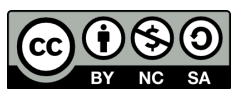
CDM

Verification and CTL

Klaus Sutner

Carnegie Mellon University

Fall 2019



Our automata-based decision algorithms are quite impressive, but still too weak for "real world" applications.

What kind of logic is required to verify the correctness of a "system" (which could be hardware, software, a protocol, ...)?

We need three components:

- A framework for modeling the system in question.
- A specification language that describes the desired properties.
- A verification algorithm that checks the specification against the description.

One might think that classical first order logic could be used for this purpose, but unfortunately there are several issues that make FOL less than suitable for our task.

The main problem is that the more expressive a language we chose, the harder the associated difficult decision problems become: even for propositional logic we already have to contend with $\mathbb{NP}$-hardness, so one should expect to encounter worse hardness or even undecidability when moving to a stronger system.

However, again in analogy to the propositional case, there is hope to develop good algorithms for "practical problems".

So the question is: is there a nice class of problems that is

- close enough to real problems so that a solution is of interest, and
- simple enough so that one can find good algorithms.

We can get some guidance by looking at a concrete example of a system that we might want to deal with.

There are four players in ABP:

- A sender that tries to transmit messages.
- A receiver that would like to receive these messages.
- A message channel that is responsible for the actual transmission of the messages.
- An acknowledgment channel that used for confirmation.

The problem is that the channels are faulty: they may delete or duplicate messages. We do not consider other errors here.

How should sender and receiver communicate so that the messages are correctly transmitted?

Note that we must assume that the channels are not completely broken (if no message gets through ever no protocol will help).

The sender maintains a single bit b, initialized to 0.

It sends a banana . . .

Let's focus on systems that have a clear notion of state transition: the system evolves in a sequence of steps

$$s_0 \to s_1 \to s_2 \to \ldots \to s_n \to \ldots$$

This type of behavior is more typical of hardware and protocols than of software.

We can think of the possible transitions as a binary relation on the space of all possible states:

$$\to \subseteq S \times S$$

or, if you prefer, as a digraph on S.

An important and very general class of systems is composed of states that are words over some alphabet: just think about binary words, representing a fixed number of bits.

So we have a structure

$$\mathfrak{C}_\rho = \langle \Sigma^\star, \to \rangle$$

where $\to$ is a binary predicate that expresses one step in the evolution of the system.

Obviously it makes sense to start with systems where $\to$ is very simple.

One way to formalize simplicity is to insist that $\to$ is a regular relation: there has to be a finite state machine that, given two points x and y, can check if $x \to y$.

How do we describe properties of a regular structure

$$\mathfrak{C}_{\rightarrow} = \langle \Sigma^{\star}, \rightarrow \rangle$$

We need some kind of specification language. Obviously we want to be able to use $\rightarrow$ as an atomic proposition and we want equality (which is trivially regular).

We also want a bit of logic, at the very least propositional logic.

$\bot$, $\top$	constants false, true
$x \rightarrow y$	atomic formulae
$\neg$	not
$\wedge$	and, conjunction
$\vee$	or, disjunction
$\Rightarrow$	conditional (implies)

The semantics of this language is clear. For example, $x \rightarrow y \Rightarrow x = y$ means that x is fixed point with respect to $\rightarrow$.

We can strengthen our language by adding quantifiers:

$\exists\, s$	there exists a state s
$\forall\, s$	for all states s

For example

$$\forall\, y \exists\, x\, (x \to y)$$

means that every state has a predecessor state (there is no Garden-of-Eden).

And

$$\forall\, x, y, z\, (x \to y \wedge x \to z \Rightarrow y = z)$$

means that the system is reversible (deterministic).

We would like some algorithm that takes as input

- a finite state machine that checks $\rightarrow$, and
- a first order sentence φ

and decides whether φ holds over $\mathfrak{C}_{\rightarrow}$.

Note that the carrier set of $\mathfrak{C}_{\rightarrow}$ is infinite, so brute-force will not work, we need a clever idea.

Also note that there is no hope to do this when the specification language is more complicated. For example, we must not be able to express questions about the behavior of Turing machines.

Wait a Minute . . .

Recall that configurations of Turing machines are just words $u\,p\,v$.

Also, it is easy to check that the one-step relation for Turing machines is regular.

But then can't we write down a formula φ that means something highly undecidable such as

The Turing machine halts on all inputs.

No, we cannot.

The relation $\left|\frac{1}{M}\right.$ is regular, and $\left|\frac{k}{M}\right.$ is regular for any fixed k, but $\left|\frac{}{M}\right.$ is not. Our language is not strong enough to talk about chains of arbitrary length.

Consider a formula $\varphi(x_1, \ldots, x_k)$ with k free variables as indicated.

We will construct an automaton $\mathcal{A}_\varphi$ that accepts precisely those k-track words that satisfy the formula:

$$\mathcal{L}(\mathcal{A}_\varphi) = \{\, u_1{:}u_2{:}\ldots{:}u_k \in \Gamma^\star \mid \mathfrak{C}_\to \models \varphi(u_1, u_2, \ldots, u_k) \,\}$$

Questions about $\mathfrak{C}_\to$ can then be answered by checking properties of $\mathcal{A}_\varphi$.

Example: Reversibility

The structure $\mathfrak{C}_{\to}$ is reversible if it satisfies

$$\forall\, x, y, z\, (x \to y \wedge x \to z \Rightarrow y = z)$$

Now let $\varphi(x, y, z) = x \to y \wedge x \to z \Rightarrow y = z$.

Then $\mathfrak{C}_{\to}$ is reversible iff $\mathcal{A}_{\varphi}$ is universal: it must accept all inputs.

Note that there is a price to pay: universality testing is fast only when the automaton is deterministic. Alas, $\mathcal{A}_{\varphi}$ will usually be nondeterministic.

We proceed by induction on the subformulae of φ.

Assume that ψ is a subformula of φ.

- If ψ is atomic, $\mathcal{A}_\varphi$ is essentially the given machine that checks $\rightarrow$ (modified to deal with k-track words.
- If ψ is of the form $\psi_1 \vee \psi_2$, $\mathcal{A}_\psi$ can be chosen to be the disjoint union of $\mathcal{A}_{\psi_1}$ and $\mathcal{A}_{\psi_2}$, maintaining the same alphabet.
- Universal quantifiers are translated into existential quantifiers. The automaton for $\psi = \exists\, y\, \psi_0$ can be obtained from $\mathcal{A}_{\psi_0}$ by erasing the track corresponding to variable y.

So far the operations are all quite efficient, but do produce nondeterministic machines.

To deal $\psi = \neg\psi_0$ we need to determinize the automaton $\mathcal{A}_{\psi_0}$ first so we can perform negation.

At this point, there may be exponential blow-up. □

We have a collection P of atomic properties $p_1, p_2, \ldots$ that may or may not hold at any specific state $s \in S$. The exact nature of these properties depends on the system in question, think about

- printer is busy
- register R_0 is initialized to 0
- process 2 is in its critical state

The details don't matter, but we have to be able to check easily whether property p holds in state s. We do this formally via a map $L : S \to \mathfrak{P}(P)$ describing which properties hold at which states:

$$s \models p \qquad \Longleftrightarrow \qquad p \in L(s).$$

For finitely many atomic properties and states this can be done by storing a table of bit-vectors.

Definition

A computation tree logic (CTL) structure for P has the form

$$\mathcal{A} = \langle S, \rightarrow, L \rangle$$

where $\rightarrow$ is a binary relation on S and $L : S \rightarrow \mathfrak{P}(P)$ is a property map. To avoid special cases we require that $\forall\, s\, \exists\, t\, (s \rightarrow t)$.

One can think of such a structure as a digraph on S whose nodes have out-degree at least 1 and are labeled by subset of P.

These structures will model the systems whose properties we are trying to verify.

Exercise

Explain how these CTL structures can be construed as specialized first order structures. What is the appropriate first order language here?

CTL structures describe the systems, so next we need a specification language in which to describe the properties we are interested in.

The easy part of this is to deal with atomic propositions and logical connectives. We use the symbols

$\bot$, $\top$	constants false, true
p, q, r, ...	propositional variables
$\neg$	not
$\wedge$	and, conjunction
$\vee$	or, disjunction
$\Rightarrow$	conditional (implies)

So this is very similar to propositional logic, but an assertion p now means: the current state has property p.

We will give a more detailed definition of the semantics of this language below.

It is tempting to enhance our language by adding quantifiers:

$\exists\, s$	there exists a state s
$\forall\, s$	for all states s

Unfortunately, this is less helpful than one might think.

For example, we would like to be able to say something like:

> *If the current state s has property p, then there is a path to some state that has property q.*

We could write down a formula along the lines of

$$(s, p) \Rightarrow \exists\, s_1, s_2\, (s \rightarrow s_1 \rightarrow s_2 \wedge (s_2, q))$$

to indicate that in two steps we can get to a state with q but that's much too weak.

To obtain the right notion of quantifiers, recall that we are dealing with systems that evolve via state transitions:

$$s_0 \to s_1 \to s_2 \to \ldots \to s_n \to \ldots$$

Since the relation $\to$ is not required to be deterministic, the evolution may be branching: for any $s \in S$ there may be several "next" states.

Since we want to be able to make assertions about next states, their next states, and so on, it makes sense to introduce path quantifiers that refer to whole paths in the digraph.

- $\mathbf{A}$: for all paths (starting at some state)
- $\mathbf{E}$: there exists a path (starting at some state)

Of course, being able to say "for all paths" is not quite enough: we want to be able to make assertions about what happens along these paths.

This is really a temporal logic problem: think of the transitions as time, so we want to make assertions about the future.

- $\mathbf{X}$: at the next state in the path
- $\mathbf{F}$: at some point along the path
- $\mathbf{G}$: always along the path
- $\mathbf{U}$: until some point along the path

The last one is a bit more complicated and combines two subformulae as we will see in a minute.

Definition

In addition to the propositional part from above, the language for CTL also allows the constructs

- $\mathbf{EX}\varphi$, $\mathbf{AX}\varphi$,
- $\mathbf{EG}\varphi$, $\mathbf{AG}\varphi$,
- $\mathbf{EF}\varphi$, $\mathbf{AF}\varphi$,
- $\mathbf{E}(\psi\mathbf{U}\varphi)$, $\mathbf{A}(\psi\mathbf{U}\varphi)$.

So there are 8 (in words: eight) quantifiers in this logic.

Each involves quantification over paths and some temporal assertion.

For the $\mathbf{U}$ quantifier the matrix consists of two formulae, not just one as usual.

This may seem a whole lot more complicated than ordinary first-order logic (which has just a universal and an existential quantifier), but it isn't.

We have a way to express a system (CTL structures) and we have a way to express specifications (CTL formulae).

The next step is to define the semantics. The appropriate setting here is to choose a state s and define

$$\mathcal{A}, s \models \varphi$$

meaning: the formula φ holds along the paths starting at s. And, of course, ultimately we want to solve the decision problem:

Problem:	**CTL Validity**
Instance:	A CTL formula φ, a CTL structure $\mathcal{A}$ and a state s in $\mathcal{A}$.
Question:	Does $\mathcal{A}, s \models \varphi$ hold?

For the propositional part of CTL the semantics are no different from ordinary propositional logic. For the quantifiers we define

- $\mathbf{EX}\varphi$: for some s' such that $s \to s'$: $\mathcal{A}, s' \models \varphi$.
- $\mathbf{AX}\varphi$: for all s' such that $s \to s'$: $\mathcal{A}, s' \models \varphi$.
- $\mathbf{EG}\varphi$: there exists a path (s_i) starting at s such that $\mathcal{A}, s_i \models \varphi$ for all i.
- $\mathbf{AG}\varphi$: for all paths (s_i) starting at s we have $\mathcal{A}, s_i \models \varphi$ for all i.
- $\mathbf{EF}\varphi$: there exists a path (s_i) starting at s such that $\mathcal{A}, s_i \models \varphi$ for some i.
- $\mathbf{AF}\varphi$: for all paths (s_i) starting at s we have $\mathcal{A}, s_i \models \varphi$ for some i.
- $\mathbf{E}(\psi \mathbf{U} \varphi)$: there exists a path (s_i) starting at s and some i such that $\mathcal{A}, s_i \models \varphi$ and for all $j < i$ $\mathcal{A}, s_j \models \psi$.
- $\mathbf{A}(\psi \mathbf{U} \varphi)$: for all paths (s_i) starting at s there is some i such that $\mathcal{A}, s_i \models \varphi$ and for all $j < i$ $\mathcal{A}, s_j \models \psi$.

That's it.

Since we are dealing with a digraph, one might wonder what kind of graph-theoretic statements one can make in CTL.

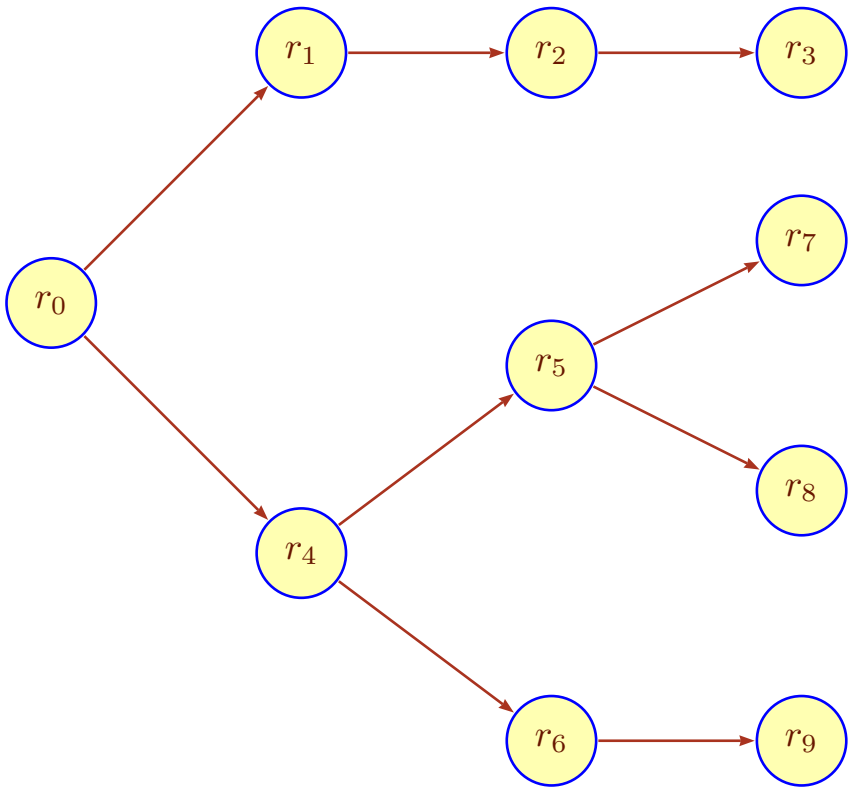
- $\mathcal{A}, s \models \mathbf{EX}p$ means there is a neighbor of s with property p.
- $\mathcal{A}, s \models \mathbf{AX}p$ means that all neighbors of s have property p.
- $\mathcal{A}, s \models \mathbf{EF}p$ means there is a path from s to t provided that t is the only state with property p.

As is clear from the last example, the labeling (the assignment of atomic properties) is crucial here.

Also note that we can only make statements about the weakly connected component of s in $\mathcal{A}, s \models \varphi$: none of the other states play any role in the evaluation of the formula.

Unfolding the Diagram

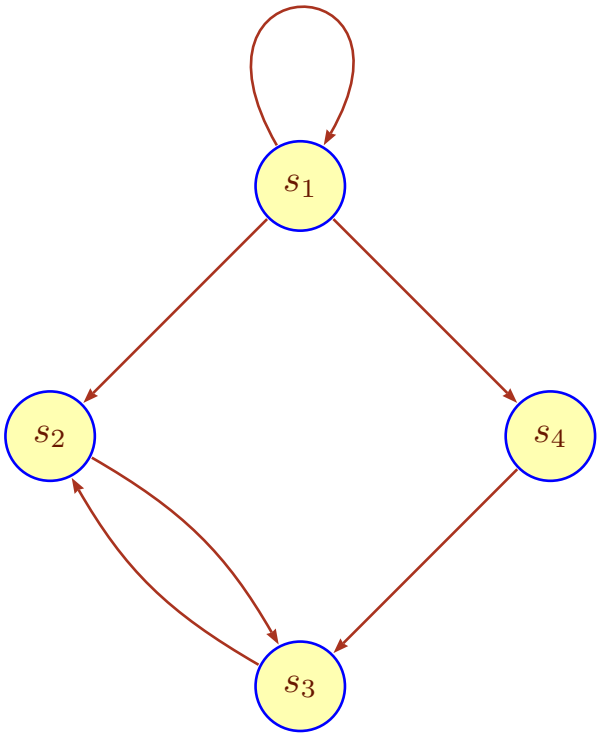
A good way to visualize the meaning of CTL formulae is to unfold the digraph into a tree, something like



One can then trace the paths in this tree.

A Tiny Example

Here is system with 4 states s_1, s_2, s_3, s_4 and 4 atomic properties p, q, r, t.
The atomic properties at each state are given in the table:



s_1	r
s_2	q, r, t
s_3	q, r
s_4	p, q

0	0	1	0
0	1	1	1
0	1	1	0
1	1	0	0

Some Assertions

The following assertions are all valid in the structure $\mathcal{A}$ from the last slide.

- $\mathcal{A}, s_1 \models \mathbf{EX}(q \wedge r)$
- $\mathcal{A}, s_1 \models \neg\mathbf{AX}(q \wedge r)$
- $\mathcal{A}, s_1 \models \neg\mathbf{EF}(p \wedge r)$
- $\mathcal{A}, s_3 \models \mathbf{EG}r$
- $\mathcal{A}, s_3 \models \mathbf{AG}r$
- $\mathcal{A}, s_1 \models \mathbf{AF}r$
- $\mathcal{A}, s_1 \models \mathbf{E}(p \wedge q\mathbf{U}r)$
- $\mathcal{A}, s_1 \models \mathbf{A}(p\mathbf{U}r)$

Exercise

Verify that all these assertions are indeed valid in $\mathcal{A}$.

Exercise

Find more valid and invalid assertions for this structure.

Here are some important assertions one can make in this language.

p holds infinitely often, no matter what happens starting at s (think of p: some process is enabled):

$$\mathcal{A}, s \models \mathbf{AGAF}p$$

We can make p hold again, no matter has happened so far (think of p: system is reset):

$$\mathcal{A}, s \models \mathbf{AGEF}p$$

Ultimately, p will hold everywhere, no matter has happened so far (think of p: deadlock has occurred):

$$\mathcal{A}, s \models \mathbf{AFAG}p$$

So the last property is not desirable and the system specification would presumably forbid it.

Recall from our discussion of propositional and predicate logic that equivalence is an important way to rewrite and simplify formulae.

How would we define equivalence here?

$$\psi \equiv \varphi \iff \forall \mathcal{A}, s\, (\mathcal{A}, s \models \psi \leftrightarrow \mathcal{A}, s \models \varphi)$$

Here are some easy equivalences for CTL.

- $\mathbf{AX}\varphi \equiv \neg\mathbf{EX}\neg\varphi$
- $\mathbf{EF}\varphi \equiv \neg\mathbf{AG}\neg\varphi$
- $\mathbf{AF}\varphi \equiv \neg\mathbf{EG}\neg\varphi$
- $\mathbf{AF}\varphi \equiv \mathbf{A}(\top\mathbf{U}\varphi)$
- $\mathbf{EF}\varphi \equiv \mathbf{E}(\top\mathbf{U}\varphi)$

Proposition

$$\mathbf{A}(\psi \mathbf{U} \varphi) \equiv \neg(\mathbf{E}(\neg\varphi \mathbf{U}(\neg\psi \wedge \neg\varphi)) \vee \mathbf{EG}\neg\varphi)$$

The reason these equivalences are important is that they allow us to eliminate some of the connectives and quantifiers from the language and thus simplify the decision algorithm.

In particular a system using only

$$\bot, \neg, \wedge, \mathbf{EX}, \mathbf{EG}, \mathbf{EU}$$

is already adequate.

The idea behind the algorithm is to, given $\mathcal{A}$ and φ, compute

$$\{\, s \in S \mid \mathcal{A}, s \models \varphi \,\}.$$

To this end, label the states of $\mathcal{A}$ with all the subformulae of φ that are satisfied at the state.

This is done by induction, starting with atomic formulae and gradually building up to φ itself.

The process starts with the trivial subformulae $\bot$ and p: nothing is labeled $\bot$, and for p the labeling is determined by $L(s)$.

We assume that the formula is built from connectives and quantifiers

$$\neg, \wedge, \mathbf{EX}, \mathbf{AF}, \mathbf{EU}.$$

- $\varphi = \neg\psi$: label s with φ if s is not labeled with ψ.
- $\varphi = \psi_1 \wedge \psi_2$: label s with φ if s is labeled by both ψ_1 and ψ_2.
- $\varphi = \mathbf{EX}\psi$: Label any state with φ if at least one of their immediate successors is labeled ψ.
- $\varphi = \mathbf{AF}\psi$: First label all states labeled with ψ with φ. Then label all states all of whose immediate successors are labeled φ by φ, until a fixed point is reached.
- $\varphi = \mathbf{E}(\psi_1 \mathbf{U} \psi_2)$: First label all states labeled with ψ_2 with φ. Then label all states with φ if they are labeled ψ_1 and one of their immediate successors is labeled φ, until a fixed point is reached.

Proposition

The running time of this algorithm is $O(mn(n+e))$ where m is the number of logical operators in the formula, n the number of states in $\mathcal{A}$ and e the number of transitions.

Proof.

To see this, note that each logical operator is handled only once, and that processing each such operator except for the last two cases is linear in $n+e$, the size of the digraph.

The last two cases, $\varphi = \mathbf{AF}\psi$ and $\varphi = \mathbf{E}(\psi_1 \mathbf{U} \psi_2)$ require time $O(n(n+e))$: we have to repeat the basic operation until no changes occur (which might take $n-1$ rounds).

□

This performance is sufficient for small structures but becomes a problem when $\mathcal{A}$ is large.

The fixed point method from above ignores the structure of the digraph. We can exploit this structure if we switch to another system of quantifiers:

$$\bot, \neg, \wedge, \mathbf{EX}, \mathbf{EG}, \mathbf{EU}.$$

$\mathbf{EX}$ and $\mathbf{EU}$ can be handled in linear time with a bit of care.

For $\mathbf{EG}\psi$ we first produce the subgraph of all states labeled ψ. Then compute the strongly connected components and the acyclic skeleton of the restricted graph. Then determine all nodes from which such SCC is reachable. All of this can be done in time linear in $n + e$ using, say, Tarjan's algorithm.

Proposition

The improved version of the algorithm has running time $O(m(n + e))$.

Even the fast model checking algorithm is often not good enough: in practical applications the size of the structure is often enormous.

Its size is often exponential in the number of variables, so the adding one Boolean variable to the system doubles the size of the CTL structure.

There is no silver bullet for this problem, but a number of methods exist to deal with state explosion.

- Highly efficient data structures such as ordered binary decision diagrams (OBDDs).
- Abstraction: Shrinking the model by removing variables that are not relevant for the formula in question.
- Reduction: for asynchronous systems many different traces may be equivalent as far as the formula in question is concerned.
- Induction: if there is a large number of similar components some type of induction may be used to deal with them.
- Composition: try to break the problem into a number of smaller ones.

Example: Mutual Exclusion

Suppose we have 2 processes that share are resource and we want to make sure that at any time only one of them can be in a critical section where the resource is used.

We think of each of the processes as being in one of three states:

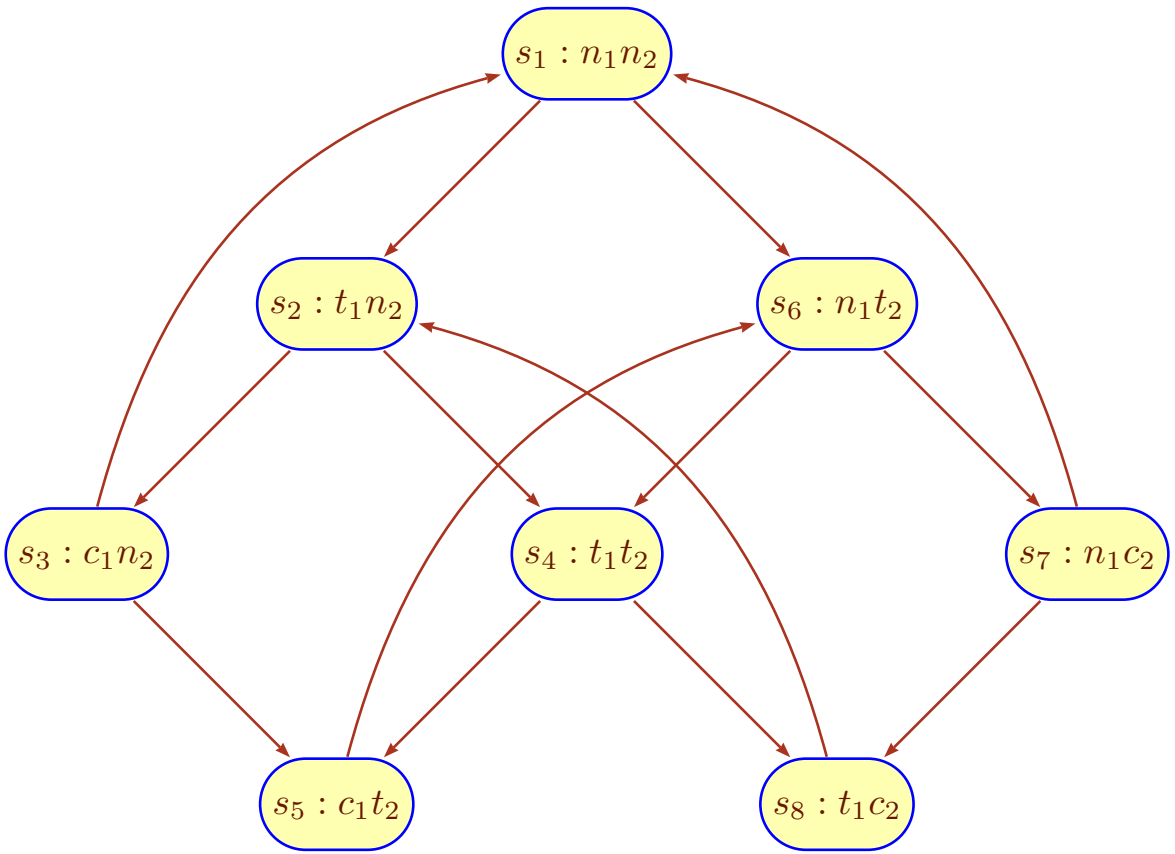
- n: non-critical,
- t: waiting to enter its critical state, and
- c: in its critical section.

So each process is moving in the cycle $n \to t \to c \to n \to \ldots$.

The problem is to devise a protocol that coordinates the two processes.

Needless to say, the protocol would have to guarantee certain properties. E.g., each process waiting to enter its critical section should ultimately get a chance to do so. We will write down these conditions later.

Instead of spelling out the protocol in words we give the corresponding CTL structure cA.



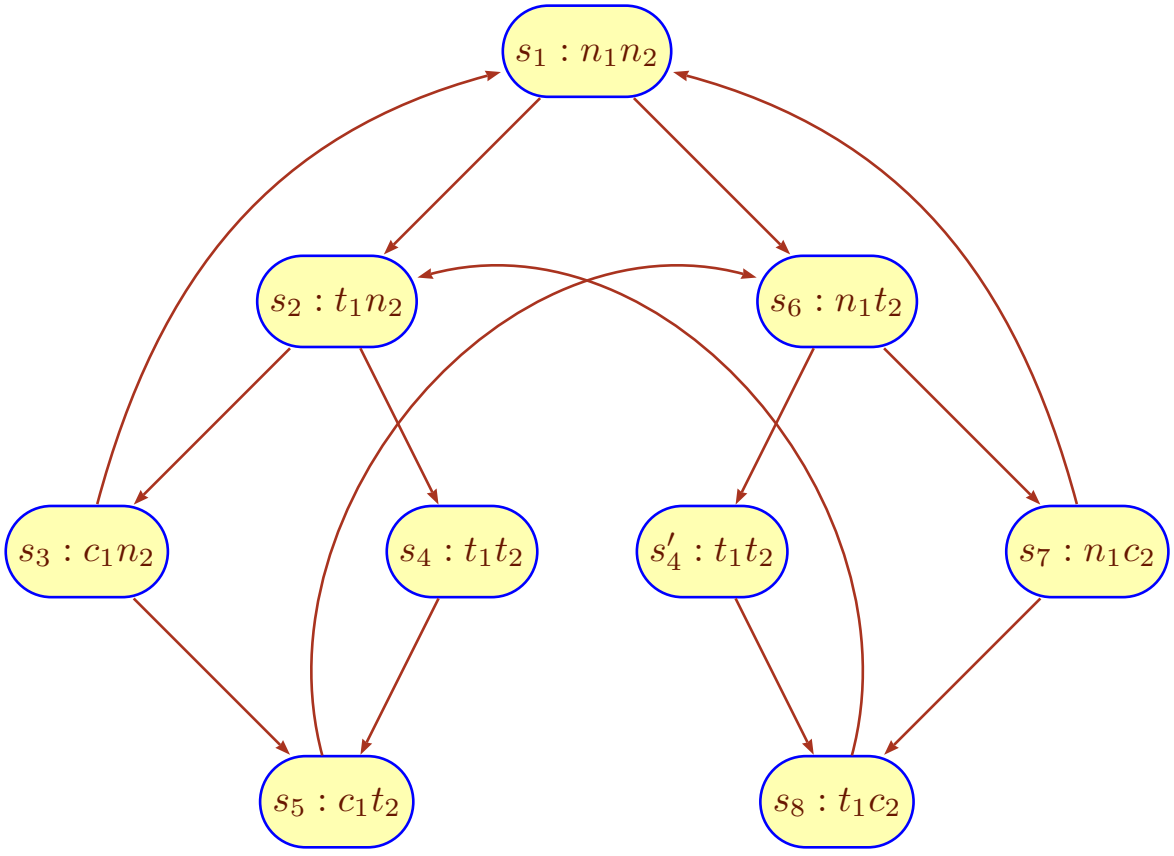
What properties should one demand from the protocol?

- Safety: only one process can be in its critical section at any time.
  $\Psi_1 = \mathbf{AG}\neg(c_1 \wedge c_2)$.
- Liveness: any process waiting to enter its critical section will ultimately do so.
  $\Psi_2 = \mathbf{AG}((t_1 \rightarrow \mathbf{AF}c_1) \wedge (t_2 \rightarrow \mathbf{AF}c_2))$.
- Non-Blocking: Any process can always request to move into its critical section.
  $\Psi_3 = \mathbf{AG}((n_1 \rightarrow \mathbf{EX}t_1) \wedge (n_2 \rightarrow \mathbf{EX}t_2))$
- Sequencing: processes need not enter their critical sections in alternating fashion.
  $\Psi_4 = \mathbf{EF}(c_1 \wedge \mathbf{E}(c_1\mathbf{U}(\neg c_1 \wedge \mathbf{E}(\neg c_2\mathbf{U}c_1)))) \wedge \ldots$.

Claim

Properties Safety, Non-Blocking and Sequencing are satisfied but Liveness is not.

A Better Protocol



We can fix the Liveness problem by splitting state s_4.

Claim

The second protocol satisfies all four properties.

It is still not ideal, though.

For example, the protocol insists that at every step one of the state properties changes. But we cannot just let a process stay in its critical section, either: otherwise the other process never gets a chance.

This leads to the issue of fairness.

Exercise

Verify that the second model really satisfies Ψ_1 through Ψ_4.