Examples in LTL

Lecture Scribe: Baichuan Zhou (baichuan@bu.edu)

March 19, 2015

Hints about homework (topological sorting)

Consider the following DAG:

[Diagram of a DAG]

If we project every node in the DAG onto the project line below, we will be able to “order” these nodes. \( x <_o y \) iff \( x \rightarrow y \). The partial order \( < \) defined by the DAG is the transitive closure of \( <_o \), i.e. \( < \) is \( <_o ^* \). (\( R^+ \) is transitive and \( R^* \) is transitive reflexive closure.)

Every partial order \( < \) can be embedded in a total order \( \preceq \). We can write this as \( < \subseteq \preceq \). In problem 1 in the homework, you should try to extend to infinite order.

Temporal logic (LTL))

(See LCS Page 178 Definition 3.4 for more details.) Consider the transition system \((S, \rightarrow, L)\), where:

1. \( S \) is a set of states;
2. \( \rightarrow \subseteq S \times S \). Instead of writing \((S, S') \in \rightarrow\), we write \( S \rightarrow S' \);
3. \( L : S \rightarrow \mathcal{P}(AP) \).

[Diagram of a transition system]

You should know that there are many ways of defining a transition system (automata). And there are many ways of using symbols/graphs to represent it. We can also use \((S, Act, \rightarrow, I, AP, L)\), where \( Act \) represents set of actions, \( I \) represents set of initial states, \( \rightarrow \subseteq S \times Act \times S \).

Example: In stead of writing \((S, a, S') \in \rightarrow\), we write \( S^a \rightarrow S' \).
Regular Set/Expression

- **Regular Sets/Expressions**
  In theoretical computer science and formal language theory, a **Regular Expression** (abbreviated regex or regexp and sometimes called a rational expression) is a sequence of characters that forms a search pattern, mainly for use in pattern matching with strings, or string matching, i.e. "find and replace"-like operations \[1\]. Using the rules of regular expression, we can get **Regular Set** like \{ab*\}. (This is the theory of finite automata, you will need to refer to the web for more details.) Specifically, if regular expression is denoted by \(E\), then \(L(E)\) represents Language/set corresponding to/defined by \(E\). \(L(ab^*) = \{a, ab, abb,\ldots\}\). We define alphabet \(\sum = \{a, b, c\}\), then:
  1. \(\alpha \in \sum\) is a regular expression.
  2. If \(E_1\) and \(E_2\) are regular expressionss, then \(E_1 + E_2\) is a regular expression and \(E_1 \cdot E_2\) is also a regular expression (\(\cdot\) means concatenate).
  3. If \(E\) is a regular expression, then \(E^*\) is also a regular expression. Example: \(E = ((a + b)^* + c) \cdot c\).

- **ω-regular Sets/Expressions**
  Ω-regular exp. \(G = E_1 F_1^\omega + E_2 F_2^\omega + \ldots + E_n F_n^\omega\), where each \(E_i\) and \(F_i\) is a regular expression and \(\epsilon \notin L(F_i)\) (\(\epsilon\) means empty). Definition of \(\omega\):

\[
\begin{cases}
  a^7 \\
  aaaaaaa \\
  a^w \\
  aaaa\ldots
\end{cases}
\]

\(L(G)\) is the \(\omega\) regular language/set defined by \(G\).

- **Regular expressions and automatas**: Regular expressions and languages are corresponded to finite automatas(FA), while \(\omega\)-regular expressions and languages are corresponded to Buchi automatas\[2\]. For the following example:

```
\begin{align*}
\text{start} & \rightarrow s_1 \rightarrow s_2 \rightarrow s_3 \\
\text{start} & \rightarrow s_1 \rightarrow s_2 \rightarrow s_3
\end{align*}
```
When it comes to FA, the corresponding accepted/recognized expression should be \((a + b)^*b(a + b)\). However, Buchi automata for here is different. It won’t accept anything. We should change it like this:

\[ E = (a + b)^*b(a + b)a^*, a \notin \mathcal{L}(E). \]

Acceptance/Recognition in a Buchi automata requires that the input string visits a final state infinitely often.

### Highlights of different Notations in the Scribe, Handout and the book

We didn’t spend much time on this during class. Please refer to this scribe, handout 18, LCS, and the web for different notations.

- A model \(\mathcal{M}\) of LTL is a **transition system**. A **path** \(\pi\) in \(\mathcal{M}\) is an infinite sequence of states \(s_1, s_2, s_3, \ldots\) such that \(s_1 \rightarrow s_2, s_2 \rightarrow s_3, s_3 \rightarrow s_4, \ldots\). We also write \(\pi \triangleq s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots\). For example (\(AP = \{q, p, r\}\)):

(Please note \(s_1/p, q\) means the name of the current state is \(s_1\) and \(p\) and \(q\) evaluates to \(T\) and \(r\) evaluates to \(F\) at this state). A more formal way of writing this should be \(\pi \triangleq s_{i_1} \rightarrow s_{i_2} \rightarrow s_{i_3} \rightarrow \ldots, \pi \models \varphi\), where \(\{i_1, i_2, i_3, \ldots\} \subseteq \{1, 2, 3, 4\}\).

- We also talked about some other notation differences on pages 3-8 in handout 18. For example, on page 6, instead of writing \(X, G, F\), we sometimes use \(\bigotimes, \Box\) and \(\lozenge\) respectively. On page 3, we should also get to know the notation of \(U\)(until), \(W\)(weak until) and \(R\)(release).

### References: