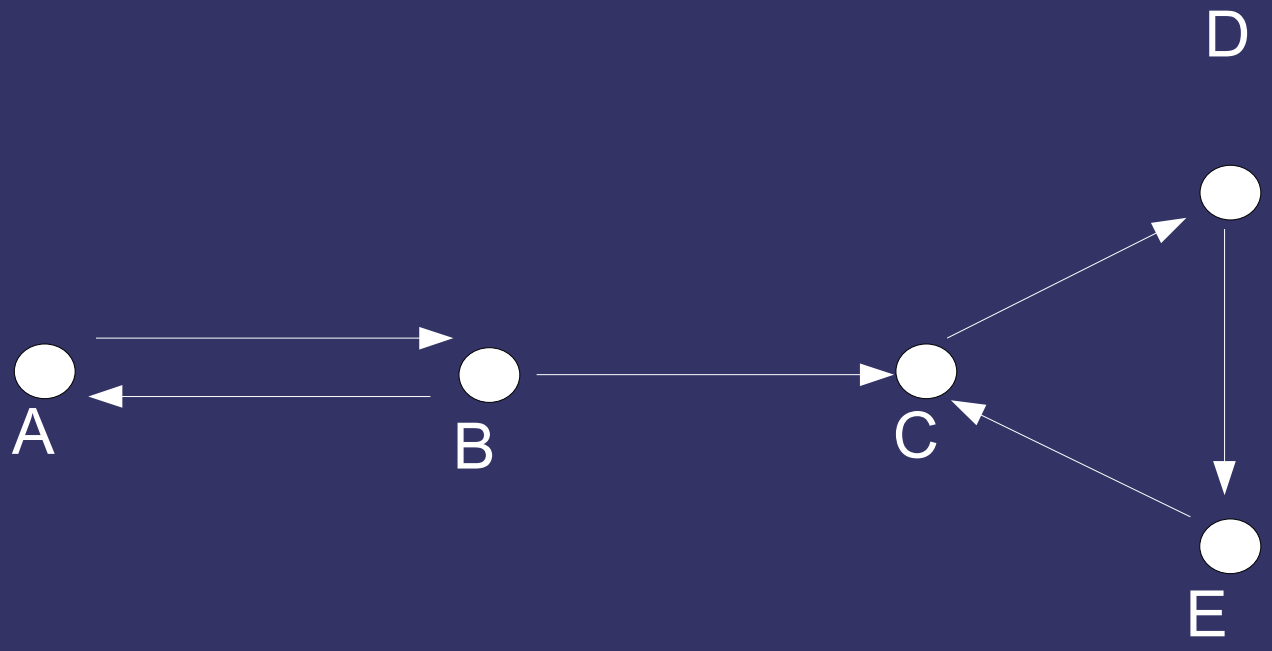
An algorithm for computing Semi-Stable Semantics

Martin Caminada

University of Luxembourg

Abstract Argumentation

Argumentation Framework: (Arguments, Defeat)



Args+ = { A | ∃ B∈Args: B defeats A }

example: {B, D}+ = {A, C, E}

Argumentation Semantics

Let AF = (Arguments, Defeat)
A set of arguments *Arg* is

- stable iff *Args*+ = Arguments \ *Args*

- preferred iff it is an admissible set where *Args* is maximal

Argumentation Semantics

Let AF = (Arguments, Defeat)
A set of arguments *Arg* is

- stable iff it is an admissible set with *Args* ∪ *Args*+ = Arguments

- preferred iff it is an admissible set where *Args* is maximal

Argumentation Semantics

Let AF = (Arguments, Defeat)
A set of arguments *Arg* is

- stable iff it is an admissible set with *Args* ∪ *Args*+ = Arguments
- semi-stable iff it is an admissible set where *Args* ∪ *Args*+ is maximal
- preferred iff it is an admissible set where *Args* is maximal

Argumentation Semantics

Let AF = (Arguments, Defeat)
A set of arguments *Arg* is

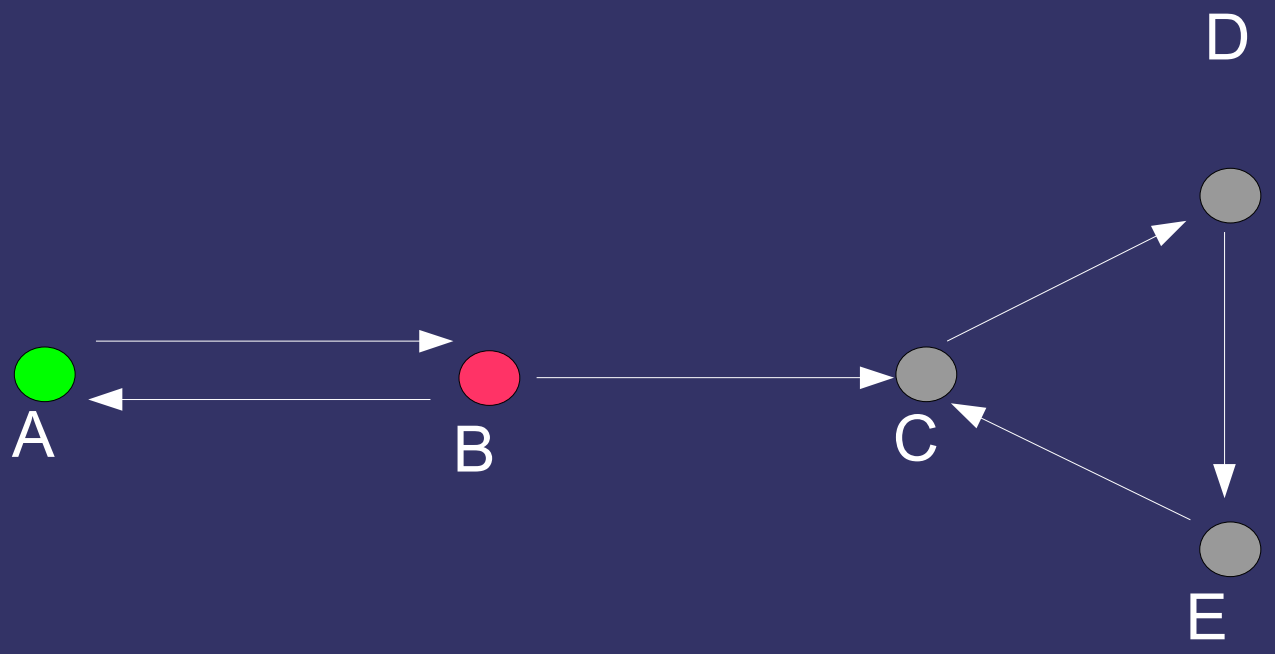
- stable iff it is an admissible set with *Args* ∪ *Args*+ = Arguments
- semi-stable iff it is an admissible set where *Args* ∪ *Args*+ is maximal
- preferred iff it is an admissible set where *Args* is maximal

each stable is a semi-stable is a preferred

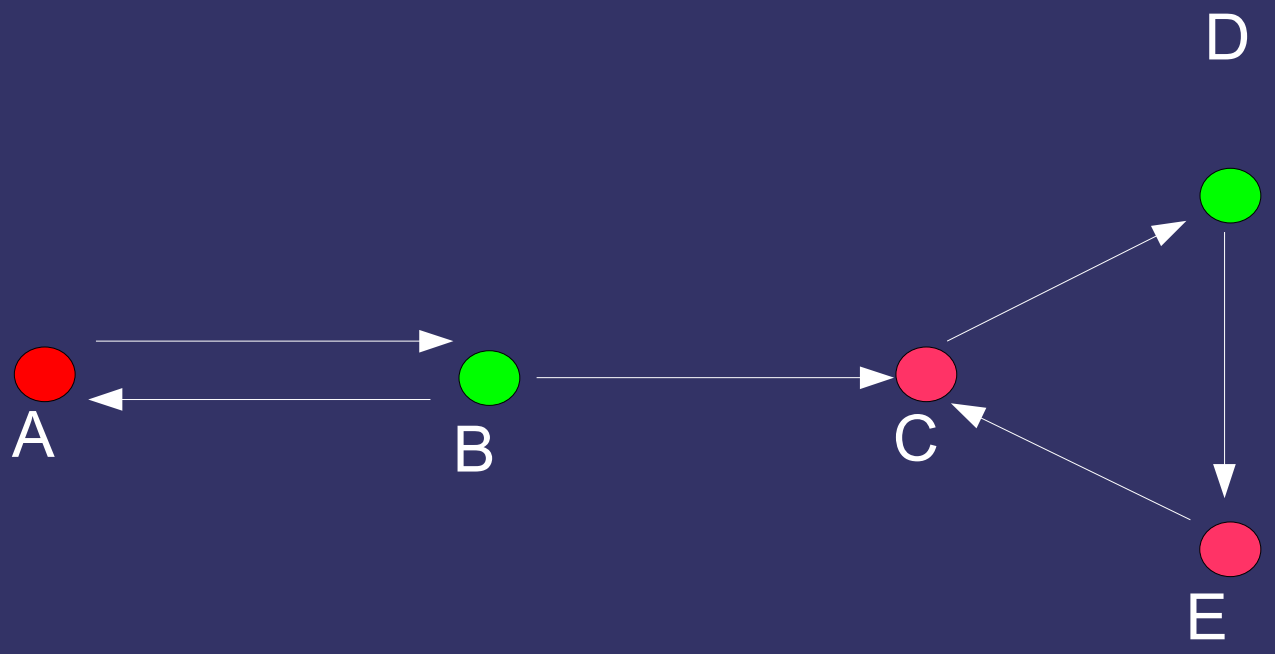
Argument Labellings

- A *labelling* assigns to each argument exactly one label: in, out or undec
- A *complete labelling* is a labelling that satisfies:
  - if an argument is in,
    then all its defeaters are out
  - if an argument is out,
    then it has at least one defeater that is in
  - if an argument is undec,
    then not all its defeaters are out
    and it does not have a defeater that is in

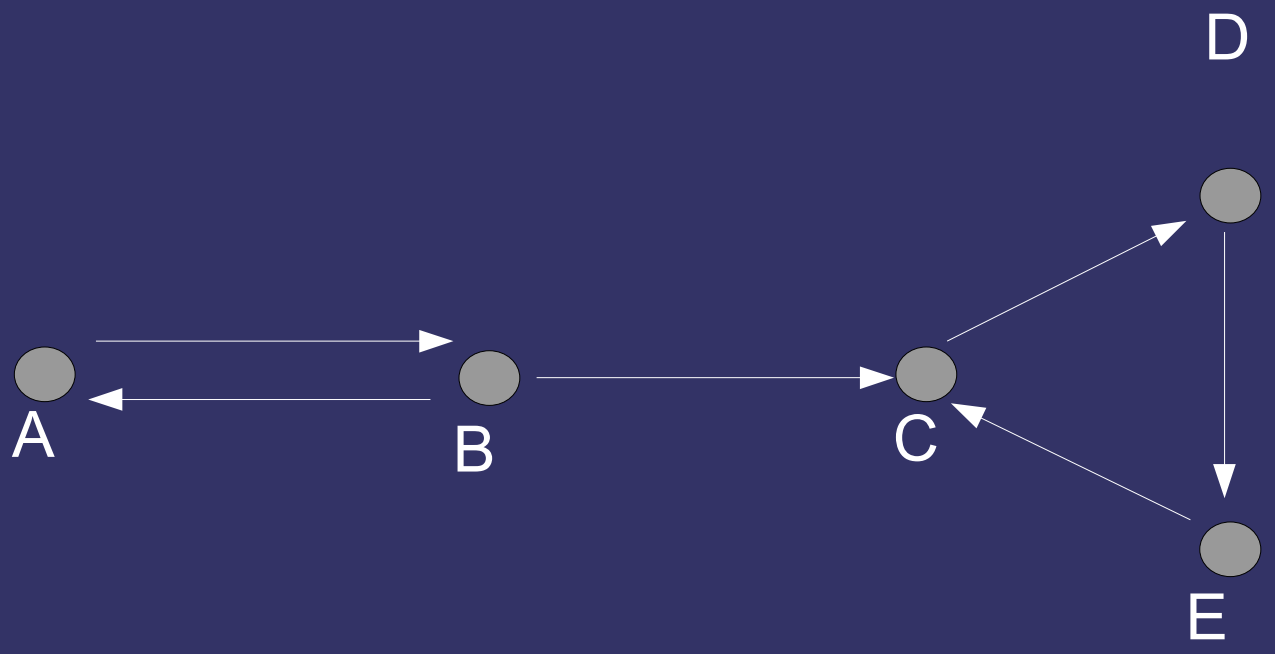
Example



Example



Example



Semantics and Labellings

restriction on complete labeling	*Dung-style semantics*
no restrictions	complete
no undec	stable
maximal in	preferred
maximal out	preferred
maximal undec	grounded
minimal in	grounded
minimal out	grounded
minimal undec	semi-stable

Complete Labellings

- A *labelling* assigns to each argument exactly one label: in, out or undec
- A *complete labelling* is a labelling that satisfies:
  - if an argument is in,
    then all its defeaters are out
  - if an argument is out,
    then it has at least one defeater that is in
  - if an argument is undec,
    then not all its defeaters are out
    and it does not have a defeater that is in

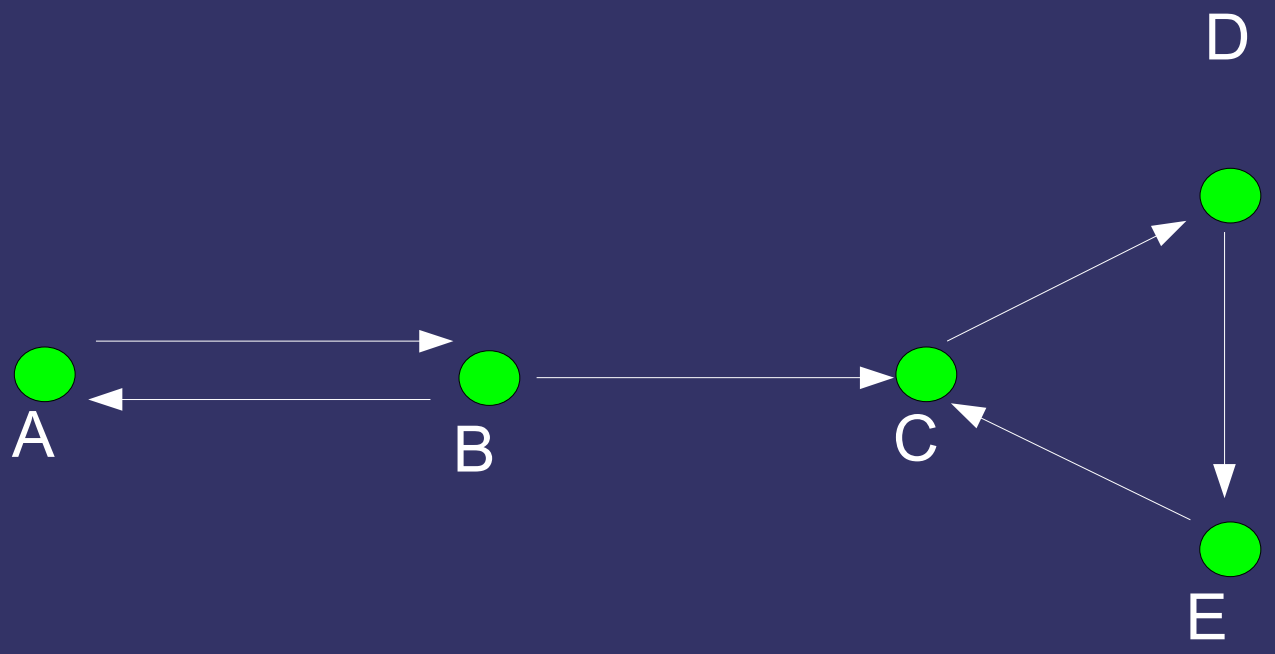
Admissible Labellings

- A *labelling* assigns to each argument exactly one label: in, out or undec
- An *admissible labelling* is a labelling that satisfies:
  - if an argument is in,
    then all its defeaters are out
  - if an argument is out,
    then it has at least one defeater that is in

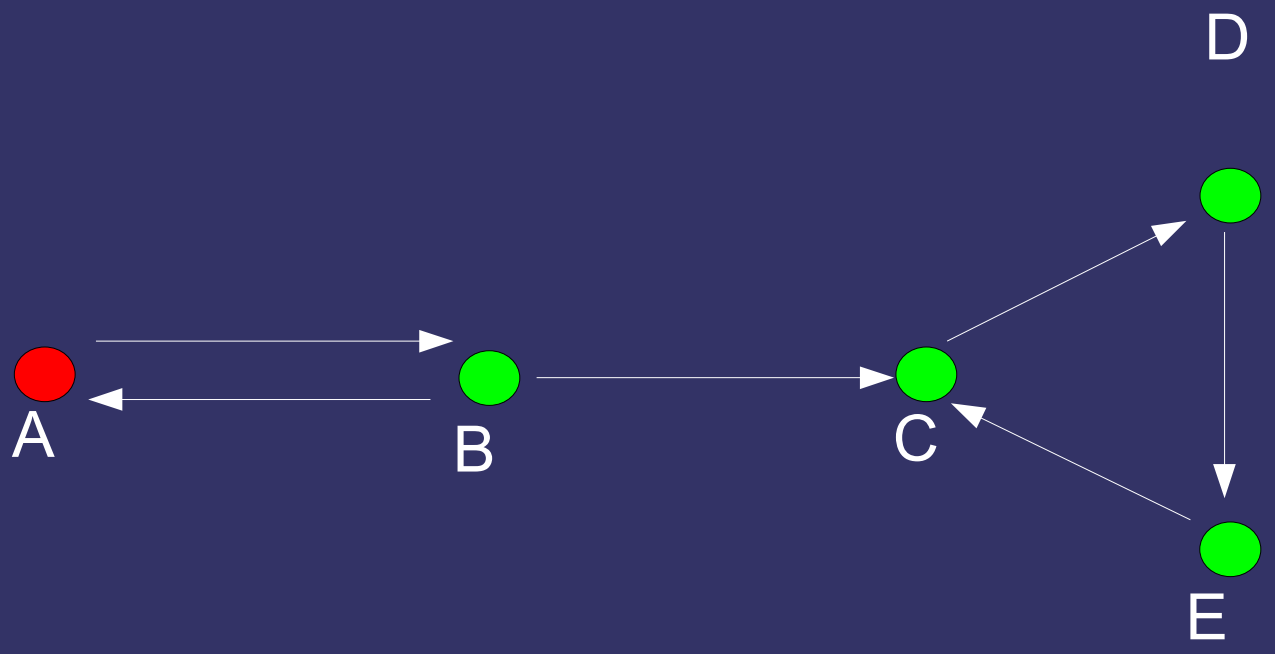
Outline of the Algorithm

- start with the *all-in labelling*
- then, as long as there are illegal ins:
  - select an argument (A) that is illegally in
  - relabel it to out
  - check if {A} ∪ {A}+ contains any arguments that have become illegally out;
    if so, relabel them to undec
- If there are no illegal ins anymore,
  then we have an admissible labelling
  (don't forget to backtrack to other possibilities)

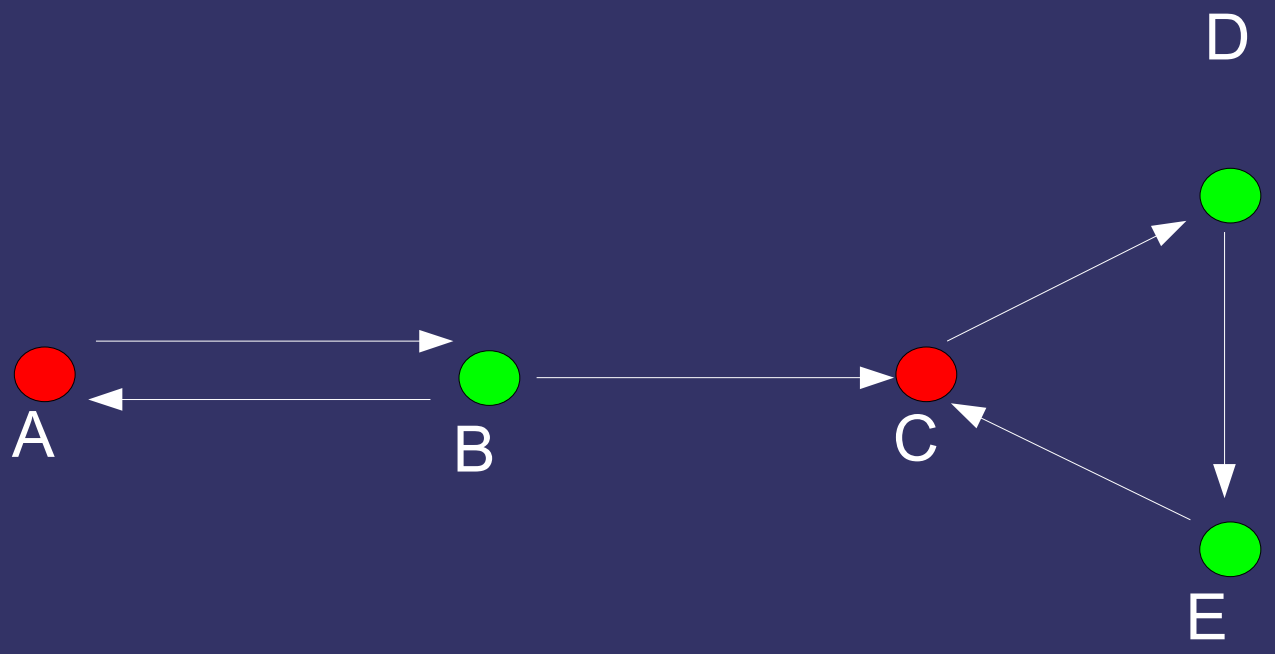
Example



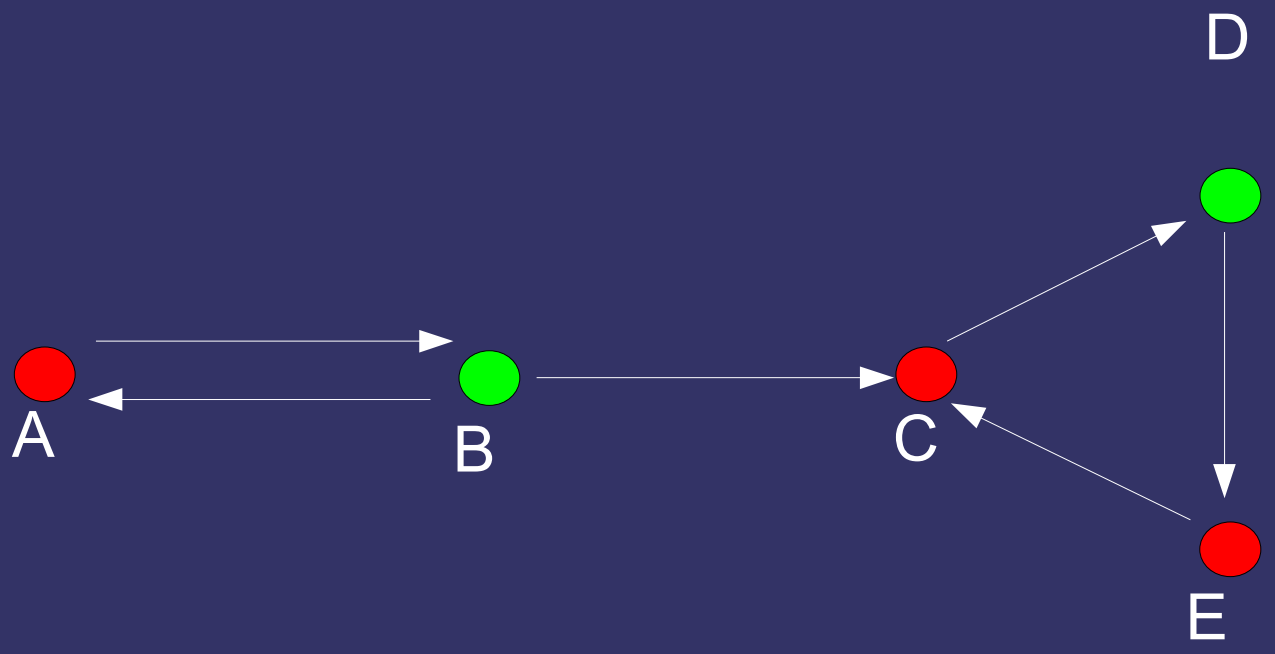
Example



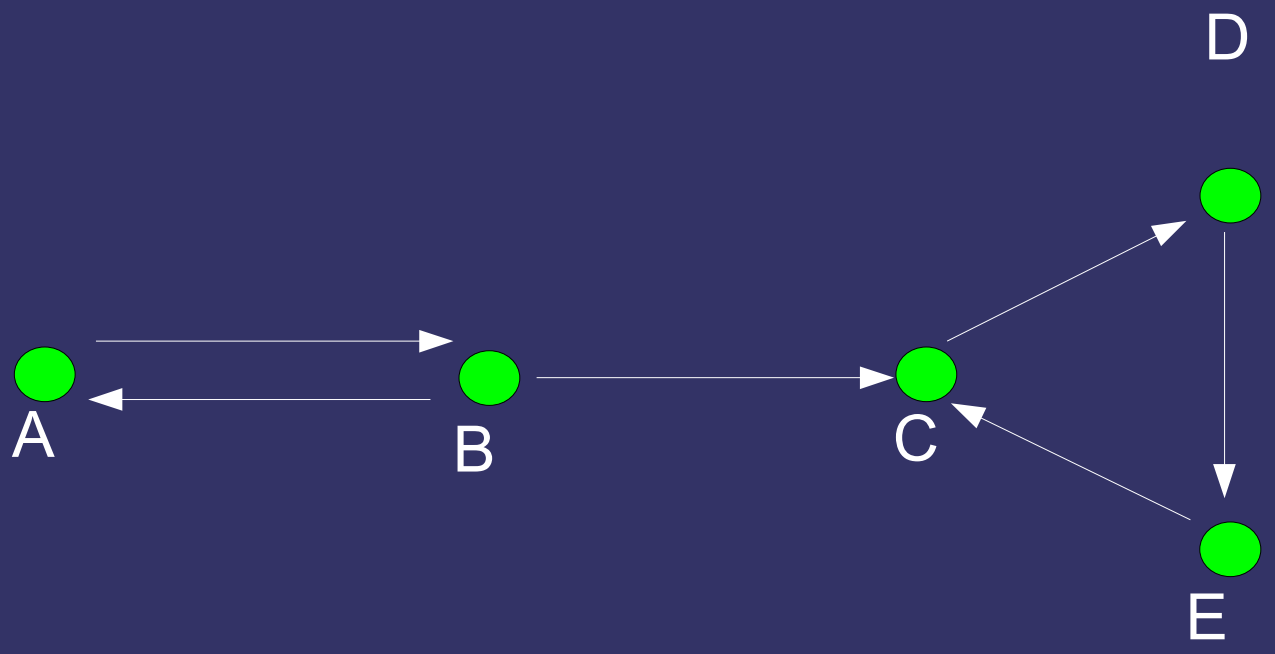
Example



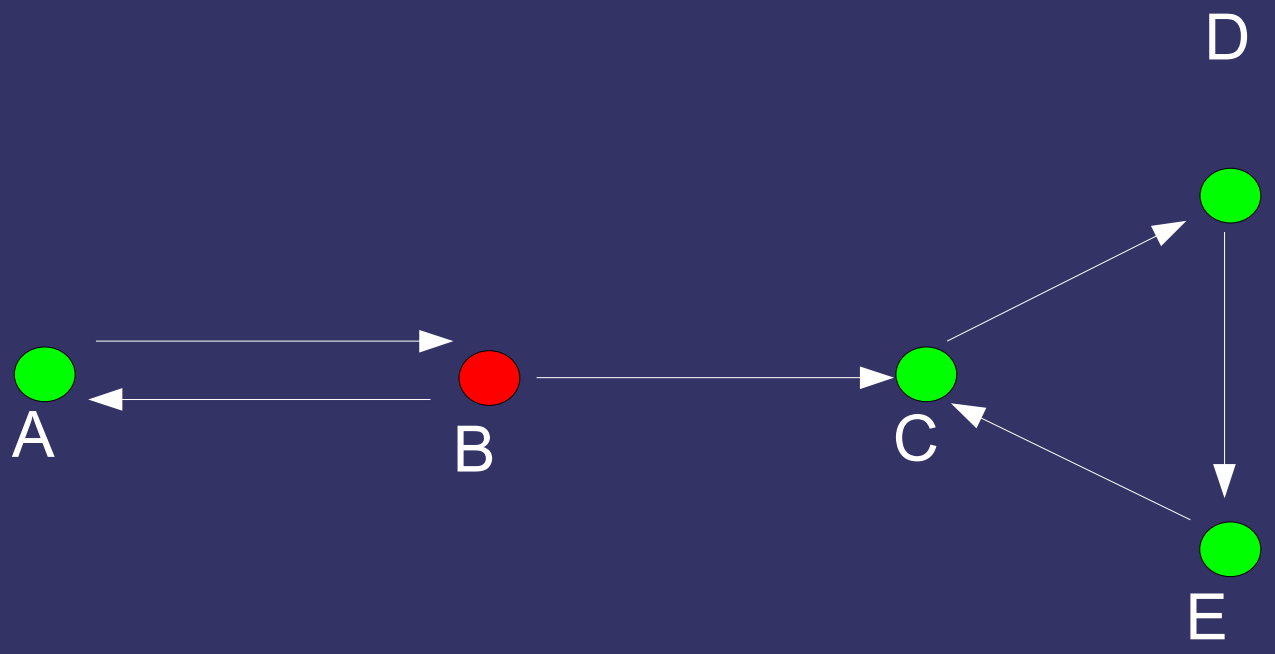
Example



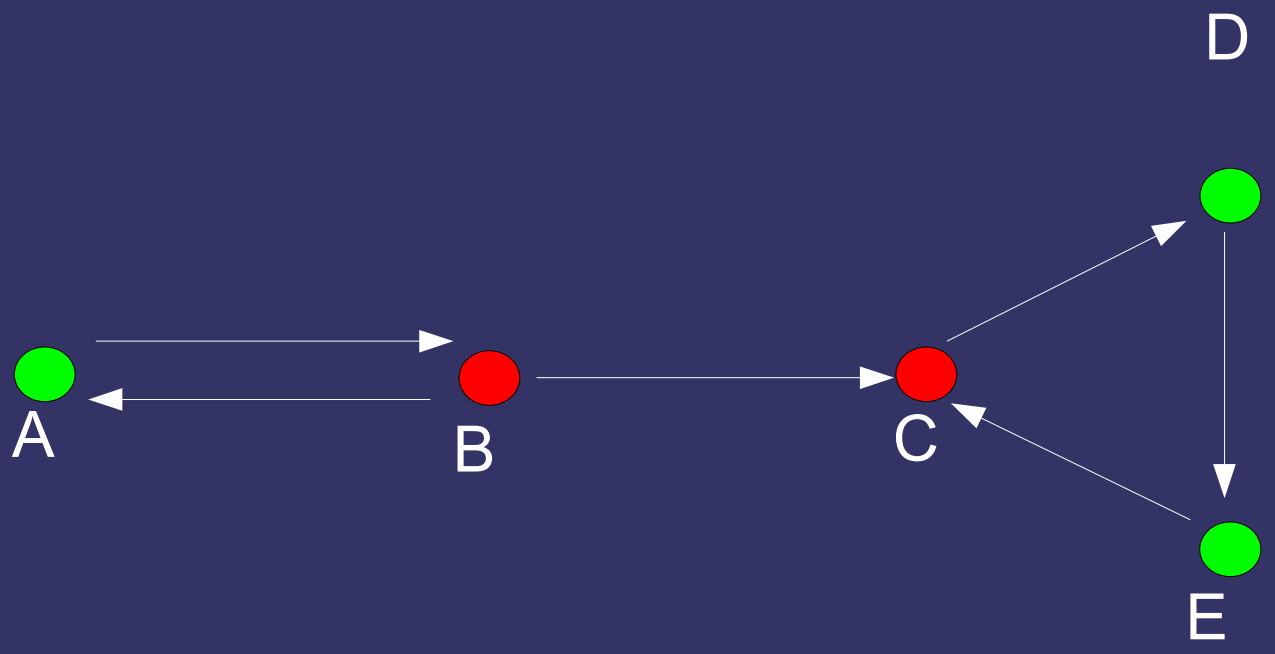
Example



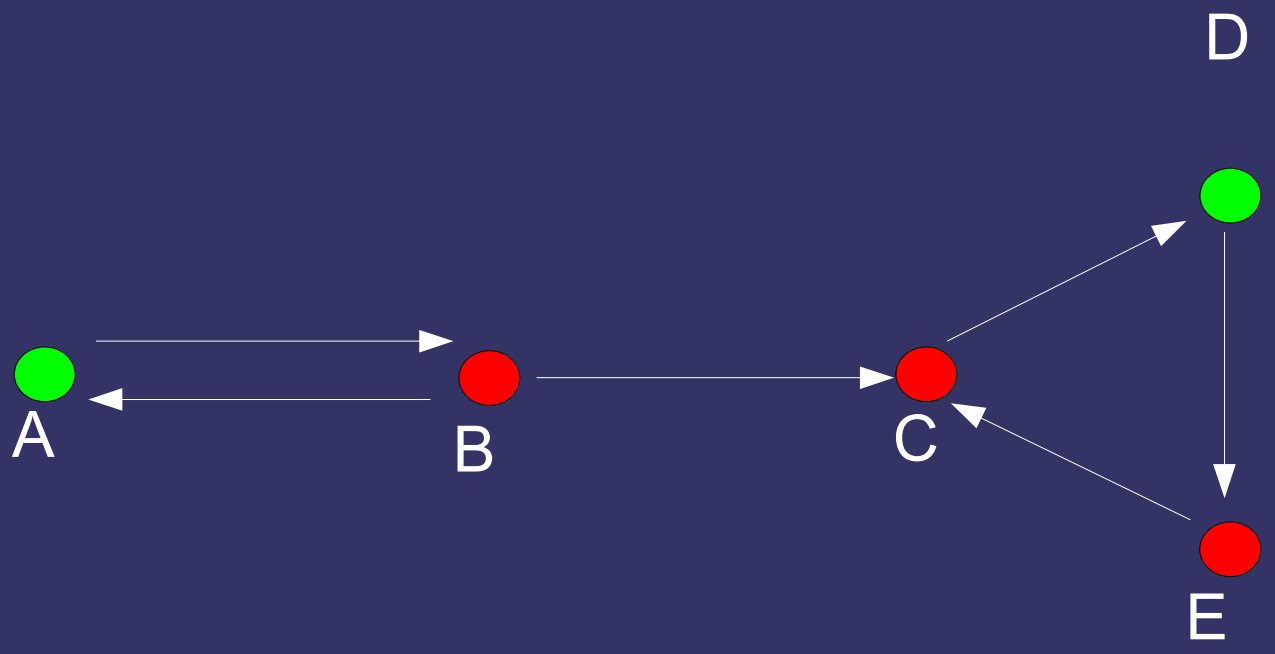
Example



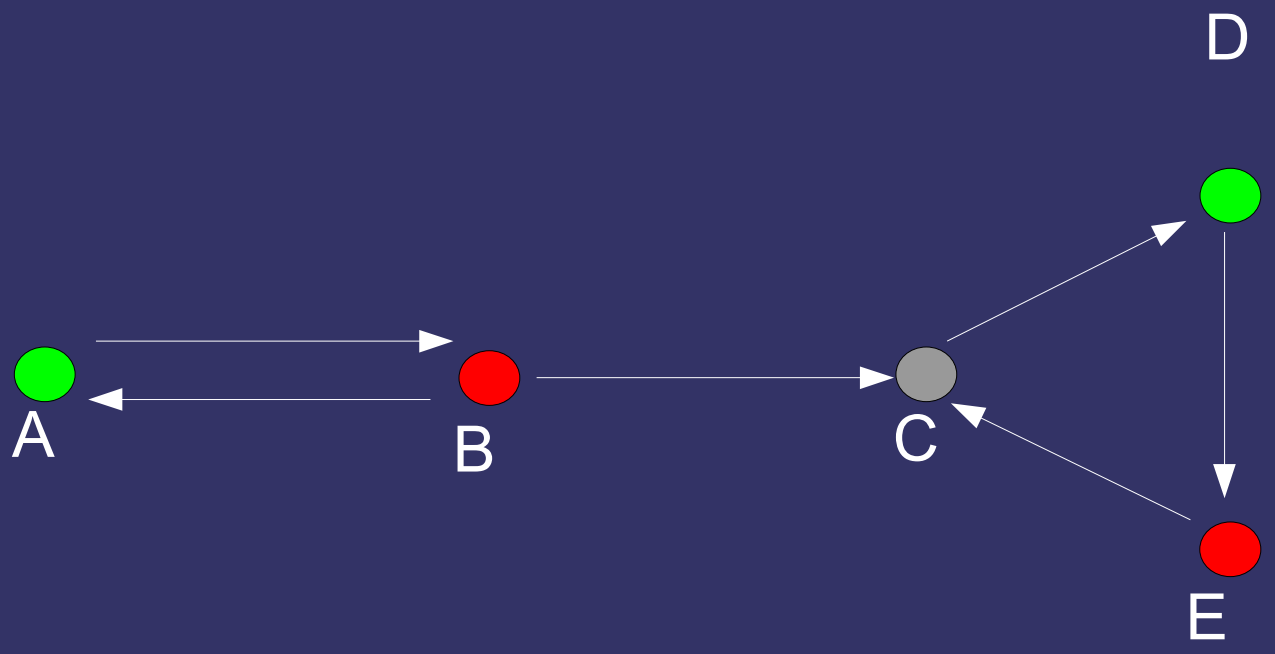
Example



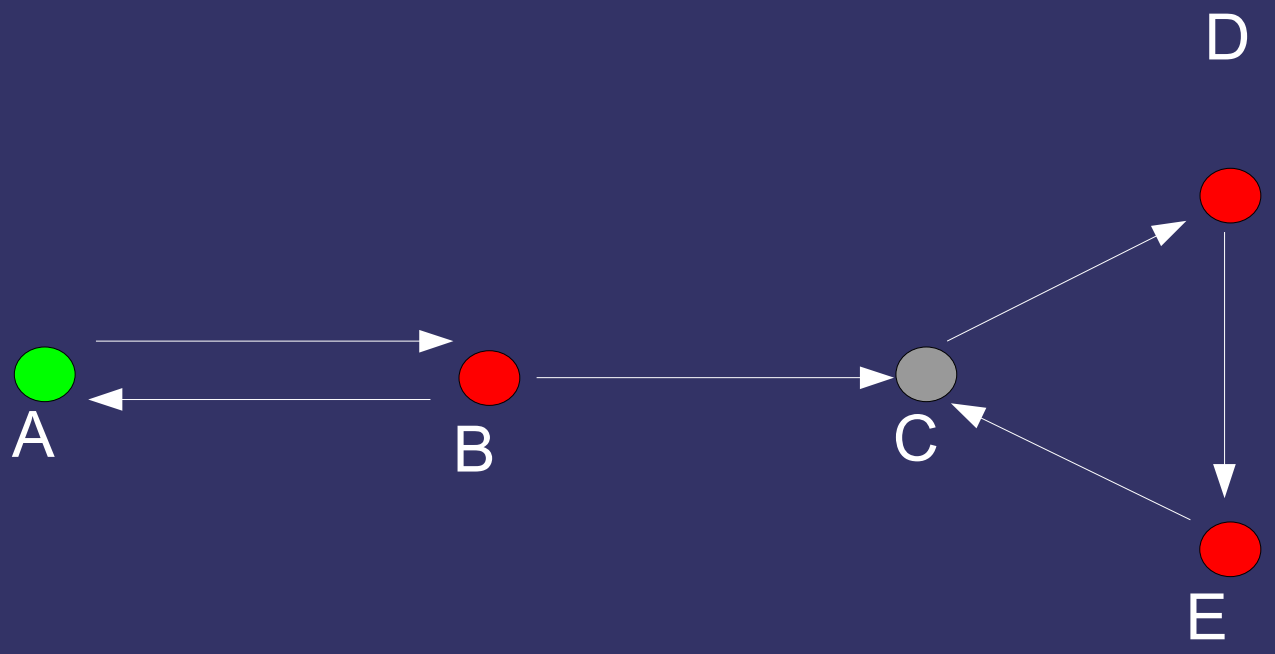
Example



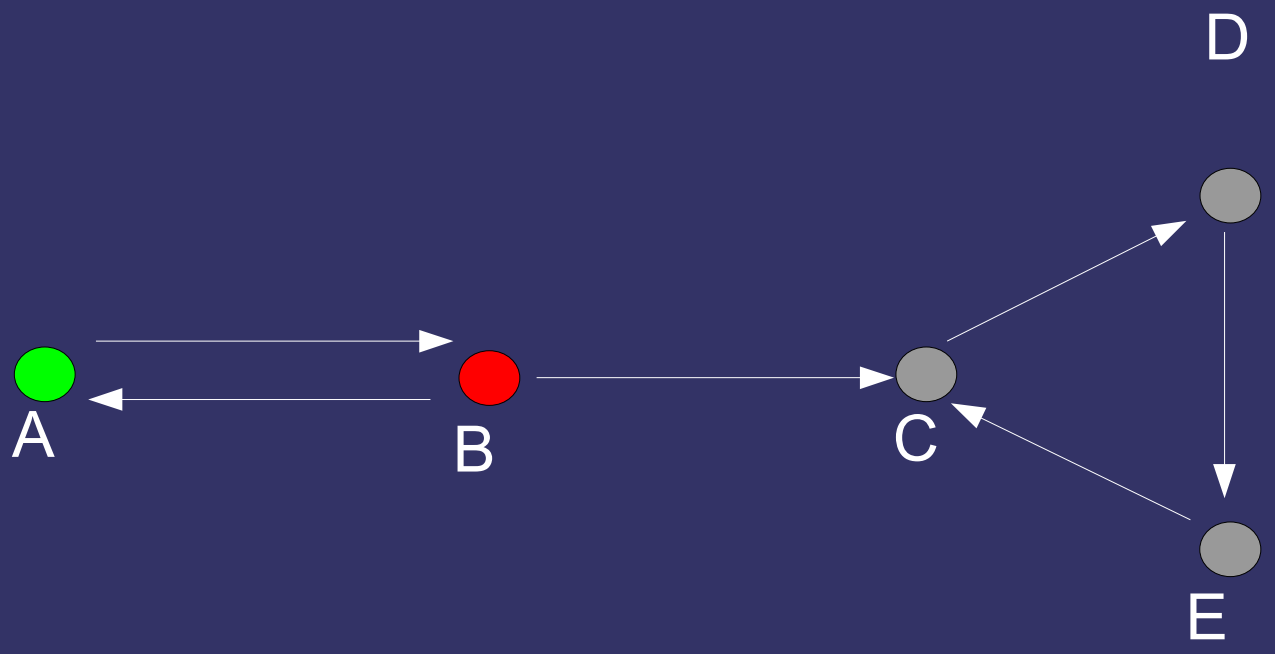
Example



Example



Example



Final Remarks

- The trick of transition steps is:
  - they preserve the absence of illegal outs
  - the set of in-labelled arguments decreases
  - the set of undec-labelled arguments increases
- Several slightly different versions of the algorithm can be used for:
  - computing the stable extensions
  - computing the semi-stable extensions
  - computing the preferred extensions