Abstract—This paper presents an efficient procedure to perform model checking of a concurrent process against a temporal logic formula, through the checking of a simulation between the description of the system and of the formula in the same formalism. The approach is applied, as an example, to processes defined through a specification language very compact, the well-known Calculus of Communicating Systems (CCS) defined by Milner, but it can be applied also to different languages as Lotos and CSP. The algorithm to explore the search space for the simulation problem is based on a greedy technique. The experiments show that a considerable reduction of both state space size and time can be achieved with respect to traditional model checking algorithms.

Keywords—Model checking; heuristic searches; temporal logic.

I. INTRODUCTION

A very used method to verify concurrent and distributed systems is model checking [1]. It is an automated technique that, given a finite-state model of a system and a formal property, systematically checks whether this property holds for (a given state in) that model.

Equivalence checking is the process of determining whether two systems are related to each other according to some mathematically defined notion of equivalence. There are several kinds of equivalences, starting from the two equivalences defined by Milner for CCS processes: the strong equivalence that means that the processes are not distinguishable; the weak equivalence that defines a notion of observability for the processes and requires that equivalence be defined only with respect to the observable actions. Other equivalences exploit a different point of view on the observability, for example, equivalences were defined based on ignoring non-$\tau$ actions irrelevant for the expected system behavior [2], [3].

It is possible to use equivalence checking to realize model checking of a system, through comparison of the system model against the model derived from the formula; automata theoretic approach follows that way, by performing the product between the automaton representing the system and the automaton representing the formula (actually the complement of the formula): an empty result of the product signifies the satisfaction of the formula. However, this formal method cannot be scaled as easily with the increasing complexity of the systems, since at least the two automata have to be built to perform their product; moreover the construction of the automaton of the formula is not so straightforward for branching temporal logics.

This paper proposes to perform model checking of a concurrent system against a formula by applying heuristic search techniques on AND/OR graphs to the problem of verifying whether a concurrent system simulates another one (actually, one is the system to be verified, the other corresponds to the formula to be satisfied). Heuristic search is one of the classical techniques in Artificial Intelligence and has been applied to a wide range of problem-solving tasks including puzzles, two player games and path finding problems. In heuristic searches some utility or cost can be assigned to each state to guide the search by suggesting the next state to expand; in this way the most promising paths are considered first. There are several heuristic search algorithms for AND/OR graphs: a main difference among them is due to the fact of considering cyclic AND/OR graphs, or not cyclic AND/OR graphs. In the latter class the automaton of the formula is not so straightforward for branching temporal logics.

In any case, the AND/OR graph is expanded incrementally during the execution of the algorithm, starting from the initial node; the heuristic function assigns a cost to each node not yet considered, and the optimality of the solution can be guaranteed by the property of admissibility of the function, i.e., the heuristic never overestimate the distance to the goal. Optimality is not really an issue, from the point of view of the efficiency of the equivalence checking, for example: it is sometime better to find as soon as possible a solution than to repeatedly discard solutions until the optimum is found. Discarding optimality, a greedy approach can be followed: at any step, the next state to be expanded is chosen only on the basis of the foreseen cost of that expansion.

To model check the examples we show in this work a poorly informative heuristic function is used and, for this reason, it is not shown in detail here. The interesting thing is that, also in this case, the method avoids the construction of the entire model of the system and produces a very good result in terms of state space size and time with respect to traditional model checking algorithms, for example those defined inside the Concurrency WorkBench of the New Century (CWB-NC) [6]. The heuristic function can be refined so producing hopefully even better results. We consider concurrent systems (and formulae) specified by means of the Calculus of Communicating Systems (CCS)
II. Preliminaries

A. Calculus of Communicating Systems

Below we present a brief overview of the main features of Milner’s Calculus of Communicating Systems (CCS)\[2\]. The syntax of processes is the following:

\[ p ::= \text{nil} \mid x \mid \alpha.p \mid p + p \mid p|p \mid p;L \mid p[f] \]

where \( \alpha \) ranges over a finite set of actions \( A = \{\tau, a, \pi, b, \ldots\} \). The action \( \tau \in A \) is called the \textit{internal action}. The set of \textit{visible actions}, \( V \), ranged over by \( l, l', \ldots \) is defined as \( A - \{\tau\} \). Each action \( l \in V \) (resp. \( \bar{l} \in \bar{V} \)) has a complementary action \( \bar{l} \) (resp. \( l \)). The restriction set \( L \), in the processes of the form \( p|L \), is a set of actions such that \( L \subseteq V \). The relabeling function \( f \), in processes of the form \( p[f] \), is a total function, \( f: A \rightarrow A \), such that the constraint \( f(\tau) = \tau \) is respected. The constant \( x \) ranges over a set of constant names: each constant \( x \) is defined by a constant definition \( x \mathrel{\triangleright} p \), where \( p \) is called the \textit{body} of \( x \). We denote the set of processes by \( \mathcal{P} \). The semantics of a process \( p \), defined in Table 1, is given by a set of conditional rules describing the transition relation of the automaton corresponding to the behavior expression defining \( p \), called the \textit{standard transition system} for \( p \) and denoted by \( \mathcal{S}(p) \).

Many equivalence relations have been defined on CCS processes; they are based on the notion of bisimulation between states of the related transition systems. In the following we consider the \( \rho \)-equivalence introduced in \[2\] formally characterizing the notion of “the same behavior with respect to a set \( \rho \) of actions”. The precise definition of \( \rho \)-simulation uses the transition relation \( \mathrel{\longrightarrow}_\rho \), parametric with respect to \( \rho \subseteq A \), which ignores all non-interesting actions (i.e. those in \( A - \rho \)).

\begin{definition} \text{Let} \( p \) and \( q \) be two CCS processes and \( \rho \subseteq A \) a set of actions, we define the relation \( \mathrel{\longrightarrow}_\rho \subseteq \mathcal{P} \times \rho \times \mathcal{P} \) \text{in the following way:}

\[ \forall \alpha \in \rho:\ p \mathrel{\longrightarrow}_\rho q \iff p \mathrel{\delta_\alpha} q, \text{ where } \delta \in (A - \rho)^* \]

By \( p \mathrel{\longrightarrow}_\rho q \) we express the fact that it is possible to pass from \( p \) to \( q \) by performing a (possibly empty) sequence of actions not belonging to \( \rho \) and then the action \( \alpha \) in \( \rho \). Note that \( \mathrel{\longrightarrow}_A = \mathrel{\longrightarrow} \).
\end{definition}

\begin{definition} \( (\rho\text{-simulation}) \text{ Let} \( p \) and \( q \) be two CCS processes and let \( \rho \subseteq A \).

- A \( \rho \)-simulation, \( \mathcal{R} \), is a binary relation on \( \mathcal{P} \times \mathcal{P} \) such that \( p\mathcal{R}q \) means that:

\[ p \mathrel{\rightarrow}_\rho q' \text{ implies } q \mathrel{\rightarrow}_\rho q' \text{ with } p'\mathcal{R}q' \]

- \( q \)-\( \rho \)-simulates \( p \) (\( p \prec_\rho q \)) if there exists a \( \rho \)-simulation \( \mathcal{R} \) containing the pair \( (p,q) \).
\end{definition}

Note that the \( \rho \)-simulation, with \( \rho = V \) does not coincide with the weak simulation. Suppose: \( p = a.nil + \tau.nil \) and \( q = a.nil \). It holds that \( q \mathcal{V} \)-simulates \( p \), but \( q \) does not weak simulate \( p \). On the other hand, if \( p = a.nil + a.(c.nil + \tau.nil) \) and \( q = a.(c.nil + \tau.nil) \), \( q \) weak simulates \( p \) but it is not true that \( q \mathcal{V} \)-simulates \( p \). In the following, with \( \rho \)-derivatives of a CCS process \( p \), we denote the set of all processes reachable from \( p \) by \( p \mathrel{\longrightarrow}_\rho \).

B. SHML logic

To the purpose of explaining our methodology without too much technicality, system properties will be defined through the so-called Selective Hennessy-Milner Logic (SHML) instead of the full selective mu-calculus, introduced by Barbuti et al. \[2\]. However, SHML is more expressive than the Hennessy-Milner logic \[2\] because of the intrinsic recursion of the selective operators. Given a set \( A \) of actions, with \( K, R \subseteq A \), the SHML logic is the set of formulae so defined:

\[ \varphi ::= \top \mid \bot \mid \mathcal{F} \mathcal{F} \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \langle K \rangle R \varphi \mid \langle K \rangle R \varphi . \]

The satisfaction of a formula \( \varphi \) by a state \( s \) of a transition system, written \( s \models \varphi \), is defined as follows: each state satisfies \( \top \) and no state satisfies \( \bot \); a state satisfies \( \varphi_1 \wedge \varphi_2 \) (\( \varphi_1 \vee \varphi_2 \)) if it satisfies \( \varphi_1 \) (and \( \varphi_2 \)), \( \langle K \rangle R \varphi \) and \( \langle K \rangle R \varphi \) are the selective modal operators: \( \langle K \rangle R \varphi \) is satisfied by a state which, for every performance of a sequence of actions not belonging to \( R \cup K \), followed by an action in \( K \),
which can evolve to a state obeying $\varphi$. \((K)_R \varphi\) is satisfied by a state which can evolve to a state obeying $\varphi$ by performing a sequence of actions not belonging to $R \cup K$, followed by an action in $K$. The selective modal operators $\langle K \rangle_R \varphi$ and $[K]_R \varphi$ substitute the standard modal operators $\langle K \rangle \varphi$ and $[K] \varphi$. The basic characteristic of the SHML is that each formula allows us to immediately point out the parts of the transition system that do not alter the truth value of the formula itself. More precisely, the only actions relevant for checking a formula are the ones explicitly mentioned by the selective modal operators used in the formula itself. Thus, the result of checking the formula is independent from all other actions. In this paper we exploit this information to efficiently check $\rho$-simulation. A transition system $T$ satisfies a SHML formula $\varphi$, written $T \models \varphi$, if and only if $p \models \varphi$, where $p$ is the initial state of $T$. A CCS process $p$ satisfies $\varphi$ if $S(p)$ satisfies $\varphi$. The precise definition of the satisfaction of a SHML formula $\varphi$ by a state of a transition system is given in Table II.

### C. Heuristic search: AND/OR Graphs

In this section, the general concept of AND/OR graphs to represent a search problem and the Greedy algorithm defined in [9] to visit AND/OR graphs and to find a suitable solution are presented.

An AND/OR graph $G$ is a directed graph with a special node $s$, called the start (or root) node, and a nonempty set of terminal leaf nodes denoted as $t, t_1, \ldots$. The start node $s$ represents the given problem to be solved, while the terminal leaf nodes correspond to subproblems with known solutions. The nonterminal nodes of $G$ are of three types: OR, AND, and nonterminal leaf. An OR node is solved if one of its immediate subproblems is solved, while an AND node is solved only when every one of its immediate subproblems is solved. A nonterminal leaf node has no successors and is unsolvable.

Given an AND/OR graph $G$, a solution of $G$ is represented by an AND/OR subgraph, called solution (sub)graph of $G$ with the characteristics given below.

**Definition 2.3:** A finite subgraph $D$ of an AND/OR graph $G$ is a solution subgraph of $G$ if:

1. the start node of $G$ is in $D$;
2. if $n$ is an OR node in $G$ and $n$ is in $D$, then exactly one of the immediate successors of $n$ in $G$ is in $D$;
3. if $n$ is an AND node in $G$ and $n$ is in $D$, then all the immediate successors of $n$ in $G$ are in $D$;
4. every non cyclic maximal path in $D$ ends in a terminal leaf node.

In most domains the AND/OR graph $G$ representing the problem is unknown in advance, so it cannot be supplied explicitly to the search algorithm. Instead, a distinction is drawn between the explicit graph $G'$ and the implicit graph $G$. The implicit graph $G$ is specified by a start node $s$ and a successor function. The search algorithm works on the explicit graph $G'$, which initially consists of the start node $s$ only. The start node is then expanded using the successor function of $G$ to obtain a set of nodes and arcs that are added to $G'$. At any instant, the explicit graph $G'$ has a number of nodes yet to be expanded, called rip nodes. The search algorithm chooses one of these tip nodes for expansion. In this manner, more and more nodes and arcs get added to the explicit graph, until it finally has one or more solution graphs as subgraphs. One of these solution graphs is then output by the search algorithm. In this paper we use the Greedy algorithm for the heuristic search in AND/OR graphs that has been proposed in [9].

### III. The Approach

In this section the heuristic-approach to deduce a property satisfaction by checking $\rho$-simulation is presented. From now on, to simplify the presentation, SHML formulae containing only $[K]_R$ modal operators and $\land$ operators are considered: they are referred to as $\llbracket \cdot \rrbracket$-formulae, expressing safety properties. In particular, to verify if the property $\varphi$ holds for the CCS process $p$, we (1) translate $\varphi$ into a CCS process, say it $f$; and (2) check that $p$ is simulated by $f$ according to the notion of $\rho$-simulation, with $p$ the set of actions occurring in $\varphi$. We analyse each step deeply.

#### A. Formula-to-CCS operator

We describe the formula-to-CCS operator: two special processes, $\bot$ and $\top$, are used with the following property: $\forall p \in P, p \prec_\rho \top$ and $p \not\prec_\rho \bot$.

The $T$ operator, given a $\llbracket \cdot \rrbracket$-formula and a set of actions, returns a special CCS process, called sCCS, that is a CCS process that can terminate either with a constant or with one of the special processes $\bot$ and $\top$.

**Definition 3.1 (The formula-to-CCS operator $T$):** Let $\varphi$ be a $\llbracket \cdot \rrbracket$-formula. The operator $T$ is inductively defined on the structure of $\varphi$.

\[
T(\text{ff}) = \{\bot\}, \quad T(\text{tt}) = \{\top\},
\]
\[
T([K]_R \varphi) = \{x \mid x \overset{\text{def}}{=} \sum_{k \in K} k.p + \sum_{a \in A - [K \cup R]} a.x + \sum_{b \in R} b.T, \quad p \in T(\varphi)\},
\]
\[
T(\varphi_1 \land \varphi_2) = T(\varphi_1) \cup T(\varphi_2).
\]
To check whether \( p \models \varphi \) with \( \text{alph}(\varphi) = \{ \} \), \( \mathcal{T} \) builds the sCCS processes corresponding to \( \varphi \): \( p \) \( \rho \)-simulates all of them if and only if \( p \models \varphi \). If \( \varphi = \text{ff} \), the set built by \( \mathcal{T} \) contains only the special process \( \bot \), which \( \rho \)-simulates no process; while if \( \varphi = \text{tt} \) the set contains only the process \( \top \) which \( \rho \)-simulates any process. When \( \varphi = [K]_R \varphi' \), for each \( q' \in \mathcal{T}(\varphi') \), a new constant \( x \) is defined as the summation of \( \alpha \) such that:

- if \( \alpha \in K \), the process \( x \) performs \( \alpha \) moving to \( q' \);
- if \( \alpha \in A - (K \cup R) \), the process \( x \) performs \( \alpha \) and then returns to itself;
- if \( \alpha \in R \), the process \( x \) performs \( \alpha \) moving to then \( \top \).

The idea is that \( x \) represents a state in which only actions in \( A - (K \cup R) \) (i.e., actions that have no effect on the satisfiability of formula \( \varphi \)) have already been performed. Any action in \( R \) from this state leads to vacuous satisfaction of \( \varphi \), while any action in \( K \) leads to the check for the subformula \( \varphi' \).

B. \( \rho \)-simulation checking

The second step of our approach consists of deducing whether the CCS process \( p \) satisfies the formula \( \varphi \). This is obtained by checking whether all the sCCS processes obtained by the translation of \( \varphi \) \( \rho \)-simulate \( p \). The following theorem states the correctness of this step.

**Lemma 3.1:** Let \( p \) be a CCS process and \( \varphi \) be a []-formula. For all \( q \in \mathcal{T}(\varphi) \), \( p \prec_{\text{alph}(\varphi)} q \) implies \( p \prec_{\rho} q \) for all \( \text{alph}(\varphi) \subseteq q \subseteq A \).

**Theorem 3.1:** Let \( p \) be a CCS process and \( \varphi \) a []-formula. It holds that: \( (\forall q \in \mathcal{T}(\varphi), p \prec_{\text{alph}(\varphi)} q) \iff p \models \varphi. \)

**Proof.** The proof is by induction on the structure of the formula \( \varphi \).

To check \( \rho \)-simulation we use an heuristic approach defined in the next section

C. AND/OR graph for \( \rho \)-simulation checking

Let \( p \) be a CCS finite process and \( q \) a finite sCCS process: to check the requirements of the \( \rho \)-simulation (Definition 2.2) an AND/OR graph is built representing the fact that \( p \) and \( q \) move in alternating turns as long as they can perform the same move. Thus, the AND/OR graph has a solution iff \( p \prec_{\rho} q \), since the construction halts with nonterminal leafs only when there is a move of \( p \) that cannot be matched by \( q \).

In order to apply the algorithm described in Section II-C, the operators used to expand a node have been defined. In the following, we denote by \( G(p, q) \) the implicit AND/OR graph built from \( p \) and \( q \) and the transition relation \( \rightarrow_{s} \). The nodes of \( G(p, q) \) are 3-uples \( \langle r, s, \gamma \rangle \) where \( r \) is a \( \rho \)-derivative of \( p \); \( s \) is a \( \rho \)-derivative of \( q \) or the processes \( \top \) and \( \bot \); and \( \gamma \in \{ \lambda \} \cup A \).

\[ \text{op}_1 \begin{cases} \{ (y', \alpha) \mid p \xrightarrow{\alpha} x \} = \{(p_1, \alpha_1), \ldots, (p_n, \alpha_n)\} \neq \emptyset \\ (p, q, r) \rightarrow (p_1, q, \alpha_1) \land \cdots \land (p_n, q, \alpha_n) \end{cases} \]

\[ \text{op}_2 \begin{cases} \{ y' \mid q \xrightarrow{\beta} y' \text{ and } \beta = \alpha \} = \{q_1, \ldots, q_n\} \neq \emptyset \\ (p, q, \alpha) \rightarrow (p, q_1, \lambda) \lor \cdots \lor (p, q, \alpha) \rightarrow (p, q_n, \lambda) \end{cases} \]

**Table III**

**GENERAL OPERATORS.**

We assume \( \lambda \notin A \) and \( s \) be a sCCS process. When \( \gamma = \lambda \) it is the turn of \( r \) to move; when \( \gamma = \alpha \) then \( s \) has to move performing the action \( \alpha \), i.e. the action that the process \( r \) has performed in the previous turn.

Let \( n = \langle r, s, \gamma \rangle \) be a node of \( G(p, q) \). Node \( n \) is an AND node, OR node, terminal or nonterminal leaf according to the value of \( \gamma \) and the following rules:

- when \( \gamma = \lambda \), node \( n \) is an AND node if \( r \neq s \) and \( s \notin \{ \bot, \top \} \);
- when \( \gamma = \alpha \), node \( n \) is an OR node if \( s \) has a \( \alpha \)-transition, otherwise it is a nonterminal leaf;
- when \( \gamma = \lambda \), node \( n \) is a terminal leaf, if \( r = s \) or \( s = \top \);
- when \( \gamma = \lambda \), node \( n \) is a nonterminal leaf if \( s = \bot \).

The start node of \( G(p, q) \) is \( \langle p, \lambda, \lambda \rangle \). The successor function is given by the operators in Table III. The operators generate the outgoing arcs and the successor nodes of each node. The operators with the form

<table>
<thead>
<tr>
<th>premise</th>
<th>( n \rightarrow n_1 ) and ( n \rightarrow n_m )</th>
</tr>
</thead>
</table>

where \( \text{premise} \) is the antecedent, possibly empty, of the rule, generate all the outgoing arcs and successor nodes of the AND node \( n \). On the other hand, the operators with the form

<table>
<thead>
<tr>
<th>premise</th>
<th>( n \rightarrow n_1 ) or ( n \rightarrow n_m )</th>
</tr>
</thead>
</table>

generate all the outgoing arcs and successor nodes of the OR node \( n \).

The rule \( \text{op}_1 \) points out the possible moves (\( \alpha_i \)) of \( p \) when \( \gamma = \lambda \); in this way an AND node can be connected with its successor nodes in the graph, all such successors have \( \gamma = \alpha_i \); roughly speaking if \( p \) can move performing an action \( \alpha_i \) and reaches the process \( p_i \) then it is the turn of \( q \) to move with the same action \( \alpha_i \). In rule \( \text{op}_2 \), \( q \) must simulate the action \( \alpha \) performed by \( p \); in this case an OR node can be connected with its successor nodes in the graph, all such successors have \( \gamma = \lambda \) i.e. nodes that can be transformed only through the operator \( \text{op}_1 \) like the initial node.

It can be proved that finding a solution of \( G(p, q) \) is equivalent to checking whether \( p \prec_{\rho} q \). In this preliminary work, since the idea is to evaluate goodness of the application of
heuristic search to check properties through simulation, we use a heuristic function always equal to zero. As reflected in the experiments we made, this non informative heuristic has the advantage of a lower computational cost, while retaining the reduction of state space due to the greedy algorithm.

IV. EXPERIMENTAL RESULTS

In this section we present and discuss our experience with using the tool, called GreASE (Greedy Algorithm for System Equivalence); this tool applies the presented approach to model checking (via ρ-simulation) to several well-known CCS processes. The aim is to evaluate the performances of the approach and compare it against the CWB-NC. Experiments were executed on a 64bit, 2.67 GHz Intel i5 CPU equipped with 8 GiB of RAM and running Gentoo Linux. The tool is freely available for download at the url: //www2.ing.unipi.it/~a080224/grease. For the evaluation, a sample of well known systems is selected from the literature. The results are shown in Table [IV] where the number of generated nodes and the time (expressed in sec) resulting from model checking the selected case studies are reported.

Railway system (crail): we applied our tool to the system specification given in [10]. This system describes the British Rail’s Solid State Interlocking which is devoted “to adjust, at the request of the signal operator, the setting of signal and points in the railway to permit the safe passage of trains”. On this system we checked the following formula ϕ_1 expresses the property that any multicast delivered by the protocol cannot deliver the i-th segment (2 ≤ i ≤ 10) to the receiving client before having delivered the (i−1)-th one. ϕ_2 means that, after accepting a data packet of length 10, the protocol cannot issue an ok confirmation (action in(ok)) to the sending client unless the last segment of the packet (action out(d10,ok)) has been delivered to the receiving client.

Multicast Protocol for Mobile Computing (MPMC): a protocol for reliable multicast in distributed mobile systems, presented in [12]. On this system we checked the following two formulae:

ϕ_1 = [deliver(m, msg)_[rend(msg)]]_0 [deliver(m, msg)]_0
ϕ_2 = [deliver(m, msg)]_0 [deliver(m, msg)]_0

ϕ_1 expresses the property that any multicast delivered by a group member has been originated by a group member; and ϕ_2 expresses the property that no group member delivers duplicate multicasts, i.e. duplicates are discarded. Both properties are true.

These examples provide some experimental evidence of the reduction of the state space that may result when applying the presented methodology instead of the standard algorithm used by the CWB-NC. Note that in all cases we obtain great reductions both in time and in state space.

V. CONCLUSION AND RELATED WORK

A method has been proposed for model checking safety properties for concurrent systems described in CCS. The novel contributions of the work are the following:

- application of algorithms taken from heuristic search environment for the ρ-simulation checking;
- model checking safety properties through ρ-simulation.

The most challenging task when applying automated model checking in practice is to conquer the state explosion problem. Hence, simulation algorithms with minimal space complexity are of particular interest. Two algorithmic families can be considered: one is based on the refinement principle, that is, given an initial partition, find the coarsest partition stable with respect to the transition relation, see

<table>
<thead>
<tr>
<th>case study</th>
<th>our approach</th>
<th>CWB-NC</th>
</tr>
</thead>
<tbody>
<tr>
<td>crail (ϕ_1)</td>
<td>1197 0.020 5628 0.564</td>
<td></td>
</tr>
<tr>
<td>Mutual8 (ϕ_1)</td>
<td>644 0.507 6912 1.616</td>
<td></td>
</tr>
<tr>
<td>Mutual8 (ϕ_2)</td>
<td>387 0.049 6912 1.410</td>
<td></td>
</tr>
<tr>
<td>Mutual10 (ϕ_1)</td>
<td>2564 7.002 17408 6.885</td>
<td></td>
</tr>
<tr>
<td>Mutual10 (ϕ_2)</td>
<td>1539 0.322 17408 5.448</td>
<td></td>
</tr>
<tr>
<td>BRP (ϕ_1)</td>
<td>40 0.025 759 0.262</td>
<td></td>
</tr>
<tr>
<td>BRP (ϕ_2)</td>
<td>15 0.044 759 0.189</td>
<td></td>
</tr>
<tr>
<td>MPMC (ϕ_1)</td>
<td>19 0.004 13803 3.598</td>
<td></td>
</tr>
<tr>
<td>MPMC (ϕ_2)</td>
<td>1611 1.439 13803 3.609</td>
<td></td>
</tr>
</tbody>
</table>

Table IV
RESULTS FOR CCS SYSTEMS

The formula ϕ_1 means that, after accepting a data packet of length 10 (action in(d1_d10)), the protocol cannot deliver the i-th segment (2 ≤ i ≤ 10) to the receiving client before having delivered the (i−1)-th one. ϕ_2 means that, after accepting a data packet of length 10, the protocol cannot issue an ok confirmation (action in(ok)) to the sending client unless the last segment of the packet (action out(d10,ok)) has been delivered to the receiving client.
for example the algorithm proposed by Paige and Tarjan in [13]. The other family of algorithms is based on a Cartesian product traversal from the initial state [14]. These algorithms are both applied on the whole state graph, and require an explicit enumeration of such space so producing the well-known state explosion problem.

The algorithms in [15], [16] consider a problem with \( N \) states, and \( S \) simulation equivalence classes: the space complexity of both algorithms is \( O(S^2 + N\log N) \).

Recently great interest has grown in combining model checking and heuristics to guide the exploration of the state graph of a system. In the domain of software validation, the work of Yang and Dill [17] is one of the original ones. They enhance the bug-finding capability of a model checker by using heuristics to search the states that are most likely to lead to an error. In [18] heuristics have been used for real-time model checking in UPPAAL. In [19] heuristic search has been combined with on-the-fly techniques, while in [20] with symbolic model checking. Previous works, as for example [21], [22], [23], [24], used heuristic search to accelerate verification.

The presented approach starts from representing the system and the logic formula in the same formalism and, then, tries to state if the formula is satisfied by the system by proving that the system behavior is simulated the formula behavior. In [20] Vardi and Wolper describe an automata-theoretic approach to verify finite-states systems: the idea underlying this approach is that, for any linear temporal logic (LTL) formula, one can construct an automaton that accepts precisely the computations satisfied by the formula. Thus, if \( p \) is a system and \( \varphi \) a LTL formula, \( p \) meets \( \varphi \) if every infinite word generated by \( p \), viewed as a finite-state generator, is accepted by \( \varphi \), viewed as a finite state acceptor. This reduces the model checking problem to a purely automata-theoretic problem: the problem of determining if the automaton \( p \cap \varphi \) is empty, i.e. if it accepts no word. On the other hand, for branching temporal logic, automata-theoretic techniques have long been thought to introduce an exponential penalty, making them essentially useless. But, Bernholtz and Grumberg in [27] suggested that an automata-theoretic approach to branching-time model checking can be based on the concepts of amorphous automata and simultaneous trees, and they showed that CTL model checking is linearly reducible to the acceptance of a simultaneous tree by an amorphous automaton, problem that can be solved in quadratic running time. Although this constitutes a meaningful first step towards applying automata-theoretic techniques to branching-time model checking, it is not quite satisfactory, above all for the complexity of the resulting algorithm which is quadratic in both the size of the formula and the size of the system.

Our attempt is in this direction: the heuristic search methodology avoids the construction of both automata and the results in terms of space and running time are encouraging. The next step consists in managing full selective mu-calculus formulae and in defining a more informative heuristic function.

Other approaches have been developed to solve or reduce the state explosion problem for model checking. Among them, reduction techniques based on symbolic model checking techniques [28], partial order techniques [29], [30], compositional techniques [31], [32], and abstraction approaches [33], [34]. Moreover, all the above approaches has been successfully applied in several fields, like for example clone detection [35], [36], analysis of social networks [37], incremental construction of complex systems [38], secure information flow in concurrent programs [39], banking industry [40], [41].

REFERENCES


