MAU22C00 Lecture 27

John Stalker

Trinity College Dublin

Another example

Let's construct an FSA to recognise the language generated by the grammar

```
%%
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 : | H error | L k3 ;
error : H error : L error ;
```

There will be 7 states, `heishinshiki`, `teikishiki`, `k0`, `k1`, `k2`, `k3` and `error`.

`heishinshiki` and `teikishiki` will be start states. `k2` and `k3` will be accepting states.

Each state has two arrows coming out, one labelled `H` and one labelled `L`, although we can combine these for the `error` state.

Another example

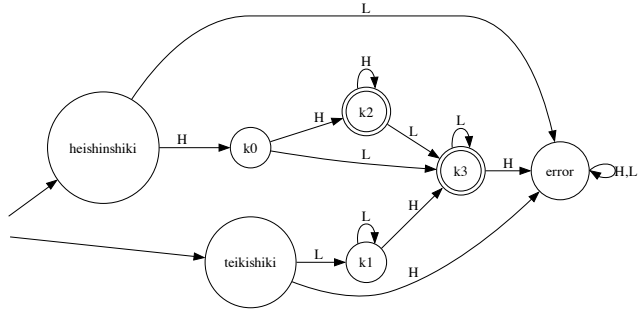

Figure 1: A finite state automaton for the kansai language

Does this finite state automaton accept the string LHLL?

What about LLHL?

Questions about FSAs

- What if someone gave us a left regular grammar rather than a right regular grammar?
- These finite state automata may be non-deterministic. Can we get deterministic ones?
- Can we implement set operations on languages, i.e. union, intersection, relative complement, at the level of finite state automata?
- Can we implement reversal?
- What about concatenation and Kleene star?

I'll answer some of these and defer others until we have more points of view.

Eliminating non-determinism

There are two sources of non-determinism: multiple initial states and multiple possible transitions on reading a single token in a single state.

We say a finite state automaton is deterministic if there is at most one start state and at most one allowed transition from any given state on any given input token.

We say it's strongly deterministic if there's also at least one start state and at least one allowed transition from any given state on any given input token.

There is at most one state a deterministic finite state automata could be in after reading any particular list of tokens.

It might not be in any state though. The computation may have failed before getting to this point. That can't happen for strongly deterministic FSAs though.

Given a non-deterministic finite state automaton can we find a deterministic one which recognises the same language? Can we find a strongly deterministic one?

The Tokyo FSA revisited

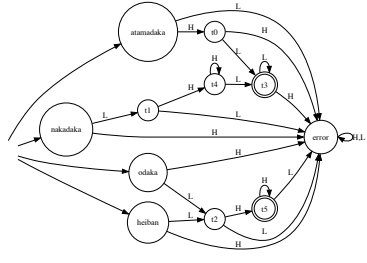

Figure 2: Yes, we're still looking at this graph

After reading 0 tokens there is a set of four states we could be in.

After reading 1 token, if it's an `H` we could be in `t0` or `error`. If it's an `L` we could be in `error`, `t1` or `t2`.

What about two tokens in? After `HH` or `LL` we could only be in `error`. After `HL` we could be in `t3` or `error`. After `LH` we could be in `error`, `t4` or `t5`.

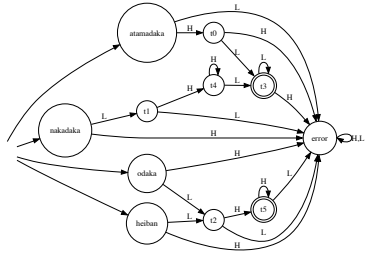

Figure 3: Yes, we're still looking at this graph

After `HH` or `LL` we could only be in `error`. After `HL` we could be in `t3` or `error`. After `LH` we could be in `error`, `t4` or `t5`.

After `HHH`, `HHL`, `LLH` or `LLL` we must be in error. Also after `HLH`.

After `HLL` we could be in `t3` or `error`. After `LHH` we could be in `error`, `t4` or `t5`. After `LHL` we could be in `error` or `t3`.

These sets of states are all ones we've seen before, so no matter how many more tokens we read we won't see any new ones.

The Tokyo FSA revisited.

The sets of states possible states are different depending on what we've read, but they are limited to

- A0: `atamadaka`, `nakadaka`, `odaka`, `heiban`
- A1: `t0`, `error`
- A2: `error`, `t1`, `t2`
- A3: `error`
- A4: `t3`, `error`
- A5: `error`, `t4`, `t5`

Initially we're in one of the members of A0. If we're a member of A4 or A5 at the end the we could be in the state `t3` or `t5` and so the input should be accepted.

For each of the six sets of states above there's one we transition to on reading an `H` and another one we transition to on reading an `L`.

From this information we can construct a strongly deterministic finite state automaton which recognises the same language.

A strongly deterministic finite state automaton

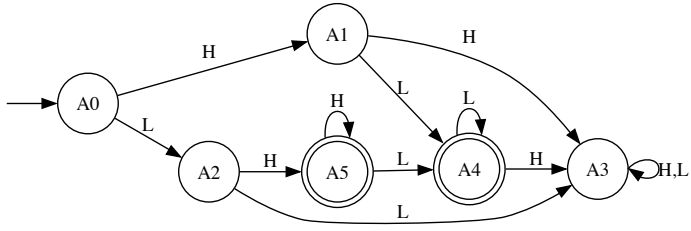

Figure 4: Finally, a simple graph!

The key idea is to construct a new finite state automaton whose states represent sets of states in the original one. There are two variants of this procedure.

- Identify those sets which are actually reachable with some input. In the example there are 6 of these. This is what you want to do in practical problems.
- Just use all sets of states. In the example there are 2048 of these. Some won't be reachable, but who cares? This is what we usually do in theoretical problems.

Another strongly deterministic finite state automaton

Of course we can do the same thing for the Kansai language

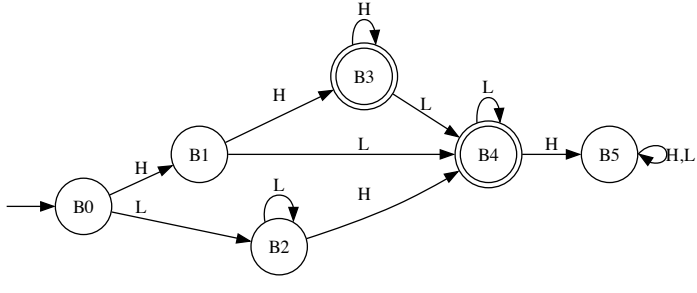

Figure 5: Another simple graph

FSAs for tokyo and kansai languages

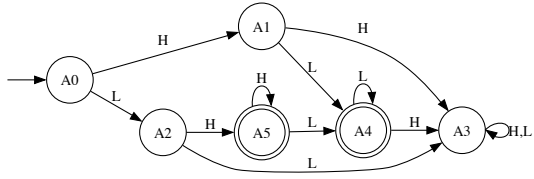

Figure 6: FSA for tokyo

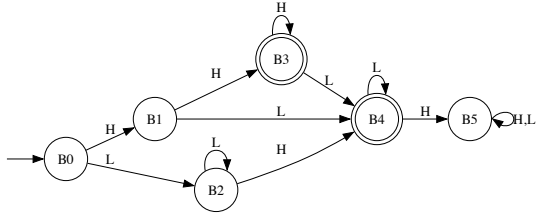

Figure 7: FSA for Kansai

Can we build an FSA which accepts only those strings accepted by both of these?

Intersection

"Can we build an FSA which accepts only those strings accepted by both of these?" is the same question as "can we construct a recogniser for the *intersection* of the two languages?"

How would you manually check that a string is recognised by both?

- For each FSA check go through the input token by token and check where you end up.
- For each input token look at each FSA and see what state it moves to.

The second of these is something we can turn into a finite state automaton. Its states will correspond to pairs of states for the two original automata.

An FSA for the intersection

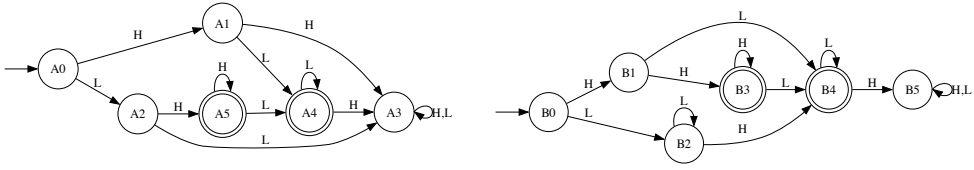

Figure 8: Both FSAs

We start in `A0B0`. If the first character is `H` we move to `A1B1`. If it's `L` we move to `A2B2`.

If the first two characters are `HH` we move to `A3B3`. If they're `HL` we move to `A4B4`. If they're `LH` we move to `A5B4`. If they're `LL` we move to `A3B2`.

The possibilities after three characters are `A3B3`, `A3B4`, `A3B5`, `A4B4`, `A5B5`, `A4B4`, `A3B4` and `A3B2`. Only six of those are distinct though.

The only *new* possibility after four characters is `A4B5`. After that there's nothing new.

An FSA for the intersection, continued

Out of the 36 state combinations we might have seen only 11 are reachable. We can form a finite state automaton with those combinations as its states.

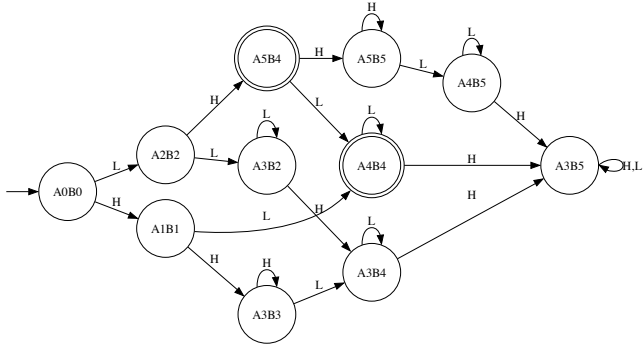

Figure 9: An FSA for the intersection

The accepting states are the ones where both automata are accepting.