1 Introduction
Verification of a model is one of the most useful technologies for automatic system verifications. In the model checking approach, the system behaviour is modeled by a Kripke structure. The specification properties that must be accomplished by a system are expressed as formulas in a propositional temporal logic (for example CTL).

The model checking takes as input the model and a formula, and then checks if the formula is satisfied or not by the Kripke model. If the formula is not satisfied by the model, the system will provide an error and any information necessary to know where the formula was not satisfied.

In this paper is defined the principle of minimal change for the update of a CTL model. The necessary semantic and the computational properties are provided for the CTL model update. Based on these ascertainments is developed the algorithm for updating the CTL model. Presenting a case study, we will show the correction of the original system specification.

2 CTL Syntax and semantics
CTL is a branching-time logic, meaning that its formulas are defined by the following rules [1]: (a) The logical constants true and false are CTL formulas; (b) Every atomic proposition, ap ∈ AP is a CTL formula; (c) If f1 and f2 are CTL formulas, then so are ¬f1, f1∧f2, f1∨f2, EX f1, AX f1, E[f1Uf2], and A[f1Uf2].

2.1 Syntax definition of CTL model checker
A CTL has the following syntax given in Backus-Naur form [4]:

\[
j :: \top | \bot | \neg f_1 | f_1 \land f_2 | f_1 \lor f_2 | \exists x.f_1 | \forall x.f_1 | \text{AG} f_1 | \text{EG} f_1 | \text{AF} f_1 | \text{EF} f_1 | A[f_1Uf_2] | E[f_1Uf_2] \quad \forall p \in \text{AP}
\]

A CTL formula is evaluated on a Kripke model M. A path in M from a state s is an infinite sequence of states and is denoted as \( \pi = [s_0, s_1, ..., s_i, s_{i+1}, s_{i+2}, ...] \) such that \( s_0 = s \) and \( (s_{i-1}, s_i) \in \text{Rel} \) holds for all \( i \geq 0 \). We write \( (s_i, s_{i+1}) \subseteq \pi \) and \( s_i \in \pi \). If we express a path as \( \pi = [s_0, s_1, ..., s_i, ..., s_j, ..., s_l] \) and \( i < j \), we say that \( s_i \) is a state earlier than \( s_j \) in \( \pi \) and \( s_i < s_j \). For simplicity, we may use \( \text{succ}(s) \) to denote state \( s_0 \) if there is a relation \( (s, s_0) \) in Rel.

2.2 Semantic definition of CTL model checker
Let \( M = (S, \text{Rel}, P; AP \rightarrow 2^{AP}) \) be a Kripke model for CTL. Given any \( s \in S \), we define whether a CTL formula \( f \) holds in state \( s \). We denote this by \( (M, s) \models f \). The satisfaction relation \( \models \) is defined by structural induction on fourteen CTL formulas [10].

Without a detailed declaration we presuppose that all the five formulae CTL presented in the contextually are by-path satisfied. Toward example, we consider the update of a Kripke model with CTL formulas, beginning from the fact that \( f \) is not satisfied.

The input is a CTL Kripke model which not satisfies the formula \( f \) and the output is an updated CTL Kripke model which satisfies the input formula.

In the beginning is presented a general definition of the CTL model update.
The next definition presents just a prerequisite for requirements of the CTL model updater and not tells how the update to be performed.

The CTL Update Definition [4]: Be a CTL Kripke model \( M = (S, \text{Rel}, P) \) and a CTL formula \( f \). An update of \( M = (M, s_0) \), where \( s_0 \in S \) with \( f \) is a CTL Kripke model \( M' = (S', \text{Rel}', P') \) such that \( M' = (M', s_0') \), \( (M', s_0') \Vdash f \) where \( s_0' \in S' \). We use \( \text{Update}(M, f) \) to denote the result \( M \) and \( \text{Update}(M, f) = M' \) if \( M' \Vdash f \).

3 Primitive update operations

Pr1. Add an only relation. Given \( M = (S, \text{Rel}, P) \), its updated model \( M' = (S', \text{Rel}', P') \) is the result of \( M \) having added only one new relation. That is \( S' = S, P' = P \), and \( \text{Rel}' = \text{Rel} \cup \{ (s_{\text{addrel}}, s_{\text{addrel2}}) \} \) where \( (s_{\text{addrel}}, s_{\text{addrel2}}) \in \text{Rel} \) for one pair of \( s_{\text{addrel}} \in S \).

Pr2. Remove an only relation. Given \( M = (S, \text{Rel}, P) \), its updated model \( M' = (S', \text{Rel}', P') \) is the result of \( M \) having removed only one existing relation. That is, \( S' = S, P' = P \), and \( \text{Rel}' = \text{Rel} \setminus \{ (s_{\text{remrel}}, s_{\text{remrel2}}) \} \) where \( (s_{\text{remrel}}, s_{\text{remrel2}}) \in \text{Rel} \) for one pair of \( s_{\text{remrel}} \in S \).

Pr3. Substitute a state and its associated relations. Given \( M = (S, \text{Rel}, P) \), its updated model \( M' = (S', \text{Rel}', P') \) is the result of \( M \) having substituted only one existing state and its associated relations. That is, \( S' = S' \cup \{ s_{\text{subst}} \} \). \( S' \) is the set of states where one state \( s \) in \( S \) is substituted by \( s_{\text{subst}} \), \( \text{Rel}' = \text{Rel} \cup \{ (s_{\text{ kullandı}}, s_{\text{subst}}) \} \) where \( (s_{\text{ữu}}, s_{\text{subst}}) \in \text{Rel} \), for some \( s_{\text{ữu}}, s_{\text{subst}} \in S \), \( s_{\text{ữu}} \in S' \) and \( P'(s_{\text{subst}}) = \tau (s_{\text{subst}}) \), where \( \tau \) is a truth assignment on \( s_{\text{_subs}} \).

Pr4. Add a state and its associated relations. Given \( M = (S, \text{Rel}, P) \), its updated model \( M' = (S', \text{Rel}', P') \) is the result of \( M \) having added only one new state and its associated relations. That is, \( S' = S \cup \{ s_{\text{addstate}} \} \). \( S' \) is the set of states where one state \( s \) in \( S \) is added by \( s_{\text{addstate}} \), \( \text{Rel}' = \text{Rel} \cup \{ (s_{\text{addstate}}, s_{\text{addstate}}) \} \) where \( (s_{\text{addstate}}, s_{\text{addstate}}) \in \text{Rel} \) and \( P'(s_{\text{addstate}}) = \tau (s_{\text{addstate}}) \), where \( \tau \) is a truth assignment on \( s_{\text{addstate}} \).

Pr5. Remove a state and its associated relations. Given \( M = (S, \text{Rel}, P) \), its updated model \( M' = (S', \text{Rel}', P') \) is the result of \( M \) having removed only one existing state and its associated relations. That is, \( S' = S \setminus \{ s_{\text{remstate}} \} \). \( S' \) is the set of states where one state \( s \) in \( S \) is removed by \( s_{\text{remstate}} \), \( \text{Rel}' = \text{Rel} \setminus \{ (s_{\text{remstate}}, s_{\text{remstate}}) \} \) for some \( s_{\text{remstate}}, s_{\text{remstate}} \in S \) and \( P'(s) = P(s) \) for all \( s \in S \setminus S' \) and \( P'(s_{\text{remstate}}) = \tau (s_{\text{remstate}}) \), where \( \tau \) is a truth assignment on \( s_{\text{remstate}} \).

All changes on CTL model can be specified in terms of these five operations. We can argue that Pr3 is accomplished by Pr4 and Pr5. Anyway, we treat state substitution differently than a combination of state addition and state deletion. Whenever a substitute of a state is needed, is applied Pr3 directly more than Pr4 followed by Pr5. This thing will simplify the definition of minimal change of the CTL model [10, 2].

The Definition of Admissible Update is given by assertion: Given a CTL Kripke model \( M = (S, \text{Rel}, P), M' = (M, s_0) \), where \( s_0 \in S \), and a CTL formula \( f \), \( \text{Update}(M, f) \) is called admissible if the following conditions hold:

1. \( \text{Update}(M, f) = (M', s_0') \models f \) where \( M' = (S', \text{Rel}', P') \) and \( s_0' \in S' \);
2. \( \text{There not exist another resulting model} \ M'' = (S'', \text{Rel}'', P'') \text{ and } s_0'' \in S'' \text{ such that } (M'', s_0'') \models f \text{ and } M'' \prec M' \).

We denote \( M_1 \preceq M_2 \) if \( M_1 \preceq M_2 \) and \( M_2 \preceq M_1 \). If given three CTL Kripke models \( M_1, M_2, M_3 \), where \( M_1 \) and \( M_2 \) are obtained from \( M_3 \) by applying \( Pr_1, Pr_2, Pr_3, Pr_4, Pr_5 \) operations, we say that \( M_1 \) is closer to \( M_2 \) as \( M_2 \), denoted as \( M_1 \preceq M_2 \), iff \( \text{Diff}(M_1, M_2) \prec \text{Diff}(M, M_2) \). We denote \( \text{Diff}(M_1, M_2) \prec \text{Diff}(M, M_2) \) iff for each \( i \) with \( i = 1, ..., 5 \), \( \text{Diff}(M_1, M_2) \preceq \text{Diff}(M, M_3) \) where \( M_1, M_2, M_3 \) are three CTL models.

4 Semantic Characterizations

In this section we will present a variety of semantic characterizations of the CTL model updater that present possible solutions to achieve admissible updates under certain conditions. In many situations, as will be shown in the following, a single type primitive operation will be enough to achieve an admissible updated model. These model characterizations also play an essential role in simplifying the implementation of the CTL model update.

First of all we shall return to the definition of the CTL Model Update. The algorithm is designed following a similar style of CTL model checking algorithm SAT [4], where an updated formula is parsed through its structure and recursive calls to proper functions are made to its sub-formulas [2].

From the definition of Admissible Update (stated above) we observe that, for a given CTL Kripke model \( M \) and a formula \( f \), there may be many admissible updates satisfying \( f \), where some updates are simpler than others.

We consider a special form of path \( \pi \) where the first \( i \) states starting from \( s_0 \) already satisfy formula \( f \). Under this situation, we can simply cut off the path. For example we can apply one time \( Pr_2 \) or \( Pr_3 \) to disconnect all other states not satisfying \( f \) [10, 2].

The implementation can be formulated in the following form:

```c
Function Update_{AG}(M,f)
/* M \# AG f. Update M to satisfy AG f */
{
```
if $M_0 = (M,s_0) \not\models f$, then $Pr_3$ is applied to $s_0$ such that $M' = CTLUpdate(M_0, f)$;
else {
1. select a path $\pi = \{s_0, s_1, \ldots\}$, where $\exists s_i \in \pi$ such that $M_i = (M, s_i) \not\models f$;
2. select a state $s_i \in \pi$ such that $\neg \exists s_j < s_i$ with $(M, s_j) \not\models f$ then (1) Applying $Pr_2$ to remove relation $(pre(s_i), s_i)$,
   then $S' := S$;
   $Rel' := Rel - \{(pre(s_i), s_i)\}$;
   $P' := P$, since is removed a only relation;
   or (2) Applying $Pr_2$ to remove state $s_i$, its associated relation,
   then $S' := S - \{s_i\}$;
   $Rel' := Rel - \{(pre(s_i), s_i), (s_i, succ(s_i))\}$
   where if associated relations of $s_i$;
   $P' := P(s)$, is removed a only relation;
   or (3) Applying $Pr_3$, $M' = CTLUpdate(M_i, f)$;
  if $M' \models AG f$ then return $M'$;
  else return $\{Update_{EG}(M', f)\}$;
}
for start the moving. The aim of the model is to find where the faulty process is. The objective of model updating, with other words, is to correct the original model which contains the faulty process. Starting from the original CTL structure for the elevator control system presented in the Fig. 2 with seven states of the system denoted with \( s_1, s_2, \ldots, s_7 \), we have added the state \( s_4 \) for checking if another passenger is requesting the elevator.

The Kripke model has seven states and the propositional variables are from the set \{Start, Close, Move, Error\}. Start represented the start button for start moving up or down the elevator, Close represent the close door to the lift cabin, Move is moving up or down the elevator and Error means occur some error.

The formal definition of the Kripke structure of the elevator controller is given by: \( M = (S, R, P) \), where \( S = \{s_1, s_2, \ldots, s_7\} \), \( R = \{(s_1, s_2), (s_2, s_3), (s_3, s_4), (s_4, s_3), (s_4, s_5), (s_5, s_6), (s_6, s_7), (s_7, s_7), (s_7, s_4), (s_4, s_1), (s_1, s_d), (s_d, s_4)\} \), \( P \) assigns state \( s_1 \) in \( M \) with not start, not close, not move and not error, that is set \( \neg \text{Start} \), \( \neg \text{Close} \), \( \neg \text{Move} \), \( \neg \text{Error} \). We denote Start with St, Close with Cl, Move with Mv and Error with Er. P assigns state \( s_1 \) in \( M \) with \{St, Cl, Mv, Er\}, the state \( s_3 \) in \( M \) with \{St, Cl, \neg Mv, Er\}, the state \( s_5 \) in \( M \) with \{St, Cl, \neg Mv, \neg Er\}, the state \( s_6 \) in \( M \) with \{St, Cl, Mv, \neg Er\} and the state \( s_7 \) in \( M \) with \{St, Cl, Mv, \neg Er\}.

The model is shown below:

![Fig 2. The CTL structure of Elevator Controller](image)

In this figure START represents the Start elevator, Open and Close represents the open door and close the door, RESET is for a new initialization and DONE represents the done moving of elevator.

The faulty process from this graph is the path \( s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4 \). The interpretation is following. In first state is started the elevator \( \{s_1, s_2\} \). In the state \( s_2 \) we observed that have not Cl (that is the door don't close) and the moving is out of order and it pointed some error which is passed from the state \( s_2 \) in the state \( s_3 \) where the door elevator shall be closed. In the state \( s_3 \) there is an error and the elevator moving don’t starts so must be performed a reset for the reestablishment. That is, the control is passed from the state \( s_3 \) to the state \( s_4 \). The process with normal behavior is the case obtained from the original CTL Kripke structure through \( s_1 \rightarrow s_4 \rightarrow s_5 \rightarrow s_6 \rightarrow s_7 \).

Notice that this model don't satisfies the property \( f = \neg EF (\text{St} \land \neg \text{EG} \rightarrow \neg Mv) \) \[4\]. The CTL model updater brings a minimum modification of the Kripke model which satisfies the property \( f \). Firstly, must be analyzed \( f \) in \( AG(\neg (\text{St} \land \text{EG} \rightarrow \neg Mv)) \) for removing the symbol \( \neg \). The translation is doing with the function Update\(\neg\). Then is necessary to check whether each state satisfies \( \neg (\text{St} \land (\text{EG} \rightarrow \neg Mv)) \). This string must be parsed before it is checked. Selecting the \( \text{EG} \rightarrow \neg Mv \) for elevator model means that is selected any path having any state with \( \neg Mv \). Then must be searched the paths in the form \( [s_1, s_2, s_3, s_4, s_1, \ldots] \) and \( [s_5, s_6, s_7, \ldots] \) which represents the connected components loops satisfying \( \text{EG} \rightarrow \neg Mv \). After that are identified all states with \( St \), these are \( \{s_2, s_3, s_6, s_7\} \). Then are selected the states with \( St \) and \( \neg Mv \), these are \( \{s_2, s_3\} \).

Because the \( AG(\neg (\text{St} \land \neg \text{EG} \rightarrow \neg Mv)) \) formula identifies that the model has no both states \( St \) and \( \neg Mv \), is necessary an update of states \( s_2 \) and \( s_3 \) so as they will appear in the updated model. For the execution of Update\(\neg\) function we can use three minimum updates: (case 1) applying Pr2 to remove the connection \( (s_1, s_2) \); (case 2) applying the property Pr5 to removed the state \( s_2 \) and the associate relations e.g. \( (s_1, s_2) \) and \( (s_4, s_3) \); and (case 3) applying Pr3 on the state \( s_2 \) and \( s_3 \). Using case three, the first translation will be from \( \neg (\text{St} \land \text{EG} \rightarrow \neg Mv) \) to \( \neg \text{St} \land \neg \text{EG} \rightarrow \neg Mv \), therefore \( s_2 \) and \( s_3 \) are updated with either \( \neg \text{St} \) or \( \neg \text{EG} \rightarrow \neg Mv \) by the main function CTLUpdate what is dealing with \( \lor \) and with the Update function.

In other words, the new states of \( s_2 \) and \( s_3 \) will be denoted with \( s'_2 \) and \( s'_3 \). The Update\(\neg\)(\( M \), \( \neg (\text{St} \land \text{EG} \rightarrow \neg Mv) \)) function calls the main function CTLUpdate(\( M \), \( \neg \text{St} \)) or CTLUpdate(\( M \), \( \neg \text{EG} \rightarrow \neg Mv \)) for the case \( f_1 \lor f_2 \). We choose the \( \neg \text{St} \) because this is simplest than \( \neg \text{EG} \rightarrow \neg Mv \). In this case is necessary to update the atomic proposition \( St \) in states \( s_2 \) and \( s_3 \) of path \( \pi \) with \( \neg \text{St} \) instead, thus there are no states on path \( \pi \) having the specification \( EF (\text{St} \land \text{EG} \rightarrow \neg Mv) \). That is \( M = (M', \pi) \equiv \neg EF (\text{St} \land \text{EG} \rightarrow \neg Mv) \).

The resulting model is presented in Fig. 3.
where \( s_2' \) is set \( \{ \neg St, \neg Cl, \neg Mv, Er \} \) and the state \( s_3' \) is set \( \{ \neg St, Cl, \neg Mv, Er \} \).

The algorithm will generate one of the three resulting models without specific indication. Because of criteria used they are all minimally changed from the original model.

6 ANTLR

In this section, we will present the first formula \( f_1 = AG(\neg (St \land (EG (\neg Mv))) \) that was checked by our implementation using the ANTLR tools. The ANTLR 3 [7] tools are used for the construction of software instruments as translators, compilers, recognition and parser of static/dynamic programs. The ANTLR is a generator of compilers; it receives as input a grammar, with a precise description of a language, and generates the source code and other auxiliary files for lexer and parser. The source code of our ANTLR grammar must contain the specification of all macro-operations presented in [6]. We “decorate” the grammar for formula language with actions. ANTLR inserts our actions in the generated code for parser, and parser must execute embedded actions after matching the preceding grammar element. This is the mechanism of formula evaluation for a given model.

We define a CTL model checker as an algebraic compiler [6]. The algebraic methodology was used in specification of the ANTLR grammar for CTL formulas. This grammar can be used for the generation and the verification of different models. Using our CTL model checker generator, the user can correct a model in case in which the model was not correctly defined and more important is that the result received is easy to be traced by the user.

An essential contribution of the implementation presented in this section is that it can be used as a model checker generator. For a given model is automatically generated the java code for a verification tool of satisfiability of CTL formulas.

In ANTLRWorks have been implemented all macro-operations of CTL model checker. The program receives as input the model \( M \) where are defined the sets \( S, Rel \) and \( P \). In figure 4 is showed the implementation of the AG macro. We use Java as target language for the implementation of our model checker tool.

Fig.3 The updated model for an elevator control using the primitive P3

Fig.4 Specification of CTL formula grammar in ANTLRWorks

The detailed specification of AG operation is defined in [6], and the specification of and operation as actions in ANTLR grammar is presented in the following:

```java
//Ctlformula→"ag" formula
ctlFormula returns [HashSet set]
   : 'ag' f=formula
   { HashSet rez = new HashSet(); // Z:=∅;
     HashSet rez1 = new HashSet($f.set); // Z':= dctl(t);
     while (!rez.equals(rez1)) // while Z
         { rez.clear(); rez.addAll(rez1); //Z:=Z'
           HashSet tmp = new HashSet();
           boolean include;
           for (int i=0; i<MAX_STARI; i++) // succ(s) ⊆ Z'
             { include = true;
               for (int j=0; j<MAX_STARI; j++) // Z'=dctl(t)
                 if (rel[i][j]==1) //∃ succ(s), s ∈ S
                   if (!(rez1.contains(new Integer(j))))
                     include = false;
                 if (include)
                   tmp.add(new Integer(i));
           rez1.retainAll(tmp); //Z':=Z'
         } // end while
     trace("ctlFormula",6);
```
For checking the first formula $f_1$ of our example, we will use the ANTLR debugging facility to visualize the parse tree and execution of actions of grammar rules, i.e. evaluation of the CTL formula.

The CTL rules set which are directly specified in the syntactic algebra of CTL, denoted by $\text{Sin}_{ctl}$ can be ambiguous, therefore it is necessary that the set of formulas from $\text{Sin}_{ctl}$ to be divided in the non-terminal symbols denoted with $\text{CtlFormula}, \text{Formula}, \text{Factor}, \text{Termen}$ and $\text{Expresie}$. Thus, the defined rules deliver non-ambiguous specification in algebra $\text{Sin}_{ctl}$.

In section 5 we presented as an example the elevator controller. Starting from the original model it was seen that there is a situation in which the $f_1$ formula is not satisfied. This has been verified with our implementation of model checker. The states from our implementation begin from state $s_0$. The result obtained using the ANTLR implementation of the CTL model checker is shown below:

factor
  Move: 5 6 ;
ctlFormula
  notMove: 0 1 2 3 4 ;
ctlFormula
  eg (notMove): 0 1 2 3 ;
expresie
  ( Start and (eg(notMove)) ) : 1 2 ;
termen
  not ((Startand(eg(notMove)))): 0 3 4 5 6 ;
ctlFormula
  ag (not((Startand(eg(notMove))))): ;

For the original model has been found that the $f_1$ formula was not satisfied in all the model states. We proved in section 5 that if we are applying the algorithm for CTL model update, the states $s_2$ and $s_3$ will be updated. Thus was obtained a new model with two new states $s_2$ and $s_3$. Using our CTL model checker generator on the new model has shown that the $f_1$ formula was satisfied in all the new states:

factor
  Start: 4 5 ;
factor
  Move: 5 6 ;
termen
  not Move: 0 1 2 3 4 ;
ctlFormula
  eg (notMove): 0 1 2 3 ;

References: