How fast can we count solutions?

Theory:
• Courcelle (1990): For problems that are expressible in monadic second-order logic (MSO), we can decide in linear time whether a solution exists, if the problem instances have bounded treewidth.
• Amborg et al. (1991): The same holds for counting the number of solutions.

Practice:
Problems given by their MSO description can be transformed into a tree language recognition problem, which is then solved via a finite tree automaton (FTA).

Problem:
• For algorithms based on the transformation above, even very simple MSO formulas quickly lead to a „state explosion“ of the FTA.
• This renders those algorithms useless for practical applications.
• Moreover formulating an appropriate tree language recognition problem by hand is highly nontrivial.

Solution:
• We presented a new approach using quasi-guarded datalog programs.
• This method is based on the work of Gottlob et al. (2007) for solving decision problems.

Quasi-guarded datalog approach
1 Create a tree decomposition of the problem instance.
2 Construct a quasi-guarded datalog program based on the MSO description of the problem.
3 Evaluate the quasi-guarded datalog program on the tree decomposition in linear time.
4 Approach was successfully applied for solving the problems:
  #Sat ... Counting all models of a propositional formula.
  #Circumscripton ... Counting the (subset) minimal models of a propositional formula.
  #Horn-abduction ... Counting all solutions of an abduction problem given by Horn clauses.

Normalized tree decomposition

Problem instance for #Sat
\( (x_1 \lor \lnot x_2 \lor x_3) \land (\lnot x_1 \lor x_4 \lor \lnot x_5) \land (x_2 \lor \lnot x_4 \lor x_6) \)

Quasi-guarded program for #Sat (excerpt)

```
/* variable introduction node. */

sat(v, P \{ x \}, N, C_n, SUM(j)) ← bag(v, X \{ x \}, C_n), child_1(v_1, v), bag(v_1, X, C_n),
    sat(v_1, P, N, C_1, j), true((x), C_2, C_n),
    C_1 \cup C_2 = C_n.

/* clause introduction node. */

sat(v, P, N, C_n, j) ← bag(v, X, C \{ x \}), child_1(v_1, v), bag(v_1, X, C_n),
    sat(v_1, P, N, C_1, j), true(P, N, C_2, \{ i \}), C_1 \cup C_2 = C_n.

/* branch node. */

sat(v, P, N, C_n, SUM(j)) ← child_3(v_1, v), bag(v_1, X, C_n), sat(v_1, P, N, C_1, j_1),
    child_2(v_2, v), bag(v_2, X, C_n), sat(v_2, P, N, C_2, j_2),
    bag(v, X, C_n), C_1 \cup C_2 = C_w, j_1 + j_2 = j.
```

Results

Theory:
• We extended datalog by some arithmetic constructs.
• The adaption of the concept of quasi-guarded rules to our extension of datalog yields a fragment which is evaluable in linear time.
• Each MSO problem description can be transformed into an extended datalog program whose size depends only on the treewidth and the size of the MSO formula.
  → Alternative proof for counting variant of Courcelle's theorem.

Practice:
• We applied our approach to solve the three counting problems #Sat, #Circumscripton and #Horn-abduction.
• The algorithm for #Sat was already implemented.