MAU22C00 Lecture 25

John Stalker

Trinity College Dublin

Quotients

Remember from last time: given a semigroup (A,f) and a relation R on A such that if $(u,x) \in R$ and $(v,y) \in R$ then $(f(u,v),f(x,y)) \in R$ there is a semigroup (B,g) and a homomorpism h from (A,f) to (B,g) such that h is surjective and $(x,y) \in R$ if and only if $h(x) = h(y)$.

This (B,g) is called the quotient of (A,f) by R.

If (A,f) is a monoid then so is (B,g). If (A,f) is a group then so is (B,g).

Example

Define R to be the set of ordered pairs (a, b) of natural numbers such that there is a natural number d such that $a = b + 17 \cdot d$ or $b = a + 17 \cdot d$.

This relation is reflexive and symmetric. Is it transitive?

Suppose $(a, b) \in R$ and $(b, c) \in R$, i.e that there's a d such that $a = b + 17 \cdot d$ or $b = a + 17 \cdot d$ and an e such that $b = c + 17 \cdot e$ or $c = b + 17 \cdot e$.

If $a = b + 17 \cdot d$ and $b = c + 17 \cdot e$ then $a = c + 17 \cdot f$, where $f = d + e$, so $(a, c) \in R$.

A similar argument works if $b = a + 17 \cdot d$ and $c = b + 17 \cdot e$, but the other two cases are more complicated.

Suppose $a = b + 17 \cdot d$ and $c = b + 17 \cdot e$.

Either $d \leq e$ or $e \leq d$. In other words, there is an f such that $d + f = e$ or $e + f = d$.

If $d + f = e$ then $c = b + 17 \cdot e = b + 17 \cdot d + 17 \cdot f = a + 17 \cdot f$.

If $e + f = d$ then $a = b + 17 \cdot d = b + 17 \cdot e + 17 \cdot f = c + 17 \cdot f$.

Example, continued

We've just seen that there is an f such that $a = c + 17 \cdot f$ or $c = a + 17 \cdot f$, so $(a, c) \in R$.

So R is reflexive, transitive and symmetric, i.e. is an equivalence relation.

If $(u, x) \in R$ and $(v, y) \in R$ then $(u + v, x + y) \in R$.

The proof is similar to the proof of transitivity above.

So we can quotient $(N, +)$ by the relation R to get a monoid.

That monoid is actually a group. Any $C \in B$ is the equivalence class of some natural number a. There is then a natural number b such that $a + b$ is a multiple of 17. The equivalence class of b is an inverse to C.

The group constructed in this way is called the cyclic group of order 17.

There's nothing special about 17. I could have used any non-zero natural number.

It's also possible to use $(N, \cdot)$ rather than $(N, +)$, but then you just get a monoid, not a group.

Integers

The standard construction of the integers from the natural numbers is via a quotient.

We start from the monoid N^2 with the operation $f((a,b),(c,d)) = (a+c, b+d)$ and identity $(0,0)$.

We define an equivalence relation by $((a,b),(c,d)) \in R$ if and only if $a+d = b+c$.

These satisfy the compatibility condition needed for the quotient construction.

The quotient is the group of integers, with addition as the operation.

It's a group because every element is invertible. The inverse of the equivalence class of (a,b) is the equivalence class of (b,a).

The interpretation is that the equivalence class of (a,b) is the integer $a-b$. Different representatives of the same equivalence class give the same integer.

We can do the same thing with the operation $f((a,b),(c,d)) = (a \cdot c, b \cdot d)$ to get the multiplicative structure on the integers.

Rationals, reals

The rationals are constructed from the integers by a similar construction. The monoid we start from is $Z \times P$ with the operation $f((a,b),(c,d)) = (a \cdot d + b \cdot c, c \cdot d)$ and the relation $((a,b),(c,d)) \in R$ if and only if $a \cdot d = b \cdot c$.

This time (a,b) is to be interpreted as the rational number a/b.

There are a number of constructions of the real numbers from the rational numbers. The usual one is again a quotient construction.

Regular languages

Remember that a language is just a set of lists of tokens, chosen from some set of tokens.

In applications tokens may be characters, strings, etc. For simplicity I'll consider only characters as tokens in examples. Lists of characters are then strings, so our example languages are sets of strings. That's a self-imposed limitation on the examples, not a limitation on the theory.

Also, I'll conflate terminal symbols and tokens.

The notes have different examples from lecture, but the same comment applies.

What is a regular language?

I'm not going to define regular languages yet, but they are describable in many different ways:

- in terms of the types of phrase structure grammars that can generate them
- in terms of the types of automata which can recognise them
- in terms of regular expressions, to be defined later
- in terms of they syntactic monoids, to be defined later

You need to understand all of these descriptions, because some problems are *much* easier when viewed from one point view than from another.

Left and right regular grammars

We've discussed phrase structure grammars, rules, alternates, terminal and non-terminal symbols, etc.

A left regular grammar is one where

- each alternate in the rule for the start symbol is a single non-terminal symbol,
- the start symbol never appears on the right hand side of a rule, and
- each alternate in the rule for any other non-terminal symbol is either empty or is a single non-terminal symbol followed by a single terminal symbol.

A right regular grammar is the same, except the last condition becomes

- each alternate in the rule for any other non-terminal symbol is either empty or is a single terminal symbol followed by a single non-terminal symbol.

Other people use slightly different definitions, depending on what exactly they want to do. Usually it's easy to convert between different conventions.

Example

```
%%
tokyo : atamadaka | nakadaka | odaka | heiban ;

atamadaka : H t0 | L error ;
nakadaka : H error | L t1 ;
odaka : H error | L t2 ;
heiban : H error | L t2 ;
t0 : H error | L t3 ;
t1 : H t4 | L error ;
t2 : H t5 | L error ;
t3 : | H error | L t3 ;
t4 : H t4 | L t3 ;
t5 : | H t5 | L error ;
error : H error : L error ;
```

This is a right regular grammar, since the non-terminal symbols are to the right of the terminal symbols in the alternates.

`t3` and `t5` can generate the empty string. `error` can't generate any string!

Another example

```
%%
tokyo : atamadaka | nakadaka | odaka | heiban ;

atamadaka : error H | t6 L ;
nakadaka : error H | t7 L ;
odaka : t8 H | error L ;
heiban : t8 H | error L ;
t6 : empty H | t6 L ;
t7 : t8 H | t7 L ;
t8 : t8 H | empty L ;
empty : ;
error : H error : L error ;
```

This is a left regular grammar. The `empty` symbol can generate only the empty string and is the only symbol which can generate the empty string.

These two grammars generate the same language. That's not obvious.

Yet another example

```
%%
kansai : heishinshiki | teikishiki ;

heishinshiki : H k0 | L error ;
teikishiki : H error | L k1 ;
k0 : H k2 | L k3 ;
k1 : H k3 | L k1 ;
k2 : | H k2 | L k3 ;
k3 : | L k3 ;
error : H error | L error ;
```

This is a right regular grammar. It generates a different language.

The `error` symbols in these grammars are somewhat optional, you could remove the rule for them, and any alternates where they appear in rules for other symbols.