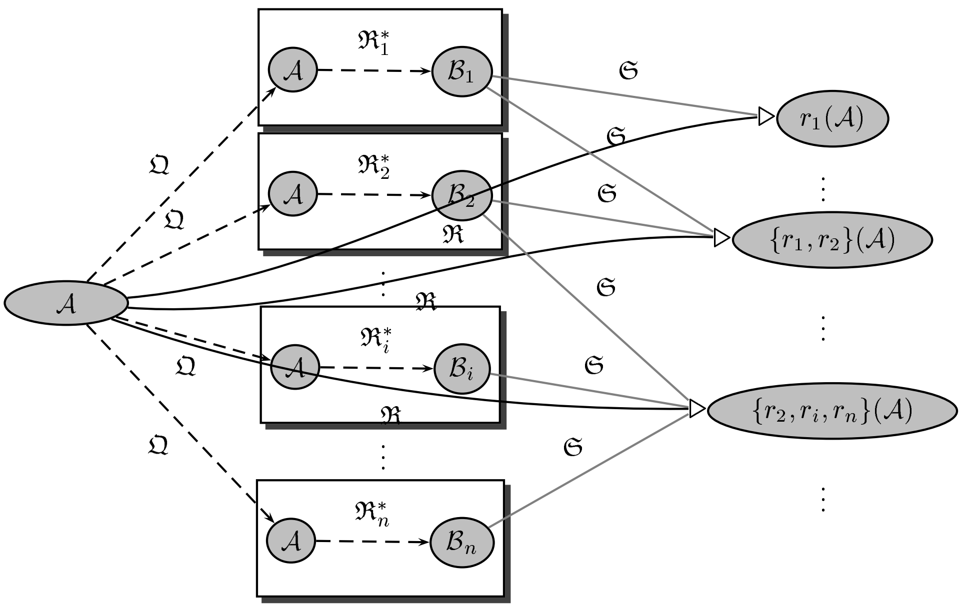


Figure 6: Hierarchically Structured State Space

The *encapsulated parallel* composition has already been introduced in Figure 5. The visible atomic transitions of a encapsulated parallel system are $\mathcal{R}_1^* \cup \mathcal{R}_2^* \cup \ldots \cup \mathcal{R}_n^*$.

Another parallel composition is obtained by taking the union of rule sets, corresponding to joining some teams to build a larger team. This leads to a *joined parallel* composition as shown in Figure 7. The visible atomic transitions of a joined parallel system are $\mathcal{R}_1 \cup \mathcal{R}_2 \cup \ldots \cup \mathcal{R}_n$.

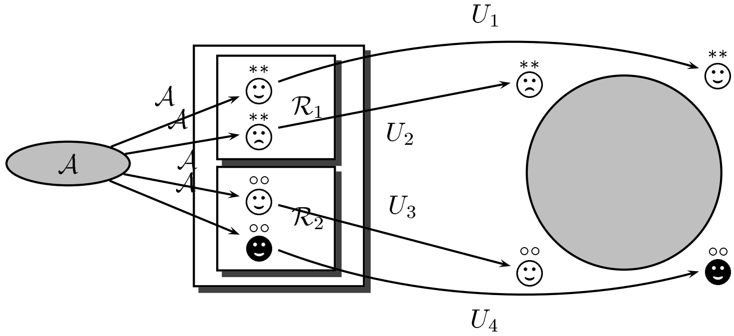


Figure 7: Joined Parallel Teams

Also, Evolving Algebras can be executed sequentially:

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

Since every Evolving Algebra is executed until it reaches a fixpoint before the resulting state is given to the next Evolving Algebra, this is an *encapsulated sequential* composition (see Figure 8). The visible atomic transitions of the system are $(\mathcal{R}_2^*) \circ (\mathcal{R}_1^*)$.

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

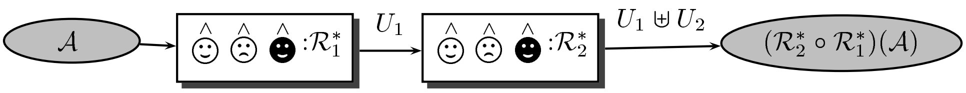


Figure 8: Sequential Composition of Teams

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.

Definition 18 *For a static algebra $\mathcal{A} = (A, \mathcal{S})$, $\mathsf{init}(\mathcal{A})$ is the rule*

$$\mathsf{init}(\mathcal{A}) := \mathsf{if\ true\ then}\ u_1, \dots, u_n$$

where

$$\begin{aligned}&\{u_1, \dots, u_n\} = \\ &\quad \{\, f(s_1, \dots, s_{\mathsf{ord}(f)}) := \mathcal{A}(f(s_1, \dots, s_{\mathsf{ord}(f)})) : \\ &\qquad f \in \Sigma, s_1, \dots, s_{\mathsf{ord}(f)} \in \mathcal{S}, \mathcal{A}(f(s_1, \dots, s_{\mathsf{ord}(f)})) \neq \mathit{undef}\}\end{aligned}$$

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

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 19 *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 20 *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.*

$$\begin{aligned}&\{r\}(\mathcal{A}) := \{r(\mathcal{A}), \mathcal{A}\} \\ &\mathcal{A}'(\mathcal{A}) := \{\mathcal{A} \uplus_{\mathsf{dom}\mathcal{A}'} \mathcal{A}'\} \\ &\{\mathsf{if}\ g\ \mathsf{then}\ \mathcal{R}\}(\mathcal{A}) := \begin{cases} \mathcal{A} & \textit{if } \mathcal{A} \not\models g\ , \\ \mathcal{A} \cup \mathcal{R}(\mathcal{A}) & \textit{otherwise}\ . \end{cases} \\ &(\mathcal{Q} \cup \mathcal{R})(\mathcal{A}) := \\ &\qquad \{\, \mathcal{A} \uplus_{\mathsf{diffs}(\mathcal{A},\mathcal{B}) \cup \mathsf{diffs}(\mathcal{A},\mathcal{B}')} (\mathcal{B}\ |_{\mathsf{diffs}(\mathcal{A},\mathcal{B})} \cup \mathcal{B}'\ |_{\mathsf{diffs}(\mathcal{A},\mathcal{B}')}) : \\ &\qquad\quad \mathcal{B} \in \mathcal{Q}(\mathcal{A}), \mathcal{B}' \in \mathcal{R}(\mathcal{A})\} \\ &(\mathcal{Q} \circ \mathcal{R})(\mathcal{A}) := \mathcal{Q}(\mathcal{R}(\mathcal{A})) = \bigcup\{\mathcal{Q}(\mathcal{B}) : \mathcal{B} \in \mathcal{R}(\mathcal{A})\} \\ &(\mathcal{R}^n)(\mathcal{A}) := (\mathcal{R} \circ \mathcal{R}^{n-1})(\mathcal{A}) \\ &(\mathcal{R}^*)(\mathcal{A}) := \{\mathcal{B} : \exists n : \mathcal{B} \in (\mathcal{R}^n)(\mathcal{A}) \wedge \mathcal{R}(\mathcal{B}) = \{\mathcal{B}\}\}\end{aligned}$$

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 21 *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 a rule (in the classical Evolving Algebra sense), then $\{r\}$ is an expression in $\mathbb{E}$.*
- *If g is a boolean term and $\mathcal{E} \in \mathbb{E}$, then $\{\textsf{if } g \textsf{ then } \mathcal{E}\}$ is an expression in $\mathbb{E}$.*
- *If $\mathcal{E}$ and $\mathcal{F}$ are expressions in $\mathbb{E}$, then $(\mathcal{E} \cup \mathcal{F})$, $(\mathcal{F} \circ \mathcal{E})$, $(\mathcal{F} \bullet \mathcal{E})$, $(\mathcal{F} \prec \mathcal{E})$, $(\mathcal{F} \lessdot \mathcal{E})$, $(\mathcal{E}^+)$, and $(\mathcal{E}^*)$ are expressions in $\mathbb{E}$.*

Definition 22 *The 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 9. Signatures and superuniverses are not dealt with.*

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

Figure 9: Semantics of System Expressions

Definition 23 *For a system $\mathcal{E}$, the set of final states is given by $\mathcal{E}^\infty := \mathcal{Q}(\mathcal{E})(\mathcal{O})$ which is a set of static algebras.*

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

- $\mathcal{I}$: Reaching an initial configuration for a system: $(\mathcal{I}(\mathcal{E}))^\infty$ is the set of initial states of $\mathcal{E}$.
- $\mathcal{P}$: Description of actions (elementary rules or complex transitions, which are represented by expressions in $\mathbb{R}$) which can be executed atomically in this system.
- $\mathcal{Q}$: The set of transitions the system $\mathcal{E}$ is able to perform when running in isolation on a given state: Starting $\mathcal{E}$ in a state $\mathcal{A}$, it stops in some state $\mathcal{A}' \in (\mathcal{Q}(\mathcal{E}))(\mathcal{A})$.

Corollary 2 *An Evolving Algebra $\mathcal{E} = (\mathcal{Z}, \mathcal{R})$ is represented by the system $\mathcal{R} \bullet \mathcal{Z}$.*

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* $(\mathcal{R} \bullet \mathcal{Z})^\infty = \mathcal{Q}(\mathcal{Q}(\mathcal{R} \bullet \mathcal{Z}))(\mathcal{O}) = (\mathcal{R}^* \circ \mathsf{init}(\mathcal{Z}))(\mathcal{O}) = \mathcal{R}^*(\mathsf{init}(\mathcal{Z})(\mathcal{O})) = \mathcal{R}^*(\mathcal{Z})$. ■

The underlying ideas are as follows:

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

$\{$if g then $\mathcal{E}\}$**:** Complex transitions: 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 of the system consists of initializing and executing $\mathcal{E}$ followed by initialization and executing $\mathcal{F}$.

$\mathcal{F} \bullet \mathcal{E}$**:** (sequencing II): The initialization consists of initializing and 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 $\mathcal{E}^\infty = \mathcal{I}(\mathcal{O} \bullet \mathcal{E})$ visible. In Figure 9 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 III): 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.

Corollary 3

$$
\begin{array}{lcl}
(\mathcal{Z}, \mathcal{R}) & = & (\mathcal{O}, \mathcal{R}) \bullet (\mathcal{Z}, \emptyset) = (\mathcal{O}, \mathcal{R}) \bullet (\mathcal{O}, \mathsf{init}(\mathcal{Z})) \\
(\mathcal{F} \bullet \mathcal{E}) \bullet \mathcal{G} & = & \mathcal{F} \bullet (\mathcal{E} \bullet \mathcal{G}) \\
(\mathcal{F} \circ \mathcal{E}) \circ \mathcal{G} & = & \mathcal{F} \circ (\mathcal{E} \circ \mathcal{G}) \\
(\mathcal{F} \cup \mathcal{E}) \cup \mathcal{G} & = & \mathcal{F} \cup (\mathcal{E} \cup \mathcal{G}) \\
\mathcal{F} \circ \mathcal{E} & = & \mathcal{F}^* \circ \mathcal{E}^* \\
\mathcal{F} \bullet \mathcal{E} & = & \mathcal{F} \bullet \mathcal{E}^* \\
\mathcal{E}^+ & = & \mathcal{E} \circ \mathcal{O} = \mathcal{O} \circ \mathcal{E}
\end{array}
$$

Corollary 4 *The operations* $\_ \cup \_$, $\mathcal{O} \circ \_$, $\mathcal{O} \bullet \_$, *and* $\_^*$ *are idempotent.*

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

$$\mathcal{E} \uplus \mathcal{F} := \mathcal{E}^* \cup \mathcal{F}^* \quad .$$

With this, the famous producer-consumer situation can easily be modeled with encapsulated agents as $Prod \uplus Cons$. The construction can serve as a general scheme for modeling concurrent complex agents of the application domain.

5.1 Operational Model

The operations of $\mathbb{E}$ are implemented by constructions of Evolving Algebras. The system is different from a classic Evolving Algebra only in the point that its 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 ideas mentioned on page 14:

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 true then $\mathsf{init}(\mathcal{A}' \mid_M)$ is a rule of $\mathcal{S}$.

Systems:

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

$\mathcal{S} := \mathcal{E} \cup \mathcal{F}$: Initialization: perform the initializations of both subsystems and join the resulting states.
Operations: operations 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.
Operations in a given state $\mathcal{A}$: copy $\mathcal{A}$, perform initialization for $\mathcal{E}$, apply the rules of $\mathcal{E}$ until a fixpoint is reached. Then perform initialization for $\mathcal{F}$ and apply the rules of $\mathcal{F}$ until a fixpoint $\mathcal{A}'$ is reached. Let M be the set of locations which are updated. Then the rule if true then $\mathsf{init}(\mathcal{A}' \mid_M)$ 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.
Operations in a given state $\mathcal{A}$: Operations 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.
Operations in a given state $\mathcal{A}$: copy $\mathcal{A}$, do one step with the rules of $\mathcal{E}$ and then do one step with the rules of $\mathcal{F}$ and obtain a new state $\mathcal{A}'$. Let M be the set of locations which are updated. Then the rule if true then $\mathsf{init}(\mathcal{A}' \mid_M)$ 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.
Operations in a given state $\mathcal{A}$: copy $\mathcal{A}$, apply the rules of $\mathcal{E}$ until a fixpoint is reached, then apply the rules of $\mathcal{F}$ until a fixpoint is reached and obtain a new state $\mathcal{A}'$. Let M be the set of locations which are updated. Then the rule if true then $\mathsf{init}(\mathcal{A}' \mid_M)$ is a rule of $\mathcal{S}$.

$\mathcal{S} := \mathcal{E}^+$: Initialization: empty.
Operations in a given state $\mathcal{A}$: copy $\mathcal{A}$, perform initialization for $\mathcal{E}$ and apply the rules of $\mathcal{E}$ until a fixpoint $\mathcal{A}'$ is reached. Let M be the set of locations which are updated. Then the rule if true then $\mathsf{init}(\mathcal{A}' \mid_M)$ is a rule of $\mathcal{S}$.

$\mathcal{S} := \mathcal{E}^*$: Initialization: initialize $\mathcal{E}$.
Operations in a given state $\mathcal{A}$: copy $\mathcal{A}$, apply the rules of $\mathcal{E}$ until a fixpoint $\mathcal{A}'$ is reached. Let M be the set of locations which are updated. Then the rule if true then $\mathsf{init}(\mathcal{A}' \mid_M)$ 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. An operational specification by a flat Evolving Algebra is given in Appendix A.

6 Related Work and Conclusion

In [GHM89], an Evolving Algebra model of distributed, concurrent computations in Occam is presented. In contrast to most other work on Evolving Algebras, instead of a global state several local states are used, leading to transitions with several sources and several targets in a complex transition graph.

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.

The modifications presented in this paper adapt the Evolving Algebra concept for modeling rule based systems with choice of several alternative actions by introducing a new kind of non-determinism wrt. application of rules into the Evolving Algebra concept. The model-theoretic characterization also permits formal reasoning about such systems. The further extension to systems of Evolving Algebras provides improved means for specifying complex systems in a modular style.

References

[BB92] G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96:217–248, 1992.

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

[BS90] E. Börger and P. H. Schmitt. A formal operational semantics for languages of type Prolog III. Number 533 in Lecture Notes in Computer Science, 1990.

[CDG95] G. Del Castillo, I. Durdanovic, and U. Glaesser. An evolving algebra abstract machine. In *Proc. CSL 95*, number 1092 in Lecture Notes in Computer Science, pages 191–214. Springer, 1995.

[GHM89] Y. Gurevich, J. Huggins, and L. Moss. Algebraic operational semantics and occam. In *CSL '89 3rd Workshop on Computer Science Logic*, number 440 in Lecture Notes in Computer Science, pages 176–192, 1989.

[GR93] P. Glavan and D. Rosenzweig. Communicating evolving algebras. In *Computer Science Logic*, number 702 in Lecture Notes in Computer Science, pages 182–215. Springer, 1993.

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

[Gur91] Y. Gurevich. An attempt to discover semantics (a tutorial introduction). *Bulletin of the European Assoc. for Theoretical Computer Science*, 43:264–284, 1991.

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

[Hug94] J. Huggins. Kermit: Specification and verification. EECS Department, University of Michigan, Ann Arbor, 1994.

[May95] W. May. Fallstudie: Korrektheitsbeweis des Alternating-Bit-Protokolls (Case Study: Proving the Correctness of the Alternating-Bit-Protocol). Appendix to Diploma Thesis, Universität Karlsruhe, 1995. (in German).

A Flat Operational Model: Changing Active Rule Sets

Another operational model can be given by a flat computation using additional control information: All rules are integrated in a "Mega-Evolving Algebra", where the calls of subsystems are simulated by changing active rule sets. Thus, some control information has to be manipulated by the rules.

Each subterm of $\mathcal{Q}(\mathcal{E})$ is associated with a system which is called in some state and itself calls systems which are associated with its subterms. The "atomic" parts are classical Evolving Algebras, corresponding to simple rule sets (atomic terms in $\mathbb{R}$). The transitions of (sub)systems are either given by rules or by computations of subsystems. Each subterm $\mathcal{Q}(\mathcal{E})$ is given a unique sid. Thus, the sids are ordered hierarchically, by the calling-relation of the system. Every subsystem can be active many times, but at any point of time, there is only one thread of activity through it.

The signature is enriched by the unary function $\rho : \mathsf{sid}'s \to \{\mathsf{started}, \mathsf{waiting}, \mathsf{getreports}, \mathsf{running}, \mathsf{finished}, \mathsf{ended}\}$ with the meaning
– started: the system just has been started. Subsystems have to be started.
– waiting: the system is waiting for results of its subsystems.
– getreports: the results of subsystems are collected, at the same time, the rules of the system are applied.
– running: only for systems whose transitions are classical Evolving Algebra-rules.
– finished: the computation of the system is finished, but the results are not yet given back.
– ended: the system is finished and its results are given back.

The structure of $\mathcal{Q}(\mathcal{E})$ also has to be modeled:
– $\tau : \sigma \to \{\mathsf{seq}, \mathsf{par}, \mathsf{iterated}, \mathsf{atomic}\}$,
– $\mathsf{is\_subsystem} : \mathsf{sid}'s \to \mathsf{sid}'s$, subsystems of parallel systems,
– $\mathsf{successor} : \mathsf{sid}'s \to \mathsf{sid}'s$, subsystems of sequential systems,
– $\mathsf{startedby} : \mathsf{sid}'s \to \mathsf{sid}'s$, gives the current calling hierarchy.

We assume that every system can detect by itself whether it has reached a fixpoint – or any point it defines as a possible end point. Since parallel systems should run in isolation, the signature of the Meta-Evolving Algebra is even more complex: Each instance of a subsystem runs on an own copy of the signature which is obtained by prefixing the signature Σ with the sid of the subsystem. Then, write-access in a system with sid S is to $S.\Sigma$ whereas read-access reads the value on the lowest level where it is defined (top-down inheritance in the hierarchy given by the calling-relation).

For different types of subsystems, there are different additional rules:

Classical Evolving Algebras: Systems, whose transitions are atomic rules ($\tau(S) = \mathsf{atomic}$) are in fact classical Evolving Algebras:
$\mathsf{if}\ \rho(S) = \mathsf{started} \wedge \tau(S) = \mathsf{atomic}\ \mathsf{then}\ \rho(S) := \mathsf{running}.$
Every rule r is transformed to
$\mathsf{if}\ \rho(S) = \mathsf{running} \wedge\ ?r[S.\Sigma/\Sigma]\ \mathsf{then}\ !r[S.\Sigma/\Sigma].$
Since systems have to detect by themselves when they reach their fixpoint, there has to be some rule of the form
$\mathsf{if}\ ?r\ \mathsf{then}\ \tau(S) = \mathsf{finished}.$

Every complex system: Complex subsystems have to synchronize themselves with their subsystems. After calling their subsystem(s), they wait for their results. When all subsystems which were called are finished, the calling system receives their results in the getreports-phase:
$\mathsf{if}\ \rho(S) := \mathsf{waiting} \wedge (\forall S' : \mathsf{startedby}(S') = S \to \rho(S') = \mathsf{finished})$
$\quad \mathsf{then}\ \rho(S) := \mathsf{getreports}.$
The subsystems return their results which are written to the locations of S. Since the subsystem could be started again, it has to make its values undefined.
$\mathsf{if}\ \rho(S) = \mathsf{getreports} \wedge \mathsf{startedby}(S') = S \wedge S'.\Sigma = V \neq \mathsf{undef}$
$\quad \mathsf{then}\ S.\Sigma := V, S'.\Sigma := \mathsf{undef}, \rho(S') := \mathsf{ended},$
$\quad\quad \mathsf{startedby}(S') = \mathsf{undef}.$

Parallel systems: Parallel systems consist of simple rules, complex rules, and unconditional calls of systems.
In each step, all subsystems called unconditionally are started:
$\mathsf{if}\ \rho(S) = \mathsf{started} \wedge \mathsf{is\_subsystem}(S') = S\ \mathsf{then}$
$\quad \mathsf{startedby}(S') = S, \rho(S') := \mathsf{started}, \rho(S) := \mathsf{waiting}.$
If a subsystem S' is called by a complex rule of a system S whose guard is satisfied, then S' has to be started: Every rule $\mathsf{if}\ ?r\ \mathsf{then}\ \mathsf{call}\ S'$ is transformed to
$\mathsf{if}\ \rho(S) = \mathsf{started} \wedge\ ?r[S.\Sigma/\Sigma]\ \mathsf{then}$
$\quad \mathsf{startedby}(S') = S, \rho(S') := \mathsf{started}, \rho(S) := \mathsf{waiting}.$

Simple rules are also carried out in the getreports-phase: every rule r is transformed to
if $\rho(S) = \mathsf{getreports} \wedge\ ?r[S.\Sigma/\Sigma]$ then $!r[S.\Sigma/\Sigma]$.

After receiving the results of its subsystems and applying its own rules, the parallel system stops:
if $\tau(S) = \mathsf{par} \wedge \rho(S) = \mathsf{getreports}$ then $\rho(S) := \mathsf{finished}$.

Sequential systems: When started, a sequential system starts its first subsystem:
if $\tau(S) = \mathsf{seq} \wedge \rho(S) = \mathsf{started} \wedge \mathsf{startedby}(S') = S$ then
$\rho(S') := \mathsf{started}, \rho(S) := \mathsf{waiting}$.

If the currently active subsystem is finished, its reports are requested:
if $\tau(S) = \mathsf{seq} \wedge \rho(S) = \mathsf{waiting} \wedge \mathsf{startedby}(S') = S \wedge \rho(S') = \mathsf{finished}$ then $\rho(S) := \mathsf{getreports}$.

After the getreports-phase, the next subsystem is started – if it exists – or the sequential system itself is finished:
if $\tau(S) = \mathsf{seq} \wedge \rho(S) = \mathsf{getreports} \wedge \rho(S') = \mathsf{finished} \wedge \mathsf{successor}(S') = T \neq \mathsf{undef}$
then $\rho(S') := \mathsf{ended}, \rho(S) := \mathsf{waiting}, \rho(T) := \mathsf{started}$.
if $\tau(S) = \mathsf{seq} \wedge \rho(S) = \mathsf{getreports} \wedge \rho(S') = \mathsf{finished} \wedge \mathsf{successor}(S') = \mathsf{undef}$
then $\rho(S') := \mathsf{ended}, \rho(S) := \mathsf{finished}$.

Iterated systems: As mentioned, iterated systems have to detect by themselves when they reach their fixpoint. So there has to be some rule of the form
if $?r[S.\Sigma/\Sigma] \wedge \tau(S) = \mathsf{iterated} \wedge \rho(S) = \mathsf{started}$ then
$\rho(S) = \mathsf{finished}$.

If not already finished, an iterated system starts its subsystem to be iterated, waits for its results, and restarts itself:
if $\neg ?r[S.\Sigma/\Sigma] \wedge \tau(S) = \mathsf{iterated} \wedge \rho(S) = \mathsf{started} \wedge \mathsf{startedby}(S') = S$
then $\rho(S') := \mathsf{started}, \rho(S) := \mathsf{waiting}$.
if $\tau(S) = \mathsf{iterated} \wedge \rho(S) = \mathsf{waiting} \wedge \rho(S') = \mathsf{finished} \wedge \mathsf{startedby}(S') = S$ then $\rho(S) := \mathsf{getreports}$.
if $\tau(S) = \mathsf{iterated} \wedge \rho(S) := \mathsf{getreports}$ then $\rho(S) := \mathsf{started}$.

The transition diagram (see Figure 10) is more detailed since finished is divided into finished and reporting. On the other hand, undef and ended are merged into inactive.

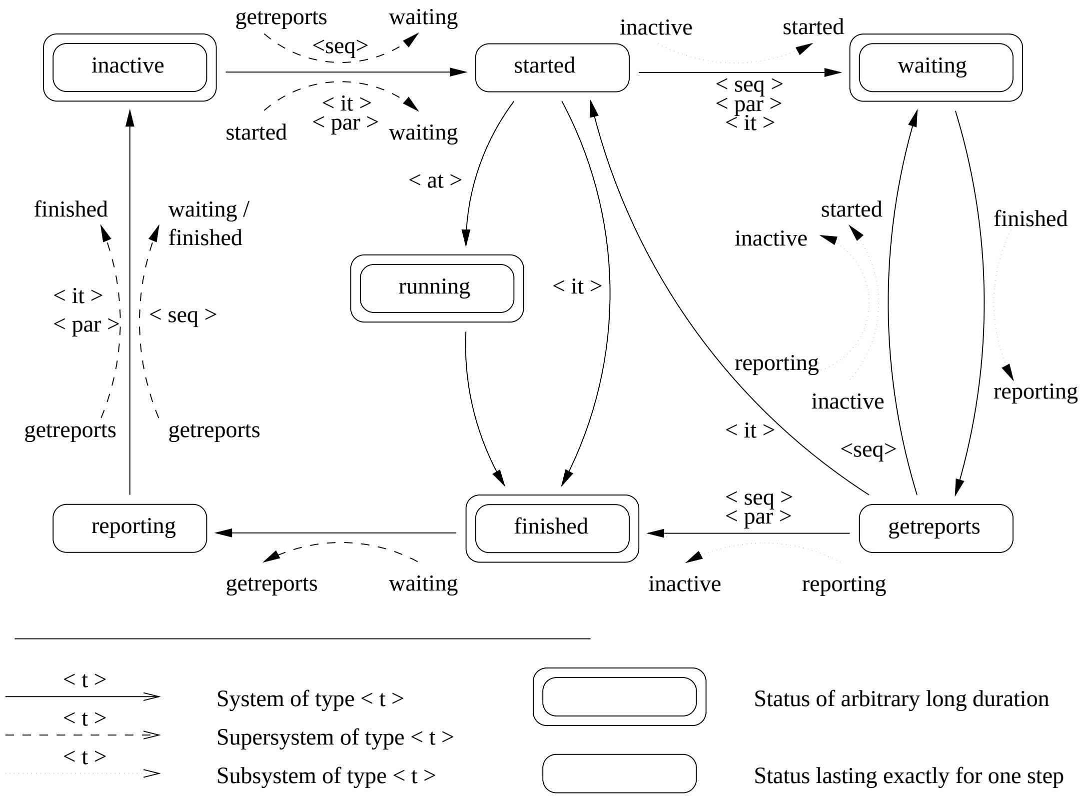


Figure 10: Transition Diagram of Activity Status