Algebraic Implementation of CTL Model Checker

FLORENTINA LAURA CACOVEAN, EMIL MARIN POPA, CRISTINA IOANA BRUMAR,
BOGDAN ALEXANDRU BRUMAR
Department of Computer Science
Lucian Blaga University of Sibiu, Faculty of Sciences
Str. Dr. Ion Ratiu 5-7, 550012, Sibiu
ROMANIA

Abstract: In this work, is presented the methodology and supporting sets of tools, what permitted the possibility automatically generate of the algorithms verify the models for temporal logics from a set of algebraic specifications. This fact is of great deal to the research of suggested aim. The development of temporal logics and the model checking the algorithms can be used to the verification property of system. These are used in to specify model checkers as the mappings of the form $\mathcal{MC}: \mathcal{L}_s \rightarrow \mathcal{L}_t$ where $\mathcal{L}_s$ is a temporal logic source language and $\mathcal{L}_t$ is a target language represents sets a state of the model M as $\mathcal{MC}\,(f \in \mathcal{L}_s) = \{s \in M \mid s \models f\}$. Is noticed how this algebraic framework can be used to specify of the model checking algorithms for *CTL* (Computation Tree Logic). The paper [10] is the base of results obtained.

Key-Words: CTL, algebraic specification, model checkers, temporal logics, algebraic structure, directed graph, implementation

1 Introduction

Temporal logic is used to express qualitative [2] the properties of the system. The model checkers are tools which can be used to verify that a given system satisfies a given temporal logic formula. The certified system may be a physical system, or concurrent either an interactive program as behave is described of the Kripke model [6]. The model is a directed graph where the nodes represent the states of the system and the edges represents the state transitions. The nodes and the edges can be labeled with atomic propositions what describe the states and the transitions of the system. A property were a certified against of a model give is written as a temporal logic formula across the propositions labeling the models. A model checker is an algorithm that determines the states of a model that satisfy a temporal logic formula.

In this paper shown how this methodology can be used to generate model checkers for *CTL* (Computation Tree Logic) [2]. Is generated symbolic model checkers for *CTL* through the replace of the target language of explicit sets with a target language of binary decision diagrams [1]. In this section is delivered a traditional definition of *CTL* and then given an algebraic description of *CTL*.

A model is defined [2] as a directed graph $M=\langle S,E,\ P{:}AP \rightarrow 2^S \rangle$ where S is a finite sets of states also called nodes, E is a finite sets of directed edges, and P represents proposition labeling function which labels each nodes with logical proposition. For each $s \in S$, use the notation $succ(s)=\{s' \in S \mid (s,s') \in E\}$. Each state in E must have at least one successor, is $\forall s \in S$, $succ(s) \neq \emptyset$. A path in M is a infinite sequence of states $(s_0,\ s_1,\ s_2,\ldots)$ such that $\forall i,\ i \geq 0$, we have $(s_i,s_{i+1}) \in E$. The labeling function P maps an atomic proposition in AP to the set of states in S on which sentences is *true*. The Figure 1 exhibits a model [2] the behavior two processes competing for the in an entrance the critical section. The atomic propositions T_i, N_i, and C_i denote, respectively, process i, $1 \leq i \leq 2$, try to enter into critical section, not to enter into critical section, and to executed in the critical section.

The CTL formulas are defined by the following rules [2]:

1. The logical constants *true* and *false* are *CTL* formulas.
2. Every atomic proposition, $ap \in AP$ is a *CTL* formula.
3. If f_1 and f_2 are *CTL* formulas, then so are $\neg f_1$, $f_1 \wedge f_2$, $f_1 \vee f_2$, $EX\ f_1$, $AX\ f_1$, $E[f_1 U f_2]$, and $A[f_1 U f_2]$.

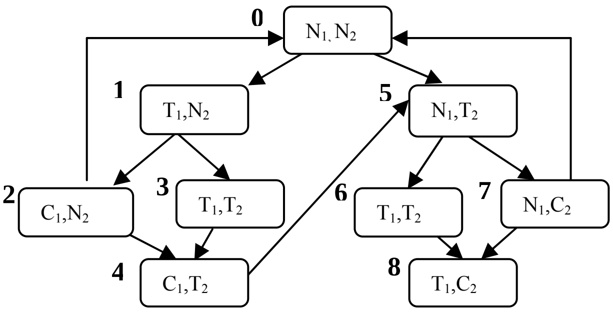

Fig. 1: Model example

Just as defined in [2], the satisfaction relation, $\models$, of *CTL* formula, f on a state s in a model M is denoted below $M, s \models f$ and is read "s in the model M satisfies f". The satisfied set of a formula f across a model M is the set $\{s \in S | M, s \models f\}$. The satisfaction of logical constants is defined as $\forall s \in S, s \models true$ and $\neg \exists s \in S, s \models false$. The relation $\models$ is defined as fallow:

$s \models ap$ iff $ap \in AP$ and $s \in P(ap)$
$s \models \neg f$ iff $not\ s \models f$
$s \models f_1 \wedge f_2$ iff $s \models f_1$ and $s \models f_2$
$s \models f_1 \vee f_2$ iff $s \models f1$ or $s \models f2$
$s \models AX f$ iff $\forall (s,t) \in E, t \models f$
$s \models EX f$ iff $\exists (s,t) \in E, t \models f$
$s \models A[f_1 U f_2]$ iff $\forall path(s_0, s_1, s_{2,\dots}), s=s_0$ and $\exists i,[i \geq 0 \wedge s_i \models f_2 \wedge \forall j[0 \leq j < i \Rightarrow s_j \models f_1]]$
$s \models E[f_1 U f_2]$ iff $\exists path(s_0, s_1, s_2, \dots), s=s_0$ and $\exists i,[i \geq 0 \wedge s_i \models f_2 \wedge \forall j[0 \leq j < i \Rightarrow s_j \models f_1]]$

Many model checking algorithms were developed for different temporal logics [2], thus in this paper is presented a simple universal algorithm based on the algorithm of homeomorphism computation which is used in an algebraic compiler [8]. The generic homeomorphism algorithm is customized by a set of specifications to implement a model checkers, implemented as an algebraic compiler $C: \mathcal{L}_s \rightarrow \mathcal{L}_t$, the source language $\mathcal{L}_s$ is the language of temporal logic and the target language $\mathcal{L}_t$ is the language of sets of states of a given models M. The algebraic compiler translates temporal logic formulas their in of satisfy $C(f)=\{s \in S | M, s \models f\}$. The primary advantage of this approach is that the model checking algorithm can be automatically generated by the tools from its specifications. These specifications consist of a finite set of rules in which each the rule defines the syntax of a class of constructs in the source language and the semantic values these constructs as the expressions in the syntax of the target language.

2 Algebraic description of *CTL*

The source and the target language used in an algebraic compiler $C: \mathcal{L}_s \rightarrow \mathcal{L}_t$ are defined using heterogeneous Σ-algebras and Σ- homeomorphisms [5]. The operator scheme of a Σ-algebras is a tuple $\Sigma = \langle S, O, \sigma \rangle$ where S is a finite set of sorts, O is it a finite set of operator names, and $\sigma : O \rightarrow S^* \times S$ is a function which defines the signature of the operators. These signatures are denote as $\sigma(o) = s_1 \times s_2 \times \dots \dots \times s_n \rightarrow s$, where $s, s_i \in S$, $1 \leq i \leq n$. A Σ - algebra is a tuple $A_\Sigma = \langle \{A_s\}_{s \in S}, Op(O) \rangle$, where $\{A_s\}_{s \in S}$ is a family of non-empty sets indexed by the sorts S of Σ, called the carrier sets of the algebra, and $Op(O)$ is a set of operations across the sets in $\{A_s\}_{s \in S}$ such that for each $o \in O$ with signature $\sigma(o) = s_1 \times s_2 \times \dots \times s_n \rightarrow s$, $Op(o)$ is a function $Op(o): A_{s1} \times A_{s2} \times_{\dots} \times A_{sn} \rightarrow A_s$. Is identified $Op(o)$ with O. A Σ- algebra is a tuple defined through $L = \langle Sem, Syn, \mathcal{L}: Sem \rightarrow Syn \rangle$ [8], where Sem is a Σ-algebra called the language semantics, Syn is a Σ word or a term algebra called the language syntax and $\mathcal{L}$ is a partial mapping called the language learning function. Is the maps semantic objects in Sem to their expressions as the valid constructs in Syn and there exists a homeomorphism $\varepsilon: Syn \rightarrow Sem$. If was $\mathcal{L}(\alpha)$ defined, therefore $\varepsilon(\mathcal{L}(\alpha)) = \alpha$, $\alpha \in Sem$, where ε is the language evaluation function. Thus, the semantics algebra Sem contains computing objects denoted by the elements of the syntax algebra Syn.

2.1. Algebraic structure of *CTL* language.

Defined *CTL* as a Σ - language, define an operator scheme Σ_{ctl} as the tuple $\langle S_{ctl}, O_{ctl}, \sigma_{ctl} \rangle$ where the sorts $S_{ctl} = \{F\}$ represents the formulas: $O_{ctl} = \{true, false, not, and, or, AX, EX, AU, EU\}$, and σ_{ctl} is represented in Figure 2.

Operator	Description in A^w_{ctl}
$true: \varnothing \rightarrow F$	$true \in A^w_F$
$false: \varnothing \rightarrow F$	$false \in A^w_F$
$not: F \rightarrow F$	if $f \in A^w_F$ then $\neg f \in A^w_F$
$and: F \times F \rightarrow F$	if $f_1, f_2 \in A^w_F$ then $f_1 \wedge f_2 \in A^w_F$
$or: F \times F \rightarrow F$	if $f_1, f_2 \in A^w_F$ then $f_1 \vee f_2 \in A^w_F$
$AX: F \rightarrow F$	if $f \in A^w_F$ then $AX f \in A^w_F$
$EX: F \rightarrow F$	if $f \in A^w_F$ then $EX f \in A^w_F$
$AU: F \times F \rightarrow F$	if $f_1, f_2 \in A^w_F$ then $A[f_1 U f_2] \in A^w_F$

Operator	Description in A^w_{ctl}
$EU: F\times F\to F$	if $f_1, f_2 \in A^w_F$ then $E[f_1 U f_2] \in A^w_F$

Fig. 2: The operator scheme Σ_{ctl} in A^w_{ctl}

CTL can be defined as the Σ_{ctl}-language give in the form $L_{ctl}=\langle A^m_{ctl}, A^w_{ctl}, \mathcal{L}_{ctl}: A^m_{ctl} \to A^w_{ctl} \rangle$. A^w_{ctl} is the word algebra of the operator scheme Σ_{ctl} generated of the operations from O_{ctl} and a finite sets of variables, denoting atomic propositions, *AP*. A^m_{ctl} represents *CTL* semantics algebra defined across the satisfy sets of *CTL* formulas for a given models *M*. $\mathcal{L}_{ctl}$ is a mapping which associates satisfiability sets in A^m_{ctl} from the *CTL* expressions in A^w_{ctl} that they satisfy and ε_{ctl} is a homeomorphism that evaluates *CTL* expressions in A^w_{ctl} to their satisfy sets in A^m_{ctl}. How long the rules for forming *CTL* formulas are independent of any model, the signification of the resulting formulas are dependent upon a given model. Thus in the algebraic definition of *CTL*, A^w_{ctl} is independent from any model while A^m_{ctl} is dependent on the given model *M*.

The word algebra A^w_{ctl} is unique to homeomorphism in the class of algebras with operator scheme Σ_{ctl}. This has as the carrier set A^w_F, the collection of *CTL* formulas, called terms or words, created by the juxtaposition of variables in *AP* and the operator symbols in O_{ctl} by the rules shown in Figure 2. The constants *true* and *false* are the free generators of the ground terms of A^w_{ctl} since they form the set of nullary operators $O^0_{ctl}=\{true, false\}$. The carrier set A^w_F is formed by the following rules [3]:

1. If $f \in AP$ or $f \in O^0_{ctl}$, then f is a *CTL* formulas, $f \in A^w_F$.
2. If $o \in O_{ctl}$ is an operator name with signature $\sigma(o)=F\times F\to F$ or $\sigma(o)=F\to F$ and $w, w_1, w_2 \in A^w_F$, then the words $o_{ctl}(w_1, w_2)$ or $o_{ctl}(w)$ constructed by the rules shown in Figure 2 are *CTL* formulas, $(w_1)o_{ctl}(w_2) \in A^w_F$ or $o_{ctl}(w) \in A^w_F$.

The *CTL* semantic algebra A^m_{ctl} has the same operator scheme Σ_{ctl} as the syntax algebra, but he has a different carrier set, A^m_F, and thus different operators. For a models $M=\langle S, E, P: AP\to 2^S\rangle$ the carrier set A^m_F is the set of sets of states in S what satisfy the *CTL* formulas in A^w_F, represented through $A^m_F \subseteq 2^S$. Of when A^w_{ctl} and A^m_{ctl} are similar, each operator in A^w_{ctl} corresponds to an operator from A^m_{ctl} with the same signature. We will used different symbols for denote of the operations, because the operators in A^m_{ctl} operate on sets and the operators from A^w_{ctl} operate on terms. The operators from A^m_{ctl} corresponding to the names *{true, false, not, and, or, AX, EX, AU, EU}* in O_{ctl} of Σ_{ctl} are respectively named $\{S, \varnothing, C, \cap, \cup, Next_{all}, Next_{some}, LFP_{all}, LFP_{some}\}$. These actions in A^m_{ctl} are defined as follows:

- S is the constant set of all states in M and $\varnothing$ is the constant empty set.
- C is the unary operator which produces the complement in S of its argument.
- $\cap$ and $\cup$ are the binary set intersection and union operators.
- For $t \in A^m_F$, are unary operators $Next_{all}$ and $Next_{some}$ are defined by the equations $Next_{all}(t)=\{s \in S \mid succ(s) \subseteq t\}$, and, $Next_{some}(t)=\{s \in S \mid succ(s) \cap t \neq \varnothing\}$, where *succ(s)* denotes the successors of the state *s* in *M*.
- LFP_{all} and LFP_{some} are inspired by the fixed point construction operator *Y* [4]. For sets $LFP_{all}(t_1, t_2)$, with $t_1, t_2 \in A^m_F$, computes the least fixed point of the equation $Z=t_2 \cup \{s \in t_1 \mid succ(s) \subseteq Z\}$, and $LFP_{some}(t_1, t_2)$, with $t_1, t_2 \in A^m_F$, computes the least fixed point of the equation $Z=t_2 \cup \{s \in t_1 \mid (succ(s) \cap Z) \neq \varnothing\}$ [2].

2.2 Algebraic structure of the model.

Defined as an algebraic compiler, the *CTL* model checking algorithm maps *CTL* formulas in the syntax algebra A^w_{ctl} into set expressions of a set expression language defined by the model *M*, while preserving the satisfiability semantics of the *CTL* formulas. In order to understand these mapping we structure the model $M=\langle S, E, P: AP\to 2^S\rangle$ as a Σ-language whose syntax algebraic contains the sets of the expressions and whose semantics algebra contains the sets in 2^S. The operator scheme for this language $\Sigma_{sets}=\langle S_{sets}, O_{sets}, \sigma_{sets}\rangle$ where $S_{sets}=\{S, B\}$, again S is the sort for sets and B is the sort for the boolean values, $O_{sets}=\{\varnothing, \cap, \cup, \setminus, equiv, \subseteq, succ, \neg,$

$\wedge, \vee\}$ and σ_{sets} is presented in Figure 3. The model M defined as Σ_{sets} - language $L_M = \langle A^m_{sets}, A^w_{sets}, \mathcal{L}_{sets}: A^m_{sets} \rightarrow A^w_{sets} \rangle$, where $\varepsilon_{sets}: A^w_{sets} \rightarrow A^m_{sets}$. He evaluates set expressions to the sets they represent.

In this language, A^m_{sets} is the semantic algebra with the carrier sets $A^m_S = 2^S$ and $A^m_B = \{true, false\}$. The operators in algebra and their signatures as defined by σ_{sets} are shown in Figure 3. Semantic algebra A^m_{sets} and A^m_{ctl} are the carrier sets in the relation $A^m_F \subseteq A^m_S$. This permits for the identity show in the map all the elements of the carrier sets of A^m_{ctl} through their occurrences in the carrier sets of A^m_{sets}

Operator	Description in A^m_{sets}
$\varnothing$:$\varnothing \rightarrow$S	$\varnothing \in A^m_S$
$\cap$:S×S→S	if $S_1, S_2 \in A^m_S$ then $S_1 \cap S_2 \in A^m_S$
$\cup$:S×S→S	if $S_1, S_2 \in A^m_S$ then $S_1 \cup S_2 \in A^m_S$
\\:S×S→S	if $S_1, S_2 \in A^m_S$ then $S_1 \backslash S_2 \in A^m_S$
equiv:S×S→B	if $S_1, S_2 \in A^m_S$ then S_1 *equiv* $S_2 \in A^m_B$
$\subseteq$:S×S→B	if $S_1, S_2 \in A^m_S$ then $S_1 \subseteq S_2 \in A^m_B$
succ:S→S	if $S_1 \in A^m_S$ then
	$\{s \in S \mid s' \in S_1 \wedge s \in succ\ (s')\} \in A^m_S$
$\neg$:B→B	if $b_1 \in A^m_B$ then $\neg$ b₁ $\in A^m_B$
$\wedge$:B×B→B	if $b_1, b_2 \in A^m_B$ then $b_1 \wedge b_2 \in A^m_B$
$\vee$:B×B→B	if $b_1, b_2 \in A^m_B$ then $b_1 \vee b_2 \in A^m_B$

Fig. 3: Operator scheme Σ_{sets} in A^m_{sets}

The syntax algebra A^w_{sets} is word algebra for operator scheme Σ_{sets} generated from the operator names in O_{sets} and a finite set of variables. The ground terms of A^w_{sets}. Are the set expressions don't contain any variables. All these evaluates to the empty set. These terms are only through met the in the introduction of the variables specified by a model that we can generate. The set of variables used in A^w_{sets} is specified of the model $M = \langle S, E, P: AP \rightarrow 2^S \rangle$ and contains the atomic proposition *AP* and the variable *S* represents the sets of states *S*. In a set expression, the atomic proposition *ap* is used to generate the set of the states *P(ap)*, this is the set of states on which in *ap* is satisfied.

2.3 Algebraic describe of *CTL* model checker

We define a *CTL* model checker as an algebraic compiler $\mathcal{MC}: L_{ctl} \rightarrow L_M$ by the pair of embedding morphisms $\langle T_{MC}, H_{MC} \rangle$. $T_{MC}: A^w_{ctl} \rightarrow A^w_{sets}$ maps *CTL* formulas from word algebra A^w_{ctl} to set expressions in A^w_{sets}, which evaluate to the stisfiability sets of the *CTL* formulas, $H_{MC}: A^m_{ctl} \rightarrow A^m_{sets}$, maps sets in A^m_{ctl} by the identity mapping to sets in A^m_{sets}. The morphism T_{MC} is constructed by the following approach:

1. The associate each operation o_{ctl} from the algebra A^m_{ctl} with a set expression $d(o_{ctl})$ from the algebra A^w_{sets} with the property $H_{MC}(o_{ctl}(S_1,\ldots,S_n)) = \varepsilon_{sets}\ (d(o_{ctl})\ (d_{ctl}(S_1),\ldots,d_{ctl}(S_n)))$. The variables are treaties as nullary operators each $ap \in AP$ is associated with the set expression *P(ap)*, *d(ap)=P(ap)*. This construction is presented in the Figure 4. The sets expressions exists in an extension of A^w_{sets} and contains constants sets $(\varnothing, S, P(ap) \mid ap \in AP)$, set variables (Z, Z'), assignment statements, and a while loop constructs. The $d_{ctl}(t)$, $d_{ctl}(t_1)$, $d_{ctl}(t_2)$ are the set expressions associated with the arguments of the operators $o \in A^m_{ctl}$.

$o \in A^m_{ctl}$	$d_{ctl}\ (o) \in A^w_{sets}$
$\varepsilon_{ctl}(ap)$	$P(ap)$
S	S
$\varnothing$	$\varnothing$
$C(t)$	$S \setminus d_{ctl}(t)$
$t_1 \cap t_2$	$d_{ctl}(t_1) \cap d_{ctl}(t_2)$
$t_1 \cup t_2$	$d_{ctl}(t_1) \cup d_{ctl}(t_2)$
$Next_{all}(t)$	$\{s \in S \mid succ(s) \subseteq d_{ctl}(t_1)\}$
$Next_{some}(t)$	$\{s \in S \mid \neg(succ(s) \cap d_{ctl}(t)\ equiv\ \varnothing)\}$
$LFP_{all}(t_1, t_2)$	$Z = d_{ctl}(t_2);\ Z' = \varnothing;$
	$while\ (\neg Z\ equiv\ Z')\ Z' = Z;$
	$Z = \{s \in S \mid succ(s) \subseteq d_{ctl}(t_1)$
	$d_{ctl}(LFP_{all}(t_1, t_2)) = Z;$
$LFP_{some}(t_1, t_2)$	$Z = d_{ctl}(t_2);\ Z' = \varnothing;$
	$while\ (\neg Z\ equiv\ Z')\ Z' = Z;$
	$Z = \{s \in S \mid \neg(succ(s) \cap d_{ctl}(t_1)\ equiv\ \varnothing)\}$
	$d_{ctl}(LFP_{some}(t_1, t_2)) = Z;$

Fig. 4: Construction *d* over the generators and operations of A^m_{ctl} and A^w_{ctl}

2. The set expressions the second column of Figure. 4 define the operations of an algebra A'^w_{sets} which is similar to the operations from

A^m_{ctl} and from A^w_{ctl}. This algebra is generated by the operators $d(o_{ctl})$ using a free generators the constants sets $d(true)$, $d(false)$, $d(ap)$, $ap \in AP$ and is isomorphic with the algebra A^w_{ctl}. This results from the fact that they are similar and their generators are one to one through the operators in O_{ctl}. Using the unique extension lemma [8] this one to one mapping on the generators extends to a unique isomorphism T'_{MC}: $A^w_{ctl} \rightarrow A'^w_{sets}$

3. A'^w_{sets} is a subalgebra of A^w_{sets}, the embedding T_{MC}: $A^w_{ctl} \rightarrow A'^w_{sets}$ is constructed by the composition of T'_{MC} and the injection function I: $A'^w_{sets} \rightarrow A^w_{sets}$, with $T_{MC}=T'_{MC} \circ I$.

$$\begin{tikzcd}
L_{ctl}: A^m_{ctl} \arrow[r, "L_{ctl}"] \arrow[d, "H_{MC}"'] \arrow[dr, "D_{ctl}"] & A^w_{ctl} \arrow[r, "\varepsilon_{ctl}"] \arrow[d, "T_{MC}"] & A^m_{ctl} \arrow[d, "H_{MC}"] \arrow[dl, "D_{ctl}"'] \\
L_M: A^m_{sets} & A^w_{sets} \arrow[l, "\varepsilon_{sets}"] & A^m_{sets} \arrow[l, "L_{sets}"]
\end{tikzcd}$$

Fig. 5: Σ - language L_{ctl} and L_M and the model checker $\mathcal{MC}_M$: $L_{ctl} \rightarrow L_{sets}$

The morphisms T_{MC} and H_{MC} thus constructed are presented in the diagram from Figure 5. The diagram is commutative. Commutatively assures the fact as T_{MC} keeps the meaning of the formulas from A^w_{ctl} when mapping them to set expressions in A^w_{sets}. The diagonal mapping D_{ctl}: $A^m_{ctl} \rightarrow A^w_{sets}$ in Figure 5 is generated by d_{ctl} defined in Figure 4 and show the translation process performed by T_{MC} using derived operations [3, 8].

2.4 Algebraic implementation of CTL model checker

Construction of T_{MC} in the Section 2.3 can be entered into an algorithm which implements the *CTL* model checker. This algorithm is universal in the sense that given operator scheme Σ_{ctl} and a model M the model checker of L_{ctl} is automatically generated from the specifications of $\langle \Sigma_{ctl}, D_{ctl} \rangle$. This specification is obtained by associating each operation $o \in O_{ctl}$ with an operation a derivative $d_{ctl}(o) \in D_{ctl}$. To define the derived operations that implement the operations of Σ_{ctl} from the algebra A^w_{sets} using meta-variables that take as values set expressions of the carrier sets of A^w_{sets}. For each operation $o \in O_{ctl}$ such that $\sigma_{ctl}(o)=s_1 \times \ldots \times s_n \rightarrow s$, $d_{ctl}(o)$
takes it as the formal parameters the meta-variables denoted by $@_i$, $1 \le i \le 2$, where $@_i$ denotes the set expression associated with i - th argument of $d_{ctl}(o)$; the meta-variable $@_0$ is used to denote the resulting set expression, as example $@_0=d_{ctl}(o)(@_1, \ldots, @_n)$.

Since it is not always possible expressed the semantics of all of the operations from Σ_{ctl} using of the compositions of the operations from A^w_{sets} are written the derivative operations using a semantics expression language which includes the set operations in A^w_{sets} and extends it with local set variables, set assignment statements and looping constructs. The derivative operations written in this extended semantics expression language are called macro-operations.

Formally T_{MC} is implemented by associating with each the operation $o \in O_{ctl}$ from A^w_{ctl} a parameterized macro-operation denoted by $d(o)$, that is constructed by the following rules:

1. Define the macro-operations for the generators of the A^w_{ctl} by assigning $d(true)=S$ and $d(false)=\varnothing$, and for each $ap \in AP$, $d(ap)=P(ap)$.
2. Embed the generators of A^w_{ctl} in A^w_{sets} by the function: $T_{MC}(true)=d(true)$, $T_{MC}(false)=d(false)$, and $\forall ap \in AP$, $T_{MC}(ap)=d(ap)$.
3. Extend the function defined in (2) above to the entire algebra A^w_{alc} by the equality: $\forall f \in A^w_{ctl}$ with $f=o(f_1,\ldots,f_n)$, $T_{MC}(f)=d(o)(T_{MC}(f_1), \ldots, T_{MC}(f_n))$.

Where through T_{MC} is the unique extension of the second function to a homeomorphism, T_{MC} is well-defined. This homeomorphism maps *CTL* formulas into set expressions which evaluate to the satisfiability sets of the formulas in the model M defining the target language L_M. The general aim of the algorithm for computing of the image of a formula f in A^w_{ctl} under the embedding T_{MC}: $A^w_{ctl} \rightarrow A^w_{sets}$ assumes the existence of the following:

1. The function S: $A^w_{ctl} \rightarrow S_{ctl} \cup \{none\}$ recognizes the generators of A^w_{ctl} defined by:

$$S(f)= \begin{cases} s, & \text{if } f \text{ is a generator in carrier set } A^w_s \\ & \text{for some } s \in S_{ctl} \\ none, & \text{otherwise} \end{cases}$$

2. The function $\mathcal{P} : A_F^w \setminus AP \rightarrow \bigcup_{n=0}^{\infty} O_{ctl} \times A_{s_1}^w \times \ldots \times A_{s_n}^w$ recognizes *CTL* formulas $f \in A_F^w$ in terms of their constructions operator and their component sub formulas, that is if $f=o(f_1,\ldots,f_n)$ then $\mathcal{P}(f)=(o,(f_1,\ldots,f_n))$.

$\mathcal{S}$ is a scanning algorithm that is automatically generated from regular expressions of conditions [9] specifying the generators of A_{ctl}^w and $\mathcal{P}$ is a parser that determines which operation and which sub formulas where used to create the formula *f*. Since $\mathcal{P}$ is compositional we use a pattern-matching parser [7] for the suggested aim. For each $f \in A_s^w$, $s \in S_{ctl}$, the behavior of the algorithm that implements T_{MC} is described by the following recursive functions:

$T_{MC}(f)=$ **if** $\mathcal{S}(f)=s, s \in S_{ctl}$ **then** $d(f)$
else if $\mathcal{P}(f)=(o,(f_1,\ldots,f_n))$
then $d(o)(T_{MC}(f_1),\ldots,T_{MC}(f_n))$.

We improve the efficiency of a model checker by replacing the operations of expressions set construction by functions which evaluate the incoming set expressions and thus generates sets that are the values of the set expressions.

3. Conclusion.

The behavior of the model checker algorithm demonstrated in the section 2.4 consists of identifying the sets of states of a model *M* whence satisfy each of a sub formula of a given *CTL* formula *f* and constructing the set of states, from these sets, that satisfy the formula *f*. This is certainly the behavior of the algorithm for the homeomorphism computation performed by an algebraic compilers; Thus is evaluated an expression by repeatedly identifying its generating sub expressions and replacing them with their images in the target algebra. In the case of the model checking algorithm, sub expressions are *CTL* sub formulas and their images are the sets of states in the model satisfies the sub formulas.

References:

[1] R.E. Bryant. *Graph-based algorithms for Boolean function manipulation.* IEEE Transactions on Computers, 35(8):677-691, August 1986.

[2] E.M. Clarke, E.A. Emerson, and A.P. Sistla. *Automatic verifications of finite-state concurrent systems using temporal logic specifications. ACM Transactions on programming Languages and Systems,* No. 8(2), 1986, pp.244-263.

[3] P.M. Cohn. *Universal Algebra.* Reidel, London, 1981.

[4] M. Gordon. *Programming Language Theory and its Implementation.* Prentice Hall, 1988.

[5] P.J. Higgins. *Algebras with scheme of operators.* Mathematische Nachrichten, No.27, 1963/64, pp.115-132,

[6] S.Kripke. *Semantical analysis of modal logic: Normal modal propositional calculi.* Zeitschrift f. Math. Logik und Grundlagen d. Math., 9, 1963.

[7] T. Rus. *Parsing languages by pattern matching*. IEEE Transactions on Software Engineering, 1988, No.14(4), pp.498-510.

[8] T. Rus. *Algebraic construction of computers. Theoretical* Computer Science, No.90, 1991, pp.271-308.

[9] T. Rus and T. Halverson. *A language independent scanner generator.* Available at ftp://ftp.cs.uiowa.edu/pub/rus/scan3.ps, 1998.

[10] T. Rus, E.V. Wyk, T. Halverson. *Generating model checkers from algebraic specifications*, 1998.