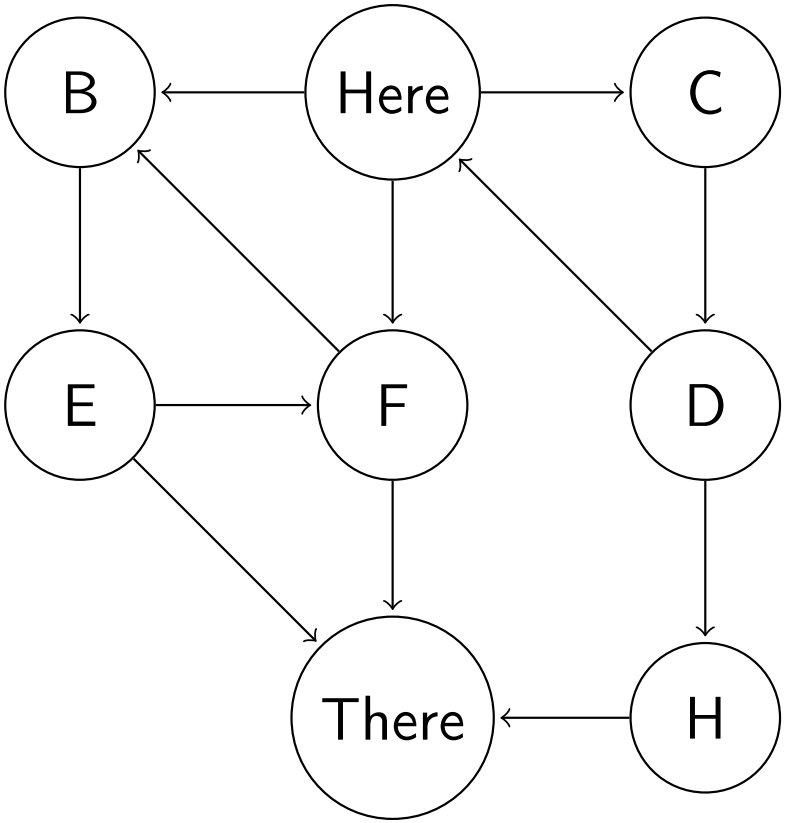
Parser Combinators for Context-Free Path Querying

Kate Verbitskaia, Ilya Kirillov, Ilya Nozkin, Semyon Grigorev

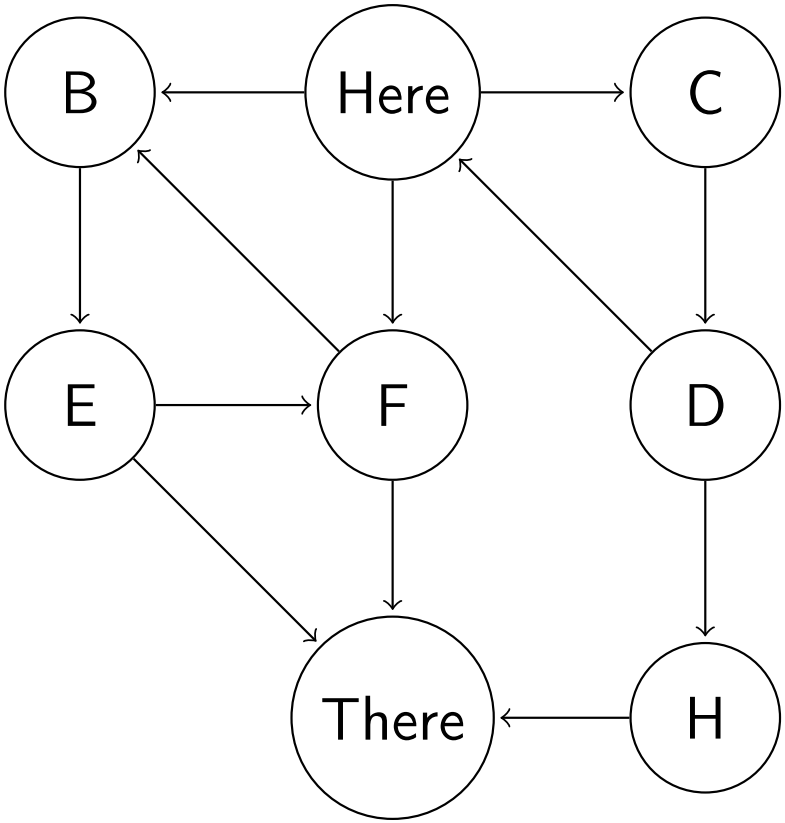
JetBrains Research, Programming Languages and Tools Lab
Saint Petersburg University

September 28, 2018

Path Querying

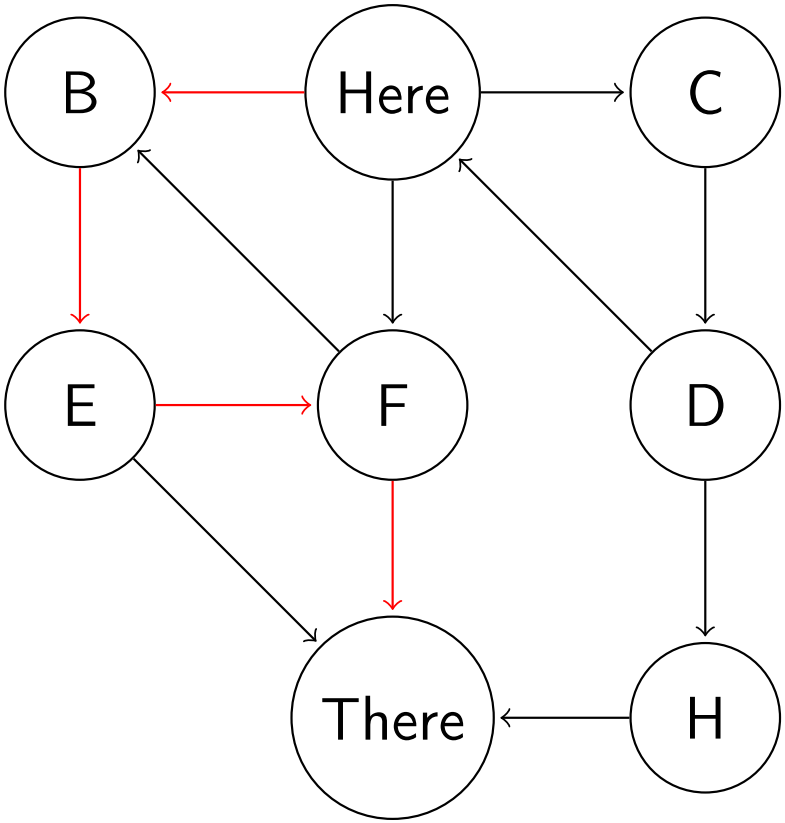


Path Querying



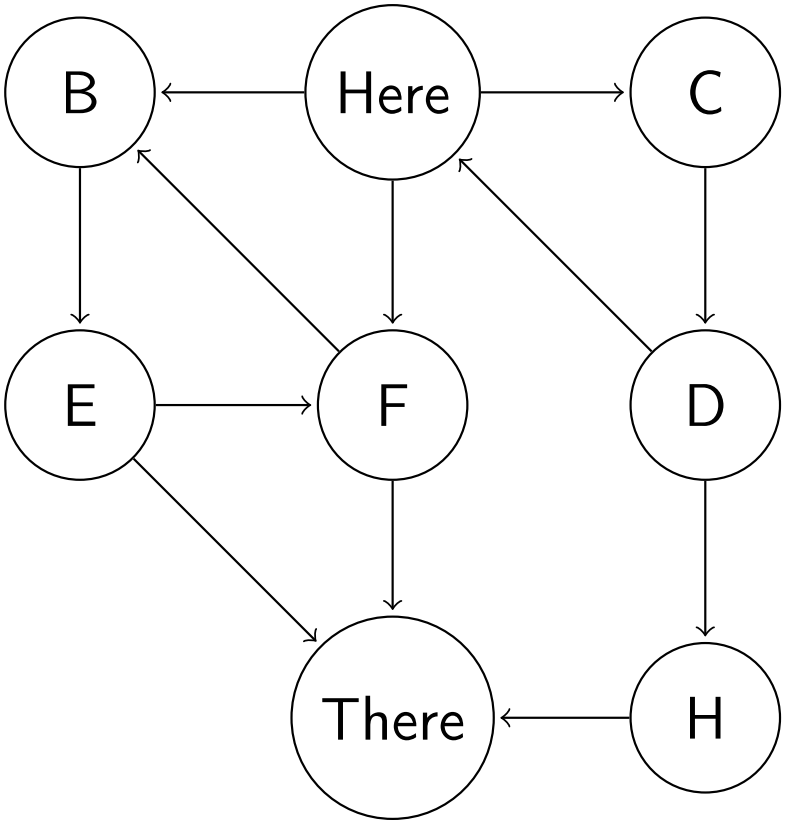
Is there a path from **Here** to **There**?

Path Querying



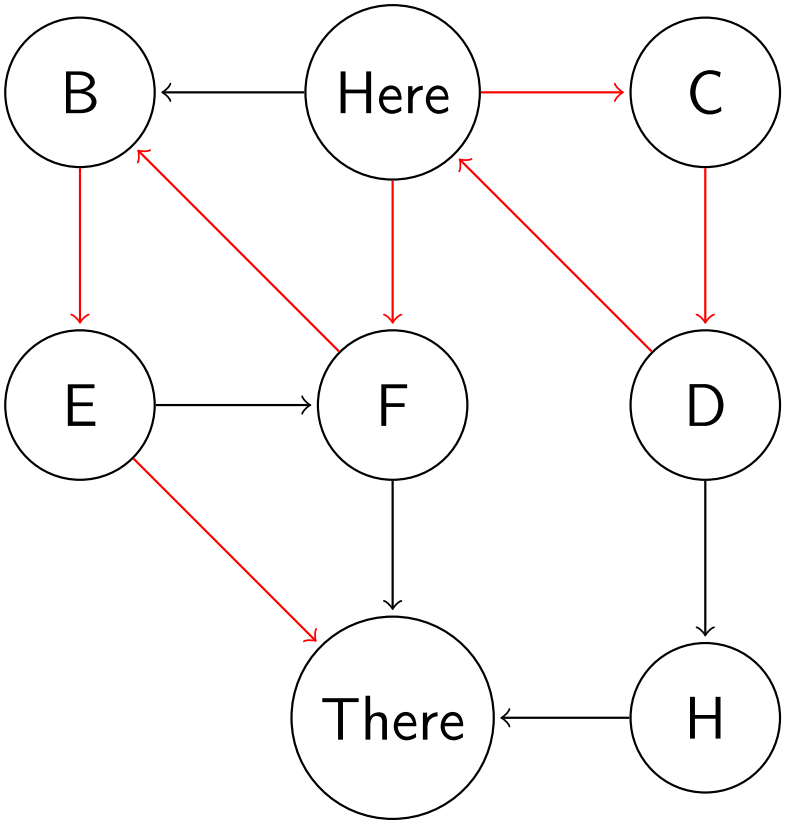
Is there a path from **Here** to **There**?

Path Querying



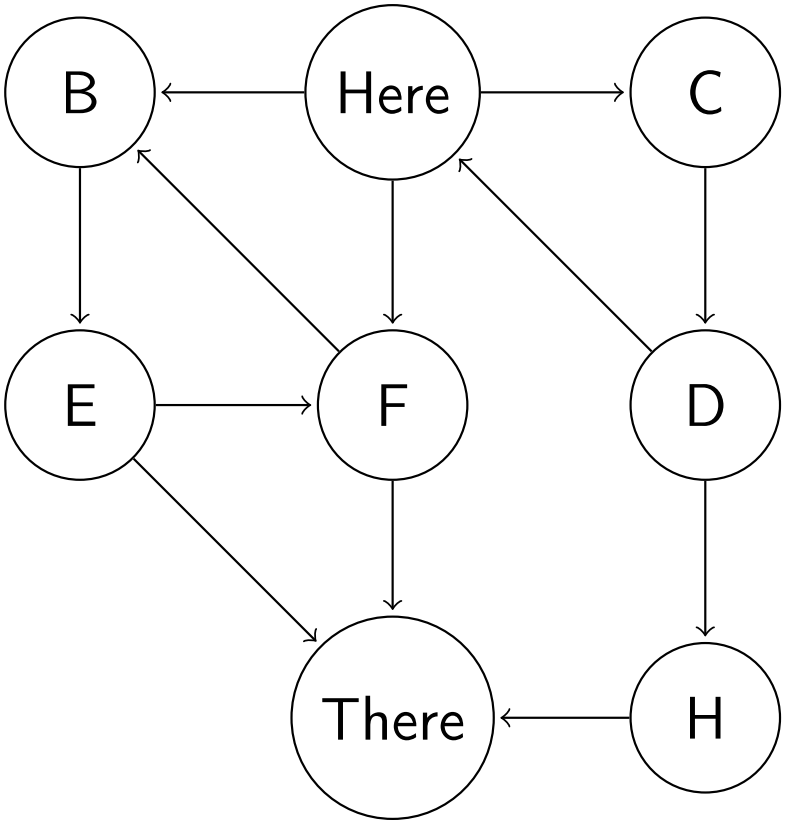
Is there a path from **Here** to **There** of length 13?

Path Querying



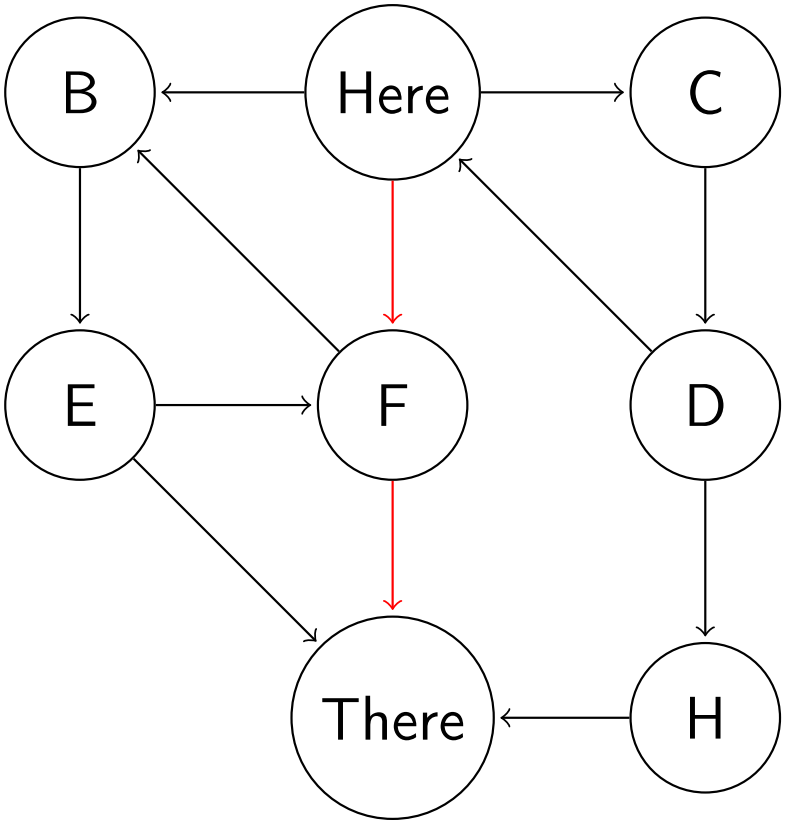
Is there a path from **Here** to **There** of length 13?

Path Querying



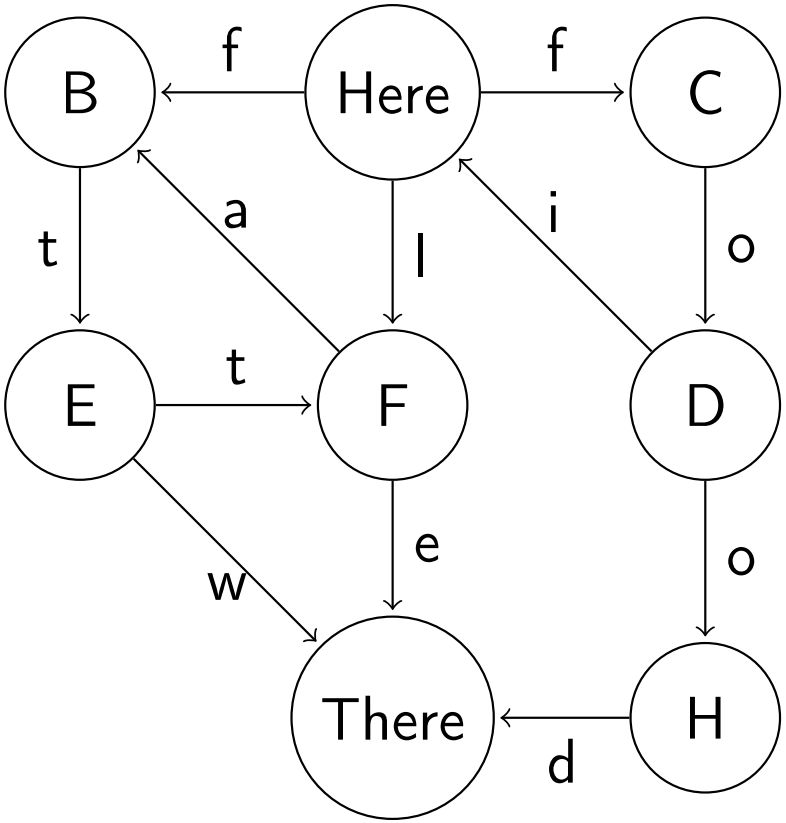
What is the shortest path from **Here** to **There**?

Path Querying

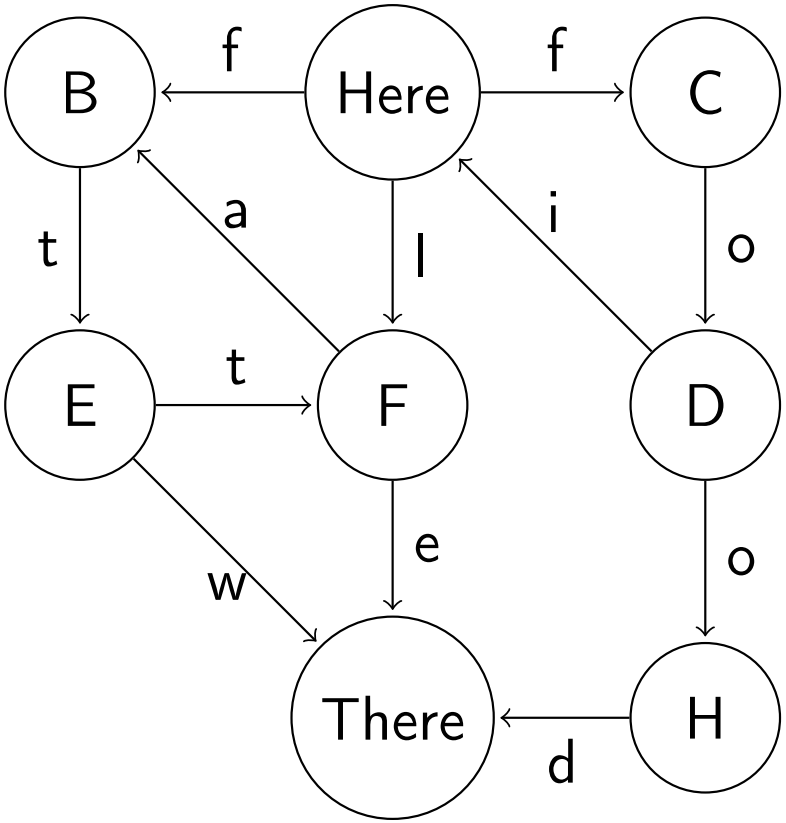


What is the shortest path from **Here** to **There**?

Path Querying: Regular Constraints

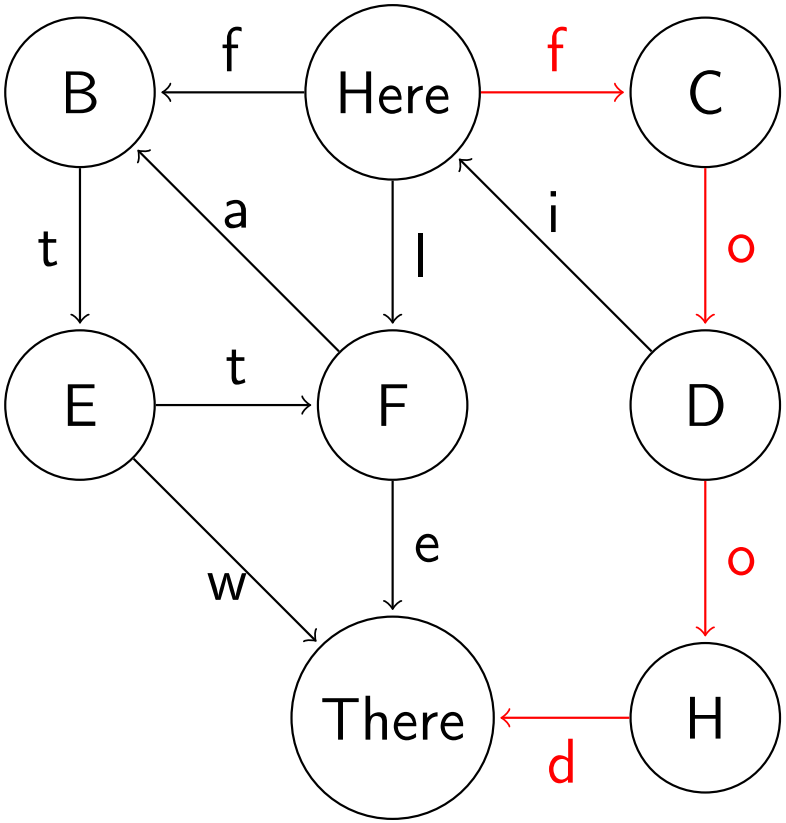


Path Querying: Regular Constraints



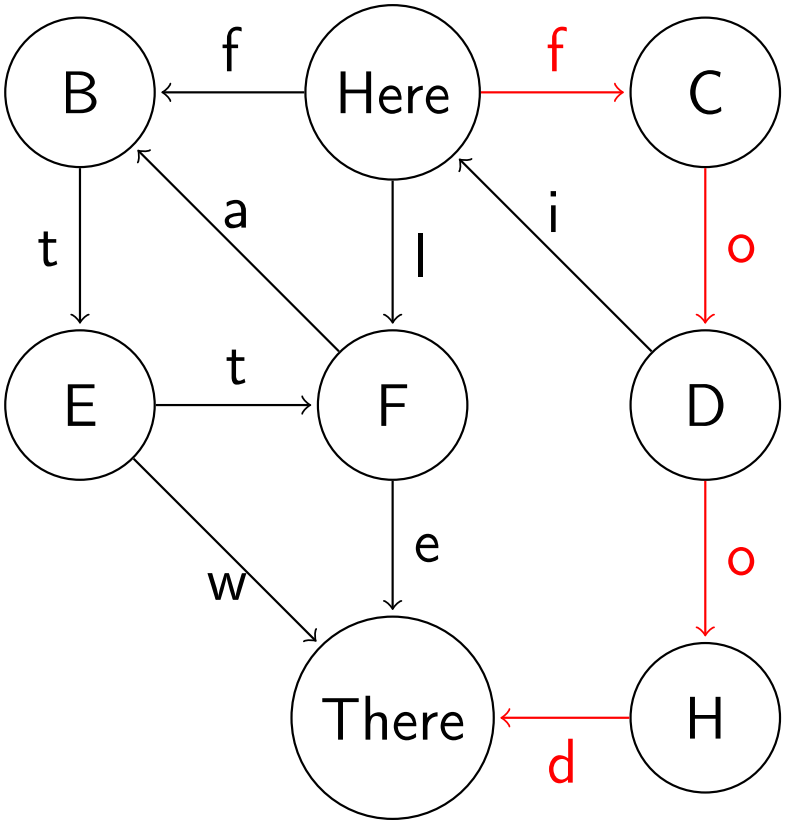
Is there a path from **Here** to **There** of form $f\ o^*(d \mid l)$?

Path Querying: Regular Constraints



Is there a path from **Here** to **There** of form $f\ o^*(d \mid l)$?

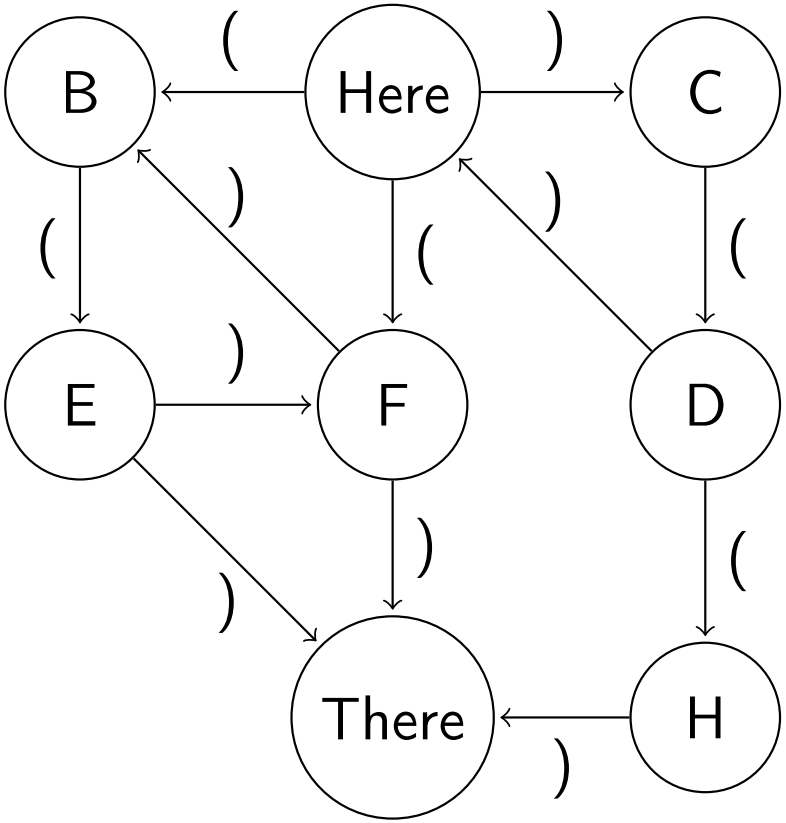
Path Querying: Regular Constraints



Is there a path from **Here** to **There** of form $f\ o^*(d \mid l)$?

Constraints are a regular language

Path Querying: Context-Free Constraints

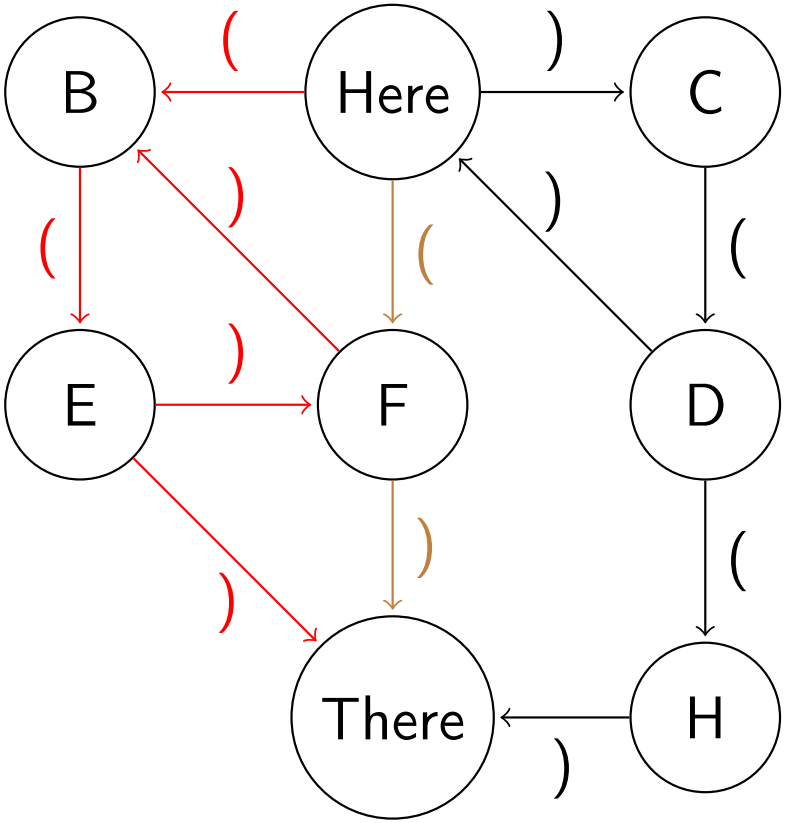


Is there a path from **Here** to **There** which is a string of balanced brackets?

Constraints are a context-free language given by the grammar:

$$S \to \varepsilon \mid (S)S$$

Path Querying: Context-Free Constraints



Is there a path from **Here** to **There** which is a string of balanced brackets?

Constraints are a context-free language given by the grammar:

$$S \to \varepsilon \mid (S)S$$

(())() () ()()

Context-Free Path Querying

Given:

- an input graph
- a context-free language specification

Construct the set of paths in the graph which are in the language

Applications of Context-Free Path Querying

- Querying graph databases
- Static code analysis
- Network analysis

Parsing for Path Querying

We need to parse strings along the paths

Can we generalize parsing to work on graphs instead of strings?

Parsing for Path Querying

We need to parse strings along the paths

Can we generalize parsing to work on graphs instead of strings?

Yes, we can!

Goals

Writing queries should be user friendly!
It should be liberated from the particularities of parsing

- Transparent integration of the query language
- Work with any context-free constraint
- Work with any input graph

Parser Combinators as a Way of Transparent Integration

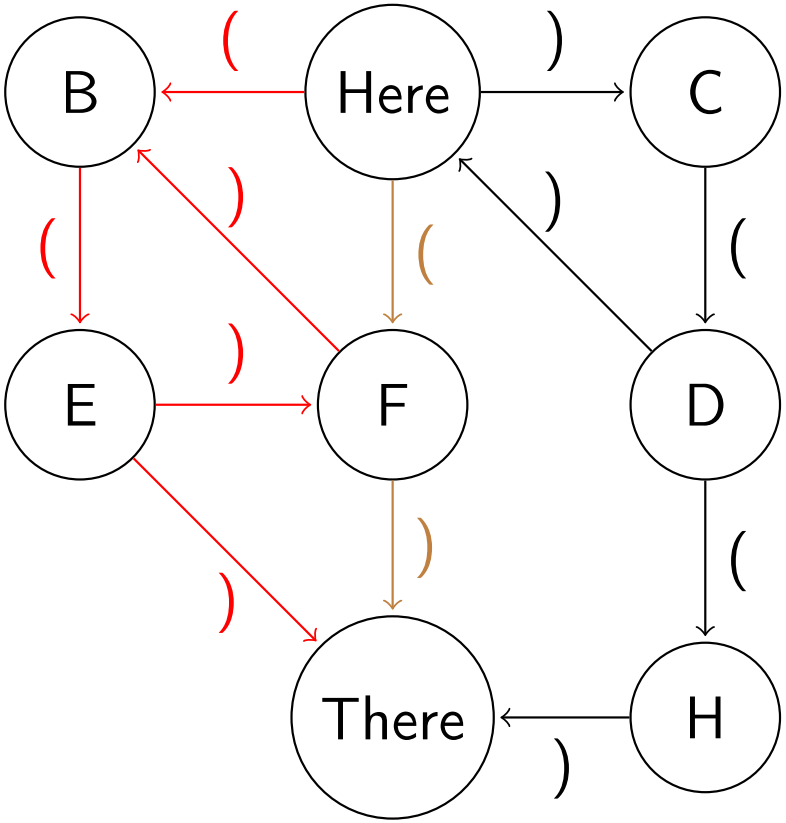
- Encode the query in the programm itself
- Types work for us: harder to write nonsensical grammar
- Composability
- Natural representation of the result

Meerkat

- Generic parser combinator library in Scala
- CPS-parsing with memoization
- Supports any context-free grammar (left recursive, ambiguous)
- Semantics calculation

```
val E: Nonterminal
= syn ( E ~ "^" ~ E
      | "_" ~ E
      | E ~ "*" ~ E
      | E ~ "/" ~ E
      | E ~ "+" ~ E
      | E ~ "_" ~ E
      | "(" ~ E ~ ")"
      | "[0-9]".r
      )
```

CFPQ in Meerkat



Is there a path from **Here** to **There** which is a string of balanced brackets?

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

(())() () ()()

Supported Combinators

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

Supported Combinators

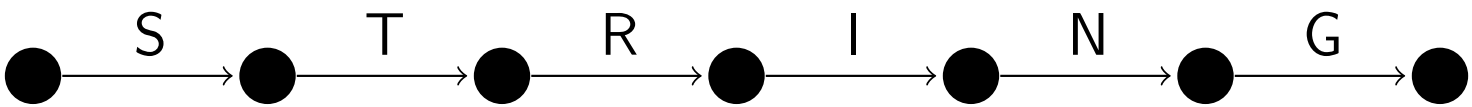
Combinator	Description
a ~ b	sequential parsing: a then b
a | b	choice: a or b
a ?	optional parsing: a or nothing
a *	repetition of zero or more a
a +	repetition of at least one a

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

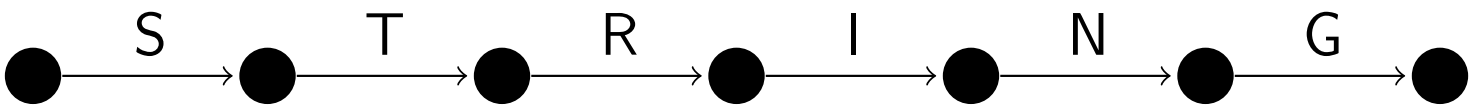
How is it Done? Input

String = linear graph



How is it Done? Input

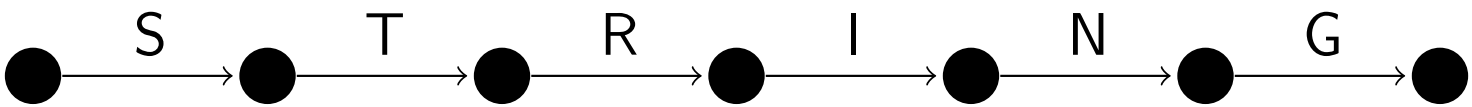
String = linear graph



What makes a graph **not** a string?

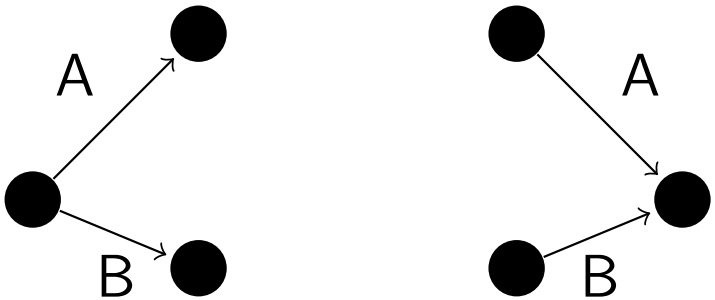
How is it Done? Input

String = linear graph



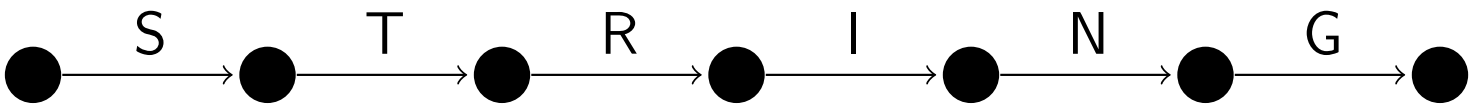
What makes a graph **not** a string?

Branches



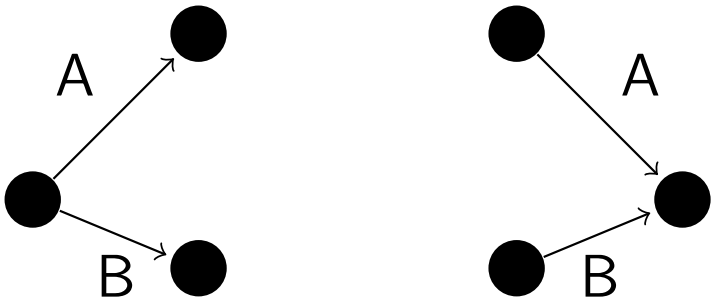
How is it Done? Input

String = linear graph

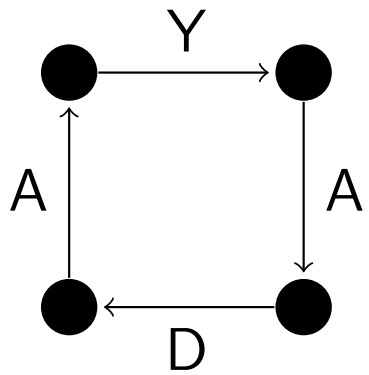


What makes a graph **not** a string?

Branches



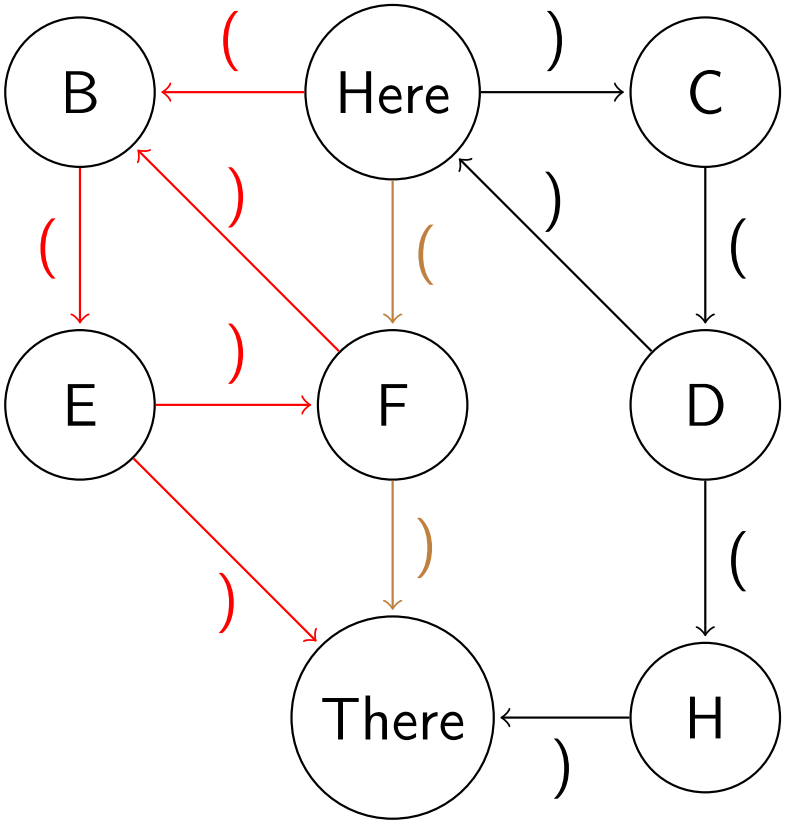
Cycles



How is it Done? General Idea

- Each input position corresponds to a set of intermidiate parsing results
- Continue parsing along each branch independently
- Merge the sets of intermidiate results whenever two paths lead to the same input position
- Continue parsing only from the **new** results
- Memoize the results obtained

Peeking into Vertices



Is there a path from **Here** to **There** which is a string of balanced brackets?

```
val Query : Nonterminal
= syn (LV("Here") ~ S ~
       LV("There"))

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

(())() () ()()

Input abstraction

```
trait Input[+E, +N] {
  def checkNode(nodeId: Int,
      predicate: N => Boolean): Option[N] }
  def filterEdges(nodeId: Int,
      predicate: E => Boolean): Seq [(E, Int)]
```

We provide the integration for GraphX and Neo4j

Example

```
val query =
  syn((
    syn(LV("Actor") ^^) ~ outLE("ACTS_IN") ~
    LV("Movie")) & (a => (a.name, a.toInt)))
executeQuery(query, input)
  .groupBy{ case(a, i) => i }
  .toIndexedSeq
  .map{ case(i, ms) => (ms.head._1, ms.length) }
  .sortBy{ case(a, mc) => -mc }}
  .take (10)
```

Evaluation

- Ontology querying
  - About 3 times faster than GLL-based solution
- Static code analysis: may-alias relations
  - Up to 5 times faster than Trails
- Movies database queries
  - Up to 10 times slower than Neo4j + Cypher
  - All queries are regular constraints

Limitations

- Overhead for the regular constraints
- Not exactly clear how to compute arbitrary semantics for the paths
  - Paths can be lazily extracted, but in what order?
  - Is it possible to compute some semantics in case of cycles?

Future work

- Reduce overhead for regular queries and improve performance
- Apply the library for static analysis of code
- Go beyond context-free constraints (ex. conjunctive grammars)

Summary

Parser combinators for CFPQ:

- Transparent integration of the query language
- Any context-free constraints
- Any input graph

Contacts

- Semyon Grigorev: s.v.grigoriev@spbu.ru
- Kate Verbitskaia: kajigor@gmail.com

- Meerkat for CFPQ: https://github.com/YaccConstructor/Meerkat