Parsing Techniques for Contex-Free Path Querying

Semyon Grigorev
s.v.grigoriev@spbu.ru
Semen.Grigorev@jetbrains.com

JetBrains Research, Programming Languages and Tools Lab
Saint Petersburg University

April 05, 2019

Formal language constrained path querying

- Finite directed edge-laballed graph $\mathcal{G} = (V, E, L)$
- The path is a word over L
  $\omega(p) = \omega(v_0 \xrightarrow{l_0} v_1 \xrightarrow{l_1} \dots \xrightarrow{l_{n-1}} v_n) = l_0 \cdot l_1 \cdot \ldots \cdot l_{n-1}$
- The language $\mathcal{L}$ (over L)

Formal language constrained path querying

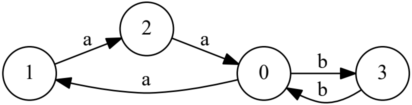
- Finite directed edge-laballed graph $\mathcal{G} = (V, E, L)$
- The path is a word over L
  $\omega(p) = \omega(v_0 \xrightarrow{l_0} v_1 \xrightarrow{l_1} \ldots \xrightarrow{l_{n-1}} v_n) = l_0 \cdot l_1 \cdot \ldots \cdot l_{n-1}$
- The language $\mathcal{L}$ (over L)

- Reachability problem: $Q = \{(v_i, v_j) \mid \exists p = v_i \ldots v_j, \omega(p) \in \mathcal{L}\}$
- Path querying problem: $Q = \{p \mid \omega(p) \in \mathcal{L}\}$
  - Single path, all paths, shortest path...

Context-Free Path Querying

- $\mathcal{L}$ is a context-free language
- $G_{\mathcal{L}} = (N, \Sigma, R, S)$
- Reachability problem: $Q = \{(v_i, v_j) \mid \exists p = v_i \dots v_j, S \xrightarrow[G_L]{*} \omega(p)\}$
- Path querying problem: $Q = \{p \mid \omega(p) \in \mathcal{L}\}$

Example of CFPQ



Input graph

$S \to a\ S\ b$
$S \to Middle$
$Middle \to a\ b$

Query: language $\{a^n b^n \mid n > 0\}$

Paths:

$2 \xrightarrow{a} 0 \xrightarrow{b} 3$

$1 \xrightarrow{a} 2 \xrightarrow{a} 0 \xrightarrow{b} 3 \xrightarrow{b} 0$

$p_1 = 0 \xrightarrow{a} 1 \xrightarrow{a} 2 \xrightarrow{a} 0 \xrightarrow{b} 3 \xrightarrow{b} 0 \xrightarrow{b} 3$

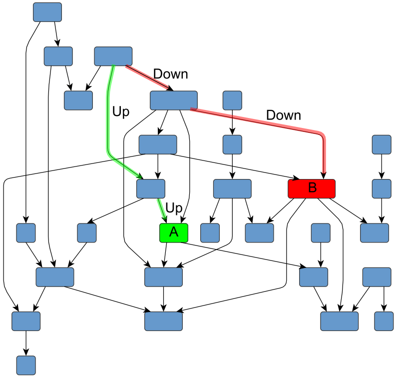
$p_2 = 0 \xrightarrow{a} 1 \xrightarrow{a} 2 \xrightarrow{a} 0 \xrightarrow{a} 1 \xrightarrow{a} 2 \xrightarrow{a} 0 \xrightarrow{b} 3 \xrightarrow{b} 0 \xrightarrow{b} 3 \xrightarrow{b} 0 \xrightarrow{b} 3 \xrightarrow{b} 0$

$\ldots$

Applications

- Graph databases querying
  Mihalis Yannakakis. "Graph-theoretic methods in database theory." 1990.
- Static code analysis
  Thomas Reps et al. "Precise interprocedural dataflow analysis via graph reachability." 1995
- ...

Graph databases querying



Navigation through a graph

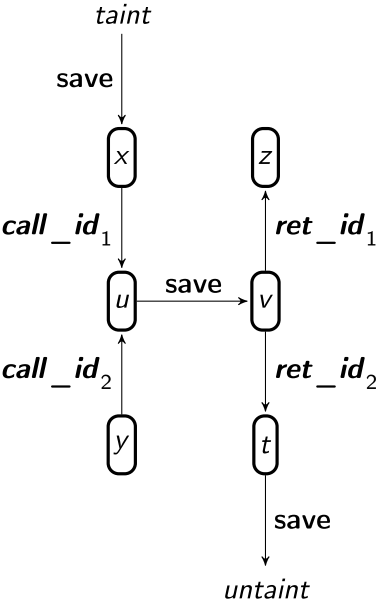
- Are nodes A and B on the same level of hierarchy?
- Is there a path of form **Up**^{n} **Down**^{n}?
- Find all paths of form **Up**^{n} **Down**^{n} which start from the node A

Context-free path querying

- *Sevon P., Eronen L.* "Subgraph queries by context-free grammars." 2008
- *Hellings J.* "Conjunctive context-free path queries." 2014
- *Zhang X. et al.* "Context-free path queries on RDF graphs." 2016

Static code analysis

```
int id(int u)
{
  v = u;
  return v;
}
int main()
{
  //taint
  int x;
  int z, y;
  //untaint
  int t;
  z = id(x);
  t = id(y);
}
```



Static code analysis (Language Reachability Framework)

- *Thomas Reps et al.* "Precise interprocedural dataflow analysis via graph reachability." 1995
- *Dacong Yan et al.* "Demand-driven context-sensitive alias analysis for Java." 2011
- *Jakob Rehof and Manuel Fahndrich.* "Type-base flow analysis: from polymorphic subtyping to CFL-reachability." 2001

Static code analysis (Language Reachability Framework)

- *Thomas Reps et al.* "Precise interprocedural dataflow analysis via graph reachability." 1995
- *Dacong Yan et al.* "Demand-driven context-sensitive alias analysis for Java." 2011
- *Jakob Rehof and Manuel Fahndrich.* "Type-base flow analysis: from polymorphic subtyping to CFL-reachability." 2001
- *Qirun Zhang and Zhendong Su.* "Context-sensitive data-dependence analysis via linear conjunctive language reachability." 2017

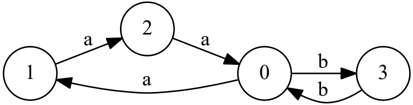
Parsing algorithms for CFPQ

- Structural representation of results
- Number of algorithms with different properties
- Number of theoretical results

Parsing algorithms for CFPQ

- Structural representation of results
- Number of algorithms with different properties
- Number of theoretical results
- Interconnection between different areas

Structural representation of result

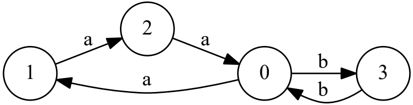


Input graph

$S \to a\ S\ b$
$S \to Middle$
$Middle \to a\ b$

Grammar

Structural representation of result



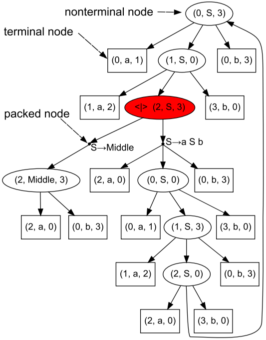
Input graph

$S \rightarrow a\ S\ b$

$S \rightarrow Middle$

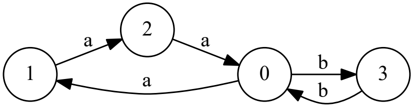
$Middle \rightarrow a\ b$

Grammar



Query result (SPPF)

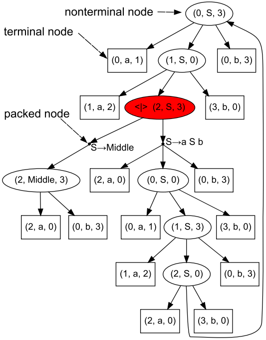
Structural representation of result



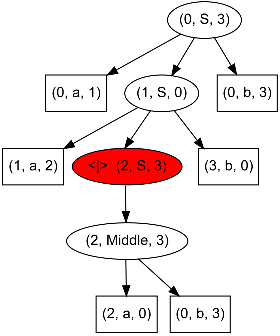
Input graph

$$S \to a\ S\ b$$
$$S \to Middle$$
$$Middle \to a\ b$$

Grammar

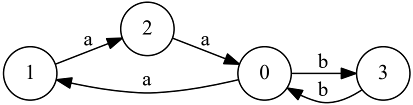


Query result (SPPF)



Tree for p_1

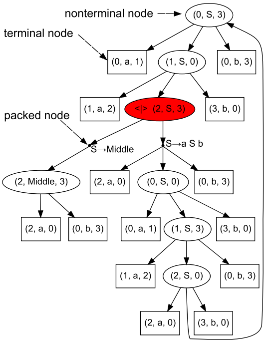
Structural representation of result



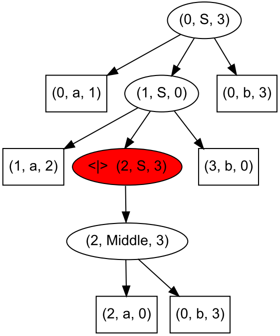
Input graph

$$S \to a\ S\ b$$
$$S \to Middle$$
$$Middle \to a\ b$$

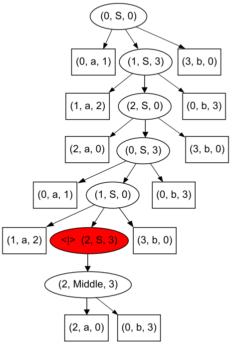
Grammar



Query result (SPPF)

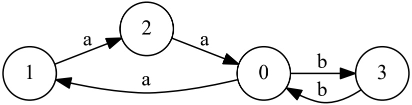


Tree for p_1

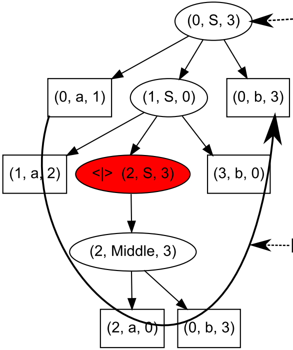


Tree for p_2

Paths extraction



$$S \to a\ S\ b$$
$$S \to Middle$$
$$Middle \to a\ b$$



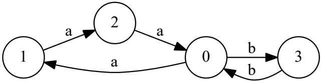
There exists a path p from vertex 0 to vetex 3 such that $\omega(p)$ is derivable from S. Subtree which is rooted by this node is a derivation tree of $\omega(p)$.

Path is a seqence of leaves

Path: $0 \xrightarrow{a} 1 \xrightarrow{a} 2 \xrightarrow{a} 0 \xrightarrow{b} 3 \xrightarrow{b} 0 \xrightarrow{b} 3$

Bar-Hillel theorem

Context-free languages are closed under intersection with regular languages



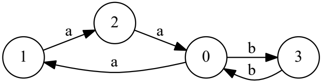
Regular language

$$S \rightarrow a\ S\ b$$
$$S \rightarrow Middle$$
$$Middle \rightarrow a\ b$$

Context-free language

Bar-Hillel theorem

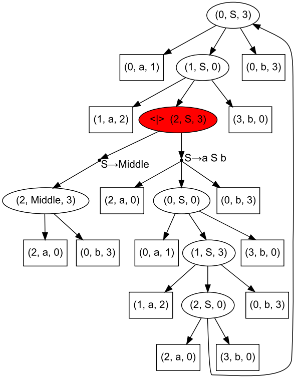
Context-free languages are closed under intersection with regular languages



$$S \to a\ S\ b$$
$$S \to Middle$$
$$Middle \to a\ b$$

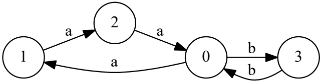
Regular language

Context-free language



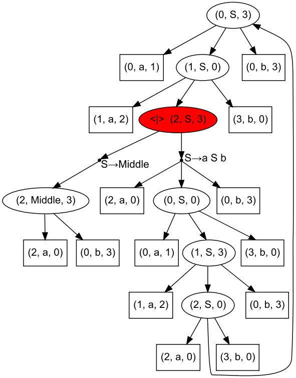
Bar-Hillel theorem

Context-free languages are closed under intersection with regular languages



$$S \to a\ S\ b$$
$$S \to Middle$$
$$Middle \to a\ b$$

Regular language Context-free language



$$
\begin{aligned}
(0, S, 3) &\to (0, a, 1)\ (1, S, 0)\ (0, b, 3) \\
(1, S, 0) &\to (1, a, 2)\ (2, S, 3)\ (3, b, 0) \\
(2, S, 3) &\to (2, a, 0)\ (0, S, 0)\ (0, b, 3) \\
(2, S, 3) &\to (2, Middle, 3) \\
(0, S, 0) &\to (0, a, 1)\ (1, S, 3)\ (3, b, 0) \\
(1, S, 3) &\to (1, a, 2)\ (2, S, 0)\ (0, b, 3) \\
(2, S, 0) &\to (2, a, 0)\ (0, S, 3)\ (3, b, 0) \\
(0, Middle, 3) &\to (2, a, 0)\ (0, b, 3)
\end{aligned}
$$

Our experiments

- Generalized LR for CFPQ
  - Based on Right Nulled Generalized LR: *Scott E., Johnstone A.* "Right Nulled GLR Parsers"
  - *Ekaterina Verbitskaia, Semyon Grigorev, and Dmitry Avdyukhin.* "Relaxed Parsing of Regular Approximations of String-Embedded Languages" 2015

Our experiments

- Generalized LR for CFPQ
  - Based on Right Nulled Generalized LR: *Scott E., Johnstone A.* "Right Nulled GLR Parsers"
  - *Ekaterina Verbitskaia, Semyon Grigorev, and Dmitry Avdyukhin.* "Relaxed Parsing of Regular Approximations of String-Embedded Languages" 2015
- Generalized LL for CFPQ (**GLL**)
  - Based on Generalized LL: *Scott E., Johnstone A.* "GLL parsing"
  - *Semyon Grigorev and Anastasiya Ragozina.* "Context-free path querying with structural representation of result." 2017

Query language integration

How to integrate query language into a general-purpose programming language?

- Transparency
- Compositionality
- Static error checking

Query language integration

How to integrate query language into a general-purpose programming language?

- Transparency
- Compositionality
- Static error checking

- String-embedded languages
- ORMs
- Combinators

Combinators for CFPQ

- Implemented in Scala
- Based on Meerkat parser combinator library: *Anastasia Izmaylova, Ali Afroozeh, and Tijs van der Storm.* “Practical, general parser combinators” 2016
- *Ekaterina Verbitskaia, Ilya Kirillov, Ilya Nozkin, Semyon Grigorev.* “Parser Combinators for Context-Free Path Querying” 2019

Supported combinators

Combinator	Description
a ~ b	sequential parsing: a then b
a | b	choice: a or b

Supported combinators

Combinator	Description
a ~ b	sequential parsing: a then b
a | b	choice: a or b
a ?	optional parsing: a or nothing
a *	repetition of zero or more a
a +	repetition of at least one a

Supported combinators

Combinator	Description
a ~ b	sequential parsing: a then b
a | b	choice: a or b
a ?	optional parsing: a or nothing
a $*$	repetition of zero or more a
a $+$	repetition of at least one a
a ^ f	apply f function to a if a is a token
a ^^	capture output of a if a is a token
a & f	apply f function to a if a is a parser
a &&	capture output of a if a is a parser

A set of functions to handle values of edges and vertices

```
def LV(labels: String*) =
  V(e => labels.forall(e.hasLabel))
def outLE(label:String) = outE(_.label() == label)
def inLE (label:String) = inE (_.label() == label)
```

Basic example

Is there a path from vertex 0 to vertex 3 which has form $a^n b^n$?

```
val Query : Nonterminal
    = syn (LV("0") ~ S ~ LV("3"))

val S: Nonterminal
    = syn ( "a" ~ S ~ "b"
          | "a" ~ "b"
          )
```

Example of generalization

```
def sameGen(brs) =
  reduceChoice(
    brs.map {case (lbr, rbr) =>
      lbr ~ syn(sameGen(brs).?) ~ rbr})
```

Example of generalization

```
def sameGen(brs) =
  reduceChoice(
    brs.map {case (lbr, rbr) =>
      lbr ~ syn(sameGen(brs).?) ~ rbr})

val query1 = syn(sameGen(List(("a", "b"))))

val query2 = syn(
  sameGen(List((p1, p2),("(",")"))) ~ p3)
```

Example of values handling

Actors who played in some film

In Cypher

```
MATCH (m:Movie {title: 'Forrest Gump'})
      <-[:ACTS_IN]-(a:Actor)
RETURN a.name, a.birthplace;
```

In Meerkat

```
val query =
  syn((
    (LV("Movie")::V(_.title == "Forrest Gump")) ~
    inLE("ACTS_IN") ~
    syn(LV("Actor") ^
          (e => (e.name, e.birthplace)))) &&)
executeQuery(query, input)
```

Limitations

- Overhead for the regular constraints
- Not exactly clear how to compute arbitrary semantics for the paths
  - Paths can be lazily extracted, but in which order?
  - What kind of semantics can be calculated when there are cycles?

Boolean Matrix Multiplication for CFPQ

- *Rustam Azimov, Semyon Grigorev.* "Context-free path querying by matrix multiplication." 2017
- *Semyon Grigorev, et. al.* "Evaluation of the Context-Free Path Querying Algorithm Based on Matrix Multiplication" 2019

Transitive closure

- Subset multiplication, $N_1, N_2 \subseteq N$
  - $N_1 \cdot N_2 = \{A \mid \exists B \in N_1, \exists C \in N_2 \text{ such that } (A \to BC) \in P\}$
- Subset addition: set-theoretic union.

- Matrix multiplication
  - Matrix of size $|V| \times |V|$
  - Subsets of N are elements
  - $c_{i,j} = \bigcup_{k=1}^{n} a_{i,k} \cdot b_{k,j}$

- Transitive closure
  - $a^{cf} = a^{(1)} \cup a^{(2)} \cup \cdots$
  - $a^{(1)} = a$
  - $a^{(i)} = a^{(i-1)} \cup (a^{(i-1)} \times a^{(i-1)}),\ i \geq 2$

The algorithm

Algorithm Context-free recognizer for graphs

function CONTEXTFREEPATHQUERYING(D, G)
    $n \leftarrow$ the number of nodes in D
    $E \leftarrow$ the directed edge-relation from D
    $P \leftarrow$ the set of production rules in G
    $T \leftarrow$ the matrix $n \times n$ in which each element is $\varnothing$
    for all $(i, x, j) \in E$ **do** ▷ Matrix initialization
        $T_{i,j} \leftarrow T_{i,j} \cup \{A \mid (A \rightarrow x) \in P\}$
    while matrix T is changing **do**
        $T \leftarrow T \cup (T \times T)$ ▷ Transitive closure T^{cf} calculation
    return T

Boolean Matrix Multiplication for CFPQ

- A matrix for a nonterminal is a set of boolean matrices
- Matrix multiplication can be implemented efficiently by using modern harware and high-performance libraries

Performance comparison setup

We use graphs from the classical set of ontologies: *skos*, *foaf*, *univ-bench*, *wine*, *pizza*, *etc.*

Queries are classical variants of the same-generation query

$$
\begin{aligned}
\mathbf{S} &\rightarrow subClassOf^{-1}\ \mathbf{S}\ subClassOf \\
\mathbf{S} &\rightarrow type^{-1}\ \mathbf{S}\ type \\
\mathbf{S} &\rightarrow subClassOf^{-1}\ subClassOf \\
\mathbf{S} &\rightarrow type^{-1}\ type
\end{aligned}
$$

Query 1

$$
\begin{aligned}
\mathbf{S} &\rightarrow \mathbf{B}\ subClassOf \\
\mathbf{S} &\rightarrow subClassOf \\
\mathbf{B} &\rightarrow subClassOf^{-1}\ \mathbf{B}\ subClassOf \\
\mathbf{B} &\rightarrow subClassOf^{-1}\ subClassOf
\end{aligned}
$$

Query 2

Performance comparison results

№	#V	#E	Query 1 (ms)			Query 2 (ms)	
			CYK[1]	GLL	GPGPU	GLL	GPGPU
1	144	323	1044	10	12	1	1
2	129	351	6091	19	13	1	0
3	131	397	13971	24	30	1	10
4	179	413	20981	25	15	11	9
5	337	834	82081	89	32	3	6
6	291	685	515285	255	22	66	2
7	341	711	420604	261	20	45	24
8	671	2604	3233587	697	24	29	23
9	733	2450	4075319	819	54	8	6
10	6224	11840	–	1926	82	167	38
11	5864	19600	–	6246	185	46	21
12	5368	20832	–	7014	127	393	40

[1] Zhang, et al. "Context-free path queries on RDF graphs."

Performance comparison results

- Data from *Zhiwei Fan, et.al.* "Scaling-Up In-Memory Datalog Processing: Observations and Techniques." 2018.
- Graphs with names of form $Gn - p$: n is a number of vertices, edge between two vertices exists with probability p

Graph	M4RI(sec)	GPU_N(sec)	GPGPU(sec)
G10k-0.01	1.455	0.138	47.525
G10k-0.1	1.050	0.114	395.393
G20k-0.001	11.025	1.274	-
G40k-0.001	97.841	8.393	-
G80k-0.001	1142.959	65.886	-

Directions for research: engineering part

- Develop parallel and distributed algorithms
  - Destributed GLL(GLR)-based algortihms
  - Destributed matrix multiplicatioms algorithms
  - Efficient implementation of sparse boolean matrices multiplication algorithms
- Adopt other parsing algortihms
  - Brzozowski's derivatives
  - Derivatives for graph querying: *Maurizio Nole and Carlo Sartiani.* "Regular path queries on massive graphs." 2016
  - Derivatives for context-free parsing: *Matthew Might, David Darais, and Daniel Spiewak.* "Parsing with derivatives: a functional pearl." 2011.
- Utilize other classes of languages for constraints specification
- Investigate incremental queries evaluation

Directions for research: theoretical part

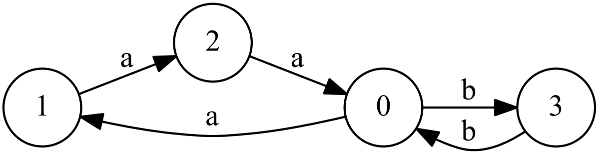
- Time complexity of GLL(GLR)-based algorithms for different classes of grammars
  - Current result for GLL-based: $O\left(|V|^3 * \max_{v \in V}(deg^+(v))\right)$
- Theoretical lower bound for CFPQ
  - Is it possible to reduce CFPQ to Õ(BMM)?
    - ⋆ Our result is $O(|V|^2|N|^3(BMM(|V|) + BMU(|V|)))$
  - $\tilde{O}(n^\omega)$ solution for Dyck with one type of brackets: *Phillip G. Bradford.* "Efficient Exact Paths For Dyck and semi-Dyck Labeled Path Reachability." 2018.

BMM-based algorithm: the worst case

Input graph: two cycles connected via a shared node

- first cycle has $2^k + 1$ edges labeled a
- second cycle has 2^k edges labeled b

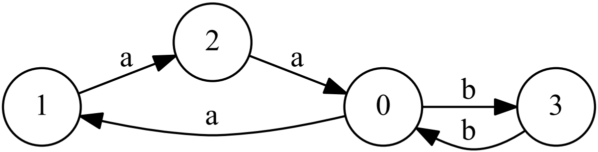
$$
\begin{array}{lcl}
S & \rightarrow & a\ S\ b \\
  & \mid & a\ b
\end{array}
$$



The worst case: step by step

$$\begin{array}{lcl} S & \to & A\ B \\ & | & A\ S_1 \\ S_1 & \to & S\ B \\ A & \to & a \\ B & \to & b \end{array}$$

$$T_0 = \begin{pmatrix} \varnothing & \{A\} & \varnothing & \{B\} \\ \varnothing & \varnothing & \{A\} & \varnothing \\ \{A\} & \varnothing & \varnothing & \varnothing \\ \{B\} & \varnothing & \varnothing & \varnothing \end{pmatrix}$$



The worst case: step by step

$$T_0 \times T_0 = \begin{pmatrix} \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \{S\} \\ \varnothing & \varnothing & \varnothing & \varnothing \end{pmatrix}$$

$$T_1 = T_0 \cup (T_0 \times T_0) = \begin{pmatrix} \varnothing & \{A\} & \varnothing & \{B\} \\ \varnothing & \varnothing & \{A\} & \varnothing \\ \{A\} & \varnothing & \varnothing & \{\boldsymbol{S}\} \\ \{B\} & \varnothing & \varnothing & \varnothing \end{pmatrix}$$

The worst case: step by step

$$T_2 = \begin{pmatrix} \varnothing & \{A\} & \varnothing & \{B\} \\ \varnothing & \varnothing & \{A\} & \varnothing \\ \{A, \boldsymbol{S_1}\} & \varnothing & \varnothing & \{S\} \\ \{B\} & \varnothing & \varnothing & \varnothing \end{pmatrix}$$

The worst case: step by step

$$T_3 = \begin{pmatrix} \varnothing & \{A\} & \varnothing & \{B\} \\ \{\boldsymbol{S}\} & \varnothing & \{A\} & \varnothing \\ \{A, S_1\} & \varnothing & \varnothing & \{S\} \\ \{B\} & \varnothing & \varnothing & \varnothing \end{pmatrix}$$

The worst case: step by step

$$T_4 = \begin{pmatrix} \varnothing & \{A\} & \varnothing & \{B\} \\ \{S\} & \varnothing & \{A\} & \{\boldsymbol{S_1}\} \\ \{A, S_1\} & \varnothing & \varnothing & \{S\} \\ \{B\} & \varnothing & \varnothing & \varnothing \end{pmatrix}$$

The worst case: step by step

$$T_5 = \begin{pmatrix} \varnothing & \{A\} & \varnothing & \{B, \boldsymbol{S}\} \\ \{S\} & \varnothing & \{A\} & \{S_1\} \\ \{A, S_1\} & \varnothing & \varnothing & \{S\} \\ \{B\} & \varnothing & \varnothing & \varnothing \end{pmatrix}$$

The worst case: step by step

$$T_6 = \begin{pmatrix} \{\boldsymbol{S_1}\} & \{A\} & \varnothing & \{B, S\} \\ \{S\} & \varnothing & \{A\} & \{S_1\} \\ \{A, S_1\} & \varnothing & \varnothing & \{S\} \\ \{B\} & \varnothing & \varnothing & \varnothing \end{pmatrix}$$

The worst case: step by step

$$T_7 = \begin{pmatrix} \{S_1\} & \{A\} & \varnothing & \{B, S\} \\ \{S\} & \varnothing & \{A\} & \{S_1\} \\ \{A, S_1, \boldsymbol{S}\} & \varnothing & \varnothing & \{S\} \\ \{B\} & \varnothing & \varnothing & \varnothing \end{pmatrix}$$

The worst case: step by step

$$T_8 = \begin{pmatrix} \{S_1\} & \{A\} & \varnothing & \{B, S\} \\ \{S\} & \varnothing & \{A\} & \{S_1\} \\ \{A, S_1, S\} & \varnothing & \varnothing & \{S, \boldsymbol{S_1}\} \\ \{B\} & \varnothing & \varnothing & \varnothing \end{pmatrix}$$

The worst case: step by step

$$T_9 = \begin{pmatrix} \{S_1\} & \{A\} & \varnothing & \{B, S\} \\ \{S\} & \varnothing & \{A\} & \{S_1, \boldsymbol{S}\} \\ \{A, S_1, S\} & \varnothing & \varnothing & \{S, S_1\} \\ \{B\} & \varnothing & \varnothing & \varnothing \end{pmatrix}$$

The worst case: step by step

$$T_{10} = \begin{pmatrix} \{S_1\} & \{A\} & \varnothing & \{B, S\} \\ \{S, \boldsymbol{S_1}\} & \varnothing & \{A\} & \{S_1, S\} \\ \{A, S_1, S\} & \varnothing & \varnothing & \{S, S_1\} \\ \{B\} & \varnothing & \varnothing & \varnothing \end{pmatrix}$$

The worst case: step by step

$$T_{11} = \begin{pmatrix} \{S_1, \boldsymbol{S}\} & \{A\} & \varnothing & \{B, S\} \\ \{S, S_1\} & \varnothing & \{A\} & \{S_1, S\} \\ \{A, S_1, S\} & \varnothing & \varnothing & \{S, S_1\} \\ \{B\} & \varnothing & \varnothing & \varnothing \end{pmatrix}$$

The worst case: step by step

$$T_{12} = \begin{pmatrix} \{S_1, S\} & \{A\} & \varnothing & \{B, S, \boldsymbol{S_1}\} \\ \{S, S_1\} & \varnothing & \{A\} & \{S_1, S\} \\ \{A, S_1, S\} & \varnothing & \varnothing & \{S, S_1\} \\ \{B\} & \varnothing & \varnothing & \varnothing \end{pmatrix}$$

The worst case: step by step

$$T_{13} = \begin{pmatrix} \{S_1, S\} & \{A\} & \varnothing & \{B, S, S_1\} \\ \{S, S_1\} & \varnothing & \{A\} & \{S_1, S\} \\ \{A, S_1, S\} & \varnothing & \varnothing & \{S, S_1\} \\ \{B\} & \varnothing & \varnothing & \varnothing \end{pmatrix}$$