TECHNISCHE UNIVERSITÄT DRESDEN

Artificial Intelligence, Computational Logic

SEMINAR ABSTRACT ARGUMENTATION

Computational Complexity of Abstract Argumentation

*slides adapted from Stefan Woltran's lecture on Abstract Argumentation

Sarah Gaggl

Dresden, 8th November 2013

DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

Outline

Computational Complexity Theory

. . . studies the asymptotic worst case complexity of problems.

That is Computational Complexity Theory addresses the

- costs to solve a problem (and not the runtime of an algorithm).
- computational costs, i.e. how hard is it for a computer to solve the problem.
- asymptotic behavior, i.e. how the costs scale with the size of the problem instances.
- worst case behavior, i.e. the costs to solve the hardest instances of a problem.

Methodology of CT

To show that a problem A is at most as hard as a problem B, (denoted as $A \leq^R B$) one gives a reduction R from A to B satisfying:

Methodology of CT

To show that a problem A is at most as hard as a problem B, (denoted as $A \leq^R B$) one gives a reduction R from A to B satisfying:

- If I is an instance of A then $R(I)$ is an instance of B.
- $A(I) = B(R(I))$, e.g. I is a yes instance of A iff $R(I)$ is a yes instance of B
- R can be efficiently computed (i.e. in polynomial time).

Complexity classes: Class of problems that can be solved within some computational limits.

A problem A is a complete problem for a complexity class, if A is in the class and each problem B in the class can be reduced to A.
So the complete problems w.r.t. a complexity class form a equivalence class w.r.t. reducibility.

Some Complexity Classes

We only consider decision problems, i.e. problems that can be answered by either yes or no.

Polynomial Time P: Problems that can be solved within polynomial time.

Some Complexity Classes

We only consider decision problems, i.e. problems that can be answered by either yes or no.

Polynomial Time P: Problems that can be solved within polynomial time.

Complete problem: Given a boolean circuit and values for the input gates, deciding whether a given gate results true or false.

Some Complexity Classes

We only consider decision problems, i.e. problems that can be answered by either yes or no.

Polynomial Time P: Problems that can be solved within polynomial time.

Complete problem: Given a boolean circuit and values for the input gates, deciding whether a given gate results true or false.

Logarithmic Space L: Problems that can be solved within logarithmic space and polynomial time.

Some Complexity Classes

We only consider decision problems, i.e. problems that can be answered by either yes or no.

Polynomial Time P: Problems that can be solved within polynomial time.

Complete problem: Given a boolean circuit and values for the input gates, deciding whether a given gate results true or false.

Logarithmic Space L: Problems that can be solved within logarithmic space and polynomial time.

Typical problem: Does there exists a path between two vertices in a given undirected graph.

Some Complexity Classes

We only consider decision problems, i.e. problems that can be answered by either yes or no.

Polynomial Time P: Problems that can be solved within polynomial time.

Complete problem: Given a boolean circuit and values for the input gates, deciding whether a given gate results true or false.

Logarithmic Space L: Problems that can be solved within logarithmic space and polynomial time.

Typical problem: Does there exists a path between two vertices in a given undirected graph.

Non-Deterministic Polynomial Time NP: Problems that can be decided by an non-deterministic algorithm in polynomial time such that the overall answer is yes iff at least one possible computation results yes.

Some Complexity Classes

We only consider decision problems, i.e. problems that can be answered by either yes or no.

Polynomial Time P: Problems that can be solved within polynomial time.

Complete problem: Given a boolean circuit and values for the input gates, deciding whether a given gate results true or false.

Logarithmic Space L: Problems that can be solved within logarithmic space and polynomial time.

Typical problem: Does there exists a path between two vertices in a given undirected graph.

Non-Deterministic Polynomial Time NP: Problems that can be decided by an non-deterministic algorithm in polynomial time such that the overall answer is yes iff at least one possible computation results yes.

Complete problem: Deciding whether a given propositional formula is satisfiable (SAT).

Some Complexity Classes

co-NP: Problems that can be decided by an non-deterministic algorithm in polynomial time such that the overall answer is yes iff all possible computation result yes.

Some Complexity Classes

co-NP: Problems that can be decided by an non-deterministic algorithm in polynomial time such that the overall answer is yes iff all possible computation result yes.

Complete problem: Deciding whether a given propositional formula it is unsatisfiable (UNSAT).

Some Complexity Classes

co-NP: Problems that can be decided by an non-deterministic algorithm in polynomial time such that the overall answer is yes iff all possible computation result yes.

Complete problem: Deciding whether a given propositional formula it is unsatisfiable (UNSAT).

Some classes from the polynomial hierarchy:

the class Σ_2^p:

Complete problem: Deciding whether a given quantified boolean formula (QBF) of the form $\exists Y \forall Z \varphi(Y, Z)$ is valid.

Some Complexity Classes

co-NP: Problems that can be decided by an non-deterministic algorithm in polynomial time such that the overall answer is yes iff all possible computation result yes.

Complete problem: Deciding whether a given propositional formula it is unsatisfiable (UNSAT).

Some classes from the polynomial hierarchy:

the class Σ_2^p:

Complete problem: Deciding whether a given quantified boolean formula (QBF) of the form $\exists Y \forall Z \varphi(Y, Z)$ is valid.

the class Π_2^p:

Complete problem: Deciding whether a given QBF of the form $\forall Y \exists Z \varphi(Y, Z)$ is valid.

Some Complexity Classes

co-NP: Problems that can be decided by an non-deterministic algorithm in polynomial time such that the overall answer is yes iff all possible computation result yes.

Complete problem: Deciding whether a given propositional formula it is unsatisfiable (UNSAT).

Some classes from the polynomial hierarchy:

the class Σ_2^p:

Complete problem: Deciding whether a given quantified boolean formula (QBF) of the form $\exists Y \forall Z \varphi(Y, Z)$ is valid.

the class Π_2^p:

Complete problem: Deciding whether a given QBF of the form $\forall Y \exists Z \varphi(Y, Z)$ is valid.

the class Θ_2^p:

Complete problem: Given several propositional formulae. Deciding whether the number of satisfiable formulae is odd/even.

Some Complexity Classes

co-NP: Problems that can be decided by an non-deterministic algorithm in polynomial time such that the overall answer is yes iff all possible computation result yes.

Complete problem: Deciding whether a given propositional formula it is unsatisfiable (UNSAT).

Some classes from the polynomial hierarchy:

the class Σ_2^p:

Complete problem: Deciding whether a given quantified boolean formula (QBF) of the form $\exists Y \forall Z \varphi(Y, Z)$ is valid.

the class Π_2^p:

Complete problem: Deciding whether a given QBF of the form $\forall Y \exists Z \varphi(Y, Z)$ is valid.

the class Θ_2^p:

Complete problem: Given several propositional formulae. Deciding whether the number of satisfiable formulae is odd/even.

Relating Complexity classes

The following relations hold for the introduced complexity classes:

$$\mathsf{L} \subseteq \mathsf{P} \subseteq \begin{matrix}\mathsf{NP}\\ \text{co-NP}\end{matrix} \subseteq \Theta_2^p \subseteq \begin{matrix}\Sigma_2^p\\ \Pi_2^p\end{matrix}$$

We consider problems in L and P to be computationally tractable and problems complete one of the classes NP, co-NP, Θ_2^p, Σ_2^p, Π_2^p as computationally intractable.

Why doing Complexity Analysis?

- To understand the Computational Costs that underlie a certain reasoning problem. Such that we know the best worst-case behaviour an algorithm can provide (Complexity Theoretic View).

Why doing Complexity Analysis?

- To understand the Computational Costs that underlie a certain reasoning problem. Such that we know the best worst-case behaviour an algorithm can provide (Complexity Theoretic View).
- Understand the Expressibility
  If a formalism has too low complexity, there are a lot of interesting problems of higher complexity that can not be expressed within the formalism (Knowledge-Representation View).

Why doing Complexity Analysis?

- To understand the Computational Costs that underlie a certain reasoning problem. Such that we know the best worst-case behaviour an algorithm can provide (Complexity Theoretic View).
- Understand the Expressibility
  If a formalism has too low complexity, there are a lot of interesting problems of higher complexity that can not be expressed within the formalism (Knowledge-Representation View).
- For applying the Translation-Approach, i.e. encode my problem in other formalisms. The target formalism must be at least of the same complexity. To avoid unnecessary computational effort one would prefer a (fragment of a) formalism providing a complexity close to the original problem (Practitioners View).

Decision Problems on AFs

Credulous Acceptance

Cred$_\sigma$: Given AF $F = (A, R)$ and $a \in A$; is a contained in at least one σ-extension of F?

Skeptical Acceptance

Skept$_\sigma$: Given AF $F = (A, R)$ and $a \in A$; is a contained in every σ-extension of F?

If no extension exists then all arguments are skeptically accepted and no argument is credulously accepted[1].

[1] This is only relevant for stable semantics.

Decision Problems on AFs

Credulous Acceptance

Cred_σ: Given AF $F = (A, R)$ and $a \in A$; is a contained in at least one σ-extension of F?

Skeptical Acceptance

Skept_σ: Given AF $F = (A, R)$ and $a \in A$; is a contained in every σ-extension of F?

If no extension exists then all arguments are skeptically accepted and no argument is credulously accepted[1].

Hence we are also interested in the following problem:

Skeptically and Credulously accepted

Skept'_σ: Given AF $F = (A, R)$ and $a \in A$; is a contained in every and at least one σ-extension of F?

[1] This is only relevant for stable semantics.

Further Decision Problems

Verifying an extension

Ver$_\sigma$: Given AF $F = (A, R)$ and $S \subseteq A$; is S a σ-extension of F?

Further Decision Problems

Verifying an extension

Ver$_\sigma$: Given AF $F = (A, R)$ and $S \subseteq A$; is S a σ-extension of F?

Does there exist an extension?

Exists$_\sigma$: Given AF $F = (A, R)$; Does there exist a σ-extension for F?

Further Decision Problems

Verifying an extension

Ver$_\sigma$: Given AF $F = (A, R)$ and $S \subseteq A$; is S a σ-extension of F?

Does there exist an extension?

Exists$_\sigma$: Given AF $F = (A, R)$; Does there exist a σ-extension for F?

Does there exist a nonempty extensions?

Exists$_\sigma^{\neg\emptyset}$: Does there exist a non-empty σ-extension for F?

Computationally Easy Problems

Given an Argumentation Framework $F = (A, R)$, an argument $a \in A$ and a set $E \subseteq A$.

- Cred$_{cf}$: $\exists S \in cf(F) : a \in S$?

Computationally Easy Problems

Given an Argumentation Framework $F = (A, R)$, an argument $a \in A$ and a set $E \subseteq A$.

- Cred_{cf}: $\exists S \in cf(F) : a \in S$?
  Test whether $(a, a) \in R$ or not.

Computationally Easy Problems

Given an Argumentation Framework $F = (A, R)$, an argument $a \in A$ and a set $E \subseteq A$.

- Cred$_{cf}$: $\exists S \in cf(F) : a \in S$?
  Test whether $(a, a) \in R$ or not.
- Skept$_{cf}$: $\forall S \in cf(F) : a \in S$?

Computationally Easy Problems

Given an Argumentation Framework $F = (A, R)$, an argument $a \in A$ and a set $E \subseteq A$.

- Cred_{cf}: $\exists S \in cf(F) : a \in S$?
  Test whether $(a, a) \in R$ or not.
- Skept_{cf}: $\forall S \in cf(F) : a \in S$?
  Trivially false, as $\emptyset \in cf(F)$

Computationally Easy Problems

Given an Argumentation Framework $F = (A, R)$, an argument $a \in A$ and a set $E \subseteq A$.

- Cred_{cf}: $\exists S \in cf(F) : a \in S$?
  Test whether $(a, a) \in R$ or not.
- Skept_{cf}: $\forall S \in cf(F) : a \in S$?
  Trivially false, as $\emptyset \in cf(F)$
- $\text{Exists}_{cf}^{\neg\emptyset}$: $\exists S \in cf(F) : S \neq \emptyset$?

Computationally Easy Problems

Given an Argumentation Framework $F = (A, R)$, an argument $a \in A$ and a set $E \subseteq A$.

- Cred_{cf}: $\exists S \in cf(F) : a \in S$?
  Test whether $(a, a) \in R$ or not.
- Skept_{cf}: $\forall S \in cf(F) : a \in S$?
  Trivially false, as $\emptyset \in cf(F)$
- $\text{Exists}_{cf}^{\neg\emptyset}$: $\exists S \in cf(F) : S \neq \emptyset$?
  Test whether there is an argument b such that $(b, b) \notin R$.

Computationally Easy Problems

Given an Argumentation Framework $F = (A, R)$, an argument $a \in A$ and a set $E \subseteq A$.

- Cred_{cf}: $\exists S \in cf(F) : a \in S$?
  Test whether $(a, a) \in R$ or not.
- Skept_{cf}: $\forall S \in cf(F) : a \in S$?
  Trivially false, as $\emptyset \in cf(F)$
- $\text{Exists}_{cf}^{\neg\emptyset}$: $\exists S \in cf(F) : S \neq \emptyset$?
  Test whether there is an argument b such that $(b, b) \notin R$.
- Ver_{cf}: $E \in cf(F)$?
  Test whether there are $b, c \in E$ such that $(b, c) \in R$

Computationally Easy Problems

Given an Argumentation Framework $F = (A, R)$, an argument $a \in A$ and a set $E \subseteq A$.

- Cred_{cf}: $\exists S \in cf(F) : a \in S$?
  Test whether $(a, a) \in R$ or not.
- Skept_{cf}: $\forall S \in cf(F) : a \in S$?
  Trivially false, as $\emptyset \in cf(F)$
- $\text{Exists}_{cf}^{\neg\emptyset}$: $\exists S \in cf(F) : S \neq \emptyset$?
  Test whether there is an argument b such that $(b, b) \notin R$.
- Ver_{cf}: $E \in cf(F)$?
  Test whether there are $b, c \in E$ such that $(b, c) \in R$
- Cred_{naive}: $\exists S \in naive(F) : a \in S$?
  $\text{Cred}_{naive} = \text{Cred}_{cf}$

Computationally Easy Problems

Given an Argumentation Framework $F = (A, R)$, an argument $a \in A$ and a set $E \subseteq A$.

- Cred_{cf}: $\exists S \in cf(F) : a \in S$?
  Test whether $(a, a) \in R$ or not.
- Skept_{cf}: $\forall S \in cf(F) : a \in S$?
  Trivially false, as $\emptyset \in cf(F)$
- $\text{Exists}_{cf}^{\neg\emptyset}$: $\exists S \in cf(F) : S \neq \emptyset$?
  Test whether there is an argument b such that $(b, b) \notin R$.
- Ver_{cf}: $E \in cf(F)$?
  Test whether there are $b, c \in E$ such that $(b, c) \in R$
- Cred_{naive}: $\exists S \in naive(F) : a \in S$?
  $\text{Cred}_{naive} = \text{Cred}_{cf}$
- $\text{Skept}_{ground} = \text{Cred}_{ground}$: $a \in ground(F)$?
  Compute the grounded extension $ground(F)$.

Computationally Easy Problems

Given an Argumentation Framework $F = (A, R)$, an argument $a \in A$ and a set $E \subseteq A$.

- Cred_{cf}: $\exists S \in cf(F) : a \in S$?
  Test whether $(a, a) \in R$ or not.
- Skept_{cf}: $\forall S \in cf(F) : a \in S$?
  Trivially false, as $\emptyset \in cf(F)$
- $\text{Exists}_{cf}^{\neg\emptyset}$: $\exists S \in cf(F) : S \neq \emptyset$?
  Test whether there is an argument b such that $(b, b) \notin R$.
- Ver_{cf}: $E \in cf(F)$?
  Test whether there are $b, c \in E$ such that $(b, c) \in R$
- Cred_{naive}: $\exists S \in naive(F) : a \in S$?
  $\text{Cred}_{naive} = \text{Cred}_{cf}$
- $\text{Skept}_{ground} = \text{Cred}_{ground}$: $a \in ground(F)$?
  Compute the grounded extension $ground(F)$.

Intractable problems in Abstract Argumentation

Most problems in Abstract Argumentation are computationally intractable, i.e. at least NP-hard. To show intractability for a specific reasoning problem we follow the schema given below:

Goal: Show that a reasoning problem is NP-hard.

Method: Reducing the NP-hard SAT problem to the reasoning problem.

- Consider an arbitrary CNF formula φ
- Give a reduction that maps φ to an Argumentation Framework F_φ containing an argument φ.
- Show that φ is satisfiable iff the argument φ is accepted.

Canonical Reduction

Definition

For $\varphi = \bigwedge_{i=1}^{m} l_{i1} \vee l_{i2} \vee l_{i3}$ over atoms Z, build $F_\varphi = (A_\varphi, R_\varphi)$ with

$$
\begin{aligned}
A_\varphi &= Z \cup \bar{Z} \cup \{C_1, \ldots, C_m\} \cup \{\varphi\} \\
R_\varphi &= \{(z, \bar{z}), (\bar{z}, z) \mid z \in Z\} \cup \{(C_i, \varphi) \mid i \in \{1, \ldots, m\}\} \cup \\
&\quad \{(z, C_i) \mid i \in \{1, \ldots, m\}, z \in \{l_{i1}, l_{i2}, l_{i3}\}\} \cup \\
&\quad \{(\bar{z}, C_i) \mid i \in \{1, \ldots, m\}, \neg z \in \{l_{i1}, l_{i2}, l_{i3}\}\}
\end{aligned}
$$

Canonical Reduction

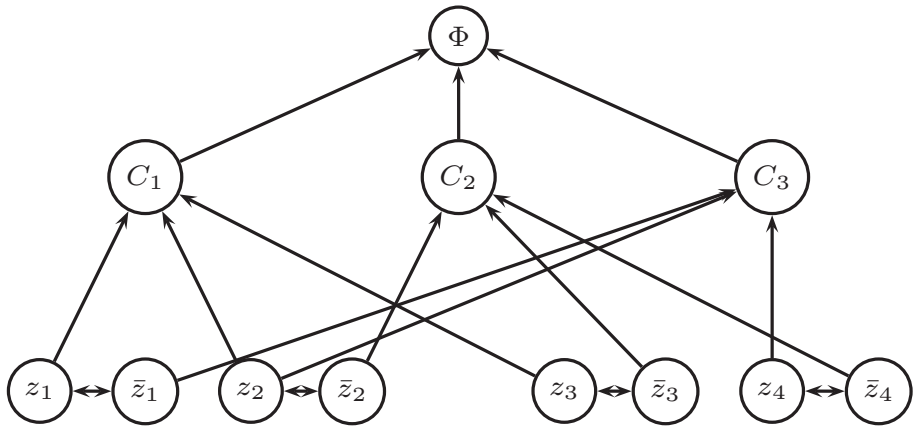
Definition

For $\varphi = \bigwedge_{i=1}^{m} l_{i1} \vee l_{i2} \vee l_{i3}$ over atoms Z, build $F_\varphi = (A_\varphi, R_\varphi)$ with

$$\begin{aligned} A_\varphi &= Z \cup \bar{Z} \cup \{C_1, \ldots, C_m\} \cup \{\varphi\} \\ R_\varphi &= \{(z, \bar{z}), (\bar{z}, z) \mid z \in Z\} \cup \{(C_i, \varphi) \mid i \in \{1, \ldots, m\}\} \cup \\ &\quad \{(z, C_i) \mid i \in \{1, \ldots, m\}, z \in \{l_{i1}, l_{i2}, l_{i3}\}\} \cup \\ &\quad \{(\bar{z}, C_i) \mid i \in \{1, \ldots, m\}, \neg z \in \{l_{i1}, l_{i2}, l_{i3}\}\} \end{aligned}$$

Example

Let $\Phi = (z_1 \vee z_2 \vee z_3) \wedge (\neg z_2 \vee \neg z_3 \vee \neg z_4) \wedge (\neg z_1 \vee z_2 \vee z_4)$.



Canonical Reduction: CNF $\Rightarrow$ AF (ctd.)

Theorem

The following statements are equivalent:

1. φ is satisfiable
2. F_φ has an admissible set containing φ
3. F_φ has a complete extension containing φ
4. F_φ has a preferred extension containing φ
5. F_φ has a stable extension containing φ

Proof.

$(1) \Leftrightarrow (5)$: blackboard $\square$

Complexity Results

Theorem

1. Cred_{stable} is NP-complete
2. Cred_{adm} is NP-complete
3. Cred_{comp} is NP-complete
4. Cred_{pref} is NP-complete

Proof.

(1) The hardness is immediate by the last theorem.
For the NP-membership we use the following guess & check algorithm:

- Guess a set $E \subseteq A$
- verify that E is stable
  - for each $a, b \in E$ check $(a, b) \notin R$
  - for each $a \in A \setminus E$ check if there exists $b \in E$ with $(b, a) \in R$

As this algorithm is in polynomial time we obtain NP-membership. □

Complexity Results

What about skeptical acceptance ?

Complexity Results

What about skeptical acceptance ?

Theorem

1. Skept$_{stable}$ is co-NP-complete.
2. Skept$_{adm}$ is computationally trivial.
3. Skept$_{comp}$ is in P.
4. Skept$_{pref}$ is co-NP-hard.

Complexity Results

What about skeptical acceptance ?

Theorem

1. Skept$_{stable}$ is co-NP-complete.
2. Skept$_{adm}$ is computationally trivial.
3. Skept$_{comp}$ is in P.
4. Skept$_{pref}$ is co-NP-hard.

Proof.

(1) & (4) using a modification of F_φ
(2) exercise □

Complexity Results

What about skeptical acceptance ?

Theorem

1. Skept$_{stable}$ is co-NP-complete.
2. Skept$_{adm}$ is computationally trivial.
3. Skept$_{comp}$ is in P.
4. Skept$_{pref}$ is co-NP-hard.

Proof.

(1) & (4) using a modification of F_φ
(2) exercise □

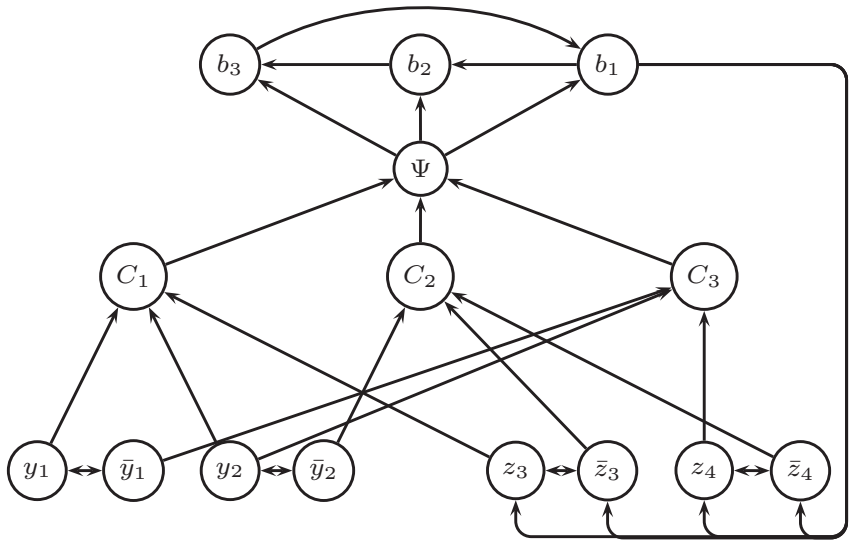
Skept$_{pref}$ is actually harder than co-NP.

Reduction for Skept$_{pref}$ (sketched)

Idea

Use a reduction from quantified Boolean formulae (QBFs) to AFs.

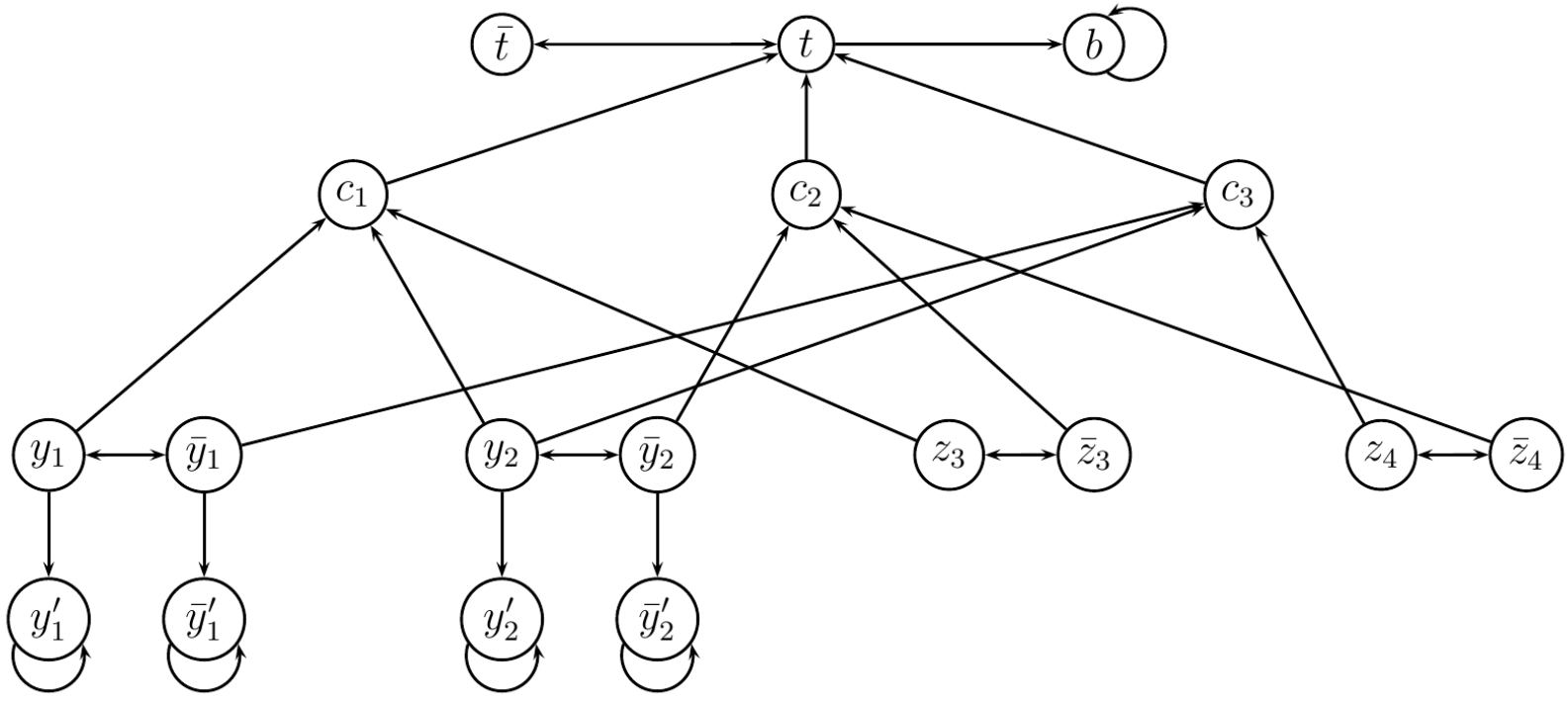
For $\forall y_1 y_2 \exists z_3 z_4 (y_1 \vee y_2 \vee z_3) \wedge (\neg y_2 \vee \neg z_3 \vee \neg z_4) \wedge (\neg y_1 \vee y_2 \vee z_4)$ use



Note that Ψ is contained in each preferred extension iff the QBF is true.

Reduction for Semi-Stable / Stage Semantics (sketched)

For $\forall y_1, y_2\ \exists z_3, z_4\ (y_1 \vee y_2 \vee z_3) \wedge (\neg y_2 \vee \neg z_3 \vee \neg z_4) \wedge (\neg y_1 \vee \neg y_2 \vee z_4)$ use



t is contained in all semi-stable extensions iff the QBF is true.

Complexity Results (Summary)

Complexity for decision problems in AFs.

σ	Cred_σ	Skept_σ
ground	P-c	P-c
naive	in L	in L
stable	NP-c	co-NP-c
adm	NP-c	trivial
comp	NP-c	P-c
pref	NP-c	Π_2^p-c

σ	Cred_σ	Skept_σ
semi	Σ_2^p-c	Π_2^p-c
stage	Σ_2^p-c	Π_2^p-c
ideal	in Θ_2^p	in Θ_2^p
eager	Π_2^p-c	Π_2^p-c
ground*	NP-c	co-NP-c
cf2	NP-c	co-NP-c

see [Baroni et al.2011, Coste-Marquis et al.2005, Dimopoulos and Torres1996, Dung1995, Dunne2008, Dunne and Bench-Capon2002, Dunne and Bench-Capon2004, Dunne and Caminada2008, Dvořák et al.2011, Dvořák and Woltran2010a, Dvořák and Woltran2010b]

Tractable Fragments

Complexity results show that many argumentation problems are computationally intractable! In order to obtain tractable fragments, imposing restrictions on the graph structure seems natural:

[2] Selfattacks, i.e. attacks $(a, a) \in R$, are prohibited

[3] Problems w.r.t. stage semantics remain hard.

Tractable Fragments

Complexity results show that many argumentation problems are computationally intractable! In order to obtain tractable fragments, imposing restrictions on the graph structure seems natural:

- acyclic AFs . . . problems become tractable (folklore)
- symmetric AFs [Coste-Marquis et al. 05][2]
- bipartite AFs [Dunne 07]
- AFs without even-length cycles [Dunne and Bench-Capon 02][3]

[2] Selfattacks, i.e. attacks $(a, a) \in R$, are prohibited

[3] Problems w.r.t. stage semantics remain hard.

Fixed-Parameter Tractability (FPT) – Motivation

Some Observations

- For intractable problems, computational costs often depend primarily 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.
- Typical parameters for graphs: tree-width and clique-width.
  - Meta-theorems allow for rather easy proofs of FPT results w.r.t. these parameters
  - Dedicated dynamic algorithms required for practical realization!

  (Recall: AFs are naturally represented as graphs)

Fixed-Parameter Tractability (FPT)

FPT is one branch in the area of Parameterized Complexity

- Downey & Fellows: Parameterized Complexity. Springer, 1999
- Flum & Grohe: Parameterized Complexity Theory. Springer, 2006
- Niedermeier: Invitation to Fixed-Parameter Algorithms. OUP, 2006

Fixed-Parameter Tractability (FPT)

FPT is one branch in the area of Parameterized Complexity

- Downey & Fellows: Parameterized Complexity. Springer, 1999
- Flum & Grohe: Parameterized Complexity Theory. Springer, 2006
- Niedermeier: Invitation to Fixed-Parameter Algorithms. OUP, 2006

Goal:

- Find a parameter k and algorithm such that each problem instance I can be decided in time $f(k) \cdot |I|^{O(1)}$ or
- prove that an given parameter does not provide such an algorithm.

Fixed-Parameter Tractability (FPT)

FPT is one branch in the area of Parameterized Complexity

- Downey & Fellows: Parameterized Complexity. Springer, 1999
- Flum & Grohe: Parameterized Complexity Theory. Springer, 2006
- Niedermeier: Invitation to Fixed-Parameter Algorithms. OUP, 2006

Goal:

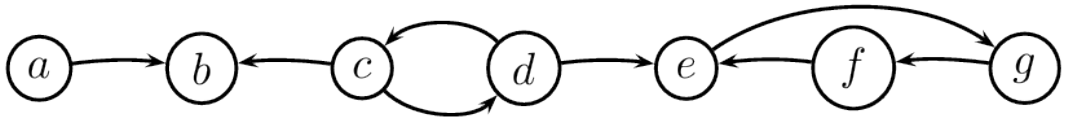
- Find a parameter k and algorithm such that each problem instance I can be decided in time $f(k) \cdot |I|^{O(1)}$ or
- prove that an given parameter does not provide such an algorithm.

Here we briefly address:

- Definition and comparison: tree-width and clique-width
- FPT results for abstract argumentation via meta-theorems [Dunne, 2007; Dvořák et al., 2012]

Tree-Width

Argumentation Framework

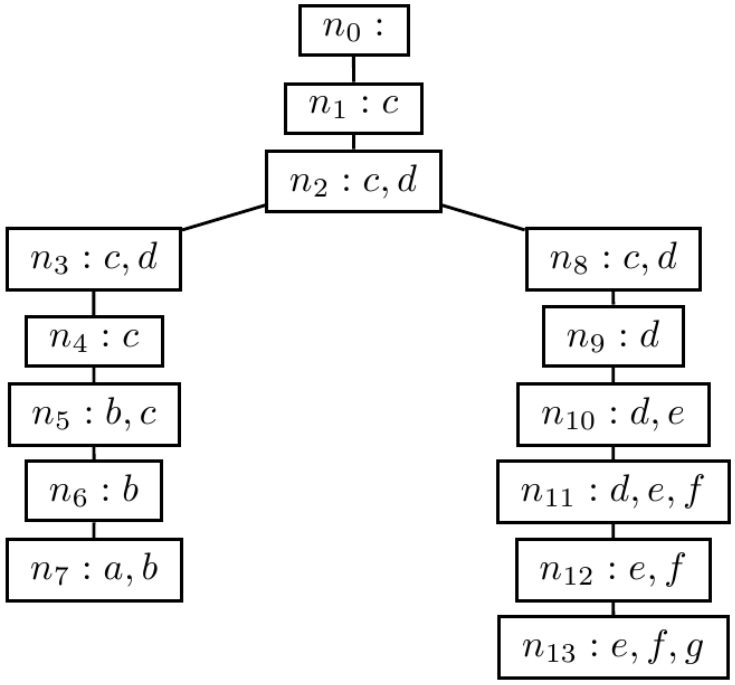


Properties

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

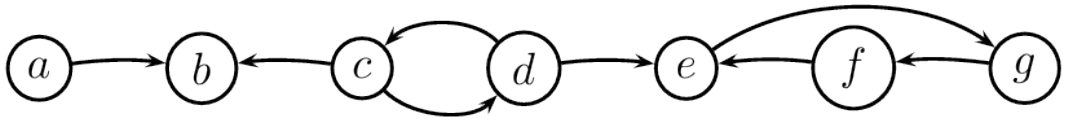
1. Each argument $a \in A$ and each attack $(b, c) \in R$ is contained in at least one bag
2. Bags containing the same argument are connected

Tree-Decomposition:



Tree-Width

Argumentation Framework

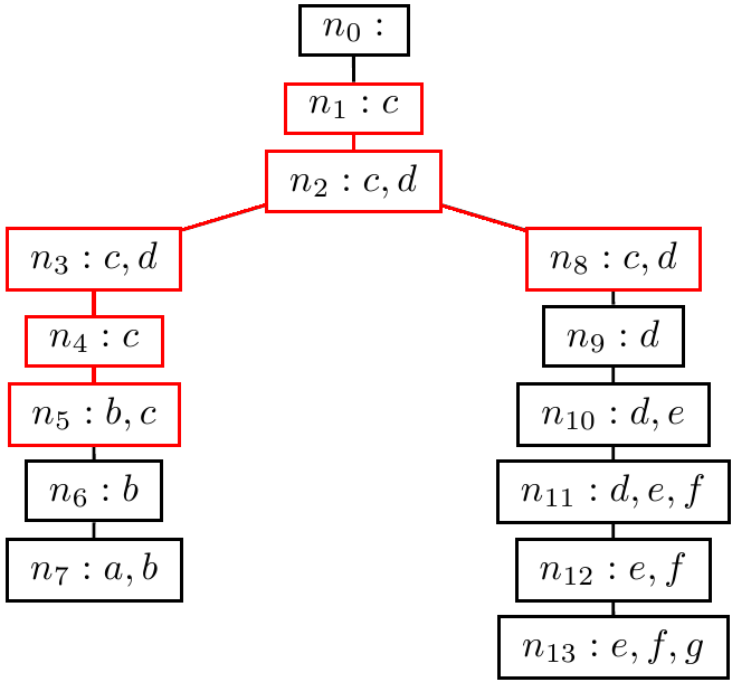


Properties

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

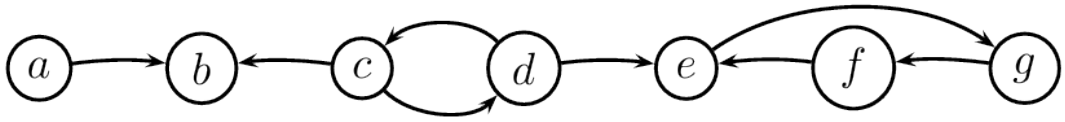
1. Each argument $a \in A$ and each attack $(b, c) \in R$ is contained in at least one bag
2. Bags containing the same argument are connected

Tree-Decomposition:



Tree-Width

Argumentation Framework

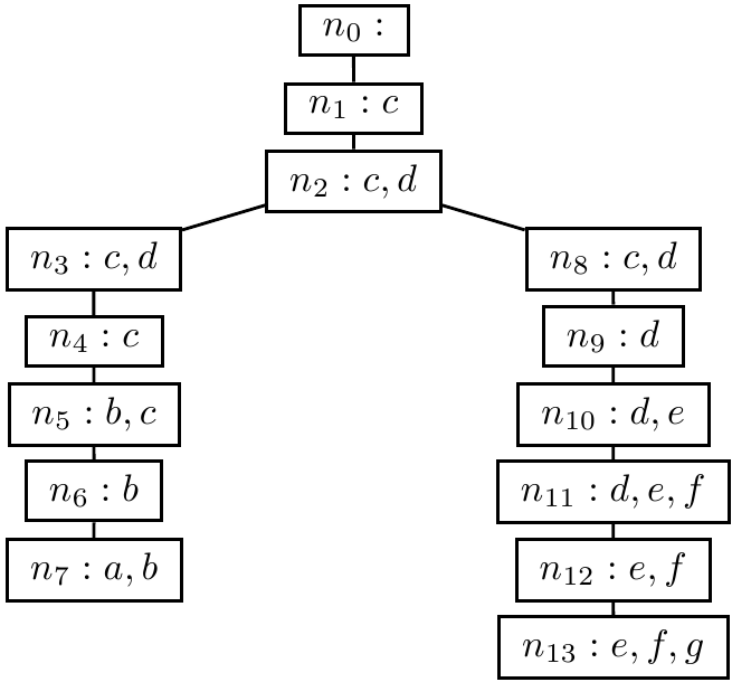


Definition

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

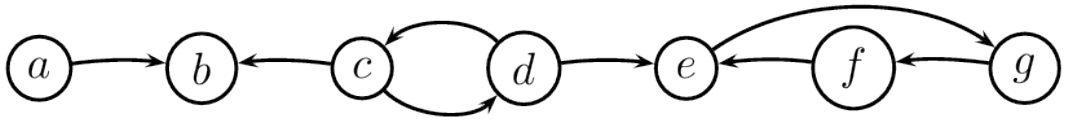
tree-width of an AF is the minimum width over all possible tree-decompositions

Tree-Decomposition:



Tree-Width

Argumentation Framework



Definition

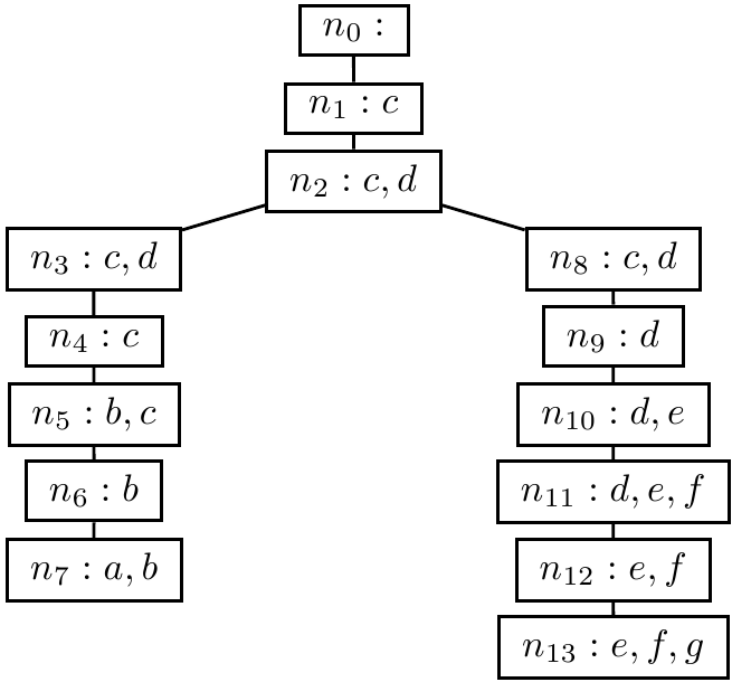
width of a tree-decomposition is the size off the largest bag -1

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

Example

AF has tree-width 2

Tree-Decomposition:



Tree-Width (ctd)

DYNPARTIX System

- Implements dynamic algorithms on tree-decompositions for several argumentation semanitcs following [Dvořák et al., 2012].
- Available under

  http://www.dbai.tuwien.ac.at/research/project/argumentation/dynpartix/

Clique-Width

Operations

We allow the following operations on labeled AFs:

- Node introduction (denote by $i(v)$);
- Disjoint union (denoted by $\oplus$);
- Relabeling: changing all labels i to j (denoted by $\rho_{i \to j}$);
- Edge insertion: connecting all vertices labeled by i with all vertices labeled by j (denoted by $\eta_{i,j}$); already existing edges are not doubled.

Note: $\rho_{i \to j}$ and $\eta_{i,j}$ treat equally labeled arguments in the same way.

Clique-Width

Operations

We allow the following operations on labeled AFs:

- Node introduction (denote by $i(v)$);
- Disjoint union (denoted by $\oplus$);
- Relabeling: changing all labels i to j (denoted by $\rho_{i \to j}$);
- Edge insertion: connecting all vertices labeled by i with all vertices labeled by j (denoted by $\eta_{i,j}$); already existing edges are not doubled.

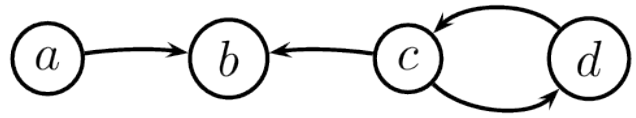
Note: $\rho_{i \to j}$ and $\eta_{i,j}$ treat equally labeled arguments in the same way.

Definition

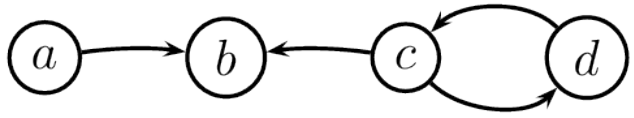
A term composed of $i(v)$, $\oplus$, $\rho_{i \to j}$, and $\eta_{i,j}$ using labels from $\{1, \ldots, k\}$ is called k-expression.
The clique-width of an AF F is the smallest integer k such that F can be defined by a k-expression.

Clique-Width – Example

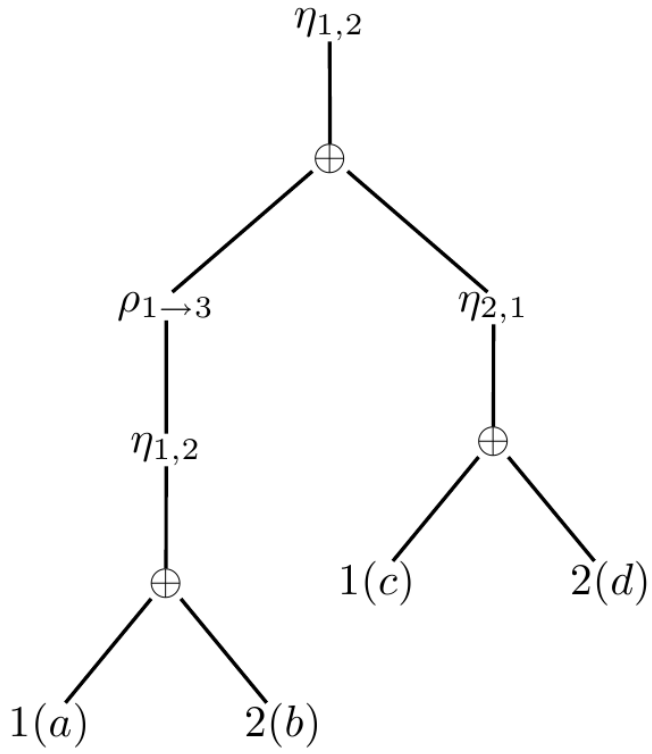
k-expression for the following AF?



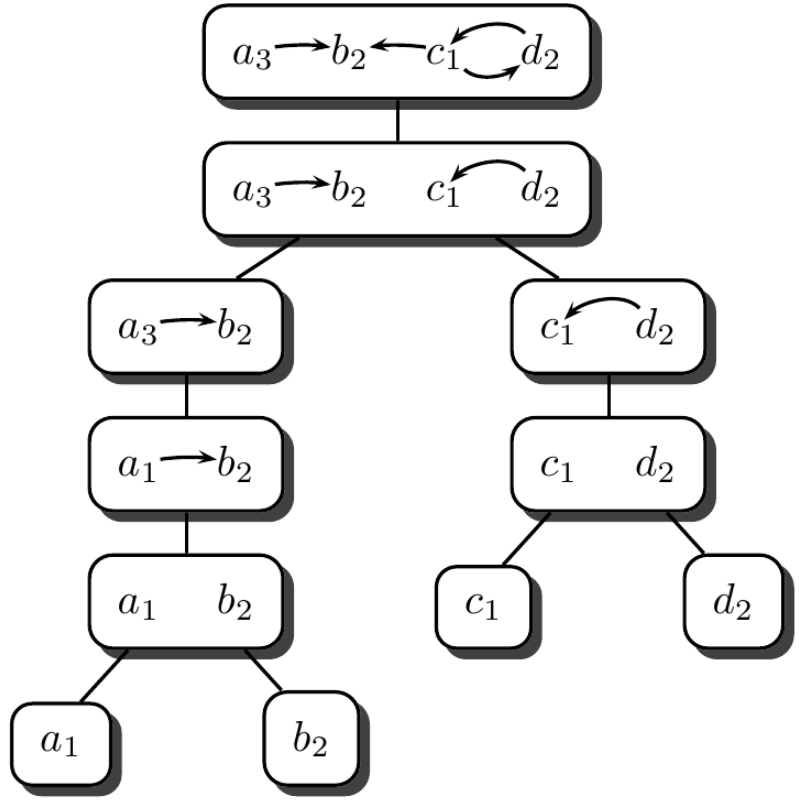
Clique-Width – Example



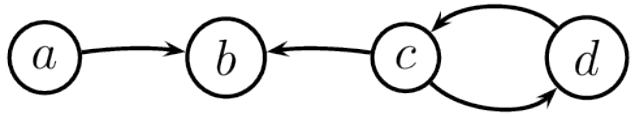
Parse Tree



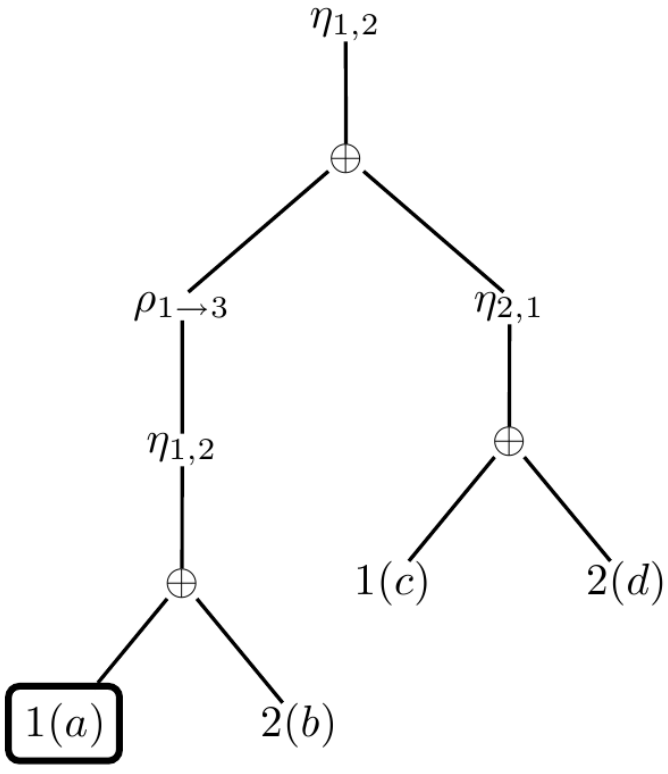
associated AFs



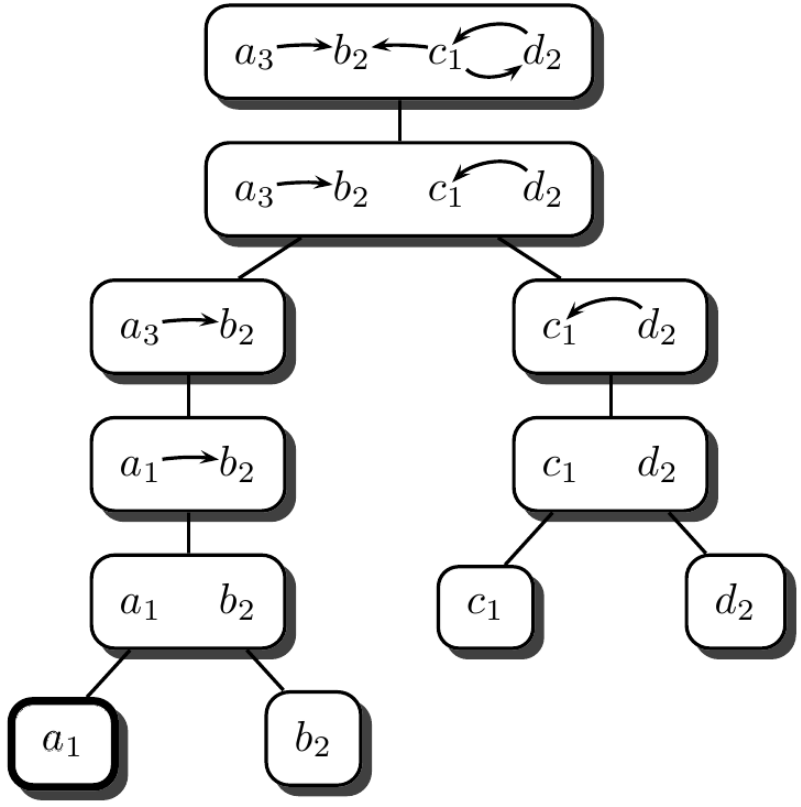
Clique-Width – Example



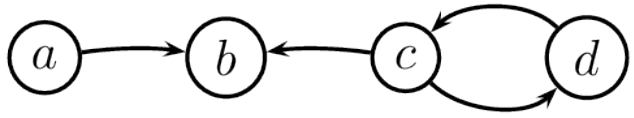
Parse Tree



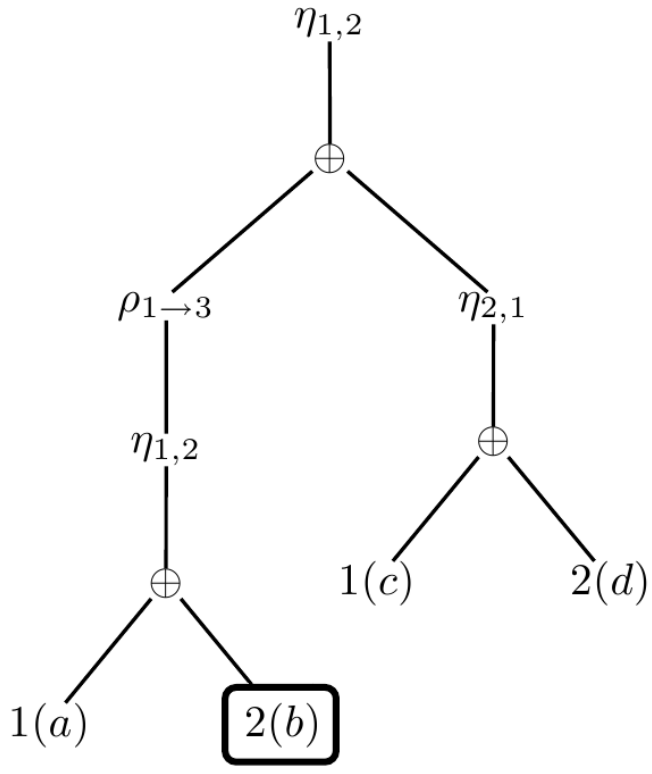
associated AFs



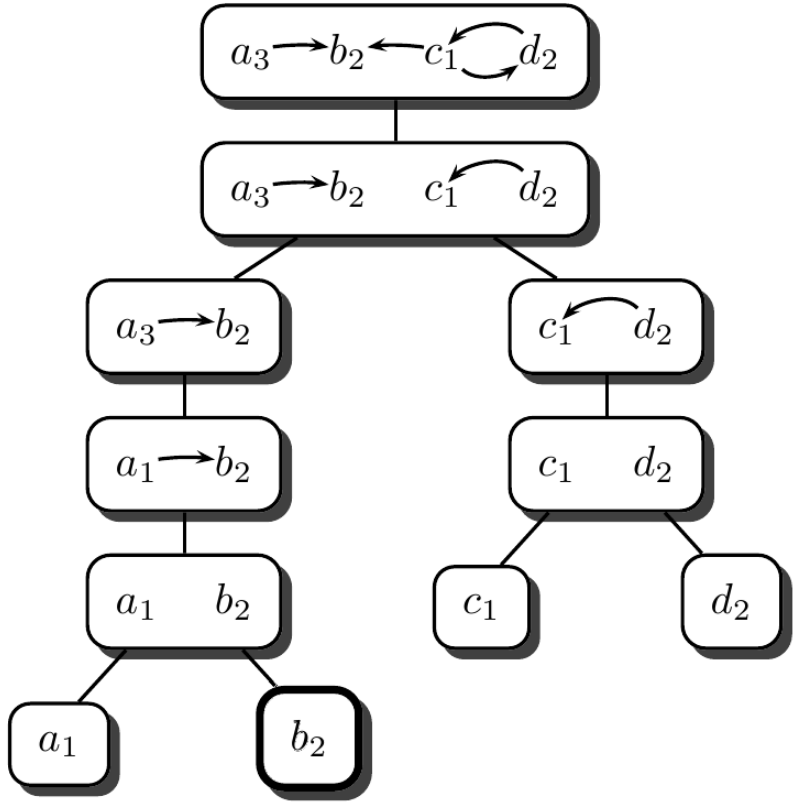
Clique-Width – Example



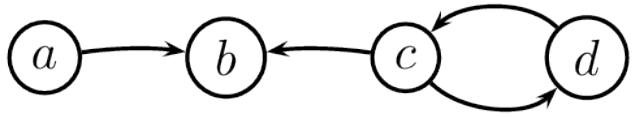
Parse Tree



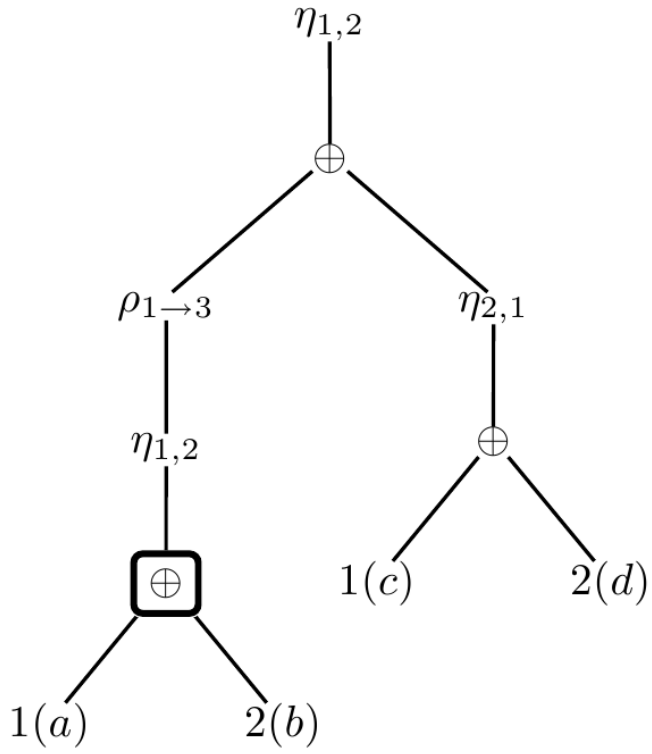
associated AFs



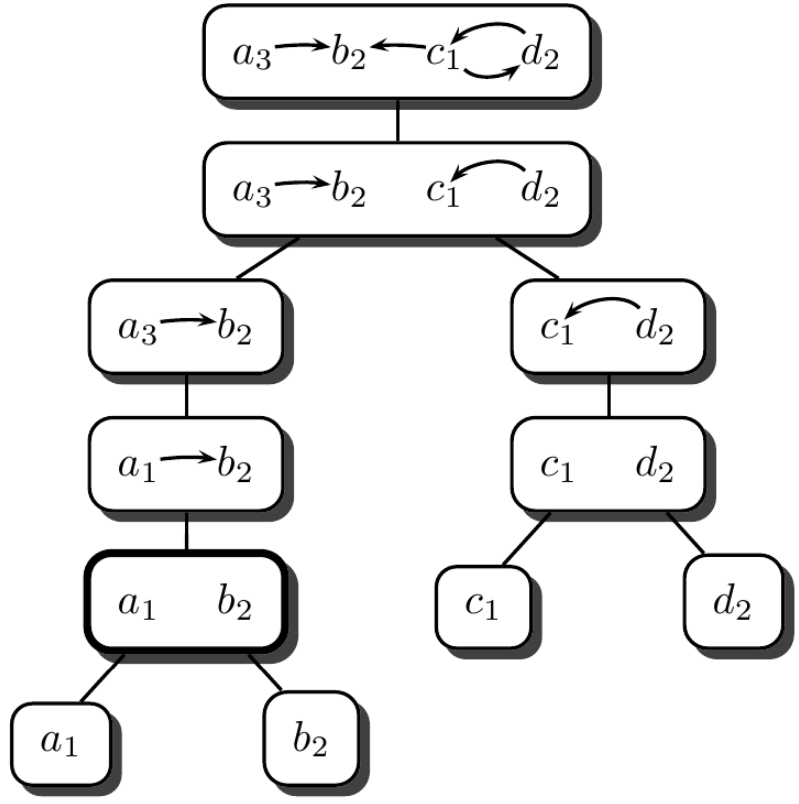
Clique-Width – Example



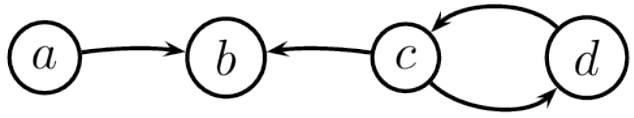
Parse Tree



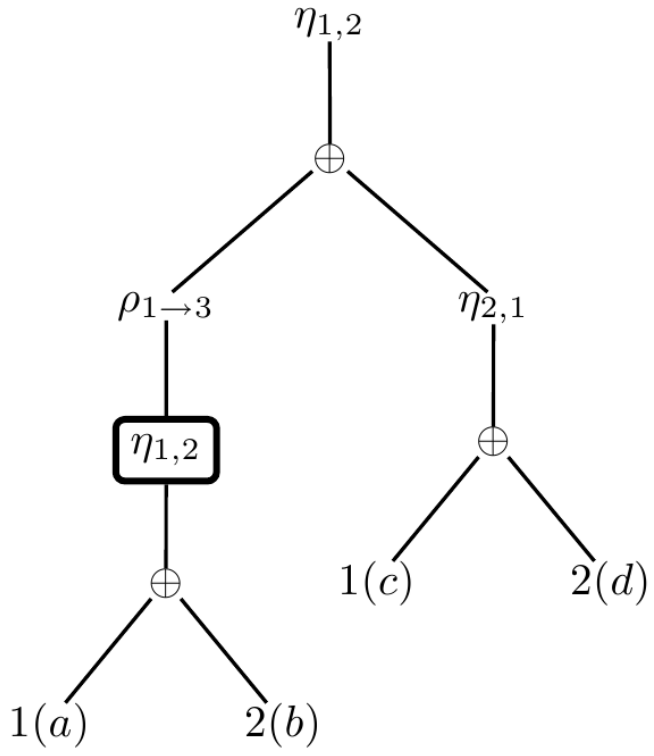
associated AFs



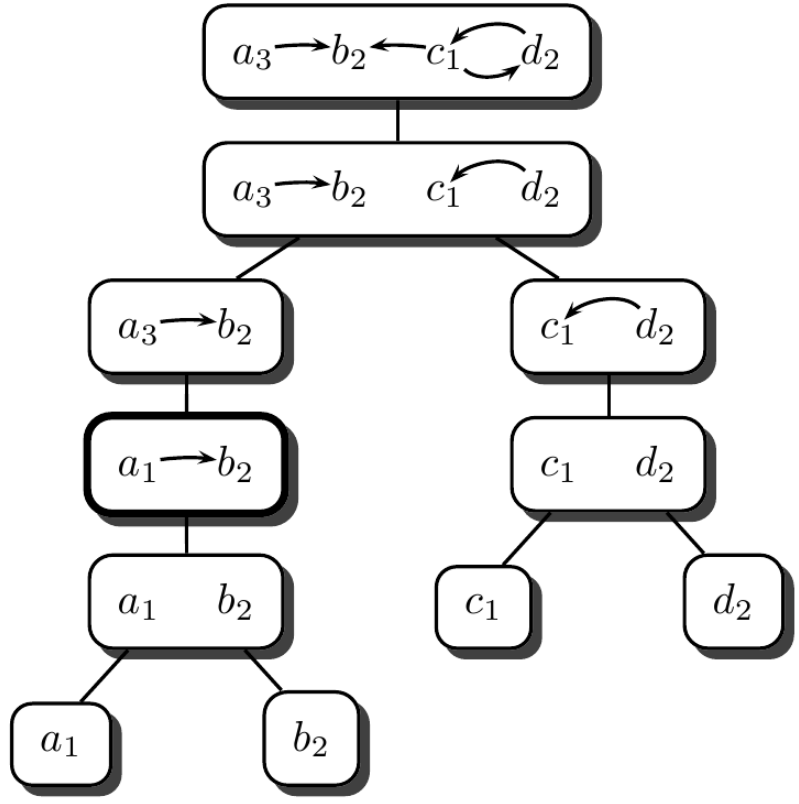
Clique-Width – Example



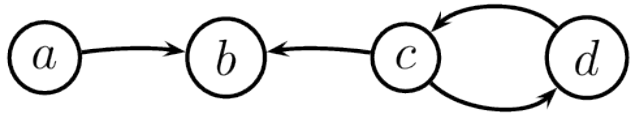
Parse Tree



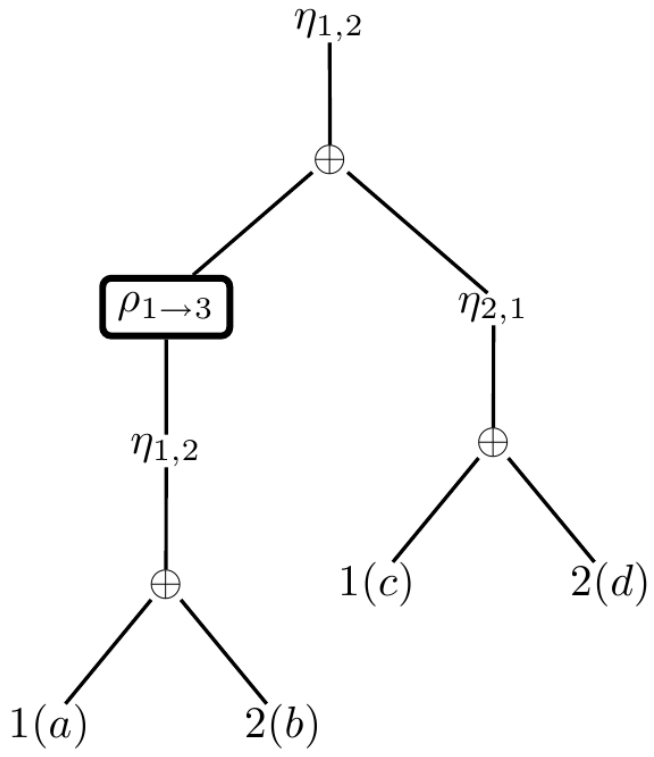
associated AFs



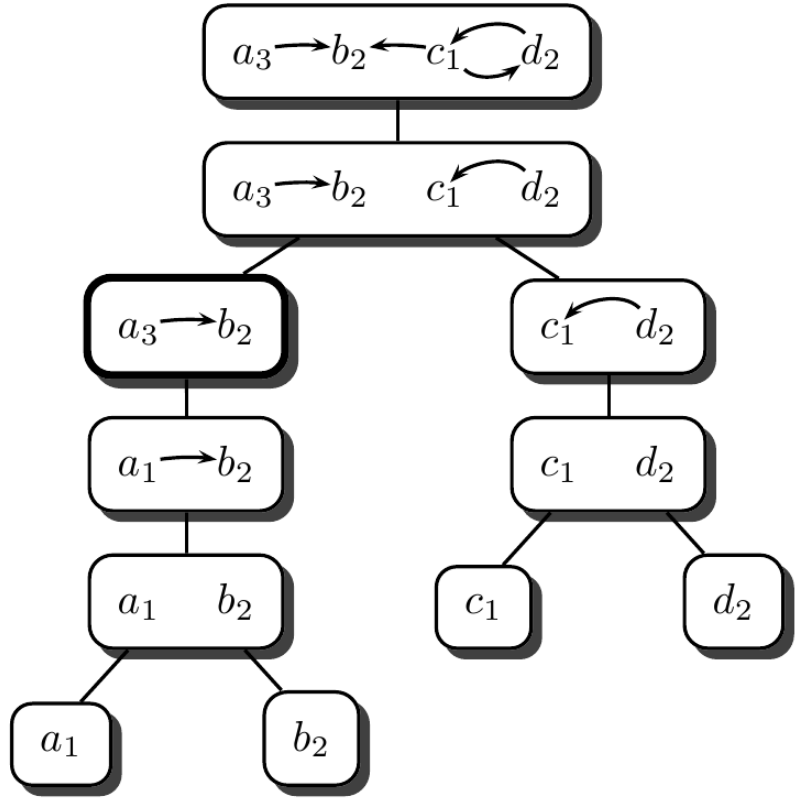
Clique-Width – Example



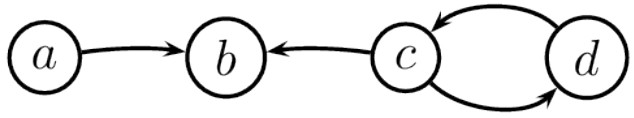
Parse Tree



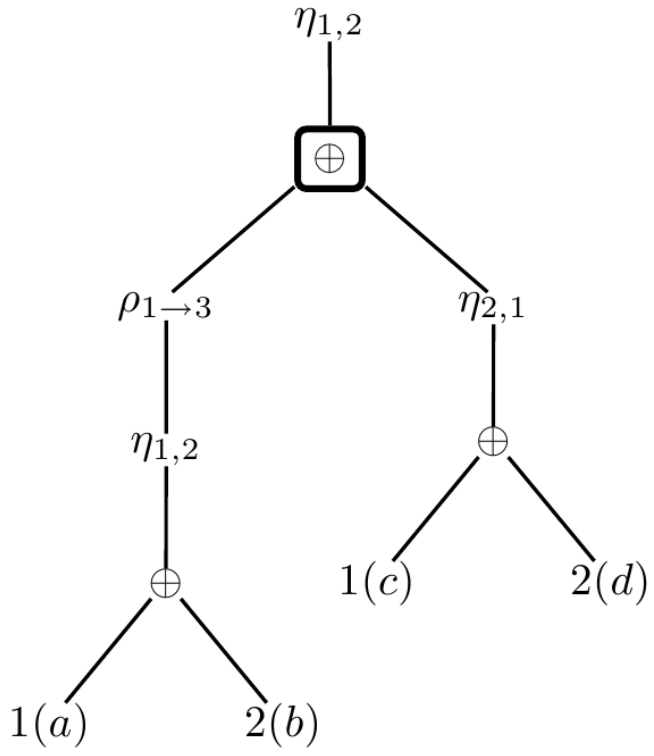
associated AFs



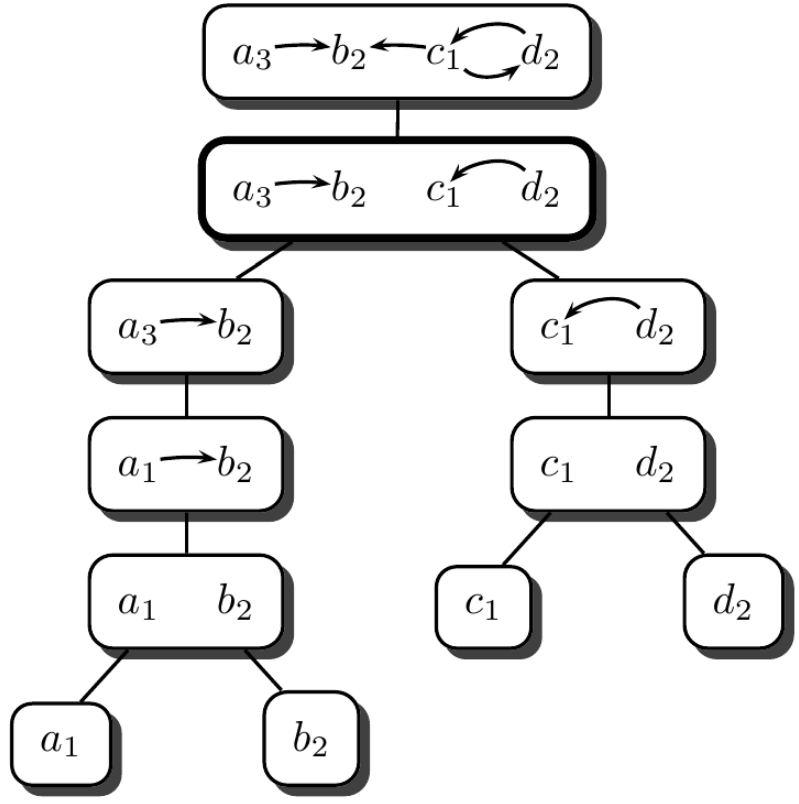
Clique-Width – Example



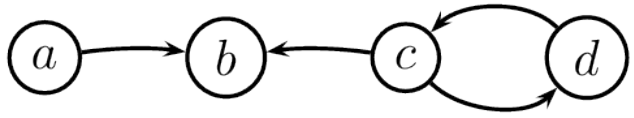
Parse Tree



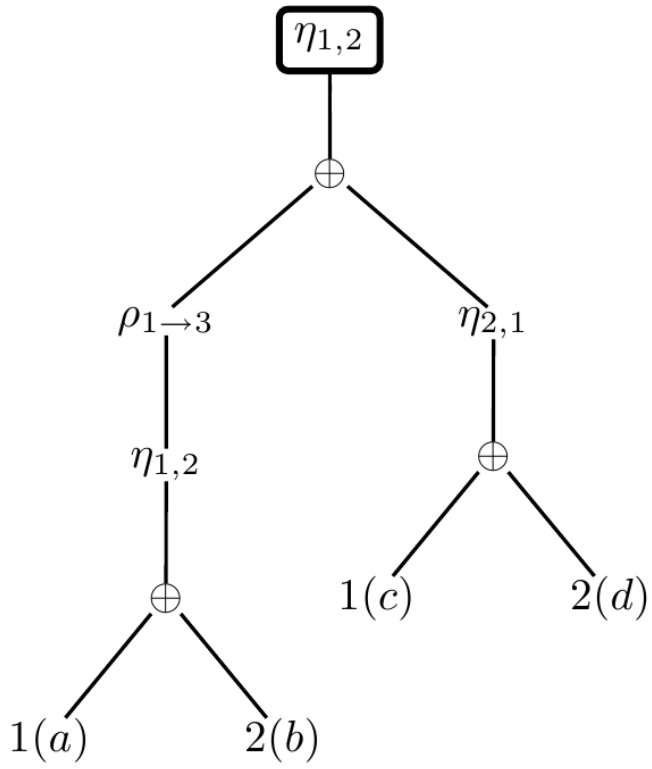
associated AFs



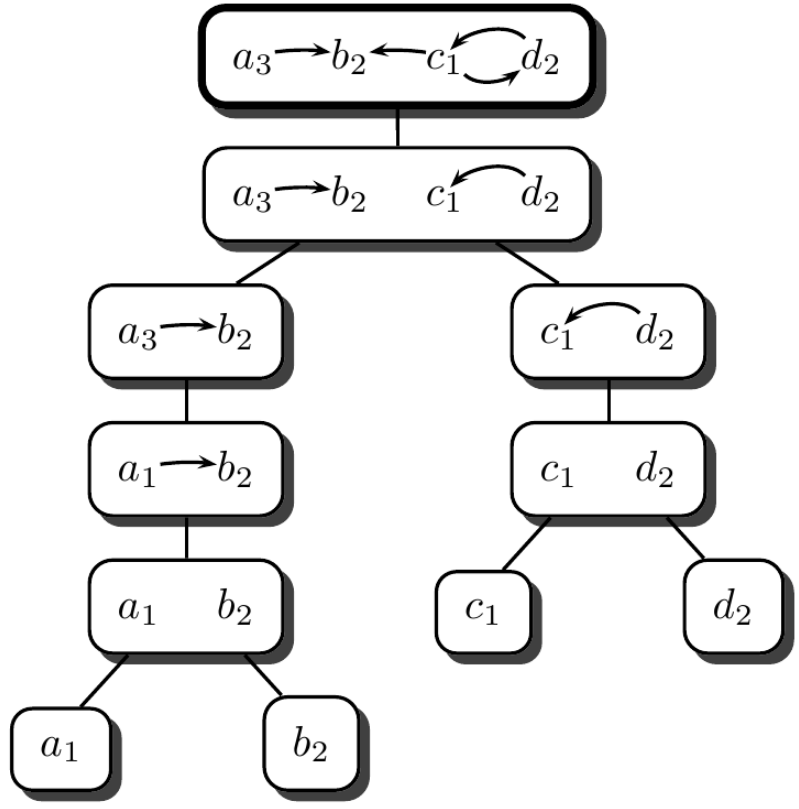
Clique-Width – Example



Parse Tree

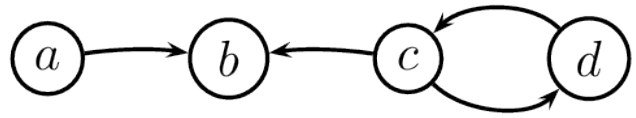


associated AFs



Clique-Width – Example

k-expression for the following AF?



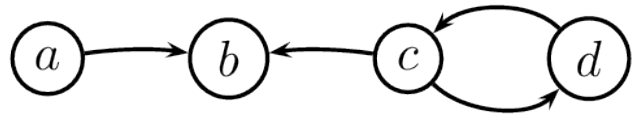
Result:

we found a k-expression describing the AF with $k = 3$:

$$\eta_{1,2}(\rho_{1\to 3}(\eta_{1,2}(1(a) \oplus 2(b))) \oplus \eta_{2,1}(1(c) \oplus 2(d)))$$

Clique-Width – Example

k-expression for the following AF?



Result:

we found a k-expression describing the AF with $k = 3$:

$$\eta_{1,2}(\rho_{1\to 3}(\eta_{1,2}(1(a) \oplus 2(b))) \oplus \eta_{2,1}(1(c) \oplus 2(d)))$$

In fact, above AF has clique-width 3.

Tree-Width vs. Clique-Width

Some Observations

- Class of AFs with bounded clique-width subsumes and significantly extends the class of AFs with bounded tree-width
- There are both, sparse (e.g. tree-like AFs) and dense AFs (e.g. clique-like AFs) that possess small clique-width
- Clique-width allows a direct handle for the orientation of attacks
- However, finding a tree-decomposition of width $\leq k$ (for given k) is much easier than finding a k-expression

Meta-Theorems

[Courcelle 1987; Courcelle, Makowsky & Rotics, 2000]

Any graph problem that can be expressed in MSO_2 (resp. MSO_1) can be solved in linear time for graphs of tree-width (resp. clique-width) bounded by some constant k.

Meta-Theorems

[Courcelle 1987; Courcelle, Makowsky & Rotics, 2000]

Any graph problem that can be expressed in MSO_2 (resp. MSO_1) can be solved in linear time for graphs of tree-width (resp. clique-width) bounded by some constant k.

Example: Existence of a stable extension [Dunne 2007]

$$\begin{aligned} SE(A,R) \quad = \quad \exists U \quad & \forall x,y\big(\langle x,y\rangle \in R \rightarrow (\neg x \in U \vee \neg y \in U)\big) \wedge \\ & \forall x\big((x \in A \wedge \neg x \in U) \rightarrow \exists z(z \in U \wedge \langle z,y\rangle \in R)\big) \end{aligned}$$

Meta-Theorems

[Courcelle 1987; Courcelle, Makowsky & Rotics, 2000]

Any graph problem that can be expressed in MSO_2 (resp. MSO_1) can be solved in linear time for graphs of tree-width (resp. clique-width) bounded by some constant k.

Example: Existence of a stable extension [Dunne 2007]

$$SE(A,R) \quad = \quad \exists U \quad \forall x,y\big(\langle x,y\rangle \in R \rightarrow (\neg x \in U \vee \neg y \in U)\big) \wedge \\ \forall x\big((x \in A \wedge \neg x \in U) \rightarrow \exists z(z \in U \wedge \langle z,y\rangle \in R)\big)$$

Corollary

Cred_σ and Skept_σ when restricted to graphs of bounded clique-width (and thus of bounded tree-width) can be decided in linear time.

Meta-Theorems

[Courcelle 1987; Courcelle, Makowsky & Rotics, 2000]

Any graph problem that can be expressed in MSO_2 (resp. MSO_1) can be solved in linear time for graphs of tree-width (resp. clique-width) bounded by some constant k.

Example: Existence of a stable extension [Dunne 2007]

$$SE(A,R) = \exists U \quad \forall x,y\big(\langle x,y\rangle \in R \rightarrow (\neg x \in U \vee \neg y \in U)\big) \wedge \forall x\big((x \in A \wedge \neg x \in U) \rightarrow \exists z(z \in U \wedge \langle z,y\rangle \in R)\big)$$

Corollary

Cred_σ and Skept_σ when restricted to graphs of bounded clique-width (and thus of bounded tree-width) can be decided in linear time.

But the meta-theorems do not lead to efficient algorithms

Exercises

1. Show that the following problems can be decided in polynomial time Skept$_{adm}$, Skept$_{comp}$, Skept$_{naive}$, Exists$^{\neg\emptyset}_{comp}$.
2. Show that the problems Cred$_{adm}$, Cred$_{comp}$, Cred$_{pref}$ are NP-complete.
3. Give MSO encodings of the following problems: Cred$_{adm}$, Cred$_{pref}$, Skept$_{pref}$.


P. Baroni, P. E. Dunne, and M. Giacomin.
On the resolution-based family of abstract argumentation semantics and its grounded instance.
Artif. Intell., 175(3-4):791–813, 2011.


S. Coste-Marquis, C. Devred, and P. Marquis.
Symmetric argumentation frameworks.
In Proc. ECSQARU 2005, pages 317–328. Springer, 2005.


B. Courcelle.
Recognizability and second-order definability for sets of finite graphs.
Technical Report I-8634, Université de Bordeaux, 1987.


B. Courcelle, J. A. Makowsky, and U. Rotics.
Linear time solvable optimization problems on graphs of bounded clique-width.
Theory Comput. Syst., 33(2):125–150, 2000.


Y. Dimopoulos and A. Torres.
Graph theoretical structures in logic programs and default theories.
Theor. Comput. Sci., 170(1-2):209–244, 1996.


P. M. Dung.
On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games.
Artif. Intell., 77(2):321–358, 1995.


P. E. Dunne.

Computational properties of argument systems satisfying graph-theoretic constraints.
Artif. Intell., 171(10-15):701–729, 2007.


P. E. Dunne.
The computational complexity of ideal semantics I: Abstract argumentation frameworks.
In Proc. COMMA'08, pages 147–158. IOS Press, 2008.


P. E. Dunne and T. J. M. Bench-Capon.
Coherence in finite argument systems.
Artif. Intell., 141(1/2):187–203, 2002.


P. E. Dunne and T. J. M. Bench-Capon.
Complexity in value-based argument systems.
In Proc. JELIA 2004, pages 360–371. Springer, 2004.


P. E. Dunne and M. Caminada.
Computational complexity of semi-stable semantics in abstract argumentation frameworks.
In Proc. JELIA 2008, pages 153–165. Springer, 2008.


W. Dvořák, P. E. Dunne and S. Woltran.
Parametric Properties of Ideal Semantics.
In Proc. IJCAI 2011, 851-856 , 2011.


W. Dvořák and S. Woltran.

Complexity of semi-stable and stage semantics in argumentation frameworks.
Inf. Process. Lett., 110(11):425–430, 2010.


W. Dvořák and S. Woltran.
On the intertranslatability of argumentation semantics.
Journal of Artificial Intelligence Research, 41 (2011): 445–475.


W. Dvořák, R. Pichler und S. Woltran.
Towards fixed-parameter tractable algorithms for abstract argumentation.
Artif. Intell. 186(1): 1–37, 2012.


W. Dvořák, S. Szeider and S. Woltran.
Abstract argumentation via monadic second order logic.
In Proc. SUM 2012, pages 85–98. Springer, 2012.


W. Dvořák
On the Complexity of Computing the Justification Status of an Argument.
First International Workshop on the Theory and Applications of Formal Argumentation