Towards Fixed-Parameter Tractable Algorithms for Argumentation[◇]

KR'2010 (Toronto, Canada)

Wolfgang Dvořák, Reinhard Pichler, Stefan Woltran

Database and Artificial Intelligence Group
Institut für Informationssysteme
Technische Universität Wien

May 12, 2010

[◇] This work was supported by the Vienna Science and Technology Fund (WWTF) under grant ICT08-028 and by the Austrian Science Fund (FWF) under grant P20704-N18.

Motivation

Abstract Argumentation

- Increasing interest for reasoning in argumentation frameworks (AFs).
- Many reasoning tasks are computationally intractable.
- As AFs can be considered as graphs there is broad range of graph parameters we can consider to identify tractable fragments.
- FPT results (in terms of treewidth) for Argumentation already exist [Dunne, 2007]; obtained via Courcelle's Theorem.
- But: This theorem *"is a very elegant and powerful tool for quickly deciding about FPT, but it is far from any efficient implementation"* [Niedermeier, 2006].

Motivation

Fixed-Parameter Tractability

- Often computational costs primarily depend on some problem parameters rather than on the mere size of the instances.
- Many hard problems become tractable if some problem parameter is fixed or bounded by a fixed constant.
- In the arena of graphs an important parameter is treewidth, which measures the "tree-likeness" of a graph. The treewidth has served as the key to many fixed-parameter tractability (FPT) results.

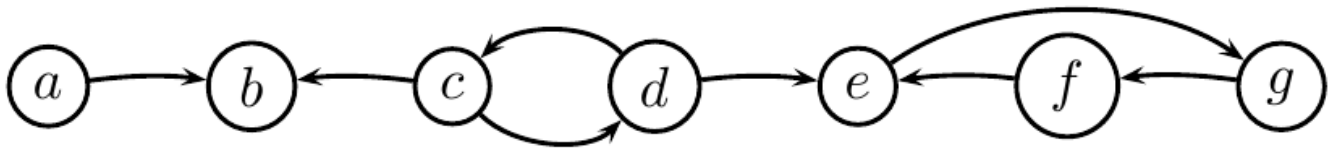
Argumentation Frameworks

Argumentation Frameworks

An argumentation framework (AF) is a pair (A, R) where

- A is a set of arguments
- $R \subseteq A \times A$ is a relation representing "attacks" ("defeats")

Example:



Extension based Semantics - Admissible Extensions

Admissible Extension

Given an AF (A, R). A set $S \subseteq A$ is admissible in F, if

- S is conflict-free in F
  - $\{a, b\} \subseteq S \Rightarrow (a, b) \notin R$
- each $a \in S$ is defended by S in F,
  - $a \in A$ is defended by S in F, if for each $b \in A$ with $(b, a) \in R$, there exists a $c \in S$, such that $(c, b) \in R$.

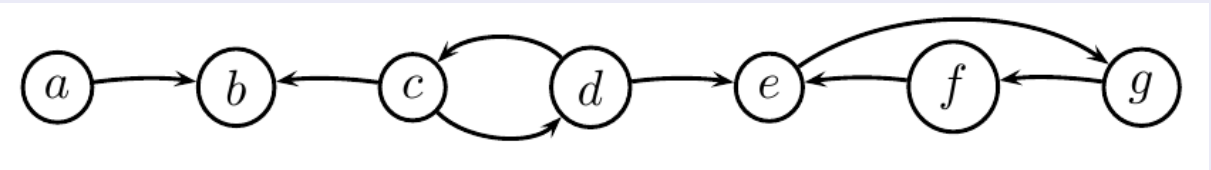
Extension based Semantics - Admissible Extensions

Admissible Extension

Given an AF (A, R). A set $S \subseteq A$ is admissible in F, if

- S is conflict-free in F
  - $\{a, b\} \subseteq S \Rightarrow (a, b) \notin R$
- each $a \in S$ is defended by S in F,
  - $a \in A$ is defended by S in F, if for each $b \in A$ with $(b, a) \in R$, there exists a $c \in S$, such that $(c, b) \in R$.

Example



$adm(F) = \{\{a, d, g\}, \{a, c\}, \{a, d\}, \{d, g\}, \{a\}, \{c\}, \{d\}, \emptyset\}$

Extension based Semantics - Preferred Extension

Preferred Extension

Given an AF (A, R). A set $S \subseteq A$ is preferred in F, if

- S is admissible in F
- for each $T \subseteq A$ admissible in T, $S \not\subset T$

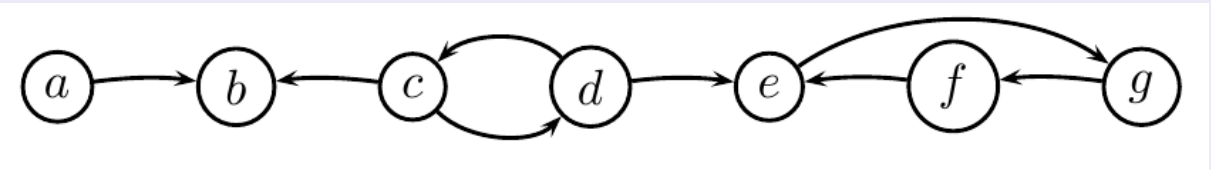
Extension based Semantics - Preferred Extension

Preferred Extension

Given an AF (A, R). A set $S \subseteq A$ is preferred in F, if

- S is admissible in F
- for each $T \subseteq A$ admissible in T, $S \not\subset T$

Example



$adm(F) = \{\{a, d, g\}, \{a, c\}, \{a, d\}, \{d, g\}, \{a\}, \{c\}, \{d\}, \emptyset\}$
$pref(F) = \{\{a, d, g\}, \{a, c\}\}$

Decision Problems

Credulous Acceptance

Given an AF $F = (A, R)$ and an argument $x \in A$.
Is x in at least one preferred extension ?

For credulous acceptance it suffices to consider admissible extensions.

Skeptical Acceptance

Given an AF $F = (A, R)$ and an argument $x \in A$.
Is x in every preferred extension ?

Decision Problems

Credulous Acceptance

Given an AF $F = (A, R)$ and an argument $x \in A$.
Is x in at least one preferred extension ?

For credulous acceptance it suffices to consider admissible extensions.

Skeptical Acceptance

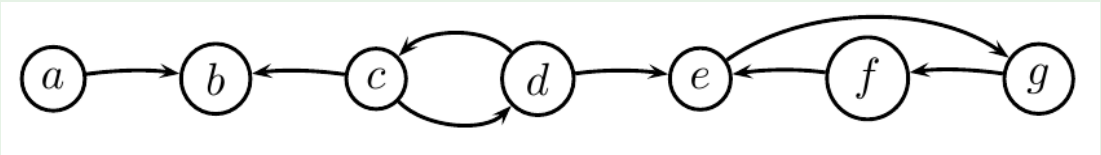
Given an AF $F = (A, R)$ and an argument $x \in A$.
Is x in every preferred extension ?

Complexity:

- The credulous acceptance problem is NP-complete (Dimopoulos and Torres, 1996).
- The skeptical acceptance problem is Π_2^p-complete (Dunne and Bench-Capon, 2002).

Tree-Decomposition

Argumentation Framework

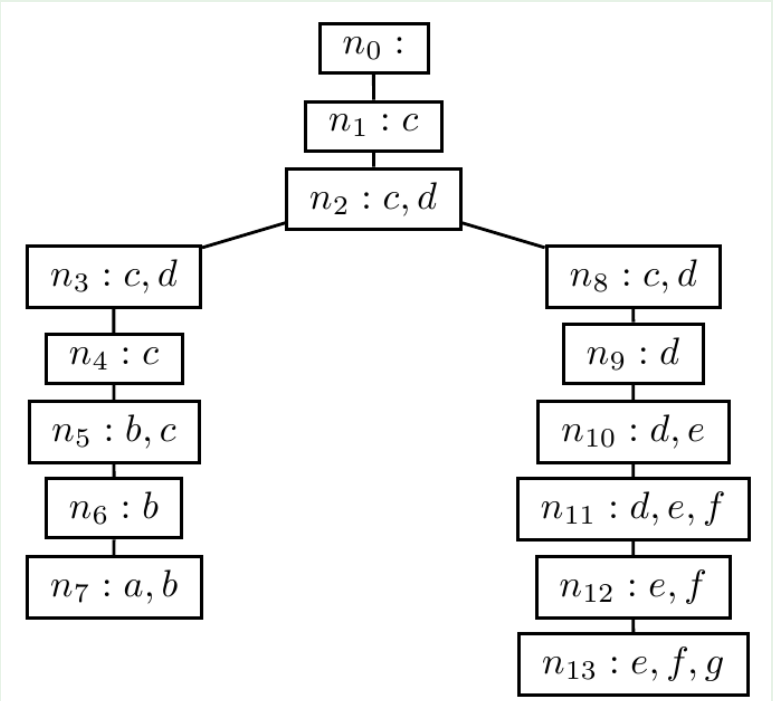


Properties

For an AF $F = (A, R)$:

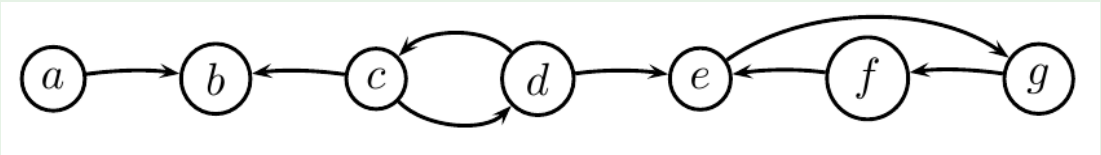
1. Each argument $a \in A$ and each attack $(b, c) \in R$ is contained in at least one bag
2. Bags containing the same argument are connected

Tree-Decomposition



Tree-Decomposition

Argumentation Framework

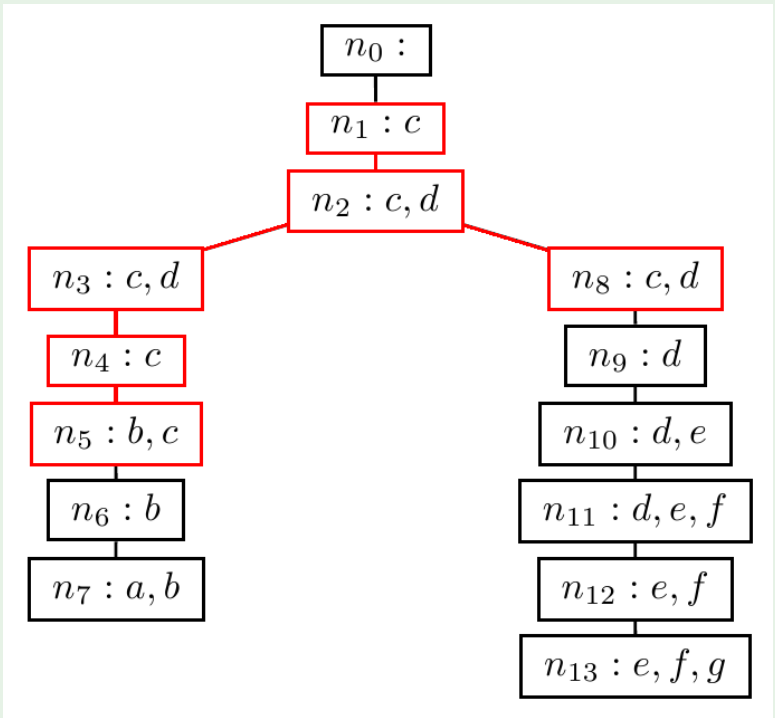


Properties

For an AF $F = (A, R)$:

1. Each argument $a \in A$ and each attack $(b, c) \in R$ is contained in at least one bag
2. Bags containing the same argument are connected

Tree-Decomposition



Tree-Width

Width

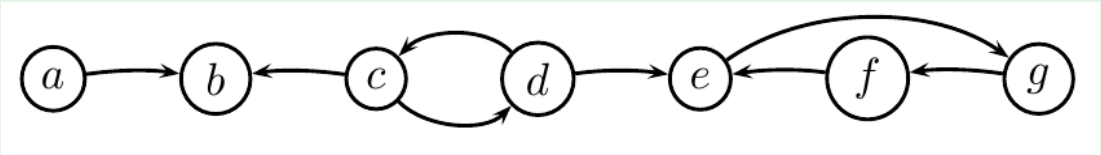
The width of a tree-decomposition is the size off the largest bag - 1.

Tree-Width

The tree-width of an AF is the minimum width over all possible tree-decompositions.

Nice Tree-Decomposition

Argumentation Framework



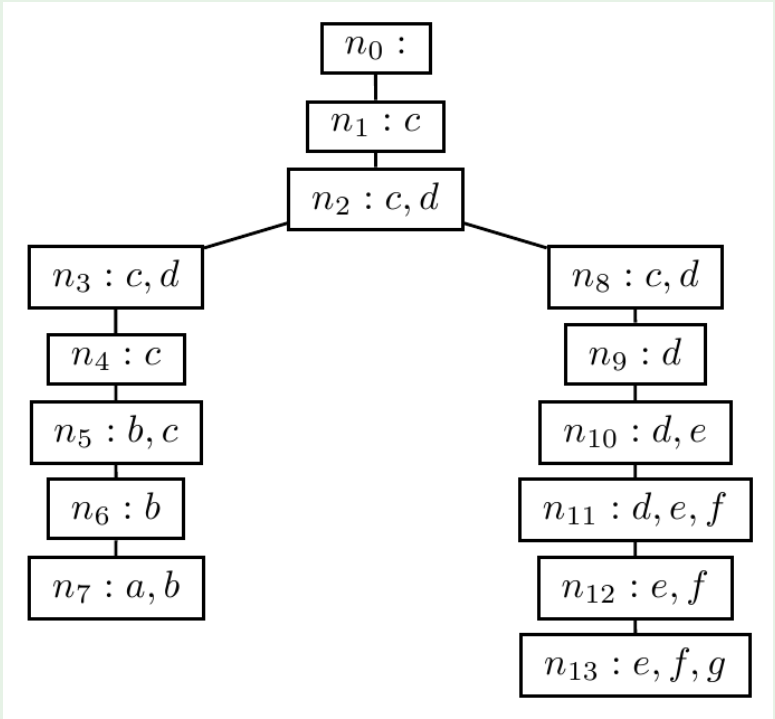
Node Types

- ROOT: root node of the tree, with empty bag

Each node is eiter a

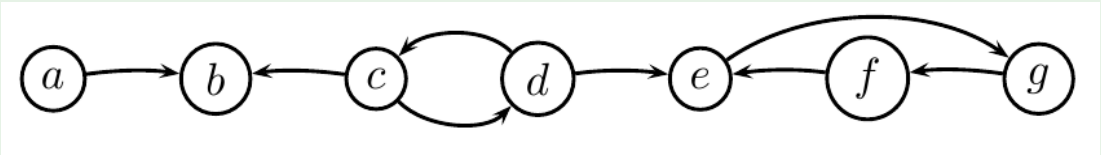
- LEAF: leaf node of the tree
- FORGET: eliminates one argument of the successor
- INSERT: adds an argument
- JOIN: combines two nodes, the successors bags coincide

Nice Tree-Decomposition



Nice Tree-Decomposition

Argumentation Framework



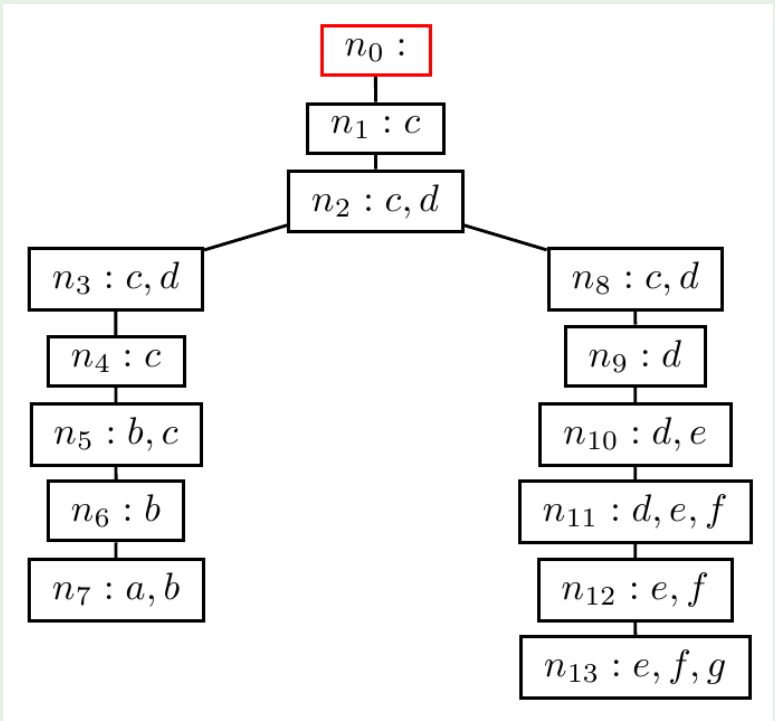
Node Types

- **ROOT:** root node of the tree, with empty bag

Each node is eiter a

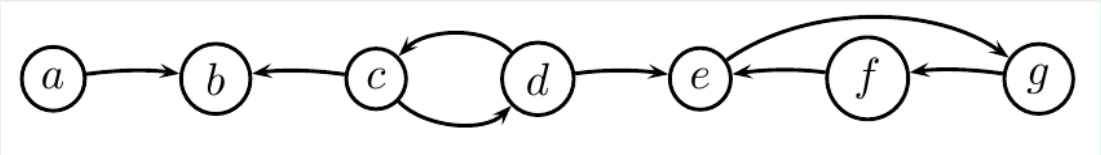
- LEAF: leaf node of the tree
- FORGET: eliminates one argument of the successor
- INSERT: adds an argument
- JOIN: combines two nodes, the successors bags coincide

Nice Tree-Decomposition



Nice Tree-Decomposition

Argumentation Framework



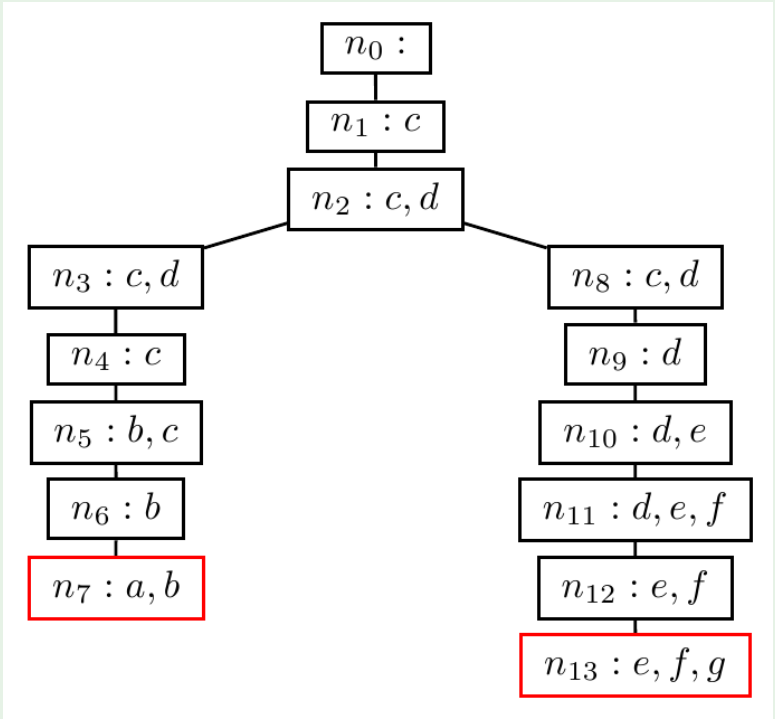
Node Types

- ROOT: root node of the tree, with empty bag

Each node is eiter a

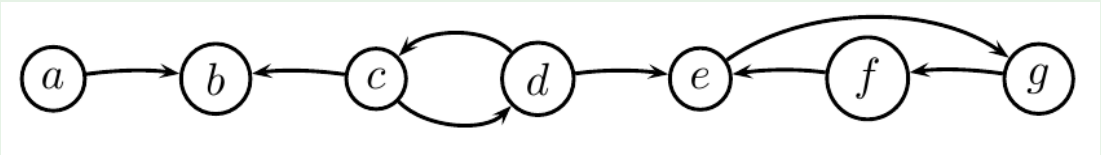
- **LEAF:** leaf node of the tree
- FORGET: eliminates one argument of the successor
- INSERT: adds an argument
- JOIN: combines two nodes, the successors bags coincide

Nice Tree-Decomposition



Nice Tree-Decomposition

Argumentation Framework



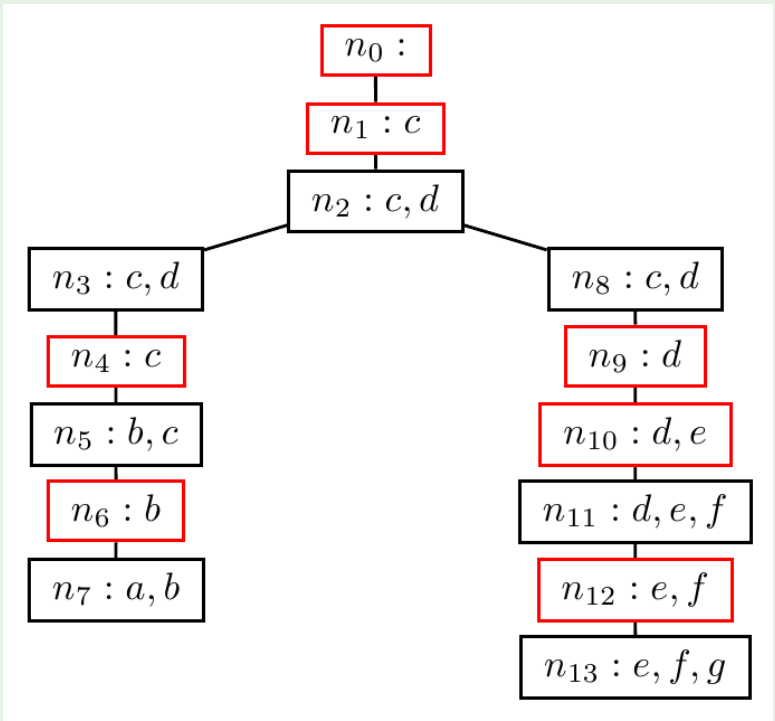
Node Types

- ROOT: root node of the tree, with empty bag

Each node is eiter a

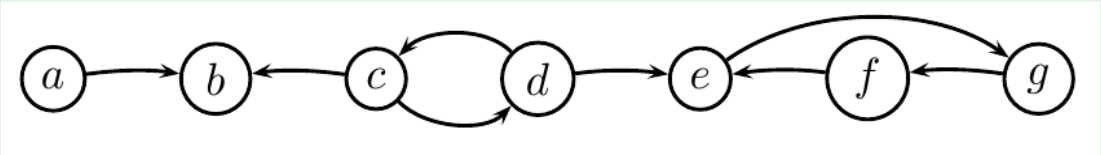
- LEAF: leaf node of the tree
- **FORGET:** eliminates one argument of the successor
- INSERT: adds an argument
- JOIN: combines two nodes, the successors bags coincide

Nice Tree-Decomposition



Nice Tree-Decomposition

Argumentation Framework



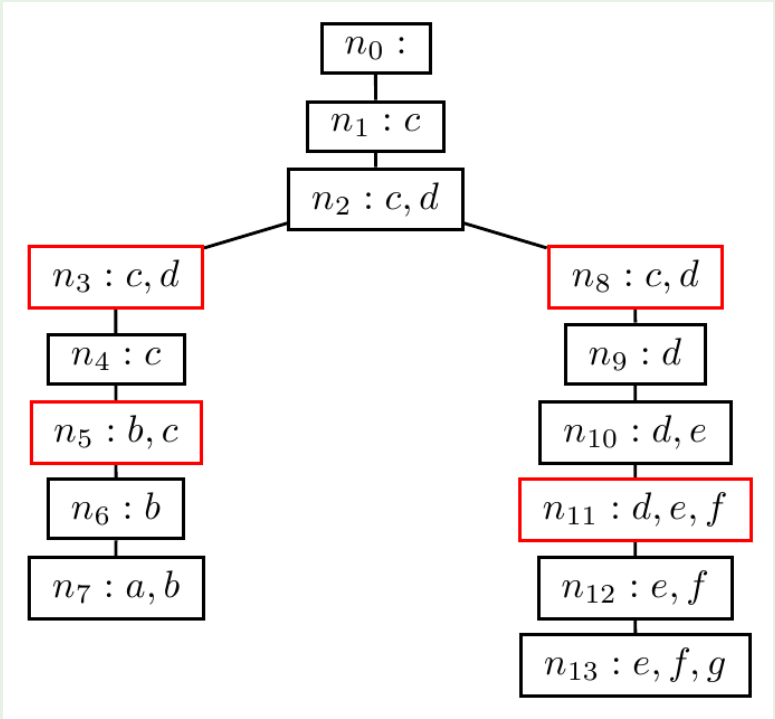
Node Types

- ROOT: root node of the tree, with empty bag

Each node is eiter a

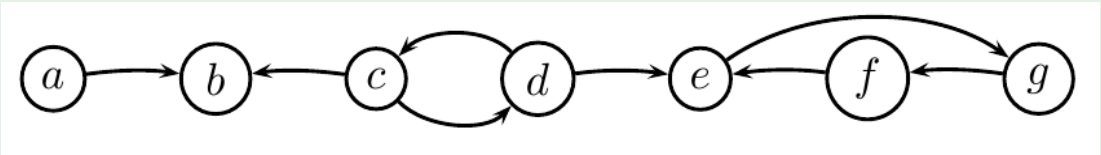
- LEAF: leaf node of the tree
- FORGET: eliminates one argument of the successor
- **INSERT:** adds an argument
- JOIN: combines two nodes, the successors bags coincide

Nice Tree-Decomposition



Nice Tree-Decomposition

Argumentation Framework



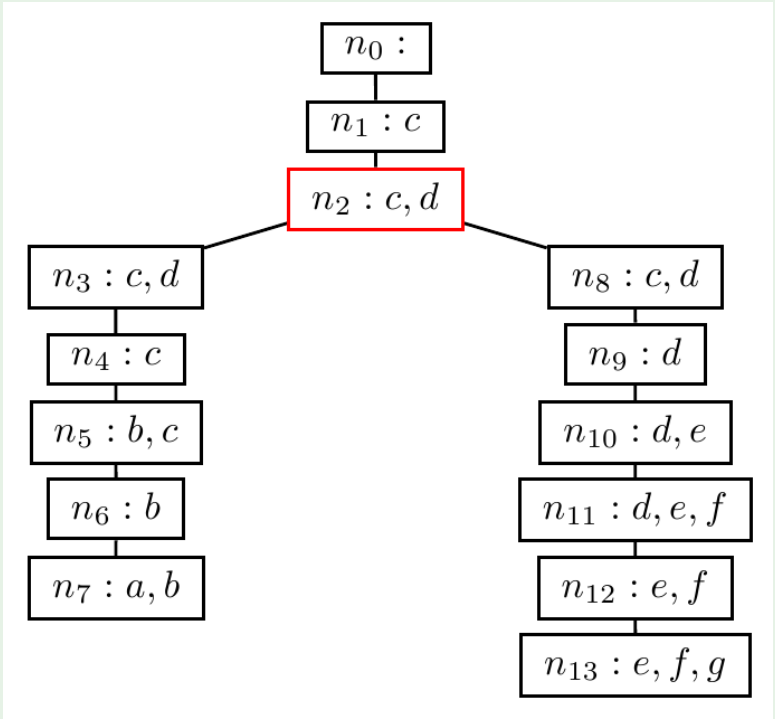
Node Types

- ROOT: root node of the tree, with empty bag

Each node is eiter a

- LEAF: leaf node of the tree
- FORGET: eliminates one argument of the successor
- INSERT: adds an argument
- **JOIN:** combines two nodes, the successors bags coincide

Nice Tree-Decomposition



Dynamic Programming

Basic Ideas:

- Compute locally admissible sets for each bag with a bottom-up algorithm on the tree-decomposition
  - For a bag t we only store information about nodes in X_t
  - The information about the "forgotten" nodes is implicitly encoded
- The results for the entire problem can be read of the root

Dynamic Programming

Basic Ideas:

- Compute locally admissible sets for each bag with a bottom-up algorithm on the tree-decomposition
  - For a bag t we only store information about nodes in X_t
  - The information about the "forgotten" nodes is implicitly encoded
- The results for the entire problem can be read of the root

Bag - Colorings

A coloring for a bag is a function $C_t : X_t \to \{in, out, att, def\}$. A coloring corresponds to an locally admissible set S in the following way:

$$x \in X_t : C(x) = \begin{cases} in & iff\ x \in S \\ out & iff\ x \notin S \wedge x \not\rightarrowtail S \wedge S \not\rightarrowtail x \\ att & iff\ x \notin S \wedge x \rightarrowtail S \wedge S \not\rightarrowtail x \\ def & iff\ x \notin S \wedge S \rightarrowtail x \end{cases}$$

DP - Leaf-Node

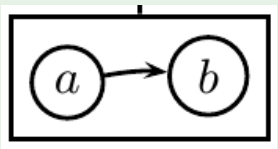
Leaf Nodes

Compute all conflict-free sets over X_t.

As there are no forgotten arguments the conflict-free sets and locally admissible sets coincide.

Tree Decomposition

$n_7 : \{a, b\}$



DP - Leaf-Node

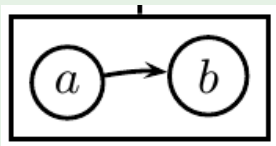
Leaf Nodes

Compute all conflict-free sets over X_t.

As there are no forgotten arguments the conflict-free sets and locally admissible sets coincide.

Tree Decomposition

$n_7 : \{a, b\}$



Colorings for n_7

There are three conflict-free sets $\{a\}, \{b\}, \emptyset$

a	b	#
in	*def*	1
att	*in*	1
out	*out*	1

DP - Forget-Node

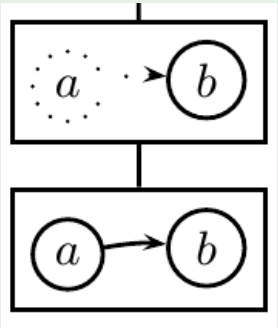
Forget-Node for argument x

Eliminate all colorings C with $C(x) = att$

Remove the variable x

Tree Decomposition

$n_7 : \{a, b\} \rightarrow n_6 : \{b\}$



DP - Forget-Node

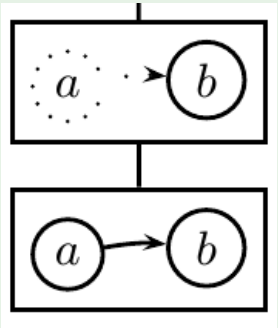
Forget-Node for argument x

Eliminate all colorings C with $C(x) = att$

Remove the variable x

Tree Decomposition

$n_7 : \{a, b\} \rightarrow n_6 : \{b\}$



Colorings for n_7

a	b	#
in	*def*	1
att	*in*	1
out	*out*	1

Colorings for n_6

b	#
def	1
out	1

DP - Insert-Node

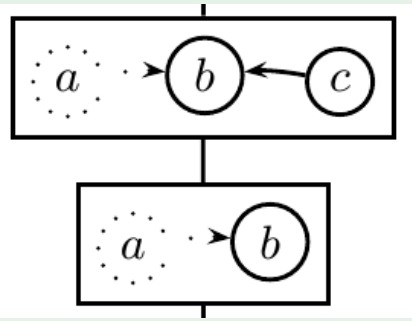
Insert-Node for argument x

A coloring C may create two colorings:

1) C extended by $C(x) \in \{out, att, def\}$
2) C extended by $C(x) = in$
   (if $[C] \cup \{x\}$ it is conflict-free)

Tree Decomposition

$n_6 : \{b\} \rightarrow n_5 : \{b, c\}$



DP - Insert-Node

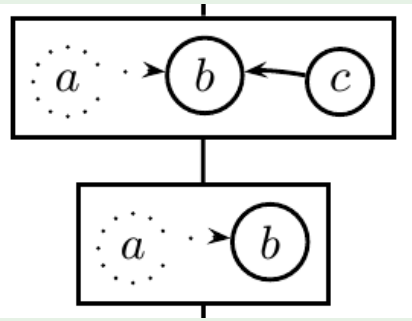
Insert-Node for argument x

A coloring C may create two colorings:

1) C extended by $C(x) \in \{out, att, def\}$
2) C extended by $C(x) = in$
   (if $[C] \cup \{x\}$ it is conflict-free)

Tree Decomposition

$n_6 : \{b\} \rightarrow n_5 : \{b, c\}$



Colorings for n_6

b	#
def	1
out	1

Colorings for n_5

b	c	#
def	*out*	1
def	*in*	1
out	*out*	1
def	*in*	1

DP - Insert-Node

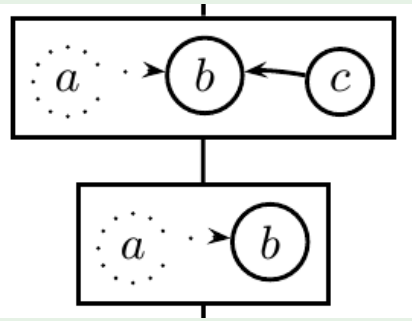
Insert-Node for argument x

A coloring C may create two colorings:

1) C extended by $C(x) \in \{out, att, def\}$
2) C extended by $C(x) = in$
   (if $[C] \cup \{x\}$ it is conflict-free)

Tree Decomposition

$n_6 : \{b\} \rightarrow n_5 : \{b, c\}$



Colorings for n_6

b	#
def	1
out	1

Colorings for n_5

b	c	#
def	*in*	2
def	*out*	1
out	*out*	1

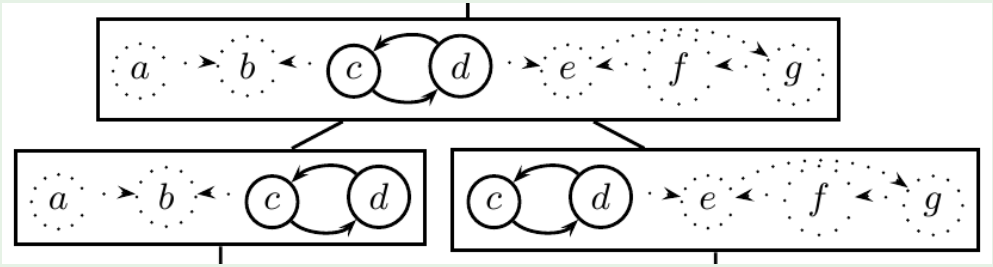
DP - Join-Node

Join-Node

Combine the colorings of the child-nodes that map the same arguments to *in*

Tree Decomposition

$n_3 : \{c, d\}$; $n_8 : \{c, d\} \rightarrow n_2 : \{c, d\}$



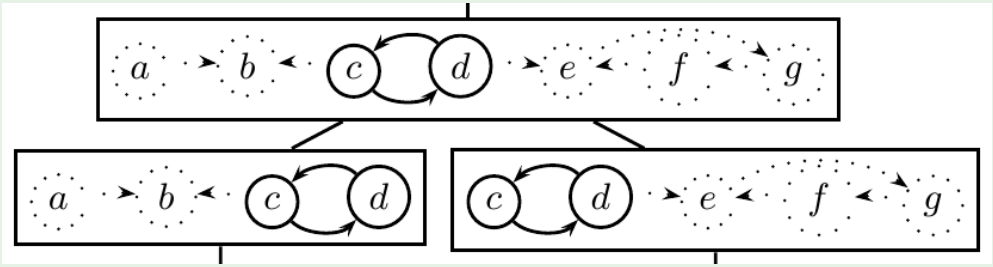
DP - Join-Node

Join-Node

Combine the colorings of the child-nodes that map the same arguments to *in*

Tree Decomposition

$n_3 : \{c, d\}$; $n_8 : \{c, d\} \rightarrow n_2 : \{c, d\}$



Colorings for n_3, n_8

n_3:

c	d	#
in	*def*	2
def	*in*	2
out	*out*	2

n_8:

c	d	#
in	*def*	1
def	*in*	2
out	*out*	1

Colorings for n_2

c	d	#
in	*def*	2
def	*in*	4
out	*out*	2

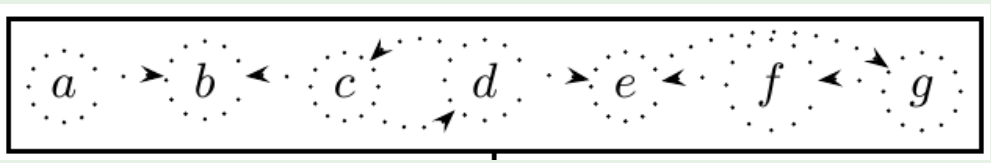
DP - Root-Node

Root-Node

As there are no visible nodes in the root, locally admissible sets and admissible sets of coincide.

Tree Decomposition

$n_0 : \{\}$



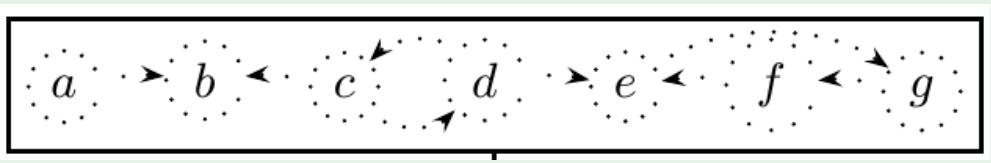
DP - Root-Node

Root-Node

As there are no visible nodes in the root, locally admissible sets and admissible sets of coincide.

Tree Decomposition

$n_0 : \{\}$



Colorings for n_0

–	#
ϵ	8

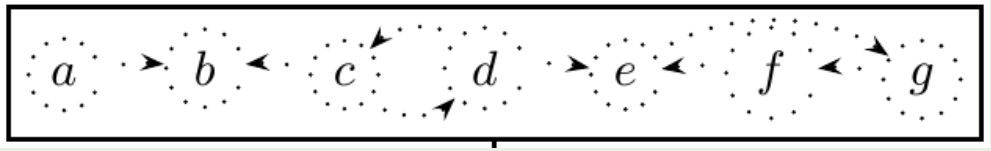
DP - Root-Node

Root-Node

As there are no visible nodes in the root, locally admissible sets and admissible sets of coincide.

Tree Decomposition

$n_0 : \{\}$



Colorings for n_0

–	#
ϵ	8

Credulous Acceptance

For credulous acceptance of an argument x we only consider colorings C with $C(x) = in$ (for bags X_t with $x \in X_t$).
Then x is credulously accepted iff there is a coloring for the root.

Complexity

Complexity

Given an AF of tree-width w and an argument x, our algorithm decides if x is credulously accepted in time $O(f(w) \cdot |AF|)$.

Complexity

Complexity

Given an AF of tree-width w and an argument x, our algorithm decides if x is credulously accepted in time $O(f(w) \cdot |AF|)$.

New Results via Extensions of our DP-Algorithm

This algorithms can be extended for

- Computing extensions (with linear delay)
- Counting extensions

Complexity

Complexity

Given an AF of tree-width w and an argument x, our algorithm decides if x is credulously accepted in time $O(f(w) \cdot |AF|)$.

New Results via Extensions of our DP-Algorithm

This algorithms can be extended for

- Computing extensions (with linear delay)
- Counting extensions

Skeptical Acceptance

By extending the concept of colorings to characterise preferred extension we get a similar algorithm for skeptical reasoning.

Conclusion

Main Contributions of the paper:

- Hardness results for AFs of bounded cycle-rank
  $\hookrightarrow$ Hardness for directed tree-w., directed path-w., DAG-w., Kelly-w.
- Fixed-parameter tractable algorithms for reasoning in AFs of bounded tree-width
  - Credulous / Skeptical Reasoning w.r.t. preferred semantics
- The techniques presented for preferred semantics are prototypical, i.e. can be easily applied to several other semantics

Conclusion

Main Contributions of the paper:

- Hardness results for AFs of bounded cycle-rank
  ↪ Hardness for directed tree-w., directed path-w., DAG-w., Kelly-w.
- Fixed-parameter tractable algorithms for reasoning in AFs of bounded tree-width
  - Credulous / Skeptical Reasoning w.r.t. preferred semantics
- The techniques presented for preferred semantics are prototypical, i.e. can be easily applied to several other semantics

Future and Ongoing Work:

- Implementation of these algorithms
- Identifying larger tractable fragments (e.g. clique-width)
  ↪ Developing fixed-parameter tractable algorithms