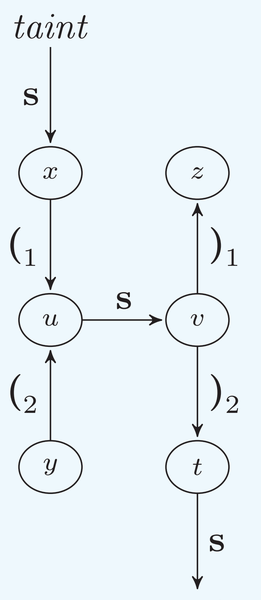
Graph Querying by Parsing

Kate Verbitskaia
JetBrains Research, Saint Petersburg State University, Russia
kajigor@gmail.com

Application

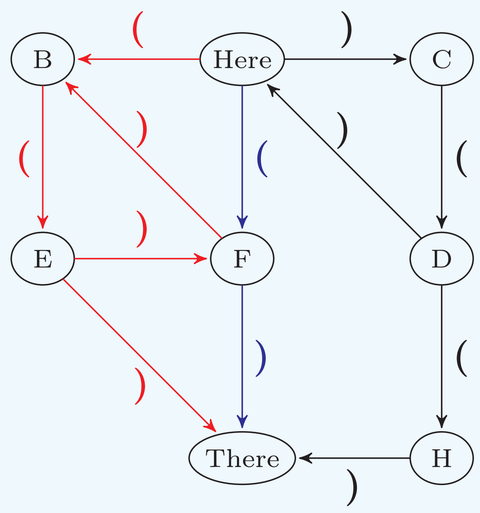
Static code analysis

```
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);
}
```



Problem

Find paths which satisfy constraints in form of a formal language



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

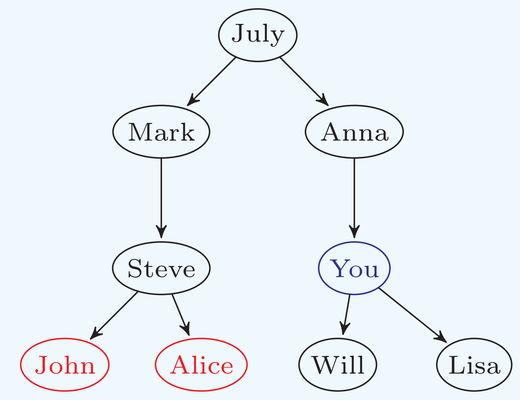
Constraints are context-free

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

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

Application

Querying of graph databases

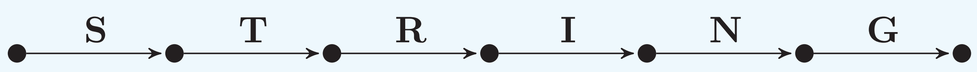


Find your cousins once removed

$$S \to H\downarrow$$
$$H \to \varepsilon \mid \uparrow H\downarrow$$

Solution: Idea

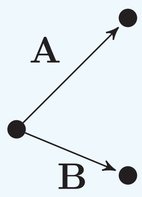
String = linear graph



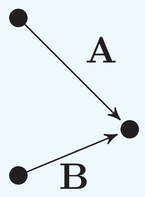
Parsing = checking if the (only) path satisfies the constraints

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

Branches

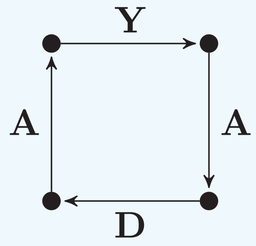


Parse along each branch



Merge the parsing results

Cycles

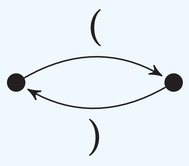


Make sure to not do the job twice

- 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

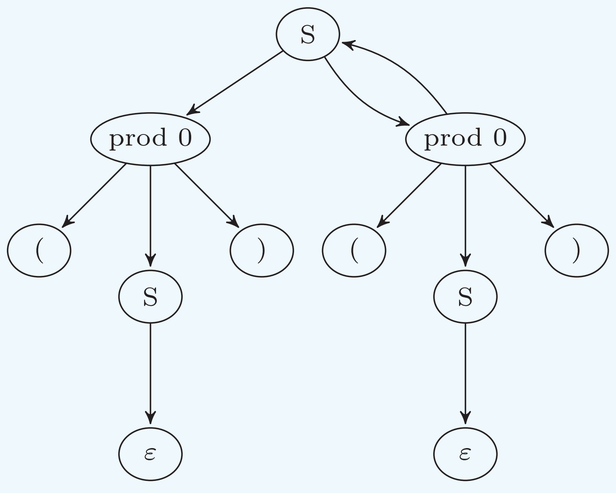
Generalized Parsing

Modification of the GLR and GLL algorithms



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

All derivation trees are constructed explicitly



Parser Combinators

CPS Parser Combinators with memoization

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

val S: Nonterminal
= syn(sameGen(List(("(", ")"))))

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

Common query patterns can be written as parser combinators and reused

Matrix Multiplication

Transitive closure of a special matrix

T — adjacency matrix
The grammar in the normal form

$$T_{ij} = \{N \mid N \overset{*}{\Rightarrow} \omega, \omega \text{ path bw } i \text{ and } j\}$$
$$T_{ik} \times T_{kj} = \{A \mid B \in T_{ik}, C \in T_{kj}, A \to BC\}$$
$$T^{(i)} = T^{(i-1)} \cup \left(T^{(i-1)} \times T^{(i-1)}\right)$$

$$\begin{pmatrix} \{S\} & \{A\} & \varnothing & \{B,S\} \\ \{S\} & \varnothing & \{A\} & \{S\} \\ \{A,S\} & \varnothing & \varnothing & \{S\} \\ \{B\} & \varnothing & \varnothing & \varnothing \end{pmatrix}$$

Easy to run in parallel environments:
GPUs, multithreaded CPUs, clusters

Any matrix multiplication library can be used

Contact us

Everything is available on GitHub:
https://github.com/YaccConstructor

Acknowledgments

The research is supported by the JetBrains Research grant and the Russian Science Foundation grant 18-11-00100.

References

[1] Ekaterina Verbitskaia, Semyon Grigorev, and Dmitry Avdyukhin. Relaxed parsing of regular approximations of string-embedded languages. In *Perspectives of System Informatics*, pages 291–302, 2016.

[2] Ekaterina Verbitskaia, Ilya Kirillov, Ilya Nozkin, and Semyon Grigorev. Parser combinators for context-free path querying. In *Proceedings of the 9th ACM SIGPLAN International Symposium on Scala*, Scala 2018, pages 13–23, 2018.

[3] Rustam Azimov and Semyon Grigorev. Context-free path querying by matrix multiplication. In *Proceedings of the 1st ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA)*, GRADES-NDA '18, pages 5:1–5:10, 2018.