COMP4141 Theory of Computation

Lecture 3 On Proofs, Regular Expressions

Ron van der Meyden

CSE, UNSW

Revision: 2013/03/11

(Credits: David Dill, Thomas Wilke, Kai Engelhardt, Peter Höfner, Rob van Glabbeek)

Proof Expectations

We don't want to lose sight of the forest because of the trees. Here are the "forest-level" points with proofs.

- What is the proof strategy?
  - Induction on strings. *What are the base and induction steps?*
  - Induction on expressions. *What are the base and induction steps?*
  - Diagonalization
  - Reduction from another problem. *Which direction is the reduction?*
- What are the key insights in the proof?
- Often this is a construction (often something that can be implemented as a computer program)
  - Translation between regular expressions, various finite automata.
  - Translation from one problem to another.

Explain these things clearly in your proofs. If we can see *quickly* that you did the right kind of proof and got the major points right, you may get nearly full marks.

Proof Guidelines

1. State *what* is being proved precisely and clearly.
2. Start proof with an explanation of the *strategy* (e.g. "induction on y")
3. Provide guideposts (e.g. *Base*, *Induction*)
4. Highlight the interesting key parts of the proof (where did you have to be clever?)
5. Make it easy for the graders to see these things.

NB

Use Sipser's proofs as blueprints. As beginners, you need to provide more detail than he typically does. The license to be brief has to be earned by repeatedly demonstrating the capability of filling in all omitted detail. Do not omit detail your average reader/fellow student cannot be expected to fill in.

Regular Expressions

Regular expressions are an *algebraic notation* for regular languages.
Extensively used

- String pattern matching utilities (e.g. `grep` in unix)
- Computer language definitions ("lexical structure")
- Compiler generation tools

Regular expressions are a completely different "formalism" from finite automata — but they have *exactly the same expressive power*.
Tools based on regular expressions usually translate them to finite automata, which are more suitable than regular expressions for many applications.

Concatenation of Languages

If L_1 and L_2 are languages, we can define their *concatenation* $L_1 L_2$ to be $\{ xy \mid x \in L_1 \wedge y \in L_2 \}$.

Examples

What is $\{ab, ba\}\{cd, dc\}$? $\{abcd, abdc, bacd, badc\}$
What is $\emptyset\{ab, bc\}$? $\emptyset$

Exponentiation of Languages

L^i is the language L concatenated with itself i times.

Definition

Base: $L^0 = \{\epsilon\}$.
Induction: $L^{i+1} = LL^i$.

Examples

$\{ab, ba\}^2 = \{abab, abba, baab, baba\}$
$\emptyset^0 = \{\epsilon\}$
$\emptyset^2 = \emptyset$

Kleene Closure

$$L^* = \bigcup_{i \in \mathbb{N}} L^i = L^0 \cup L^1 \cup \ldots$$

This operator can build infinite sets from finite sets.

Examples

$\{ab, ba\}^* = \{\epsilon, ab, ba, abab, abba, \ldots\}$
$\emptyset^* = \{\epsilon\}$
$\{\epsilon\}^* = \{\epsilon\}$

Regular Expression Definition

Regular expressions are defined relative to some alphabet Σ.
This is a recursive definition of the structure of regular expressions.
The structure of regular expressions is the basis for other recursive definitions and induction proofs. Every recursive definition and proof has to handle these six cases:

- a is a regular expression, if $a \in \Sigma$.
- $\emptyset$ is a regular expression.
- ϵ is a regular expression.
- $R_1 \cup R_2$ is a regular expression if R_1 and R_2 are.
- $R_1 \circ R_2$ is a regular expression if R_1 and R_2 are.
- R^* is a regular expression if R is.

Parentheses can be added in the obvious places to override precedence: * has the highest precedence, followed by $\circ$, and finally $\cup$ which has the lowest precedence, so $a \cup b \circ c^* = a \cup (b \circ (c^*))$. The first three definitions can be considered base cases while the last three are inductive.

Concise Regular Expression Definition

Using so called *EBNF* (for *extended Backus-Naur form*) the syntax of regular expressions RE_Σ over Σ can be defined by:

$$\begin{aligned} \mathrm{RE}_\Sigma \ni R &::= a \mid \emptyset \mid \epsilon \mid R \cup R \mid R \circ R \mid R^* \\ \Sigma \ni a &::= \ldots \end{aligned}$$

(The second line is for enumerating the letters of the alphabet Σ.)

Language of a Regular Expression

Let R be a regular expression. The language $L(R)$ of R is defined recursively on the structure of R.

- Case R is a for some $a \in \Sigma$: $L(a) = \{a\}$
- Case R is $\emptyset$: $L(\emptyset) = \emptyset$
- Case R is ϵ: $L(\epsilon) = \{\epsilon\}$
- Case R is $R_1 \cup R_2$: $L(R_1 \cup R_2) = L(R_1) \cup L(R_2)$
- Case R is $R_1 \circ R_2$: $L(R_1 \circ R_2) = L(R_1)L(R_2)$
- Case R is R_1^*: $L(R_1^*) = (L(R_1))^*$

NB

L relates each syntactic object to a semantic object, whence we also call it a semantics for reg. exps.

$$L : RE_\Sigma \longrightarrow 2^{\Sigma^*}$$

Regular Expression Questions

For the first 6 points, let $\Sigma = \{0, 1\}$.

1. How do we write $\Sigma =$ "any one symbol" using existing operators?
2. What is 0^*1^*?
3. How do we wite "all strings ending in 11"?
4. What is $(0^*1^*)^*$?
5. how do we write "at least one 0"?
6. And how do we write "at least one 0 and at least one 1"?
7. How do we write $R^+ = R^1 \cup R^2 \cup \ldots$ using existing operators?
8. $R?$ means "an optional R". How do we write it?

Answers

1. Σ can be expressed as $0 \cup 1$; we use the abbreviation Σ below
2. any number of 0s followed by any number of 1s
3. $\Sigma^* 11$
4. the same as Σ^* i.e. all strings over the alphabet $\{0, 1\}$
5. $\Sigma^* 0 \Sigma^*$
6. $\Sigma^* (01 \cup 10) \Sigma^*$
7. RR^*
8. $\epsilon \cup R$

Some simple proof exercises

1. Suppose that R is a regular expression formed without use of $*$. Show that $L(R)$ is finite.
2. Suppose that R is a regular expression that contains a use of $*$ but does not contain a use of $\emptyset$ or ϵ. Show that $L(R)$ is infinite.