Bounded Treewidth in Non-Monotonic Reasoning[1]

Reinhard Pichler

Institut für Informationssysteme
Arbeitsbereich DBAI
Technische Universität Wien

PCCR, Brno, August 2010

[1] Joint work with G. Gottlob, M. Jakl, S. Rümmele, F. Wei, and S. Woltran

to the memory of Marko Samer

to the memory of Marko Samer

Foundation of most of the work reported here:

Marko Samer, Stefan Szeider:
Algorithms for propositional model counting.
J. Discrete Algorithms 8(1): 50-64 (2010)

Outline

Outline

Motivation

Complexity in non-monotonic reasoning

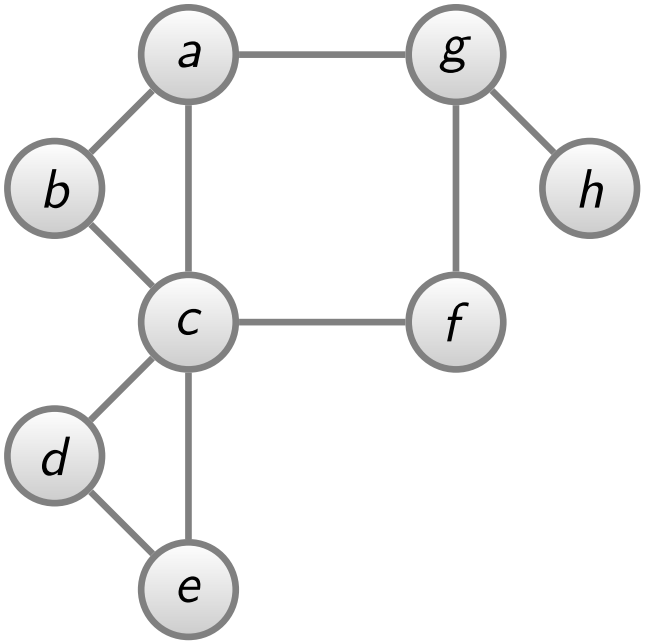
- Most forms of non-monotonic reasoning are intractable (even on the second or third level of the polynomial hierarchy)
- Examples: propositional abduction, closed world reasoning, answer set programming, belief revision
- treewidth as a key to fixed-parameter tractability: Courcelle's Theorem and extensions
- theoretical tractability vs. efficient computation
- efficient algorithms via dynamic programming

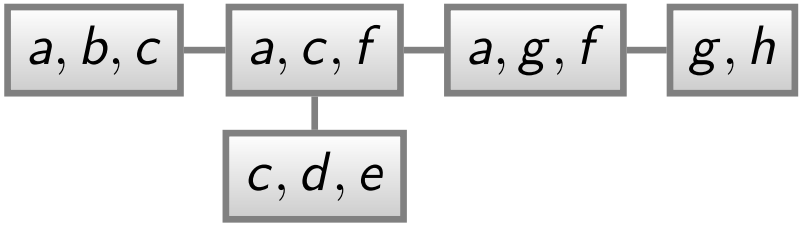
Outline

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.

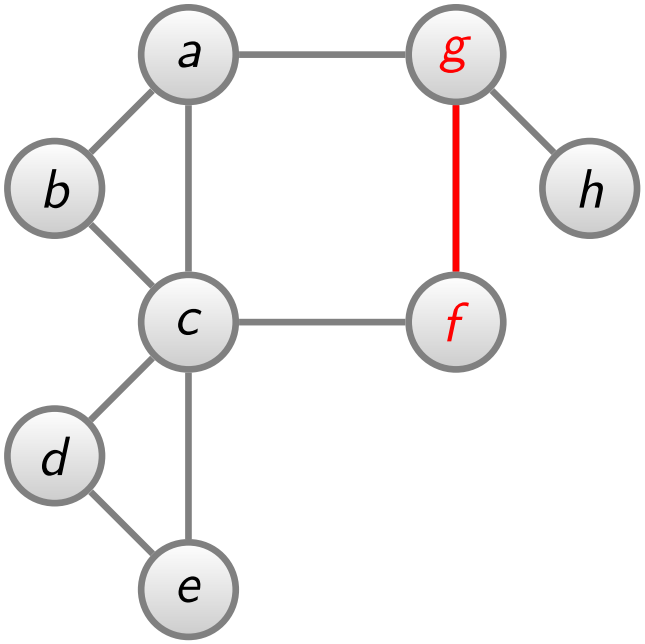


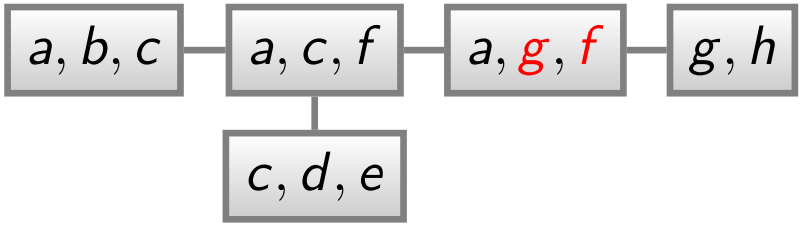


Tree Decomposition

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.

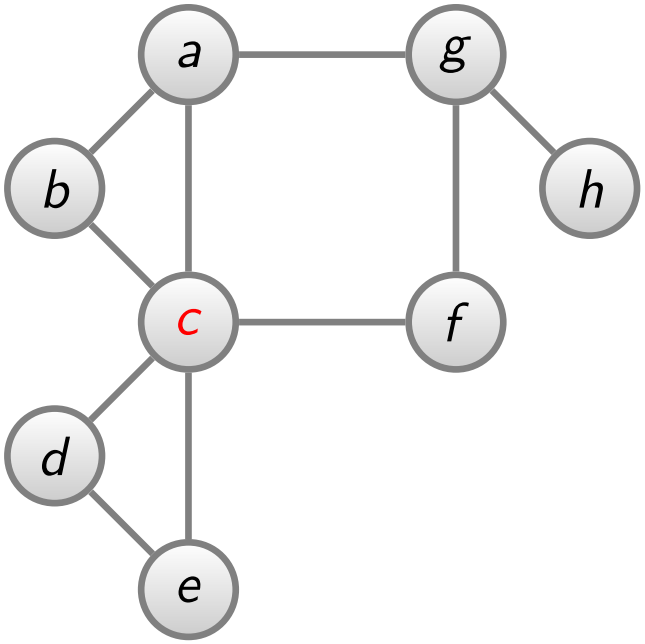


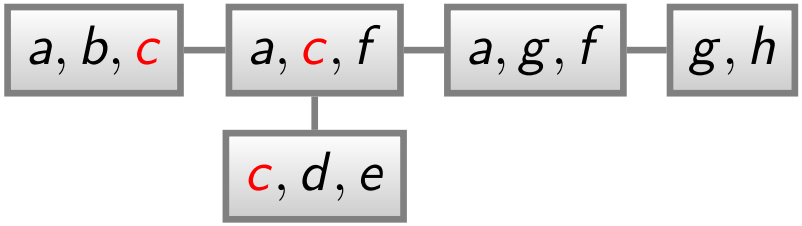


Tree Decomposition

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.

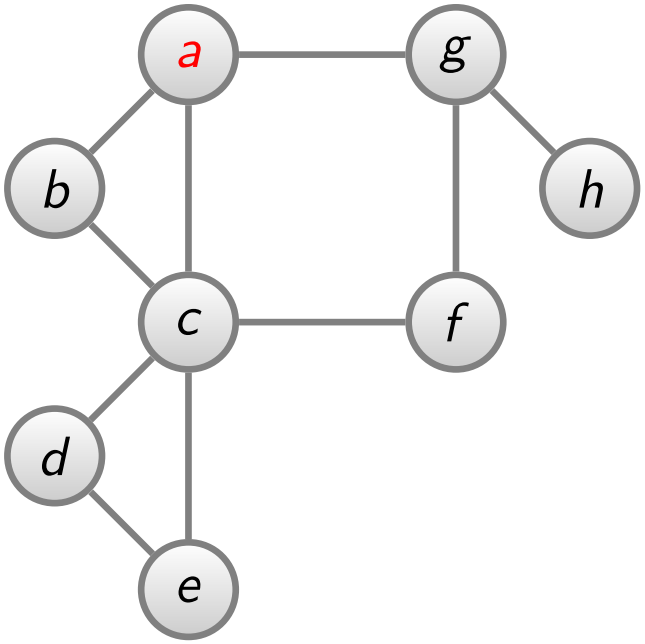


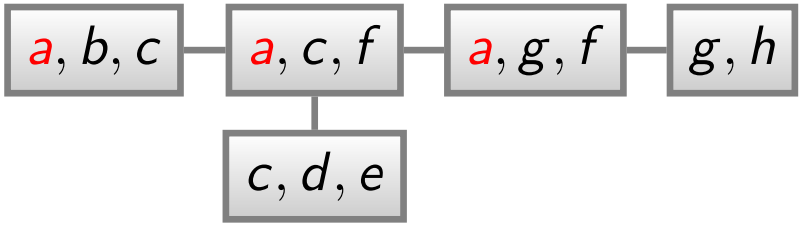


Tree Decomposition

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

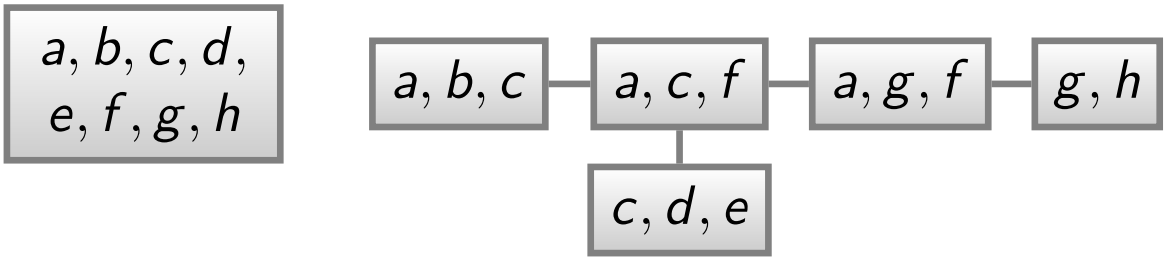
1. Tree with a vertex set (= "bag") associated with every node.
2. For every edge (v, w): there is a bag containing both v and w.
3. For every v: the nodes that contain v form a connected subtree.

Tree Decomposition of a Structure

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

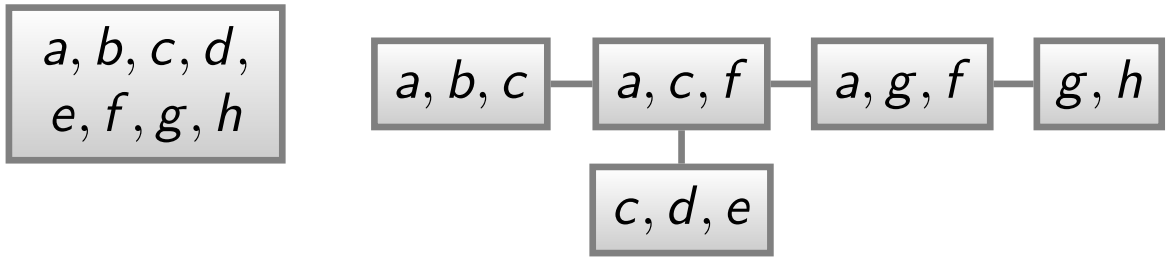
Treewidth

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



- (Bodlaender, 1996) For fixed k, it is feasible in linear time
  – to decide if a given graph/structure has treewidth $\leq k$
  – if so, to compute a tree decomposition of width $\leq k$.

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

$A(F)$

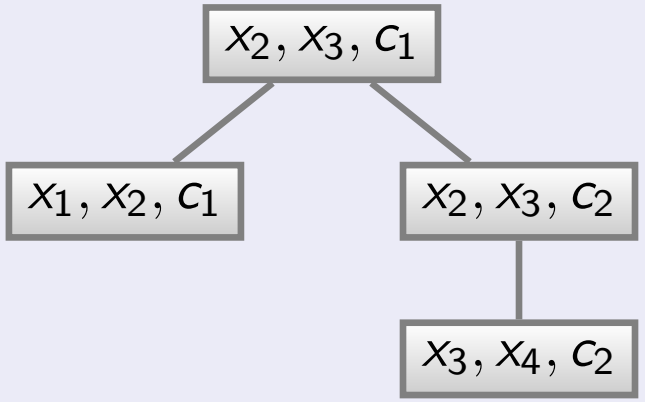
$cl(c_1)$, $cl(c_2)$,
$var(x_1)$, $var(x_2)$,
$var(x_3)$, $var(x_4)$,
$pos(x_1, c_1)$,
$pos(x_3, c_1)$,
$pos(x_2, c_2)$,
$pos(x_4, c_2)$,
$neg(x_2, c_1)$,
$neg(x_3, c_2)$.

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 $A(F)$

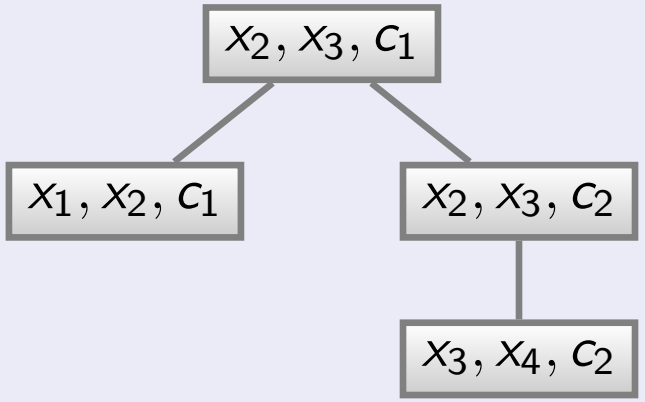


Tree Decomposition

$A(F)$

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

Tree Decomposition of $A(F)$



Remark. We have $tw(F) := tw(A(F)) = 2$.

Outline

Monadic-Second Order Logic (MSO)

Definition of MSO

Monadic Second Order logic (MSO) extends First Order logic (FO) by the use of set variables (usually denoted by upper case letters), which range over sets of domain elements.

In contrast, the individual variables (which are usually denoted by lower case letters) range over single domain elements.

Monadic-Second Order Logic (MSO)

Definition of MSO

Monadic Second Order logic (MSO) extends First Order logic (FO) by the use of set variables (usually denoted by upper case letters), which range over sets of domain elements.

In contrast, the individual variables (which are usually denoted by lower case letters) range over single domain elements.

Atomic formulae in an MSO-formula φ over a τ-structure:

- atoms with some predicate symbol from τ,
- atoms whose predicate symbol is a monadic second order variable (i.e., a set variable), or
- equality atoms.

Monadic-Second Order Logic (MSO)

Expressive Power of MSO

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

- Graph problems, database design problems, etc.
- KR and Reasoning: SAT, propositional abduction, belief revision, closed world reasoning, answer set programming, etc.

Monadic-Second Order Logic (MSO)

Expressive Power of MSO

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

- Graph problems, database design problems, etc.
- KR and Reasoning: SAT, propositional abduction, belief revision, closed world reasoning, answer set programming, etc.

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 1: 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 1: 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 1: 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:
$(\exists X)X \models F$

MSO-Example 2: Propositional Abduction

Definition

A propositional abduction problem (PAP) is a quadruple $\mathcal{P} = \langle V, H, M, \mathcal{C} \rangle$ consisting of

- a finite set of propositional *variables* V,
- a *theory* $\mathcal{C}$, which is a consistent set of clauses over the variables V,
- a set of *hypotheses* $H \subseteq V$,
- and a set of *manifestations* $M \subseteq V$.

MSO-Example 2: Propositional Abduction

Definition

A propositional abduction problem (PAP) is a quadruple $\mathcal{P} = \langle V, H, M, \mathcal{C} \rangle$ consisting of

- a finite set of propositional *variables* V,
- a *theory* $\mathcal{C}$, which is a consistent set of clauses over the variables V,
- a set of *hypotheses* $H \subseteq V$,
- and a set of *manifestations* $M \subseteq V$.

A set S satisfying $S \subseteq H$ is a solution iff

- $\mathcal{C} \cup S$ is consistent (i.e., satisfiable)
- and $\mathcal{C} \cup S \models M$ holds.

MSO-Example 2: Propositional Abduction

Abduction in System Diagnosis

A diagnosis problem can be represented by a propositional abduction problem $\mathcal{P} = \langle V, H, M, \mathcal{C} \rangle$ as follows:

- The clausal theory $\mathcal{C}$ is the system description.
- The hypotheses $H \subseteq V$ describe the possibly faulty system components.
- The manifestations $M \subseteq V$ are the observed symptoms (describing some malfunction of the system).
- The solutions $S \in \mathit{Sol}(\mathcal{P})$ are the possible explanations of the malfunction.

MSO-Example 2: Propositional Abduction

The main decision problems

Given a PAP $\mathcal{P}$ we ask:

- Solvability. Does there exist at least one solution for $\mathcal{P}$?

MSO-Example 2: Propositional Abduction

The main decision problems

Given a PAP $\mathcal{P}$ we ask:

- Solvability. Does there exist at least one solution for $\mathcal{P}$?

Given a PAP $\mathcal{P}$ and a hypothesis $h \in H$, we ask:

- Relevance: Is h contained in at least one solution of $\mathcal{P}$?
- Necessity: Is h contained in all solutions of $\mathcal{P}$?

MSO-Example 2: Propositional Abduction

The main decision problems

Given a PAP $\mathcal{P}$ we ask:

- Solvability. Does there exist at least one solution for $\mathcal{P}$?

Given a PAP $\mathcal{P}$ and a hypothesis $h \in H$, we ask:

- Relevance: Is h contained in at least one solution of $\mathcal{P}$?
- Necessity: Is h contained in all solutions of $\mathcal{P}$?

Remark. These problems are $\Sigma_2 P$-complete (Solvability and Relevance) resp. $\Pi_2 P$-complete (Necessity).

MSO-Example 2: Propositional Abduction

MSO-Encoding of the main Abduction problems

Let a PAP $\mathcal{P} = \langle V, H, M, \mathcal{C} \rangle$ be represented as a τ-structure with $\tau = \{cl, var, pos, neg, H, M\}$, where the predicates cl, var, pos, neg represent the clause set $\mathcal{C}$ and the unary predicates H and M identify the hypotheses and manifestations.

MSO-Example 2: Propositional Abduction

MSO-Encoding of the main Abduction problems

Let a PAP $\mathcal{P} = \langle V, H, M, \mathcal{C} \rangle$ be represented as a τ-structure with $\tau = \{cl, var, pos, neg, H, M\}$, where the predicates cl, var, pos, neg represent the clause set $\mathcal{C}$ and the unary predicates H and M identify the hypotheses and manifestations.

MSO encoding of $Sol(S)$, i.e., "S is a solution of $\mathcal{P}$":
$S \subseteq H \wedge (\exists X)[X \models \mathcal{C} \wedge S \subseteq X] \wedge (\forall Y)[(Y \models \mathcal{C} \wedge S \subseteq Y) \rightarrow M \subseteq Y]$

MSO-Example 2: Propositional Abduction

MSO-Encoding of the main Abduction problems

Let a PAP $\mathcal{P} = \langle V, H, M, \mathcal{C} \rangle$ be represented as a τ-structure with $\tau = \{cl, var, pos, neg, H, M\}$, where the predicates cl, var, pos, neg represent the clause set $\mathcal{C}$ and the unary predicates H and M identify the hypotheses and manifestations.

MSO encoding of $Sol(S)$, i.e., "S is a solution of $\mathcal{P}$":
$S \subseteq H \wedge (\exists X)[X \models \mathcal{C} \wedge S \subseteq X] \wedge (\forall Y)[(Y \models \mathcal{C} \wedge S \subseteq Y) \rightarrow M \subseteq Y]$

MSO encoding of the main Abduction problems:

Solvability: $(\exists S)Sol(S)$

Relevance: $(\exists S)[Sol(S) \wedge h \in S]$

Necessity: $(\forall S)[Sol(S) \rightarrow h \in S]$

Further Results

Closed World Reasoning

- Closed world assumption: assume negative information if the positive information is not explicitly given by a theory T.
- Several forms of closed world assumption: CWA (Closed World Assumption), GCWA (Generalized CWA), EGCWA (Extended GCWA), CCWA (Careful CWA), and ECWA (Extended CWA).

Further Results

Closed World Reasoning

- Closed world assumption: assume negative information if the positive information is not explicitly given by a theory T.
- Several forms of closed world assumption: CWA (Closed World Assumption), GCWA (Generalized CWA), EGCWA (Extended GCWA), CCWA (Careful CWA), and ECWA (Extended CWA).

Formal Definition of CWR

- $\mathrm{CWA}(T) = T \cup \{\neg K \mid K \text{ positive literal s.t. } T \not\models K\}$
- $\mathrm{GCWA}(T) = T \cup \{\neg K \mid K \text{ pos. literal and } \forall X \in MM(T)\colon X \not\models K\}$
- $\mathrm{EGCWA}(T) \models F$ iff $\forall X \in MM(T)\colon X \models F$

Further Results

Closed World Reasoning

- Closed world assumption: assume negative information if the positive information is not explicitly given by a theory T.
- Several forms of closed world assumption: CWA (Closed World Assumption), GCWA (Generalized CWA), EGCWA (Extended GCWA), CCWA (Careful CWA), and ECWA (Extended CWA).

Formal Definition of CWR

- $\mathrm{CWA}(T) = T \cup \{\neg K \mid K \text{ positive literal s.t. } T \not\models K\}$
- $\mathrm{GCWA}(T) = T \cup \{\neg K \mid K \text{ pos. literal and } \forall X \in MM(T)\colon X \not\models K\}$
- $\mathrm{EGCWA}(T) \models F$ iff $\forall X \in MM(T)\colon X \models F$
- CCWA, ECWA: like GCWA and EGCWA but distinguishing 3 kinds of variables (to be minimized, fixed, varying).

Further Results

Answer-set programming

- disjunctive datalog programs with negation in the body
- stable model semantics (defined via Gelfond-Lifschitz reduct)
- Gelfond-Lifschitz reduct and stable models can be defined in MSO
- MSO-definable decision problems: consistency, cautious reasoning, brave reasoning

Further Results

Answer-set programming

- disjunctive datalog programs with negation in the body
- stable model semantics (defined via Gelfond-Lifschitz reduct)
- Gelfond-Lifschitz reduct and stable models can be defined in MSO
- MSO-definable decision problems: consistency, cautious reasoning, brave reasoning

Belief Revision

- incorporate new information B into belief base A
- keep only those models of B with *minimal* symmetric set difference from any of the models of A.
- Satoh's revision operator: subset-minimality
- MSO-definable decision problems: cautious/brave reasoning

Extended MSO

Extension of Courcelle's Theorem

- Extended MSO: weight functions, cardinality, sum, min, max, etc.
- (Arnborg/Lagergren/Seese, 1991) Linear extended MSO extremum problems over finite structures can be decided in linear time (data complexity) if the structures have bounded treewidth.

Extended MSO-Encodings

- Propositional Abduction: restriction to solutions of minimal cardinality or minimal weight
- Belief revision: Dalal's revision operator: cardinality-minimality

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

Problem 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

Problem 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: (Grohe 1999), similarly (Niedermeier, 2006)

- 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

Nice Tree Decompositions

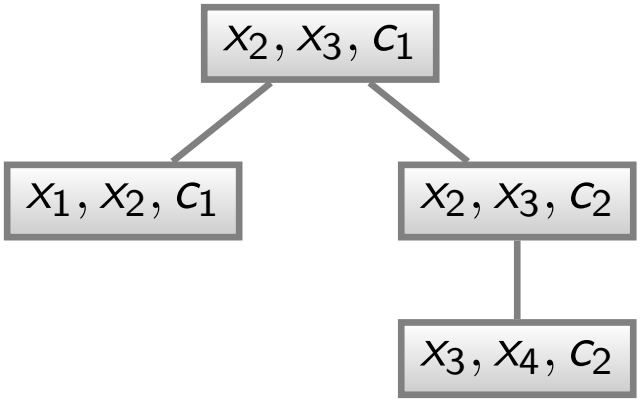
A tree decomposition $\mathcal{T}$ is called nice if

- $\mathcal{T}$ is a rooted binary tree,
- nodes with 2 children have the same bags as their children,
- the bags of nodes with one child differ in exactly 1 element from the bag of the child.

Nice Tree Decompositions

A tree decomposition $\mathcal{T}$ is called nice if

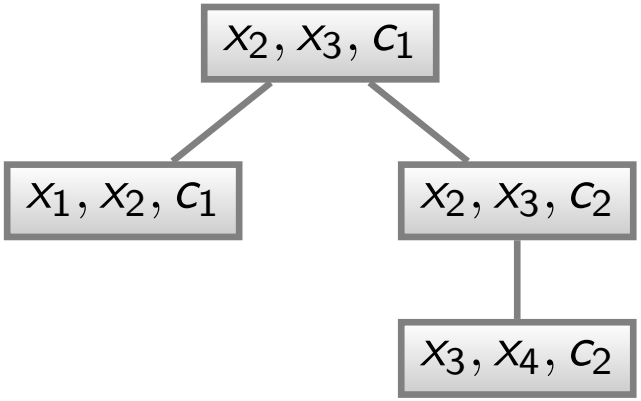
- $\mathcal{T}$ is a rooted binary tree,
- nodes with 2 children have the same bags as their children,
- the bags of nodes with one child differ in exactly 1 element from the bag of the child.

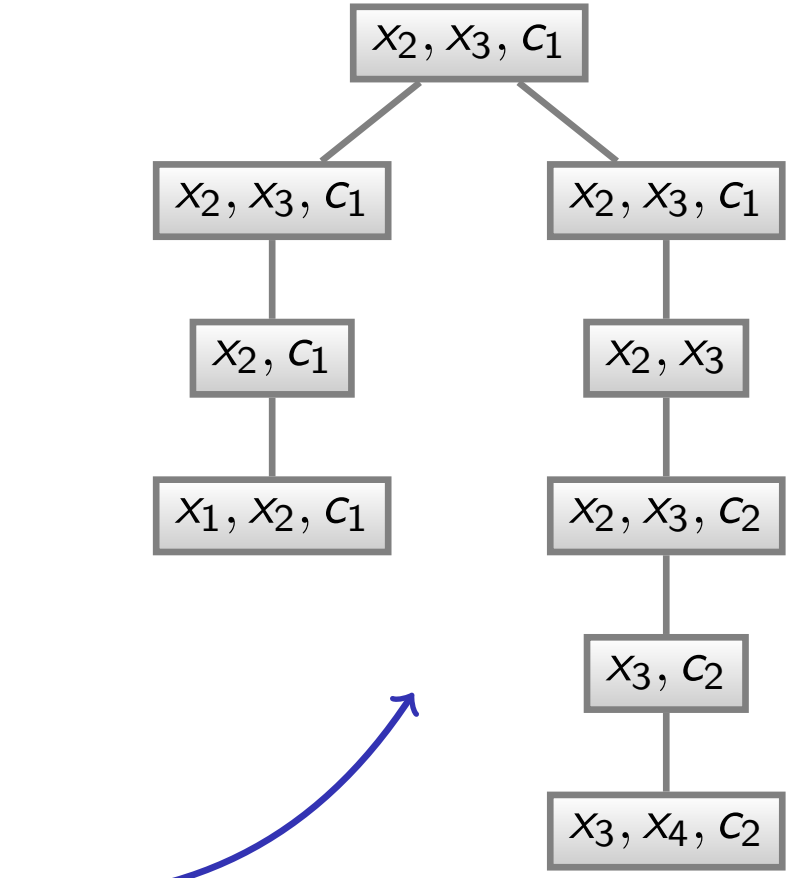


Nice Tree Decompositions

A tree decomposition $\mathcal{T}$ is called nice if

- $\mathcal{T}$ is a rooted binary tree,
- nodes with 2 children have the same bags as their children,
- the bags of nodes with one child differ in exactly 1 element from the bag of the child.

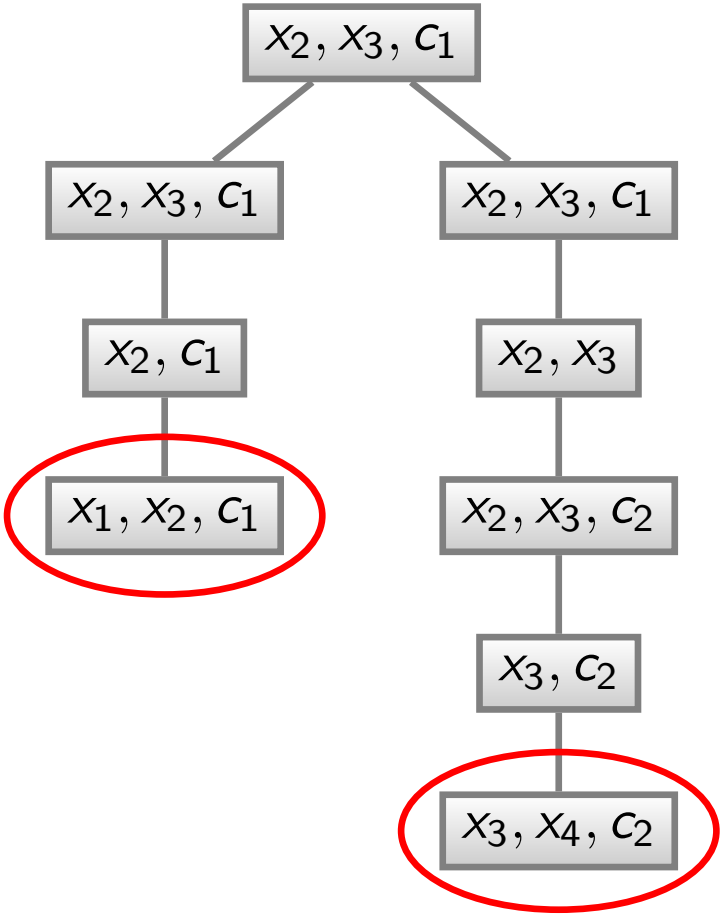




Nice Tree Decompositions

We distinguish:

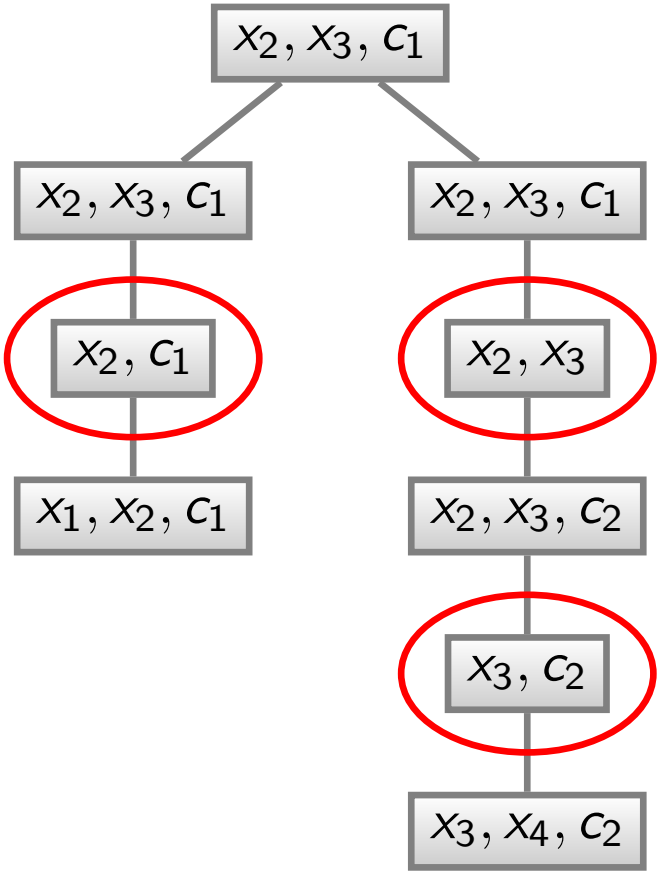
- Leaf nodes.
- Forget nodes (variable or clause).
- Introduce nodes (variable or clause).
- Branch nodes.



Nice Tree Decompositions

We distinguish:

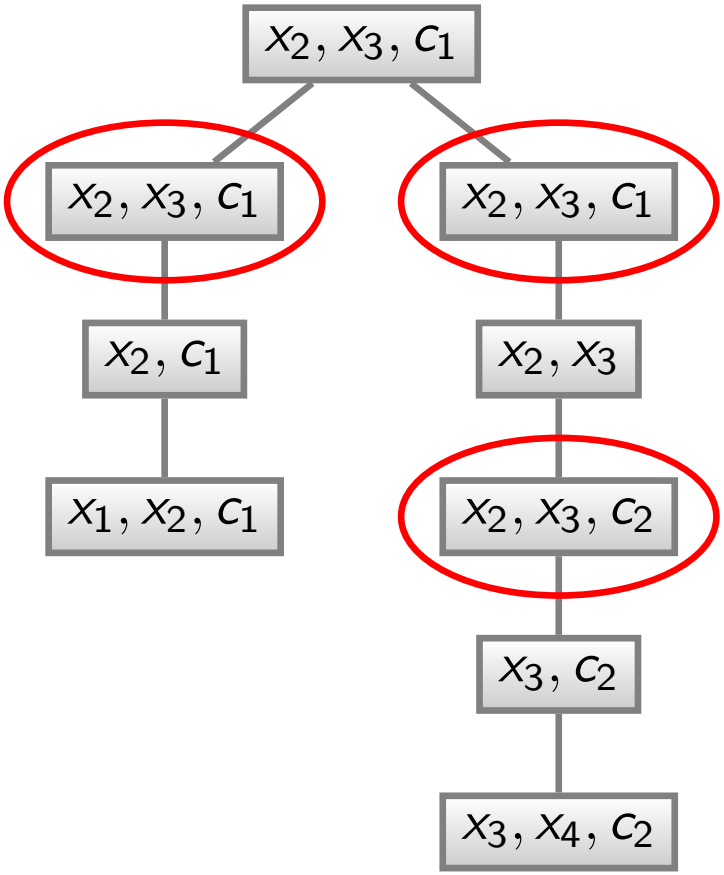
- Leaf nodes.
- Forget nodes (variable or clause).
- Introduce nodes (variable or clause).
- Branch nodes.



Nice Tree Decompositions

We distinguish:

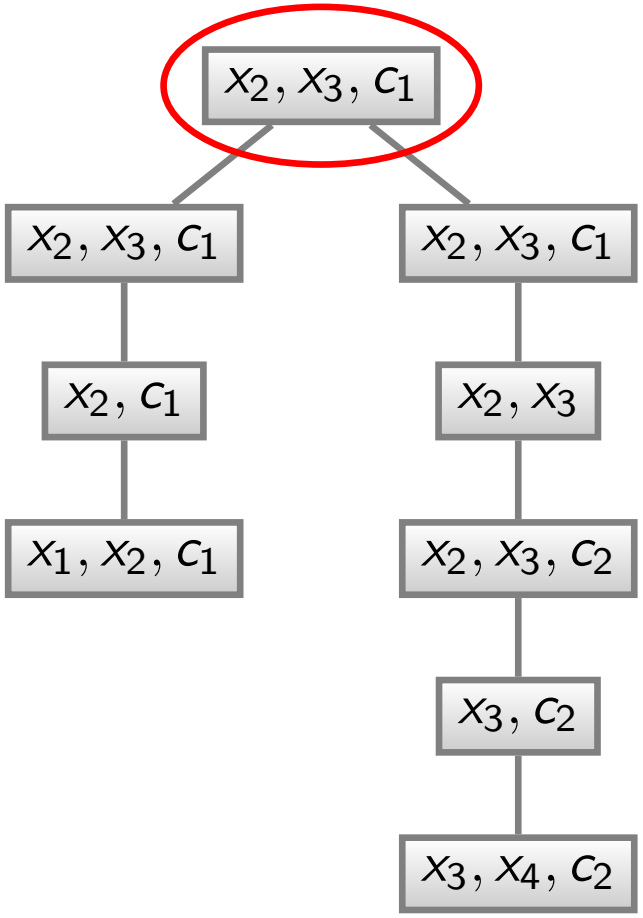
- Leaf nodes.
- Forget nodes (variable or clause).
- Introduce nodes (variable or clause).
- Branch nodes.



Nice Tree Decompositions

We distinguish:

- Leaf nodes.
- Forget nodes (variable or clause).
- Introduce nodes (variable or clause).
- Branch nodes.



#SAT-Algorithm (Samer/Szeider, 2010)

Notation

- F ... CNF formula.
- (T, χ, r) ... Nice tree decomposition of F.
- χ_V, χ_C ... χ restricted to variables/clauses.
- T_t ... Subtree of T rooted at node t.
- V_t, C_t ... Variables/Clauses in the bags of T_t.

Data Structure

For each truth assignment $\alpha : \chi_V(t) \to \{0,1\}$ and subset $A \subseteq \chi_C(t)$ let $N(t, \alpha, A)$ be the set of truth assignments $\tau : V_t \to \{0,1\}$, s.t.

- $\tau(x) = \alpha(x)$ for all $x \in \chi_V(t)$.
- A is exactly the set of clauses in C_t that are not satisfied by τ.

#SAT-Algorithm (Samer/Szeider, 2010)

Notation

- F ... CNF formula.
- (T, χ, r) ... Nice tree decomposition of F.
- χ_V, χ_C ... χ restricted to variables/clauses.
- T_t ... Subtree of T rooted at node t.
- V_t, C_t ... Variables/Clauses in the bags of T_t.

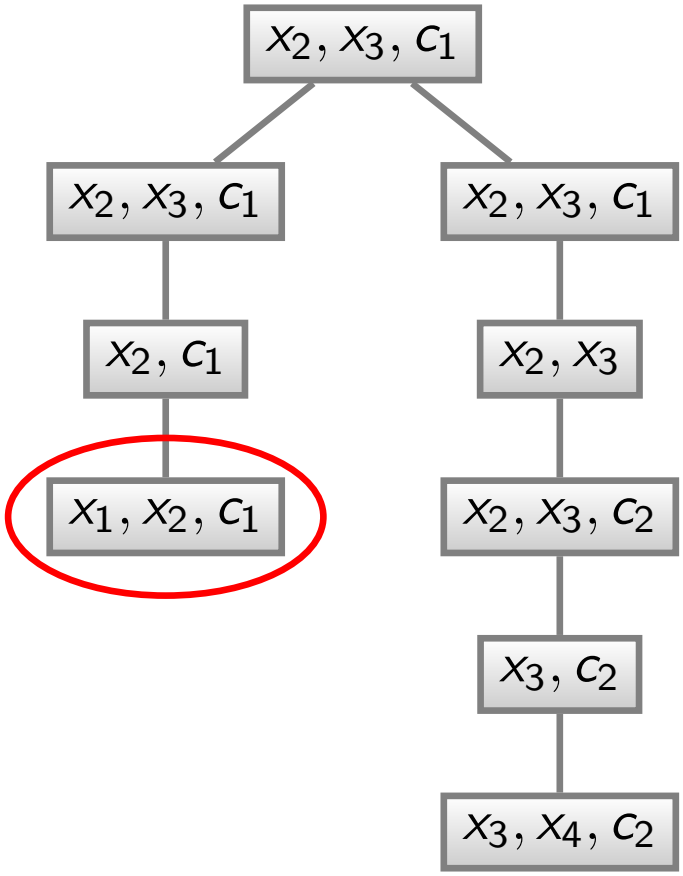
Data Structure

For each truth assignment $\alpha : \chi_V(t) \to \{0,1\}$ and subset $A \subseteq \chi_C(t)$ let $N(t, \alpha, A)$ be the set of truth assignments $\tau : V_t \to \{0,1\}$, s.t.

- $\tau(x) = \alpha(x)$ for all $x \in \chi_V(t)$.
- A is exactly the set of clauses in C_t that are not satisfied by τ.

Let $n(t, \alpha, A) = |N(t, \alpha, A)|$.

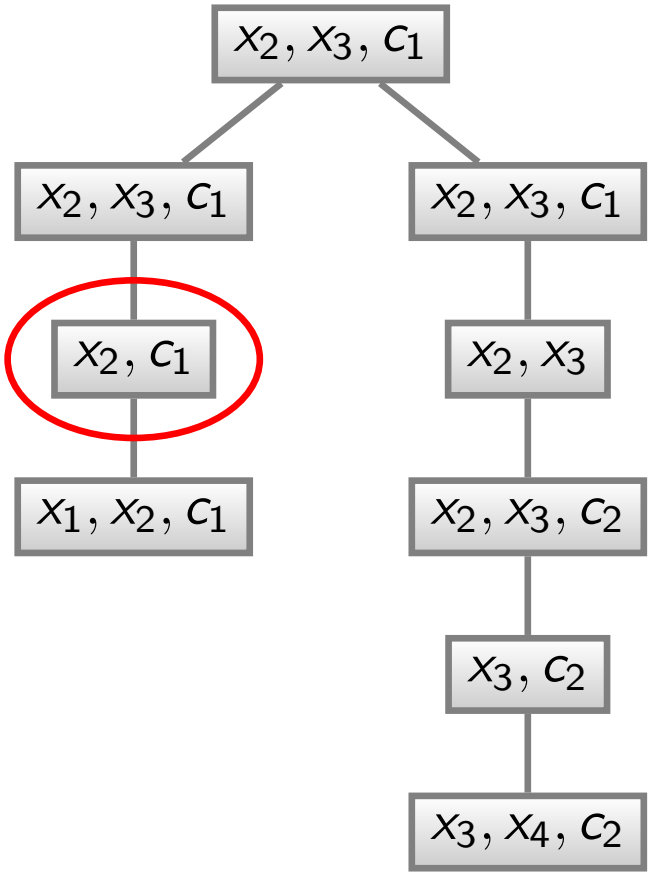
Example



$$\underbrace{(x_1 \vee \bar{x}_2 \vee x_3)}_{c_1} \wedge \underbrace{(x_2 \vee \bar{x}_3 \vee x_4)}_{c_2}$$

α	A	n
-	-	1
x_1	-	1
x_2	c_1	1
x_1, x_2	-	1

Example

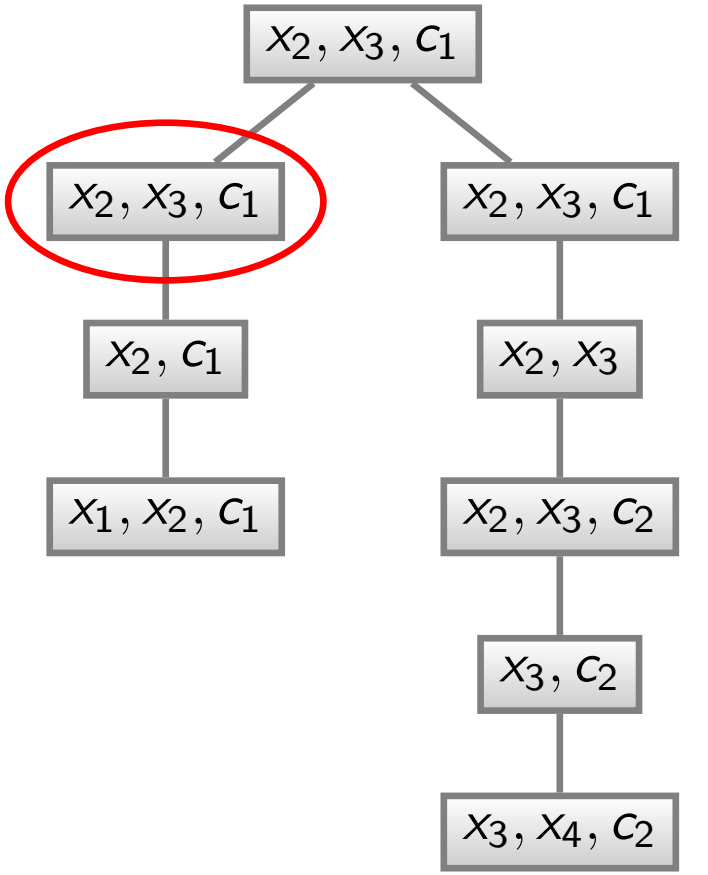


$$\underbrace{(x_1 \vee \bar{x}_2 \vee x_3)}_{c_1} \wedge \underbrace{(x_2 \vee \bar{x}_3 \vee x_4)}_{c_2}$$

α	A	n
-	-	1
x_1	-	1
x_2	c_1	1
x_1, x_2	-	1

α	A	n
-	-	2
x_2	c_1	1
x_2	-	1

Example

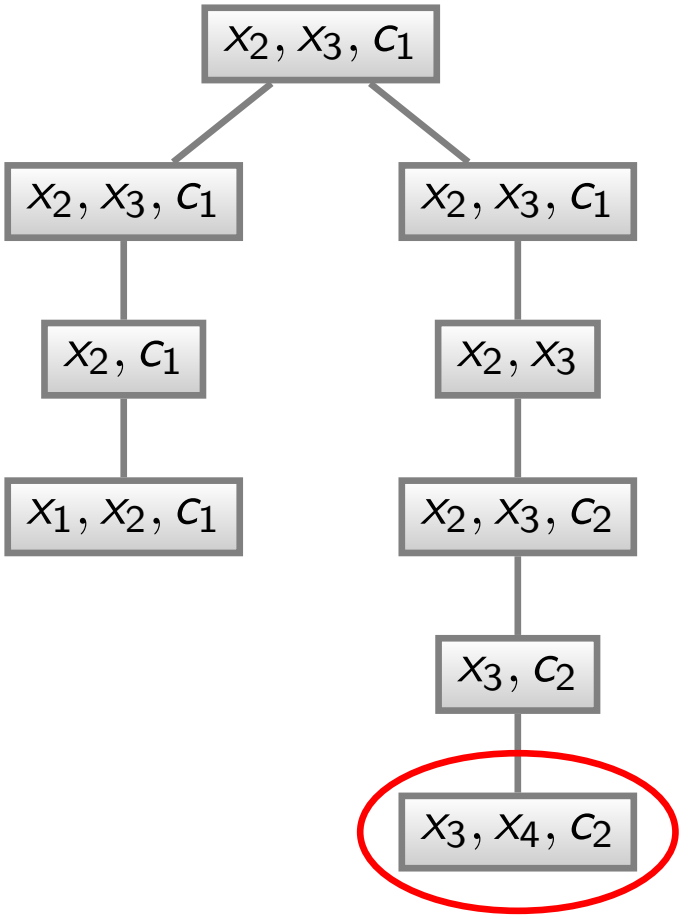


$$\underbrace{(x_1 \vee \bar{x}_2 \vee x_3)}_{c_1} \wedge \underbrace{(x_2 \vee \bar{x}_3 \vee x_4)}_{c_2}$$

α	A	n
-	-	2
x_2	c_1	1
x_2	-	1

α	A	n
-	-	2
x_2	c_1	1
x_2	-	1
x_3	-	2
x_2, x_3	-	2

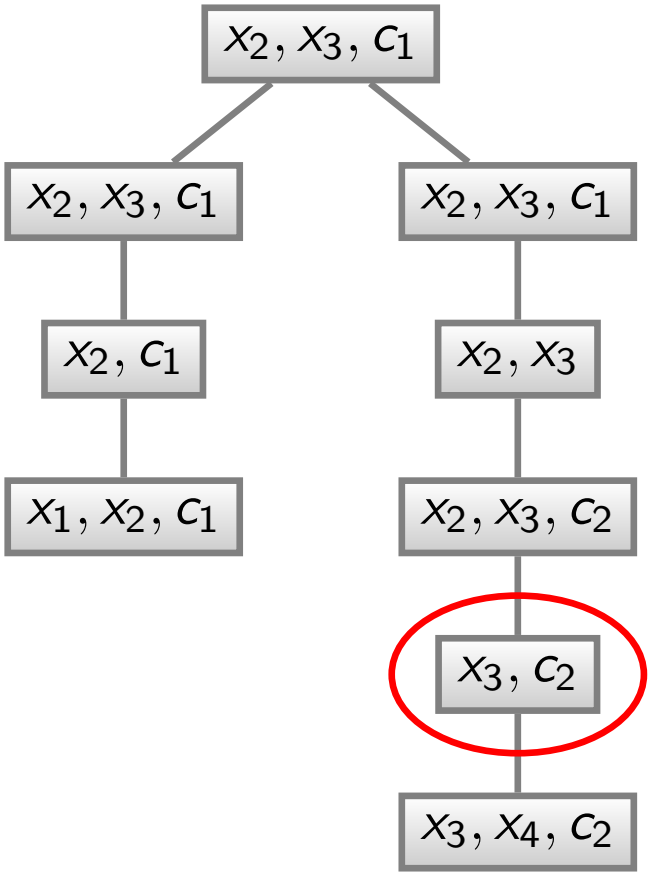
Example



$$\underbrace{(x_1 \vee \bar{x}_2 \vee x_3)}_{c_1} \wedge \underbrace{(x_2 \vee \bar{x}_3 \vee x_4)}_{c_2}$$

α	A	n
-	-	1
x_3	c_2	1
x_4	-	1
x_3, x_4	-	1

Example

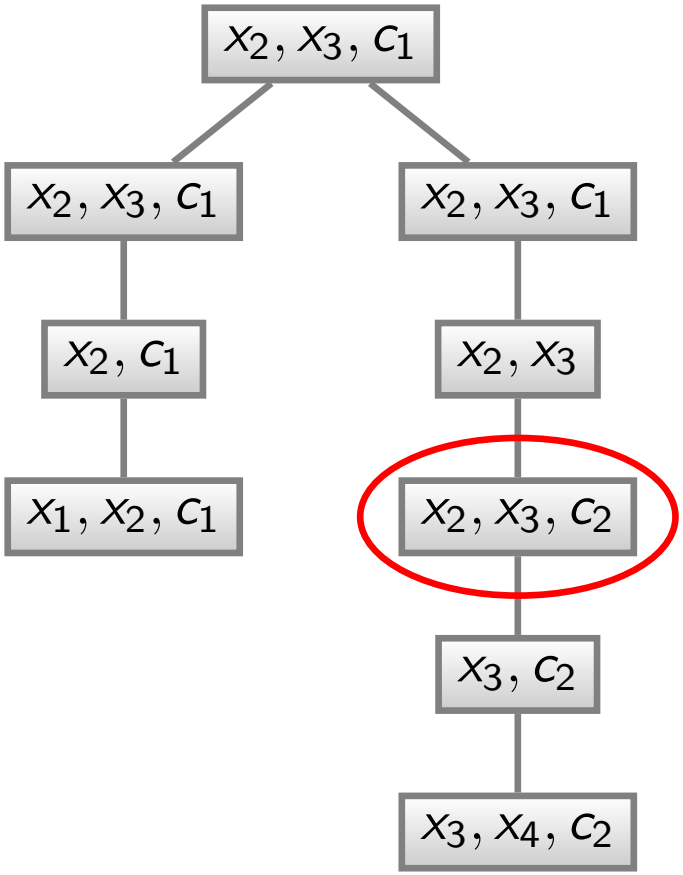


$$\underbrace{(x_1 \vee \bar{x}_2 \vee x_3)}_{c_1} \wedge \underbrace{(x_2 \vee \bar{x}_3 \vee x_4)}_{c_2}$$

α	A	n
-	-	1
x_3	c_2	1
x_4	-	1
x_3, x_4	-	1

α	A	n
-	-	2
x_3	c_2	1
x_3	-	1

Example

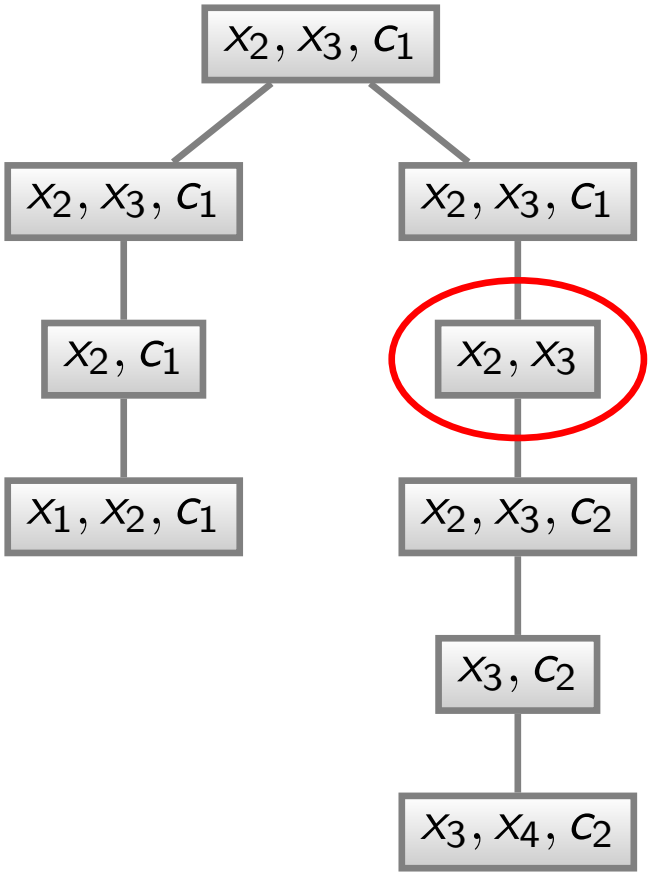


$$\underbrace{(x_1 \vee \bar{x}_2 \vee x_3)}_{c_1} \wedge \underbrace{(x_2 \vee \bar{x}_3 \vee x_4)}_{c_2}$$

α	A	n
-	-	2
x_3	c_2	1
x_3	-	1

α	A	n
-	-	2
x_2	-	2
x_3	c_2	1
x_3	-	1
x_2, x_3	-	2

Example

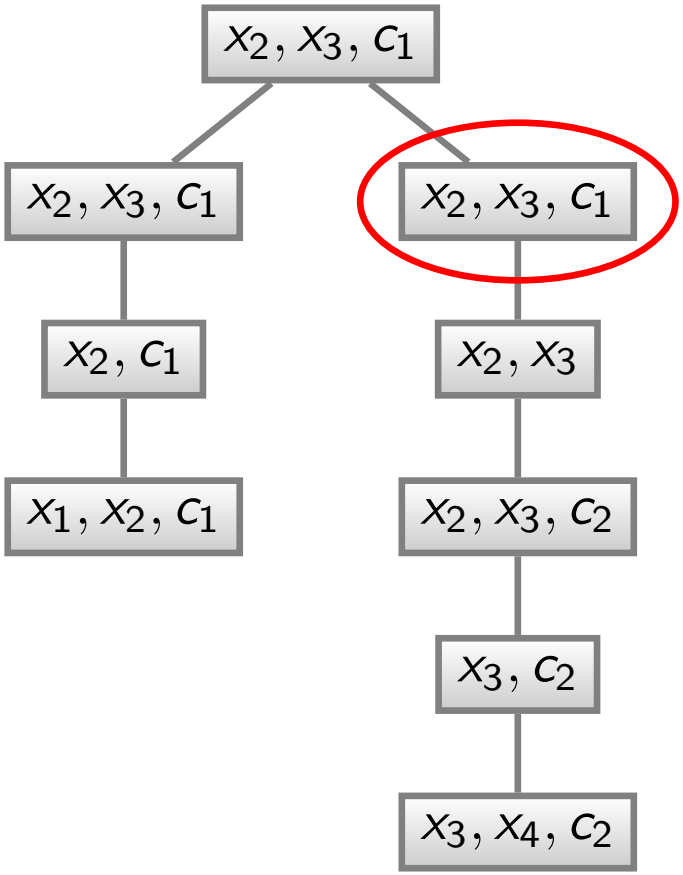


$$\underbrace{(x_1 \vee \bar{x}_2 \vee x_3)}_{c_1} \wedge \underbrace{(x_2 \vee \bar{x}_3 \vee x_4)}_{c_2}$$

α	A	n
-	-	2
x_2	-	2
x_3	c_2	1
x_3	-	1
x_2, x_3	-	2

α	A	n
-	-	2
x_2	-	2
x_3	-	1
x_2, x_3	-	2

Example

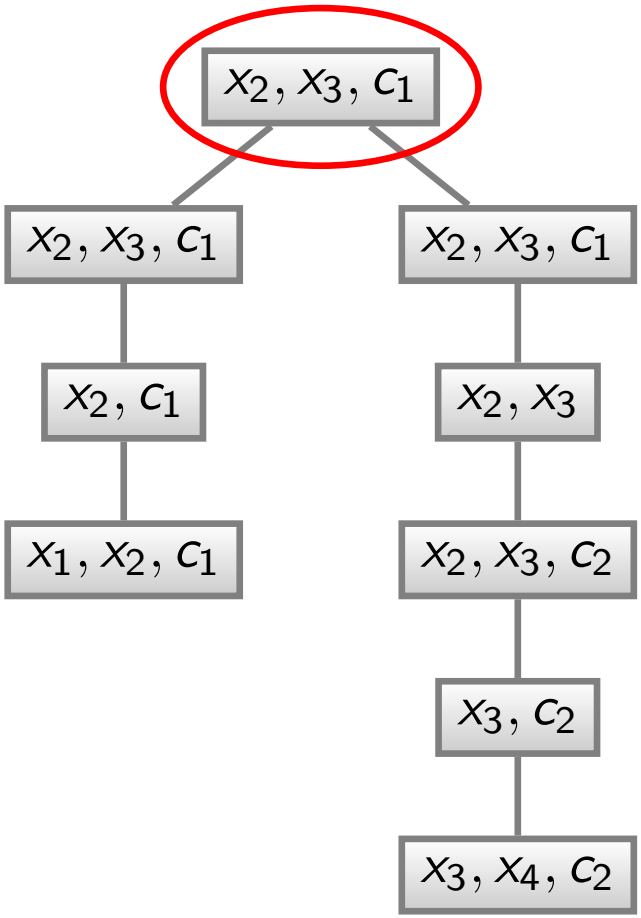


$$\underbrace{(x_1 \vee \bar{x}_2 \vee x_3)}_{c_1} \wedge \underbrace{(x_2 \vee \bar{x}_3 \vee x_4)}_{c_2}$$

α	A	n
-	-	2
x_2	-	2
x_3	-	1
x_2, x_3	-	2

α	A	n
-	-	2
x_2	c_1	2
x_3	-	1
x_2, x_3	-	2

Example



$$\underbrace{(x_1 \vee \bar{x}_2 \vee x_3)}_{c_1} \wedge \underbrace{(x_2 \vee \bar{x}_3 \vee x_4)}_{c_2}$$

α	A	n
-	-	2
x_2	c_1	1
x_2	-	1
x_3	-	2
x_2, x_3	-	2

α	A	n
-	-	2
x_2	c_1	2
x_3	-	1
x_2, x_3	-	2

α	A	n
-	-	4
x_2	c_1	2
x_2	-	2
x_3	-	2
x_2, x_3	-	4

Leaf and Join Nodes

Lemma

Let t be a leaf node. Then, for each truth assignment $\alpha : \chi_V(t) \to \{0,1\}$ and set $A \subseteq \chi_C(t)$, we have

$$n(t,\alpha,A) = \begin{cases} 1 & \text{if } A = \{c \in \chi_C(t) : \alpha \text{ does not satisfy } c\}; \\ 0 & \text{otherwise.} \end{cases}$$

Lemma

Let t be a join node of T with children t_1, t_2. Then, for each truth assignment $\alpha : \chi_V(t) \to \{0,1\}$ and set $A \subseteq \chi_C(t)$, we have

$$n(t,\alpha,A) = \sum_{A_1,A_2 \subseteq \chi_C(t), A_1 \cap A_2 = A} n(t_1,\alpha,A_1) \cdot n(t_2,\alpha,A_2).$$

Variable Introduce Nodes

Lemma

Let t be a variable introduce node with child t' and $\chi(t) = \chi(t') \cup \{x\}$ for a variable x. Then, for each truth assignment $\alpha : \chi_V(t) \to \{0,1\}$ and set $A \subseteq \chi_C(t)$, we have

$$n(t, \alpha_{x=0}, A) = \begin{cases} 0 & \text{if } \neg x \in c \text{ for some } c \in A; \\ \sum_{B' \subseteq \bar{B}} n(t', \alpha, A \cup B') & \text{otherwise;} \end{cases}$$

$$\text{where } \bar{B} = \{c \in \chi_C(t) : \neg x \in c\};$$

$$n(t, \alpha_{x=1}, A) = \begin{cases} 0 & \text{if } x \in c \text{ for some } c \in A; \\ \sum_{B' \subseteq B} n(t', \alpha, A \cup B') & \text{otherwise;} \end{cases}$$

$$\text{where } B = \{c \in \chi_C(t) : x \in c\}.$$

Clause Introduce Nodes

Lemma

Let t be a clause introduce node with child t' and $\chi(t) = \chi(t') \cup \{c\}$ for a clause c. Then, for each truth assignment $\alpha : \chi_V(t) \to \{0,1\}$ and set $A \subseteq \chi_C(t)$, we have

$$n(t,\alpha,A) = \begin{cases} n(t',\alpha,A) & \text{if } c \notin A \text{ and } \alpha \text{ satisfies } c; \\ n(t',\alpha,A \setminus \{c\}) & \text{if } c \in A \text{ and } \alpha \text{ does not satisfy } c; \\ 0 & \text{otherwise.} \end{cases}$$

Forget Nodes

Lemma

Let t be a variable forget node with child t' and $\chi(t) = \chi(t') \setminus \{x\}$. Then, for each truth assignment $\alpha : \chi_V(t) \to \{0,1\}$ and set $A \subseteq \chi_C(t)$, we have

$$n(t, \alpha, A) = n(t', \alpha_{x=0}, A) + n(t', \alpha_{x=1}, A).$$

Lemma

Let t be a clause forget node with child t' and $\chi(t) = \chi(t') \setminus \{c\}$. Then, for each truth assignment $\alpha : \chi_V(t) \to \{0,1\}$ and set $A \subseteq \chi_C(t)$, we have

$$n(t, \alpha, A) = n(t', \alpha, A).$$

Summary

Data Structure

In this dynamic programming algorithm, we have to maintain the following data structure at each node t in the tree decomposition T:

(α, A, n), where $\alpha : \chi_V(t) \to \{0, 1\}$, $A \subseteq \chi_C(t)$, and n is an integer.

Summary

Data Structure

In this dynamic programming algorithm, we have to maintain the following data structure at each node t in the tree decomposition T:

(α, A, n), where $\alpha : \chi_V(t) \to \{0, 1\}$, $A \subseteq \chi_C(t)$, and n is an integer.

Alternative view: represent (α, A) as a subset of $\chi(t)$. Hence, if T has width $\leq w - 1$, we have to store a table of size $\mathrm{O}(w2^w)$ at each node.

Summary

Data Structure

In this dynamic programming algorithm, we have to maintain the following data structure at each node t in the tree decomposition T:

(α, A, n), where $\alpha : \chi_V(t) \to \{0,1\}$, $A \subseteq \chi_C(t)$, and n is an integer.

Alternative view: represent (α, A) as a subset of $\chi(t)$. Hence, if T has width $\leq w-1$, we have to store a table of size $\mathrm{O}(w2^w)$ at each node.

Theorem (Samer/Szeider, 2010)

The #SAT-problem can be solved in space $\mathrm{O}(w2^w \log(N))$ and time $\mathrm{O}(w4^w N)$, where N denotes the number of nodes in a tree decomposition of width $w-1$ for an input CNF formula F (assuming unit cost for arithmetic operations).

Summary

Lessons learned

- The models of a CNF-formula F can be computed by means of a single bottom-up traversal of a tree decomposition T of F:
  - assign truth value to the variables in the bags at leaf nodes and
  - extend the truth assignments at the variable introduce nodes.

Summary

Lessons learned

- The models of a CNF-formula F can be computed by means of a single bottom-up traversal of a tree decomposition T of F:
  - assign truth value to the variables in the bags at leaf nodes and
  - extend the truth assignments at the variable introduce nodes.
- We cannot afford to store all truth assignments on V_t.
- It suffices to store the "projection" of each truth assignment on V_t to the variables and clauses in $\chi(t)$.
- The resulting data structure is single-exponential in the treewidth.
- Each "projection" may represent many truth assignments on V_t: these are indistinguishable for the further bottom-up traversal.

Dynamic Programming in Propositional Abduction

Definition

A propositional abduction problem (PAP) is a quadruple $\mathcal{P} = \langle V, H, M, \mathcal{C} \rangle$ consisting of

- a finite set of propositional *variables* V,
- a *theory* $\mathcal{C}$, which is a consistent set of clauses over the variables V,
- a set of *hypotheses* $H \subseteq V$,
- and a set of *manifestations* $M \subseteq V$.

Dynamic Programming in Propositional Abduction

Definition

A propositional abduction problem (PAP) is a quadruple $\mathcal{P} = \langle V, H, M, \mathcal{C} \rangle$ consisting of

- a finite set of propositional *variables* V,
- a *theory* $\mathcal{C}$, which is a consistent set of clauses over the variables V,
- a set of *hypotheses* $H \subseteq V$,
- and a set of *manifestations* $M \subseteq V$.

A set S satisfying $S \subseteq H$ is a solution iff

- $\mathcal{C} \cup S$ is consistent (i.e., satisfiable)
- and $\mathcal{C} \cup S \models M$ holds.

Dynamic Programming in Propositional Abduction

Intuition of Computing a Solution

- The clause set $\mathcal{C}$ has, in general, many models.
- There may exist models of $\mathcal{C}$ where a manifestation $m \in M$ is false.
- Idea of constructing a solution S: adding a variable $h \in H$ to S means that we discard all models of $\mathcal{C}$ where h is false.

Dynamic Programming in Propositional Abduction

Intuition of Computing a Solution

- The clause set $\mathcal{C}$ has, in general, many models.
- There may exist models of $\mathcal{C}$ where a manifestation $m \in M$ is false.
- Idea of constructing a solution S: adding a variable $h \in H$ to S means that we discard all models of $\mathcal{C}$ where h is false.
- Difficulty (and reason for Σ_2-completeness):
  - S must be "big enough" so that every manifestation $m \in M$ is true in the remaining models of $\mathcal{C}$.
  - S must be "small enough" so that at least one model of $\mathcal{C}$ is left.

Dynamic Programming in Propositional Abduction

Dynamic Programming Algorithm

- In a bottom-up traversal of the tree decomposition T, at each node $t \in T$, consider all subsets $S \subseteq H_t$.
- For every such S do:
  - Consider all possible truth assignments I for $\mathcal{C} \cup S$.
  - Keep track if some manifestation $m \in M$ is set to false in I.
- Read off the solutions at the root node:
  - $\mathcal{C} \cup S$ has at least one model.
  - In all models of $\mathcal{C} \cup S$, no $m \in M$ is set to false.

FPT-Algorithm for Propositional Abduction

Data Structure and Complexity

- Recall from the #SAT-algorithm the idea of projecting truth assignments on V_t onto the bag $\chi(t)$.

FPT-Algorithm for Propositional Abduction

Data Structure and Complexity

- Recall from the #SAT-algorithm the idea of projecting truth assignments on V_t onto the bag $\chi(t)$.
- Data structure required for Abduction at each node $t \in T$:
  - Restriction of each set $S \subseteq H_t$ to $S \cap \chi(t)$.
  - Set of projections of all possible truth assignments for $\mathcal{C} \cup S$.
  - One bit for each such projection indicating if some $m \in M_t$ is false in the corresponding truth assignments.
- Formal definition of the data structure: $2^{\chi(t)} \times 2^{2^{\chi(t)} \times \{0,1\}}$.
- Complexity of the algorithm: double-exponential in the treewidth.

Dynamic Programming in Closed World Reasoning

Minimal Models of a CNF-Formula F

- In a bottom-up traversal of the tree decomposition T, at each node $t \in T$, consider all possible truth assignments I for F.
- For every truth assignment I, keep track of all strictly smaller truth assignments I', i.e.: at every variable introduce node, if the new variable x is set to true in I, then a new I' is obtained by setting x to false (and letting I' coincide with I on all other variables).
- Read off the minimal models I of F at the root node:
  - I is a model of F, i.e., no clause of F is false in I.
  - For all assignments I', that are strictly smaller than I: check that I' is not a model of F, i.e., at least one clause of F is false in I.

FPT-Algorithm for Closed World Reasoning

Data Structure and Complexity

- We again use the idea of projecting truth assignments on V_t onto the bag $\chi(t)$ for every $t \in T$.
- Data structure required for CWR at each node $t \in T$:
  - Projection of a truth assignment I of F.
  - Set of projections of all possible truth assignments I' of F, s.t. I' is strictly smaller than I.
- Formal definition of the data structure: $2^{\chi(t)} \times 2^{2^{\chi(t)}}$.
- Complexity of the algorithm: double-exponential in the treewidth.

Dynamic Programming in Answer Set Programming

Definition of Answer Sets

- Definition of the Gelfond/Lifschitz reduct P^I:
  - Delete all rules, s.t. some negative literal in the body is false in I.
  - In the remaining rules, delete the negative literals from the body.
- Definition of Answer Sets I of P:
  - I is a model of P.
  - I is a minimal model of P^I.

Dynamic Programming in Answer Set Programming

Definition of Answer Sets

- Definition of the Gelfond/Lifschitz reduct P^I:
  - Delete all rules, s.t. some negative literal in the body is false in I.
  - In the remaining rules, delete the negative literals from the body.
- Definition of Answer Sets I of P:
  - I is a model of P.
  - I is a minimal model of P^I.

Dynamic Programming Algorithm

- In a bottom-up traversal of the tree decomposition T, at each node $t \in T$, consider all possible truth assignments I for P.
- For every truth assignment I, keep track of all strictly smaller truth assignments I' for P^I.
- Read off the answer sets I of P at the root node:
  I is a model of P, but no I' strictly smaller than I is a model of P^I.

Dynamic Programming in Belief Revision

Satoh's Revision Operator

- Let A, B be propositional formulae in CNF.
- Goal: incorporate new information B into belief base A.
- Idea: keep only those models of B with *subset-minimal* symmetric set difference from any of the models of A.

Dynamic Programming in Belief Revision

Satoh's Revision Operator

- Let A, B be propositional formulae in CNF.
- Goal: incorporate new information B into belief base A.
- Idea: keep only those models of B with *subset-minimal* symmetric set difference from any of the models of A.

Dynamic Programming Algorithm

- Keep track of pairs of truth assignments (I, J) for A resp. B.
- For every such pair, keep track of all pairs (I', J') with strictly smaller symmetric set difference.
- Read off desired models of B at the root node:
  - There exists a pair (I, J), s.t. I is a model of A and J a model of B.
  - For all pairs (I', J') with strictly smaller symmetric set difference, either I' is not a model of I' or J' is not a model of B.

Extensions

Other types of problems

- Optimization Problems: during the bottom-up traversal, keep track of the intermediate value of the target function.
- Counting Problems: compute the number of solutions in abduction, the number of minimal models, the number of answer sets, etc.

Extensions

Other types of problems

- Optimization Problems: during the bottom-up traversal, keep track of the intermediate value of the target function.
- Counting Problems: compute the number of solutions in abduction, the number of minimal models, the number of answer sets, etc.
- Enumeration Problems: compute all solutions, all minimal models, all answer sets, etc. with fixed-parameter linear delay:
  - during bottom-up traversal: keep track of the relationship between the data structure at the child node(s) and the parent node
  - compute the output by one top-down traversal per solution, minimal model, answer set, etc.

Extensions

Other types of problems

- Optimization Problems: during the bottom-up traversal, keep track of the intermediate value of the target function.
- Counting Problems: compute the number of solutions in abduction, the number of minimal models, the number of answer sets, etc.
- Enumeration Problems: compute all solutions, all minimal models, all answer sets, etc. with fixed-parameter linear delay:
  - during bottom-up traversal: keep track of the relationship between the data structure at the child node(s) and the parent node
  - compute the output by one top-down traversal per solution, minimal model, answer set, etc.
- (Special Case) Enumeration Problem defined by a unary predicate: For instance, compute all relevant resp. necessary hypotheses in an abduction problem. A single top-down traversal suffices.

Experimental Results

Implementation

Some of the algorithms presented here have been implemented in
Datalog (SAT),
C (propositional abduction), or
Haskell (answer set programming).

Experience

- Datalog: reasonable performance for instances of low treewidth, FPL-behaviour not achievable by using datalog engine as black box.
- C: FPL-behaviour achievable, e.g.: abduction for treewidth 3.
- Haskell: almost as fast as C-programme; formal description of dynamic programming algorithm is very close to programme code, e.g.: answer set programming with treewidth ≤ 7.

Conclusion: Functional programming language seems to be best suited.

Outline

Conclusion

Main Results

- Application of treewidth to non-monotonic reasoning
- Many FPT-results via Courcelle's Theorem
- Efficient computation with dynamic programming algorithms

Conclusion

Main Results

- Application of treewidth to non-monotonic reasoning
- Many FPT-results via Courcelle's Theorem
- Efficient computation with dynamic programming algorithms

Future Work

- Dynamic programming algorithms for further problems
- Other structural decomposition methods like clique-width
- Further extensions of Courcelle's Theorem