The Supervisor Synthesis Problem for Unrestricted CTL is $\mathcal{NP}$-complete

Marco Antoniotti† Bud Mishra‡*

November 1995

† International Computer Science Institute
Berkeley, CA, USA
marcoxa@icsi.berkeley.edu

‡ Courant Institute of Mathematical Sciences
New York University
New York, NY, USA
mishra@cs.nyu.edu

Abstract

The problem of restricting a finite state model (a Kripke structure) in order to satisfy a set of unrestricted CTL formulæ is named the *"Unrestricted* CTL *Supervisor Synthesis Problem"* . The finite state model has the characteristics described in [RW87b], that is, its transitions are partitioned between *controllable* and *uncontrollable* ones. The set of CTL formulæ represents a specification of the *desired behavior* of the system, which may be achieved through a *control action*. This note shows the problem to be $\mathcal{NP}$-complete.

*This TR also appears as TR-95-062 at ICSI.

1 Introduction

This note contains a proof of the $\mathcal{NP}$-completeness for the *Unrestricted* CTL *Supervisor Synthesis Problem* as defined in [Ant95].

The problem is defined in terms of a *model* of the *unrestrained behavior* of a discrete system or *plant* $\mathcal{P}$ (typically represented as a *Finite State Machine*) and a *specification* $\mathcal{S}$ of the *desired behavior* of the system, given as a set of CTL formulæ [Eme90].

The model of the system is given following the conventions established by Ramadge and Wonham [RW87b]. The plant is modeled as a *generator* $\mathcal{P} = (\Sigma, S, \delta, s_0)$. In particular the set of transitions (or events) Σ of the plant is partitioned into two subsets of *controllable* and *uncontrollable* events, Σ_c and $\Sigma_u = \Sigma - \Sigma_c$. Controllable events can be enabled or disabled by a *supervisor*, while uncontrollable ones are always enabled.

The Supervisor Synthesis problem for CTL specifications consists in searching for ways to enable and disable the transitions of a finite state model in order to satisfy the specification. I.e. the problem is to *synthesize* map $\varphi : S \times \Sigma \rightarrow \{0, 1\}$ which given a *state* and a *transition* will tell us whether the transition is enabled or not.

The choice of the map φ must be such that the *constrained plant* becomes a *model*, in the modal logic sense, of the CTL specification[1]. The notation

$$\Lambda[\mathcal{P}], \varphi, s \; || \!\models \mathcal{S}$$

indicates that the language generated by $\mathcal{P}$ $(\Lambda[\mathcal{P}])$, under the supervision of φ satisfies the specification $\mathcal{S}$. (The abbreviated notation $\Lambda[\mathcal{P}], s \; ||\!\models \mathcal{S}$ will also be used as well as the standard one $\mathcal{P}, s \models \mathcal{S}$. The meaning should always be clear from the context.)

1.1 An Introduction to the CTL Supervisor Synthesis Problem

This section contains a brief and intuitive example of the Supervisor Synthesis Problem (for CTL specifications).

Consider the simple representation of the behavior of two pieces of equipment M_1 and M_2 in Figure 1.1. M_1 and M_2 have to request and use a given resource. The interleaving composition $M_1 || M_2$ yields the 9-state machine also shown in Figure 1.1.

Using the standard CTL semantics (e.g. as described in [Eme90]) it is immediate to realize that the combined behavior of M_1 and M_2 satisfies the following *liveness* condition:

if a machine requests the resource, it will eventually use it.

This condition is translated in CTL as

$$\mathbf{AG}(\text{Request}_i \Rightarrow \mathbf{AF}(\text{Use}_i)), \quad \text{for } i \in \{1, 2\}.$$

However, the following *mutual exclusion* property (a *safety* property) is not satisfied:

no more than one machine can use the resource at a time.

[1] The definitions and the overall construction used here are different from Ramadge and Wonham's. The reasons behind such differences are discussed in [Ant95], but are not relevant in the current discussion.

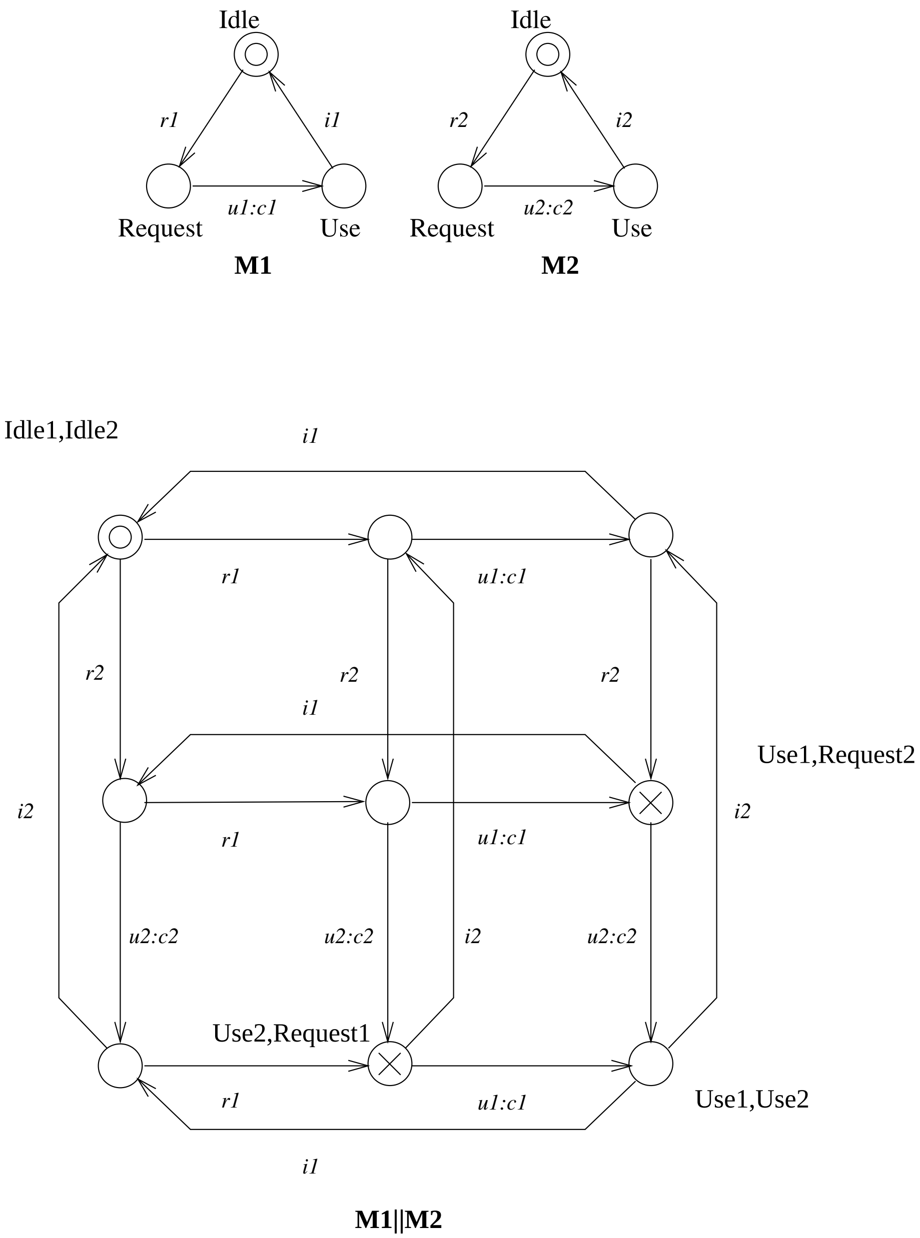

Figure 1: *The figure depicts the behavior of the two machines M_1 and M_2 and their resulting shuffle product $M_1||M_2$. The transitions u_1 and u_2 are* controllable *(via the "control variables" c_1 and c_2). It is immediate to note that the state labeled $[Use_1, Use_2]$ is reachable from the start state of the composed system. A control action is therefore needed at the states marked by $\times$ in order to satisfy $\mathbf{AG}(\neg(Use_1 \wedge Use_2))$.*

This condition has the following CTL form.

$$\mathbf{AG}(\neg(\mathrm{Use}_1 \wedge \mathrm{Use}_2)).$$

A supervisor that makes the model satisfy the mutual exclusion specification has

$$\begin{aligned}
\varphi([\mathrm{Request}_1, \mathrm{Use}_2])(u_1 : c_1) &\leftarrow 0, \\
\varphi([\mathrm{Request}_1, \mathrm{Use}_2])(i_2) &\leftarrow 1, \\
\varphi([\mathrm{Use}_1, \mathrm{Request}_2])(u_1 : c_1) &\leftarrow 0, \\
\varphi([\mathrm{Use}_1, \mathrm{Request}_2])(i_1) &\leftarrow 1.
\end{aligned}$$

All the other transitions are enabled, i.e. φ is set to 1. The finite state machine obtained from $M_1||M_2$ by disabling the transitions u_i in states $[\mathrm{Request}_1, \mathrm{Use}_2]$ and $[\mathrm{Use}_1, \mathrm{Request}_2]$ is a model for the mutual exclusion specification.
For this case, a partial trace of the Control-D system is

```
       ...
Maps at state (REQUEST1 REQUEST2) :
  (AG (NOT (STATE (USE1 USE2))))
unconstrained (null)
Maps at state (REQUEST1 USE2) :
  (AG (NOT (STATE (USE1 USE2))))
     : [U1 -|-> ((USE1 USE2))] enabled? NIL
     : [I2 ---> ((REQUEST1 IDLE2))] enabled? T
Maps at state (USE1 IDLE2) :
  (AG (NOT (STATE (USE1 USE2))))
unconstrained (null)
Maps at state (USE1 REQUEST2) :
  (AG (NOT (STATE (USE1 USE2))))
     : [U2 -|-> ((USE1 USE2))] enabled? NIL
     : [I1 ---> ((IDLE1 REQUEST2))] enabled? T
Maps at state (USE1 USE2) :
  (AG (NOT (STATE (USE1 USE2))))
blocked
       ...
```

The notation is a prefix rendering of the standard CTL syntax and the terms (STATE s) denote states in the composed machine.

1.1.1 Algorithms for the Supervisor Synthesis Problem.

Two algorithms for the original Supervisor Synthesis problem specified by Ramadge and Wonham can be found in [RW87a]. An algorithm for the Supervisor Synthesis problem for *restricted* CTL specifications is described in [Ant95]. This last algorithm is based on a *labeling scheme*, which was used for the *model checking* algorithm by Emerson and Clarke [CES86].

2 $\mathcal{NP}$-completeness of the CTL Supervisor Synthesis Problem

In [Ant95] the search for tractable algorithms for the CTL Supervisor Synthesis Problem leads to a restriction on the form of CTL formulæ admitted as specifications of the desired behavior of a plant.

In particular, it was recognized that disjunctions of arbitrary CTL formulæ, do not admit a simple algorithm for the synthesis of a supervisor. The resulting implementation in the Control-D system, therefore either allowed only disjunctions with only one *path* disjunct, or used some heuristics to synthesize the supervisor.

The rationale behind this choices stemmed from the problematic example discussed in Section 2.1, which eventually lead to the construction of a reduction from SAT [GJ79] which proves that the supervisor synthesis problem for arbitrary CTL specifications is $\mathcal{NP}$-complete.

2.1 Disjunction Problem

Figure 2 contains a problematic example for the temporal logic supervisor synthesis problem with disjunction. A supervisor is needed that satisfies the formula

$$\mathbf{AX}(\mathbf{AG}(p_1)) \vee \mathbf{AX}(\mathbf{AG}(p_2)). \tag{1}$$

for the plant language L of Figure 2.

The language L generated by the automata in fig. 2 is simply

$$L = \alpha_1\beta_1^* + \alpha_2\beta_2^* + \alpha_3(\beta_1^* + \beta_2^*). \tag{2}$$

The controllable events are α_1, α_2 and α_3.
Suppose now that the following assignment of propositions to states are given.

$$\begin{aligned}
\Pi(s_1) &= \{p_1\},\\
\Pi(s_2) &= \{p_1, p_2\},\\
\Pi(s_3) &= \{p_2\}.
\end{aligned}$$

With this assignment it is possible to start labeling the states with CTL formulæ in the following way

$$\begin{aligned}
\beta_2^*, s_3 &\mid\models \{\mathbf{AG}(p_2), \mathbf{AX}(\mathbf{AG}(p_2))\},\\
\beta_1^*, s_1 &\mid\models \{\mathbf{AG}(p_1), \mathbf{AX}(\mathbf{AG}(p_1))\},\\
\beta_1^*, s_2 &\mid\models \{\mathbf{AG}(p_1)\}, \text{ and}\\
\beta_2^*, s_2 &\mid\models \{\mathbf{AG}(p_2)\}.
\end{aligned}$$

Note that state s_0 cannot be labeled with a meaningful language. I.e., no meaningful language L' (under a control action implemented by φ) can be found such that

$$\begin{aligned}
L', s_0 &\mid\models \mathbf{AX}(\mathbf{AG}(p_1)) \vee \mathbf{AX}(\mathbf{AG}(p_2)), \quad \textit{i.e.}\\
M[L', \varphi], s_0 &\models \mathbf{AX}(\mathbf{AG}(p_1)) \vee \mathbf{AX}(\mathbf{AG}(p_2)).
\end{aligned}$$

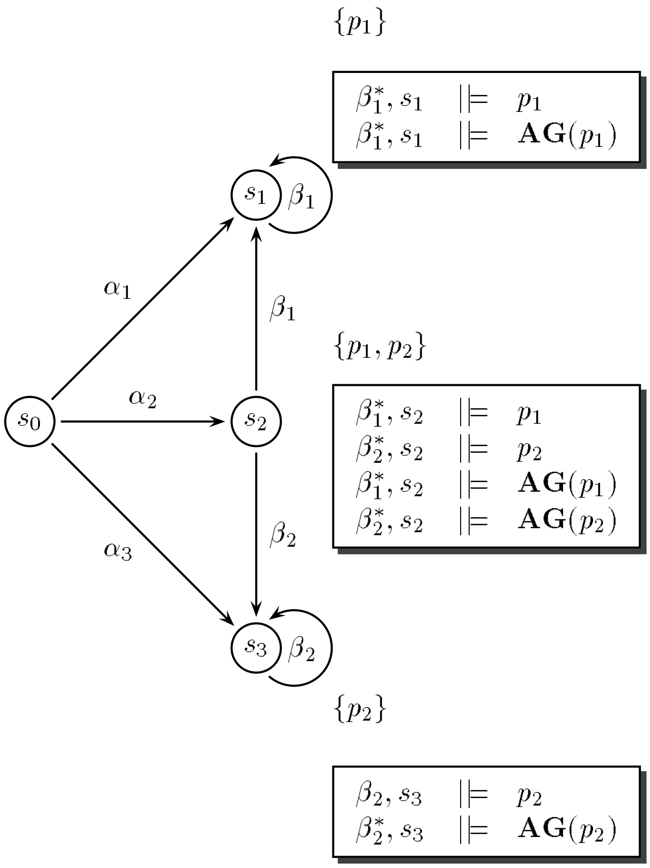

Figure 2: *A counterexample which shows the difficulties for the synthesis of a supervisor for a* CTL *disjunction. We cannot find a reasonable supervisor* φ *for* $\mathbf{AX}(\mathbf{AG}(p_1)) \vee \mathbf{AX}(\mathbf{AG}(p_2))$ *even if the supervisors for the subformulæ are well defined.*

In fact, when formula (1) is eventually considered in state s_1, neither of the two **AX** disjuncts can be satisfied unless some control action is enforced, either by disabling the α_1 or the α_2 transitions (as it was implicitly assumed when labeling s_3 with $\beta_1^* \; ||\!\!= \mathbf{AG}(p_1)$ – here it was assumed that β_2 was disabled).

For the two disjuncts $\mathbf{AX}(\mathbf{AG}(p_1))$ and $\mathbf{AX}(\mathbf{AG}(p_2))$ in (1), there are the following cases.

1. *disabling* α_1:

$$(\alpha_2 + \alpha_3)\beta_2^*, s_1 \; ||\!\!= \mathbf{AX}(\mathbf{AG}(p_2)), \tag{3}$$

2. *disabling* α_2:

$$(\alpha_1 + \alpha_3)\beta_1^*, s_1 \; ||\!\!= \mathbf{AX}(\mathbf{AG}(p_1)). \tag{4}$$

The following abbreviations will be useful

$$\begin{aligned} L_1 &\stackrel{\triangle}{=} (\alpha_1 + \alpha_3)\beta_1^*, \\ L_2 &\stackrel{\triangle}{=} (\alpha_2 + \alpha_3)\beta_2^*. \end{aligned}$$

It can now be noted that it is impossible to label s_1 with the union of L_1 and L_2, as intuition may suggest,

$$L_1 \cup L_2, s_1 \; ||\!\!= \mathbf{AX}(\mathbf{AG}(p_1)) \vee \mathbf{AX}(\mathbf{AG}(p_2)),$$

because of the control action that must be performed in order to ensure that either of the disjuncts is satisfied[2]. That is, by labeling state s_1 and formula (1) with $L_1 \cup L_2$ there is some *lost information* about what control actions we are enforcing. This problem arises because of the *choice* inherent to the disjunction.

2.2 $\mathcal{NP}$-completeness of Supervisor Choice

The example of Section 2.1 is problematic because of the inherent choice that must be made in order to satisfy either of the disjuncts. In particular, there is no intuitive construction of a reasonable supervisor and therefore no elementary characterization of the language L satisfying the disjunction. This section proves a stronger result, i.e., it shows that any algorithm capable of finding a nontrivial (nonempty) supervisor for a disjunction of the kind (1) is $\mathcal{NP}$-complete.

A specific example of the reduction used is given in the next section, before giving the full formalization of the $\mathcal{NP}$-completeness argument.

[2]The same problem happens in state s_2.

2.2.1 Reduction Example

As already mentioned, the reduction used in the $\mathcal{NP}$-completeness proof is from SAT. The problematic example of Section 2.1 yields the basic "gadget" used for the construction.

Consider the SAT formula

$$(x_1 \vee \overline{x_2} \vee x_4) \wedge (\overline{x_3} \vee \overline{x_4} \vee x_5) \wedge (\overline{x_1} \vee x_2 \vee x_4 \vee \overline{x_5}). \tag{5}$$

This formula is satisfied, e.g., by the assignment $x_1 \leftarrow 1$, $x_2 \leftarrow 1$, and $x_3 \leftarrow 0$.

Using the model relative to formula (1) as a basic gadget, formula (5) can now be encoded in a CTL supervisor synthesis problem. Two things will be built.

1. A model derived from the structure of the SAT formula.
2. A set of CTL formulæ whose satisfiability depends on the construction of an appropriate supervisor.

The model used is depicted in Figure 3. Each of the variables $x_1, \ldots, x_5$ is represented by a gadget, which will be used as a "truth-setting" device, rooted at states $X_1, \ldots, X_5$. Each clause in formula (5) is represented by a state C_1, C_2, C_3 connected to the variables appearing in it. A start state S_0 represents the conjunction.

The transitions labeled κ_t are *controllable*, while those labeled υ_m and $\upsilon_{m,n}$ are *uncontrollable.*

It is immediate to note that this construction can be done in polynomial time, given an arbitrary SAT formula.

The next step builds the following CTL formulæ, corresponding to the SAT formula in (5).

$$\mathbf{AXAX}\left(\begin{array}{ll} (X_1 \Rightarrow (\mathbf{AXAG}(p_{1F}) \vee \mathbf{AXAG}(p_{1T}))) & \wedge \\ (X_2 \Rightarrow (\mathbf{AXAG}(p_{2F}) \vee \mathbf{AXAG}(p_{2T}))) & \wedge \\ (X_3 \Rightarrow (\mathbf{AXAG}(p_{3F}) \vee \mathbf{AXAG}(p_{3T}))) & \wedge \\ (X_4 \Rightarrow (\mathbf{AXAG}(p_{4F}) \vee \mathbf{AXAG}(p_{4T}))) & \wedge \\ (X_5 \Rightarrow (\mathbf{AXAG}(p_{5F}) \vee \mathbf{AXAG}(p_{5T}))) & \end{array}\right), \tag{6}$$

$$\mathbf{AX}\left(\begin{array}{ll} (C_1 \Rightarrow \mathbf{EX}(\mathbf{AXAG}(p_{1T}) \vee \mathbf{AXAG}(p_{2F}) \vee \mathbf{AXAG}(p_{4T}))) & \wedge \\ (C_2 \Rightarrow \mathbf{EX}(\mathbf{AXAG}(p_{3F}) \vee \mathbf{AXAG}(p_{4F}) \vee \mathbf{AXAG}(p_{5T}))) & \wedge \\ (C_3 \Rightarrow \mathbf{EX}(\ \mathbf{AXAG}(p_{1F}) \vee \mathbf{AXAG}(p_{2T}) \vee & \\ \qquad\qquad \mathbf{AXAG}(p_{4T}) \vee \mathbf{AXAG}(p_{5F}))) & \end{array}\right). \tag{7}$$

The formulæ (6) and (7) are constructed in such a way as to force the supervisor to make certain choices in order to ensure their satisfiability. Formula (6) controls whether variable x_j is set to 0 or 1, while formula (7) expresses the satisfiability of each clause C_i by encoding the structure of the SAT formula. The uncontrollable events also play a role, by forcing the satisfaction of all the "next" states from S_0 and the C_i's.

In order to satisfy the conjunction of formulæ (6) and (7) with a non-trivial supervisor (i.e. a supervisor that would not simply yield an empty language), each of the states C_i must lead to at least one subsequent state where either the *false* or *true* state (i.e. the states labeled only with p_{jT} and p_{jF}) is removed. Because of the discussion regarding the gadget

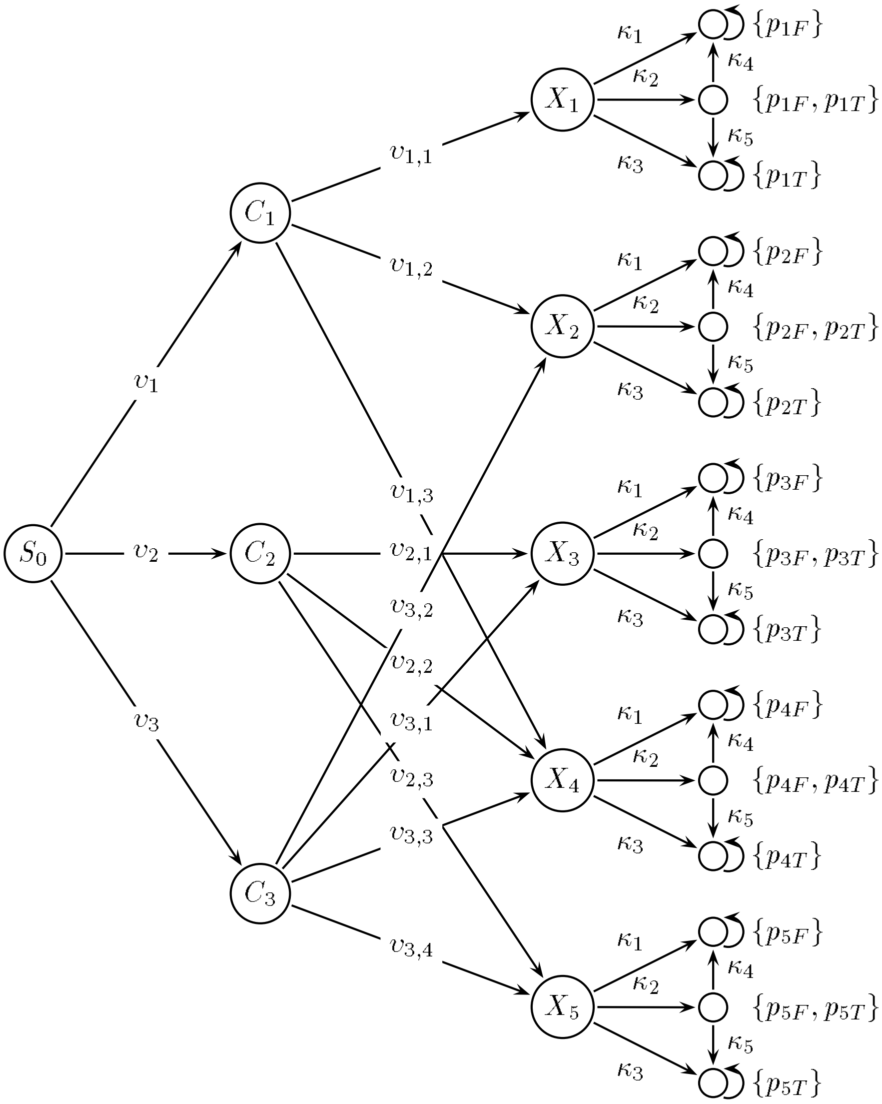

Figure 3: *Encoding of the SAT formula. Each clause C_i of the original formula has a corresponding state, and each variable is represented by a gadget rooted at X_j (The selfloops on the states with propositions p_{iF} should be labelled k_4 and similarly the ones with propositions p_{iT} should be labelled k_5 – the labels were left out in order to make the picture cleaner.)*

rooted at X_j of Section 2.1, there are only two possibilities: one "cutting off" the upper part of the gadget and the other cutting off the lower one.

Now, any algorithm capable of making the conjunction of (6) and (7) satisfiable, must make an appropriate choice at each of the gadgets. But this means that such an algorithm will be able to determine the satisfiability of the SAT formula (5). This algorithm is therefore $\mathcal{NP}$-hard.

The next section contains the formal proof of the $\mathcal{NP}$-completeness of the problem.

2.2.2 $\mathcal{NP}$-completeness Proof

The $\mathcal{NP}$-completeness theorem is stated following the "format" used in [GJ79] and the terminology introduced in [Ant95].

Theorem 1 Supervisor Synthesis for Unrestricted CTL Specifications*:*
INSTANCE*: A* Controlled Discrete Event System *(cfr. [RW87b]) and a specification $\mathcal{S}$ consisting of a set of* CTL *formulæ.*
QUESTION*: Is there a* supervisor map φ *such that*

$$L, \varphi, s_0 \;||\!\!= \mathcal{S}?$$

(Where s_0 is the "initial state" of the CDES.)

This problem is $\mathcal{NP}$-complete.

Proof.

$\mathcal{NP}$ Membership: The problem is in $\mathcal{NP}$ because of the following argument. Guess a supervisor assignment and verify it using the Model Checking (linear) algorithm for CTL [Eme90]. Since the model checking algorithm is linear, the whole algorithm is clearly in $\mathcal{NP}$.

$\mathcal{NP}$-hardness: The argument is the same as the one used in the discussion of Section 2.2.1. The construction is here formalized for generic SAT formulæ.
We are given a SAT formula

$$f = \bigwedge_{i=1}^{m} \left(\bigvee_{j=1}^{n} \ell_{i,j} \right).$$

Each $\ell_{i,j}$ is either a positive or negative literal built on the boolean variable x_k (where $k = 1, \ldots, v$).

The labeled graph is constructed by building a gadget for each variable x_k. Name the "start" node for each of these gadgets X_k. Next build a node C_i for each of the m clauses. The graph will contain a transition $C_i \stackrel{v_{i,j}}{\rightarrow} X_k$ for each $\ell_{i,j}$ in C_i. Finally, connect the start state S_0 corresponding to the whole formula f to each "clause" state by adding the transitions $S_0 \stackrel{v_i}{\rightarrow} C_i$.

Next the two CTL formulæ which are part of the specification $\mathcal{S}$, are constructed as follows.

- *Truth Setting formula*

$$\mathbf{AXAX}\left(\bigwedge_{k=1}^{v}(X_k \Rightarrow (\mathbf{AXAG}(p_{kF}) \vee \mathbf{AXAG}(p_{kT})))\right). \tag{8}$$

- *Clauses Encoding*

$$\mathbf{AX}\left(\bigwedge_{i=1}^{m}\left(C_i \Rightarrow \mathbf{EX}\left(\bigvee_{j=1}^{n}(\mathbf{AXAG}(P_{i,j})\right)\right)\right), \tag{9}$$

where each of the $P_{i,j}$ is either p_{kF} or p_{kT} depending on the "sign" of $\ell_{i,j}$.

It is immediate to note that the construction of the graph and of the two CTL formulæ is a polynomial process.

After the construction of the "gadget graph" and of the two CTL formulæ, the remaining step is to show that the SAT formula is satisfiable if and only if there exists a non trivial supervisor for the CTL specification $\mathcal{S}$.

Suppose that there exist a satisfying assignment for the SAT formula f. Then, by cutting off the appropriate branches of the "end" gadgets we can ensure the satisfiability of both formulæ (8) and (9). Each of the suffix languages generated from each of the X_j nodes is nonempty because of the choice imposed by the satisfying assignment of the SAT formula f. Therefore, the language generated from the start state S_0 is non empty too, since the uncontrollable events between S_0, the C_i, and the X_j states cannot be disabled. The satisfiability of the two formulæ (8) and (9) at state S_0 follows immediately by their construction.

Conversely, suppose that there exists a supervisor generating a nonempty language starting at S_0. Since this supervisor must respect the *controlled semantics*, it must not disable any of the uncontrollable events up to the X_j states.

Consider now the formulæ (8) and (9). The first one must be satisfied by the supervisor by "cutting off" either the *true* or *false* part of each gadget associated to each variable x_i. The second one is satisfied by a specific choice at each gadget (due to the encoding of the clauses' structure). Since the supervisor is assumed to produce a nonempty language, we must conclude that the two CTL formulæ are conjunctively satisfied. But this means that we have created a satisfying assignment also for the SAT formula f.

The conclusion is therefore that an algorithm capable of producing a supervisor is also capable of solving SAT. Hence the SUPERVISOR SYNTHESIS PROBLEM FOR CTL SPECIFICATIONS is $\mathcal{NP}$-complete.
□

3 Concluding Remarks

In this paper we have shown that the supervisory synthesis problem based on unrestricted CTL is $\mathcal{NP}$-complete. However, the proof is based on a specific set of CTL disjunctive formulæ. Other CTL disjunctions do not lead to the inherent choices, resulting in a combinatorial explosion of the search space. Elsewhere we have shown that if one restricts the attention to such a fragment of CTL then it is possible to build better algorithms for these cases (e.g. by limiting one of the disjuncts to be fully propositional, as in [Ant95]).

References

[Ant95] M. Antoniotti. *Synthesis and Verification of Discrete Controllers for Robotics and Manufacturing Devices with Temporal Logic and the* Control-D *System.* PhD thesis, Courant Institute of Mathematical Sciences, New York University, September 1995.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[Eme90] E. A. Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 16, pages 995–1072. MIT Press, 1990.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability.* W. H. Freeman and Company, 1979.

[RW87a] P. J. Ramadge and W. M. Wonham. On the Supremal Controllable Sublanguage of a Given Language. *SIAM J. Control and Optimization*, 25(3):637–659, May 1987.

[RW87b] P. J. G. Ramadge and W. M. Wonham. Supervisory Control of a Class of Discrete Events Processes. *SIAM J. Control and Optimization*, 25(1):206–230, 1987.