Computation Tree Logic

- Handles branching time
  - Add modal operators A, E for “on all paths”, “there exists a path”
  - $\phi ::= \bot \mid \top \mid p \mid (\neg \phi) \mid (\phi \land \phi) \mid (\phi \lor \phi) \mid (\phi \rightarrow \phi) \mid AX \phi \mid EX \phi \mid AF \phi \mid EF \phi \mid AG \phi \mid EG \phi \mid A[\phi U \phi] \mid E[\phi U \phi]$
  - A,E always paired with F,G,X,U

Semantics

- Propositional connectives as before
- $M,s \models AX \phi$ iff $\forall s_i$ such that $s \rightarrow s_i$ we have $M,s_i \models \phi$.
- $M,s \models EX \phi$ iff $\exists s_i$ such that $s \rightarrow s_i$ & $M,s_i \models \phi$.
- $M,s_i \models AG \phi$ holds iff $\forall$ paths $s_i \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$, and $\forall s_i$ along the path, $M,s_i \models \phi$.
- $M,s_i \models EG \phi$ holds iff there is a path $s_i \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$, and $\forall s_i$ along the path, $M,s_i \models \phi$.

Equivalences

- $\neg AF \phi \equiv EG \neg \phi$
- $\neg EF \phi \equiv AG \neg \phi$
- $\neg AX \phi \equiv EX \neg \phi$.
- $EG \psi \equiv \psi \land EX EG \psi$
- $E(\psi_1 U \psi_2) \equiv \psi_2 \lor (\psi_1 \land EX E(\psi_1 U \psi_2))$

*Last two provide basis for model-checking algorithm!*
Model-Checking

- Reason for considering is existence of efficient algorithm for determining where formulas true.
- Rather than figure out if $M, s_0 \models \phi$, instead, given $M, \phi$, determine set of states $s$ such that $M, s \models \phi$
- Involves computing fixed points of labeling function on finite graph.

Algorithm

- Use equivalences to convert $\phi$ to be written in terms of connectives $\text{AF}$, $\text{EU}$, $\text{EX}$, $\land$, $\neg$ and $\bot$
  - $\text{AG} \phi \equiv \neg \text{EF} \neg \phi$
  - $\text{EG} \phi \equiv \neg \text{AF} \neg \phi$
  - $\text{AF} \phi \equiv A[\top U \phi]$
  - $\text{EF} \phi \equiv E[\top U \phi]$
- Recursive labeling algorithm based on having labeled all subformulas first.

Algorithm

- Suppose have labeled states w/ all subformulas $\phi$ of $\psi$. Label those states satisfying $\psi$:
  - $\psi = \bot$: no states are labeled with $\bot$.
  - $\psi = p$: label $s$ with $p$ if $p \in L(s)$
  - $\psi = \psi_1 \land \psi_2$: label $s$ with $\psi_1 \land \psi_2$ if $s$ already labeled both with $\psi_1$ and with $\psi_2$.
  - $\psi = \neg \psi_1$: label $s$ with $\neg \psi_1$ if $s$ is not already labeled with $\psi_1$.

(More) Algorithm

- Label those states satisfying $\psi$ by:
  - $\psi = \text{AF} \psi_1$: If state $s$ labelled with $\psi_1$, label with $\text{AF} \psi_1$.
    Repeat: label any state with $\text{AF} \psi_1$ if all successor states are labelled with $\text{AF} \psi_1$, until there is no change.
  - $\psi = \text{E}[\psi_1 U \psi_2]$: If state $s$ labelled with $\psi_2$, label with $\text{E}[\psi_1 U \psi_2]$.
    Repeat: label any state with $\text{E}[\psi_1 U \psi_2]$ if labelled with $\psi_1$ and at least one of successor labelled with $\text{E}[\psi_1 U \psi_2]$, until there is no change.
  - $\psi = \text{EX} \psi_1$: label any state with $\text{EX} \psi_1$ if one of its successors is labelled with $\psi_1$. 
**Using Algorithm**

- If works, great.
  - Can provide even more refined specifications
- If it fails, can produce counterexample
  - State where algorithm does not make formula true
  - Helps debug system.

**Algorithm Complexity**

- $O(f \cdot V \cdot (V + E))$
  - where $f$ is number of connectives in $\phi$, $V = \#$ vertices, $E = \#$ edges
  - *note one subformula for each connective in $\phi$*
- Can get to $O(f \cdot (V + E))$ with more efficient algorithm for handling $EG$ instead of $AF$
  - See text.

**Example**

- Which states satisfy $AG(n_i \rightarrow EX t_j)$?
  - Look at equivalent: $\neg E[\top U \neg(n_i \rightarrow EX t_j)]$

**Model-Checking Summary**

- Model checking eschews proofs for validity.
- Floyd-Hoare logic:
  - Manual verification of proofs did not scale up well.
  - Not clear how to apply to concurrent or non-terminating programs.
- Model checking algorithmic!
- Formal tools available to help
# Turing Machines

- Grammars and machine models rich enough to represent every effective algorithm
- FSM’s no extra storage space
- PDA’s can use unbounded push-down stack
- Expand to unrestricted (but finite) storage

## Models

- Many possible:
  - RAM: FSM with potentially infinite memory directly addressable.
  - Turing Machine: FSM with potentially infinite (both directions) tape for storage.
  - TM historically most important, but RAM more natural today.
  - Many other models possible -- but all equivalent!!

## Beyond PDA’s

- Powerful enough to describe all computations
- Simple enough that we can reason formally about it
Turing Machines

- At each step, the machine must:
  - choose its next state,
  - write on the current square, and
  - move left or right.

Definition

- Turing machine $M$ is a sixtuple $(K, \Sigma, \Gamma, \delta, s, H)$:
  - $K$ is a finite set of states;
  - $\Sigma$ is the input alphabet, which does not contain $\square$;
    - $\square$ represents “blank”
  - $\Gamma \supseteq \Sigma \cup \{\square\}$ is the tape alphabet.
  - $s \in K$ is the initial state;
  - $H \subseteq K$ is the set of halting states;
  - $\delta$ is ...

Definition (cont)

- $\delta$ is the transition function:
  
  \[
  (K - H) \times \Gamma \rightarrow (K \times \Gamma \times \{\rightarrow, \leftarrow\})
  \]

- At each step, look at what is on tape and based on current state, move to new state, write replacement on tape, and move left or right.

Notes on Definition

- The input tape is infinite in both directions.
- $\delta$ is a function, so defining deterministic TMs.
- $\delta$ must be defined for all state, input pairs unless the state is a halting state.
- TMs do not necessarily halt.
- Turing machines generate output so can compute functions.
  - Takes contents of tape at start to contents at end.
Example

- Input to M is a string in \{a^i b^j, 0 \leq j \leq i\},
- Goal: adds b's to make \# b's = \# a's.

- Input to M looks like:

```
... \square a a a b \square \square \square ...
```

- Output should be:

```
... \square a a a b b b \square ...
```

TM Program

\(K = \{1, 2, 3, 4, 5, 6\}, \ \Sigma = \{a, b\}, \ \Gamma = \{a, b, \square, \$, \#\}, \ s = 1, \ H = \{6\}, \ \delta = \)