DFA minimization

Hopcroft's algorithm

문순원

Lexer generator's work

From regex to DFA

- convert regex into ε-NFA (Thompson's construction)
- convert ε-NFA into NFA (Dealing with ε-closure)
- convert NFA into DFA (Powerset construction)
- minimize DFA (Hopcroft's algorithm)

Lexer generator's work

From regex to DFA

- convert regex into ε-NFA (Thompson's construction)
- convert ε-NFA into NFA (Dealing with ε-closure)
- convert NFA into DFA (Powerset construction)
- minimize DFA (Hopcroft's algorithm)

Definition of minimal DFA and some facts about it

DFA M is said to be *minimal* if no DFA with fewer states recognizes $L(M)$.

The minimial DFA for any recognizable language is unique up to isomorphism.

Given an arbitrary DFA M, one can effectively construct the minimal DFA equivalent to M.

State equivalence and reduced DFA

Let DFA $M = (Q, \Sigma, \delta, q_0, F)$

Two states a and b are *equivalent* if $(a, b) \in \rho_M$ where

$$\rho_M = \{(a, b) \mid (\forall w \in \Sigma^*)\ (\delta_w(a) \in F \Leftrightarrow \delta_w(b) \in F)\}$$

State equivalence and reduced DFA

Let DFA $M = (Q, \Sigma, \delta, q_0, F)$

Two states a and b are *equivalent* if $(a, b) \in \rho_M$ where

$$\rho_M = \{(a, b) \mid (\forall w \in \Sigma^*)\ (\delta_w(a) \in F \Leftrightarrow \delta_w(b) \in F)\}$$

DFA M is *reduced* if equivalence implies identity. i.e.

$$(a, b) \in \rho_M \Rightarrow a = b$$

State equivalence and reduced DFA

Let DFA $M = (Q, \Sigma, \delta, q_0, F)$

Two states a and b are *equivalent* if $(a, b) \in \rho_M$ where

$$\rho_M = \{(a, b) \mid (\forall w \in \Sigma^*)\ (\delta_w(a) \in F \Leftrightarrow \delta_w(b) \in F)\}$$

DFA M is *reduced* if equivalence implies identity. i.e.

$$(a, b) \in \rho_M \Rightarrow a = b$$

Reduced DFA is minimal!

How can we construct reduced DFA?

Congruence and quotient of DFA

An equivalence relation $\rho \subseteq Q \times Q$ is a *congruence* of M if

- δ_x is compatible with ρ
  $(a, b) \in \rho \Rightarrow (\delta_x(a), \delta_x(b)) \in \rho$
- F is a union of some ρ-classes (ρ saturates F)
  $[a]_\rho \in F/\rho \Rightarrow [a]_\rho \subseteq F$

Congruence and quotient of DFA

An equivalence relation $\rho \subseteq Q \times Q$ is a *congruence* of M if

- δ_x is compatible with ρ
  $(a, b) \in \rho \Rightarrow (\delta_x(a), \delta_x(b)) \in \rho$
- F is a union of some ρ-classes (ρ saturates F)
  $[a]_\rho \in F/\rho \Rightarrow [a]_\rho \subseteq F$

The *quotient DFA* M/ρ by a congruence ρ is defined as follows

$$M/\rho = (Q/\rho, \Sigma, \delta/\rho, [q_0]_\rho, F/\rho)$$
$$\textit{where} \quad (\delta/\rho)_x([a]_\rho) = [\delta_x(a)]_\rho$$

Quotient preserves recognizable language. i.e. $L(M) = L(M/\rho)$

Congruence and quotient of DFA

An equivalence relation ρ is coarser than π if

$$(a, b) \in \pi \Rightarrow (a, b) \in \rho$$

Congruence and quotient of DFA

An equivalence relation ρ is coarser than π if

$$(a, b) \in \pi \Rightarrow (a, b) \in \rho$$

Following two statements are true

- ρ_M is the coarsest congruence
- M/ρ_M is reduced. Hence it's minimal

The classical minimization algorithm

We can find ρ_M by following iterative algorithm

Let ρ_i be a sequence of equivalence relation on Q such that

$$
\begin{aligned}
\rho_0 &= \{(a,b) \mid a,b \in F\} \cup \{(a,b) \mid a,b \in Q \setminus F\} \\
\rho_{i+1} &= \{(a,b) \in \rho_i \mid (\forall x \in \Sigma)\ (\delta_x(a), \delta_x(b)) \in \rho_i\}
\end{aligned}
$$

The classical minimization algorithm

We can find ρ_M by following iterative algorithm

Let ρ_i be a sequence of equivalence relation on Q such that

$$\begin{aligned} \rho_0 &= \{(a,b) \mid a,b \in F\} \cup \{(a,b) \mid a,b \in Q \setminus F\} \\ \rho_{i+1} &= \{(a,b) \in \rho_i \mid (\forall x \in \Sigma)\ (\delta_x(a), \delta_x(b)) \in \rho_i\} \end{aligned}$$

Then there exists $k \in \mathbb{N}$ s.t. $\rho_{k+1} = \rho_k$ and
ρ_k is the coarsest congruence of M. (i.e. $\rho_M = \rho_k$)

Hopcroft's algorithm

$Q/\rho \leftarrow \{F, Q \setminus F\}$
$L \leftarrow \{F\}$
while there exists $A \in L$ do
  $L \leftarrow L \setminus \{A\}$
  for each $x \in \Sigma$ do
    let $X = \delta_x^{-1}(A)$
    for each $Y \in Q/\rho$ s.t. $(Y' = Y \cap X \neq \emptyset) \wedge (Y'' = Y \setminus X \neq \emptyset)$ do
      $Q/\rho \leftarrow (Q/\rho \setminus \{Y\}) \cup \{Y', Y''\}$
      if $Y \in L$ then
        $L \leftarrow (L \setminus \{Y\}) \cup \{Y', Y''\}$
      else
        $L \leftarrow L \cup \{\min(Y', Y'')\}$
    end
  end
end

Hopcroft's algorithm

$Q/\rho \leftarrow \{F, Q \setminus F\}$
$L \leftarrow \{F\}$
while there exists $A \in L$ do
    $L \leftarrow L \setminus \{A\}$
    for each $x \in \Sigma$ do
        let $X = \delta_x^{-1}(A)$
        for each $Y \in Q/\rho$ s.t. $(Y' = Y \cap X \neq \emptyset) \wedge (Y'' = Y \setminus X \neq \emptyset)$ do
            $Q/\rho \leftarrow (Q/\rho \setminus \{Y\}) \cup \{Y', Y''\}$
            if $Y \in L$ then
                $L \leftarrow (L \setminus \{Y\}) \cup \{Y', Y''\}$
            else
                $L \leftarrow L \cup \{\min(Y', Y'')\}$
        end
    end
end

$L \subseteq Q/\rho$ is a loop invariant!

Modified version

Let $R = Q/\rho \setminus L$

$R \leftarrow \{Q \setminus F\}$
$L \leftarrow \{F\}$
while there exists $A \in L$ do
  $L \leftarrow L \setminus \{A\}$
  $R \leftarrow R \cup \{A\}$
  for each $x \in \Sigma$ do
    let $X = \delta_x^{-1}(A)$
    for each $Y \in R$ s.t. $(Y' = Y \cap X \neq \emptyset) \wedge (Y'' = Y \setminus X \neq \emptyset)$ do
      $R \leftarrow (R \setminus \{Y\}) \cup \{\max(Y', Y'')\}$
      $L \leftarrow L \cup \{\min(Y', Y'')\}$
    end
    for each $Y \in L$ s.t. $(Y' = Y \cap X \neq \emptyset) \wedge (Y'' = Y \setminus X \neq \emptyset)$ do
      $L \leftarrow (L \setminus \{Y\}) \cup \{Y', Y''\}$
    end
  end
end

Contribution to open source project

Alex: A lexical analyser generator for Haskell

- Found some inefficiency in the alex's implementation of Hopcroft's algorithm

Contribution to open source project

Alex: A lexical analyser generator for Haskell

- Found some inefficiency in the alex's implementation of Hopcroft's algorithm
- I improved it in various ways
  1. Use better data structure and functions
  2. Exclude trivial cases
  3. The idea described before in this presentation

`https://github.com/haskell/alex/pull/176`

Contribution to open source project

Alex: A lexical analyser generator for Haskell

- Found some inefficiency in the alex's implementation of Hopcroft's algorithm
- I improved it in various ways
    1. Use better data structure and functions
    2. Exclude trivial cases
    3. The idea described before in this presentation

`https://github.com/haskell/alex/pull/176`

Most performance gain was came from 2.

References

Re-describing an algorithm by Hopcroft(Timo Knuutila, 2001)