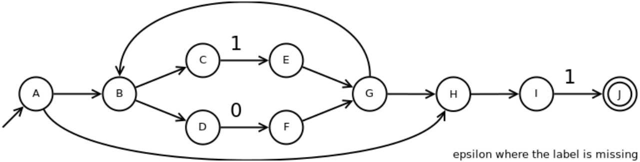
NFA to DFA

NFA 2 DFA

Given an NFA accepting a language $\mathscr{L}$ there exists a DFA accepting the same language

- The derivation of a DFA from an NFA is based on the concept of ϵ-*closure*
- The subset construction algorithm makes the transformation using the following operations:
  - $\epsilon\text{-}closure(s)$ with $s \in \mathcal{S}$
  - $\epsilon\text{-}closure(\mathcal{T}) = \bigcup_{s \in \mathcal{T}} \epsilon\text{-}closure(s)$ where $\mathcal{T} \subseteq \mathcal{S}$
  - $move(\mathcal{T}, a)$ with $\mathcal{T} \subseteq \mathcal{S}$ and $a \in \Sigma$

NFA to DFA



- build the ϵ-*closure*$(\ldots)$ for different states/sets
- build *move*$(\mathcal{T}, a)$ for different sets and elements

NFA to DFA

Subset Construction Algorithm

The Subset Construction algorithm permits to derive a DFA $\langle \mathcal{S}, \Sigma, \delta_D, s_0, \mathcal{F}_D \rangle$ from an NFA $\langle \mathcal{N}, \Sigma, \delta_N, n_0, \mathcal{F}_N \rangle$

$s_0 \leftarrow \epsilon\text{-}closure(\{n_0\})$; $\mathcal{S} \leftarrow \{s_0\}$; $\mathcal{F}_D \leftarrow \varnothing$; worklist $\leftarrow \{s_0\}$;
if $(s_0 \cap \mathcal{F}_N \neq \varnothing)$ **then** $\mathcal{F}_D \leftarrow \mathcal{F}_D \cup s_0$;
end if
while (worklist $\neq \varnothing$) **do**
  take and remove q from worklist;
  for all $(c \in \Sigma)$ **do**
    $t \leftarrow \epsilon\text{-}closure(move(q, c))$;
    $\delta_D[q, c] \leftarrow t$;
    if $(t \notin \mathcal{S})$ **then**
      $\mathcal{S} \leftarrow \mathcal{S} \cup t$; worklist $\leftarrow$ worklist $\cup t$;
    end if
    if $(t \cap \mathcal{F}_N \neq \varnothing)$ **then** $\mathcal{F}_D \leftarrow \mathcal{F}_D \cup t$;
    end if
  end for
end while

Simulating DFA and NFA

DFA

$s = s_0$;
$c = nextChar()$;
while (c $\neq$ **eof**) **do**
    $s = move(s, c)$;
    $c = nextChar()$;
end while
if ($s \in \mathcal{F}$) **then return** "yes";
else return "no";
end if

NFA

$S = \epsilon\text{-}closure(s_0)$;
$c = nextChar()$;
while (c $\neq$ **eof**) **do**
    $S = \epsilon\text{-}closure(move(S, c))$;
    $c = nextChar()$;
end while
if ($S \cap \mathcal{F} \neq \varnothing$) **then return** "yes";
else return "no";
end if

Exercises NFA to DFA

- Derive an NFA for the regexp: $(a|b)^*abb$
- NFA to DFA for the obtained NFA

Exercises NFA to DFA

- Derive an NFA for the regexp: $(a|b)^*abb$
- NFA to DFA for the obtained NFA

DFA to Minimal DFA

Note

Reducing the size of the automaton does not reduce the number of moves needed to recognise a string, nevertheless it reduces the size of the transition table that could more easily fit the size of a cache

Equivalent states

Two states of a DFA are equivalent if they produce the same "behaviour" on any input string.

Let $\mathcal{D} = \langle \mathcal{S}, \Sigma, \delta, q_0, \mathcal{F} \rangle$ be a DFA. Two states s_i and s_j of $\mathcal{D}$ are considered equivalent, written $s_i \equiv s_j$, **iff**

$$\forall \mathbf{x} \in \Sigma^*.(s_i \xrightarrow{\mathbf{x}} s_i' \land s_i' \in \mathcal{F}) \iff (s_j \xrightarrow{\mathbf{x}} s_j' \land s_j' \in \mathcal{F})$$

DFA to Minimal DFA – Partition Refinement Algorithm

Deriving a minimal DFA

Transform a DFA $\langle \mathcal{S}, \Sigma, \delta_D, s_0, \mathcal{F}_D \rangle$ into a minimal *DFA* $\langle \mathcal{S}', \Sigma, \delta'_D, s'_0, \mathcal{F}'_D \rangle$

// Π *is a partition of the set of states* $\mathcal{S}$
$\Pi \leftarrow \{\mathcal{F}_D, \mathcal{S} - \mathcal{F}_D\}$ // *Initially there are only two groups of states: final states and non-final states*
repeat
  $\Pi_{\text{new}} \leftarrow \Pi$ // *create a working copy* Π_{new}
  for all groups G in Π **do**
    partition G in subgroups $G_1, \ldots, G_n$ ($n \geq 1$) such that two states s and t are in the same subgroup G_i iff
    $\forall c \in \Sigma\ ((s \stackrel{c}{\not\longrightarrow}) \wedge (t \stackrel{c}{\not\longrightarrow})) \vee ((s \stackrel{c}{\longrightarrow} s') \wedge (t \stackrel{c}{\longrightarrow} t') \wedge (s', t' \in \bar{G})$ for some group $\bar{G}$ in Π)
    // *subgroups* G_i*'s may be composed of only one state*
    $\Pi_{\text{new}} \leftarrow \Pi_{\text{new}} - G \cup \{G_1, \ldots, G_n\}$ // *Replace G with the obtained subgroups in* Π_{new}
    // *the partition is refined: the group G is possibly replaced with a finer partition* $G_1, \ldots, G_n$
  end for
until $\Pi_{\text{new}} = \Pi$ // *exit when the partition cannot be refined further*
// *Now* Π *contains a set of groups that are a partition of the states* $\mathcal{S}$
// *The algorithm continues with the construction of the minimal DFA . . .*

DFA to Minimal DFA – Partition Refinement Algorithm

```
// Continues from the previous slide . . .
// the states of the minimal DFA are representatives of groups of equivalent states, those that are in Π
S' ← ∅ and F'_D ← ∅
for all groups G in Π do
    choose a state in G as the representative for G and add it to S'
    if G ∩ F_D ≠ ∅ //  G contains either all final states or all non-final states then
        add the representative state for G also to F'_D
    end if
end for
s'_0 ← the representative state of the group G containing s_0
for all states s ∈ S' do
    for all charachters c ∈ Σ do
        if δ_D[s, c] is defined then
            δ'_D[s, c] ← the representative state of the group G containing the state δ_D[s, c]
        end if
    end for
end for
```

Uniqueness of the minimal DFA

There exists a unique DFA, up to isomorphism, that recognises a regular language $\mathscr{L}$ and has minimal number of states. Two DFA are isomorphic iff they are equal by neglecting the labels of the states.

Exercises

RegExp 2 DFA

- Minimise the DFA for the regexp $(a|b)^*abb$
- Consider the regexp $a(b|c)^*$ and derive the minimal accepting DFA
- Define an automated strategy to decide if two regular expressions define the same language combining the algorithms defined so far

Regular Languages properties

- Specify a DFA accepting all strings of a's and b's that do not contain the substring aab
- Show that the complement of a regular language, on alphabet Σ, is still a regular language
- Show that the intersection of two regular languages, on alphabet Σ, is still a regular language

Summary

Lexical Analysis

Relevant concepts we have encountered:

- Tokens, Patterns, Lexemes
- Chomsky hierarchy and regular languages
- Regular expressions
- Problems and solutions in matching strings
- DFA and NFA
- Transformations
  - RegExp $\rightarrow$ NFA
  - NFA $\rightarrow$ DFA
  - DFA $\rightarrow$ Minimal DFA