CDM
Monadic Second-Order Logic

Klaus Sutner

Carnegie Mellon Universality

52-mso-ctl 2017/12/15 23:19

1. Second-Order Logic

   - Words as Structures

   - CTL Structures

The Central Weakness

Intuitively, FOL allows us to ask “local” queries about an automatic structure.

For example, given an automatic graph $\langle V, E \rangle$ we can describe the local neighborhood of any vertex. We can also assert that there are paths of a certain, fixed length. But we cannot express more global properties such as connectivity: we cannot handle paths of arbitrary length.

This and other limitations makes it natural to try to look for stronger logics and hope they might still admit algorithms for model checking.

Exercise

Use the compactness theorem for FOL to show that connectivity is not first-order definable.

For example, suppose you are doing calculus and want to say something along the lines of

> For every differentiable function f, yarglebargle f bargleblargh.

Even if we assume that the underlying structure is $\mathbb{R}$, this is not expressible.

Things get worse when we try to make statements about some class of higher order functions, say, measures on some space.

Things get worse yet if we try to have this discussion over a tame structure like $\mathcal{N}$.

We simply need more expressive power.

Higher Order Logic

In first-order logic quantification takes place only over individuals.

Sets of individuals, and more generally relations and functions on the domain, are given by an appropriate first-order structure but cannot be quantified at the syntactical level.

And, of course, there is no way to quantify over, higher type objects such as families of functions.

This suggests a generalization: how about allowing quantification over all these objects? For example, rewritten in terms of set theory we would like to be able to make assertions

$$\forall\, x \in \mathfrak{P}(A)\ \ldots x \ldots$$

When $A = \mathbb{N}$, this would allow us to talk about sets of reals, a perfectly natural thing to do.

HOL light

J. Harrison has developed an HOL theorem prover, written in OCaml, that is surprisingly powerful and freely available.

`http://www.cl.cam.ac.uk/~jrh13/hol-light/`

As we will see in a moment, no such prover can be complete.

However, this one is capable of proving lots of interesting theorems and has had considerable success in the verification of fairly complicated fragments of math.

It has also been used to verify floating-point algorithms for Intel.

1994: Pentium FDIV Bug

Floating Point Error

$4195835.0/3145727.0 = 1.3338204491362410025$ correct

$4195835.0/3145727.0 = 1.3337390689020375894$ pentium

Alternatively

$4195835.0 - 3145727.0 * (4195835.0/3145727.0) = 0$ correct

$4195835.0 - 3145727.0 * (4195835.0/3145727.0) = 256$ pentium

Discovered in October 1994 by number theorist Thomas R. Nicely, doing research in pure math.

HOL is a huge sledge-hammer. As a first step towards more powerful quantification, consider so-called second-order logic (SOL) where one can only quantify over

- individuals,
- sets of individuals (monadic fragment),
- k-ary relations on individuals.

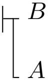
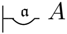
This is called second-order logic (SOL) and turns out to be in essence just as powerful as full HOL.

Incidentally, both FOL and SOL were introduced more or less by Frege in his *Begriffsschrift* in 1879.

Aside: Begriffsschrift

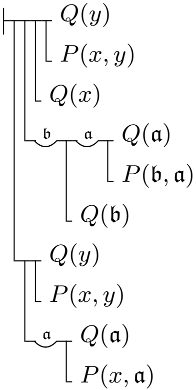
In 1879 Frege published his *Begriffsschrift*, which translates roughly as “concept script.” He establishes a language and a (very powerful) logic, the groundwork for the Grundgesetze.

Unfortunately, Frege developed a horrible, two-dimensional notation system.

	$A \Rightarrow B$
	$\neg A$
	$\forall \mathfrak{a}\, A$

This may seem harmless, but if one constructs larger formulae from these primitives, things start to look quite ominous.

The Worst Notation System, Ever



Translated

In modern notation (arguably, a huge improvement):

$$\begin{aligned}[\forall a\,(P(x,a) \Rightarrow Q(a)) \Rightarrow (P(x,y) \Rightarrow Q(y))] &\Rightarrow \\ [\forall b\,(Q(b) \Rightarrow \forall a\,(P(b,a) \Rightarrow Q(a)))] &\Rightarrow \\ Q(x) &\Rightarrow \\ P(x,y) &\Rightarrow Q(y)\end{aligned}$$

There are 4 premisses, and altogether they imply $Q(y)$.

Notation matters!

Arities

When writing SOL formulae (in modern notation), it is sometimes helpful to indicate the arity of relation and function symbols by a superscript. So, for example,

$$\forall\, R^2\, (\ldots R(x,y) \ldots)$$

is a statement about all binary relations.

Note that one can get by without function symbols. For example, we can fake unary functions like so. Instead of $\exists\, f^1\ \ldots$ we write

$$\exists\, R^2\, (\forall\, x\, \exists\, y\, R(x,y) \wedge \forall\, x, y, z\, (R(x,y) \wedge R(x,z) \Rightarrow y = z) \wedge \ldots)$$

Equality

We could also get rid of equality on individuals:

$$x = y \iff \forall\, R^1 \,(R(x) \Leftrightarrow R(y))$$

The importance of this idea was first recognized by Leibniz and is enshrined in his principium identitatis indiscernibilium: if we cannot tell two entities apart, they are identical.

Killing Non-Standard Models

With second-order quantification one can avoid the effects of the compactness theorem for first-order.

For example, suppose we use the standard Peano axioms to describe arithmetic. Then add the following axiom:

$$\forall x \forall X \big(X(0) \wedge \forall z \, (X(z) \Rightarrow X(z+1)) \Rightarrow X(x) \big)$$

Here X is a subset of the universe.

In other words, every set containing 0 and closed under successors is already the whole universe: this rules out the non-standard, "infinite" elements.

In fact, SOL can be used to produce a finite axiomatization of arithmetic that has only one model (categoricity).

Aside: Two Uses of Axioms

One standard application of axioms is to describe **all** examples of some particular kind of structure.

For example, the standard (first-order) axioms for groups describe the whole, large and very complex class of all groups.

The other standard application is to construct a set of axioms with the goal of pinning down **one** particular structure precisely.

For example, the Peano axioms form an attempt to give a precise and complete description of the natural numbers. Alas, if first-order this attempt fails, there are non-standard models.

No Free Lunch

Full SOL is quite powerful, but also quite unwieldy. In particular proof theory breaks down in the following sense: we cannot construct a proof system for SOL in a way that it satisfies the three standard requirements:

- Sound: only valid formulae are provable,
- Complete: all valid formulae are provable,
- Effective: the collection of all proofs is decidable.

This has lead some people like Willard Quine to deny that SOL is “a logic” at all. In their view it’s just a branch of set theory.

The reason there can be no adequate notion of proof for SOL is the following: for every sentence φ of arithmetic in FOL, one can effectively construct a sentence $\widehat{\varphi}$ of SOL such that

$$\widehat{\varphi} \text{ valid} \iff \mathfrak{N} \models \varphi.$$

Since proofs are always semidecidable (r.e.), the existence of a deduction system for SOL would immediately imply that the LHS is also semidecidable.

But Gödel and Tarski have shown that the RHS is far from being semidecidable.

In fact, validity of SOL as a decision problem is hideously complicated.

Monadic Second-Order Logic (MSOL)

Often it suffices to consider a weak subsystem for full SOL where quantification is restricted to just two kinds:

- individuals, and
- sets of individuals (relations of type R^1).

Notation:

$$\exists X \qquad \forall X$$

$$x \in X \qquad X(x)$$

Example: Least Upper Bounds

Assuming a binary total order $\leq$, we can express the assertion that every bounded set has a least upper bound:

$$\forall\, X\, \big(\exists\, z\, X(z) \wedge \exists\, x\, \forall\, z\, (X(z) \Rightarrow z \leq x) \Rightarrow$$

$$\exists\, x\, (\forall\, z\, (X(z) \Rightarrow z \leq x) \wedge \forall\, y\, (\forall\, z\, (X(z) \Rightarrow z \leq y) \Rightarrow x \leq y))\big)$$

This is the critical property of the reals and cannot be expressed in FOL.

Example: Well-Order

Again assume a binary total order $\leq$. We can express the assertion that we have a well-order in terms of the least-element principle: every non-empty set has a least element.

$$\forall\, X \,\big(\exists\, z\, X(z) \Rightarrow$$

$$\exists\, x\, (X(x) \wedge \forall\, z\, (X(z) \Rightarrow x \leq z))\big)$$

This is the critical property of the natural numbers with the standard order, and cannot be expressed in FOL.

Example: Reachability

Lastly, consider a digraph, a single binary edge relation →.

We can express the assertion that there is a path from s to t as follows:

$$\forall\, X \,\bigl(X(s) \land \forall\, x, y\, (X(x) \land x \to y \Rightarrow X(y)) \Rightarrow X(t)\bigr)$$

Again, FOL is not strong enough to express path existence in general (and thus other concepts like connectivity).

- Second-Order Logic

2 Words as Structures

- CTL Structures

Words as Structures

So far we have considered structures whose elements are words, and whose relations are synchronous.

Wild Idea: Can be think of a single word as a structure?

And, of course, use logic to describe the properties of the structure?

This may seem a bit weird, but bear with me. First, we need to fix an appropriate language for our logic. As always, we want at least propositional logic.

$\bot$, $\top$	constants false, true
$\neg$	not, negation
$\wedge$	and, conjunction
$\vee$	or, disjunction
$\Rightarrow$	implication
$\Leftrightarrow$	equivalence

Variables and Atomic Formulae

We will have variables x, y, z, $\ldots$ that range over positions in a word, integers in the range 1 through n where n is the length of the word.

We allow the following basic predicates between variables:

$$x < y \qquad\qquad x = y$$

Of course, we can get, say, $x \geq y$ by Boolean operations.

Most importantly, we write

$$Q_a(x)$$

for "there is a letter a in position x."

First-Order

We allow quantification for position variables.

$$\exists\, x\, \varphi \qquad \forall\, x\, \varphi$$

For example, the formula

$$\exists\, x, y\, (x < y \wedge Q_a(x) \wedge Q_b(y))$$

intuitively means “somewhere there is an a and somewhere, to the right of it, there is a b.”

The formula

$$\forall\, x, y\, (Q_a(x) \wedge Q_b(y) \Rightarrow x < y)$$

intuitively means “all the a’s come before all the b’s.”

Monadic Second-Order

We also have second-order variables X, Y, Z, ... that range over sets of positions in a word.

$$\exists X\,\varphi \qquad \forall X\,\varphi \qquad X(x)$$

Sets of positions are all there is; we do not have variables in our language for, say, binary relations on positions (we do not use full SOL).

This system is called monadic second-order logic (with less-than), written $\mathrm{MSO}[<]$.

We need some notion of satisfaction

$$w \models \varphi$$

where w is a word and φ a sentence in $\mathrm{MSO}[<]$.

We won't give a formal definition, but the basic idea is simple: Let $|w| = n$:

- the first order variables range over $[n] = \{1, 2, \ldots, n\}$,
- the second-order variables range over $\mathfrak{P}([n])$.

The basic predicates $x < y$ and $x = y$ have their obvious meaning. For the $Q_a(x)$ predicate we let

$$Q_a(x) \iff w_x = a$$

Examples

$$aaacbbb \models \forall\, x\, (Q_a(x) \lor Q_b(x) \lor Q_c(x))$$

$$aaabbb \models \exists\, x, y\, (x < y \land Q_a(x) \land Q_b(y))$$

$$bbbaaa \not\models \exists\, x, y\, (x < y \land Q_a(x) \land Q_b(y))$$

$$aaabbb \models \exists\, x, y\, (x < y \land \neg\exists\, z\, (x < z \land z < y) \land Q_a(x) \land Q_b(y))$$

$$aaacbbb \not\models \exists\, x, y\, (x < y \land \neg\exists\, z\, (x < z \land z < y) \land Q_a(x) \land Q_b(y))$$

$$aaacbbb \models \exists\, x\, (Q_c(x) \Rightarrow \forall\, y\, (x < y \Rightarrow Q_b(y)))$$

Less-Than or Successor

In applications, the atomic relation $x < y$ is slightly more useful than $y = x + 1$, but either one would have the same expressiveness.

On the one hand

$$y = x + 1 \iff x < y \land \forall\, z\,(x < z \Rightarrow y \leq z)$$

On the other hand write $\mathsf{closed}(X)$ for the formula $\forall\, z\,(X(z) \Rightarrow X(z+1))$. Then

$$x < y \iff x \neq y \land \forall\, X\,(X(x) \land \mathsf{closed}(X) \Rightarrow X(y))$$

This is sometimes written as $\mathrm{MSO}[<] = \mathrm{MSO}[+1]$.

Example

We can hardwire factors. For example, to obtain a factor abc let

$$\varphi \equiv \exists\, x, y, z\, (y = x + 1 \land z = y + 1 \land Q_a(x) \land Q_b(y) \land Q_c(z))$$

Then $w \models \varphi$ iff $w \in \Sigma^\star abc \Sigma^\star$.

Example

Scattered subwords are very similar in this setting:

$$\varphi \equiv \exists\, x, y, z\, (x < y \land y < z \land Q_a(x) \land Q_b(y) \land Q_c(z))$$

Then $w \models \varphi$ iff $w \in \Sigma^\star a \Sigma^\star b \Sigma^\star c \Sigma^\star$.

Example

We can split a word into two parts as in

$$\varphi \equiv \exists\, x\, \forall\, y\, ((y \leq x \Rightarrow Q_a(y)) \wedge (y > x \Rightarrow Q_b(y))) \vee \forall\, x\, (Q_b(x))$$

Then $w \models \varphi$ iff $w \in a^\star b^\star$.

Example

Let first(x) be shorthand for $\forall\, z\, (x \leq z)$, and last(x) shorthand for $\forall\, z\, (x \geq z)$. Then

$$\varphi \equiv \exists\, x, y\, (\mathsf{first}(x) \wedge Q_a(x) \wedge \mathsf{last}(y) \wedge Q_b(y))$$

Then $w \models \varphi$ iff $w \in a\Sigma^\star b$.

The Language of a Sentence

The examples suggest that, for any sentence φ, we should consider the collection of all words that satisfy φ:

$$\mathcal{L}(\varphi) = \{\, w \in \Sigma^\star \mid w \models \varphi \,\}.$$

One cannot fail to notice that, in the examples so far, $\mathcal{L}(\varphi)$ is always regular. Needless to say, this is no coincidence.

Also note that we have not used the second-order part of our language yet.

Example

Write $\mathsf{even}(X)$ to mean that X has even cardinality and consider

$$\varphi \equiv \exists\, X \left(\forall\, x\, (Q_a(x) \iff X(x)) \wedge \mathsf{even}(X)\right)$$

Then $w \models \varphi$ iff the number of a's in w is even.

We're cheating, of course; we need to show that the predicate $\mathsf{even}(X)$ is definable in our setting. This is tedious but not really hard:

$$\mathsf{even}(X) \iff \exists\, Y, Z\, (X = Y \cup Z \wedge \emptyset = Y \cap Z \wedge \mathsf{alt}(Y, Z))$$

Here $\mathsf{alt}(Y, Z)$ is supposed to express that the elements of Y and Z strictly alternate as in

$$y_1 < z_1 < y_2 < z_2 < \ldots < y_k < z_k$$

Missing Pieces

$$\begin{aligned}
X = Y \cup Z &\iff \forall\, u\, (X(u) \leftrightarrow Y(u) \vee Z(u)) \\
\emptyset = Y \cap Z &\iff \neg\exists\, u\, (Y(u) \wedge Z(u)) \\
\mathsf{alt}(Y, Z) &\iff \exists\, y \in Y\, \forall\, x < y\, (\neg Z(x)) \wedge \\
&\qquad \exists\, z \in Z\, \forall\, x > z\, (\neg Y(x)) \wedge \\
&\qquad \forall\, y \in Y\, \exists\, z \in Z\, (y < z \wedge \forall\, x\, (y < x < z \Rightarrow \neg Y(x) \wedge \neg Z(x))) \\
&\qquad \forall\, z \in Z\, \exists\, y \in Y\, (y < z \wedge \forall\, x\, (y < x < z \Rightarrow \neg Y(x) \wedge \neg Z(x)))
\end{aligned}$$

Exercise

The alt *formula above does not handle the case where Y and Z are empty; fix this.*

Show that one can check if the number of a's is a multiple of k, for any fixed k.

Definition

A language L is $\mathrm{MSO}[<]$ definable (or simply $\mathrm{MSO}[<]$) if there is some sentence φ such that

$$L = \mathcal{L}(\varphi) = \{\, w \in \Sigma^\star \mid w \models \varphi \,\}.$$

Our examples suggest the following theorem.

Theorem (Buechi 1960, Elgot 1961)

A language is regular if, and only if, it is $\mathrm{MSO}[<]$ *definable.*

The theorem connects complexity with definability: we can recognize a set of strings in constant space if, and only if, the set can be described by a formula in our logic.

Formula to Regular (Sketch)

Obviously, the proof comes in two parts:

- For every regular language L we need to construct a sentence φ such that $L = \mathcal{L}(\varphi)$.
- For every sentence φ we have to show that the language $\mathcal{L}(\varphi)$ is regular.

We should expect part (1) to be harder since there is no good inductive structure to exploit.

Part (2) is by straightforward induction on φ, but there is the usual technical twist: we need to deal not just with sentences but also with free variables. Since we don't have a formal semantics we will not give details of this construction, it is very similar to the argument for automatic structures.

Regular to Formula (Sketch)

We may safely assume that the regular language L is given by a DFA $M = \langle\, Q, \Sigma, \delta; q_0, F \,\rangle$.

For simplicity assume $Q = [n]$ and $q_0 = 1$.

We have to construct a formula φ such that $w \models \varphi$ iff M accepts w.

Consider a trace of M on input w

$$q_0\, w_1\, q_1\, w_2\, q_2 \ldots q_{m-1}\, w_m\, q_m.$$

Here m can be arbitrarily large.

We can think of states as being associated with the letters of the word as in

$$\begin{array}{cccccc} & w_1 & w_2 & w_3 & \ldots & w_m \\ q_0 & q_1 & q_2 & q_3 & \ldots & q_m \end{array}$$

Thus, position $x = 1, \ldots, m$ in the word is associated with state $\delta(q_0, w_1 \ldots w_x)$.

The Partition

In order to express this in a $\mathrm{MSO}[<]$ formula, we partition the set positions $[m]$ into $n = |Q|$ blocks X_1, X_2, ..., X_n such that

$$X_p(x) \iff \delta(q_0, w_1 \dots w_x) = p$$

Some of these blocks may be empty, but note that the number of blocks is always exactly n (which we can express as a formula).

But given state p in position x we can determine the state in position $x + 1$ given w_{x+1} by a table lookup – which table lookup can be hardwired in a formula.

Expressing Transitions

Technically, this is done by a formula

$$\Phi_{p,a} \equiv \forall\, x \left(X_p(x) \wedge Q_a(x+1) \Rightarrow X_{\delta(p,a)}(x+1) \right)$$

meaning “if at position x we are in state p and the next letter is an a, then the state in position $x+1$ is $\delta(p,a)$.

Note that this is not quite right, we really need a non-existing position 0 corresponding to state q_0.

Exercise

Figure out how to fix this little glitch. Also figure out how to express “the last state is final.”

Now consider the big conjunction of $\Phi_{p,a}$ where $p \in Q$ and $a \in \Sigma$. Add formulae that pin down the first and last state to arrive at a formula of the form

$$\varphi \equiv \exists\, X_1, \ldots, X_n\, \Psi$$

where Ψ is first-order as indicated above. □

Note that in conjunction with the opposite direction of Büchi's theorem, this result has the surprising consequence that every $\mathrm{MSO}[<]$ formula is equivalent to a $\mathrm{MSO}[<]$ formula containing only one block of existential second-order quantifiers.

Exercise

Fill in all the details in the last proof.

It is natural to ask whether the languages defined by the first-order fragment of $\mathrm{MSO}[<]$ have some natural characterization.

A language $L \subseteq \Sigma^\star$ is star-free iff it can be generated from $\emptyset$ and the singletons $\{a\}$, $a \in \Sigma$, using only operations union, concatenation and complement (but not Kleene star).

Note well: $a^\star b^\star a^\star$ is star-free.

Theorem

A language $L \subseteq \Sigma^\star$ is $\mathrm{FOL}[<]$ definable if, and only if, L is star-free.

Schützenberger's Theorem

While we're at it: star-free languages are quite interesting since they admit a purely algebraic characterization in terms of their syntactic semigroups.

A semigroup is aperiodic if it contains only trivial subgroups (the idempotents of the semigroup).

Theorem (Schützenberger 1965)

A regular language is star-free if, and only if, its syntactic semigroup is aperiodic.

The Büchi/Elgot theorem establishes a connection between a very low complexity class (constant space) and $\mathrm{MSO}[+1]$. In fact, there is the whole area of descriptive complexity that characterizes complexity classes in terms of logic and finite structures:

- $\mathbb{NP}$ corresponds to existential SOL (Fagin 1974).
- PH corresponds to SOL.
- PSPACE corresponds to SOL plus a transitive closure operator.

Exercise

Figure out the details of Fagin's theorem.

Upcoming Attractions

We have seen that there is a deep connection between regular languages and certain weak logics, notably monadic second-order logic.

Büchi's result may seem a bit odd and of no practical value.

However, it also holds for infinite words and in that version it is extremely useful.

Next goal: push our theory to infinite words.

- Second-Order Logic

- Words as Structures

3. CTL Structures

Describing State

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:

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

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

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

where $\to$ is a binary relation on S and $L : S \to \mathfrak{P}(P)$ is a property map.

To avoid special cases we require that $\forall\, s\, \exists\, t\, (s \to t)$.

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?

Designing a Language

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, $\ldots$	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.

Quantifiers

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

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

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

Quantifiers

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)

More Quantifiers

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.

The CTL Language

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.

Where Are We?

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.

Simple Statements

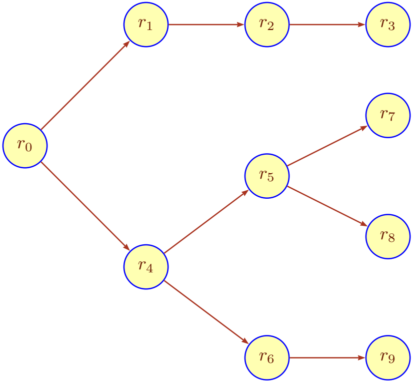
Since we are dealing with a digraph, one might wonder what kind of graph-theoretic statements one can make in CTL.

- $\mathcal{A}, s \models \mathbf{EX}p$ 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.

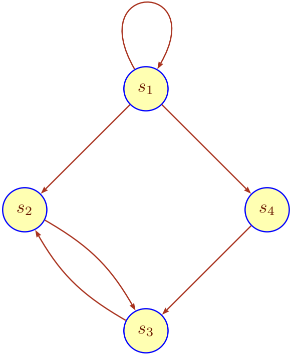
A good way to visualize the meaning of CTL formulae is to unfold the digraph into a tree, something like



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.

Some Typical Assertions

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.

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

A Hard Equivalence

Proposition

$$\mathbf{A}(\psi\mathbf{U}\varphi) \equiv \neg(\mathbf{E}(\neg\varphi\mathbf{U}(\neg\psi \land \neg\varphi)) \lor \mathbf{EG}\neg\varphi)$$

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

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

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 $L(s)$.

The Labeling Algorithm

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.

Speed-Up

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 $O(m(n + e))$.

State Explosion

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.

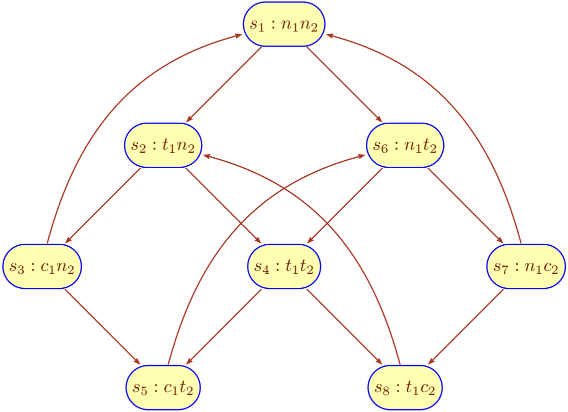
So each process is moving in the cycle $n \to t \to c \to n \to \ldots$.

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.

A Protocol

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



Specification

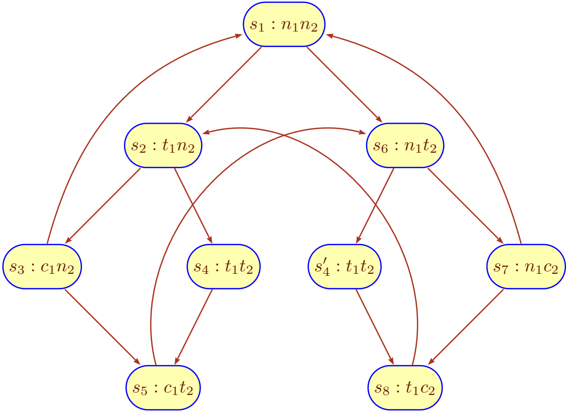
What properties should one demand from the protocol?

- 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.
  $\Psi_2 = \mathbf{AG}((t_1 \to \mathbf{AF}c_1) \wedge (t_2 \to \mathbf{AF}c_2))$.
- Non-Blocking: Any process can always request to move into its critical section.
  $\Psi_3 = \mathbf{AG}((n_1 \to \mathbf{EX}t_1) \wedge (n_2 \to \mathbf{EX}t_2))$
- 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.