KLEENE ALGEBRA WITH TESTS

6/3/2024

by Alexandra Silva

= applications to program equivalence and verification =

```
while a & b do            ?      while a do
    p;                    =          if b then
while a do                                p
    p;                                 else
    while a & b do                         q
        p;
```

two programs that both have tests in red and programs ~~/~~expressions like assignment in blue

are they doing the same thing? are they equal?

we will cover a way to decide this! for

uninterpreted simple imperative programs

regular expressions were introduced to reason about control flow

we will take an algebraic approach w/ regular expressions

Regular expressions (Syntax)

for talking about Regular languages (Semantics) denotational

$\Updownarrow$ Correspondence by Kleene

Deterministic Finite Automata (Semantics Operational)

$$RE \ni e ::= \underbrace{0 \mid 1}_{\text{basis elements/ constants}} \mid a \in \underbrace{A}_{\text{finite}} \mid \underbrace{e;e}_{\text{composition}} \mid e+e \mid e^*$$

set of words

$[\![e]\!] : 2^{A^*} = \mathcal{P}(A^*)$

Denotational Semantics

$[\![0]\!] = \emptyset$ no behavior

$[\![1]\!] = \{\varepsilon\}$ empty word

$\underbrace{[\![a]\!]}_{\text{syntax}} = \{\underbrace{a}_{\text{word in } A^* \text{ (semantics)}}\}$

$[\![e+f]\!] = [\![e]\!] \cup [\![f]\!]$

$[\![e;f]\!] = [\![e]\!] \bullet [\![f]\!]$ elements of e is followed by elements of f

$[\![e^*]\!] = \bigcup_{n \in \mathbb{N}} [\![e]\!]^n$ iterate e as many times as one can

$\mathcal{U}, \mathcal{V} : 2^{A^*}$ concatenating (sets of) words

$\underbrace{\mathcal{U} \bullet \mathcal{V}}_{\text{semantics of concatenation}} = \{ \underbrace{uv}_{\text{word concatenation}} \mid u \in \mathcal{U}, v \in \mathcal{V} \}$

$\mathcal{V}^0 = \{\varepsilon\}$ Semantics of star

$\mathcal{V}^{n+1} = \mathcal{V}^n \bullet \mathcal{V}$

Examples of Regular Expressions

$(1+a);(1+b) \mapsto \{\varepsilon, a, b, ab\}$

$(a+b)^* \mapsto \{a,b\}^*$

$(a^* b)^* a^* \mapsto \{a,b\}^*$ ö!

$(a+b)^*$

$|||$ → equal in the sense that their denotations are the same

$(a^* b)^* a^*$ called denesting! related to compiler optimizations

can apply this to go from one program to another

$0+e \equiv e$ would like to reason at the program level, rather than denotational level

syntax

is there a finite number of equations that allow program refinement (i.e. reason about syntax level)?

Kleene's Theorem

Let L be a (regular) language, then the following are equivalent

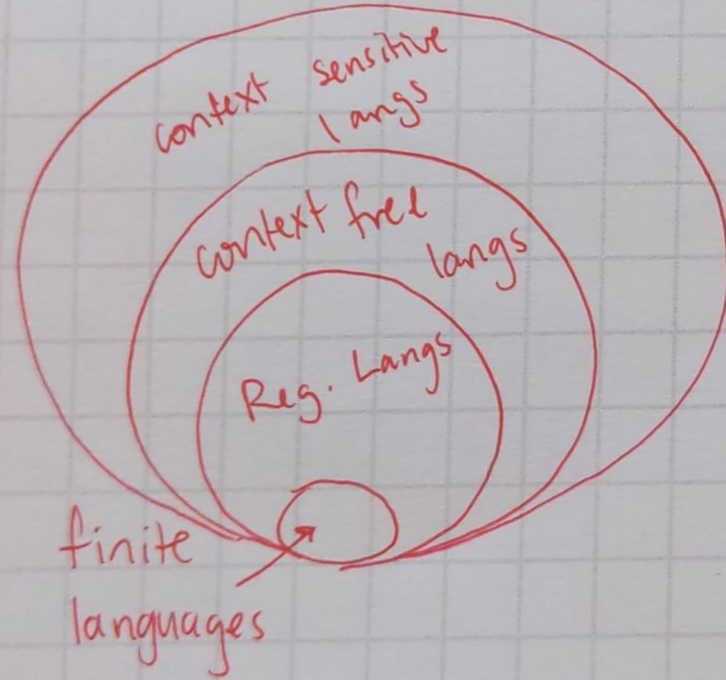
implied by

① $L = [\![e]\!]$ for some regular exp. e

② L is accepted by a DFA

eterministic inite utomata

Chomsky Hierarchy



Reg. Langs can be defined w/o respect to reg. exps.

can be defined useing regular sets

Syntax was set up to emulate/give rise to regular sets

exists research between the image of the syntax + regular sets

↳ (rational)

① → ② by Brzozowski

Syntactic way of building the automaton from expression via <u>derivatives</u> (small step semantics)

<u>DFA</u>

$F : S \to 2$ S - finite set of states

↳ is a state <u>final</u>? $2 =$ "bool"

$t : S \to \underline{\underline{S^A}}$

↳ transitions   deterministic, <u>one</u> state

need corresponding functions for reg. exps!

$E : RE \to 2$ → is this a final state, i.e does it accept <u>empty word</u>!

$D_a : RE \to RE^A$ → next state, i.e. takes one letter off!

$$E(0) = 0 \quad (\text{false}) = E(a) = 0$$
$$E(1) = 1 \quad (\text{true}) = E(e^*) = 1$$
$$E(e+f) = E(e) \vee E(f)$$
$$E(e;f) = E(e) \wedge E(f)$$
$$E(e^*) = 1$$

Given language $L : 2^{A^*}$

derivative by a is → $L_a = \{ u \mid au \in L \}$

$D_a : RE \to RE$

$$\frac{X \to X^a}{(X \to X)_{a \in A}}$$ can do bc A finite

$D_a(0) = 0$

$D_a(1) = 0$

$D_a(b) = \begin{cases} a = b & \text{then } 1 \\ a \neq b & \text{then } 0 \end{cases}$ if

$D_a(e;f) = D_a(e);f$ (removed a from e)

$+$

$E(e); D_a(f)$

if $0; - = 0$

key idea!

$2 \leftrightarrow RE$

embeds to

can write as case split $\begin{cases} D_a(e);f \ \boxed{\text{if}} \ E(e)=0 \\ D_a(e);f + D_a(f) \ \boxed{\text{if}} \ E(f)=1 \end{cases}$

corresponds to if-then-else (see tomorrow)

$D_a(e+f) = D_a(e) + D_a(f)$

$D_a(e*) = D(e); e*$

not quite a DFA though! need start states! and finiteness!

can pick e as our start state, then need to argue that the set we reach from e is finite

is this true? almost! let's see $\longrightarrow$

$e = (a^*)^*$ $\qquad \mathcal{D}_a(e) = \underbrace{(1;a^*);(a^*)^*}$

$$\mathcal{D}_a(\mathcal{D}_a(e)) = (0a^* + 1a^*)\,a^{**} + \underbrace{(1a^*)(a^*)^*}$$

this expression keeps showing up for each derivative!

$$\mathcal{D}_a(\mathcal{D}_a(\mathcal{D}_a(e))) = \ldots\ldots + (1a^*)(a^*)^* + (1a^*)(a^*)^*$$

↳ but ... this is equivalent to previous derivative!

to get the finiteness, need ★ equivalence classes of derivatives

★ need to delete repeated expressions in a $+$

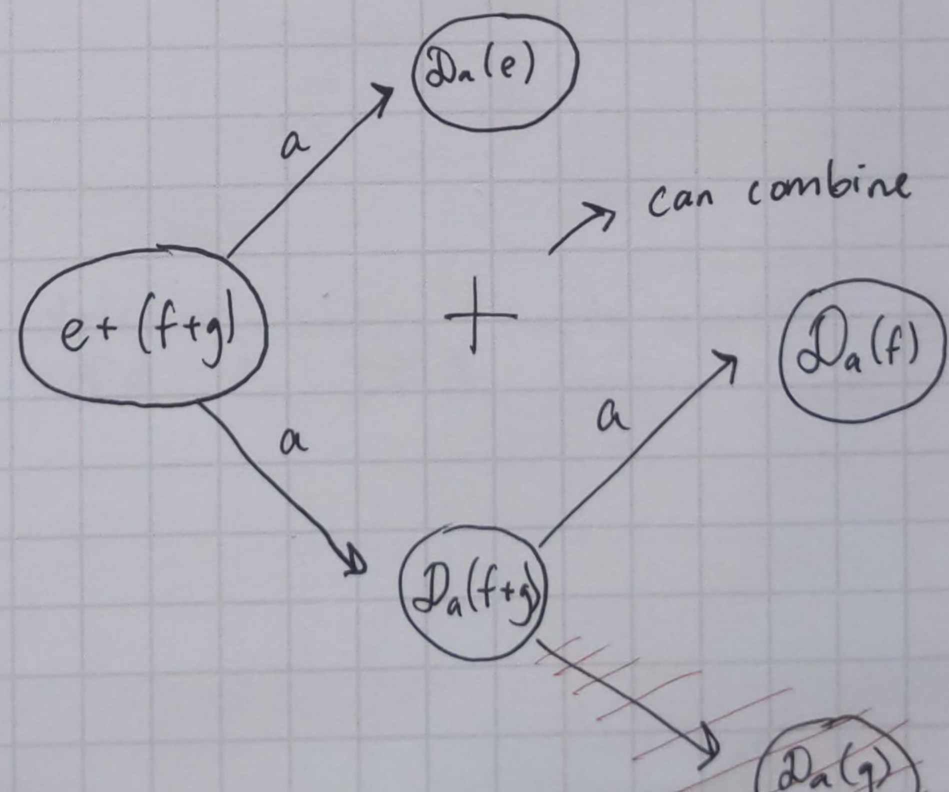
modulo ACI - associativity, commutativity, idempotence

$$\mathcal{D}_a : RE \longrightarrow P(RE)$$

can write $\mathcal{D}_a(e+f) = \{\mathcal{D}_a(e), \mathcal{D}_a(f)\}$

get ACI for free! shown by Antimorov

no need to reason syntactically i.e. modulo ACI

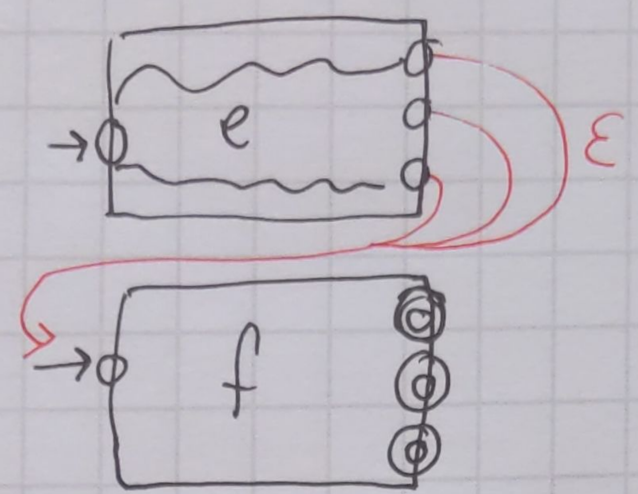


assuming $\mathcal{D}_a(e) = \mathcal{D}_a(g)$

but still NFA! but can combine into one transition

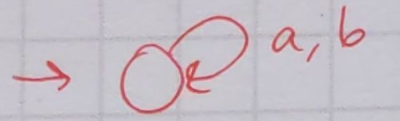
Thompson construction

e ; f

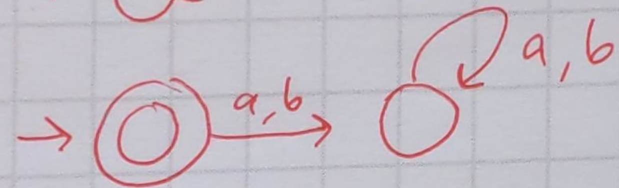


concatenate two automaton !

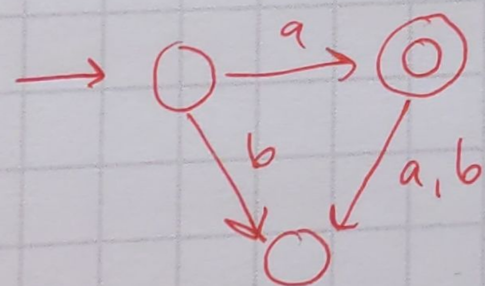
0



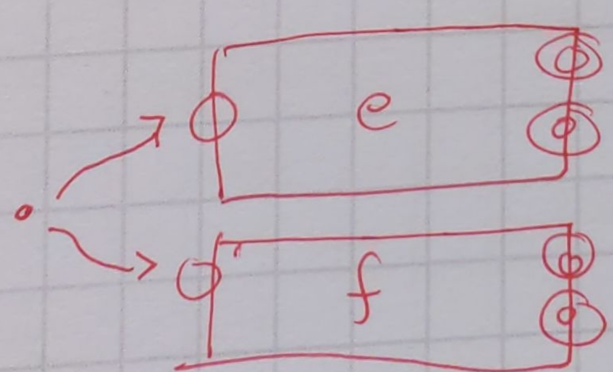
1



a



e + f



can use determinization / empty transition elimination
to turn NFA to DFA