→ SYNTAX

??

Variables : $Var = Var_{int} \cup Var_{str}$ (two-sorted)

$(m, n..)$ $(x, y, s..)$

Constant : $Con = Con_{int} \cup Con_{str}$ ← String constants

$Con_{str} \subset \Sigma^*$

$\Sigma = \{f, g, h...\}$ ← alphabets

Terms :

String term (t_{str})	Length term (t_{len})
→ element of Var_{str}	→ element of var_{int}
→ element of Con_{str}	→ element of con_{int}
→ Concatenation of those elements.	→ $len(t_{str})$

Formula : Atomic (F) | F ∧ F | F ∨ F | ¬ F | ∃ x F(x) | ∀ x F(x)

$A_{wordeqn}$ A_{len} A_{regex}

$t_{str} = t_{str}$

$abx = xab$

constants

Var_{str}

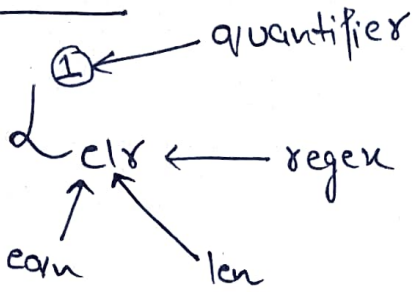
$t_{len} \leq c$

$len(x) \leq 3$

$t_{str} \in RE$

$x \in ab^*c$

→ NOMENCLATURE



$\Sigma = \{a, b, c\}$ ← alphabets

no quantifiers ① $\mathcal{L}_{elr}$: formulas with eqns, length terms & regex terms w/ no quantifier

Σ $\mathcal{L}^{0}_{er}$: only eqns & length terms w/ no quantifier.

→ SEMANTICS

We fix $\Sigma = \{a_1, a_2, a_3, \ldots a_n\}$

Given a $\mathcal{L}^{1}_{elr}$ formula θ on Σ;
we can map free variables of θ.

$$FV(\theta) \longrightarrow \Sigma^* \cup \mathbb{N}$$

If such an assignment makes θ true, its a satisfying assignment.

Representation of solutions

Ex. 1 : $\mathcal{L}_e^0$

θ : $X = aYbZa$ $\quad\quad$ $\Sigma = \{a, b\}$

$FV(\theta) = \{X, Y, Z\}$

$X \to aYbZa$

$Y \to Y$ ← unified part

$Z \to Z$ ←

Any choice of $Y, Z \in \Sigma^*$ gives a solution.

Ex 2: $\mathcal{L}_e^0$

$abX = Xba$

$X = aba$

$X = ababa$

$X = abababa$

$X = (ab)^i a$. ← assignment of integer parameter.

Ex 3: $\mathcal{L}_{elr}^0$

$abX = Xba \wedge X \in (ab \mid ba)(ab)^* a \wedge len(X) \leq 5.$

$X = aba$

$X = ababa$

DECIDABLITY THEOREM

T.P: $\mathcal{L}^{0}_{eq}$ is decidable.

→ Word eqns by themselves are already decidable [Plandowski]

→ System of inequalities over integer variables is decidable. (Presburger arithmetic)

→ $abX = Xba \wedge X = abY \wedge len(X) < 2$

word eqn — length — explicit length constraints.

~~$len(X) = 2c$~~

$X = (ab)^i a$

$Y = (ab)^{i-1} a$

UNSAT because they aren't simultaneously SAT.

$\{ len(X) = 2c+1 \;,\; len(Y) = 2c-1 \}$

↑ implicit length constraints

→ $X = abY$

$\{ len(X) = 2 + len(Y) \;;\; len(Y) \geq 0 \}$

implicit length constraints.

→ The crux of this proof is representing the finite representation of these implicit length forms.

→ Solved form

- S is a solved form of ω if every variable in ω can be defined as

$X = t$ ← concat of constants & unfixed parts

↙ Variables

- Also variable can occur only once on LHS & never on RHS.

Ex. $Xa = aY \wedge Ya = Xa$

S: $x = a^i \quad i \geq 0$

$Y = a^i$ ← constant raised by integer parameter.

Ex. $abX = Xba \wedge X = abY$

$X = (ab)^i a$

$Y = (ab)^{i-1} a$

→ The crux of this proof is representing the finite representation of these implicit length forms.

→ Solved form

- S is a solved form of ω if every variable in ω can be defined as

$X = t$ ← concat of constants & unfixed parts

Variables

- Also variable can occur only once on LHS & never on RHS.

Ex. $Xa = aY \wedge Ya = Xa$

S: $x = a^i \quad i \geq 0$

$Y = a^i$ ← constant raised by integer parameter.

Ex. $abX = Xba \wedge X = abY$

$X = (ab)^i a$

$Y = (ab)^{i-1} a$

ex. $XabY = YbaX$

$X = bb$

$Y = b$

$X = (b)^{i+1}$

$Y = b$

$Y = aa$

$X = a$

$X = (a)^i$

$Y = (a)^{i+1}$.

Not a solved form since its not in form of $\begin{Bmatrix} X = t \\ Y = t \end{Bmatrix}$. X & Y have to be represented twice to encapsulate all solutions.

Lemma 1: If ω has a solved form S then $\exists R$ of linear length constraints implied by ω that is finite & complete. Moreover there is a algorithm that computes R.

Proof: we will imply the length constraints from solved form S.

every eqn in S looks like this:

$$X = t_1 t_2 t_3 t_4 \cdots t_n$$

↓ variable in ω. $t_k \rightarrow$ either constant / constant raised by integer or unfixed part.

$$\text{len}(X) = C + i_1 t_4 + i_2 t_6 + \text{len}(t_7 + t_8 + t_{10})$$

- C: all constants
- $i_1 t_4$, $i_2 t_6$: constant raised by integer parameter
- $\text{len}(t_7 + t_8 + t_{10})$: length of unfixed parts.

~~$R = \text{len}$~~ ✱ the values on integer parameters can be chosen arbitrarily. $i_n \geq 0$

✱ length of unfixed parts. $\text{len}(t_7) \geq 0$

$$R = \underbrace{\text{len}(X) \wedge \text{len}(X_1) \wedge \text{len}(X_2) \cdots \text{len}(X_3)}_{\text{finite since finite variables.}}$$

Lemma 2: If ω has solved form S then ω is equisatisfiable with length constraints R.

→ If ω is SAT ; R is SAT: if there is a SAT assingment ~~then~~ for ω, that SAT assingment will be under the length constraints, hence R is SAT.

→ If R is SAT ; ω is SAT: if R is SAT ; there will exist atleast one assingment, that will also make ω SAT.

Theorem 11. SAT problem for $\mathcal{L}^{0}_{ex}$ is decidable given that there exists a algorithm to convert equations in solved forms.

Proof:

$$\mathcal{L}^{0}_{ex} = \phi \wedge \theta$$

ϕ → word eqn

θ → length constraint.

Step 1: Run ϕ through Plandowski's algorithm

Run θ through Presberger's arithmetic. If any of them return UNSAT the given problem is UNSAT.

Step 2: Otherwise compute R of ϕ ~~and~~ this can be done by finding S of ϕ and then finding the implied length constraints from that S like we saw on **lemma1**.

Step 3: Run $R \wedge \theta$ through Presberger.

if UNSAT then UNSAT.

if SAT, then SAT.

↳ we have already proved in lemma 2 that if there are some satisfying assingments for length constraint, atleast one of them satisfies w.

→ SATISFIABLITY OF WORD EQNS / LENGTH / RE:

Consider $\mathcal{L}^{0}_{eer}$ formula.

$$abX = Xba \quad \wedge \quad X \in (ab)^* b \quad \wedge \text{len}(X) \leq 3$$

Solved form S: $X = (ab)^i a$ or $a(ba)^i$ same thing

$$\wedge R \wedge \theta : \begin{cases} \text{len}(x) = 2i+1 \\ \text{len}(x) \leq 3 \end{cases}$$

for $i=1$; $\wedge R \wedge \theta$ is SAT.

However the formula isn't SAT since X ends on 'b' according to RE.

$X \in (ab)^* a$

& according to word eqn it should end in 'a'.

$$ab\underline{X} = Xb\underline{a}$$

So we have to put some restriction on solved form.

$S: X = (ab)^i a$ or $(ab)^* a$

Now we have two RE.

$X \in (ab)^* a$ $\quad$ $X \in (ab)^* b$

$\downarrow$ $\quad$ $\downarrow$

RE_1 $\quad$ RE_2

$L(RE_1) \cap L(RE_2) = \phi$ so UNSAT.

★ We should be able to write the solved form as Regex & then check for intersection with existing conditions. i.e. no unfixed parts and the solved forms have a regular language.

Lemma 3: If word eqn has a solved form w/o unfixed parts that is also a regular expression, then there is a finite set of linear length constraints that can be effectively computed from this solved form & which are equisatisfiable with the eqn.

What this means? if the solved form of a L^0_{elr} eqn does not have an unfixed part and is a regular expression, then we can compute the implied length constraints from the solved form & RE.

Proof: → we already know we can compute implied length constraints from solved form

→ For length constraints from RE, compute the regular set.

For example regular set of $a(aa)^*$ is

$\{a, aaa, aaaaa, aaaaaa, \dots\}$

For every regular set, the length grows as a arithmetic progression. i.e. $1_{+2}\ 3_{+2}\ 5_{+2}\ 7 \dots$ in this case.

∃ an algorithm which can extract the initial length, & arithmetic difference when given RE.

[Reference paper 1]

Theorem 13: SAT Problem for $\mathcal{L}^{0}_{elr}$ is decidable iff the solved form of word eqn doesn't have unfixed loops & is a regular expression.

$$\mathcal{L}^{0}_{elr} = \theta(X) \wedge \phi \wedge (X \in RE)$$

θ : w.e, ϕ : ℓ, $(X \in RE)$: RE.

Step 1: If $\theta(X)$ or $(X \in RE)$ or ϕ is UNSAT, return UNSAT

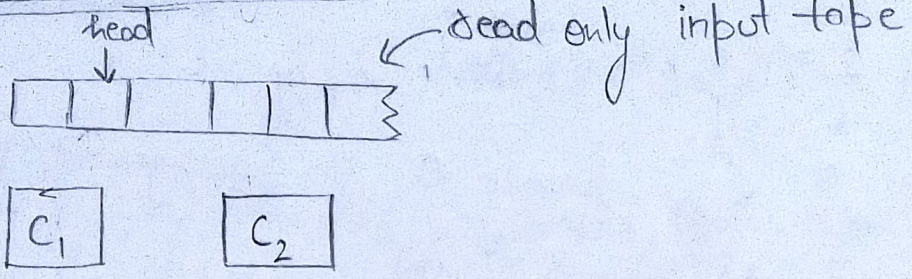
Step 2: Convert $\theta(X)$ in solved form where its a RE. then write it as $X \in RE_1$.

Step 3: Compute $X \in RE_1 \cap X \in RE$
if it is null, then return UNSAT.

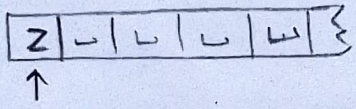
Step 4: Extract implied length constraints from $RE_1 \cap RE$. Lets call it Ψ.

Step 5: If $\Psi \wedge \phi$ is UNSAT (Presberger arithmetic) then else SAT.

Two-counter machines



- ☆ There is a single read only semi ∞ tape.
- ☆ There are two counters where we can increment, decrement, or stay as it is.
- ☆ There is no way to read the number on counters. we can only know if its zero or not.
- ☆ You can visualise the counters as storage tape w/ head.



when we have to increment w/ 1 ; the head moves right; decrement moves it to left.

This way when we see one counter; we can only read blank or Z. Blank means +ve ; Z means 0.

- ☆ A two counter machine can simulate a two stack PDA which can simulate a TURING MACHINE.

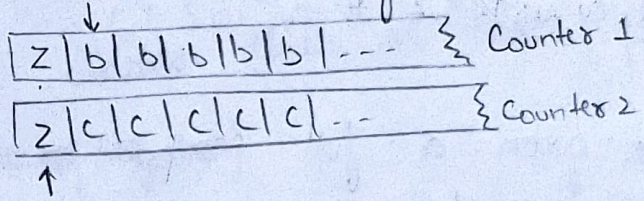
ID of Two Counter Machine

$$M = \langle Q, \Delta, \{z, b, c\}, \delta, q_0, F \rangle$$

Q : States $\{q_0, q_1 \cdots q_f\}$

Δ : Same as Σ i.e alphabets on input tape.

$\{z, b, c\}$: Alphabets on storage tape.

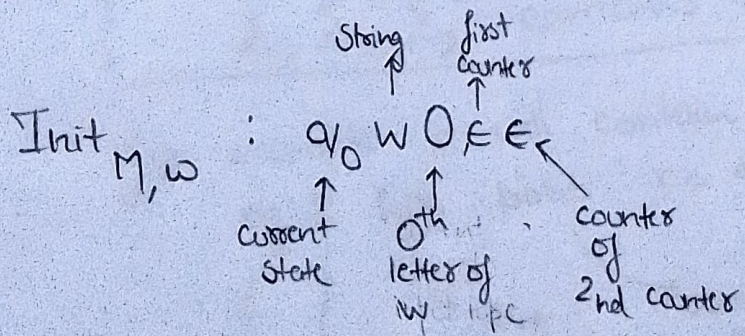


δ : Transition function.

$$Q \times \Delta \times \{z, b\} \times \{z, c\} \longrightarrow Q \times \{in, stor1, stor2\} \times \{L, R\}$$

Takes in current state; current alphabet on input tape & positions of counter (whether 0 or +ve)

Returns state; where input head moves; where counter heads move.



Final ID:

$F_{M,w} : q_f \; w \; 0 \in \in$

In the standard model the machine only halts when q_f is reached; input head is at start & both counters are 0.

Computation history of TCM

Computation history describes every stage & every transition the machine goes to before reaching final halting stage.

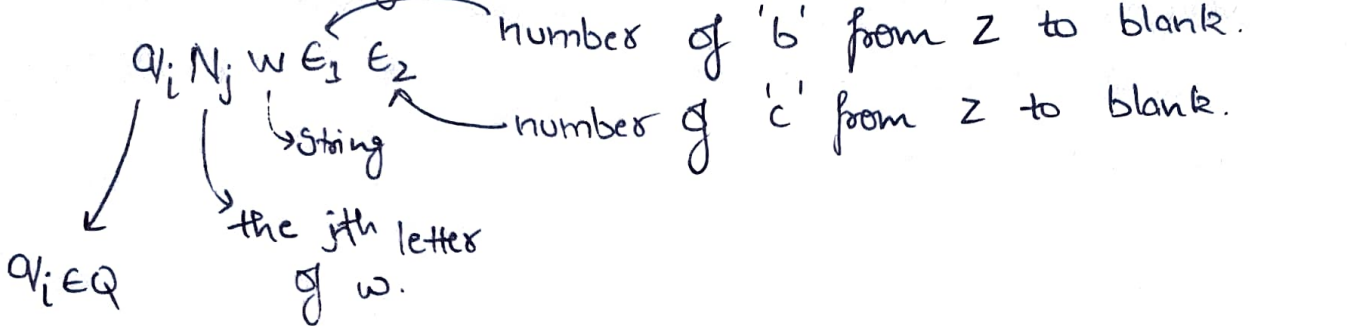
Example:

M_w: $q_0 w 0 \epsilon\epsilon \# q_1 w 1 1 \epsilon \# q_2 w 2 1 1 \ldots \# q_f w 0 \epsilon\epsilon$

So if we are given a full computation history, we can figure out if the machine accepts and halts or not.

UNIVERSE OF STRINGS

A random ID of two counter machine looks like this.



Now we divide this in three parts:

$$\Sigma_0 = \{\# q_i N_j w\}$$

$$\Sigma_1 = b.$$

$$\Sigma_2 = c$$

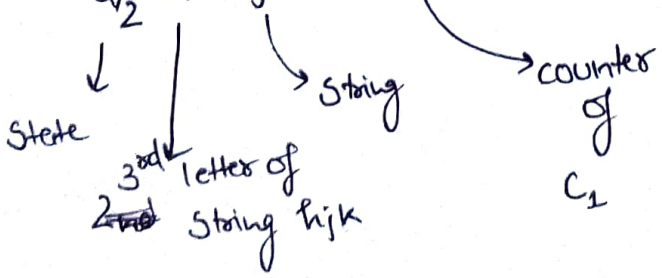
$$\Sigma = \{\Sigma_0 \cup \Sigma_1 \cup \Sigma_2\}$$

If we take all permutation, combinations of the elements in Σ i.e. Σ^* ; it will contain all valid IDs.

Or we can even say, the regular set $\Sigma_0 \Sigma_1^* \Sigma_2^*$ contains all IDs

for example it will contain

q_2 K hjk bbbb cc ← counter of c_2



UNDECIADABLITY THEOREM:

no free variables.

★ Validity problem for set of $\mathcal{L}_e^1$ (Sentences) over positive word equations with $\forall\exists$ quantifier is undecidable.

Example: $\forall X \exists Y \quad abcX = Y$ ✓ Valid.

$\therefore \exists Y (aY = a) \wedge (aYbc = abc)$ ✓ Valid $(Y = \epsilon)$

PROOF IDEA:

Reduction from halting problem for two counter.

- Take a two-counter machine M and input w.
- Construct a $\mathcal{L}_e^1$ Sentence $\forall S \exists S_1 S_2 S_3 S_4 \; \theta(S, S_1, S_2, S_3, S_4)$
- $M(w)$ doesn't halt only if sentence is valid.
- θ is constructed in such a way that this will only happen when $S \neq$ halting computation history.

PROOF:

Given $\langle M, w \rangle$

we construct a $\mathcal{L}_e^1$ sentence

Z: $\forall S \exists S_1 S_2 S_3 S_4 U V \; (\theta_{M,w}(S, S_1, S_2, S_3, S_4, U, V))$

S_1, S_2, S_3, S_4 are substrings of S.

★ Z is valid iff M doesn't halt & accept w.

How to construct such $\theta_{(M,w)}$:

- Number of ways an encoding history could be invalid is
  - → S doesn't start w/ Initial ID
  - → S doesn't end w/ final ID
  - → S isn't a well-formed history
  - → S doesn't follow transition function.