Regular Expressions, Grammars and Decidability

Marco Carbone

IT University of Copenhagen

November 2014

Course Evaluation

- Do the course evaluation

Regular Expressions (I)

FA's recognizes sets (languages), but what kind of sets do they recognize? And, what sets don't they recognize?

A **regular expression** over an alphabet Σ is set defined as:

- a is a regular expression if $a \in \Sigma$
- ϵ is a regular expression
- $\emptyset$ is a regular expression
- $(A + B)$ is a regular expression if A and B are regular expressions
- (AB) is a regular expression if A and B are regular expressions
- A^* is a regular expression if A is a regular expression

A^* represents the ***Kleene closure*** of the set represented by A.

Regular Expressions (II)

Regular expressions are often used in software dealing with text, like editors when searching for strings matching certain regular patterns.

Sets represented by a regular expression are called **regular sets**.

Example Let $\Sigma = \{0, 1\}$, then

$(0 + \epsilon)(1 + \epsilon)$ corresponds to $\{\epsilon, 0, 1, 01\}$
01^* corresponds to $\{0\omega \in \Sigma^* : \omega \in \{1\}^*\}$
0^*10^* corresponds to $\{\omega \in \Sigma^* : \omega$ contains exactly a single $1\}$
$(0 + 1)^*1(0 + 1)^*$ corresponds to $\{\omega \in \Sigma^* : \omega$ contains at least one $1\}$
$(0 + \epsilon)1^*$ corresponds to $01^* + 1^*$

Regular Expressions (III)

Exercise For each of the two following regular expressions, give two strings that are member of the language it represents, and give two that aren't:

$$a^*b^* \qquad a(ba)^*a$$

Exercise Give regular expressions for the intersection, union, and concatenation respectively of the two languages:
$A = \{\omega \in \{0,1\}^* : \omega \text{ begins with } 11\}$ and
$B = \{\omega \in \{0,1\}^* : \omega \text{ ends with } 00\}$.

Exercise Give a regular expression for decimal digits.

Exercise Let R be a regular expression over some set.

Do $(R + \emptyset)$ and $(R\epsilon)$ denote the same set?
What set does $(R + \epsilon)$ represent?
What set does $(R\emptyset)$ represent?

Kleene's Theorem (I)

FA's and regular expressions are equally expressive:

Theorem R is a regular set *if and only if* $R = L(M)$ for some FA M.

Proof. Idea: Construct an automaton for each regular expression type (structural induction). Look at the book for details!

What are Grammars?

- How may a language be defined?
- A **grammar** is a model for defining languages. I.e. it's a model used to generate the valid strings of a language.
- FA's recognise languages, but how to generate a language? It's generally impossible to list all elements of a language. How to list all possible Java programs?
- A **grammar** is a model that generates the elements of a language.

Example Let $\{0, 1\}$ be ***terminal symbols*** and let $\{S, T\}$ be ***non-terminals***. The grammar with ***productions***

$$S \to 0S \qquad S \to 1S \qquad S \to 1T \qquad T \to 01 \qquad T \to 1$$

generates the language $L = \{\omega \in \{0, 1\}^* : \omega$ ends with 11 or $101\}$.

E.g. $010101 \in L$ because of the ***derivation*** from ***start symbol*** S

$$S \Rightarrow 0S \Rightarrow 01S \Rightarrow 010S \Rightarrow 0101T \Rightarrow 010101$$

Grammars (II)

Example A sample of the productions from a grammar for a programming language:

$$
\begin{aligned}
&Program ::= program\ Identifier\ Heading\ ;\ Block\\
&Heading ::= \epsilon \mid (\ IdentifierList\ )\\
&IdentifierList ::= Identifier \mid IdentifierList\ ,\ Identifier\\
&Identifier ::= \ldots\\
&Block ::= \ldots\\
&\ldots\\
&VarDecl ::= var\ VarIdList\ :\ Type \mid VarDecl\ ;\ VarIdList\ :\ Type\\
&VarIdList ::= Identifier \mid VarIdList\ ,\ Identifier\\
&Type ::= SimpleType \mid StrucType \mid \hat{}\,Typeid\\
&StrucType ::= array\,[\ IndexList\ ]\ of\ Type \mid file\ of\ Type \mid \ldots\\
&\ldots
\end{aligned}
$$

Grammars (III)

Formally, $G = (V, T, S, P)$ is a grammar where

- V is a finite set, the ***vocabulary***,
- $T \subseteq V$ is a finite set of ***terminal*** symbols,
- $S \in V$ is the ***start symbol***,
- $P \subseteq V^* \times V^*$ is the set of ***productions*** $\omega \to \omega'$ with ω containing at least on nonterminal.

- $N = V \setminus T$ is the set of ***non-terminal*** symbols.
- If $lz_0r, lz_1r \in V^*$ and if $z_0 \to z_1 \in P$ then lz_1r is **directly derivable** from lz_0r denoted by $lz_0r \Rightarrow lz_1r$.
- If there exists a **derivation** $\omega_0 \Rightarrow \omega_1 \Rightarrow \ldots \omega_n$ we say that ω_n is **derivable** from ω_0 and write $\omega_0 \Rightarrow^* \omega_n$.
- $L(G) = \{\omega \in T^* : S \Rightarrow^* \omega\}$ is the **language generated by** G.

Grammars (IV)

Example Let $G_1 = (\{a, b, S, T, U\}, \{a, b\}, S, P)$ where P is defined by

$$S \to a \mid b \mid aT \mid aU \mid bT \mid bU \qquad T \to a \qquad U \to b$$

$\{a, b, aa, ab, ba, bb\}$ is the language generated by G_1.

Example Let G_2 be G_1 where P is changed to

$$S \to T \mid U \qquad T \to aTb \mid \epsilon \qquad U \to bUa \mid \epsilon$$

Then $L(G_2) = \{a^n b^n : n \geq 0\} \cup \{b^n a^n : n \geq 0\}$.

Exercise What's the language generated by G_1 if P is altered to:

$$S \to T \qquad S \to bSb \qquad T \to aT \qquad T \to \epsilon$$

Exercise Make a grammar generating $\{a^n b^{2n} : n \geq 0\}$.

Classifying Grammars

The grammars we have seen so far belong to two important classes.

G is a **regular grammar** if all productions have one of the forms:

$$S \to \epsilon \ , \quad A \to a \ , \quad A \to aB$$

G is a **context free grammar** if all productions have the form:

$$A \to \omega$$

Example. All regular grammars are also context free, but not all grammars are context free:
$G_3 = (\{a, b, c, S, T, U\}, \{a, b, c\}, S, P)$ where P is defined by

$$S \to aSTU \mid \epsilon \quad UT \to TU \quad aT \to ab \quad bT \to bb \quad bU \to bc \quad cU \to cc$$

is **context sensitive**. $L(G_3) = \{a^n b^n c^n : n \geq 0\}$.

Regular Sets vs. Regular Grammars (I)

Theorem R is a regular set if and only if $R = L(G)$ for some regular grammar G.

'if': Suppose a regular grammar G, recall it has productions on the form

$$S \to \epsilon \ , \quad A \to a \ , \quad A \to aB$$

Using Kleene's theorem it's sufficient to build a NFA N recognizing $L(G)$. Create a state, q_A, for each non-terminal A (q_S being the initial state) and add a single accepting state q_{acc}. Add transitions

$$q_B \in \delta(q_A, a) \ \text{ if } \ A \to aB \qquad\qquad q_{acc} \in \delta(q_A, a) \ \text{ if } \ A \to a$$

Exercise Argue why $L(G) = L(N)$ for the NFA N constructed from G in the proof sketched above.

Reg Sets vs. Reg Grammars (II)

'only if': Suppose R is a regular set, due to Kleene's theorem there exists an FA M recognizing R. Construct regular grammar G from M.

Assume the initial state q_0 of M has no incomming transitions. Select a non-terminal A_q for each state q in M. The start symbol is A_{q_0}. Add productions

$$A_q \to aA_{q'} \text{ if } q' = \delta(q,a) \ ,$$
$$A_q \to a \text{ if } q' = \delta(q,a) \text{ and } q' \text{ is accepting.}$$

Exercise Argue why $L(M) = L(G)$ for the regular grammar G constructed from M in the proof sketch above.

Exercise Argue that any FA M is equivalent to a NFA where the initial state has no incomming transitions.

Decidability

- Some languages can't be recognized by FA's (and generated by regular grammars), e.g. $A = \{0^n 1^n : n = 0, 1, 2, \ldots\}$ is ***not*** regular.

- But, since we can construct a real computer program to recognize A, we need a more powerful computational model than FA's.

- At this point, we should consider the ultimate computational model, called a **Turing Machine**. However, because we do not have enough time to introduce Turing machines, we consider an equivalent computational model:

  A standard programming language with basic statements and infinite memory

Church-Turing Thesis

Every computable function (for which an algorithm exists) can be computed by some Turing Machine

Turing Recognizable

A program P can either **accept** an input ω or **reject** it.

The set of strings accepted by P, $L(P)$, is the language **recognized** by P. A language is **Turing recognizable** if there exists some program recognising it.

Exercise Is $B = \{a^n b^n c^n : n = 0, 1, 2, \ldots\}$ Turing recognizable?

Turing Decidable

A program may ***loop***, because either it terminates (accepting or rejecting), or it doesn't terminate.

A program may fail to accept an input by either entering a rejecting configuration or by looping.

A non-looping program is called a **decider**, it always accepts or rejects an input.

A decider that recognises a language L is said to **decide** L. A language is **decidable** if some program decides it.

Example Can you write a program that decides $A = \{0^n 1^n : n = 0, 1, 2, \ldots\}$...if so A is decidable.

The Halting Problem (I)

Not all problems (languages) can be solved (decided) by a program. I.e., by Church-Turing thesis, some problems can't be solved by algorithms running on a computer :-(

Let $\langle P \rangle$ be a string encoding a program P, e.g., the code saved in a text file. Consider the **Universal Program**:

$U =$ " On input $\langle P \rangle$ and ω where P is a program and ω it's input:
i) simulate P on ω.
ii) if P accepts, accept
iii) if P rejects, reject "

U can take any P and its input ω as input and simulate P on ω.

U is a recognizer, but not a decider, for

$$Halt = \{(\langle P \rangle, \omega) : P \text{ is a program and } P \text{ accepts } \omega\}$$

because U loops if (and only if) P loops on ω.

The Halting Problem (II)

Theorem $Halt = \{(\langle P \rangle, \omega) : P$ is a TM and P accepts $\omega\}$ is undecidable.

Proof (by contradiction) Suppose there is a program H that decides $Halt$. I.e. H on input $\langle P \rangle$ and ω is defined by:

$$H(\langle P \rangle, \omega) = \begin{cases} accept, & \text{if } P \text{ accepts } \omega \\ reject, & \text{if } P \text{ doesn't accept } \omega \end{cases}$$

If that is the case, construct another program D s.t.

$D =$ " On input $\langle P \rangle$ where P is a program:
i) run H on $\langle P \rangle$ and $\langle P \rangle$.
ii) if H accepts, reject
iii) if H rejects, accept "

The Halting Problem (III)

The definition of D can be rewritten as:

$$D(\langle P \rangle) = \begin{cases} accept, & \text{if } P \text{ doesn't accept } \langle P \rangle \\ reject, & \text{if } P \text{ accepts } \langle P \rangle \end{cases}$$

What if we run D with $\langle D \rangle$ as input?

$$D(\langle D \rangle) = \begin{cases} accept, & \text{if } D \text{ doesn't accept } \langle D \rangle \\ reject, & \text{if } D \text{ accepts } \langle D \rangle \end{cases}$$

I.e.

- if D accepts $\langle D \rangle$ then D doesn't accept $\langle D \rangle$, and
- if D rejects $\langle D \rangle$ then D accepts $\langle D \rangle$.

so we have a contradiction, and hence D and neither H can exist.