**Regular Expressions**

- **Example:** `1-\d{3}-\d{3}-\d{4}` (US Phones)
- **Wide range of applications:** text and document stores, bioinformatics, data mining, etc.

**Existing Techniques**

- **Full Data Scans (NFA, DFA)**
  - Do not scale with data size
  - E.g., MongoDB, Oracle, MySQL, etc
- **m-gram Indexes with Partial Data Scans**
  - Index token of length m for multiple or all values of m
  - Avoid scans for indexed tokens.
  - Partial scans for tokens not indexed.
  - Suffer from large memory footprint
  - Can be used as a black box

**Black-box RegEx**

- Represent RegEx as an RTree
  - Leaves are tokens; Interior nodes are operators
- Search for leaves
- Traverse the RTree bottom-up, combining intermediate results at each node

**Succinct**

**Main Idea:** Transform the RTree for the RegEx such that:

- the black-box approach can be avoided for the Concat operator altogether
- Union, Repeat and Wildcard operators are pushed up the RTree

**Pull-Up Union Transformation:**

\[(RE|RE_1|RE_2) \vdash (RE_1|RE_1|RE_2)\]

**Pull-Up Wildcard Transformation:**

\[(RE,(RE_1|RE_2)) \vdash ((RE,RE_1)|RE_2)\]

**Pull-Out Repeat Transformation:**

Replace Repeat operator by Unions of Concat:

\[RE^* = (RE_1|RE_2|RE_3|...|RE_k)\]

- **k: smallest integer for which \(RE^{k+1}\) has 0 occurrences.
  - Use heuristic to upper bound value of k
  - k can be large when RegEx contains character classes
  - Use partial scans beyond threshold

**Pull-Out Concat Transformation:**

- Find Concat nodes whose children are tokens \(T_1, T_2\)
- Replace with new token \(T_1T_2\)

Transformations incorporated within Succinct data structures.

**Results**

Succinct has 8\times smaller storage footprint than uncompressed data structures

**Open Source Release**

- [https://github.com/amplab/succinct-cpp](https://github.com/amplab/succinct-cpp)
- For questions & feedback, contact us at:
  
  {anurag, ragarwal, istoica}@berkeley.edu