Lexical Analysis

(Part 2)

Lexical analysis is the first phase of compilation: The file is converted from ASCII to tokens. It must be fast!

Building Scanners with REs

- Convert RE specification into a finite state automaton (FSA) (aka FA)
- Convert FSA into a scanner implementation
  - By hand into a collection of procedures
  - Mechanically into a table-driven scanner

Finite State Automata

• A Finite State Automaton has
  - A set of states
    - One marked initial
    - Some marked final
  - A set of transitions from state to state
    - Each labeled with an alphabet symbol or ε
  - Operate by beginning at the start state, reading symbols and making indicated transitions
    - If no transition with a matching label is found, reject
    - When input ends, accept if in final state, otherwise reject

Our example from class (with state numbers added):

This figure represents a DFA even though it is not complete (i.e., not all state-character transitions have been drawn). The complete DFA is:

but it is very common to ignore state 0 (called the error state) since it is implied. The error state serves as a black hole, which doesn’t let you escape.

Determinism

- FSA can be deterministic or nondeterministic
- Deterministic: always know uniquely which edge to take
  - At most 1 arc leaving a state with a given symbol
  - No ε arcs
- Nondeterministic: may need to guess or explore multiple paths, choosing the right one later

NFAs vs DFAs

- A problem:
  - REs (e.g., specifications) map easily to NFAs
  - ...
  - Can write code for DFAs easily
- How to bridge the gap?
- Can it be bridged?
A Solution

• Cool algorithm to translate any NFA to a DFA
  – Proves that NFAs aren’t any more expressive

• Plan:
  1) Convert RE to NFA
  2) Convert NFA to DFA
  3) Convert DFA to code

• Can be done by hand or fully automatically

RE => NFA

Construct Cases Inductively

NFA => DFA

• Subset Construction
  – Construct a DFA from the NFA, where each state in the DFA represents a set of states from the NFA

• Key Idea:
  – The state of the DFA after reading some input is the set of all states the NFA could have reached after reading the same input.

Subset Construction Algorithm (NFA => DFA)

Given NFA with states and transitions:
  – label all NFA states uniquely
Create start state of DFA:
  – label it with the set of NFA states (e.g. \{s_1, ..., s_n\}) reachable from the start state of the NFA by \( \varepsilon \) transitions, i.e. w/o consuming input.
  – Add this new start state to the WorkList.

while (WorkList is not empty) {
  Remove a state \( S \) with label \( \{s_1, ..., s_n\} \) from the WorkList.
  For each symbol \( x \) in the alphabet:
  – Compute the set \( \{t_1, ..., t_m\} \) of NFA states reached from any of the NFA states in \( \{s_1, ..., s_n\} \) by an \( x \) transition (followed by any number of \( \varepsilon \) transitions – a.k.a. the E-closure).
  – If \( \{t_1, ..., t_m\} \) is not empty:
    • If a DFA state \( T \) labeled \( \{t_1, ..., t_m\} \) already exists, add a \( x \) transition from \( S \) to \( T \).
    • Else create new DFA state \( T \) labeled \( \{t_1, ..., t_m\} \), add a \( x \) transition from \( S \) to \( T \),
      add \( T \) to the WorkList.

A DFA state is final iff at least one of the NFA states in its label is final.

Subset Construction