Lecture 11

Linear Time Logic

February 12, 1998
Lecturer: Richard Ladner
Notes: Vassily Litvinov

11.1. LTL-satisfiability is PSPACE-complete

We will prove the theorem stated in the previous lecture:

Theorem 11.1: LTL-satisfiability is PSPACE-complete for log-space reductions.

Proof:

PSPACE-hardness. Given a Turing machine $T = (S, \Sigma, \Gamma, \delta, s)$ that runs in polynomial space, construct an LTL formula α which is satisfiable if and only if T accepts its input. We will assume that some $A \in S$ is the only accepting state and that once T gets in A it stays in it and keeps running forever.

Intuitively a satisfying model will represent an infinite sequence of configurations of T: $\#c_0\#c_1\#c_2\ldots$ where each c_i has the same length polynomial in the length of input. The LTL formula we will build will check that this string is a legal computation of T. The following properties need to be checked:

- c_0 is the initial configuration
- c_{i+1} follows c_i correctly
- some c_i is an accepting configuration

Define $\Gamma' = \Gamma \cup S \cup \{\#\}$; $m = |c_i| + 2$ is the length of a configuration; $w = w_1 \ldots w_n$ is the input. States in the model will represent individual characters σ_i which together form the infinite string of T's configurations: $\sigma_0\sigma_1\sigma_2\ldots \equiv \#c_0\#c_1\ldots$. The properties listed above can be restated in terms of $\sigma_0\sigma_1\sigma_2\ldots$ as follows:

- $\sigma_0 \ldots \sigma_m$ is the initial configuration
- $\forall i. f(\sigma_i, \sigma_{i+1}, \sigma_{i+2}, \sigma_{i+3}) = \sigma_{i+m}$, where f is the function that computes a symbol in the subsequent configuration of T given the corresponding four-window in the previous configuration

- $\exists i.\sigma_i = A$

All we need to do is just check these properties, because the only way for them to hold is when they represent an accepting computation of T.

The variables in our model will be z_σ for all $\sigma \in \Gamma'$. If z_τ is true in the ith state in the path, then $\sigma_i = \tau$. Now build the formula to be satisfied as the conjunction of the following:

- one symbol per position:

$$G \left(\bigvee_{\sigma\in\Gamma'} z_\sigma \wedge \bigwedge_{\sigma\in\Gamma'} \bigwedge_{\tau\in\Gamma'-\{\sigma\}} (z_\sigma \Rightarrow \neg z_\tau) \right)$$

- the first m states model the initial configuration:

$$z_\# \wedge X z_S \wedge X^2\, z_{w_1} \wedge \ldots \wedge X^{n+1}\, z_{w_n} \wedge X^{n+2}\, z_\sqcup \wedge \ldots \wedge X^{m-2}\, z_\sqcup \wedge X^{m-1}\, z_\#$$

- the computation of T is modeled correctly:

$$G \left(\bigwedge_{\sigma_0,\sigma_1,\sigma_2,\sigma_3\in\Gamma'} (z_{\sigma_0} \wedge X z_{\sigma_1} \wedge X^2\, z_{\sigma_2} \wedge X^3\, z_{\sigma_3} \Rightarrow X^m\, z_{f(\sigma_0,\sigma_1,\sigma_2,\sigma_3)}) \right)$$

- there is an accepting configuration

$$F\, z_A$$

This formula can be produced in log space. So, PSPACE-hardness has been shown.

LTL-SAT is in PSPACE. Given an LTL formula α, check if it is satisfiable. We will use a tableau method. The nondeterministic algorithm will discover an infinite sequence of states (that satisfies the formula) as a finite sequence with a loop. These days they use automata on infinite strings or trees (for branching time logic). We assume that we have only X and U as modal operators because G can be defined in terms of U.

Definition 11.2: A *tableau* for α is a pair $(\mathcal{T}, \mathcal{F})$ with the following properties:

- every subformula of α is in $\mathcal{T} \cup \mathcal{F}$ ("completeness")
- $\mathcal{T} \cap \mathcal{F} = \phi$ ("consistency")
- ("semantic consistency")

  $\neg\beta \in \mathcal{T} \Rightarrow \beta \in \mathcal{F}$

  $\neg\beta \in \mathcal{F} \Rightarrow \beta \in \mathcal{T}$

$\beta \vee \gamma \in \mathcal{T} \Rightarrow \beta \in \mathcal{T}$ or $\gamma \in \mathcal{T}$
$\beta \vee \gamma \in \mathcal{F} \Rightarrow \beta, \gamma \in \mathcal{F}$
(remember $\beta\ U\ \gamma \equiv \gamma \vee (\beta \wedge X(\beta\ U\ \gamma))$
$\beta\ U\ \gamma \in \mathcal{T}$ and $\gamma \in \mathcal{F} \Rightarrow \beta \in \mathcal{T}$
$\beta\ U\ \gamma \in \mathcal{F} \Rightarrow \gamma \in \mathcal{F}$
(other operators can be expressed in terms of the ones above)

- define relation $Next$ on two tableaus: $Next((\mathcal{T}, \mathcal{F}), (\mathcal{T}', \mathcal{F}'))$ if the following conditions hold:

  $X\alpha \in \mathcal{T} \Rightarrow \alpha \in \mathcal{T}'$
  $X\alpha \in \mathcal{F} \Rightarrow \alpha \in \mathcal{F}'$
  $\beta\ U\ \gamma \in \mathcal{T}$ and $\gamma \in \mathcal{F} \Rightarrow \beta\ U\ \gamma \in \mathcal{T}'$
  $\beta\ U\ \gamma \in \mathcal{F}$ and $\beta \in \mathcal{T} \Rightarrow \beta\ U\ \gamma \in \mathcal{F}'$

We will also define $untils(\mathcal{T})$ to be all U-formulas in $\mathcal{T}$.

To check if a is satisfiable, our algorithm will nondeterministically guess the start state as some tableau $(\mathcal{T}_0, \mathcal{F}_0)$ and the loop state $(\mathcal{T}_L, \mathcal{F}_L)$. It will then check that there is a path from $(\mathcal{T}_0, \mathcal{F}_0)$ to $(\mathcal{T}_L, \mathcal{F}_L)$ so that $Next$ holds on all pairs of adjacent tableaus. For each $\beta\ U\ \gamma \in untils(\mathcal{T}_L)$ there needs to be a state that satisfies γ. So the algorithm will make sure that there exists a loop around $(\mathcal{T}_L, \mathcal{F}_L)$ that does that.

More precisely, the algorithm does the following:

Guessa tableau $(\mathcal{T}_0, \mathcal{F}_0)$ such that $\alpha \in \mathcal{T}_0$ (and the tableau properties hold)
Guess a tableau $(\mathcal{T}_L, \mathcal{F}_L)$;
$(\mathcal{T}, \mathcal{F}) := (\mathcal{T}_0, \mathcal{F}_0)$;
while $(\mathcal{T}, \mathcal{F}) \neq (\mathcal{T}_L, \mathcal{F}_L)$ do
    $(\mathcal{T}, \mathcal{F}) := MOVE((\mathcal{T}, \mathcal{F}))$;
$(\mathcal{T}, \mathcal{F}) := MOVE((\mathcal{T}, \mathcal{F}))$;
$E := untils(\mathcal{T}_L)$;
while not ($E = \phi$ and $(\mathcal{T}, \mathcal{F}) = (\mathcal{T}_L, \mathcal{F}_L)$) do
    if $\beta\ U\ \gamma \in E$ and $\gamma \in \mathcal{T}$ then $E := E - \{\beta\ U\ \gamma\}$;
    $(\mathcal{T}, \mathcal{F})$:= $MOVE((\mathcal{T}, \mathcal{F}))$;
Accept

The operation $MOVE((\mathcal{T}, \mathcal{F}))$ is defined by the following procedure:

Guess a tableau $(\mathcal{T}', \mathcal{F}')$;
if$Next((\mathcal{T}, \mathcal{F}), (\mathcal{T}', \mathcal{F}'))$ then return $(\mathcal{T}', \mathcal{F}')$ else halt (rejecting this path)

Problem: this algorithm may not terminate. Solution: add a counter to count up to the number of all possible tableaus. If the counter is up and the `while`-loop is not done yet, it means that the loop is infinite and the input is to be rejected. (For the second `while`-loop we need to restart the counter every time a formula is removed from E.

$\square$

Example 11.3: Here's an example of a run of the algorithm. Let

$$\alpha = z\ U\ X\neg z \qquad \textit{– a little bugger}$$

$$\begin{array}{lcc}
\textit{the start state :} & \mathcal{T}_0 & \mathcal{F}_0 \\
& & \\
& z\ U\ X\neg z & X\neg z \\
& z & \neg z \\
& & \\
\textit{the loop state :} & \mathcal{T}_L & \mathcal{F}_L \\
& & \\
& z\ U\ X\neg z & \\
& z & \neg z \\
& X\neg z &
\end{array}$$

The algorithm proceeds with states as follows:

$$\begin{array}{lcc}
\textit{the start state :} & \mathcal{T}_0 & \mathcal{F}_0 \\
& & \\
& z\ U\ X\neg z & X\neg z \\
& z & \neg z \\
& & \\
\textit{next hit the loop state :} & \mathcal{T} & \mathcal{F} \\
& & \\
E = \{z\ U\ X\neg z\} & z\ U\ X\neg z & \\
& z & \neg z \\
& X\neg z & \\
& & \\
\textit{one more state in the loop :} & \mathcal{T} & \mathcal{F} \\
& & \\
E := E - \{z\ U\ X\neg z\} \rightarrow \phi & z\ U\ X\neg z & X\neg z \\
& \neg z & z \\
& & \\
\textit{came back to the loop state :} & \mathcal{T} & \mathcal{F} \\
& & \\
& z\ U\ X\neg z & \\
& z & \neg z \\
& X\neg z &
\end{array}$$

11.2. Model Checking

Remember Kripke model is (S, π, R); S is an (infinite) set of states, π is the truth assignment for variables at every state, R is a relation between states in S. Note that R defines a directed graph with states as nodes and thus defines paths as sequences of states.

Definition 11.4: LTL-Model Checking.

$M, s_0 \models \alpha$ if for every infinite path x starting at s_0, $M, x \models \alpha$. (Finite paths are ignored, the assumption being that in actual applications no path ends.)

Theorem 11.5: LTL-Model Checking is PSPACE-complete.

The proof is rather difficult. Compare with K-model checking which is in P.

11.3. Branching Time Logic

X, U, G - path operators.

E ("exists") and A ("forall") are new state operators.

State Formulas:

- include variables (denoted by z's)
- are closed under $\vee$, $\neg$ and under E, A

Path Formulas:

- include state formulas
- are closed under X, U, G

Semantics of state operators.

- $M, s \models z$ if $\pi(s)(z) = T$
- $M, s \models \neg\alpha$ same as before
- $M, s \models \alpha \vee \beta$ same as before
- $M, s \models E\alpha$ if there exists path x starting at s such that $M, x \models \alpha$ (LTL-relation)
- $M, s \models A\alpha$ if for all paths x starting at s it is true that $M, x \models \alpha$

Semantics of path operators. These are exactly as in LTL.