Reviewing Fast Congruence Closure Algorithms

Jose Abel Castellanos Joo

University of New Mexico

2017

The Algorithm

```
pending = {v ∈ V_1 | d(v) ≥ 1};
while pending ≠ do
    combine = ∅;
    for each v ∈ pending do
        if query(v) = Λ then
            enter(v);
        else
            add (v, query(v)) to combine;
        end
    end
    pending = ∅;
    for each (v, w) ∈ combine do
        if find(v) ≠ find(w) then
            if |list(find(v))| < |list(find(w))| then
                for each u ∈ list(find(v)) do
                    delete(u);
                    add u to pending;
                end
                union(find(w), find(v));
            else
                for each u ∈ list(find(w)) do
                    delete(u);
                    add u to pending;
                end
                union(find(v), find(w))
            end
```

Example

- $C_0 = \{\{a, b, f^3a, f^5a\}_1, \{fa\}_2, \{fb\}_3, \{f^2a\}_4, \{f^4a\}_5\}$
- $pending = \{fa, fb, f^2a, f^3a, f^4a, f^5a\}$
- $list(1) = \{fa, fb, f^4a\}$; $list(2) = \{f^2a\}$; $list(3) = \{\}$; $list(4) = \{f^3a\}$; $list(5) = \{f^5a\}$
- $combine = \{\}$
- $sig\_table = \{(fa, (1))\}$
- $combine = \{(fa, fb)\}$
- $sig\_table = \{(fa, (1)), (f^2a, (2))\}$
- $sig\_table = \{(fa, (1)), (f^2a, (2)), (f^3a, (4))\}$
- $combine = \{(fa, fb), (fa, f^4a)\}$
- $sig\_table = \{(fa, (1)), (f^2a, (2)), (f^3a, (4)), (f^5a, (5))\}$

Example (Contd.)

- $pending = \{\}$
- $C_1 = \{\{a, b, f^3a, f^5a\}_1, \{fa, fb\}_2, \{f^2a\}_4, \{f^4a\}_5\}$
- $sig\_table = \{(fa, (1)), (f^2a, (2)), (f^3a, (4))\}$
- $pending = \{f^5a\}$
- $C_2 = \{\{a, b, f^3a, f^5a\}_1, \{fa, fb, f^4a\}_2, \{f^2a\}_4\}$
- $list(1) = \{fa, fb, f^4a\}$; $list(2) = \{f^2a, f^5a\}$; $list(3) = \{\}$; $list(4) = \{f^3a\}$; $list(5) = \{\}$
- $combine = \{\}$
- $combine = \{(f^2a, f^5a)\}$
- $pending = \{\}$
- $sig\_table = \{(fa, (1)), (f^2a, (2))\}$
- $pending = \{f^3a\}$
- $C_3 = \{\{a, b, f^3a, f^5a, f^2a\}_1, \{fa, fb, f^4a\}_2\}$

Example (Contd.)

- $list(1) = \{fa, fb, f^4a, f^3a\}$; $list(2) = \{f^2a, f^5a\}$; $list(3) = \{\}$; $list(4) = \{\}$; $list(5) = \{\}$
- $combine = \{\}$
- $combine = \{(f^3a, fa)\}$
- $pending = \{\}$
- $sig\_table = \{(fa, (1))\}$
- $pending = \{f^2a, f^5a\}$
- $C_4 = \{\{a, b, f^3a, f^5a, fa, fb, f^2a\, f^4a\}_1\}$
- $combine = \{\}$
- $combine = \{(f^2a, fa)\}$
- $combine = \{(f^2a, fa), (f^5a, fa)\}$
- $pending = \{\}$
- $halt$

Running Time

- Operations on pending:
  - There are at most $|V_1|$ initial additions to pending (initial number of equivalence classes)
  - There are at most $2|V_1|$ elements in total in all *list*
  - Using the weighted heuristic, the length of one of the predecessor lists at least doubles after merging two equivalence classes
  - Thus, there are at most $|V_1| + 2|V_1| \log(2|V_1|)$ for pending

Running Time (Contd.)

- Operations on the UNION-FIND data structure:
  - Union operations: There are at most $|V_1| - 1$ *union* operations
  - The amount of *list* operations is bounded by a constant times the number of *union* operations (i.e. $O(|V_1|) = O(m)$)
  - The number of operations on the set *combine* is bounded by the number of additions to *pending*
  - The number of *find* operations is bounded by a constant times the number of *combine* operations
- Hence,the number of *find* operations is $O(m \log m)$
- Using the set-union algorithm, *union* operations take $O(1)(Amortized)$
- Similarly, *list* operations take also $O(1)$ adding a circularly linked list of predecessors.
- Therefore, the total work for *find* operations require $O(m \log m)$ time.

Running Time (Contd.)

- Operations on the Signature Table:
- Bounded by number of additions to *pending* (i.e. $O(m \log m)$)
  - Unary Functions (Requires to store one item per term): We can use an arrat to store the signature of each term. Thus, each operation of this part of the table takes $O(1)$ time
  - Binary Functions (Requires to store two items per term):

Running Time (Contd.)

- Balance binary tree:
  - All operations take $O(\log m)$ time
  - Total time needed: $O(m(\log m)^2)$
  - Space needed: $O(m)$
- $n \times O(m)$ array:
  - All operations take $O(1)$ time
  - Total time needed: $O(m \log m)$
  - Space needed: $O(mn)$
- Hash table:
  - All operations take $O(1)$ time on average
  - Total time needed (on average): $O(m \log m)$
  - Space needed: $O(m)$