Moving Between Argumentation Frameworks

N. Oren[1] C. Reed[2] M. Luck[3]

[1]University of Aberdeen
n.oren@abdn.ac.uk

[2]University of Dundee
chris@computing.dundee.ac.uk

[3]King's College London
michael.luck@kcl.ac.uk

Introduction

- DAFs, NAFs, BAFs, PAFs, VAFs, EAFs, ...
- How do we translate between these different frameworks?
- What does translation mean?
- (is support useful?)
- Our focus here is on evidential argument frameworks.

Overview

1. Background: DAFs, NAFs, EAFs.
2. Brute force translation.
3. Refining the approach.
4. A Domain specific refinement.

DAFs and NAFs

- $\langle Args, Att \rangle$, $Att : Args \times Args$
- $\langle Args, NAtt \rangle$, $NAtt : 2^{Args} \setminus \emptyset \times Args$
- Acceptable: x is acceptable w.r.t S iff any attack against x is itself attacked by a subset of S.
- Conflict Free: S is conflict free if no subsets of S attack each other.
- Admissible: S is admissible if it is conflict free and all its elements are acceptable w.r.t itself.
- From this, we compute admissibility based extensions.

Translation

How can we represent an AF_1 as an AF_2 (e.g. a NAF as a DAF), while ensuring that if an argument is part of some extension in one, it is within an analogous extension within the other?

DAFs and NAFs

- Moving from DAFs to NAFs is “easy”: map *Args* across. For every $(x, y) \in Att$ add $(\{x\}, y)$ to *NAtt*.
- Moving from NAFs to DAFs is harder. Introduce “virtual arguments”, or “virtual attacks”?

EAFs, Intuition

- EAFs were devised with the long term goal of "semi-abstractly" modelling the evolution of the interactions of arguments over time. Main features:
  - Support
  - Collective attack and support
  - A special argument, η whose support is, in a sense, required for further support or attack to be successful.
- Support + η
  - Support facilitates knowledge representation. When modelling the dynamics of a system, arguments can be built up from smaller components.
  - Allow for arguments that have been introduced, but have not yet been "justified": η is used to differentiate such arguments from others.
  - η is used to support defaults, external evidence, (joint) commitment to some argument, etc.

Evidential Argument Frameworks (EAFs)

- $\langle Args, R_a, R_e \rangle$, $R_a, R_e : 2^{Args} \backslash \emptyset \times Args$
- $\eta \in Args$ such that $(,\eta) \notin R_a, R_e$ and $(\{\ldots,\eta\},) \notin R_a$.
- Evidential support (e-support) of an argument a from a set S: $SR_e a$ or $\exists T \subset S$ s.t. $TR_e a$ and $\forall x \in T$, x has evidential support from $S \backslash \{x\}$.
- Evidence supported attack by S on argument a : $XR_a a$ where $X \subseteq S$ and all elements of X have evidential support from S. The attack is minimal if there is no $T \subset S$ that carries out an evidence supported attack.
- Acceptability: a is acceptable w.r.t S iff S e-supports a, and for any minimal evidence supported attack by some X against a, $\exists T \subseteq S$ such that $TR_a x$ where $x \in X$ so that $X \backslash \{x\}$ no longer carries out an evidence supported attack on a.
- Self supporting: $\forall x \in S$, S e-supports x.
- we can define conflict free, admissibility (and from there, extensions) as usual.

EAFs, η

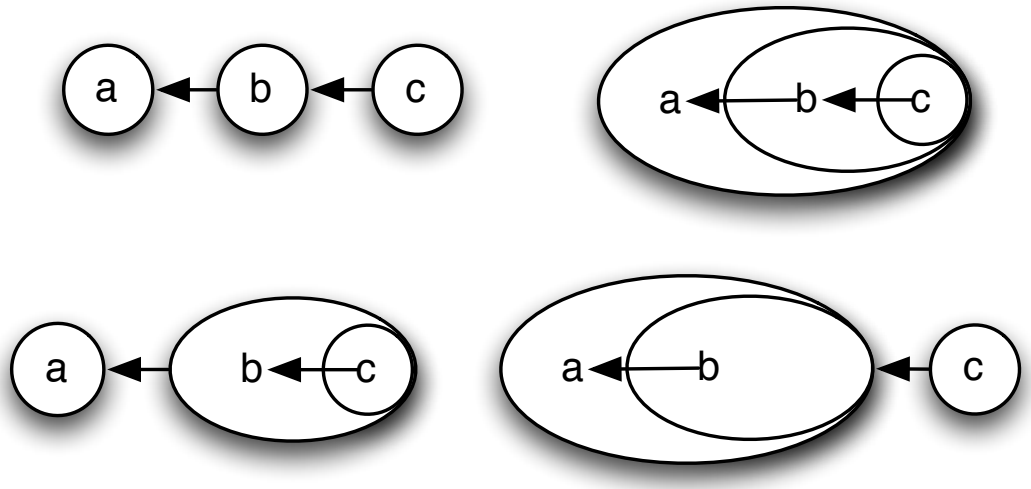
- For an argument to be considered for inclusion within an extension, or to successfully attack some other argument there must be a chain of support from η to the argument.
- η
  - eliminates the possibility of self supporting cycles.
  - Helps ensure consistency of the system.
  - Allows for a simple representation of argument schemes?

DAFs/NAFs to EAFs

- We have seen how DAFs can be mapped to NAFs.
- Mapping a NAF to a EAF is also easy.
  1. All arguments are mapped across.
  2. All attacks are mapped across.
  3. Add a link from η to each argument in the system.
- Given the additional notion of support, how do we map EAFs back to DAFs (or NAFs)?

EAFs to DAFs

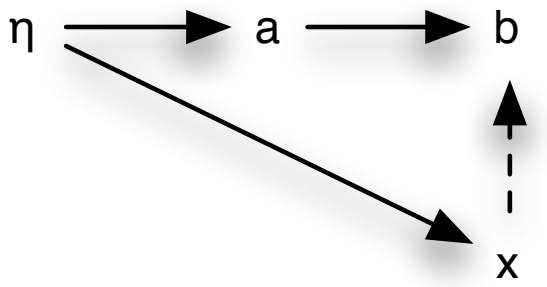
- Our approach makes use of the notion of support as an abstract inference link.
- An argument supporting another argument can be viewed as a subargument.

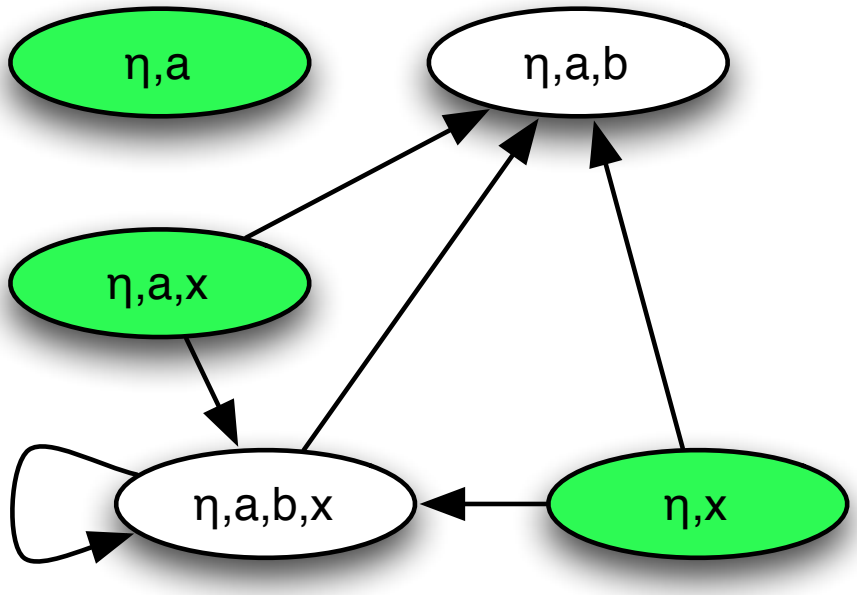


A Brute Force Approach

1. $EAF = \langle Args, R_a, R_e \rangle$
2. $DArgs = \{\}, DAtt = \{\}$
3. $DArgs =$ all self supporting subsets of $Args$
4. If there is an attack in R_a from X to a
   - add (D, A) to $DAtt$ where $X \subseteq D$ and $a \subseteq A$
5. $DAF = (DArgs, DAtt)$

A Brute Force Approach



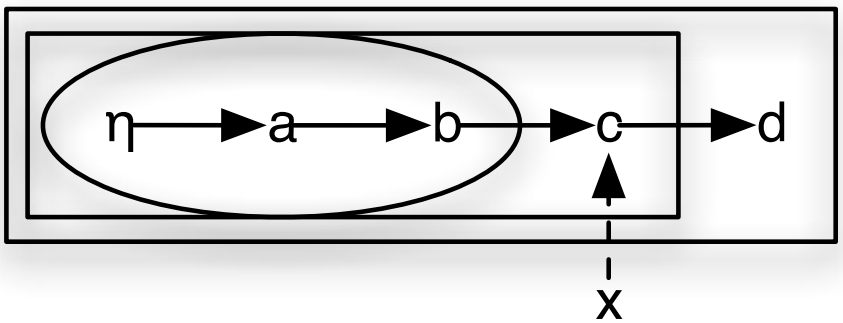


A Brute Force Approach

- If an argument is e-support attacked in the original EAF, any node (argument set) in the DAF containing the argument is attacked in the DAF.
- A set of arguments S' is admissible in the DAF iff all arguments found within the argument sets (i.e. nodes in the DAF) within S' are admissible within the EAF.
- Admissibility based semantics are thus preserved.
- In the worst case, given n arguments in the EAF 2^n nodes will exist in the DAF.
- In general, what nodes can we remove without eliminating the properties of translation?

Refining the Approach

- Definition: "X has the same status as Y": X is admissible iff Y admissible.
- Given an argument set $AS = \{a_1, \ldots, a_n\}$ and an attack from some argument against a_i,
  1. All self supporting subsets that can be formed from $AS \backslash \{a_i\}$ have the same status. Call these S_{i-1}.
  2. All subsets of $AS \backslash S_{i-1}$ have the same status.



Refining the Approach

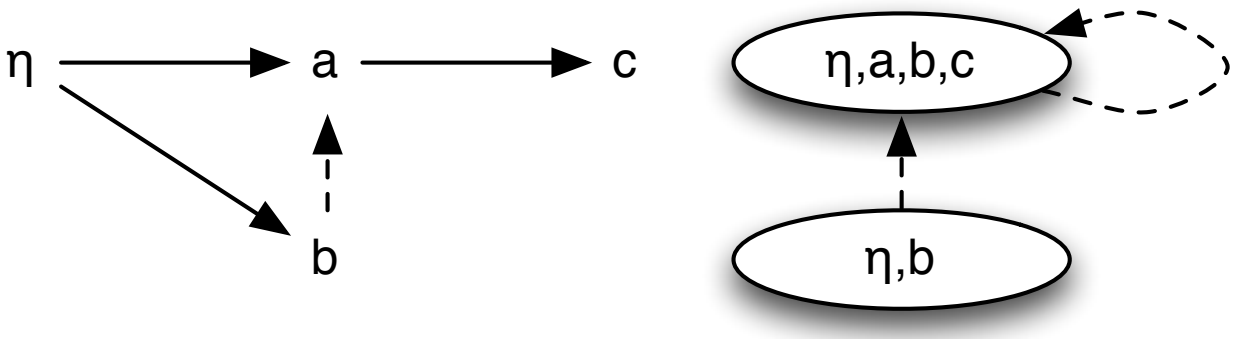
- Definition: "X has the same status as Y": X is admissible iff Y admissible.
- Given an argument set $AS = \{a_1, \ldots, a_n\}$ and an attack from some argument against a_i,
  1. All self supporting subsets that can be formed from $AS \backslash \{a_i\}$ have the same status. Call these S_{i-1}.
  2. All subsets of $AS \backslash S_{i-1}$ have the same status.
- Thus, a DAF containing only maximal argument sets, as well as subsets of the maximal argument sets up to the point in which the set is attacked, is sufficient to represent the original EAF system.
- Easy to compute in polynomial time.

Refining the Approach

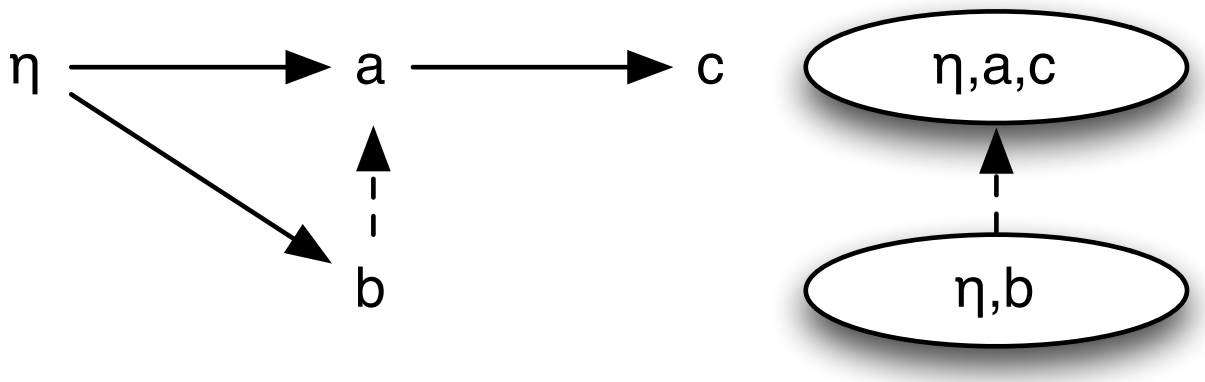
Require: An EAF $(Args, R_a, R_e)$

AC={all maximal self supporting argument sets of the EAF},Att={}
for all $(S, t) \in R_a$ **do**
  Let $TEAF = (Args \backslash \{t\}, R_a, R_e)$
  Add all maximal self supporting chains of $TEAF$ to AC
end for
for all $C \in AC$ **do**
  for all $D \in AC$ **do**
    if $(S, t) \in R_a$ such that $S \subseteq C, t \in D$ **then**
      Add (c, d) to Att
    end if
  end for
end for
return (AC, Att)

Refining the Approach



Refining the Approach



Further Improvements

- The problem stems from the fact that there will always be a single maximal argument set, starting at η.
- Solution: introduce "conceptually" different ηs.
- The way in which maximal self supporting chains of arguments are computed is modified.

Further Improvements

Require: An EAF $(Args, R_a, R_e)$

AC={all maximal self supporting argument sets of the EAF},Att={}
for all $(S, t) \in R_a$ **do**
    Let $TEAF = (Args \backslash \{t\}, R_a, R_e)$
    Add all maximal self supporting chains of $TEAF$ to AC
end for
for all $C \in AC$ **do**
    for all $D \in AC$ **do**
        if $(S, t) \in R_a$ such that $S \subseteq C, t \in D$ **then**
            Add (c, d) to Att
        end if
    end for
end for
return (AC, Att)

Further Improvements

Require: An EAF $(Args, R_a, R_e)$

AC={all maximal self supporting argument sets of the EAF},Att={}
for all $(S, t) \in R_a$ **do**
    Let $TEAF = (Args \backslash \{t\}, R_a, R_e)$
    ~~Add all maximal self supporting chains of *TEAF* to *AC*~~
end for
for all $C \in AC$ **do**
    for all $D \in AC$ **do**
        if $(S, t) \in R_a$ such that $S \subseteq C, t \in D$ **then**
            Add (c, d) to Att
        end if
    end for
end for
return (AC, Att)

Further Improvements

Require: $AF = (Args, R_a, R_e)$
1: $MaximalChains = \{\}$
2: **for all** $a \in Args$ **do**
3: $maxChain = computeMaxSupportedSet(a, \{\eta\}, AF)$
4: **if** $maxChain \cup \{\eta\}$ is a self supporting chain **then**
5: $MaximalChains = MaximalChains \cup maxChain \cup \{\eta\}$
6: **end if**
7: **end for**
8: **return** $MaximalChains$

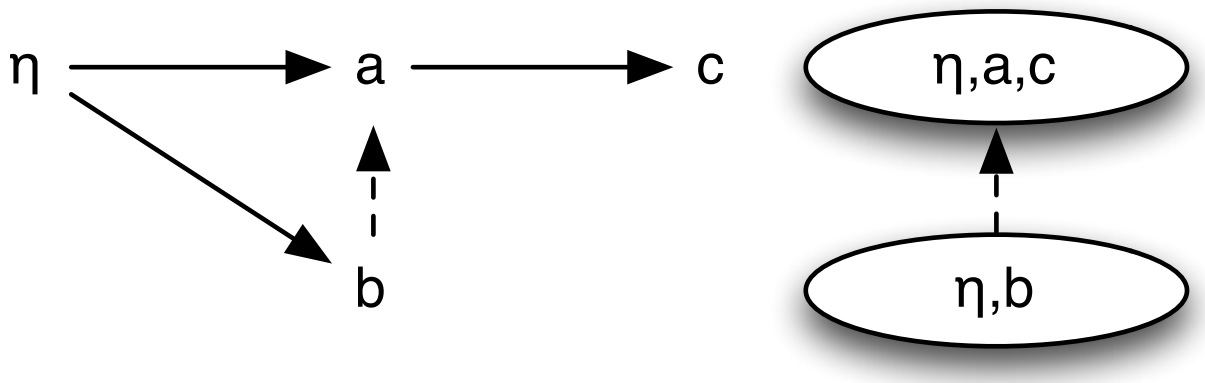
- Identify all maximal argument chains that contain a, check if they are self supporting and add η.

Further Improvements

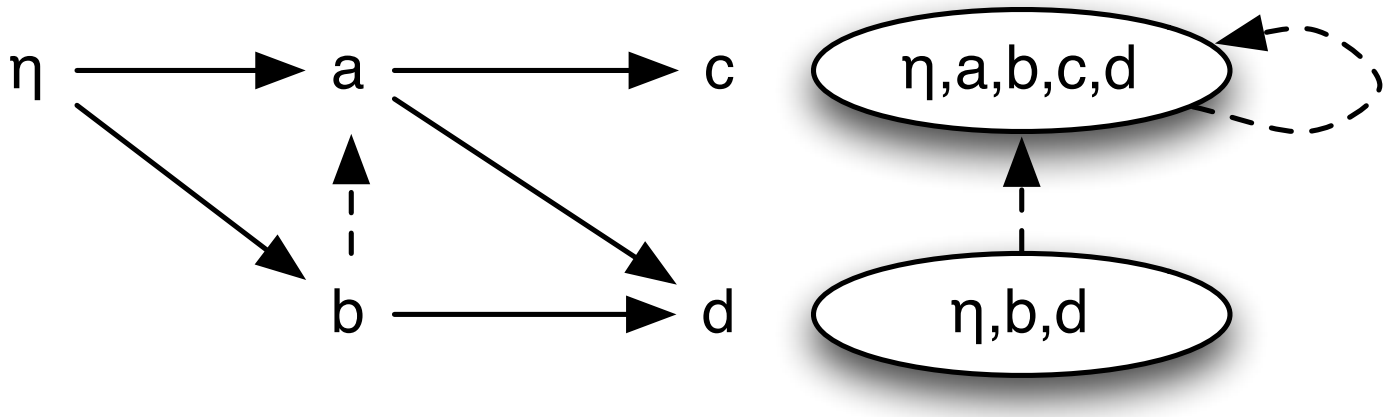
Require: a, an argument
Require: $Visited \subseteq Args$, a set of already visited nodes
Require: $AF = (Args, R_a, R_e)$
1: $Answer = \{a\}, ToVisit = \{\}$
2: $Visited = Visited \cup \{a\}$
3: **for all** $B \subseteq Args$ such that $(B, a) \in R_e$ **do**
4: $\quad ToVisit = ToVisit \cup B$
5: **end for**
6: **for all** $c \in Args$ such that $(\{\ldots, a, \ldots\}, c) \in R_e$ **do**
7: $\quad ToVisit = ToVisit \cup c$
8: **end for**
9: **for all** $d \in ToVisit \setminus Visited$ **do**
10: $\quad Answer = Answer \cup computeMaxSupportedSet(d, Visited, AF)$
11: **end for**
12: **return** $Answer$

- Starting at a, find supported and supporting arguments, and add those to the chain. Rinse, repeat (while avoiding support chain loops)...

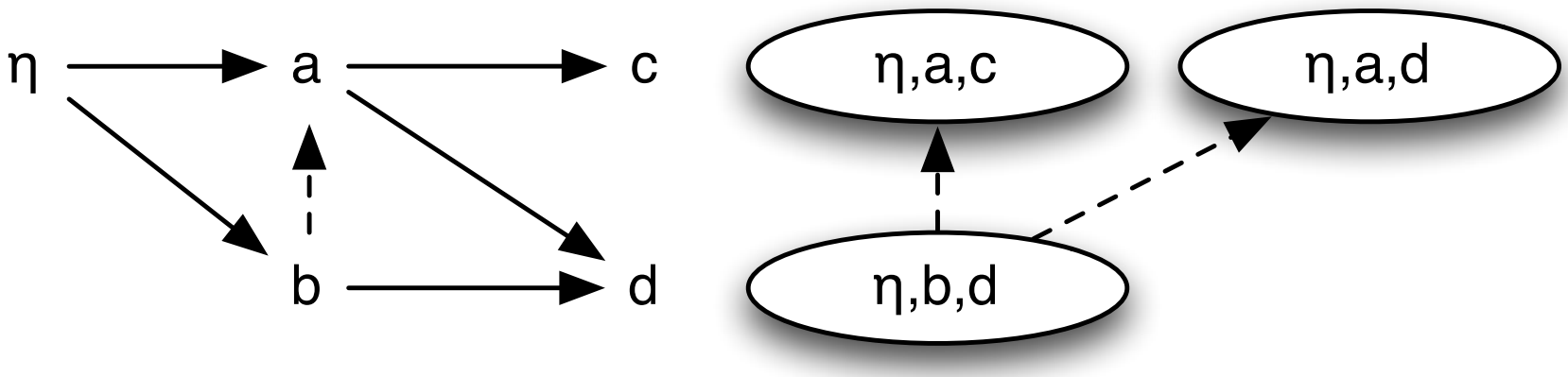
Further Improvements



Further Improvements



Further Improvements



- *a*, *c* and *a*, *d* are (at a gut feel level) independent, and should be in separate nodes.

Further Improvements II

Require: An EAF $AF = (Args, R_a, R_e)$

1: **for all** $a \in Args$ **do**
2: $\quad Answer = Answer \cup computeBackSet(a, \{\}, AF)$
3: **end for**
4: **for all** $AS \in Answer$ **do**
5: $\quad$ **if** $\eta \notin AS$ **then**
6: $\quad\quad Answer = Answer \backslash \{AS\}$
7: $\quad$ **end if**
8: $\quad$ **if** $\exists AS' \in Answer$ such that $AS \subseteq AS'$ **then**
9: $\quad\quad Answer = Answer \backslash \{AS\}$
10: $\quad$ **end if**
11: **end for**
12: **return** *Answer*

- Idea: work backwards from an argument to η, ensuring that we can actually get to η and that the chain is maximal.

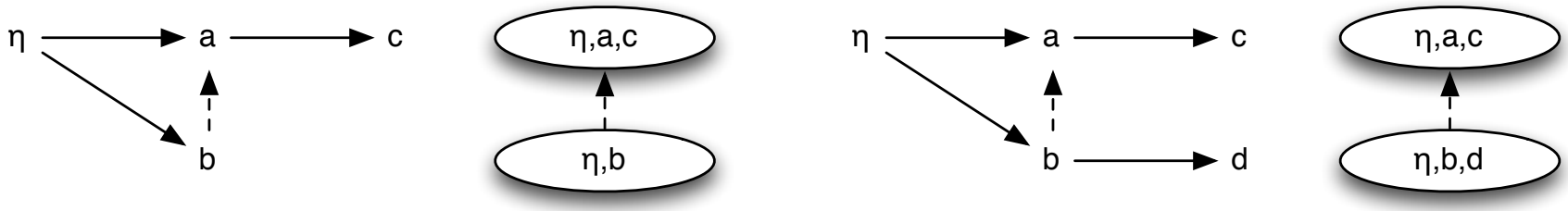
Further Improvements II

Require: An argument a
Require: A set of visited edges $Visited \subseteq R_e$
Require: An EAF $AF = (Args, R_a, R_e)$
1: **for all** $(X, a) \in R_e$ such that $(X, a) \notin Visited$ **do**
2: $B = \times_{x_i} computeBackSet(x_i, Visited \cup \{(X, a)\}, AF)$ for $x_i \in X$
3: **for all** $b \in B$ **do**
4: $b = b \cup \{a\}$
5: **end for**
6: $Ans = Ans \cup B$
7: **end for**
8: **return** Ans

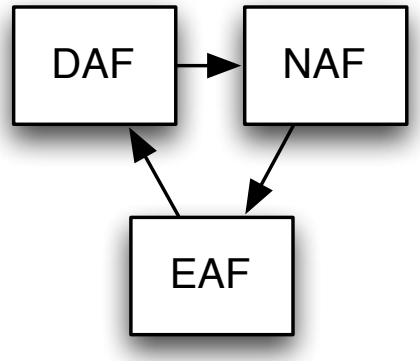
- Recursively work backwards from an argument.
- Work over all supporting nodes, taking into account the possibility of collective support.
- Ensure that loops are not repeatedly visited.

Discussion

- Multiple EAFs have the same DAF graph structure.
- Intuitively, some information is lost when going from the more specific (EAFs) to the more abstract.



Discussion



- We have shown how to translate between DAFs, NAFs and EAFs.
- While there is information loss between the approaches, the polynomial time translation between them suggests that support has no extra representational power.
- BUT
  - support is useful as a KR tool.
  - support may help overcome representational difficulties and inconsistencies.
- Loss of information when moving from EAFs and NAFs to DAFs