Context-Sensitive Data-Dependence Analysis via Linear Conjunctive Language Reachability

Qirun Zhang, Zhendong Su
POPL 2017

Presented by Ayaka Yorihiro
PLDG 11/18/2020

Static Program Analyses

Context Sensitivity:

- Distinguish between multiple calls of the same function

```python
def ident(i):
    return i
...
a = 5
b = "hi"
c = ident(a)
d = ident(b)
```

c and d have different values

Data Sensitivity:

- Being able to follow pointers /information through indirection

```python
def changeX(obj):
        obj.x = 5
a = Thing()
a.x = "hello"
b = a
changeX(b)
print(a.x)
```

a.x is now 5

Taint Analysis Example

```
class A {
  ...
   Text getT() {
      return this.t;
   }
}
A a; ...
a.t = pw;
msg = a.getT();
```

Want to prove that super important value `pw` gets passed to `msg`...

We want to be **sound**
→ No false negatives

Taint Analysis Example

```
1 class A {
2   ...
3   Text getT() {
4     return this.t;
5   }
6 }
7 A a; ...
8 a.t = pw;
9 msg = a.getT();
```

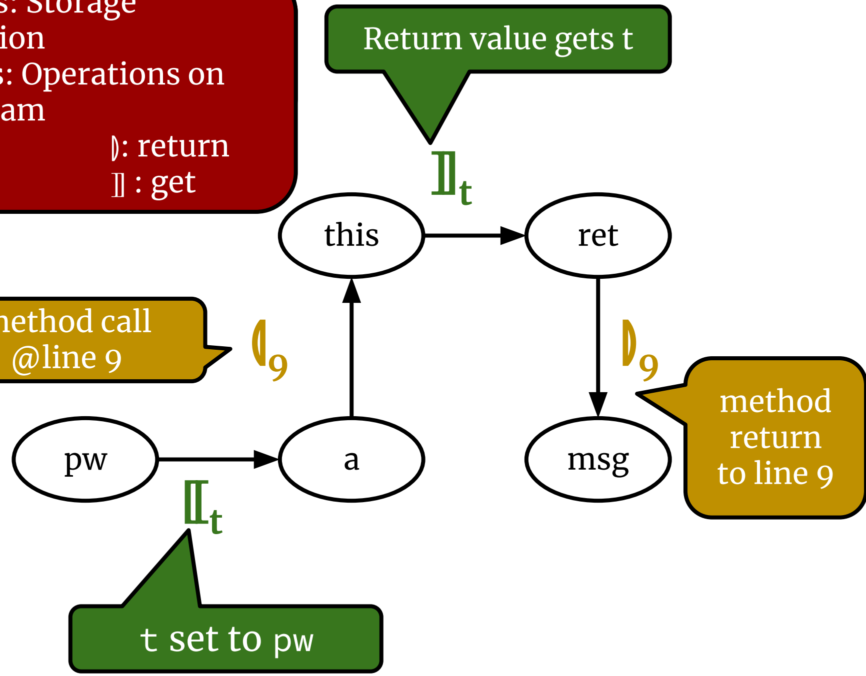
Return value gets t

method return to line 9

t set to pw

method call @line 9

Nodes: Storage Location
Edges: Operations on Program
⦇: call ⦈: return
⟦: set ⟧: get



Taint Analysis Example

If there is a **valid path** from **`pw`** to **`msg`**, then it means that the value of **`pw`** would be propagated to **`msg`**

IS $[_t (_9]_t)_9$ VALID?

Dyck Language (Single Parenthesis)

- Matched Parenthesis

- Dyck language D_k of size k...

  $S \rightarrow SS \mid \varepsilon \mid ⦅_0 S ⦆_0 \mid \dots \mid ⦅_k S ⦆_k$

- ex)
  - $⦅_a ⦆_a ⦅_a ⦆_a$ ✔
  - $⦅_a ⦅_b ⦆_b ⦆_a$ ✔
  - $⦅_a ⦅_b ⦆_a ⦆_b$ ✘

Interleaved Dyck Languages (Paren + Bracket)

$\mathbb{D}_{C}$: Context Sensitivity

ex) $\llparenthesis_9 \llparenthesis_3 \rrparenthesis_3 \rrparenthesis_9$

$\mathbb{D}_{D}$: Data/Field Sensitivity

ex) $\llbracket_a \rrbracket_a$

$\mathbb{D}_{CD}$: Interleaved Dyke Languages – both Context Sensitive and Data/Field Sensitive

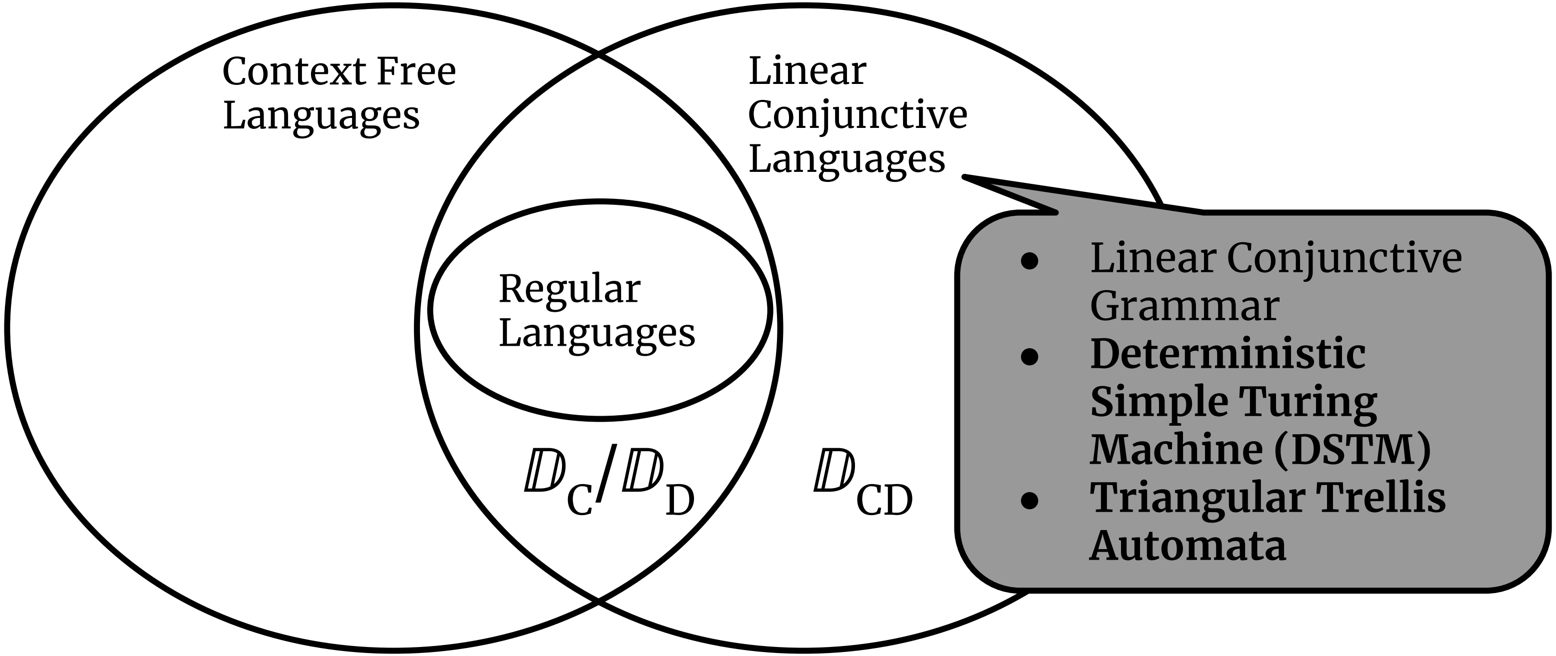
ex) $\llparenthesis_9 \llparenthesis_3 \llbracket_a \rrparenthesis_3 \rrbracket_a \rrparenthesis_9$

Reasoning perfectly about context and data sensitivity simultaneously is UNDECIDABLE!

Problem Space and Contributions

- Static Program Analyses aim for high Context- and Data-Sensitivity in order to reason about programs
- Context-Sensitive Data-Sensitive Analysis can be expressed a graph reachability problem over the Interleaved Dyck Language $\mathbb{D}_{\mathrm{CD}}$
- This problem is undecidable
- $\mathbb{D}_{\mathrm{CD}}$ is a subset of the Linear Conjunctive Languages
- Authors devise approximation algorithms for Linear Conjunctive Language Reachability

Language Classes wrt $\mathbb{D}_{\text{CD}}$...



Deterministic Simple Turing Machine (DSTM)

6-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$

Q : set of states

Σ : input alphabet

Γ : work tape alphabet

$\$$ - end of tape

$\sqcup$ - blank symbol

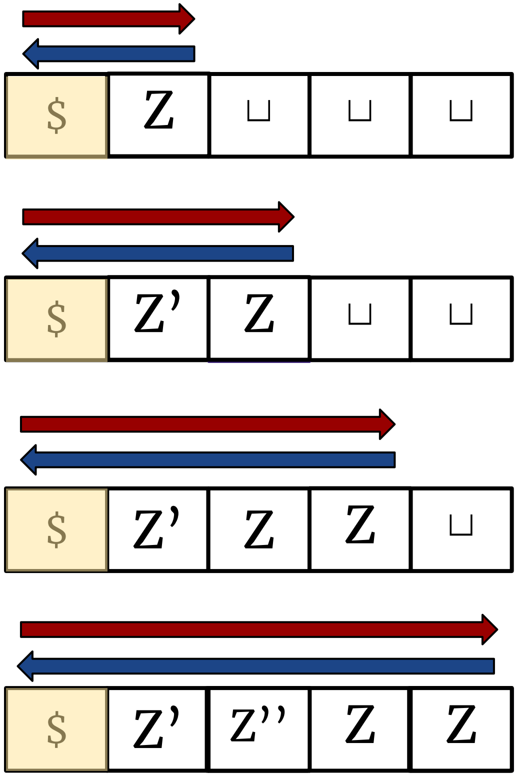
q_0: start state

$F \subseteq Q$: accepting states

δ: transition function

$$Q \times \Gamma \times (\Sigma \cup \{\varepsilon\}) \rightarrow$$
$$Q \times (\Gamma \setminus \{\sqcup\}) \times \{L, R\}$$

DSTM Transition Function Rules



Left to Right Sweep:
- Stay in designated state for L→R sweep
- Keep going Right
- Can’t read input
- Can’t change tape content

Read a $:
- End of Right to Left
- If no more input, computation ends here
- Start of Left to Right otherwise

Read a ⊔:
- End of Left to Right
- **Can** read input
- **Can** write to tape
- Start Right to Left

Right to Left Sweep:
- Can’t go to L→R Sweep state, **can go to any other state**
- Keep going Left
- Can’t read input
- **Can** change tape content

Example... $\llparenthesis_1 \llparenthesis_2 \rrparenthesis_2 \rrparenthesis_1$

q:	L → R	☹			
Z:	$	$\llparenthesis_1$	⊔	⊔	⊔

q:	L → R	☹	☹		
Z:	$	$\llparenthesis_1$	$\llparenthesis_2$	⊔	⊔

q:	L → R	☹	😊	$\llparenthesis_2$	
Z:	$	$\llparenthesis_1$	#	#	⊔

q:	DONE	😊	$\llparenthesis_1$	$\llparenthesis_1$	$\llparenthesis_1$
Z:	$	#	#	#	#

Input $\llparenthesis_i$:

- Want to remember which $\llparenthesis_i$ we read
  → Write $\llparenthesis_i$ to tape
- We need to resolve $\llparenthesis_i$ in the future
  → State ☹

Input $\rrparenthesis_i$:

- Don't need to record this input
  → Write # to tape
- "Looking for" tape symbol $\llparenthesis_i$
  → State $\llparenthesis_i$

State ☹:
There is a $\llparenthesis$

Always write
same symbol

State $\llparenthesis_i$:
Ignores #

Tape symbol $\llparenthesis_i$ →
write # & go to 😊

State 😊:
Ignores #

Tape symbol $\llparenthesis_m$ →
write $\llparenthesis_m$ & go to ☹

DSTM for $\mathbb{D}_C$

Notation: $Z, \sigma \to Z', d$
Z: current tape symbol
σ: current input
Z': tape symbol to write
d: direction to move

Goals:

Want to match $\llparenthesis_i \rrparenthesis_i$ together

$Q = \{😊, ☹\} \cup \{\llparenthesis_0, \ldots\}$

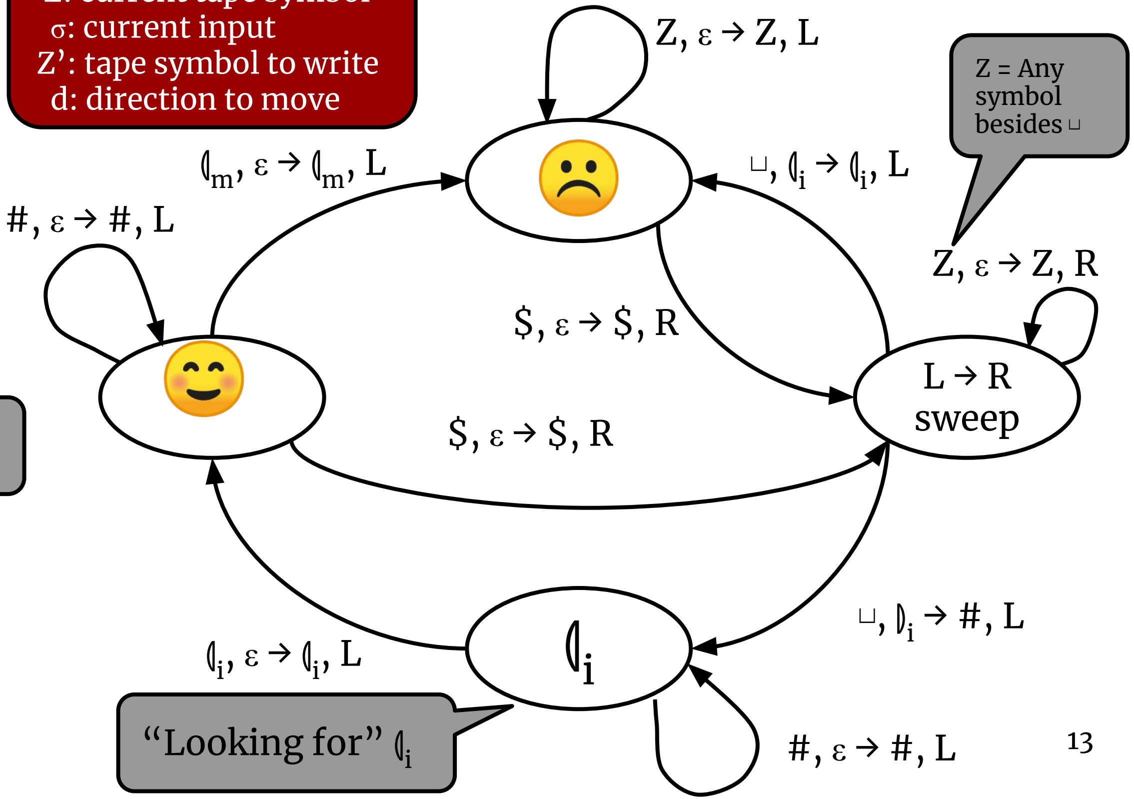
$\Sigma = \{\llparenthesis_0, \rrparenthesis_0, \ldots\}$

"Looking for" $\llparenthesis_0$

$\Gamma = \{\$, \sqcup, \#\} \cup \{\llparenthesis_0, \ldots\}$

$F = \{😊\}$

"Don't care"



Example... $⦅_1⟦_a⦆_1 ⟧_a$

q:	L → R	☹			
Z:	$	$⦅_1$	⊔	⊔	⊔

q:	L → R	☹	☹		
Z:	$	$⦅_1$	$⟦_a$	⊔	⊔

q:	L → R	☹	☹$⦅_1$	$⦅_1$	
Z:	$	#	$⟦_a$	#	⊔

q:	DONE	😊	😊	$⟦_a$	$⟦_a$
Z:	$	#	#	#	#

Input $⦅_i$/$⟦_i$:
→ Write $⦅_i$ /$⟦_i$ to tape
→ State ☹

State ☹:
There is a ⦅/⟦
Always write same symbol

State $⦅_i$:
Ignores #
Tape symbol $⦅_i$
→ write # & go to 😊
NEW: Tape Symbol $⟦_m$
→ write $⟦_m$ & go to state ☹$⦅_i$

State $⟦_i$:
Ignores #
Tape symbol $⟦_i$
→ write # & go to 😊
NEW: Tape Symbol $⦅_m$
→ write $⦅_m$ & go to ☹$⟦_i$

Input $⦆_i$/$⟧_i$:
→ Write # to tape
→ State $⦅_i$/$⟦_i$

State 😊:
Ignores #
Symbol $⦅_m$/$⟦_m$→write $⦅_m$/$⟦_m$ & go to ☹

NEW! State ☹$⦅_i$:
"Looking for $⦅_i$ but seen $⟦_m$"
Ignores #, $⟦_m$

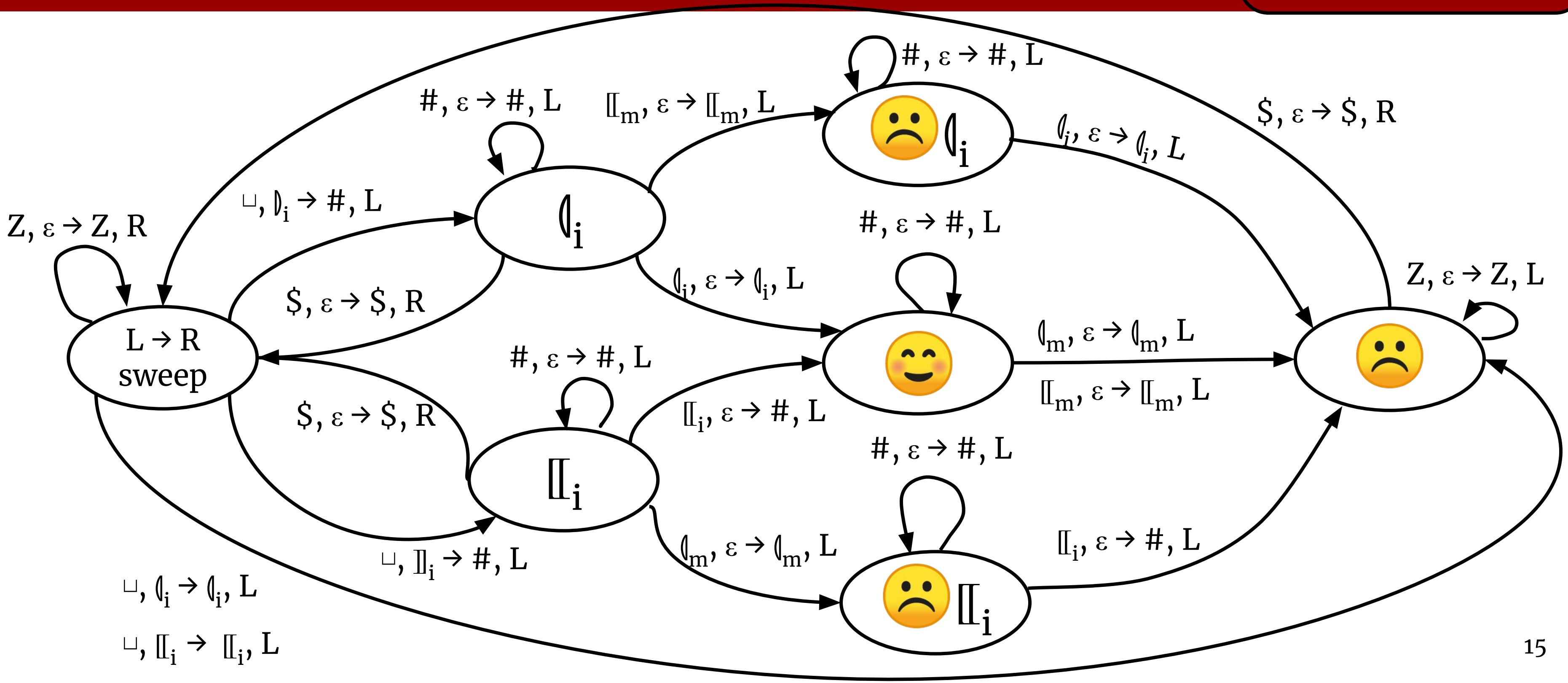
Tape symbol $⦅_i$
→ write # & go to ☹

NEW! State ☹$⟦_i$:
"Looking for $⟦_i$ but seen $⦅_m$"
Ignores #, $⦅_m$

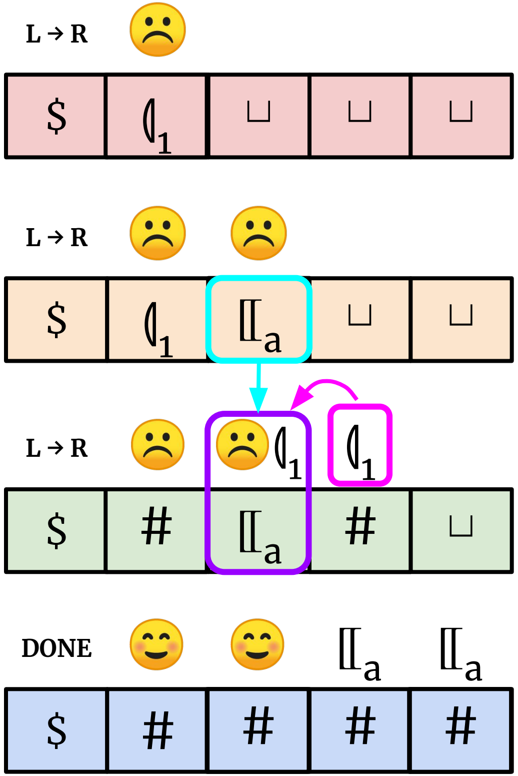
Tape symbol $⟦_i$
→ write # & go to ☹

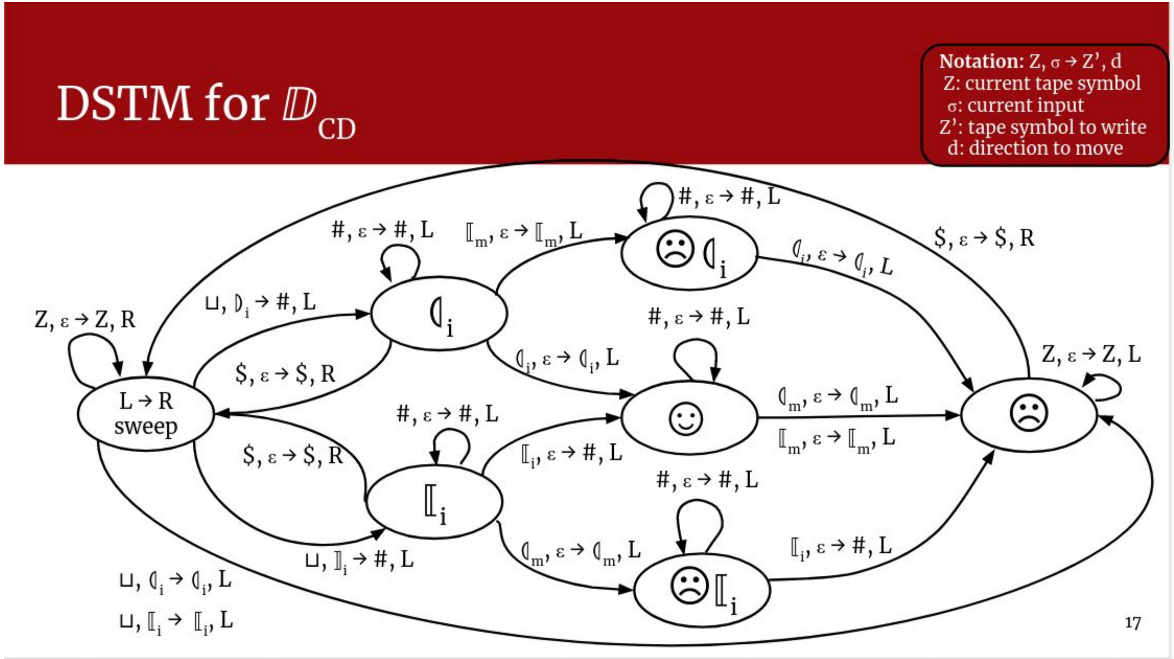
DSTM for $\mathbb{D}_{CD}$

Notation: Z, σ → Z', d
Z: current tape symbol
σ: current input
Z': tape symbol to write
d: direction to move

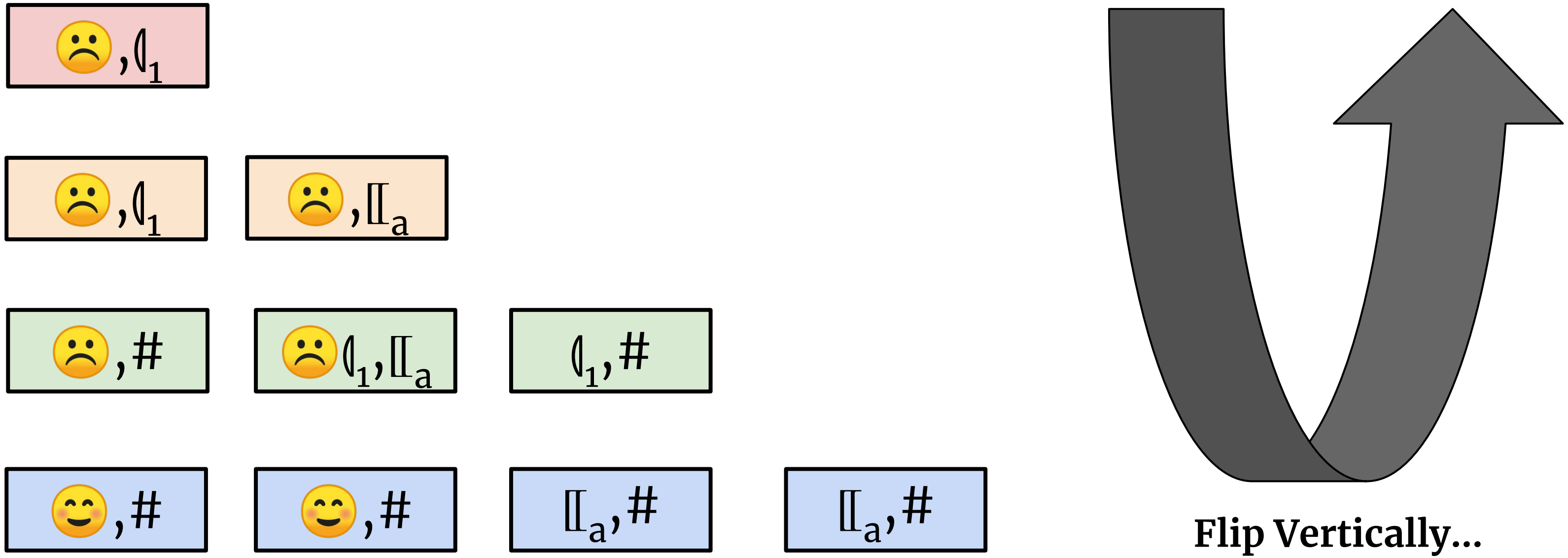


This is quite expensive...





Let’s put together the DSTM state and tape



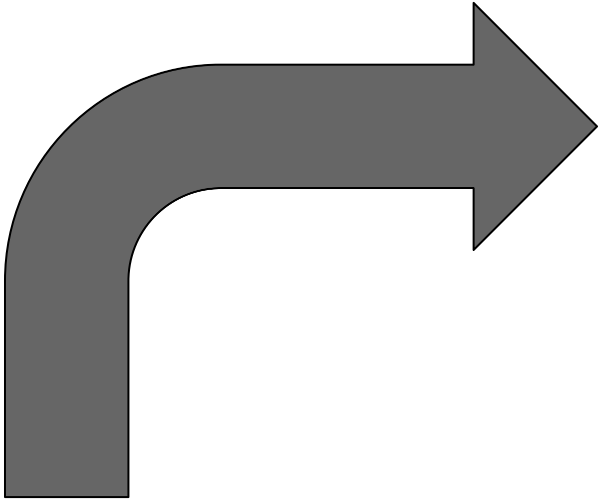
Flip it...

😊, #   😊, # $⟦_a$, # $⟦_a$, #

☹️, #   ☹️ $⦇_1$, $⟦_a$ $⦇_1$, #

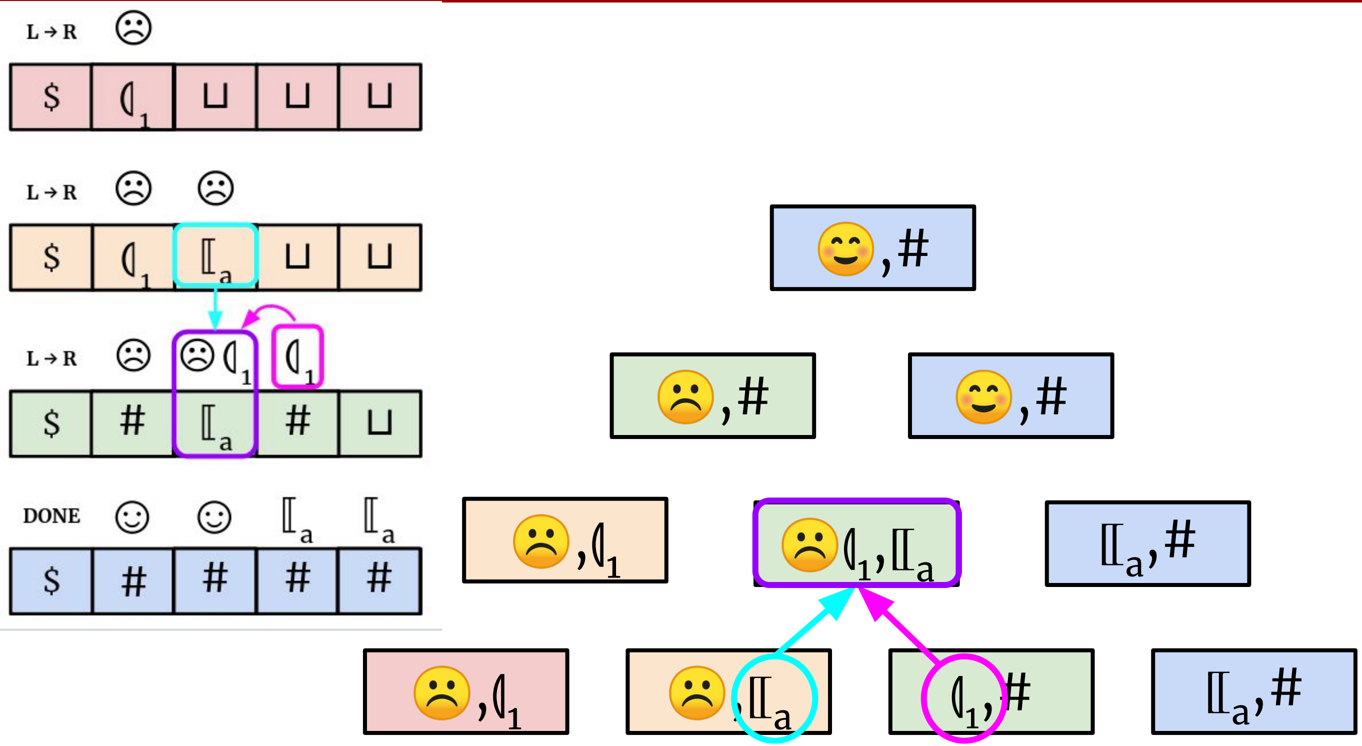
☹️, $⦇_1$ ☹️, $⟦_a$

☹️, $⦇_1$



Small Clockwise Rotation...

Rotate... gives a Triangular Trellis Automaton!!



In DSTM for $\mathbb{D}_{CD}$

$\delta(⦇_1, ⟦_a, \varepsilon) =$

$(☹⦇_1, ⟦_a)$

Triangular Trellis Automata

6-tuple $K = (\Sigma, L, Q_l, I_l, \delta_l, F)$

Σ: input alphabet

L: node labels

Q_l: set of states for L-labeled nodes

I_l: $\Sigma \rightarrow Q_l$ (function generating bottom row states from input)

$\delta : Q_{l1} \times Q_{l2} \rightarrow Q_l$ (transition function where Q_{l1}/Q_{l2} is left/right child)

$F \subseteq Q_l$: set of accepting states

Quick Trellis Example

Input: “b b a”

F: {c}

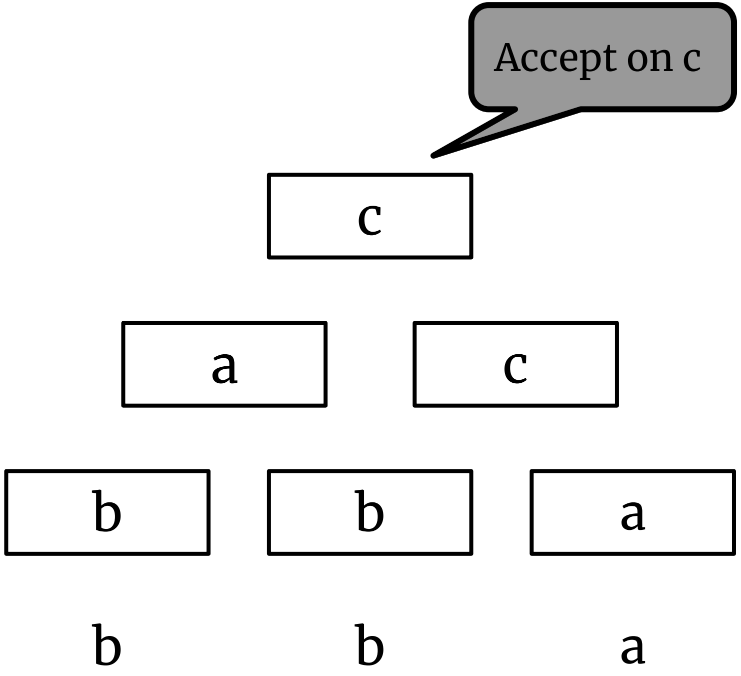
I_1 : identity

δ_1:

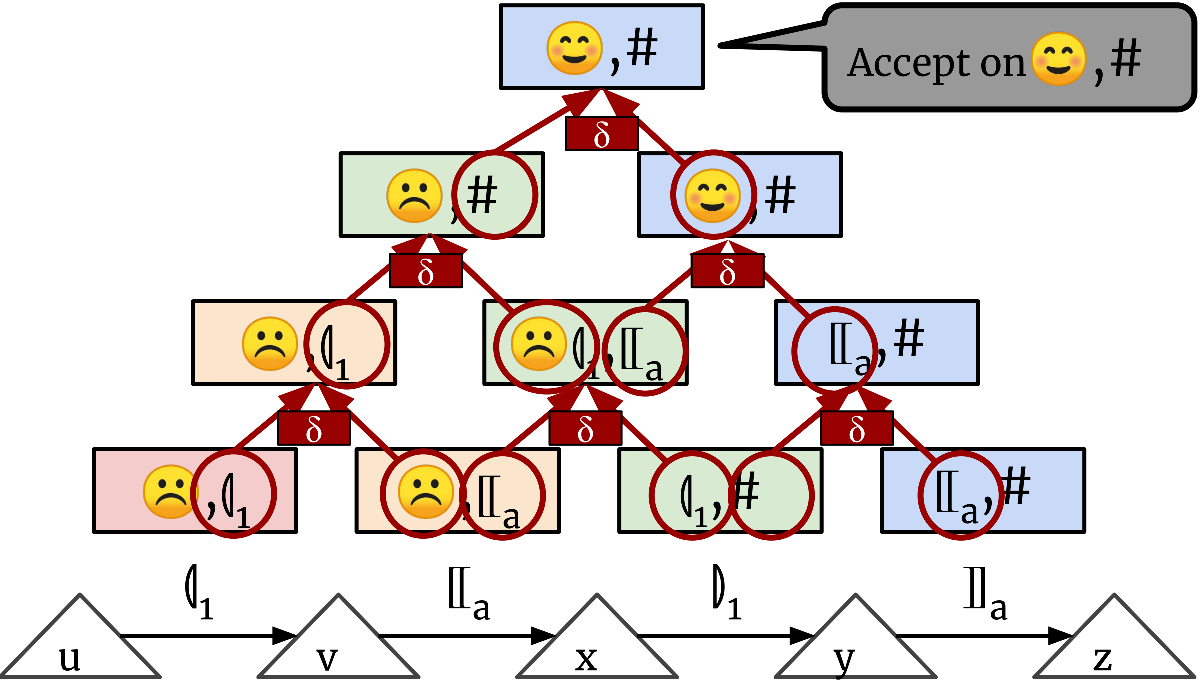
Right Child

Left Child

	a	b	c
a	a	a	c
b	c	a	b
c	c	b	a



Trellis for Computing membership of ⦇$_1$⟦$_a$⦈$_1$ ⟧$_a$



LCL Rules Part 2:

Trellis Transition:
$\delta_l((q_1,Z_1), (q_2,Z_2))= q, Z$

If $\delta(q_2,Z_1,\varepsilon) = (q, Z, L)$
in DSTM

Algorithm 1 (Baseline) Overview

- Input: Graph $G = (V, E)$ and LCL rules for $\mathbb{D}_{\mathrm{CD}}$
- Output: Set $\{S(n_1, n_2) \mid n_1, n_2 \in V\}$

- Worklist algorithm
  - Worklist consists of **Summary Edges** $S(n_1, n_2) = (q, Z)$
  - (q, Z) is result of applying Trellis transition function on a path $n_1 \rightarrow \ldots \rightarrow n_2$
  - Brute-force applies Triangular Trellis Transition function to corresponding values

- Approximation in $O(|M|mn)$ time with $O(n^2)$ space
  - $|V| = n$, $|E| = m$
  - $|M| = O(q^2Z + qZ^2)$ where q/Z are number of states/tape symbols in DSTM M

Optimizations (Algorithm 2)

- Sanity checks Heuristics for “Feasibility”

- Using Gray-White Trellis Automata (GWTA)
  - Gray nodes: (q, Z) such that q ∈ {😊,☹️}
  - Allows filtering of “spurious” nodes

- Approximation in $O(|\mathbb{D}|mn)$ time with $O(n^2)$ space
  - $|\mathbb{D}| = O((c+d)cd)$ where c/d is size of $\mathbb{D}_C/\mathbb{D}_D$

Benchmarks & Evaluation

- Apply Algorithm 2 to two different analysis
  - Java Alias Analysis
    - field sensitivity edge (u, $[\![_f$, v): field *f* of object *u* may point to *v*
  - Android app Taint analysis
    - field sensitivity edge (u, $[\![_f$, v): value of *u* flows to the field *f* of variable *v*

- Comparison with traditional CFL-reachability algorithm
  - CIFS: Context-insensitive field-sensitive
  - CSFI: Context-sensitive field-insensitive
  - CRFS: Regular-approximating context-sensitive field-sensitive
  - CSFR: Context-sensitive Regular-approximating field-sensitive

Evaluation on Alias Analysis

Program	Precision (#S-pair			Time (s)			Space (MB)		
	CIFS	CSFI	Lin	CIFS	CSFI	Lin	CIFS	CSFI	Lin
antlr	2,153,091	1,827,705	1,519,527	165.46	768.11	7.76	674.42	913.20	3822.73
bloat	2,305,351	1,904,837	1,588,287	195.15	1161.30	11.38	800.62	1142.45	5632.24
chart	6,181,087	5,746,258	4,868,772	1170.75	7190.24	70.96	1588.62	2405.58	18024.00
eclipse	2,300,305	1,826,423	1,515,782	190.66	794.00	9.20	709.95	948.41	4218.52
fop	4,733,519	4,069,404	3,575,835	824.71	4393.99	45.07	1443.70	2166.72	14220.40
hsqldb	2,094,016	1,814,974	1,513,500	149.07	675.71	7.46	630.21	834.00	3534.08
jython	2,786,450	1,878,085	1,557,348	243.20	1225.47	13.33	840.79	1271.31	5420.17
luindex	2,151,119	1,820,530	1,513,628	161.44	698.43	7.33	657.75	866.41	3484.07
lusearch	2,199,729	1,824,220	1,517,576	171.55	718.47	9.75	644.55	895.12	3909.12
pmd	2,353,255	1,879,584	1,570,281	191.28	796.97	10.22	709.25	950.11	4181.22
xalan	2,078,462	1,814,712	1,511,005	147.38	665.81	7.47	622.53	828.00	3390.27

Evaluation on Taint Analysis

Program	Precision (#*S*-pairs)					Time (s)					Space (MB)				
	CIFS	CRFS	CSFI	CSFR	Lin	CIFS	CRFS	CSFI	CSFR	Lin	CIFS	CRFS	CSFI	CSFR	Lin
Backflash	32,081	11,679	7,115	6,819	597	0.09	996.11	0.44	7.10	0.02	11.4	2,054.8	14.0	56.7	16.1
BattaryDoctor	109,662	-	15,978	-	3,071	1.86	-	2.25	-	0.12	43.6	-	47.6	-	97.0
DroidKungFu	41,072	-	11,813	9,523	1,510	0.44	-	0.89	178.34	0.03	18.6	-	19.9	686.3	23.8
Fakebanker	12,098	4,439	2,463	2,363	542	0.08	180.40	0.08	8.90	0.01	9.4	1,038.5	10.4	138.1	9.6
Fakedaum	59,104	-	6,480	6,237	1,560	0.50	-	0.59	206.43	0.04	24.3	-	27.4	1,246.2	42.9
Faketaobao	3,196	1,179	732	676	274	0.02	5.28	0.02	0.34	0.00*	4.7	163.9	5.5	21.5	3.4
Jollyserv	22,960	12,449	1,463	1,182	801	0.19	937.41	0.06	5.91	0.02	10.9	1,519.1	11.2	200.3	11.9
Loozfon	3,044	1,865	646	618	218	0.01	2.15	0.02	0.06	0.00*	3.4	37.5	4.2	5.0	2.6
Phospy	-	-	-	-	1,158K	-	-	-	-	35.57	-	-	-	-	3,383.3
Roidsec	81,485	-	18,598	16,592	611	0.38	-	1.88	55.49	0.02	-	16.7	18.1	167.8	16.8
Scipiex	976,255	-	71,759	-	146,913	333.50	-	21.69	-	2.74	476.2	-	93.3	-	478.6
simhosy	1,711,493	-	171,052	-	30,736	833.30	-	177.33	-	1.46	1,242.8	-	262.7	-	670.9
Skullkey	-	-	-	-	2,703K	-	-	-	-	190.40	-	-	-	-	23,290.8
Uranai	24,802	11,379	1,062	874	587	0.07	554.68	0.07	0.90	0.02	9.9	1,210.1	13.0	47.6	14.0
Zertsecurity	24,534	9,361	2,512	1,705	479	0.08	131.64	0.06	1.06	0.01	7.5	453.5	7.0	26.6	6.0

Summary

- Context-Sensitive Data-Dependent Static Analysis can be formulated as solving Interleaved Dyck Language $\mathbb{D}_{\text{CD}}$ on paths of graphs
  - Reachability on graphs wrt $\mathbb{D}_{\text{CD}}$ is undecidable!

- DSTM/Triangular Trellis lend to sound approximation algorithms
  - $\mathbb{D}_{\text{CD}}$ is subset of Linear Conjunctive Languages (LCL)
  - Reachability on graphs wrt LCL is undecidable!
  - $O(mn)$ time approximation algorithm using ideas from Triangular Trellis