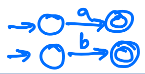
Two Page - Two pager for the final exam.

COMP2022 Models of Computation (University of Sydney)

Regex to NFA-ε: construct from units with closure

$a^* | b$, 1. NFA for a, b | 2. NFA for a^* | 3. NFA for $a^*|b$



NFA-ε to NFA: EC(q,r)

$EC(q_0) = \{q_0, q_1, q_2, q_4\}$, $EC(q_1) = \{q_1, q_2\}$, $EC(q_2) = \{q_2\}$ $E(q_5) = \{q_5\}$
$EC(q_3) = \{q_3, q_2\}$ $EC(q_4) = \{q_4\}$

1. any sets containing final states => make parent final state
2. reconstruct the NFA without ε-transitions by inspection 3. remove unreachable

NFA to DFA: use state | a | b table

1. in state column, put reachable states (or pairs), don't bother with states with no outgoing i.e comma seperate for multiple transitions of one character, and add that as new state
2. rename states and construct! => causes 2^n states

DFA to Regex

1. add new initial state and final state (s, F) with ε-transitions
2. rip transitions with In / Out method

Proving irregularity

1. assume L is regular
2. then there is a DFA M recognising it
3. By PP, two strings x_i, x_j go to same state
4. propose some other string that, when concatenated, distinguishes x_i, x_j ($x_i \in L, x_j \notin L$)
5. since M accepts one but not other → contradiction → L is not regular

Parse trees: terminals at leaves otherwise non terminal

CFG to CNF: STBEU

S: remove S from RHS of all rules (either add new variable or sometimes fine)
T: replace terminals on RHS with new rule $N_1 \to a$ (d/w when already in that form)
B: chop to two rules by adding new rules
E: eliminate ε by adding new rules (just add |'s wherever ε could be)
U: sub in terminals for solo variables or doubles if it flows to doubles

CYK Algorithm

1. height / width = len(string)
2. move bottom upwards adding in cell where you can obtain substring
3. split strings into substrings, see how to derive, x them, and see if x products are in CNF
4. G must be in CNF and if s in top cell, you can derive the string from G

Turing Machines

=> a language is recognisable if some TM recognises it (same as decidable)
=> TM fails for two reasons: rejecting or diverging

regular ⊂ context-free ⊂ decidable (computably enum) ⊂ recognisable / recursively enumerable

TM-variations

1. BTM → Must Move
   => replace S-transition by R, L
2. BTM → LBTM
   => split LB tape into 2 tracks (upper, lower)
   => upper represents right half of tape (right of initial head pos)
   => lower is left
   => extra state keeps track of which side head in
3. BTM → Multi-tape TM
   => split tape into 2k-many tracks of single tape
   => for each new track, use one track to store tape contents and one to mark head position on that tape

Closure Properties of TMs

=> simulate programmatically M with input x
=> complement

```
def my_fun(x):
    return not M(x)
```

∴ if M decidable, $\overline{M}$ is
if M recognisable, $\overline{M}$ is

=> union

```
def my_fun(x)
    if M1(x):
        return 1
    if M2(x):
        return 1
    return 0
```

∴ if M_1, M_2 decidable, $M_1 \cup M_2$ is
if $M_1 | M_2$ recognisable, $M_1 \cup M_2$ can't tell

=> intersection of decidable language is decidable
=> a language is decidable when it and it's complement are recognisable

L_{DIAG}

=> $L_{DIAG} = \{ \text{source}_M \mid M \text{ does not accept source}_M \}$
=> is not recognisable
=> showing a language is not recognisable
1. show that there is some TM B such that $L = L(B)$
2. show that for every TM B there is some string x such that either:
   i. $x \in L(B)$ and $x \notin L$
   ii. $x \notin L(B)$ and $x \in L$

1. acceptance problems for DFA, NFA, RE & CFG are decidable
2. emptiness problems for DFA, NFA, RE & CFG are decidable
3. equivalence problems DFA, NFA, RE

UTM is not a decider
UTM recognises acceptance problem for TMs (L_{TM} ... is not decidable)

Propositional Logic

=> α is an assignment to F

=> $\alpha \models F$ means α satisfies F

=> conditional $(F \to G)$

F	G	F → G
1	1	1
0	1	1
1	0	0
0	0	1

2^n assignments

=> bi-conditional $(F \leftrightarrow G)$

F	G	F ↔ G
1	1	T
0	1	F
1	0	F
0	0	T

=> true when $F = G$

=> validity = satisfiable $\forall \alpha$

(Idempotent Laws)	$F \equiv (F \wedge F)$
	$F \equiv (F \vee F)$
(Commutative Laws)	$(F \wedge G) \equiv (G \wedge F)$
	$(F \vee G) \equiv (G \vee F)$
(Associative Laws)	$(F \wedge (G \wedge H)) \equiv ((F \wedge G) \wedge H)$
	$(F \vee (G \vee H)) \equiv ((F \vee G) \vee H)$
(Absorption Laws)	$(F \wedge (F \vee G)) \equiv F$
	$(F \vee (F \wedge G)) \equiv F$
(Distributive Laws)	$(F \wedge (G \vee H)) \equiv ((F \wedge G) \vee (F \wedge H))$
	$(F \vee (G \wedge H)) \equiv ((F \vee G) \wedge (F \vee H))$
(de Morgan's Laws)	$\neg(F \wedge G) \equiv (\neg F \vee \neg G)$
	$\neg(F \vee G) \equiv (\neg F \wedge \neg G)$
(Double Negation Law)	$\neg\neg F \equiv F$
(Validity Law)	$(F \vee \top) \equiv \top$
	$(F \wedge \top) \equiv F$
(Unsatisfiability Law)	$(F \vee \bot) \equiv F$
	$(F \wedge \bot) \equiv \bot$
(Constant Laws)	$\top \equiv (F \vee \neg F)$
	$\bot \equiv (F \wedge \neg F)$
(Negating constants Laws)	$\neg\top \equiv \bot$
	$\neg\bot \equiv \top$
(Conditional Law)	$(F \to G) \equiv (\neg F \vee G)$
(Bi-conditional Law)	$(F \leftrightarrow G) \equiv ((F \to G) \wedge (G \to F))$

Normal Forms

=> NNF : negations only occur immediately in front of atoms

e.g $p, \neg p$ but not $\neg\neg p$

not in front of atom

=> algorithm for NNF: just apply negation and de Morgans' law

=> CNF: a clause or conjunction of clauses

e.g $p \wedge q$, $p \vee q$, $(p \vee \neg q) \wedge (q \vee r) \wedge r$

=> algorithm for CNF: put F in NNF, say F', substitute in F' each occurrence of a subformula of the commutative or distributive law form

Predicate Logic

=> models objects, properties of objects and relations between objects

=> domain = set of objects

=> predicates = functions that output a boolean value based on object/variable input

=> quantifiers = $\forall$, E

=> common:

There are some common forms:

1. "All As are Bs" translates as $\forall x(A(x) \to B(x))$
2. "Some As are Bs" translates as $\exists x(A(x) \wedge B(x))$
3. "No As are Bs" translates as $\forall x(A(x) \to \neg B(x))$
4. "Some As are not Bs" translates as $\exists x(A(x) \wedge \neg B(x))$

Usually:

$\wedge$ goes with $\exists$

$\to$ goes with $\forall$

Translate the statement "Every even integer is greater than some odd integer" into predicate logic.

- This is of the form "All As are Bs"
- $A(x)$ for "x is an even integer"
- $B(x)$ for "x is greater than some odd integer"

$$\forall x(\mathtt{even}(x) \to \exists y(\mathtt{odd}(y) \wedge \mathtt{greater}(x, y)))$$

(Q. Negation)	$\neg\forall x F \equiv \exists x \neg F$
	$\neg\exists x F \equiv \forall x \neg F$
(Q. Unification)	$(\forall x F \wedge \forall x G) \equiv \forall x(F \wedge G)$
	$(\exists x F \vee \exists x G) \equiv \exists x(F \vee G)$
(Q. Transposition)	$\forall x \forall y F \equiv \forall y \forall x F$
	$\exists x \exists y F \equiv \exists y \exists x F$
(Q. Extraction)	if $x \notin \mathrm{Free}(G)$:
	$(\forall x F \wedge G) \equiv \forall x(F \wedge G)$
	$(\forall x F \vee G) \equiv \forall x(F \vee G)$
	$(\exists x F \wedge G) \equiv \exists x(F \wedge G)$
	$(\exists x F \vee G) \equiv \exists x(F \vee G)$

=> negation: swap quantifiers and negate all predicates

=> a variable is bound if it occurs in F and subformulas of F (else free)