

Exploiting Bounded Treewidth with Datalog¹

Reinhard Pichler

Institut für Informationssysteme
Arbeitsbereich DBAI
Technische Universität Wien

Datalog 2.0, Oxford



¹Joint work with Georg Gottlob and Fang Wei

Outline

1. Motivation
2. Treewidth of Finite Structures
3. MSO and Courcelle's Theorem
4. Expressive Power of Monadic Datalog
5. Efficient Computation with Monadic Datalog
6. Conclusion

Outline

1. Motivation
2. Treewidth of Finite Structures
3. MSO and Courcelle's Theorem
4. Expressive Power of Monadic Datalog
5. Efficient Computation with Monadic Datalog
6. Conclusion

Motivation

Monadic second-order logic and bounded treewidth

- Monadic-second order logic (MSO): first-order logic + set variables
- MSO is a powerful query language (in general: intractable).
- Courcelle's Theorem: Evaluation of MSO becomes tractable in case of bounded treewidth.
- Algorithms via Courcelle's Theorem are problematical.

Motivation

Monadic second-order logic and bounded treewidth

- Monadic-second order logic (MSO): first-order logic + set variables
- MSO is a powerful query language (in general: intractable).
- Courcelle's Theorem: Evaluation of MSO becomes tractable in case of bounded treewidth.
- Algorithms via Courcelle's Theorem are problematical.

Monadic second-order logic vs. monadic datalog

- Efficient evaluation of monadic datalog
- On trees, MSO and monadic datalog have exactly the same expressive power (Gottlob/Koch, 2002).
- What about monadic datalog over "tree-like" structures?

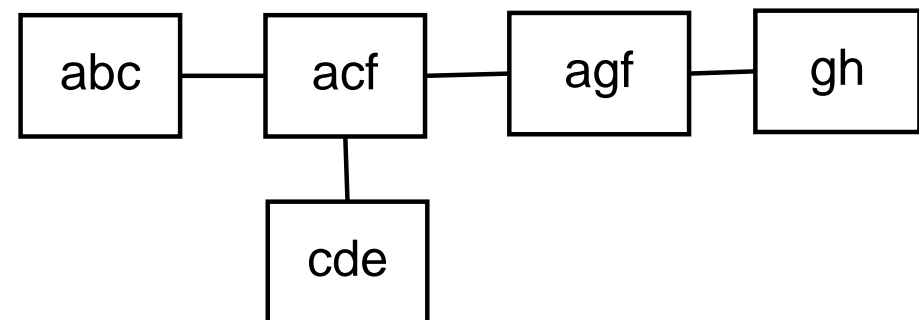
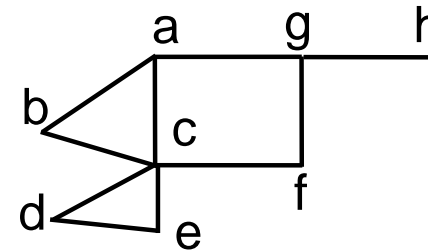
Outline

1. Motivation
2. Treewidth of Finite Structures
3. MSO and Courcelle's Theorem
4. Expressive Power of Monadic Datalog
5. Efficient Computation with Monadic Datalog
6. Conclusion

Tree Decomposition

Tree Decomposition of a Graph

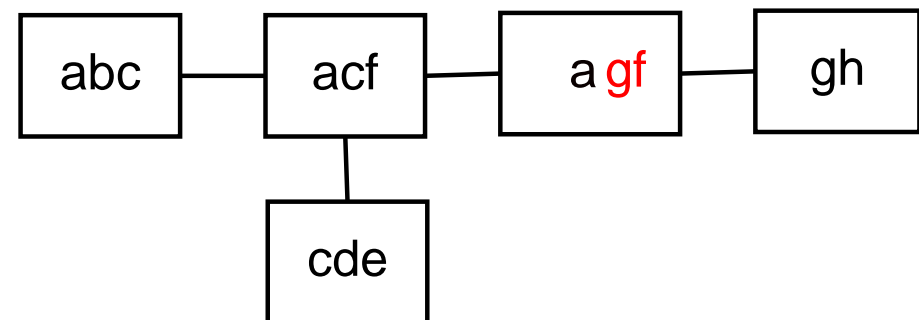
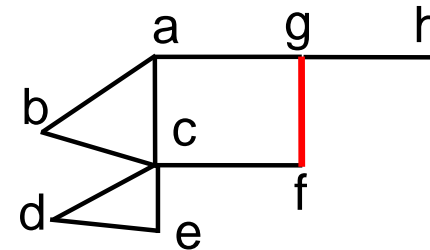
- 1 Tree with a vertex set (= “bag”) associated with every node.
- 2 For every edge (v, w) : there is a bag containing both v and w .
- 3 For every v : the nodes that contain v form a connected subtree.



Tree Decomposition

Tree Decomposition of a Graph

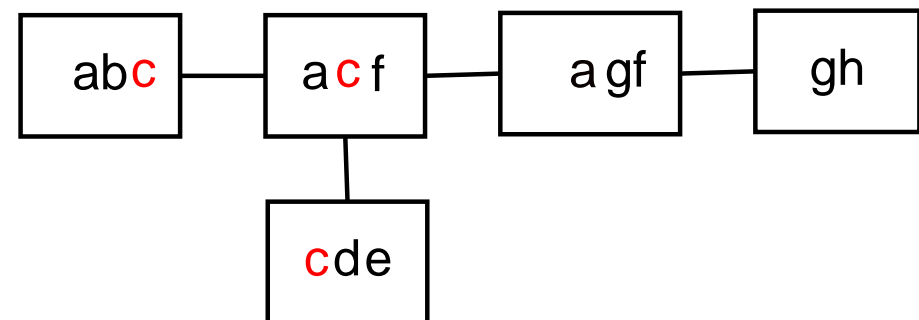
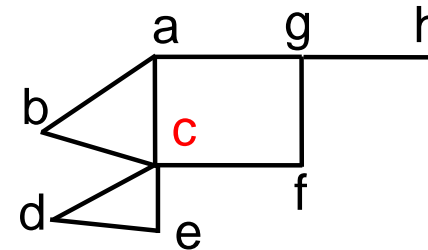
- 1 Tree with a vertex set (= “bag”) associated with every node.
- 2 For every edge (v, w) : there is a bag containing both v and w .
- 3 For every v : the nodes that contain v form a connected subtree.



Tree Decomposition

Tree Decomposition of a Graph

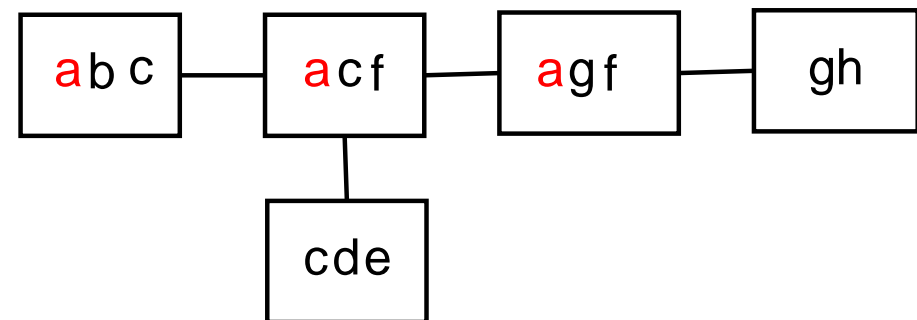
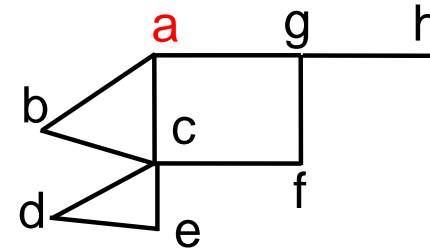
- 1 Tree with a vertex set (= “bag”) associated with every node.
- 2 For every edge (v, w) : there is a bag containing both v and w .
- 3 For every v : the nodes that contain v form a connected subtree.



Tree Decomposition

Tree Decomposition of a Graph

- 1 Tree with a vertex set (= “bag”) associated with every node.
- 2 For every edge (v, w) : there is a bag containing both v and w .
- 3 For every v : the nodes that contain v form a connected subtree.



Tree Decomposition

Tree Decomposition of a Graph

- 1 Tree with a vertex set (= “bag”) associated with every node.
- 2 For every edge (v, w) : there is a bag containing both v and w .
- 3 For every v : the nodes that contain v form a connected subtree.

Tree Decomposition

Tree Decomposition of a Graph

- 1 Tree with a vertex set (= “bag”) associated with every node.
- 2 For every edge (v, w) : there is a bag containing both v and w .
- 3 For every v : the nodes that contain v form a connected subtree.

Tree Decomposition of a Structure

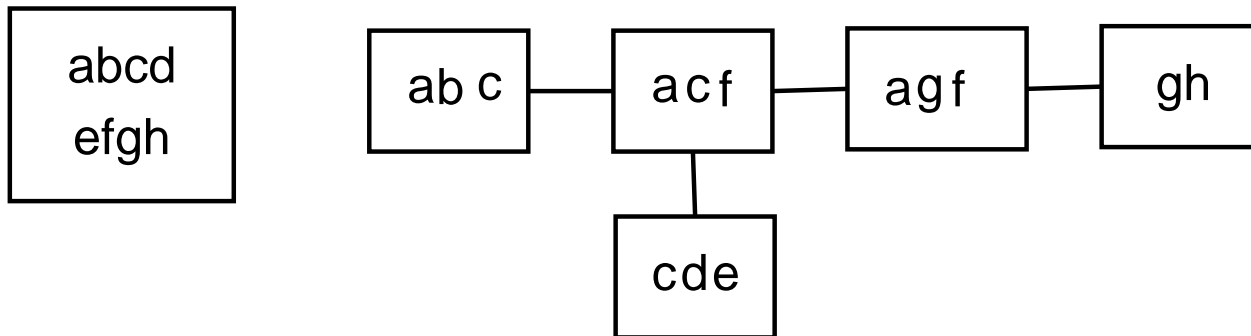
- 1 Tree with a **set of domain elements** (= “bag”) associated with every node.
- 2 **For every tuple (a_1, \dots, a_k) in any relation R_i : there is a bag containing $\{a_1, \dots, a_k\}$.**
- 3 For every a : the nodes that contain a form a connected subtree.

Treewidth

- The **width** of a tree decomposition $\langle T, (A_t)_{t \in T} \rangle$ is $\max(\{|A_t| \mid t \in T\}) - 1$, i.e., **max bag size - 1**.
- The **treewidth** $tw(\mathcal{G})$ is the minimum width over all tree decompositions of \mathcal{G} .

Treewidth

- The **width** of a tree decomposition $\langle T, (A_t)_{t \in T} \rangle$ is $\max(\{|A_t| \mid t \in T\}) - 1$, i.e., **max bag size - 1**.
- The **treewidth** $tw(\mathcal{G})$ is the minimum width over all tree decompositions of \mathcal{G} .



Treewidth of a Propositional Formula in CNF

Represent a CNF formula as a finite structure

Given a propositional formula F in CNF, represent F by a finite structure $A(F)$ over the signature τ with $\tau = \{cl, var, pos, neg\}$, where

- $cl(c), var(x)$
means that c is a clause (resp. x is a variable) in F .
- $pos(x, c), neg(x, c)$
means that x occurs unnegated (resp. negated) in the clause c .

Treewidth of a CNF formula

We define $tw(F) := tw(A(F))$.

Treewidth of CNF

Example

Given a propositional formula F in CNF

$$(x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_2 \vee \bar{x}_3 \vee x_4),$$

represent F by a finite structure $A(F)$ over τ :

Treewidth of CNF

Example

Given a propositional formula F in CNF

$$(x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_2 \vee \bar{x}_3 \vee x_4),$$

represent F by a finite structure $A(F)$ over τ :

$A(F)$ contains the following atoms:

- $cl(c_1), cl(c_2),$
- $var(x_1), var(x_2), var(x_3), var(x_4),$
- $pos(x_1, c_1), pos(x_3, c_1), pos(x_2, c_2), pos(x_4, c_2),$
- $neg(x_2, c_1), neg(x_3, c_2).$

Tree Decomposition

 $A(F)$

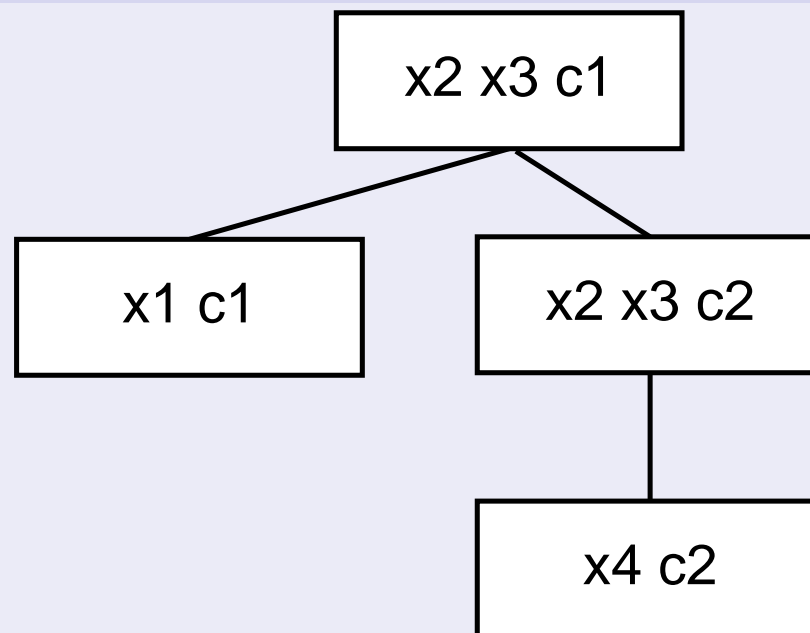
$cl(c_1), cl(c_2),$
 $var(x_1), var(x_2),$
 $var(x_3), var(x_4),$
 $pos(x_1, c_1),$
 $pos(x_3, c_1),$
 $pos(x_2, c_2),$
 $pos(x_4, c_2),$
 $neg(x_2, c_1),$
 $neg(x_3, c_2).$

Tree Decomposition

$A(F)$

$cl(c_1), cl(c_2),$
 $var(x_1), var(x_2),$
 $var(x_3), var(x_4),$
 $pos(x_1, c_1),$
 $pos(x_3, c_1),$
 $pos(x_2, c_2),$
 $pos(x_4, c_2),$
 $neg(x_2, c_1),$
 $neg(x_3, c_2).$

Tree Decomposition of $A(F)$



Outline

1. Motivation
2. Treewidth of Finite Structures
3. MSO and Courcelle's Theorem
4. Expressive Power of Monadic Datalog
5. Efficient Computation with Monadic Datalog
6. Conclusion

Expressive Power of MSO

Many interesting (intractable) properties of graphs or of finite structures are expressible in MSO, e.g.:

- Graph Problems: 3-Colorability
- Database design problems: primality, subschema-BCNF, 3NF
- KR & R: SAT, Propositional Abduction, CWR

Expressive Power of MSO

Many interesting (intractable) properties of graphs or of finite structures are expressible in MSO, e.g.:

- Graph Problems: 3-Colorability
- Database design problems: primality, subschema-BCNF, 3NF
- KR & R: SAT, Propositional Abduction, CWR

Theorem (Courcelle, 1990)

Any property of finite structures, which is expressible by an MSO sentence, can be decided in linear time (data complexity) if the structures have bounded treewidth.

MSO-Example: SAT-Problem

MSO-Encoding of the SAT-Problem

Idea. Let F be a propositional formula in CNF; an interpretation of F can be represented as a set X of variables (i.e., the variables which are true).

MSO-Example: SAT-Problem

MSO-Encoding of the SAT-Problem

Idea. Let F be a propositional formula in CNF; an interpretation of F can be represented as a set X of variables (i.e., the variables which are true).

MSO encoding of $X \models F$:

$$(\forall c)cl(c) \rightarrow (\exists z)[(pos(z, c) \wedge z \in X) \vee (neg(z, c) \wedge z \notin X)]$$

MSO-Example: SAT-Problem

MSO-Encoding of the SAT-Problem

Idea. Let F be a propositional formula in CNF; an interpretation of F can be represented as a set X of variables (i.e., the variables which are true).

MSO encoding of $X \models F$:

$$(\forall c)cl(c) \rightarrow (\exists z)[(pos(z, c) \wedge z \in X) \vee (neg(z, c) \wedge z \notin X)]$$

MSO encoding of the **SAT-Problem**:

$$(\exists X)X \models F$$

Theoretical Tractability vs. Efficient Computation

Algorithms via Courcelle's Theorem

In principle, Courcelle's theorem can be used to effectively generate a concrete algorithm from an MSO description, see e.g. (Arnborg/Lagergren/Seese, 1991), (Flum/Frick/Grohe, 2002).

- 1** Translate the MSO evaluation problem over finite structures into an equivalent MSO evaluation problem over colored binary trees.
- 2** Solve this problem via the correspondence between MSO over trees and finite tree automata (FTA), see (Thatcher/Wright, 1968), (Doner, 1970).

Theoretical Tractability vs. Efficient Computation

Problems with this approach

- “State explosion” of the FTA even for relatively simple MSO formulae *on trees*.
- MSO formula over colored binary trees is significantly more complex than the original formula for the structure with bounded treewidth.

Theoretical Tractability vs. Efficient Computation

Problems with this approach

- “State explosion” of the FTA even for relatively simple MSO formulae *on trees*.
- MSO formula over colored binary trees is significantly more complex than the original formula for the structure with bounded treewidth.

Conclusion, see also (Grohe, 1999)

- Main benefit of Courcelle's Theorem: “a simple way to recognize a property as being linear time computable”.
- Algorithms are “useless for practical applications”.

Outline

1. Motivation
2. Treewidth of Finite Structures
3. MSO and Courcelle's Theorem
- 4. Expressive Power of Monadic Datalog**
5. Efficient Computation with Monadic Datalog
6. Conclusion

Monadic Datalog

Definition

- **Monadic datalog.** All intensional predicates are monadic.
- **Guarded fragment.** Every rule r contains an extensional atom B , s.t. every variable occurring in r occurs in B .

Monadic Datalog

Definition

- **Monadic datalog.** All intensional predicates are monadic.
- **Guarded fragment.** Every rule r contains an extensional atom B , s.t. every variable occurring in r occurs in B .
- **Quasi-guarded fragment.** Every rule r contains an extensional atom B , s.t. every variable occurring in r
 - either occurs in B
 - or is functionally dependent on B .

Signature Extension for the Tree Decomposition

Definition

Let $\tau = \{R_1, \dots, R_K\}$ be a set of predicate symbols and let $w \geq 1$ denote the treewidth. Then we define the following extended signature

$$\tau_{td} = \tau \cup \{root, leaf, child_1, child_2, bag\}$$

for unary predicates *root*, *leaf* and binary predicates *child*₁ and *child*₂. Moreover *bag* has arity $k + 2$ with $k \leq w$, where $bag(t, a_0, \dots, a_k)$ means that the bag at node t is (a_0, \dots, a_k) .

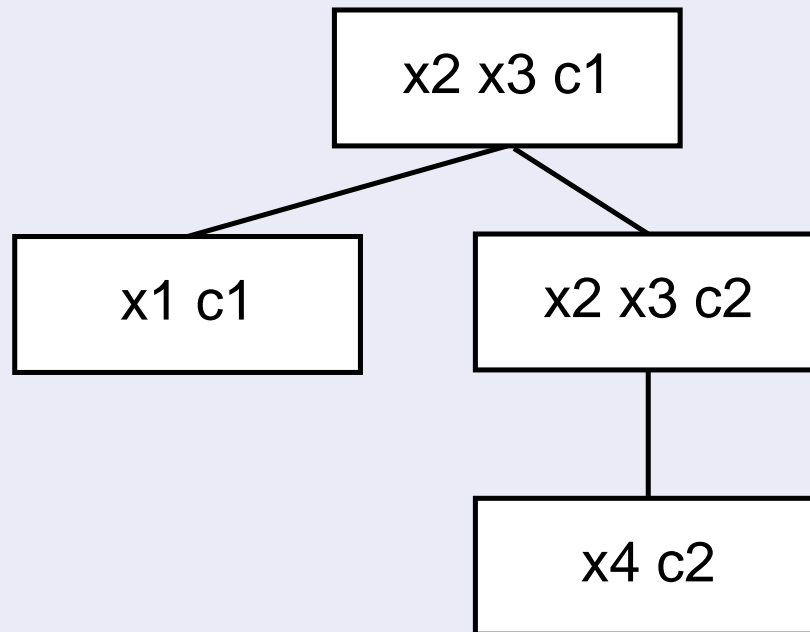
Example

Recall the CNF formula $F = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_2 \vee \bar{x}_3 \vee x_4)$

Example

Recall the CNF formula $F = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_2 \vee \bar{x}_3 \vee x_4)$

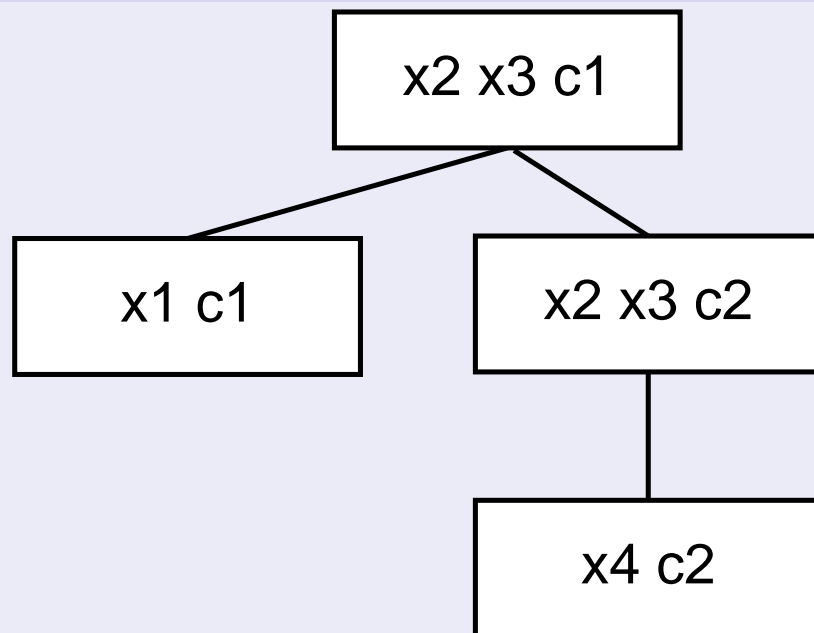
Tree Decomposition of $A(F)$



Example

Recall the CNF formula $F = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_2 \vee \bar{x}_3 \vee x_4)$

Tree Decomposition of $A(F)$



\mathcal{T}_{td} -structure

$root(t_1),$
 $child_1(t_2, t_1),$
 $child_2(t_3, t_1),$
 $child_1(t_4, t_3),$
 $leaf(t_2), leaf(t_4),$
 $bag(t_1, x_2, x_3, c_1),$
 $bag(t_2, x_1, c_1),$
 $bag(t_3, x_2, x_3, c_2),$
 $bag(t_4, x_4, c_2).$

Expressive Power of Monadic Datalog

Theorem

Let τ and $w \geq 1$ be arbitrary but fixed. Every MSO-definable unary query over τ -structures of treewidth w is also definable in the quasi-guarded fragment of monadic datalog over τ_{td} .

Expressive Power of Monadic Datalog

Theorem

Let τ and $w \geq 1$ be arbitrary but fixed. Every MSO-definable unary query over τ -structures of treewidth w is also definable in the quasi-guarded fragment of monadic datalog over τ_{td} .

Theorem

*Let P be a quasi-guarded monadic datalog program and let \mathcal{A} be a finite structure. Then P can be evaluated over \mathcal{A} in time $\mathcal{O}(|P| * |\mathcal{A}|)$.*

Proof Idea

Main steps

- Define an appropriate normal form on tree decompositions.
- Prove the MSO-equivalence of certain substructures induced by subtrees of a tree decomposition – thus generalizing results from (Neven/Schwentick, 2002).
- Construct a monadic datalog program P equivalent to a given unary MSO-query $\varphi(x)$ with some quantifier depth k .

Proof Idea

Main steps

- Define an appropriate normal form on tree decompositions.
- Prove the MSO-equivalence of certain substructures induced by subtrees of a tree decomposition – thus generalizing results from (Neven/Schwentick, 2002).
- Construct a monadic datalog program P equivalent to a given unary MSO-query $\varphi(x)$ with some quantifier depth k .

Remark

- The domain of P consists of the “original” domain elements a_i and the nodes s_j of the tree decomposition.
- Intensional predicates correspond to “ k -types”.

Outline

1. Motivation
2. Treewidth of Finite Structures
3. MSO and Courcelle's Theorem
4. Expressive Power of Monadic Datalog
5. Efficient Computation with Monadic Datalog
6. Conclusion

Putting the Monadic Datalog Approach to Work

Based on the monadic datalog idea, we have developed new algorithms for several problems in case of bounded treewidth, e.g.:

- 3-colorability problem: (extended) datalog program with 9 rules.
- SAT problem: (extended) datalog program containing 9 rules.
- Abduction Solvability problem and Relevance problem: (extended) datalog program containing < 20 rules.

Example: SAT Program

Program SAT

```

/* leaf node. */
solve(s, P, N, C1) ← leaf(s), bag(s, X, C),
    P ∪ N = X, P ∩ N = ∅, true(P, N, C1, C).

/* variable removal node. */
solve(s, P, N, C1) ← bag(s, X, C), child1(s1, s),
    bag(s1, X ⊕ {x}, C), solve(s1, P ⊕ {x}, N, C1).
solve(s, P, N, C1) ← bag(s, X, C), child1(s1, s),
    bag(s1, X ⊕ {x}, C), solve(s1, P, N ⊕ {x}, C1).

/* clause removal node. */
solve(s, P, N, C1) ← bag(s, X, C), child1(s1, s),
    bag(s1, X, C ⊕ {c}), solve(s1, P, N, C1 ⊕ {c}).

```

Example: SAT Program

Program SAT (continued)

```

/* variable introduction node. */
solve( $s, P \uplus \{x\}, N, C_1 \cup C_2$ )  $\leftarrow$  bag( $s, X \uplus \{x\}, C$ ), child1( $s_1, s$ ),
    bag( $s_1, X, C$ ), solve( $s_1, P, N, C_1$ ), true( $\{x\}, \emptyset, C_2, C$ ).
solve( $s, P, N \uplus \{x\}, C_1 \cup C_2$ )  $\leftarrow$  bag( $s, X \uplus \{x\}, C$ ), child1( $s_1, s$ ),
    bag( $s_1, X, C$ ), solve( $s_1, P, N, C_1$ ), true( $\emptyset, \{x\}, C_2, C$ ).

/* clause introduction node. */
solve( $s, P, N, C_1 \cup C_2$ )  $\leftarrow$  bag( $s, X, C \uplus \{c\}$ ), child1( $s_1, s$ ),
    bag( $s_1, X, C$ ), solve( $s_1, P, N, C_1$ ), true( $P, N, C_2, \{c\}$ ).

/* branch node. */
solve( $s, P, N, C_1 \cup C_2$ )  $\leftarrow$  bag( $s, X, C$ ), child1( $s_1, s$ ), child2( $s_2, s$ ),
    bag( $s_1, X, C$ ), bag( $s_2, X, C$ ), solve( $s_1, P, N, C_1$ ), solve( $s_2, P, N, C_2$ ).

/* result (at the root node). */
success  $\leftarrow$  root( $s$ ), bag( $s, X, C$ ), solve( $s, P, N, C$ ).

```

Implementation using DLV

System components

- 1 **Parser** transforms CNF formula into finite structure \mathcal{A} (i.e., set of atoms over the signature $\tau = \{cl, var, pos, neg\}$).
- 2 **Tree decomposer** computes a tree decomposition (in normal form) and extends \mathcal{A} to \mathcal{A}_{td} .
- 3 **Program constructor** eliminates extensions of datalog (like set variables) and produces datalog program Π (in DLV syntax).
- 4 **DLV** is called on the combined datalog program $\mathcal{A}_{td} \cup \Pi$.

Discussion

- **Level of declarativity of MSO.** MSO has a high level of declarativity.
⇒ Elegant and succinct problem specifications possible.
However, **MSO does not have an operational semantics.**

Discussion

- **Level of declarativity of MSO.** MSO has a high level of declarativity.
⇒ Elegant and succinct problem specifications possible.
However, **MSO does not have an operational semantics.**
- **Level of declarativity of Datalog.** Datalog programs often reflect both the *intuition of the problem and of the algorithmic solution*. Datalog programs can also be used as a “specification” for a dedicated implementation in an imperative programming language.

Discussion

- **Level of declarativity of MSO.** MSO has a high level of declarativity.
⇒ Elegant and succinct problem specifications possible.
However, **MSO does not have an operational semantics.**
- **Level of declarativity of Datalog.** Datalog programs often reflect both the *intuition of the problem and of the algorithmic solution*. Datalog programs can also be used as a “specification” for a dedicated implementation in an imperative programming language.
- **Flexibility.** The generic transformation from MSO to monadic datalog leads to a big program – which can possibly be represented by a small non-monadic program.

Discussion

- **Level of declarativity of MSO.** MSO has a high level of declarativity.
⇒ Elegant and succinct problem specifications possible.
However, **MSO does not have an operational semantics.**
- **Level of declarativity of Datalog.** Datalog programs often reflect both the *intuition of the problem and of the algorithmic solution*. Datalog programs can also be used as a “specification” for a dedicated implementation in an imperative programming language.
- **Flexibility.** The generic transformation from MSO to monadic datalog leads to a big program – which can possibly be represented by a small non-monadic program.
- **Extending the programming language.** Definition of built-in predicates or extensions of datalog (like set arithmetic as in DLV-Complex).

Outline

1. Motivation
2. Treewidth of Finite Structures
3. MSO and Courcelle's Theorem
4. Expressive Power of Monadic Datalog
5. Efficient Computation with Monadic Datalog
6. Conclusion

Conclusion

Main Results

- Expressive power of MSO vs. monadic datalog over finite structures with bounded treewidth.
- Finding the right “level of declarativity”.
- New method for turning theoretical tractability results (obtained via Courcelle’s Theorem) into efficient computations in practice.
- New algorithms based on the monadic datalog approach.

Conclusion

Main Results

- Expressive power of MSO vs. monadic datalog over finite structures with bounded treewidth.
- Finding the right “level of declarativity”.
- New method for turning theoretical tractability results (obtained via Courcelle’s Theorem) into efficient computations in practice.
- New algorithms based on the monadic datalog approach.

Future Work

- Apply this approach to many more problems.
- Extensions of MSO, e.g.: counting, sum, max, min, etc.