CILC 2024

A Semantics-Aware Evaluation Order for Abstract Argumentation Frameworks

Stefano Bistarelli and Carlo Taticchi

A.D. 1308
unipg
DIPARTIMENTO
DI MATEMATICA E INFORMATICA

Some motivations

- AFs simplify complex reasoning processes and predispose them to automation
- Too simple to capture certain aspects of human reasoning, like dynamics

Some motivations

- AFs simplify complex reasoning processes and predispose them to automation
- Too simple to capture certain aspects of human reasoning, like dynamics
- Example:
  a. “Everyone should eat more vegetables to reduce the risk of chronic diseases”
  b. “Excess consumption of certain vegetables can lead to nutrient overdoses”
  c. “Vegetable-induced nutrient overdoses are rare and manageable with a varied diet”

Some motivations

- AFs simplify complex reasoning processes and predispose them to automation
- Too simple to capture certain aspects of human reasoning, like dynamics
- Example:
  a. “Everyone should eat more vegetables to reduce the risk of chronic diseases”
  b. “Excess consumption of certain vegetables can lead to nutrient overdoses”
  c. “Vegetable-induced nutrient overdoses are rare and manageable with a varied diet”
- How do we trace the order of the sentences that makes the most sense?

Some motivations

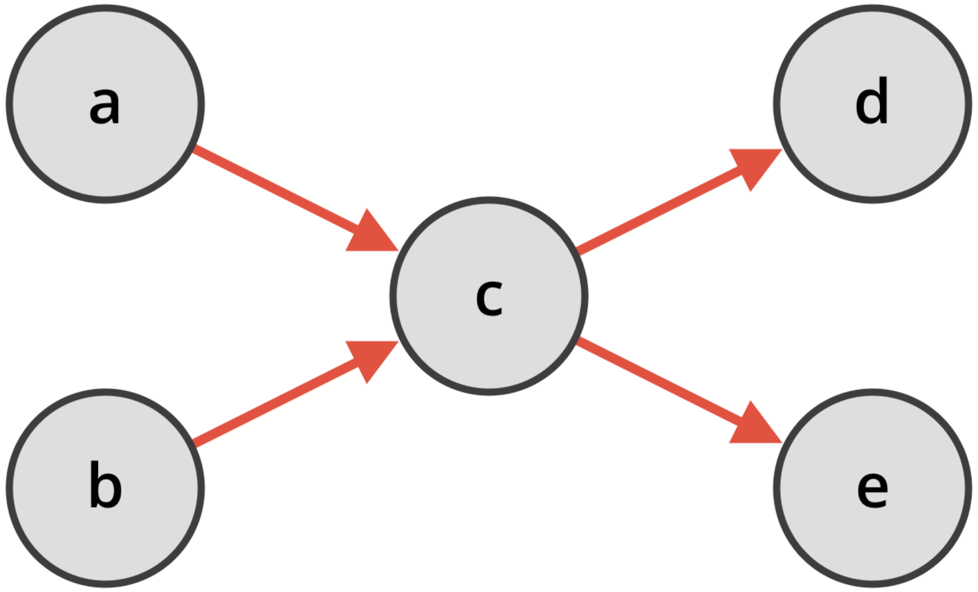
- AFs simplify complex reasoning processes and predispose them to automation
- Too simple to capture certain aspects of human reasoning, like dynamics
- Example:
    a. "Everyone should eat more vegetables to reduce the risk of chronic diseases"
    b. "Excess consumption of certain vegetables can lead to nutrient overdoses"
    c. "Vegetable-induced nutrient overdoses are rare and manageable with a varied diet"
- How do we trace the order of the sentences that makes the most sense?
- Proposed solution: capture arguments' dependency

Overview

- Dependency Graphs & Feasible Evaluation Order for AFs
- Semantic dependency
- Semantics-Aware Evaluation Order + some examples
- ASP Implementation for Minimal Invariant Attack Sets
- Conclusion and Future Work

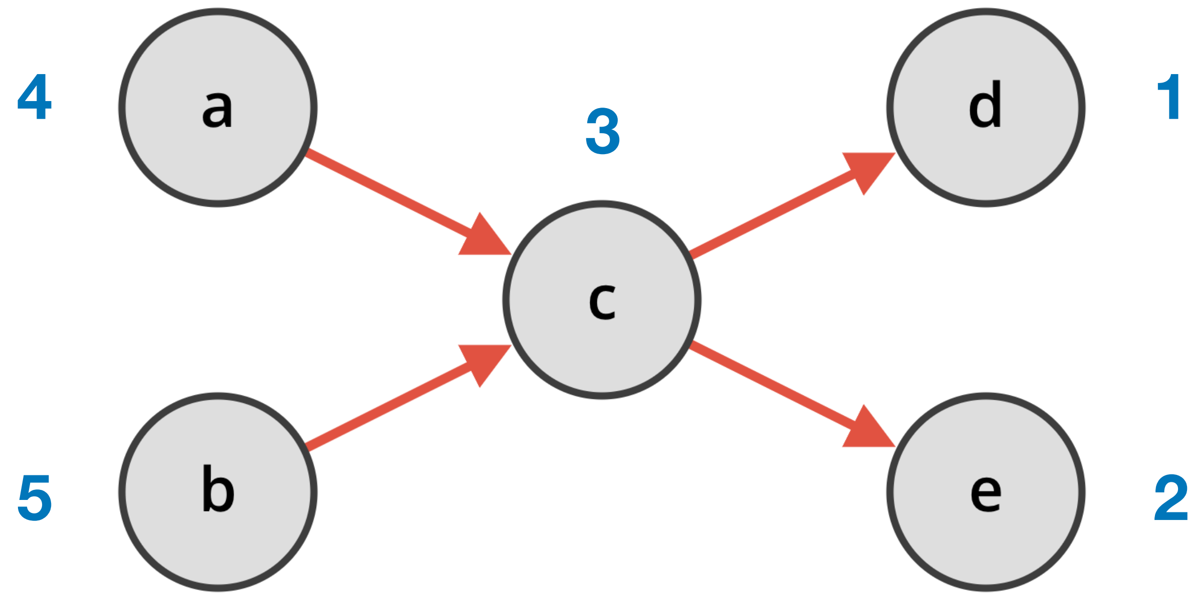
Dependency Graphs

- Represent the dependencies of various elements
- We look for a **Correct Evaluation Order**:
  - if x is evaluated before y, then x must not depend on y



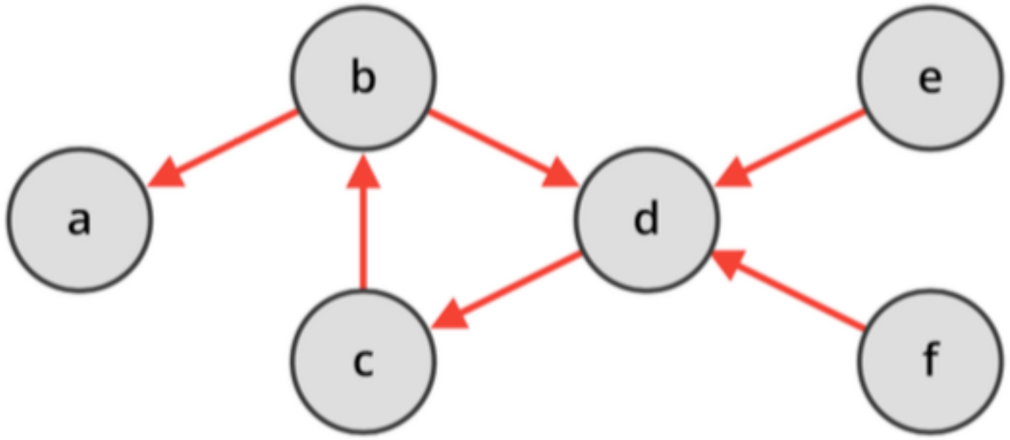
Dependency Graphs

- Represent the dependencies of various elements
- We look for a **Correct Evaluation Order**:
    - if x is evaluated before y, then x must not depend on y



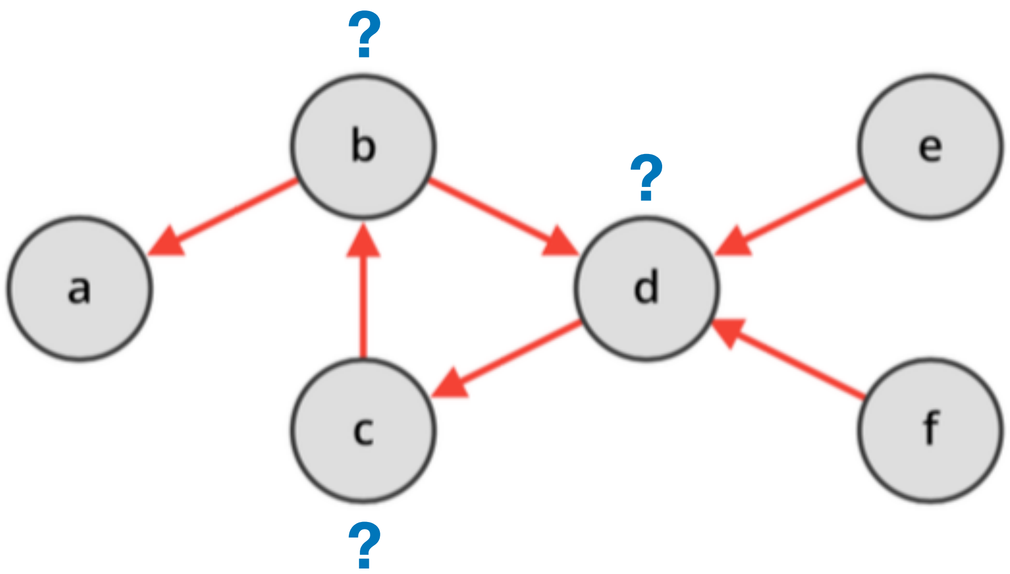
AFs as Dependency Graphs

- AFs can be interpreted as Dependency Graphs
- Finding a Correct Evaluation Order = obtain the reasoning process that generated the graph



AFs as Dependency Graphs

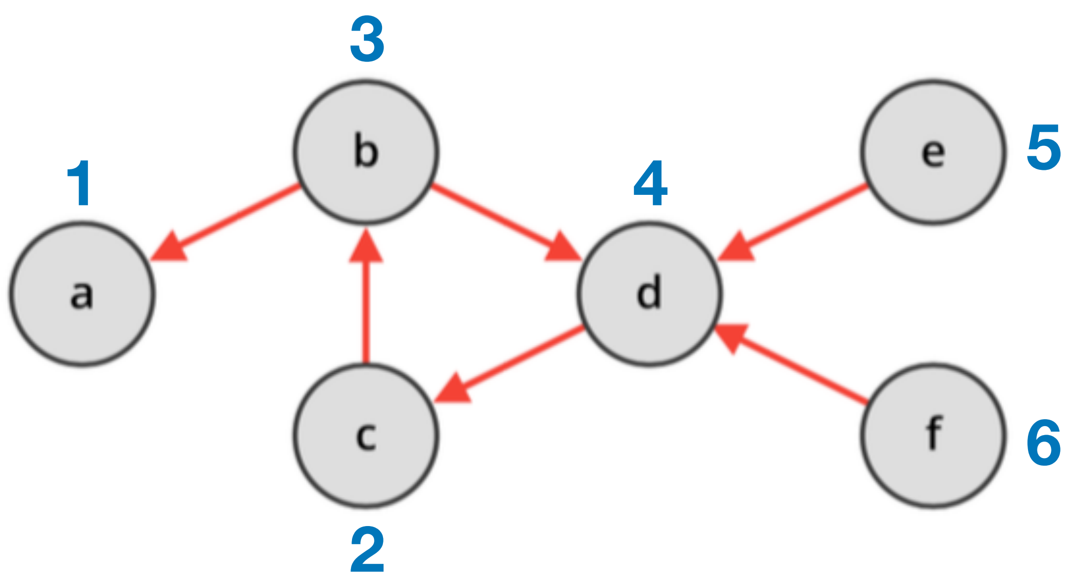
- AFs can be interpreted as Dependency Graphs
- Finding a Correct Evaluation Order = obtain the reasoning process that generated the graph
- Issue: a Correct Evaluation Order cannot be found when the graph has **circular dependencies**



Feasible Evaluation Order[1]

- Treat any cycle as an agglomeration of nodes whose evaluation order is not influential

- **Feasible Evaluation Order**: for all arguments x and y not in circular dependencies between them, if y is evaluated before y, then x and all arguments in circular dependences with x must not depend on y and all arguments in circular dependences with y

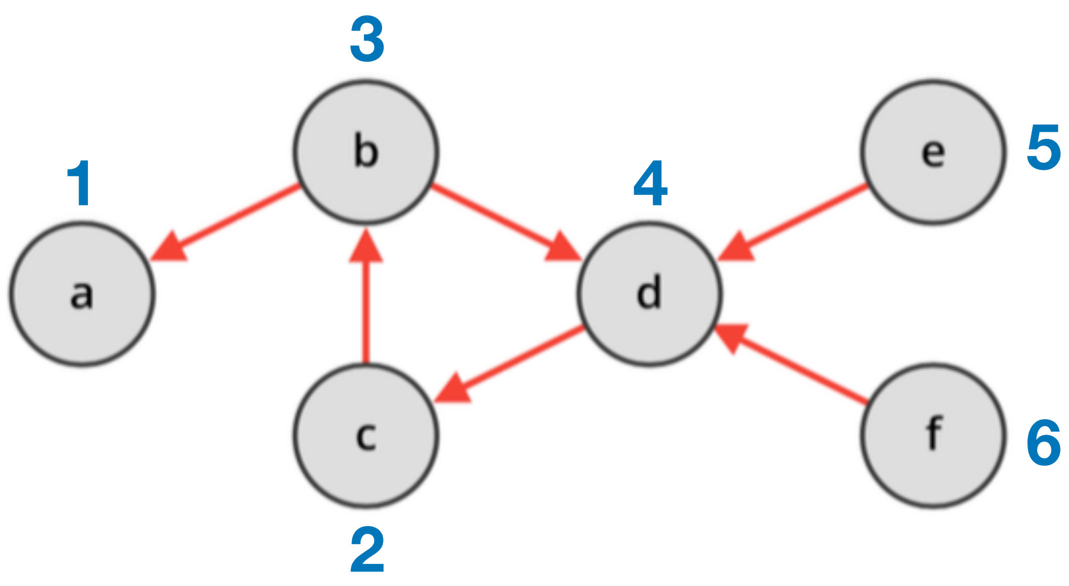


[1] Stefano Bistarelli, Carlo Taticchi: Deriving Dependency Graphs from Abstract Argumentation Frameworks. AI*IA 2023: 17-29

Feasible Evaluation Order[1]

- Treat any cycle as an agglomeration of nodes whose evaluation order is not influential

- **Feasible Evaluation Order**: for all arguments x and y not in circular dependencies between them, if y is evaluated before y, then x and all arguments in circular dependences with x must not depend on y and all arguments in circular dependences with y

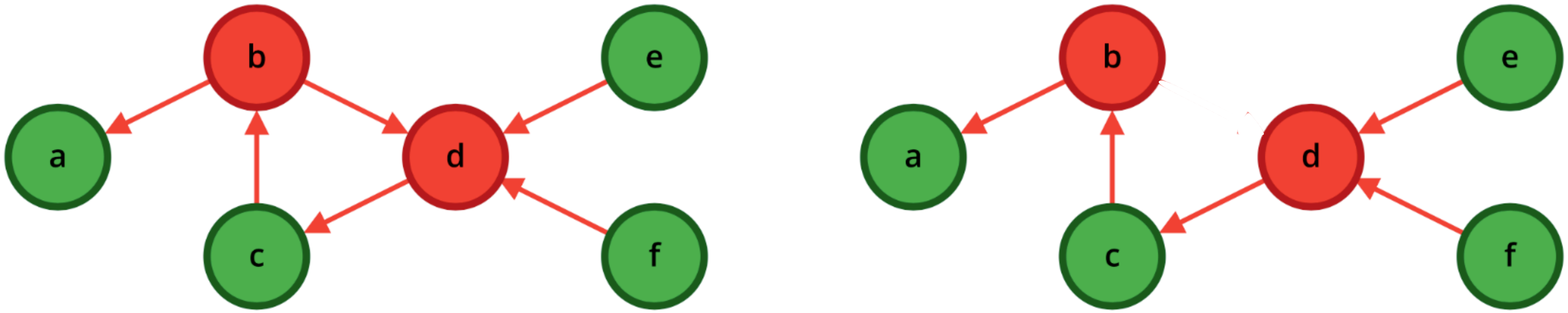


ISSUE: b, c and d can be evaluated in any order!

[1] Stefano Bistarelli, Carlo Taticchi: Deriving Dependency Graphs from Abstract Argumentation Frameworks. AI*IA 2023: 17-29

Semantic dependency

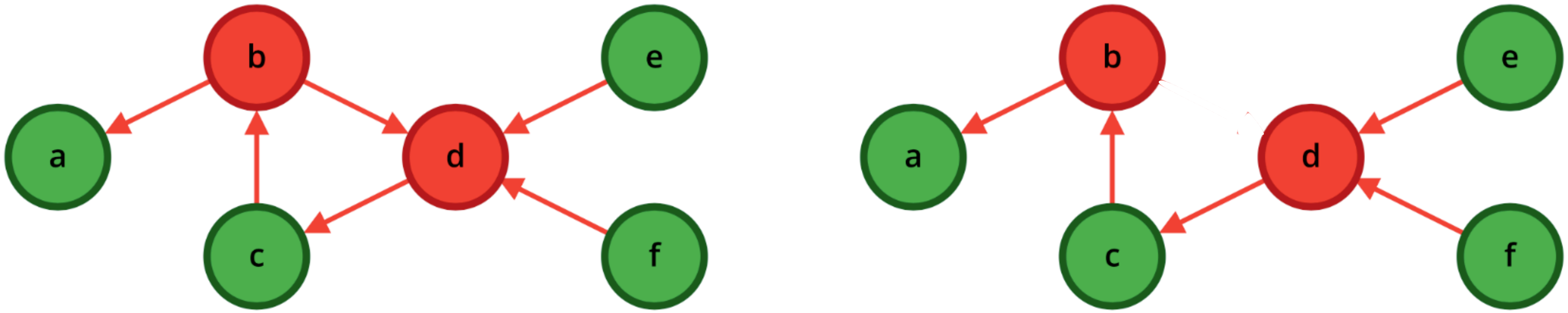
- Idea: since the inclusion of b does not alter the acceptability state of the other arguments within the cycle, it can be evaluated without being constrained by the dependency arising from the attacks it conducts



An argument is OUT if it is attacked by at least one IN

Semantic dependency

- Idea: since the inclusion of b does not alter the acceptability state of the other arguments within the cycle, it can be evaluated without being constrained by the dependency arising from the attacks it conducts

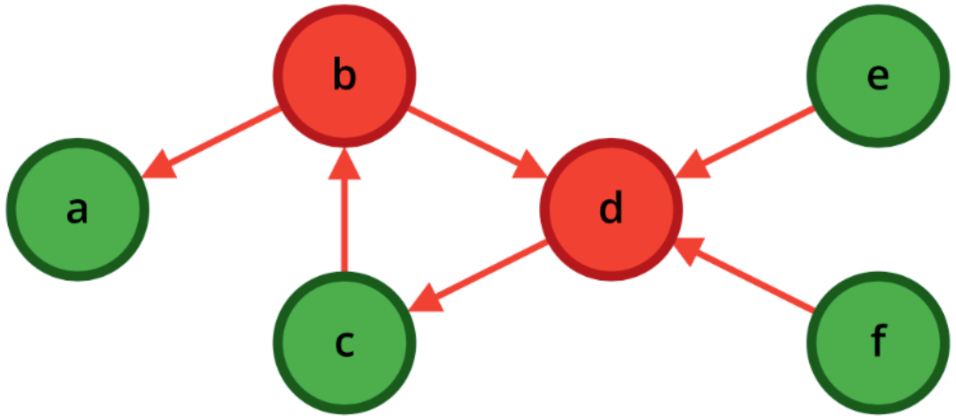


- Let $F = \langle Arg, R \rangle$, $a, b \in Arg$, and $G = \langle Arg, R \backslash \{(b, a)\} \rangle$
  a **semantically depends** on b if and only if $L^F(a) \neq F^G(a)$

- (b, a) is an **invariant attack** if a is semantically independent of b

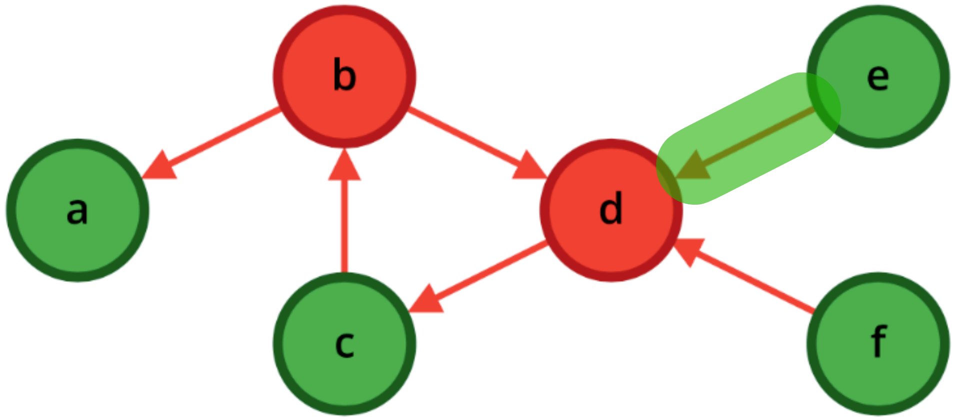
- Removing multiple invariant attacks is not guaranteed, in general, to leave the labelling unaltered.

- Example: removing the both invariant attacks (e, d) and (f, d) changes the label of d from OUT to UND



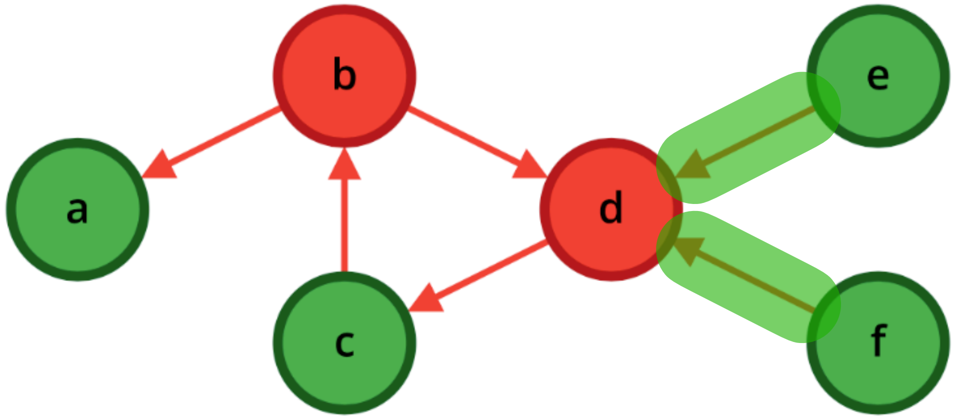
- Removing multiple invariant attacks is not guaranteed, in general, to leave the labelling unaltered.

- Example: removing the both invariant attacks (e, d) and (f, d) changes the label of d from OUT to UND



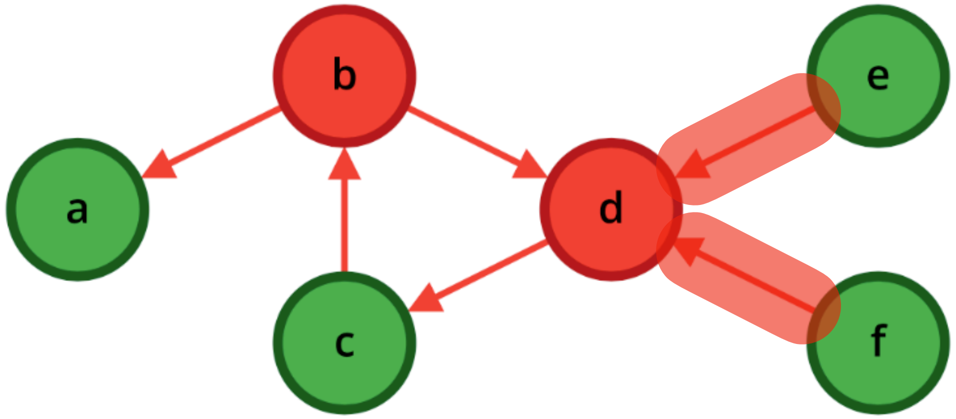
- Removing multiple invariant attacks is not guaranteed, in general, to leave the labelling unaltered.

- Example: removing the both invariant attacks (e, d) and (f, d) changes the label of d from OUT to UND



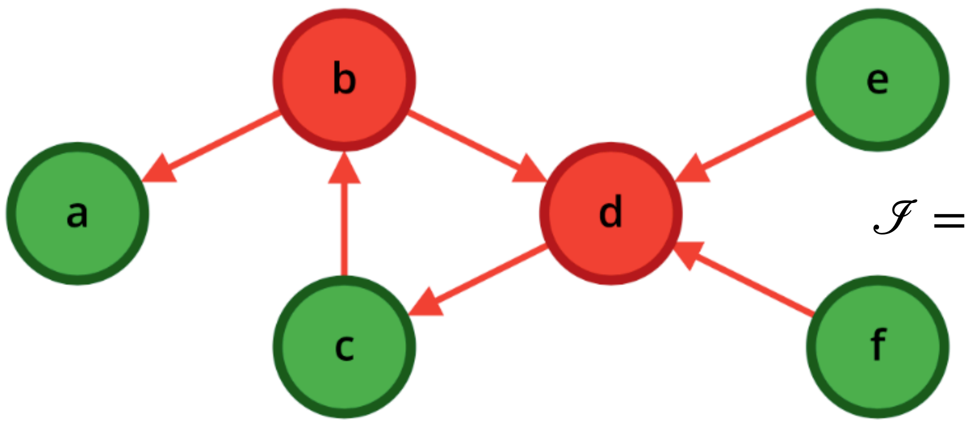
- Removing multiple invariant attacks is not guaranteed, in general, to leave the labelling unaltered.

- Example: removing the both invariant attacks (e, d) and (f, d) changes the label of d from OUT to UND



- Removing multiple invariant attacks is not guaranteed, in general, to leave the labelling unaltered.

- Example: removing the both invariant attacks (e, d) and (f, d) changes the label of d from OUT to UND

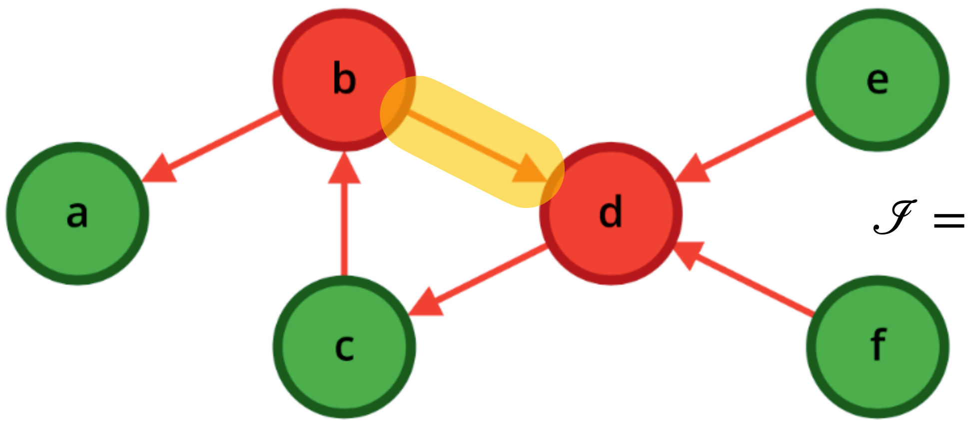


$\mathscr{I} = \{\emptyset, \{(b,d)\}, \{(d,c)\}, \{(b,d),(d,c)\}\}$

- Consider $F = \langle Arg, R \rangle$ and let $\overline{R} = \{(a,b) \mid (a,b) \in R \wedge b \in CiD(a)\}$ be the set of attacks between arguments within a cycle. Then $I \subseteq \overline{R}$ is an **invariant attack set** within cycles of F if, given $G = \langle Arg, R \backslash I \rangle$ we have that $L^F(a) = F^G(a)$.

- Removing multiple invariant attacks is not guaranteed, in general, to leave the labelling unaltered.

- Example: removing the both invariant attacks (e, d) and (f, d) changes the label of d from OUT to UND

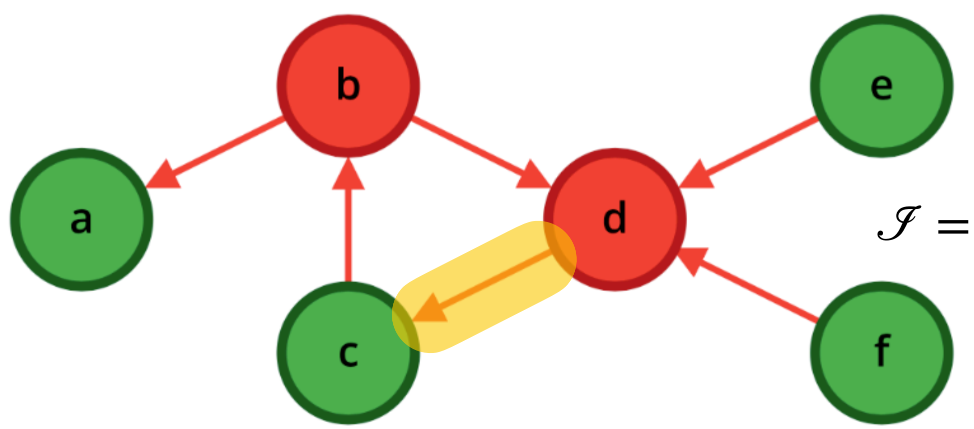


$\mathcal{I} = \{\emptyset, \{(b, d)\}, \{(d, c)\}, \{(b, d), (d, c)\}\}$

- Consider $F = \langle Arg, R \rangle$ and let $\overline{R} = \{(a, b) \mid (a, b) \in R \wedge b \in CiD(a)\}$ be the set of attacks between arguments within a cycle. Then $I \subseteq \overline{R}$ is an **invariant attack set** within cycles of F if, given $G = \langle Arg, R \backslash I \rangle$ we have that $L^F(a) = F^G(a)$.

- Removing multiple invariant attacks is not guaranteed, in general, to leave the labelling unaltered.

- Example: removing the both invariant attacks (e, d) and (f, d) changes the label of d from OUT to UND

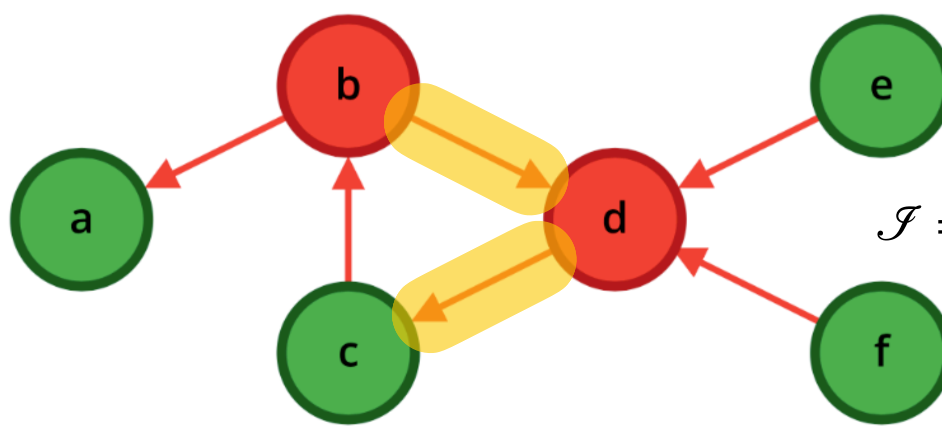


$\mathcal{I} = \{\emptyset, \{(b, d)\}, \{(d, c)\}, \{(b, d), (d, c)\}\}$

- Consider $F = \langle Arg, R \rangle$ and let $\overline{R} = \{(a, b) \mid (a, b) \in R \wedge b \in CiD(a)\}$ be the set of attacks between arguments within a cycle. Then $I \subseteq \overline{R}$ is an **invariant attack set** within cycles of F if, given $G = \langle Arg, R \backslash I \rangle$ we have that $L^F(a) = F^G(a)$.

- Removing multiple invariant attacks is not guaranteed, in general, to leave the labelling unaltered.

- Example: removing the both invariant attacks (e, d) and (f, d) changes the label of d from OUT to UND

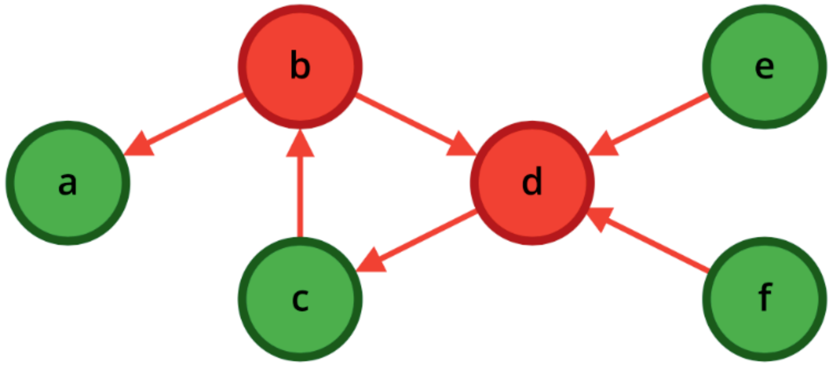


$\mathcal{I} = \{\varnothing, \{(b, d)\}, \{(d, c)\}, \{(b, d), (d, c)\}\}$

- Consider $F = \langle Arg, R \rangle$ and let $\overline{R} = \{(a, b) \mid (a, b) \in R \wedge b \in CiD(a)\}$ be the set of attacks between arguments within a cycle. Then $I \subseteq \overline{R}$ is an **invariant attack set** within cycles of F if, given $G = \langle Arg, R \backslash I \rangle$ we have that $L^F(a) = F^G(a)$.

Semantics-Aware Evaluation Order

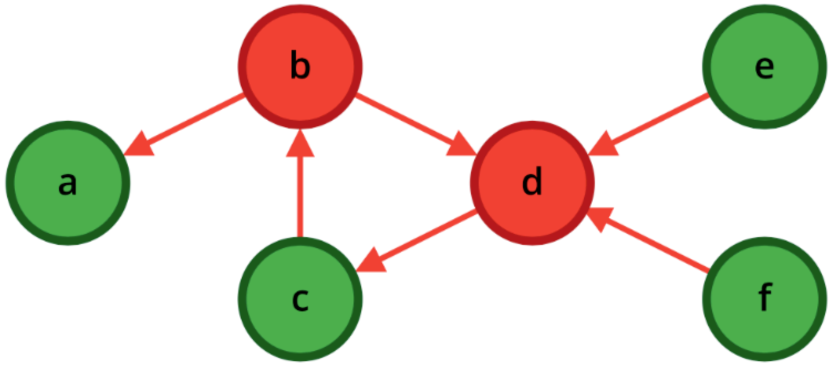
- Let $F = \langle Arg, R \rangle$ be an AF, and $\mathscr{I}$ the set of all invariant attack sets within cycles of F. A **semantics-aware evaluation order** for F is a numbering $n : Arg \rightarrow \mathbb{N}$ such that n is a feasible evaluation order for $G = \langle \text{Arg}, R \backslash I \rangle$, where $I \in \mathscr{I}$ and



Semantics-Aware Evaluation Order

- Let $F = \langle Arg, R \rangle$ be an AF, and $\mathscr{I}$ the set of all invariant attack sets within cycles of F. A **semantics-aware evaluation order** for F is a numbering $n : Arg \rightarrow \mathbb{N}$ such that n is a feasible evaluation order for $G = \langle \text{Arg}, R \setminus I \rangle$, where $I \in \mathscr{I}$ and
  - $\nexists I' \in \mathscr{I}$ such that $|\text{ArC}(\langle \text{Arg}, R \setminus I' \rangle)| < |\text{ArC}(G)|$ with $\text{ArC}(F) = \bigcup_{C \in Cycles(F)} C$

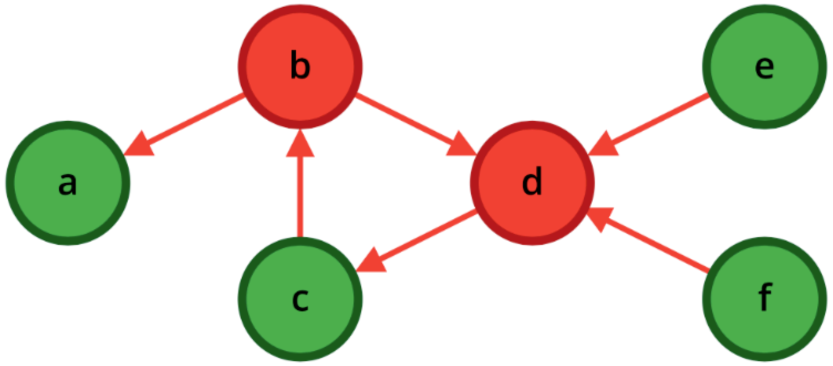
obtaining as few arguments as possible in cycles



Semantics-Aware Evaluation Order

- Let $F = \langle Arg, R \rangle$ be an AF, and $\mathscr{I}$ the set of all invariant attack sets within cycles of F. A **semantics-aware evaluation order** for F is a numbering $n : Arg \rightarrow \mathbb{N}$ such that n is a feasible evaluation order for $G = \langle \text{Arg}, R \backslash I \rangle$, where $I \in \mathscr{I}$ and
  - $\nexists I' \in \mathscr{I}$ such that $|\text{ArC}(\langle \text{Arg}, R \backslash I' \rangle)| < |\text{ArC}(G)|$ with $\text{ArC}(F) = \bigcup_{C \in Cycles(F)} C$
  - $\nexists I'' \in \mathscr{I}$ such that $I'' \subset I$ and $|\text{ArC}(\langle \text{Arg}, R \backslash I'' \rangle)| = |\text{ArC}(G)|$

obtaining as few arguments as possible in cycles

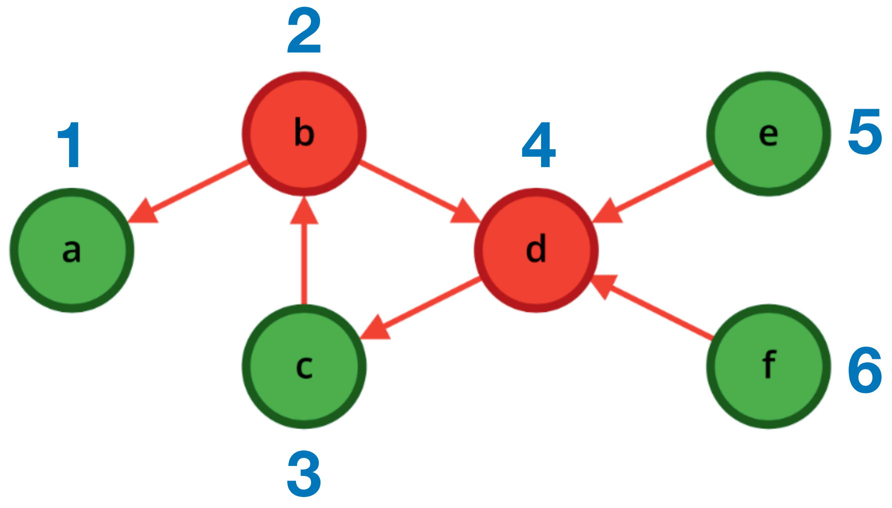


while removing as few invariant attacks as possible

Semantics-Aware Evaluation Order

- Let $F = \langle Arg, R \rangle$ be an AF, and $\mathcal{I}$ the set of all invariant attack sets within cycles of F. A **semantics-aware evaluation order** for F is a numbering $n : Arg \to \mathbb{N}$ such that n is a feasible evaluation order for $G = \langle \text{Arg}, R \setminus I \rangle$, where $I \in \mathcal{I}$ and
  - $\nexists I' \in \mathcal{I}$ such that $|\text{ArC}(\langle \text{Arg}, R \setminus I' \rangle)| < |\text{ArC}(G)|$ with $\text{ArC}(F) = \bigcup_{C \in Cycles(F)} C$
  - $\nexists I'' \in \mathcal{I}$ such that $I'' \subset I$ and $|\text{ArC}(\langle \text{Arg}, R \setminus I'' \rangle)| = |\text{ArC}(G)|$

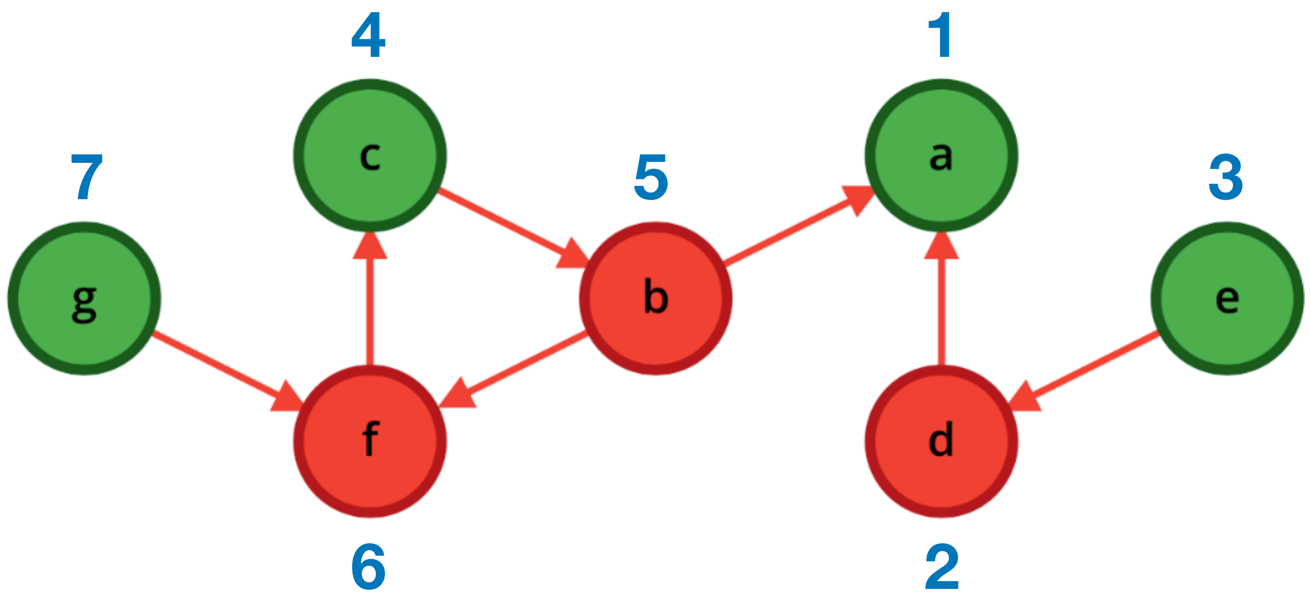
obtaining as few arguments as possible in cycles



while removing as few invariant attacks as possible

Example

a. Anna should rent the apartment she found
b. The apartment seems to have humidity problems
c. The owner is committed to solving structural problems in the apartment
d. A nightclub is set to open nearby shortly
e. Laws forbid the opening of nightclubs in the area
f. The owner, planning to sell the property soon, is unlikely to fund long-term repairs
g. Due to legal constraints, the apartment cannot be sold immediately



Example

a. Anna should rent the apartment she found

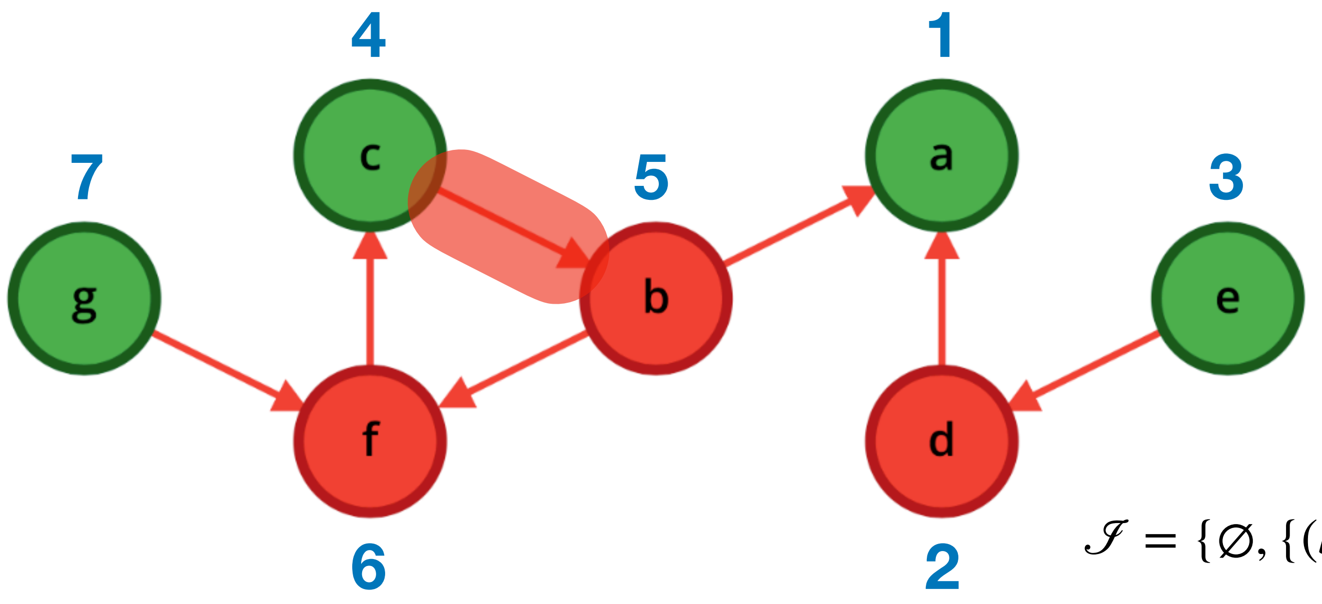
b. The apartment seems to have humidity problems

c. The owner is committed to solving structural problems in the apartment

(a,d,e,c,b,f,g) is not a semantics-aware evaluation order

f. The owner, planning to sell the property soon, is unlikely to fund long-term repairs

g. Due to legal constraints, the apartment cannot be sold immediately



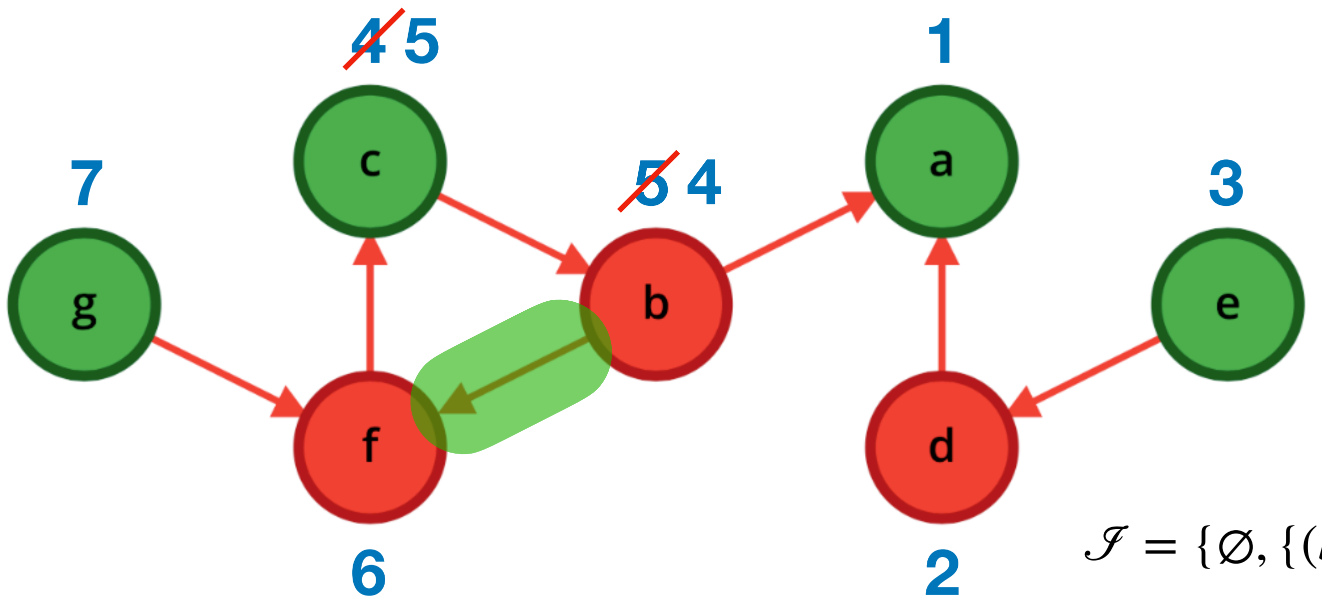
$$\mathcal{I} = \{\emptyset, \{(b,f)\}, \{(f,c)\}, \{(b,f),(f,c)\}\}$$

Example

a. Anna should rent the apartment she found
b. The apartment seems to have humidity problems
c. The owner is committed to solving structural problems in the apartment

(a,d,e,**c,b**,f,g) is not a semantics-aware evaluation order

(a,d,e,**b,c**,f,g) is a semantics-aware evaluation order



$\mathcal{I} = \{\emptyset, \{(b,f)\}, \{(f,c)\}, \{(b,f),(f,c)\}\}$

Integration with tcla

- Automatising the instantiation of an AF

Algorithm 1: Procedure for generating a CLA program to instantiate a specified AF.

Data: AF $F = \langle Arg, R \rangle$

Result: string S

procedure `gen_cla_prog_saeo(`F`):`

  I = `find_minimal_invariant_attack_set`(F) `// I: set of attacks`

  $G = \langle Arg, R \setminus I \rangle$

  S = “”

  foreach (a, b) *in I* **do**

    $S = S$ + “*checkw*$(\{a, b\}, \{\})$ −> *add*$(\{\}, \{(a, b)\})$ −> *success* || ”

  $S = S$ + `gen_cla_prog`(G)

- Output example:

```
checkw({d,a},{}) -> add({},{(d,a)}) -> success ||
checkw({g,b},{}) -> add({},{(g,b)}) -> success ||
checkw({c,f},{}) -> add({a},{(a,f),(a,c)}) -> success
```

Conclusion

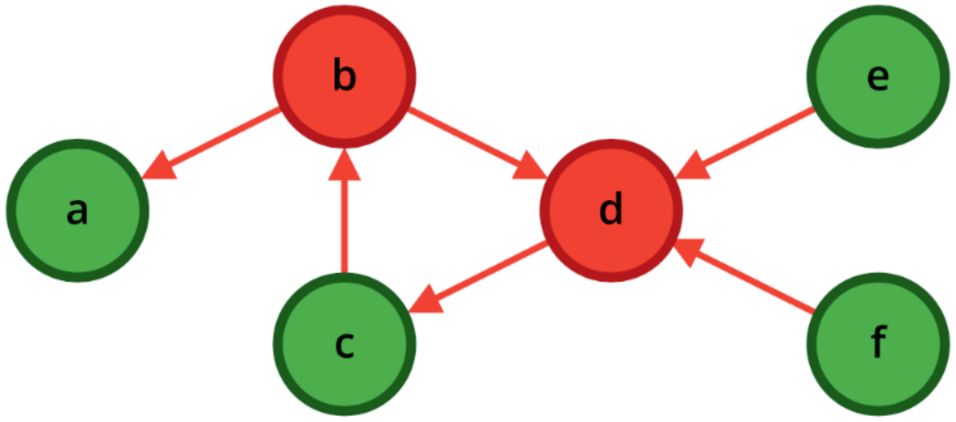
- The semantics-aware evaluation order introduces arguments in a meaningful sequence, simulating what might have happened during the AF's instantiation
- Arguments that receive attacks are evaluated first
- Inside cycles, an argument conducting invariant attacks is selected for evaluation

Conclusion

- The semantics-aware evaluation order introduces arguments in a meaningful sequence, simulating what might have happened during the AF's instantiation
- Arguments that receive attacks are evaluated first
- Inside cycles, an argument conducting invariant attacks is selected for evaluation
- Issues:
  - a cycle may contain no invariant attack
  - multiple arguments conducting invariant attacks may be candidates within a cycle for initial evaluation

Future Perspectives

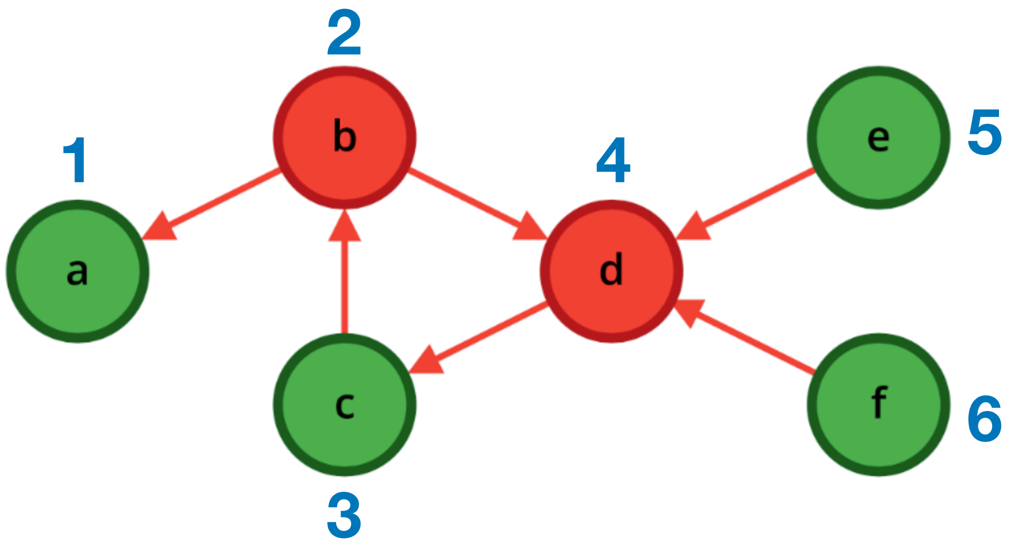
- Refine the evaluation process by using three assumptions:



Future Perspectives

- Refine the evaluation process by using three assumptions:

  1. each new argument inserted must keep the AF connected

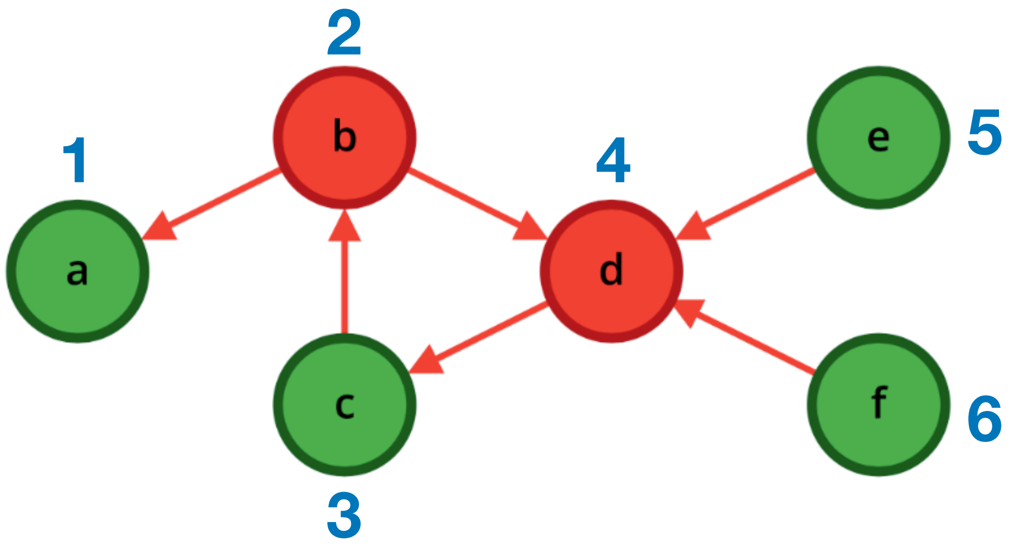


Future Perspectives

- Refine the evaluation process by using three assumptions:

  1. each new argument inserted must keep the AF connected

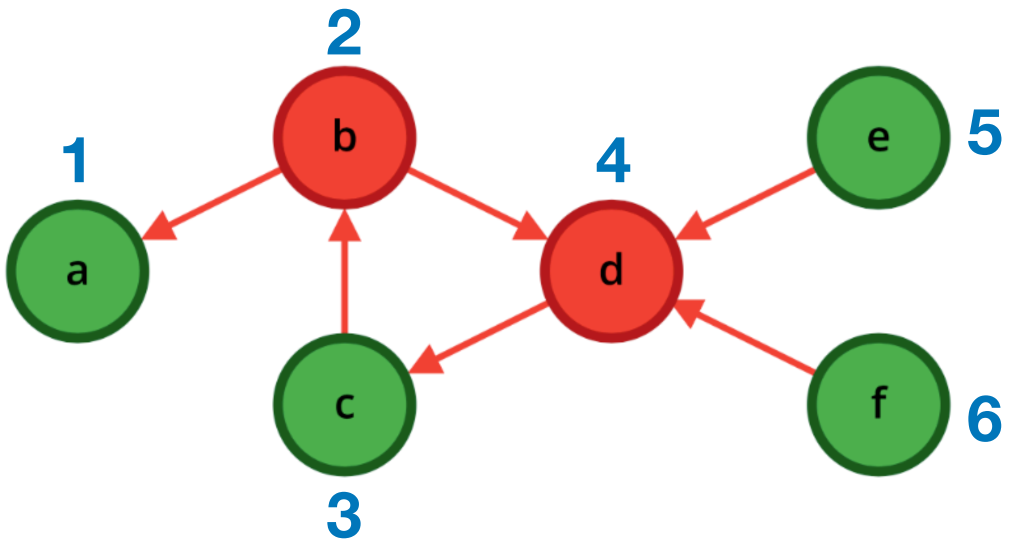
  2. freshly added arguments are considered IN



Future Perspectives

- Refine the evaluation process by using three assumptions:

  1. each new argument inserted must keep the AF connected
  2. freshly added arguments are considered IN
  3. each new argument must change the acceptability of some other argument in the AF



A Semantics-Aware Evaluation Order for Abstract Argumentation Frameworks

Stefano Bistarelli and Carlo Taticchi

Thank you for your attention!

Minimal Invariant Attack Sets

- We provide an ASP implementation to find minimal invariant attack sets
- First, we define path, arguments in cycles, and the removal choice rule

```
% AF provided in a separate file
#include "af.lp".

% Choice rule for attacks to remove
{ remove(X, Y) : att(X, Y) }.

%Paths and cycles
path(X, Y) :- att(X, Y).
path(X, Y) :- path(X, Z), att(Z, Y).
path_after(X, Y) :- att(X, Y), not remove(X, Y).
path_after(X, Y) :- path_after(X, Z), att(Z, Y), not remove(Z, Y).
in_cycle(X) :- path(X, X).
in_cycle_after(X) :- path_after(X,X).
```

- Then, we compute the grounded labelling
- (other semantics can be also defined)

```
%Grounded labelling
in(X) :- arg(X), not has_non_out_attacker(X).
has_non_out_attacker(X) :- att(Y, X), arg(Y), not out(Y).
out(X) :- arg(X), att(Y, X), arg(Y), in(Y).
und(X) :- arg(X), not in(X), not out(X).
:- arg(X), not 1 { in(X); out(X); und(X) } 1.
#minimize { 1@4, X : in(X) }.

%Grounded labelling after the removal
in_after(X) :- arg(X), not has_non_o_a_a(X).
has_non_o_a_a(X) :- att(Y, X), arg(Y), not out_after(Y), not remove(Y, X).
out_after(X) :- arg(X), att(Y, X), arg(Y), in_after(Y), not remove(Y, X).
und_after(X) :- arg(X), not in_after(X), not out_after(X).
:- arg(X), not 1 { in_after(X); out_after(X); und_after(X) } 1.
#minimize { 1@3, X : in_after(X) }.
```

- Finally, we establish semantic equivalence and minimise with respect to
    - number of arguments in cycles
    - removed attacks

```
%Semantic equivalence
:- in(X), not in_after(X).
:- out(X), not out_after(X).
:- und(X), not und_after(X).

%Minimisation
#minimize { 1@2, X : in_cycle_after(X) }.
#minimize { 1@1, X, Y : remove(X, Y) }.

%Output
#show remove/2.
```