The calculus of self-modifiable algorithms: planning, scheduling and automated programming application

E. Eberbach

Jodrey School of Computer Science, Acadia University Wolfville, Nova Scotia, Canada B0P 1X0

Abstract

It is clearly an important endeavor to introduce rigorous mathematical formalisms into AI in order to more clearly see the potential and limitations of certain approaches. Showing similarities between approaches can also be highly beneficial. In this paper we present a Calculus of Self-modifiable Algorithms, which was designed to be a universal approach to parallel and intelligent systems, integrating various styles of programming and applied to a wealth of domains of future generation computers. The formal basis of the approach is outlined, the optimization inference engine is described, and the application to planning, scheduling and automated programming is presented.

1 Introduction

Providing a mathematical foundation for the study of self-modifiable algorithms is an important research task. This paper presents a Milner/Hoare style [7,10] calculus of adaptive algorithms. A Calculus of Self-modifiable Algorithms (CSA [2-6]) is a universal approach to parallel and intelligent systems, integrating various styles of programming and applied to a wealth of domains of future generation computers. It allows to analyze and design a wide class of systems with intelligence and parallelism. The CSA results suggest that a wide spectrum of AI systems can be classified as self-modifiable algorithms. In particular, neural networks, adaptive expert systems, genetic algorithms form a special case of self-modifiable algorithms. Robot plan generators, machine learning algorithms, fault-tolerant systems can be interpreted as other classes of self-modifiable algorithms. The strength of the CSA approach lies in blending different AI techniques into one unifying framework and to use the same cost optimization mechanism either to achieve a goal in machine learning, to obtain minimum error in convergence procedures for neural nets classification, to find the optimal paths of actions for robot plan generation, the most efficient computer networks routing algorithms, or to achieve the fault-tolerance in avionics systems.
In section 2 we outline the Calculus of Self-modifiable Algorithms. In section 3 the CSA inference engine is described spanning both heuristic and algorithmic approach. Section 4 presents a new application of CSA in the planning, scheduling and automated programming area.

2 A Calculus of Self-modifiable Algorithms

The Calculus of Self-Modifiable Algorithms (CSA), as described in [3-6], was designed to be a universal theory for intelligent and parallel systems, integrating various styles of programming and applied in different domains of future generation computers. The use of artificial intelligence in future generation computers requires different forms of parallelism, learning, reasoning, experience and knowledge acquisition for which there have been a considerable lack of theoretic models proposed. The CSA proposes such a theory, by introducing a mathematical model of programs with the ability to modify their behaviors based on past experience and thus accomplish most of the outlined requirements for future generation computers. CSA can therefore be used to model adaptive computer systems (programs, processes and architectures) and such “programs” are called Self-Modifiable Algorithms (SMA). The SMAs are mathematical models of programs with the ability to remember history of their past realization, and therefore they possess a certain degree of self-knowledge and this, together with its built-in optimization mechanism provide goal directed modifications of programs or algorithms so that completely new forms of algorithms can be built to achieve the goal with minimal cost. SMAs are based on mathematical models of programs; with each component having a frame-like structure and with the use of the fixed-point theory to describe the input-output behavior, and thus, the history of computation. The entire SMA structure is dynamic and changeable because of the modification transits that are incorporated in it. The modifications either change the existing transits’ preconditions, actions, postconditions, or costs, or create new transits. Thus, unlike a conventional program where given a particular set of input data the program react in a deterministic way and always provide the same output result, a SMA has the ability to produce results based on learning and experience and therefore, given a particular set of input data, the system responds differently, with the aim of providing better solutions.


2.1 EBA-formulas and SMA-net

The EBA-formulas are useful to deal with systems with incomplete information, and such systems are characteristic for AI and neurocomputing. Some basic elements of self-modifiable algorithms, i.e. control states, preconditions, activities and postconditions in transits, are predicates taking values in the Extended Boolean Algebra (EBA-formulas). EBA-formulas are to some extent similar to predicate calculus used in logic programming. The main differences are that
predicates can take three and not two values, namely: true, false and unknown; and that besides traditional disjunction, conjunction, and negation operators, there is a new "dynamic" temporal-like operator of string, verification of matching and updating condition.

Additionally, a forward and backward updating condition ternary operators, denoted by symbols >/ and </, respectively, are defined. They are used for a creation of new control states of self-modifiable algorithms.

A SMA-net (from: Self-Modifiable Algorithm net) defines a mathematical environment for self-modifiable algorithms. Two special cases of SMA-net are an RBL-net (from: Rule-Based and Logic net) and an NNC-net (from: Neural Network and Connectionist net). The SMA-net provides the means for procedural and data abstraction, i.e. the way to build the algorithm from lower-level elements using basic programming operators. It defines a set of operators to be performed on transits like sequential composition $\circ$, parallel composition $\parallel$, nondeterministic choice $\triangledown$, general choice $\sqcup$, and general recursion $\Re$ with its special cases iteration operators. Additionally, there are distinguished four special elements of the SMA-net, i.e. the bottom element $\bot$ representing the smallest program, the top element $T$ being the largest program, and sequential unity $\varepsilon$, and parallel unity $\gamma$ representing programs which are invisible in a sequential or parallel execution, respectively.

2.2 Cost System

The Cost system is used for SMA learning and adaptation. Each transit and SMA-operator are assigned a cost and the cost of the SMA equations that are built is a composition of this cost and the cost of the SMA-Net operators. SMA learning is achieved by finding the minimum of costs for a particular subset of transits that provide the cheapest solution. This process is done by the SMA inference engine optimization process and therefore the process of finding the best solution involves the mathematical process of finding a minimum. The cost value has different interpretations and can be problem specific. For example, it can represent weights in the neurocomputing sense, time of execution penalty, or in the evolutionary programming case it can represent the fitness of the transit in contributing to a useful solution.

2.3 Self-Modifiable Algorithms

The specification of a Self-Modifiable Algorithm $SMA = (S, T)$ over a SMA-net includes:

- **Control States $S$** - Initial Control State ($\sigma$) and Final Control State ($\omega$) are specified at the beginning; other control states are created during running of the algorithm.

- **Transits $T$** - Consist of Data $D$, Actions $A$, Modifications $M$. Data are passive, Actions act on Data, and Modifications act on both Data, Actions, and Modifications. Each transit consists of Precondition (pre), Activity (act), Postcondition (post), and Cost (cost). Preconditions, activities, and
postconditions have the form of EBA-formulas, and costs are represented by real (or so-called beyond-real) numbers.

3 The CSA Inference Engine: Select, Examine and Execute Phases

The SMA Inference Engine finds not a path, but an algorithm, thus the SMA-equations contain alternative solutions, which are connected by general choice operator, $\cup$, nondeterministic choice operator, $\psi$, and parallel solutions which are connected by parallel composition operator, $\parallel$. One of the basic properties of each transit is its cost. The optimization process finds the best algorithm or solution to the problem by finding the cheapest SMA that leads from the initial to the terminal control states. The process of problem solving depends on the complexity of the problem space itself; for easy problems the entire algorithm is built up to the end of the program, for complex problems heuristic searches are used to find the best solution a few steps ahead based on the SMA cost function. The Inference Engine works in three phases: the select, examine and execute phases.

The select phase involves the verification of matching of the SMA-transits and the building of the equations. The inference engine builds sets of equations $k$ steps forward or backward, where $k = 1, 2, 3, \ldots, \infty$, where $k = \infty$ means to “the end” or to “the beginning” of the algorithm, (depending on whether forward or backwards matching was used respectively) thus building the tree of control states starting from the current control state $x$ and moving to a new state $x + 1$ by making use of the transits that match the current control state. Such process continues for $k$ steps until level $x + k$ is reached.

In the examine phase the inference engine selects the optimal program from the expanded control states based on the cost of the least fixed-points solutions of equations, and this program is executed in the execute phase. The examine phase of the inference engine therefore involves finding fixed point solutions of the SMA-equations that were built and the optimization of such equations by removal of non-determinism from the equations. The fixed point solution contains redundant threads of transits and the optimization process involves removing the non-determinism from the equation by using the cost of transits and operators to select the best algorithm to solve the problem. Hence for the least-fixed-point, $\mathfrak{l}$, of the set of equations, if $\varphi(t)$ represents the cost of transit $t$, then this process involves the application of the erasing homomorphism $h(t)$ as follows:

$$ h(t) = \begin{cases} 
  t & \text{if it does not contain } \psi \\
  t_1 & \text{if } t = t_1 \cup t_2, \text{ and } \varphi(t_1) = \min(\varphi(t_1), \varphi(t_2)) \\
  t_2 & \text{if } t = t_1 \cup t_2, \text{ and } \varphi(t_2) = \min(\varphi(t_1), \varphi(t_2))
\end{cases} $$

The execution of the optimized program takes place in the execute phase. The inference engine then starts the process from the select phase again and this goes on until either no more control states can be generated or the terminal control (or initial) state is reached.
4 Planning, scheduling and automated programming application

As the illustrating application we will consider a task from the ECAI-92 Workshop on Formal Specification Methods for Complex Reasoning Systems [14]. This design task has been solved by eight different teams of researchers using their specific software tools. We will show how CSA allows to solve the same problem using select, examine and execute phases, and employing only a small subset of the CSA potentials.

The task to specify consider a simple scheduling problem. For a number of activities, a number of time periods and a set of requirements, a schedule must be designed that satisfies these requirements. In a schedule each activity is assigned to exactly one time period. The scheduling task is restricted to three time periods (T1, T2 and T3) and four activities (A1, A2, A3 and A4). The time periods are chronologically ordered by: T1 < T2 < T3. The reasoning system that is to be specified should be able to process arbitrary sets of requirements. A schedule in which the last activity takes place as early as possible is preferred.

Set of requirements:

- A3 must occur before A2
- A3 must occur before A4
- A2 and A4 must not occur in the same time period

The following SMA solves the above scheduling requirements.

\[ S = (s, t) \], where

\[ s = \{\sigma, \omega\} \] are control states, and
\[ \sigma = start.A1 \land start.A2 \land start.A3 \land start.A4 \] is the initial state,
\[ \omega = finish.A1 \land finish.A2 \land finish.A3 \land finish.A4 \] is the terminal state,
\[ T = \{\{D1, D2, D3, D4\}, \{A1, A2, A3, A4\}\} \] is the set of transits, where

\[
\begin{align*}
D1 &= \begin{cases} 
\text{pre:} & dat.D1 \\
\text{post:} & 0 \\
\text{cost:} & 0 
\end{cases} \\
D2 &= \begin{cases} 
\text{pre:} & dat.D2 \\
\text{post:} & 0 \\
\text{cost:} & 0 
\end{cases} \\
D3 &= \begin{cases} 
\text{pre:} & dat.D3 \\
\text{post:} & 0 \\
\text{cost:} & 0 
\end{cases} \\
D4 &= \begin{cases} 
\text{pre:} & dat.D4 \\
\text{post:} & 0 \\
\text{cost:} & 0 
\end{cases}
\end{align*}
\]

/* Di represents a datum for action Ai, i = 1, 2, 3, 4 */
/* Ai(Di) means action Ai working on datum Di */

\[
A1(D1) = A2(D2) = \begin{cases} 
\text{pre:} & start.A1 \\
\text{act:} & exec.A1 \\
\text{post:} & finish.A1 \\
\text{cost:} & 1
\end{cases} \\
A3(D3) = A4(D4) = \begin{cases} 
\text{pre:} & start.A3 \\
\text{act:} & exec.A3 \\
\text{pre:} & start.A4 \land finish.A3 \\
\text{act:} & exec.A4
\end{cases}
\]
In our case $\Omega = \{A_1, A_2, A_3, A_4\}$, i.e. we do not care about costs of data; $k = \infty$, i.e. we build a complete set of equations (the problem to solve is simple). A schedule in which the last activity takes place as early as possible is preferred, means that we will use the maximum concurrency strategy in building equations. Each action is assigned cost equal to one. Then the CSA optimization mechanism minimizing the costs of solutions, in fact, will minimize the number of sequential steps - the parallel complexity of the algorithm. The scheduling task is restricted to three time periods, by enforcing an additional condition on the SMA inference engine $SMA_1.outcome.cost \leq 3$. The condition that each activity is assigned to exactly one time period are fulfilled by appropriate specifications of postconditions disallowing to fire actions once again. Common condition $finish.A_3$ (with capacity 1) in actions $A_2$ and $A_4$ disallow their parallel execution, and, additionally, will make possible execution of $A_2$ and $A_4$ only after termination of $A_3$.

The $k\Omega$-Procedure of the inference engine will consist of one loop only (because $k = \infty$):

**SELECT phase:**
The forward canonical set of equations using maximum concurrency strategy for $SMA_1$ (for simplicity, we will omit data parameters in actions):

$x(start_{A1} \land start_{A2} \land start_{A3} \land start_{A4}) = (A_1 \parallel A_3) \circ$
\[ \circ x(finish_{A1} \land start_{A2} \land finish_{A3} \land start_{A4}) \]
$x(finish_{A1} \land start_{A2} \land finish_{A3} \land start_{A4}) = A_2 \circ$
\[ \circ x(finish_{A1} \land finish_{A2} \land finish_{A3} \land start_{A4}) \psi \]
\[ A_4 \circ x(finish_{A1} \land start_{A2} \land finish_{A3} \land finish_{A4}) \]
$x(finish_{A1} \land finish_{A2} \land finish_{A3} \land start_{A4}) = A_4 \circ$
\[ \circ x(finish_{A1} \land finish_{A2} \land finish_{A3} \land finish_{A4}) \]
$x(finish_{A1} \land start_{A2} \land finish_{A3} \land finish_{A4}) = A_2 \circ$
\[ \circ x(finish_{A1} \land finish_{A2} \land finish_{A3} \land finish_{A4}) \]
$x(finish_{A1} \land finish_{A2} \land finish_{A3} \land finish_{A4}) = \varepsilon$

**EXAMINE phase:**
The least fixed point, obtained by direct substitution of variables (no recursion in equations, because actions fire only once) is equal to:

$x(start_{A1} \land start_{A2} \land start_{A3} \land start_{A4}) = (A_1 \parallel A_3) \circ A_2 \circ A_4 \psi (A_1 \parallel A_3) \circ$
\[ \circ A_4 \circ A_2 \] - this is $SMA_1.outcome$ before optimization.

The cost $SMA_1.outcome.cost = \varphi((A_1 \parallel A_3) \circ A_2 \circ A_4 \psi (A_1 \parallel A_3) \circ A_4 \circ A_2) = min(\varphi((A_1 \parallel A_3) \circ A_2 \circ A_4), \varphi((A_1 \parallel A_3) \circ A_4 \circ A_2))$
\[ = min(A_1.cost + A_3.cost - 1 + A_2.cost + A_4.cost, A_1.cost + A_3.cost - 1 + A_4.cost + A_2.cost) = min(1 + 1 - 1 + 1 + 1 + 1 - 1 + 1 + 1) \leq 3, \]
i.e. it fulfills the requirements that the number of steps should not exceed three.

To find the optimal solution $SMA_1.outcome^*$ for the execute phase, we apply the erasing homomorphism $h$ to $SMA_1.outcome$: 

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cost:</td>
<td>1</td>
<td>cost:</td>
<td>1</td>
</tr>
</tbody>
</table>
\[ \text{SMA1.outcome}^* = h((A1 \Vert A3) \circ A2 \circ A4 \cup (A1 \Vert A3) \circ A4 \circ A2)) = \]
\[ = \begin{cases} (A1 \Vert A3) \circ A2 \circ A4 \\ (A1 \Vert A3) \circ A2 \circ A4 \end{cases} \]

**EXECUTE phase:**
In the execute phase the optimal program \( \text{SMA1.outcome}^* \) will be executed being equal \((A1 \Vert A3) \circ A2 \circ A4\) or \((A1 \Vert A3) \circ A4 \circ A2\) (depending what was the choice of the inference engine - both solutions have the same cost, thus any of them can be selected by the inference engine for the execution). The solutions fulfill the set of requirements that are stated at the beginning of the problem.

The same solution can be obtained if instead of the forward equations with the maximum concurrency strategy to use backward sets of equations. The work of the inference engine in the case of \( k \neq \infty \), and another scheduling example has been described in [6].

### 5 Conclusions

This paper concentrates on one specific application of CSA in the area of planning, scheduling and automatic program synthesis. A somehow similar topic has been covered in [13], where the CSA robot plan generator has been outlined. The CSA approach can be applied to a wide range of other research areas, including expert systems, machine learning, fault tolerant computing, distributed and concurrent computing, computer network evolutionary routing algorithms, new generation computer architectures and languages, databases and knowledge bases, and neurocomputing [2-6,9,11,12,15]. CSA has been proposed to be a suitable tool to investigate fifth generation "knowledge-based" computers and sixth generation "intelligent" computers. It seems that it is an appropriate tool for machine learning, i.e., for learning by analogy, from examples, from observation and by discovery. The cost language SEMAL, for which further information can be found in [1,4,8], is based on the CSA.

### Acknowledgements

The work of the author was partially supported by a grant from the Natural Sciences and Engineering Research Council of Canada, No. OGP0046501, and a general NSERC grant.

### References


