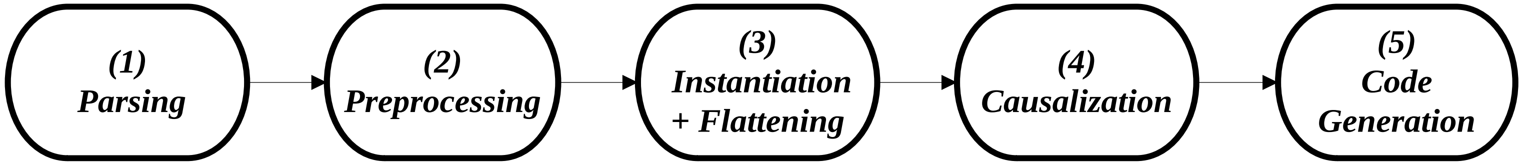
Virtual Physics Equation-Based Modeling

TUM, November 15, 2022

Compiling Modelica

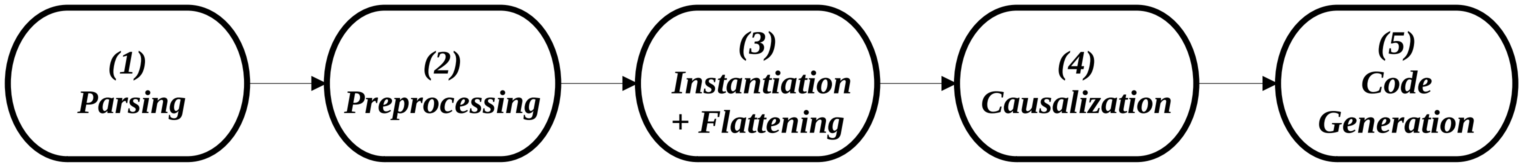


Dr. Dirk Zimmer

German Aerospace Center (DLR), Robotics and Mechatronics Centre

Compiling of Modelica

- In this lecture, we are going to investigate the compilation process of a Modelica model.



- The compilation can be roughly partitioned in 5 stages:
  - Parsing
  - Preprocessing
  - Flattening
  - Transformation into State-Space Form
  - Code Generation

Parsing

- The grammar of Modelica is specified as EBNF in the language definition.
- Modelica is not a LL(1)-language, so Parsing involves a few difficulties.
- Still, the language is rather easy to parse and no special means are required.

Preprocessing

- Preprocessing applies the means of type generation.
- Essentially, this concerns the **`extends`** clause and a few other language means that we do not know yet.
- Implementing the extension is not trivial, since an extension of a package may generate new type names. Hence the lookup of classes and the extension may happen in several interleaved steps.
- Another difficulty is that the inherited elements also inherit the namespace.

Flattening

- Flattening means that the hierarchical model structure is destroyed and that all parameters, variables and equations are collected in one global set.
- Also the connections are transformed into equations.
- Function definitions are typically not flattened.
- The flattened model then represents a (potentially very large) systems of DAEs.

Flattening

Robotics and Mechatronics Centre

```
model Circuit
  Resistor R1(R=100);
  Resistor R2(R=20);
  Capacitor C(C=1e-6);
  Inductor L(L=0.0015;
  SineVSource S(Ampl=15, Freq=50);
  Ground G;


equations
  connect(G.p,S.n)
  connect(G.p,L.n)
  connect(G.p,R2.n)
  connect(G.p,C.n)
  connect(S.p,R1.p)
  connect(S.p,L.p)
  connect(R1.n,R2.p)
  connect(R1.n,C.p)






end Pin;
```

$\rightarrow$

```
model Circuit
  parameter Real R1.R = 100;
  parameter Real R2.R = 20;
  …
  Real R1.v;  Real R1.i; Real R1.p.v;
  Real R1.p.i; Real R1.n.v;  Real R1.n.i;
  Real R2.v;  Real R2.i; …
  …
  …
equations
  R1.v = R1.R*R1.i;
  R1.v = R1.p.v - R1.n.v;
  0 = R1.p.i + R1.n.i;
  R1.i = R1.p.i;
  R2.v = R2.R*R2.i;
  …
  …
  G.p.v = S.n.v;
  G.p.v = L.n.v;
  G.p.v = R2.n;
  G.p.i + S.n.i + L.n.i + R2.n.i + C.n.i
= 0;
  …
  …
end Pin;
```

Into State-Space Form

- A system of DAEs can typically be represented in the following implicit form:

$$\mathbf{0} = \mathrm{F}(\mathrm{d}\mathbf{x}/\mathrm{d}t, \mathbf{x}, \mathbf{u}, t)$$

- The goal is, to transform this form into the explicit state-space form that is suited for numerical ODE solvers.

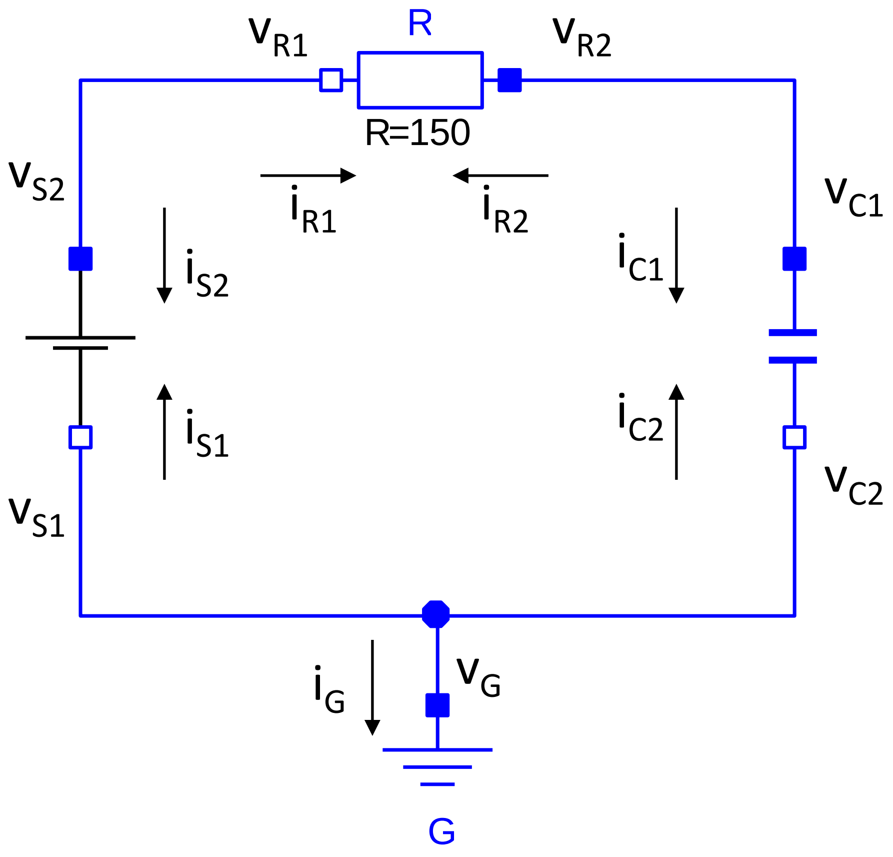
$$\mathrm{d}\mathbf{x}/\mathrm{d}t = \mathrm{f}(\mathbf{x}, \mathbf{u}, t)$$

- This transformation is also called **Index-Reduction**.
- An efficient index-reduction is the heart of any Modelica Compiler and its realization defines the remaining content of this lecture.

Example

Robotics and Mechatronics Centre

Let us review the simple electric circuit from lecture 2.



Flat System of DAEs

Robotics and Mechatronics Centre

If we flatten this system, we know that 16 equations result:

$$v_{S2} = v_{R1}$$
$$i_{S2} + i_{R1} = 0$$

$$v_{R2} = v_{C1}$$
$$i_{R2} + i_{C1} = 0$$

$$v_{C2} = v_G$$
$$v_{S1} = v_G$$
$$i_{C2} + i_{S1} + i_G = 0$$

Node equations

$$v_G = 0$$

$$i_{S1} + i_{S2} = 0$$
$$v_{S1} + 10V = v_{S2}$$

$$u_R = R \cdot i_{R1}$$
$$I_{R1} + I_{R2} = 0$$
$$v_{R1} + u_R = v_{R2}$$

$$C \cdot du_C/dt = i_{C1}$$
$$I_{C1} + I_{C2} = 0$$
$$v_{C1} + u_C = v_{C2}$$

Component Equations

Removing Alias Variables

The first thing we can do, is to eliminate the trivial equations by removing "alias" variables.

$v_{S2} = v_{R1}$
$i_{S2} + i_{R1} = 0$

$v_{R2} = v_{C1}$
$i_{R2} + i_{C1} = 0$

$v_{C2} = v_G$
$v_{S1} = v_G$
$i_{C2} + i_{S1} + i_G = 0$

Node equations

$v_G = 0$

$i_{S1} + i_{S2} = 0$
$v_{S1} + 10V = v_{S2}$

$u_R = R \cdot i_{R1}$
$I_{R1} + I_{R2} = 0$
$v_{R1} + u_R = v_{R2}$

$C \cdot du_C/dt = i_{C1}$
$I_{C1} + I_{C2} = 0$
$v_{C1} + u_C = v_{C2}$

Component Equations

Removing Alias Variables

Robotics and Mechatronics Centre

The first thing we can do, is to eliminate the trivial equations by removing "alias" variables.

$\cancel{v_{S2} = v_{R1}}$
$i_{S2} + i_{R1} = 0$

$\cancel{v_{R2} = v_{C1}}$
$i_{R2} + i_{C1} = 0$

$\cancel{v_{C2} = v_G}$
$\cancel{v_{S1} = v_G}$
$i_{C2} + i_{S1} + i_G = 0$

Node equations

$v_G = 0$

$i_{S1} + i_{S2} = 0$
$v_G + 10V = v_{R1}$

$u_R = R \cdot i_{R1}$
$I_{R1} + I_{R2} = 0$
$v_{R1} + u_R = v_{C1}$

$C \cdot du_C/dt = i_{C1}$
$I_{C1} + I_{C2} = 0$
$v_{C1} + u_C = v_G$

Component Equations

Removing Alias Variables

Robotics and Mechatronics Centre

The first thing we can do, is to eliminate the trivial equations by removing "alias" variables.

~~$v_{S2} = v_{R1}$~~
$-i_{S1} + i_{R1} = 0$

~~$v_{R2} = v_{C1}$~~
$-i_{R1} + i_{C1} = 0$

~~$v_{C2} = v_G$~~
~~$v_{S1} = v_G$~~
$-i_{C1} + i_{S1} + i_G = 0$

Node equations

$v_G = 0$

~~$i_{S1} + i_{S2} = 0$~~
$v_G + 10V = v_{R1}$

$u_R = R \cdot i_{R1}$
~~$I_{R1} + I_{R2} = 0$~~
$v_{R1} + u_R = v_{C1}$

$C \cdot du_C/dt = i_{C1}$
~~$I_{C1} + I_{C2} = 0$~~
$v_{C1} + u_C = v_G$

Component Equations

Removing Alias Variables

In total there remain 9 equations with 9 variables. (actually we could reduce even further…)

$$v_G = 0$$

$$v_G + 10V = v_{R1}$$

$$u_R = R \cdot i_{R1}$$
$$v_{R1} + u_R = v_{C1}$$

$$C \cdot du_C/dt = i_{C1}$$
$$v_{C1} + u_C = v_G$$

$$-i_{S1} + i_{R1} = 0$$

$$-i_{R1} + i_{C1} = 0$$

$$-i_{C1} + i_{S1} + i_G = 0$$

Into State Space Form

We now want to transform this set into a computable state-space form.

- Input Variables: $\mathbf{u} = ()$
- State Variables: $\mathbf{x} = (u_C)$
- State Derivatives: $d\mathbf{x}/dt = (du_C/dt)$
- Output Variables: $\mathbf{y} = ()$

- Since **u** and **y** are empty, the system only consists in the A-matrix.
  (Remember the state-space form of lecture 1)

Non-causal set:

$v_G = 0$

$v_G + 10V = v_{R1}$

$u_R = R \cdot i_{R1}$

$v_{R1} + u_R = v_{C1}$

$C \cdot du_C/dt = i_{C1}$

$v_{C1} + u_C = v_G$

$-i_{S1} + i_{R1} = 0$

$-i_{R1} + i_{C1} = 0$

$-i_{C1} + i_{S1} + i_G = 0$

Forward Causalization

In order to generate computable code, we must causalize the equations.

- Causalizing means that we determine which unknown shall be determined by which equation.
- We start with those equations that have only one unknown.
- These can be causalized immediately
- The state (here: $\mathbf{u_C}$) is assumed to be known.

Non-causal set:

$v_G = 0$ $\leftarrow$

$v_G + 10V = v_{R1}$

$u_R = R \cdot i_{R1}$

$v_{R1} + u_R = v_{C1}$

$C \cdot du_C/dt = i_{C1}$

$v_{C1} + u_C = v_G$

$-i_{S1} + i_{R1} = 0$

$-i_{R1} + i_{C1} = 0$

$-i_{C1} + i_{S1} + i_G = 0$

Forward Causalization

In order to generate computable code, we must causalize the equations.

- Causalizing means that we determine which unknown shall be determined by which equation.
- We start with those equations that have only one unknown.
- These can be causalized immediately
- The state (here: $\mathbf{u_C}$) is assumed to be known.

Non-causal Set:

$v_G + 10V = v_{R1}$

$u_R = R \cdot i_{R1}$

$v_{R1} + u_R = v_{C1}$

$C \cdot du_C/dt = i_{C1}$

$v_{C1} + u_C = v_G$

$-i_{S1} + i_{R1} = 0$

$-i_{R1} + i_{C1} = 0$

$-i_{C1} + i_{S1} + i_G = 0$

Causal List:

$v_G := 0$

Forward Causalization

- Having causalized an equation, more and more variables become known.
- Hence we can continue with the causalization procedure…

Non-causal Set:

$v_G + 10V = v_{R1}$

$u_R = R \cdot i_{R1}$

$v_{R1} + u_R = v_{C1}$

$C \cdot du_C/dt = i_{C1}$

$v_{C1} + u_C = v_G$ $\leftarrow$

$-i_{S1} + i_{R1} = 0$

$-i_{R1} + i_{C1} = 0$

$-i_{C1} + i_{S1} + i_G = 0$

Causal List:

$v_G := 0$

Forward Causalization

- Having causalized and equation, more and more variables become known.
- Hence we can continue with the causalization procedure…

Non-causal Set:

$v_G + 10V = v_{R1}$ $\leftarrow$

$u_R = R \cdot i_{R1}$

$v_{R1} + u_R = v_{C1}$

$C \cdot du_C/dt = i_{C1}$

$-i_{S1} + i_{R1} = 0$

$-i_{R1} + i_{C1} = 0$

$-i_{C1} + i_{S1} + i_G = 0$

Causal List:

$v_G := 0$

$v_{C1} := - u_C + v_G$

Forward Causalization

- Having causalized and equation, more and more variables become known.
- Hence we can continue with the causalization procedure…

Non-causal Set:

$u_R = R \cdot i_{R1}$

$v_{R1} + u_R = v_{C1}$ $\leftarrow$

$C \cdot du_C/dt = i_{C1}$

$-i_{S1} + i_{R1} = 0$

$-i_{R1} + i_{C1} = 0$

$-i_{C1} + i_{S1} + i_G = 0$

Causal List:

$v_G := 0$

$v_{C1} := - u_C + v_G$

$v_{R1} := v_G + 10V$

Forward Causalization

Robotics and Mechatronics Centre

- Having causalized and equation, more and more variables become known.
- Hence we can continue with the causalization procedure…

Non-causal Set:

$u_R = R \cdot i_{R1}$ ←

$C \cdot du_C/dt = i_{C1}$

$-i_{S1} + i_{R1} = 0$

$-i_{R1} + i_{C1} = 0$

$-i_{C1} + i_{S1} + i_G = 0$

Causal List:

$v_G := 0$

$v_{C1} := - u_C + v_G$

$v_{R1} := v_G + 10V$

$u_R := v_{C1} - v_{R1}$

Forward Causalization

Robotics and Mechatronics Centre

- Having causalized and equation, more and more variables become known.
- Hence we can continue with the causalization procedure…

Non-causal Set:

$C \cdot du_C/dt = i_{C1}$

$-i_{S1} + i_{R1} = 0$ $\leftarrow$

$-i_{R1} + i_{C1} = 0$ $\leftarrow$

$-i_{C1} + i_{S1} + i_G = 0$

Causal List:

$v_G := 0$

$v_{C1} := - u_C + v_G$

$v_{R1} := v_G + 10V$

$u_R := v_{C1} - v_{R1}$

$i_{R1} := u_R/R$

Forward Causalization

Robotics and Mechatronics Centre

- Having causalized and equation, more and more variables become known.
- Hence we can continue with the causalization procedure…

Non-causal Set:

$C \cdot du_C/dt = i_{C1}$ ←

$-i_{C1} + i_{S1} + i_G = 0$ ←

Causal List:

$v_G := 0$

$v_{C1} := - u_C + v_G$

$v_{R1} := v_G + 10V$

$u_R := v_{C1} - v_{R1}$

$i_{R1} := u_R/R$

$i_{S1} := i_{R1}$

$i_{C1} := i_{R1}$

Causality Graph

- We can represent the causalized system by means of an acyclic directed graph: The causality graph.
- The vertices of the graph are the equations or assignments
- The edges point out the computational dependence.
- The causality graph gives rise to a partial order of its vertices (equations)

Causal List:

$v_G := 0$

$v_{C1} := - u_C + v_G$

$v_{R1} := v_G + 10V$

$u_R := v_{C1} - v_{R1}$

$i_{R1} := u_R/R$

$i_{S1} := i_{R1}$

$i_{C1} := i_{R1}$

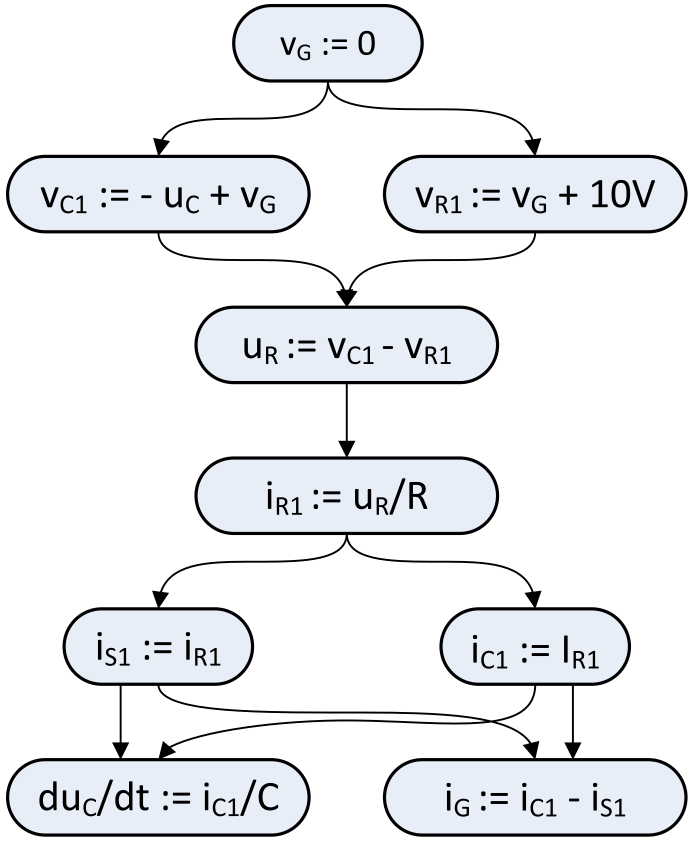
$\underline{du_C/dt := i_{C1}/C}$

$i_G := i_{C1} - i_{S1}$

Causality Graph

Robotics and Mechatronics Centre

- We can represent the causalized system by means of an acyclic directed graph: The causality graph.


$$\begin{tikzcd}
 & v_G := 0 \arrow[dl] \arrow[dr] & \\
v_{C1} := -u_C + v_G \arrow[dr] & & v_{R1} := v_G + 10V \arrow[dl] \\
 & u_R := v_{C1} - v_{R1} \arrow[d] & \\
 & i_{R1} := u_R/R \arrow[dl] \arrow[dr] & \\
i_{S1} := i_{R1} \arrow[d] \arrow[drr] & & i_{C1} := i_{R1} \arrow[d] \arrow[dll] \\
du_C/dt := i_{C1}/C & & i_G := i_{C1} - i_{S1}
\end{tikzcd}$$

Causal List:

$v_G := 0$

$v_{C1} := - u_C + v_G$

$v_{R1} := v_G + 10V$

$u_R := v_{C1} - v_{R1}$

$i_{R1} := u_R/R$

$i_{S1} := i_{R1}$

$i_{C1} := i_{R1}$

$\underline{du_C/dt := i_{C1}/C}$

$i_G := i_{C1} - i_{S1}$

Structure Incidence Matrix

- The non-causal set of equations can be represented by a structural incidence matrix.
- The row vector corresponds to the set of equations
- The column vector represents the set of unknowns.

Non-causal set:

$v_G = 0$

$v_G + 10V = v_{R1}$

$u_R = R \cdot i_{R1}$

$v_{R1} + u_R = v_{C1}$

$C \cdot du_C/dt = i_{C1}$

$v_{C1} + u_C = v_G$

$-i_{S1} + i_{R1} = 0$

$-i_{R1} + i_{C1} = 0$

$-i_{C1} + i_{S1} + i_G = 0$

Structure Incidence Matrix

Robotics and Mechatronics Centre

	i_G	i_{S1}	i_{C1}	i_{R1}	v_G	v_{C1}	v_{R1}	u_R	du_C
1)					X				
2)					X		X		
3)				X				X	
4)						X	X	X	
5)			X						X
6)					X	X			
7)		X		X					
8)			X	X					
9)	X	X	X						

Non-causal set:

1) $v_G = 0$

2) $v_G + 10V = v_{R1}$

3) $u_R = R \cdot i_{R1}$

4) $v_{R1} + u_R = v_{C1}$

5) $C \cdot du_C/dt = i_{C1}$

6) $v_{C1} + u_C = v_G$

7) $-i_{S1} + i_{R1} = 0$

8) $-i_{R1} + i_{C1} = 0$

9) $-i_{C1} + i_{S1} + i_G = 0$

Structure Incidence Matrix

- If we permute the sets of equations and variables given an order that is induced by the causality graph…
- …the structural incidence matrix has a lower triangular form with a full diagonal.

Causal List:

1) $v_G := 0$

6) $v_{C1} := - u_C + v_G$

2) $v_{R1} := v_G + 10V$

4) $u_R := v_{C1} - v_{R1}$

3) $i_{R1} := u_R/R$

7) $i_{S1} := i_{R1}$

8) $i_{C1} := i_{R1}$

5) $du_C/dt := i_{C1}/C$

9) $i_G := i_{C1} - i_{S1}$

Structure Incidence Matrix

Robotics and Mechatronics Centre

	v_G	v_{C1}	v_{R1}	u_R	i_{R1}	i_{S1}	i_{C1}	du_C	i_G
1)	X								
6)	X	X							
2)	X		X						
4)		X	X	X					
3)				X	X				
7)					X	X			
8)					X		X		
5)							X	X	
9)						X	X		X

Causal List:

1) $v_G := 0$

6) $v_{C1} := - u_C + v_G$

2) $v_{R1} := v_G + 10V$

4) $u_R := v_{C1} - v_{R1}$

3) $i_{R1} := u_R/R$

7) $i_{S1} := i_{R1}$

8) $i_{C1} := i_{R1}$

5) $du_C/dt := i_{C1}/C$

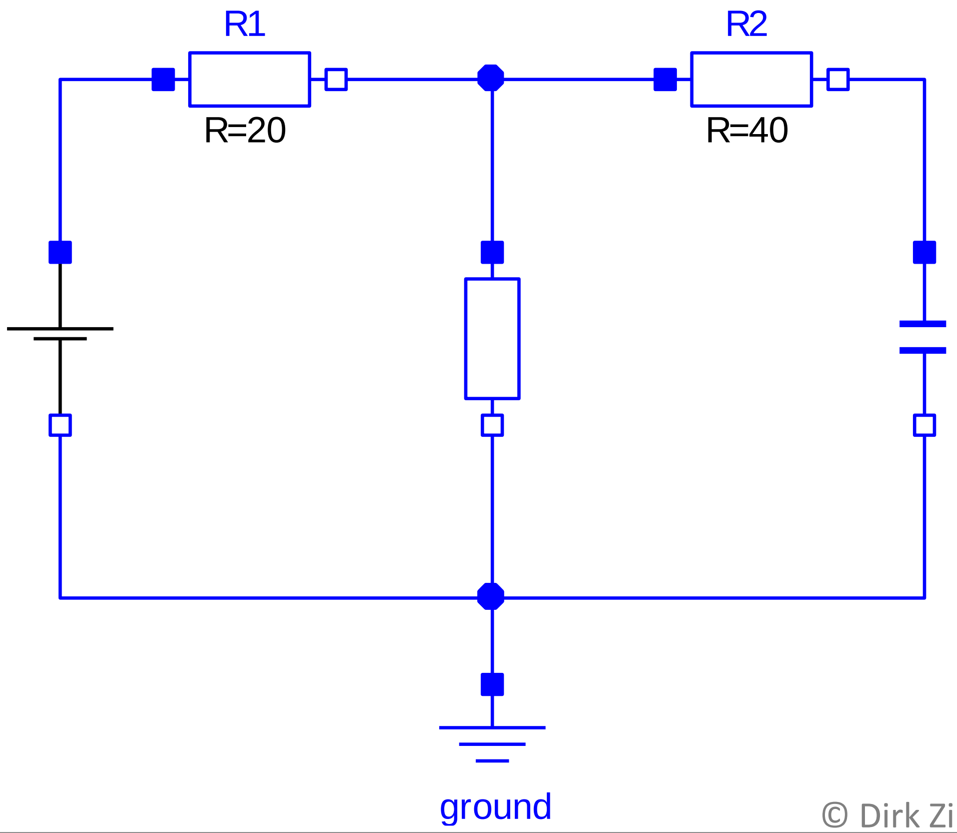
9) $i_G := i_{C1} - i_{S1}$

Lower Triangular Form

- Hence the causalization of the equation system corresponds to a permutation of the structure-incidence matrix into lower-triangular form.
- A system in lower-triangular form can be solved by forward substitution.
- This is equivalent to a sequence of explicit computations and thus represents simple program code.
- Since the causality graph gives only rise to partial order, there might be more than one valid permutation.

Block Lower Triangular Form

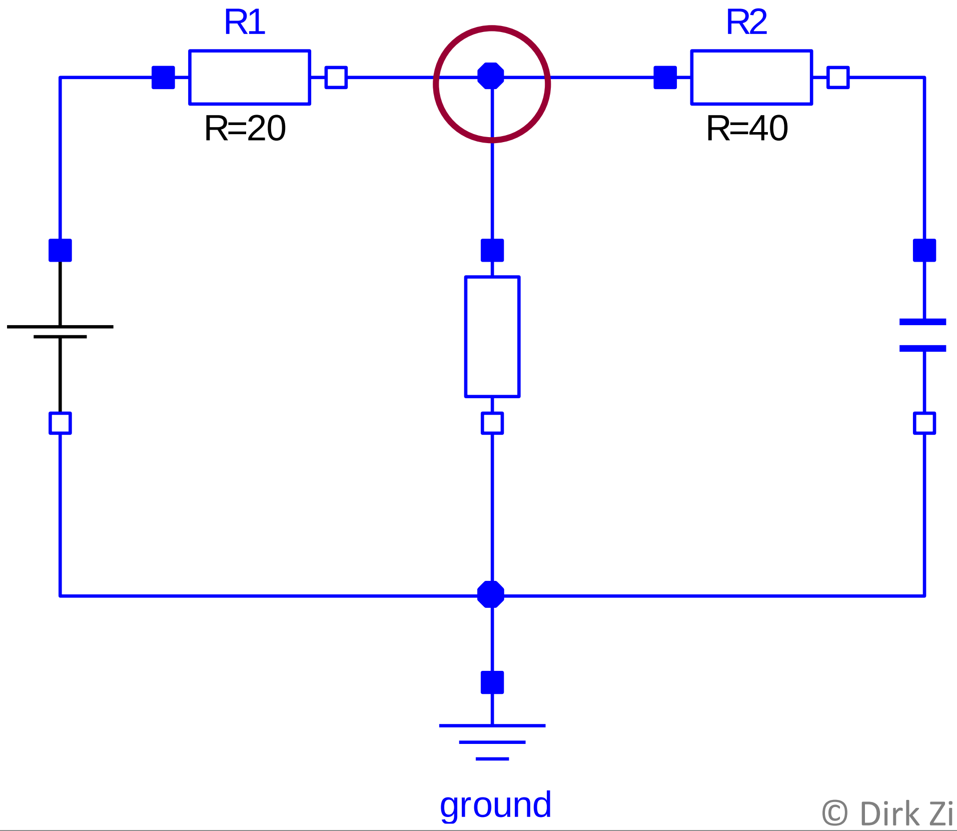
- Unfortunately, not all systems are permutable into lower-triangular form. (For instance: if the structure-incidence matrix is full, any permutation has no effect).



Block Lower Triangular Form

- Unfortunately, not all systems are permutable into lower-triangular form. (For instance: if the structure-incidence matrix is full, any permutation has no effect).

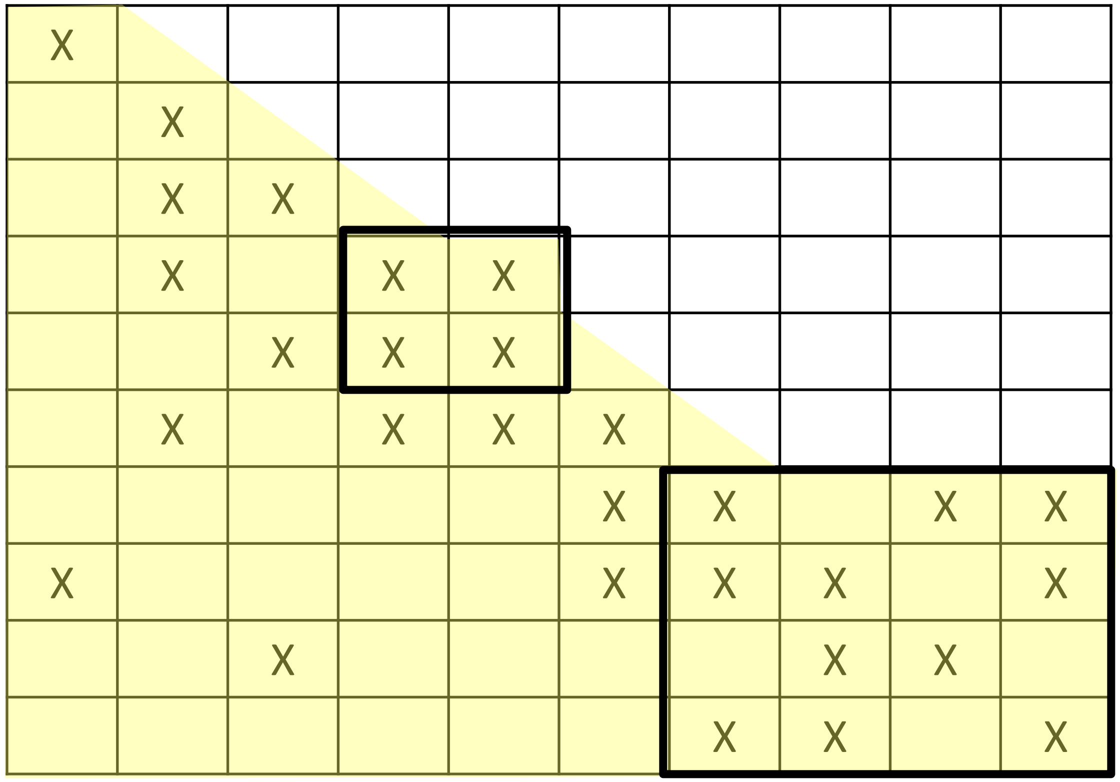
This voltage potential
cannot be explicitly determined,
just by causalizing individual equations



Block Lower Triangular Form

- Unfortunately, not all systems are permutable into lower-triangular form. (For instance: if the structure-incidence matrix is full, any permutation has no effect).
- So forward causalization is not a general procedure. It works only for very simple systems.
- If we cannot attain a lower-triangular form, we aim to be as close as possible to it. This is the block lower triangular (BLT) form.
- A matrix in BLT form is a matrix with blocks on the diagonal where the blocks are as small as possible.
- The blocks of a BLT form can be uniquely determined by the Dulmage-Mendelsohn Permutation.

Block Lower Triangular Form



X									
	X								
	X	X							
	X		X	X					
		X	X	X					
	X		X	X	X				
					X	X		X	X
X					X	X	X		X
		X					X	X	
						X	X		X

Dulmage-Mendelsohn Permutation

Robotics and Mechatronics Centre

In order to compute the block lower triangular form, we apply the Dulmage-Mendelsohn Permutation.

- This algorithm is combines a maximum matching on bipartite graphs with Tarjan`s strong component analysis.

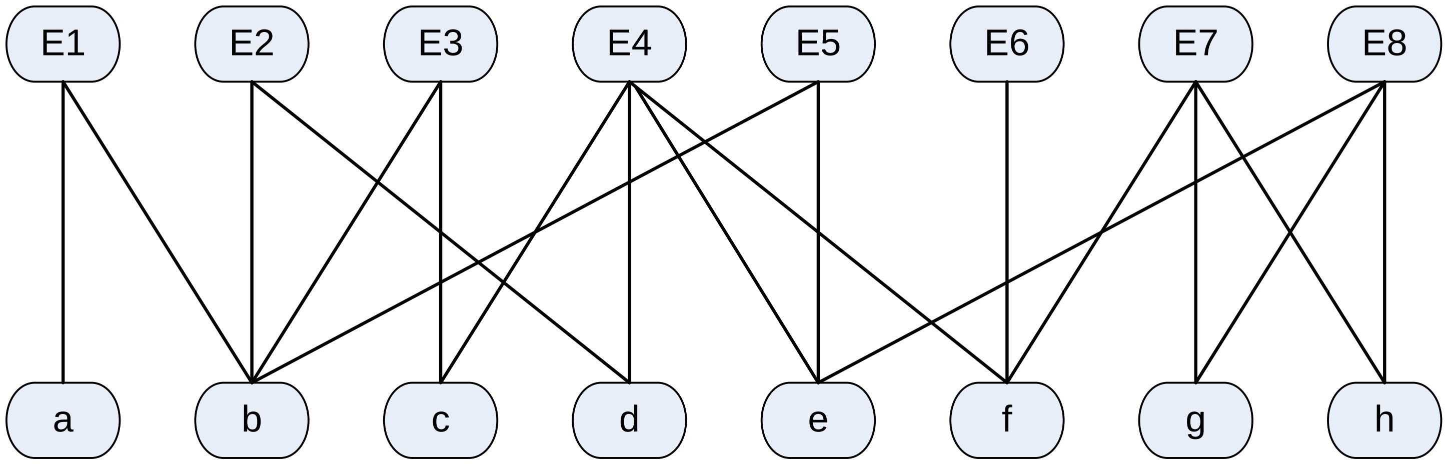
	a	b	c	d	e	f	g	h
E1	X	X						
E2		X		X				
E3		X	X					
E4			X	X	X	X		
E5		X			X			
E6						X		
E7						X	X	X
E8					X		X	X

$\Rightarrow$ **BLT?**

Bipartite Graphs

First let us look at the equation system in form of a bipartite graph.

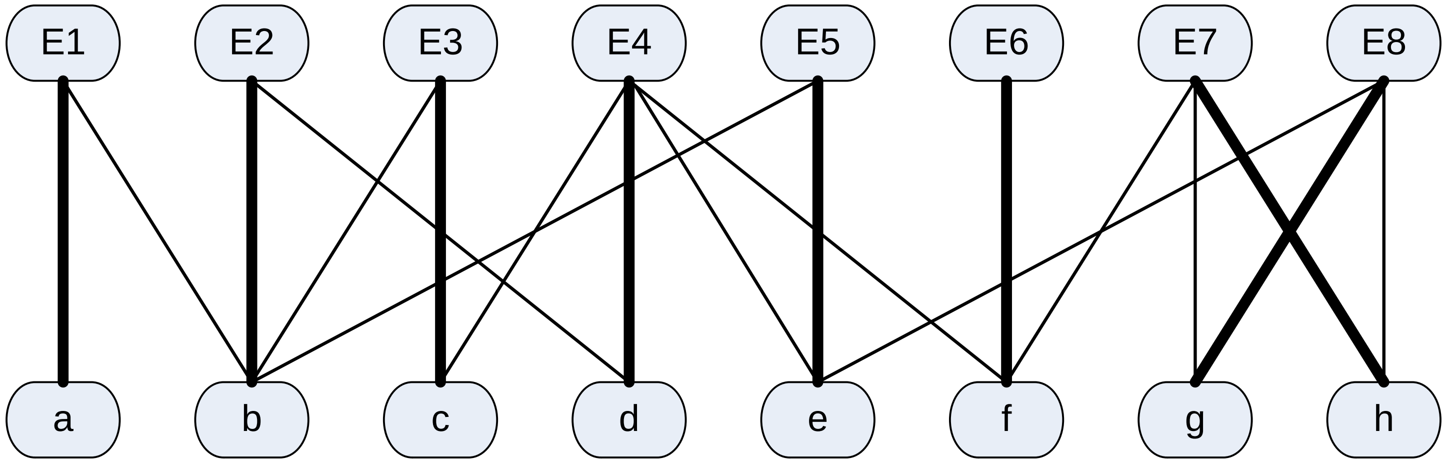
- The first set of vertices is represented by the equations (E1-E8).
- The second set of vertices is represented by the variables (a-h).
- The occurrence of a variable A in an equation B represents an edge in the graph.



Perfect Matching

In order to causalize the system, we need to assign an unknown to each equation.

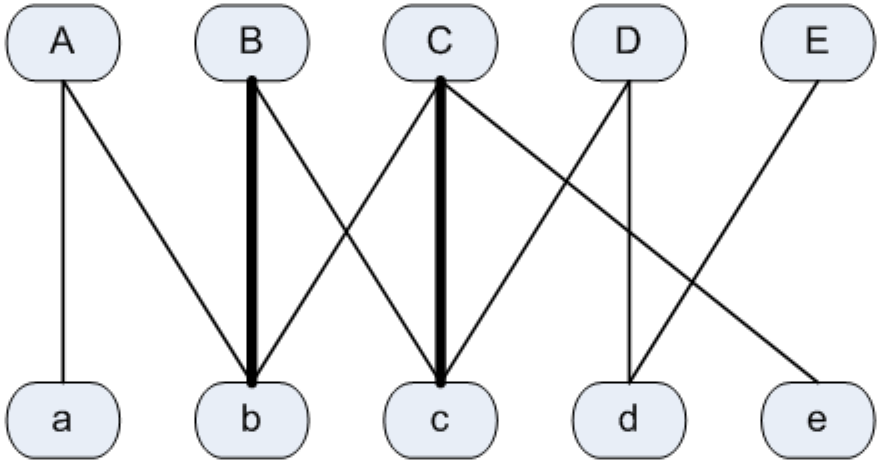
- Such an assignments equals a perfect matching.
- A structural regular system contains at least one perfect matching.



Bipartite Graphs

Given the graph G(V,E): (V is the set of vertices, E is the set of Edges)

- A **matching** M in G consists in a set of edges ($M \subseteq E$) so that no vertex in G is connected to two edges in M.
- A **maximal matching** is a matching M so that no edge in M\E can be added.
- A **maximum matching** M is matching M so that there is no other matching N for G with |N| > |M|
- A **perfect matching** is a matching with no unmatched vertex in G.



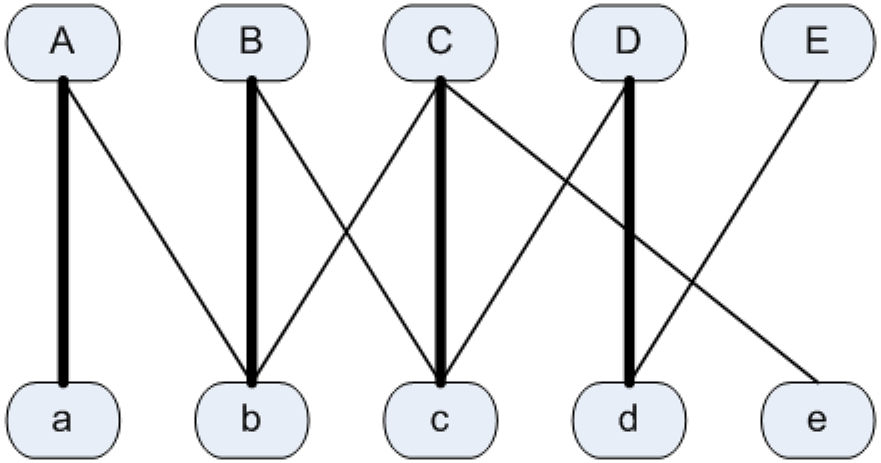
← matching

Bipartite Graphs

Robotics and Mechatronics Centre

Given the graph G(V,E):

- A **matching** M in G consists in a set of edges ($M \subseteq E$) so that no vertex in G is connected to two edges in M.
- A **maximal matching** is a matching M so that no edge in M\E can be added.
- A **maximum matching** M is matching M so that there is no other matching N for G with |N| > |M|
- A **perfect matching** is a matching with no unmatched vertex in G.



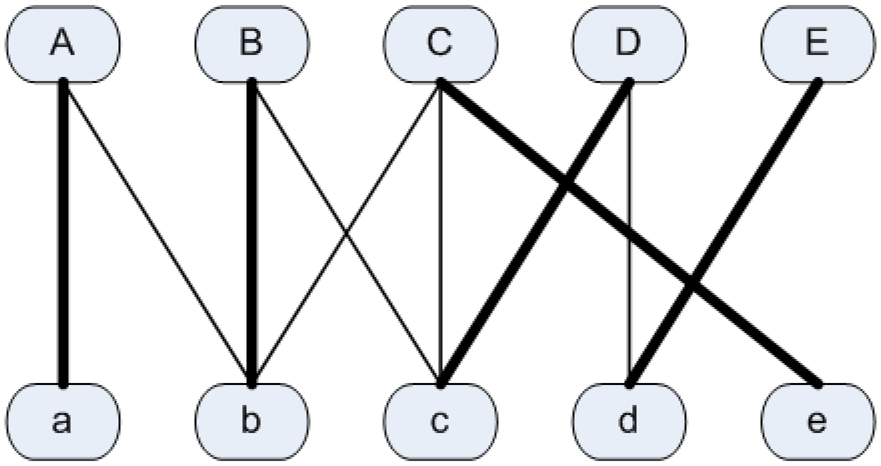
← **maximal matching**
(can be obtained by
a greedy algorithm)

Bipartite Graphs

Robotics and Mechatronics Centre

Given the graph G(V,E):

- A **matching** M in G consists in a set of edges ($M \subseteq E$) so that no vertex in G is connected to two edges in M.
- A **maximal matching** is a matching M so that no edge in M\E can be added.
- A **maximum matching** M is matching M so that there is no other matching N for G with |N| > |M|
- A **perfect matching** is a matching with no unmatched vertex in G.

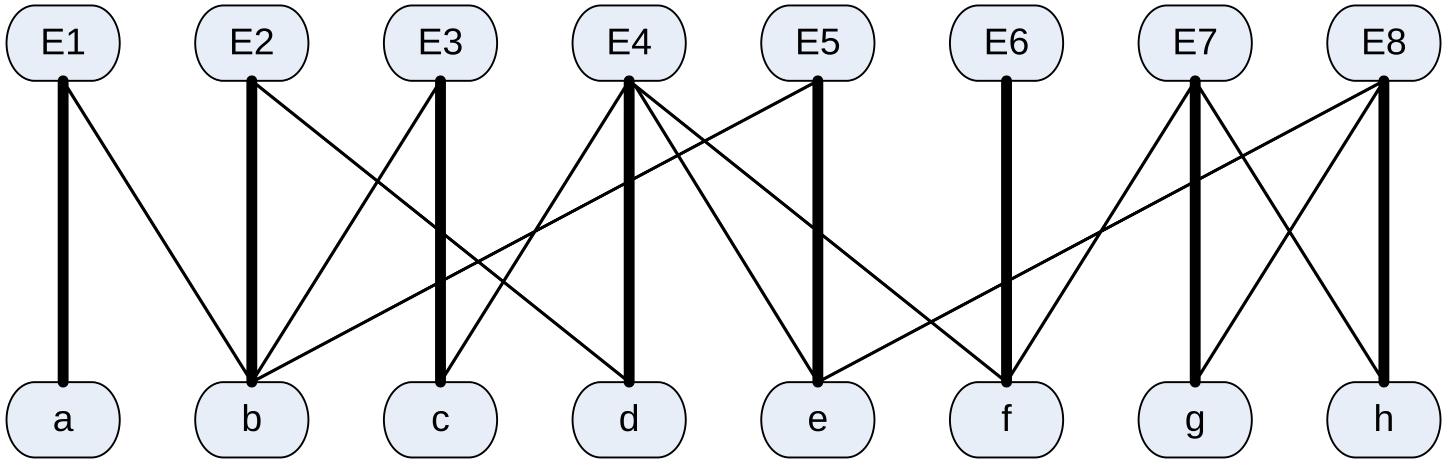


←maximum matching and perfect matching

Perfect Matching

In order to causalize the system, we need to assign an unknown to each equation.

- Such an assignments equals a perfect matching.
- A structural regular system contains at least one perfect matching.

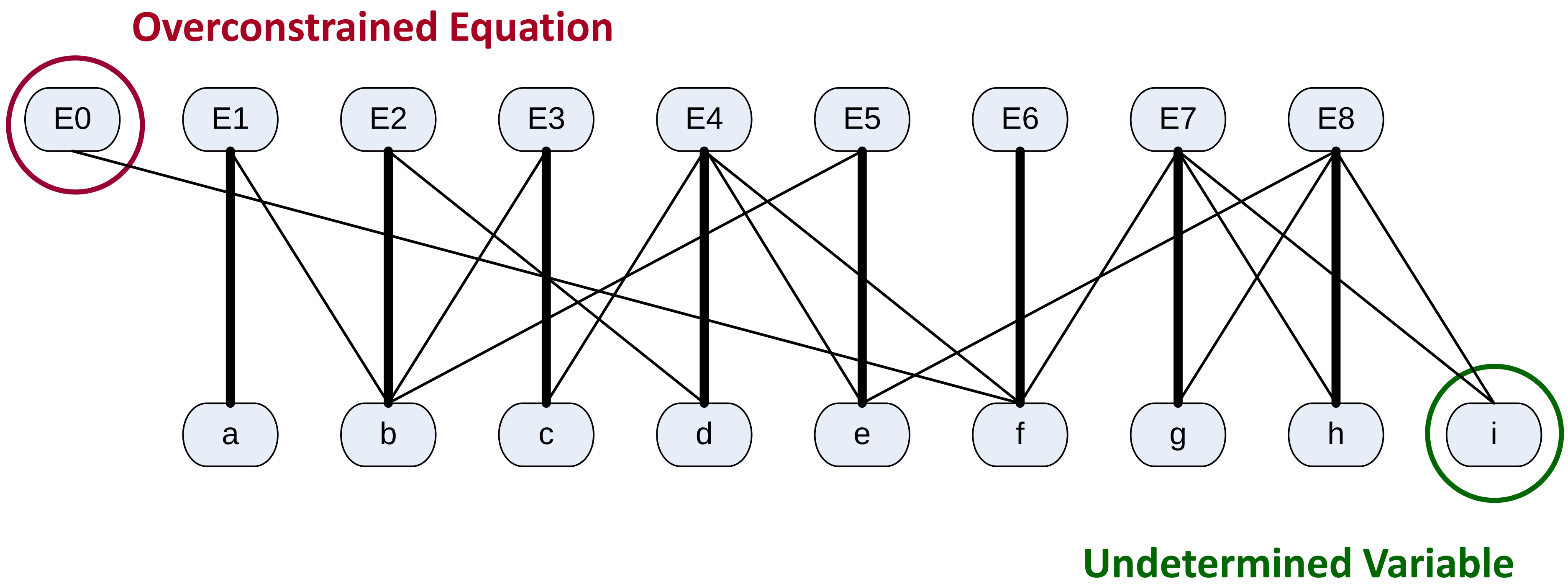


Maximum Matching

Robotics and Mechatronics Centre

Any perfect matching represents a maximum matching.

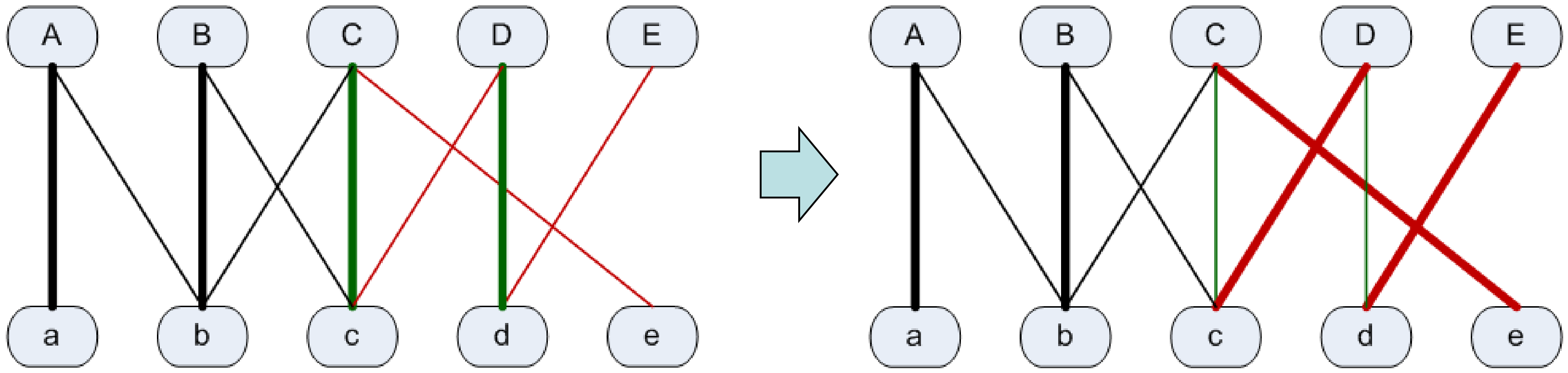
- For structurally singular systems, the maximum matching reveals the overconstrained equations and the undetermined variables.



Augmenting Paths

Getting a maximal matching is easy but how can we obtain a maximum (and hopefully perfect) matching?

- A path in G is called **alternating path** if its edges alternate between the sets E\M and M (starting and ending arbitrarily).
- An **augmenting path** is an alternating path whose end-vertices are unmatched.
- Finding an augmenting path naturally leads to a matching that is larger by one: we can flip the matched edges with the unmatched edges.



Augmenting Paths

This observation leads to the **Augmenting Path Algorithm:**

1. Look for an augmenting path by a depth-first traversal of the edges.
2. Increase the matching by one.
3. Repeat until you cannot find any augmenting path anymore.

- Finding an augmenting path takes at maximum O(|E|) time.
- At maximum |V|/2 augmenting paths can be found.
- Hence the overall complexity is O(|V||E|)

Symmetric difference

It is evident that finding an augmenting path leads to a larger matching but how can we be sure that finding none indicates a maximum matching?

- Let N be a maximum matching. Now let us look at the **symmetric difference** D between M and N: $D = (M \cup N) \setminus (M \cap N)$
- D has a special structure. Each vertex in D can be at most connected to two edges (one in M and one N). Hence each component C of D is either:
  - an isolated vertex,
  - a circle of even length, or
  - an alternating path (cycle free).
- If $|N| > |M|$ then at least one component C must have an uneven number of edges. The only option is an augmenting path.
- We can make the stronger statement: For $k = |N| - |M|$ the symmetric difference D contains k disjoint augmenting paths.

Algorithm of Hopcroft and Karp

This statement suggest an improvement of the augmenting path algorithm. Since the augmenting paths are disjoint, there cannot be many long augmenting paths.

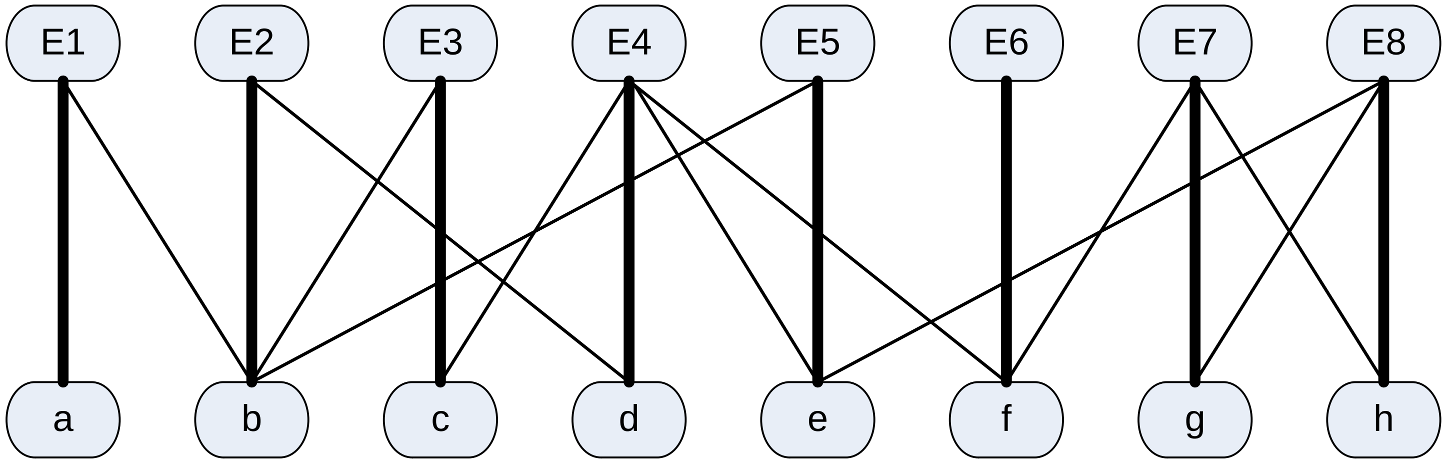
1. Perform a breath-first search on G to find the set of disjoint augmenting paths of minimal length.
2. Increase the matching for each augmenting path found
3. Repeat until no augmenting path can be found.

- Each breadth-first search takes O(|E|) time.
- After r=sqrt(|V|) steps, the minimum length of an augmenting path is r. There cannot be more than |V|/r = r augmenting paths left. Hence the algorithm computes in O(r=sqrt(|V|) |E|)
- This variant is called Hopcroft's Algorithm. (or Alg. of Hopcroft and Karp)

Construction a Digraph

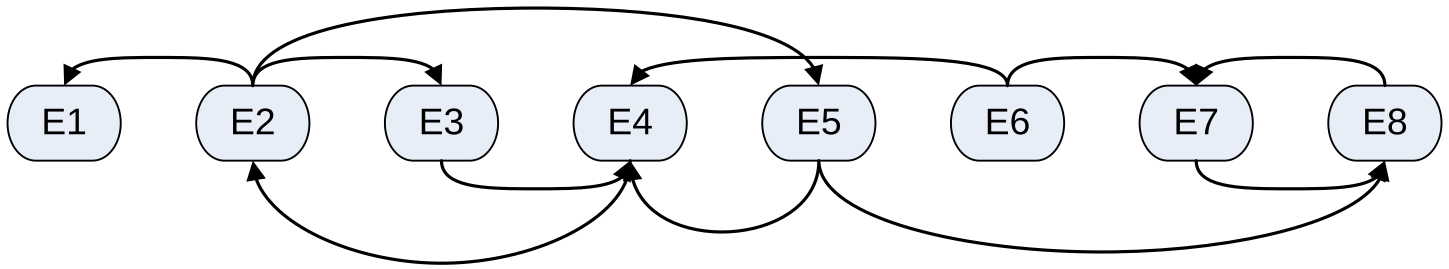
Robotics and Mechatronics Centre

- The perfect matching assigns an unknown to each equation.
- This can be interpreted as a preliminary causalization of the equations.
- Hence, we can construct a causality graph.



Construction a Digraph

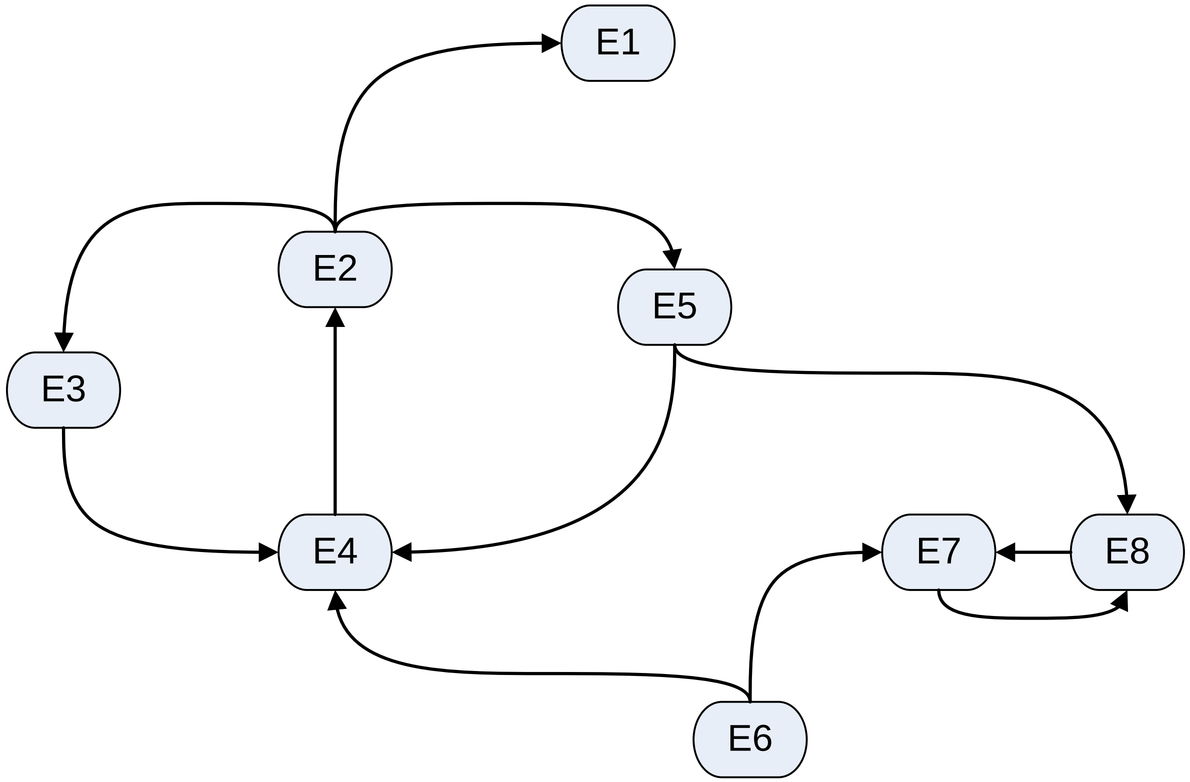
- The perfect matching assigns an unknown to each equation.
- This can be interpreted as a preliminary causalization of the equations.
- Hence, we can again construct a causality graph.



Construction a Digraph

Robotics and Mechatronics Centre

- The perfect matching assigns an unknown to each equation.
- This can be interpreted as a preliminary causalization of the equations.
- Hence, we can again construct a causality graph.



Strong Components

- But, now the causality graph is not acyclic anymore.

- The graph contains strong components

A strong component H in a directed graph G
is a vertex-induced sub-graph of G
so that there is a path from each vertex
to any other vertex in H.

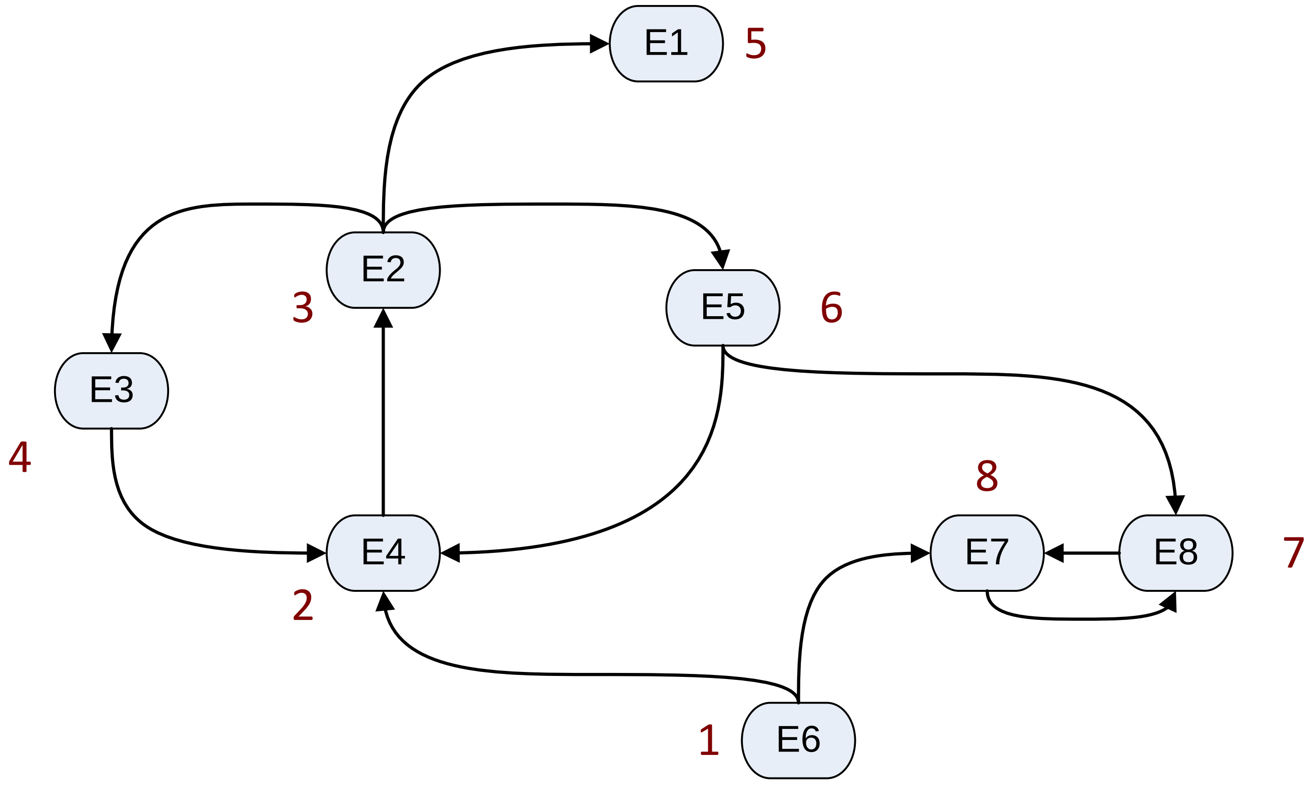
- The strong components in the digraph represent the blocks in the BLT-form of the structure-incidence matrix.

Tarjan`s Algorithm

- Tarjan’s strong component analysis is a depth-first traversal of the di-graph with marking and backtracking.
- Idea:
  - We traverse all vertices in depth-first manner.
  - Each vertex is marked by an index i that represents the traversal number.
  - A vertex that has been marked is not traversed again. So each vertex is only traversed only once.
  - While backtracking, each vertex is assigned j being the minimum index i or j of its direct neighbors or i of itself. Direct neighbors are only included in the set if they are still on the traversal stack or if their index j is pointing to the traversal stack.
  - Finally, each strong component has one root vertex where i=j.
  - For those vertices where i≠j, j points (directly or indirectly) to the root vertex of the strong component.

Tarjan`s Algorithm: Example

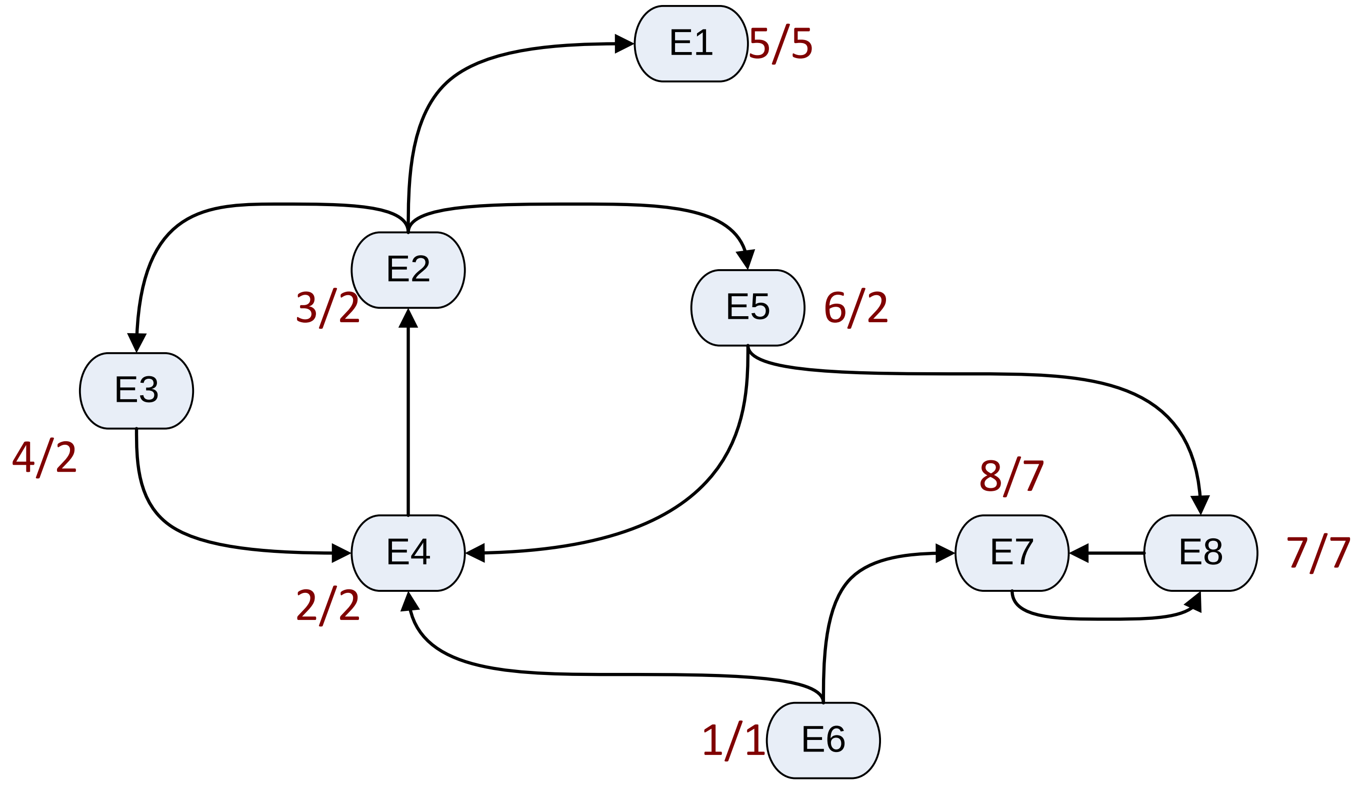
- Here, we enumerate one possible depth first traversal.



Tarjan`s Algorithm: Example

Robotics and Mechatronics Centre

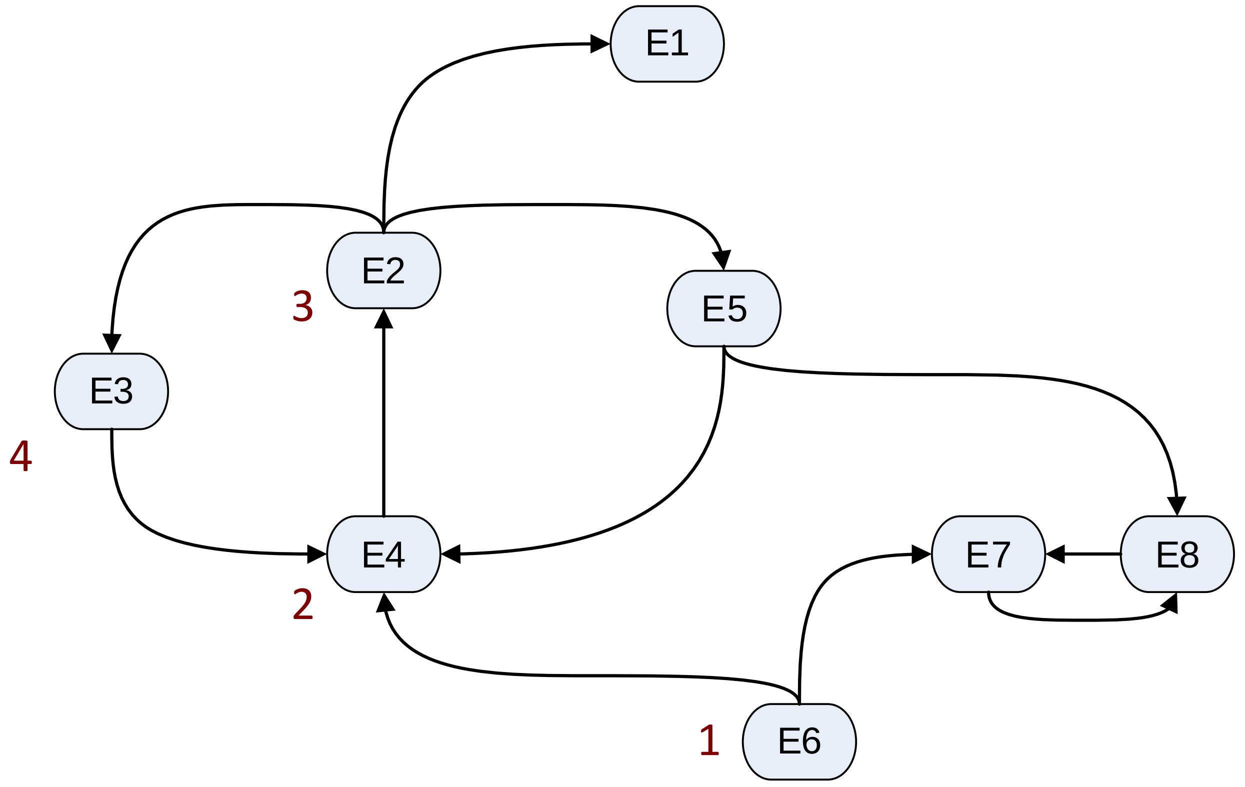
- Here, we enumerate the depth first traversal.



- While Backtracking, we reassign the minimum traversal number by the one of the vertex itself or its direct neighbors.

Tarjan`s Algorithm: Example

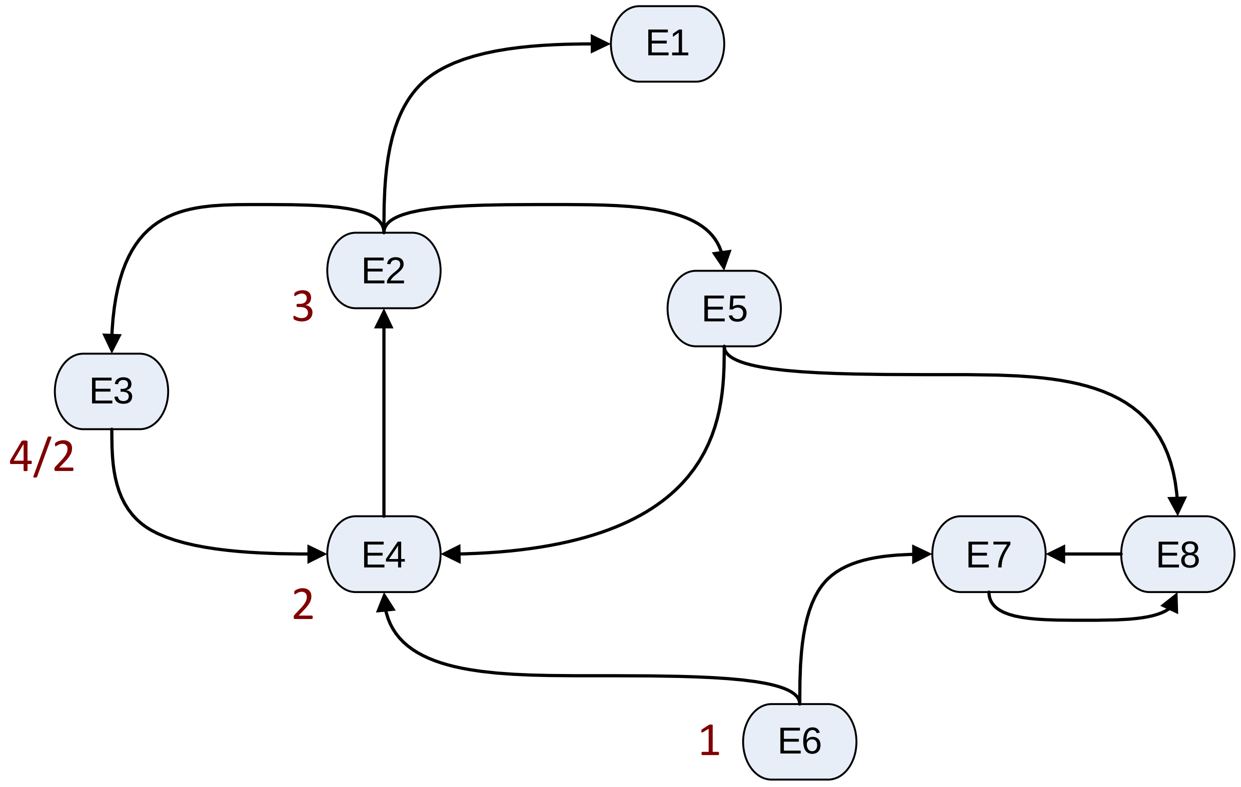
- Let's do it step by step. Depth-First Traversal



Tarjan`s Algorithm: Example

Robotics and Mechatronics Centre

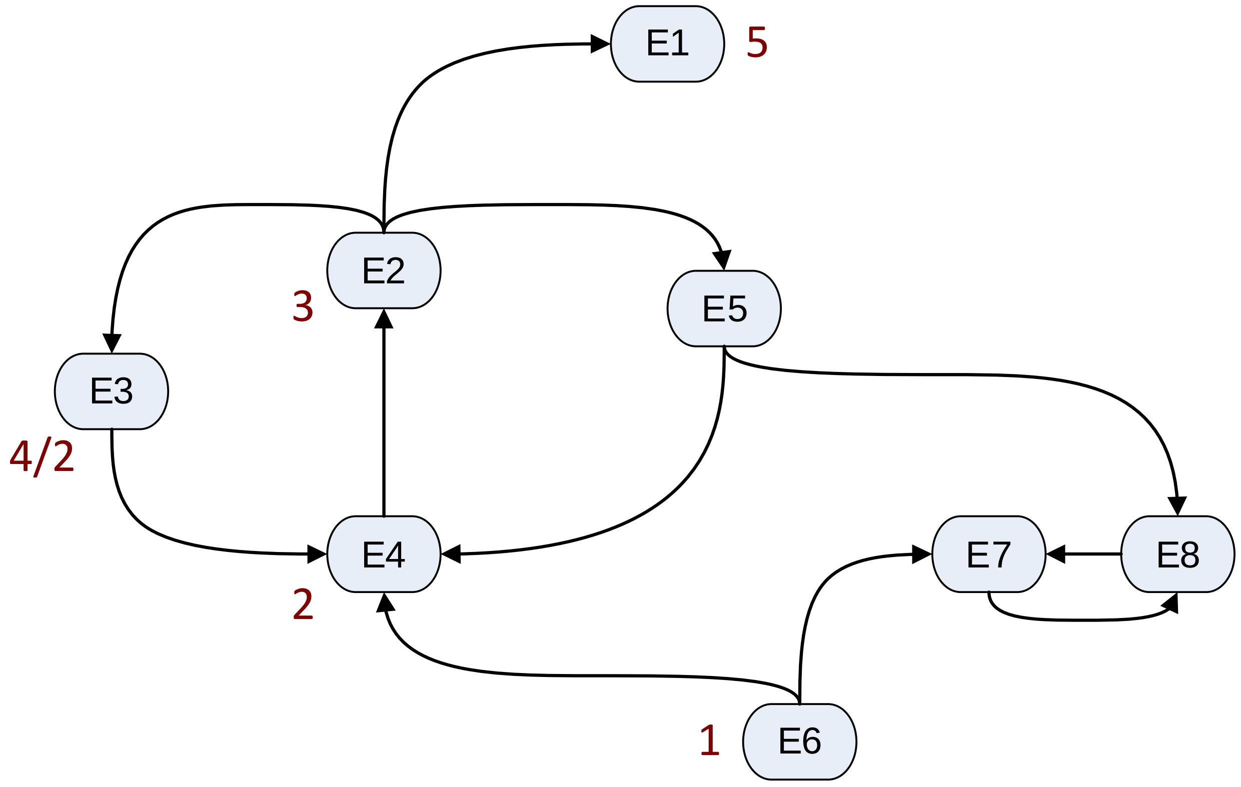
- Let’s do it step by step: Back-Tracking



Tarjan`s Algorithm: Example

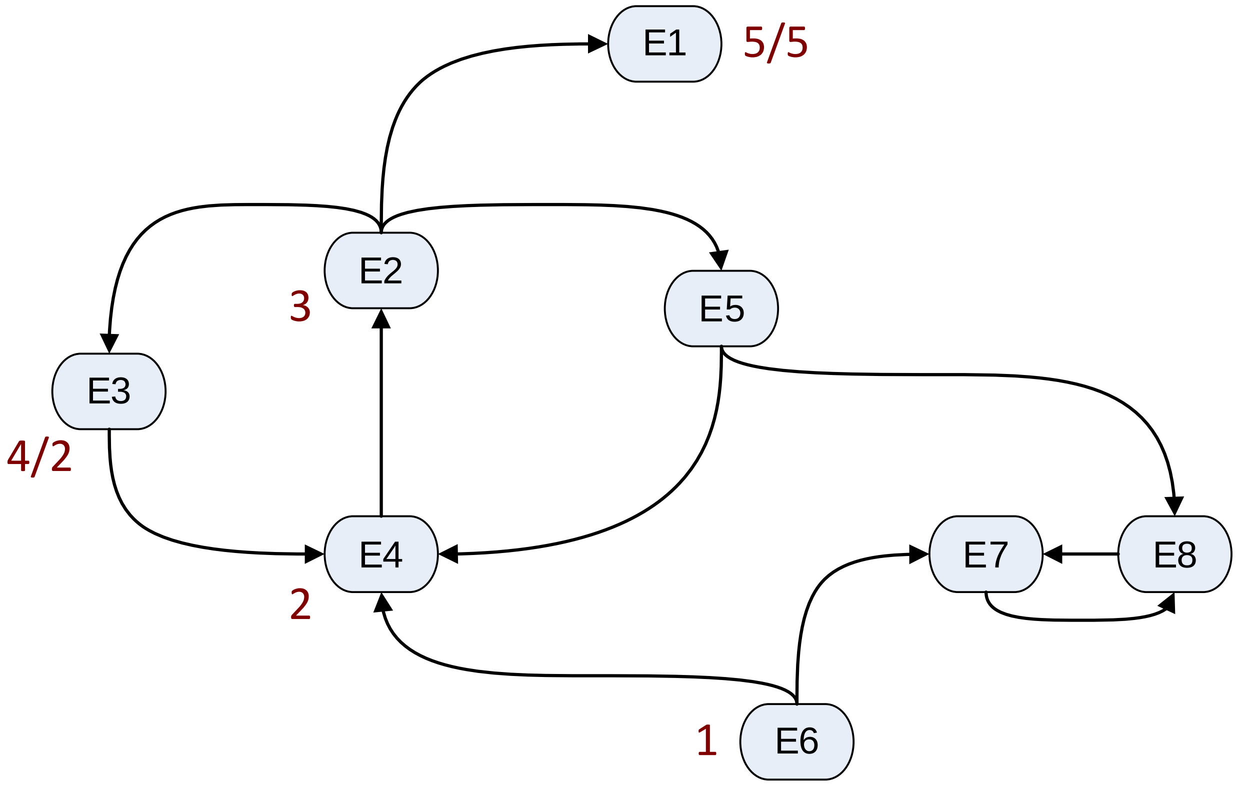
Robotics and Mechatronics Centre

- Let's do it step by step: Depth-First Traversal



Tarjan`s Algorithm: Example

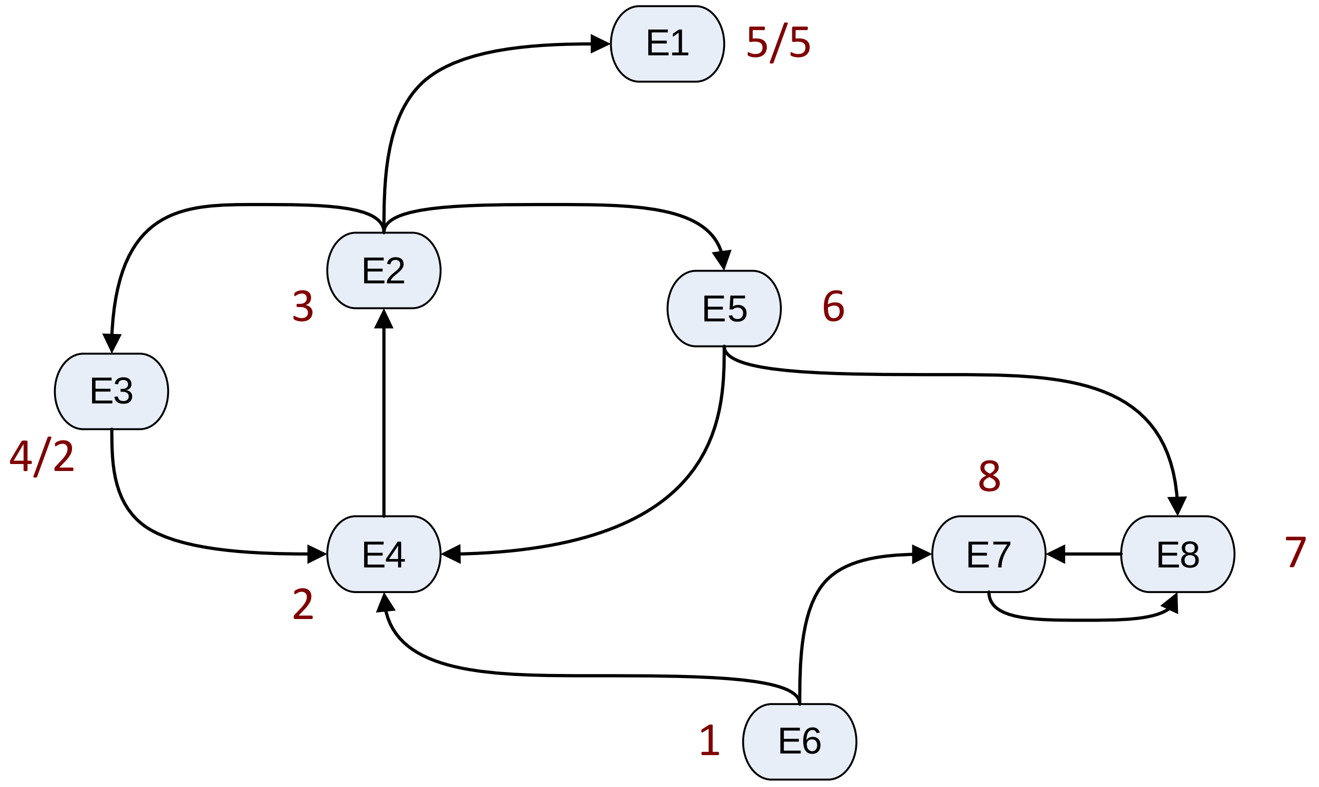
- Let’s do it step by step: Back-Tracking



Tarjan`s Algorithm: Example

Robotics and Mechatronics Centre

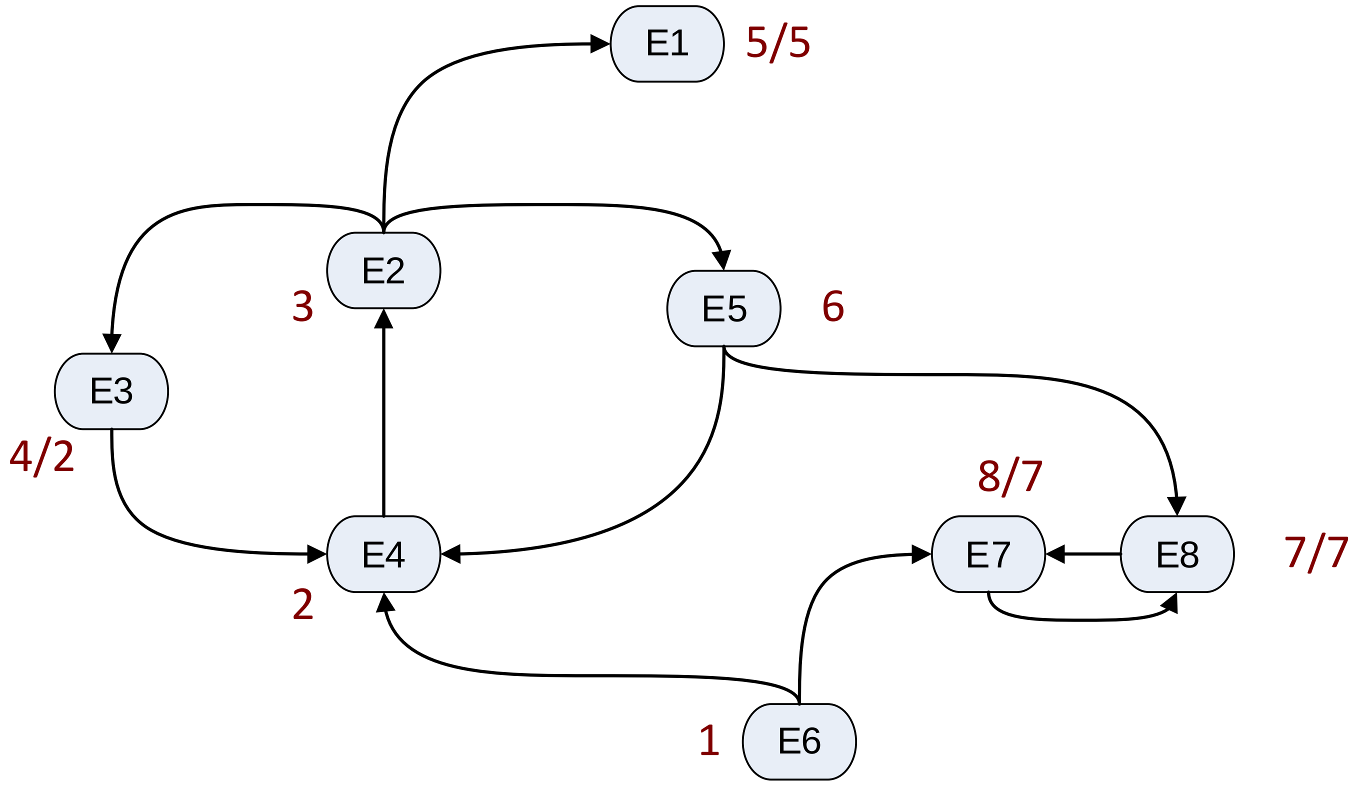
- Let's do it step by step: Depth-First Traversal



Tarjan`s Algorithm: Example

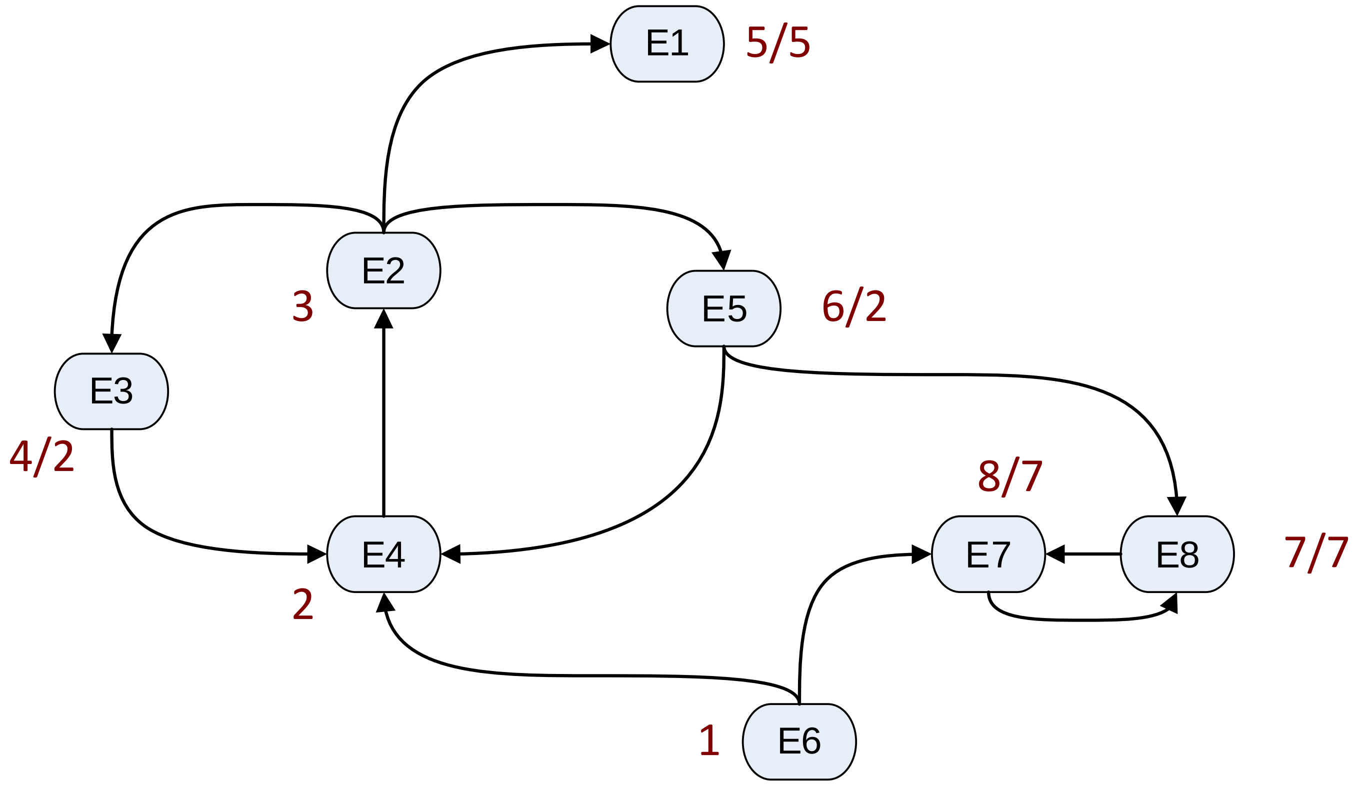
Robotics and Mechatronics Centre

- Let's do it step by step: Back-Tracking



Tarjan`s Algorithm: Example

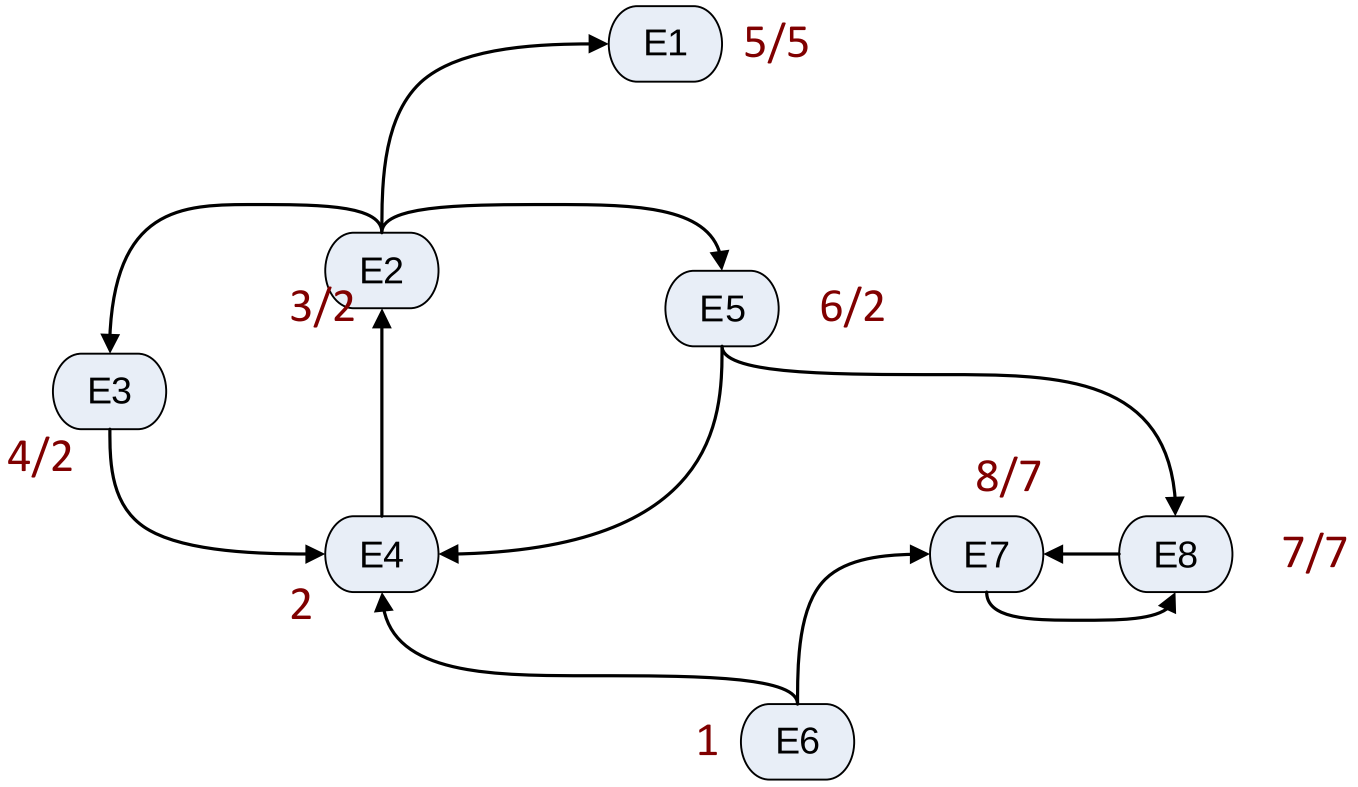
- Let’s do it step by step: Back-Tracking



Tarjan`s Algorithm: Example

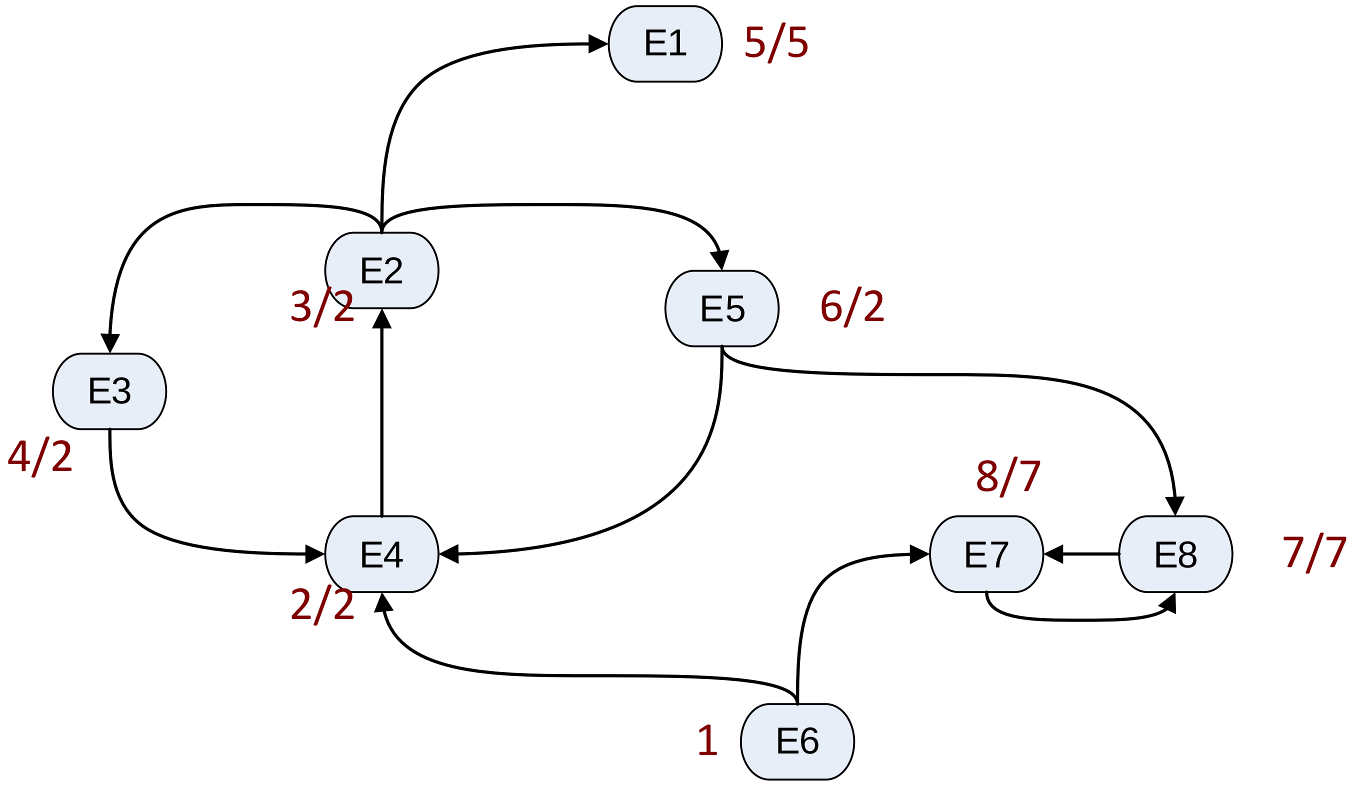
Robotics and Mechatronics Centre

- Let’s do it step by step: Back-Tracking



Tarjan`s Algorithm: Example

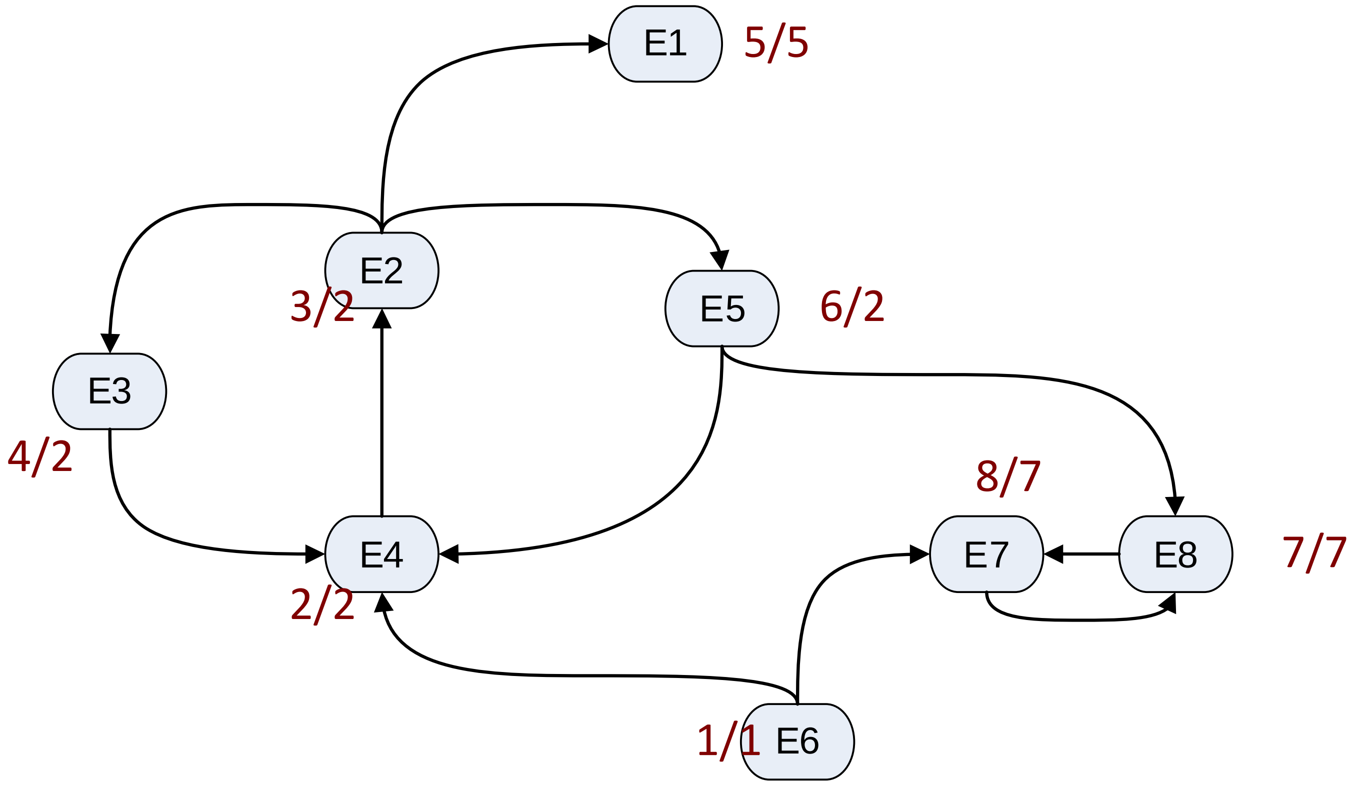
- Let’s do it step by step: Back-Tracking



Tarjan`s Algorithm: Example

Robotics and Mechatronics Centre

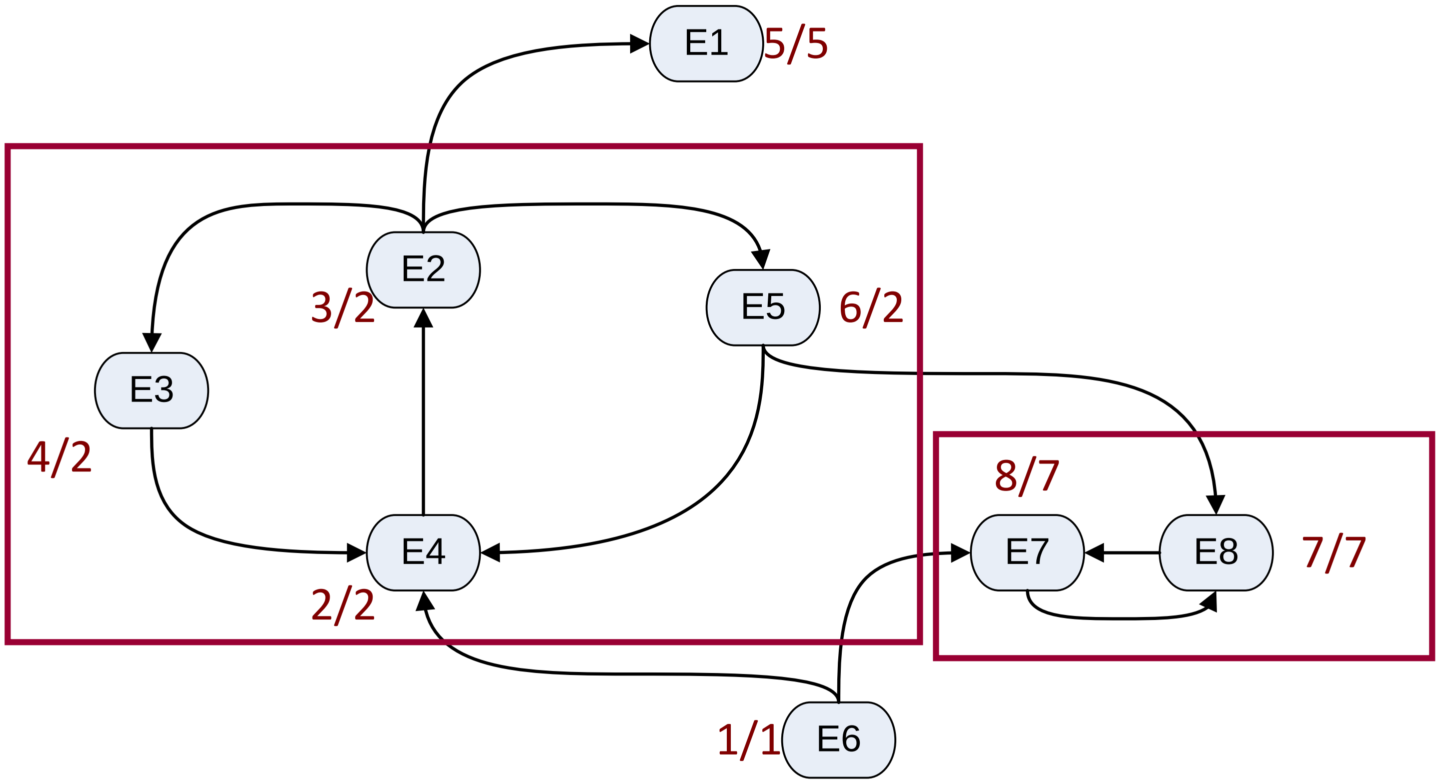
- Let's do it step by step: Back-Tracking



Tarjan`s Algorithm: Example

Robotics and Mechatronics Centre

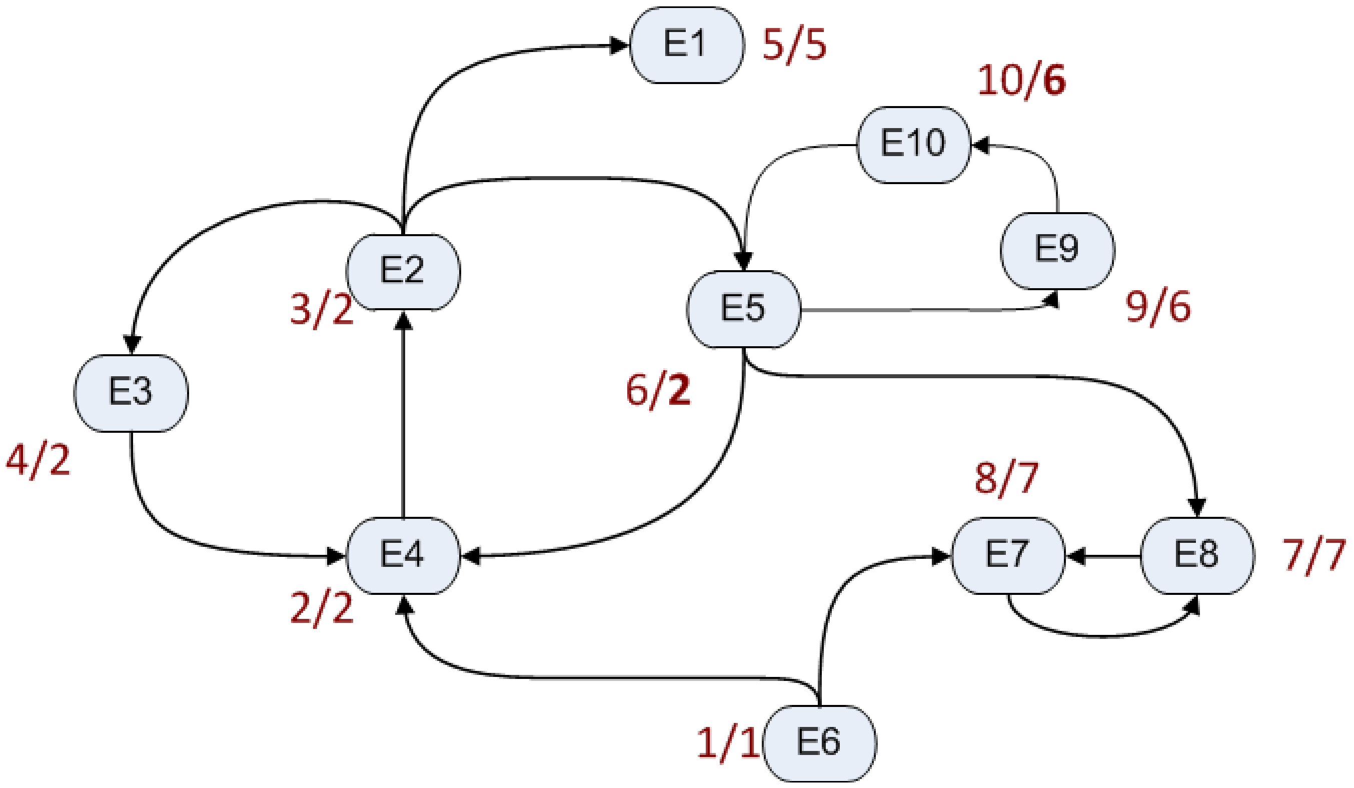
- Here, we enumerate one possible depth first traversal.



- While backtracking, we reassign the minimum traversal number by the one of the vertex itself or of those direct neighbors located on the traversal path.
- We find four strong components (where two are real components).

Tarjan`s Algorithm: Watch it! (1)

- Here, a slight extension to the graph:

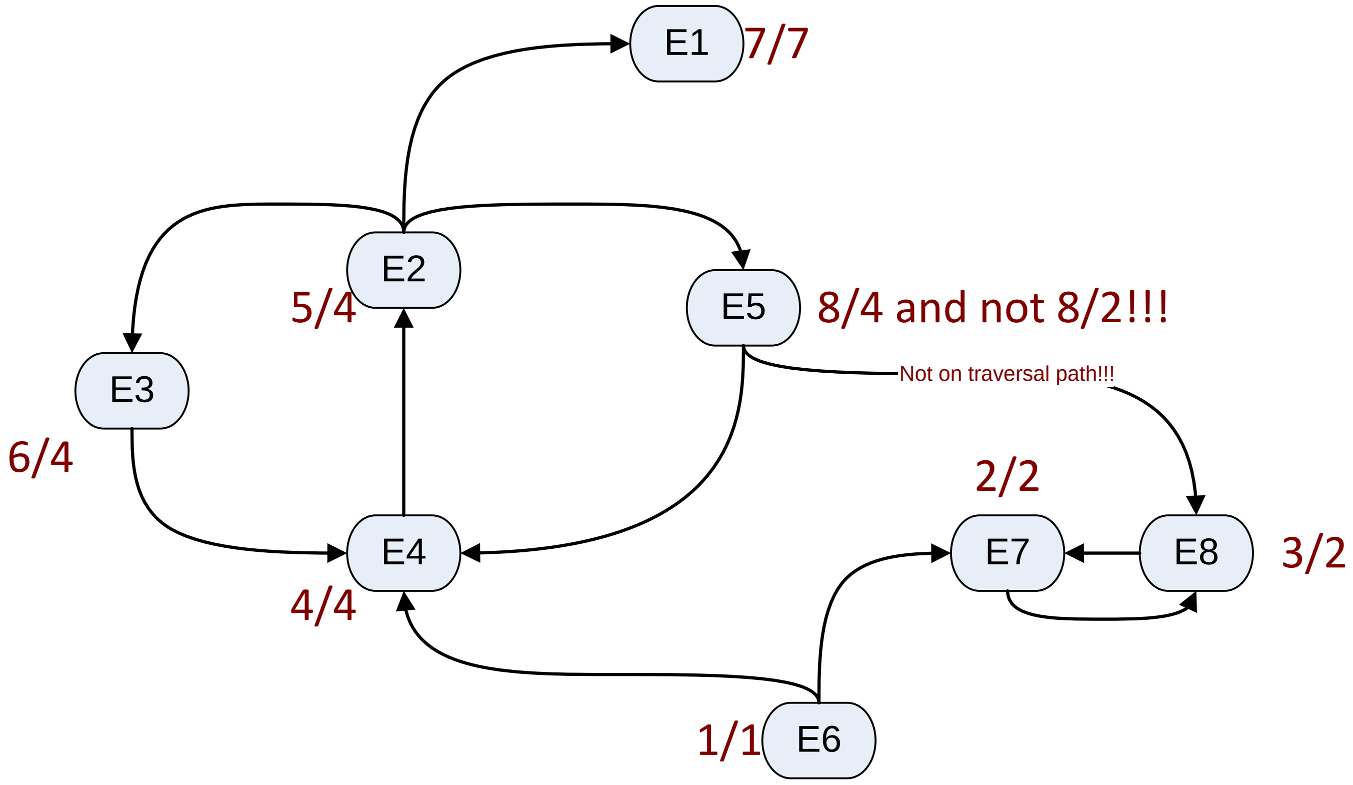


- E2, E3, E4, E5, E9 and E10 belong to one strong component.
- However, E10 and E9 point only indirectly to the root. E6 serves as a relay.

Tarjan`s Algorithm: Watch it! (2)

Robotics and Mechatronics Centre

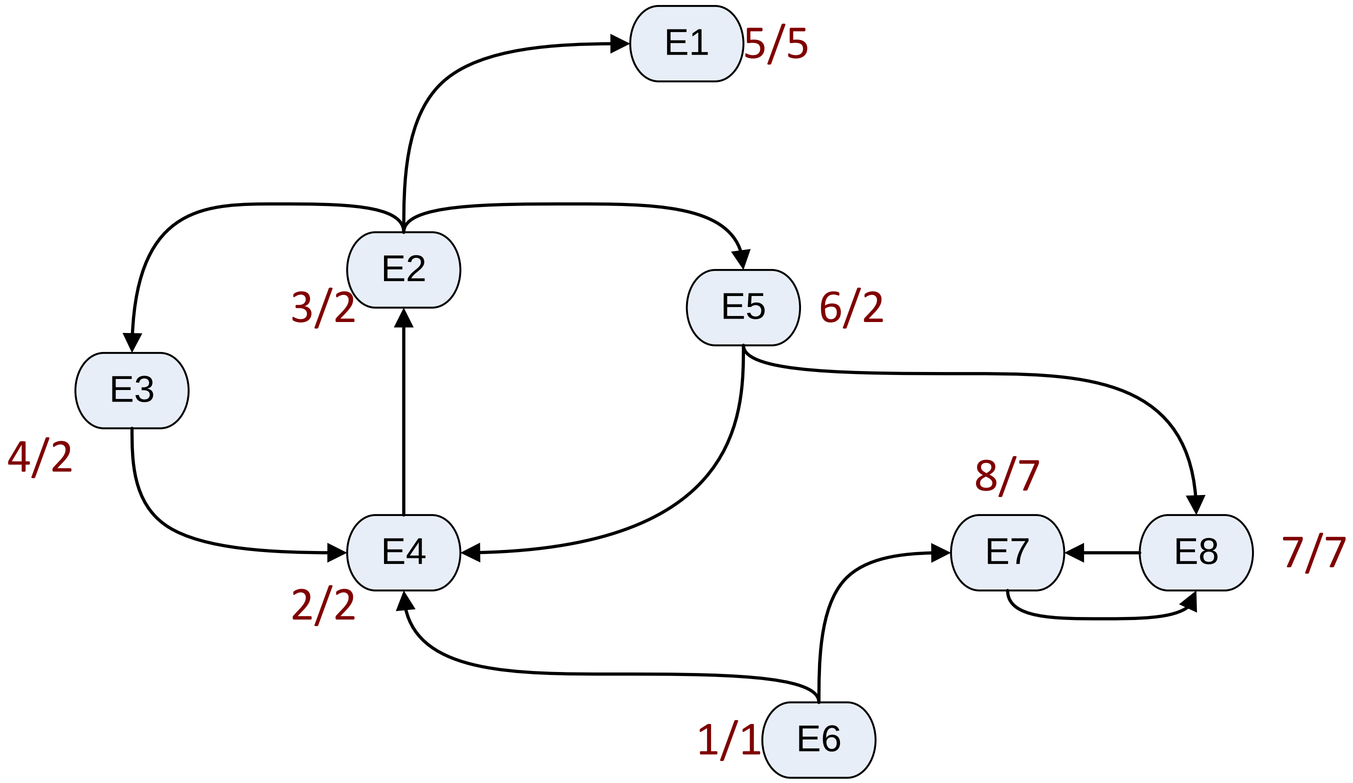
- It is important that only neighbors on the traversal path are considered…



- …because otherwise the components would be merged if we choose a different order of the depth-first traversal.

Tarjan`s Algorithm: Complexity

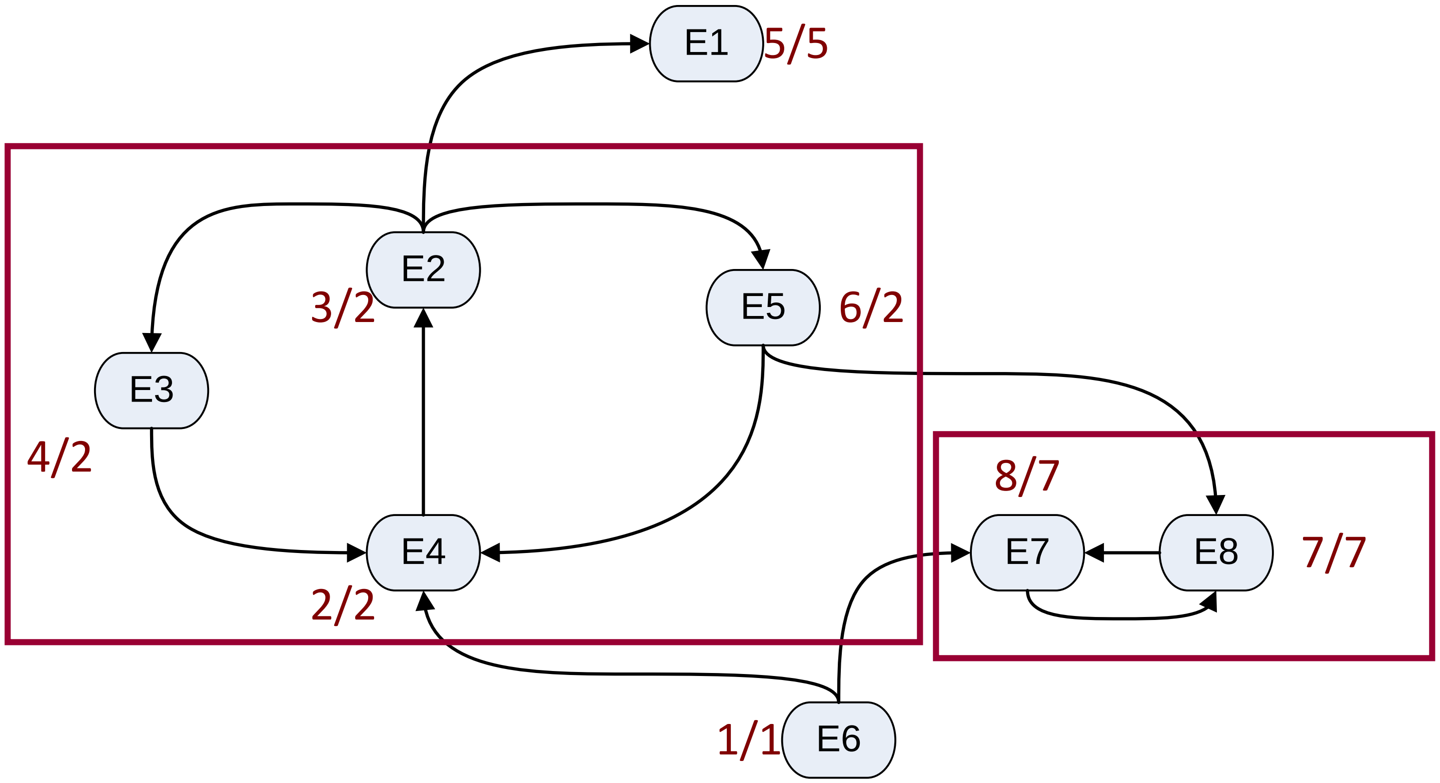
- For the traversal, we traverse each vertex and each edge exactly once.



- So the algorithmic complexity is O(V+E) or O(V^2)
- The maximum memory overhead is in O(V)

Tarjan`s Algorithm: Partial Order

- If we represent the strong components as a single vertex…



- … the causality graph is again acyclic and gives rise to a partial order.
- (E6) -> (E4,E2,E3,E5) -> (E1) -> (E7,E8)
- This order can be used to create the BLT-form.

Determining the BLT-Form

Robotics and Mechatronics Centre

- If we represent the strong components as a single vertex…

	e	d	b	c	f	a	g	h
E6	X							
E4	X	X		X	X			
E2		X	X					
E3			X	X				
E5			X		X			
E1			X			X		
E7							X	X
E8					X		X	X

- … the causality graph is again acyclic and gives rise to a partial order.
- (E6) -> (E4,E2,E3,E5) -> (E1) -> (E7,E8)
- This order can be used to create the BLT-form.

Tearing

Robotics and Mechatronics Centre

- We have managed so far to isolate the blocks…
- …but we still do not know how to generate code for the blocks.

- Idea: For each block we assume a set of its variables to be known. These are called *tearing variables*.
- Given this presumption, we can causalize all equations in the block.
- Some equations are overconstrained. These are turned into residual equations.

- So we do not generate code for the direct solution, but for an iterative numerical solver!

Tearing

- Let us look at an example:
    - E7: $g+h = 1$ (g,h are unknowns)
    - E8: $g*h = f$ (f is known)

- We assume g to be known. g is a *tearing variable*. Now, we can causalize:
    - E7: $h := 1-g$
    - E8: $\mathit{residual} := f - g*h$

- For every value of g, we get a residual value in return. If g is the correct solution the residual will be zero.

- The causalized block, now represents a function *residual* = f(g).
  We can use this code in order to solve the system iteratively.

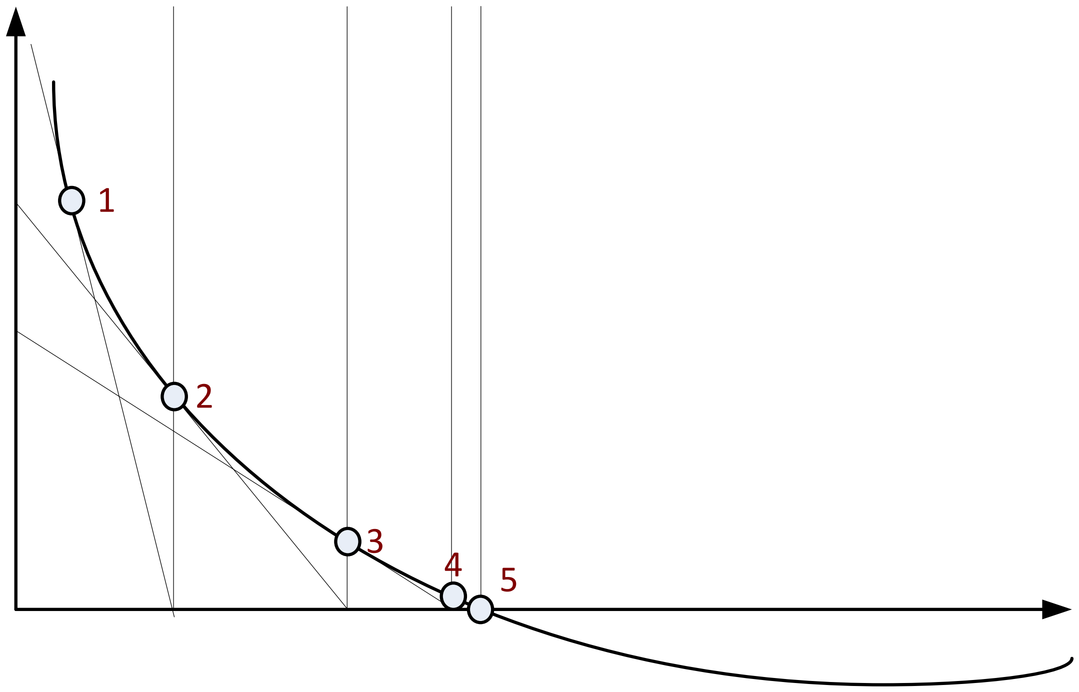
Newton’s Method

- In order to solve 0 = f(x), we can apply standard root-solving methods.
- The most prominent is Newton’s Method.
- Newton’s Method is an iterative algorithm and requires an initial guess x := x0

```
do
   y := f(x)
   x_old := x;
   x := x - y/(df(x)/dx);
while 2*|x_old - x| / (|x_old| + |x|) > tol
```

Newton's Method

- In order to solve 0 = f(x), we can apply standard root-solving methods.
- The most prominent is Newton´s Method:



Newton’s Method: Convergence

Robotics and Mechatronics Centre

- Newton Converges with order 2.
- This means that within the convergence area, the number of correct digits doubles per step.
- For linear systems of equations, Newton converges in a single step.

Newton's Method: MultiDim

Robotics and Mechatronics Centre

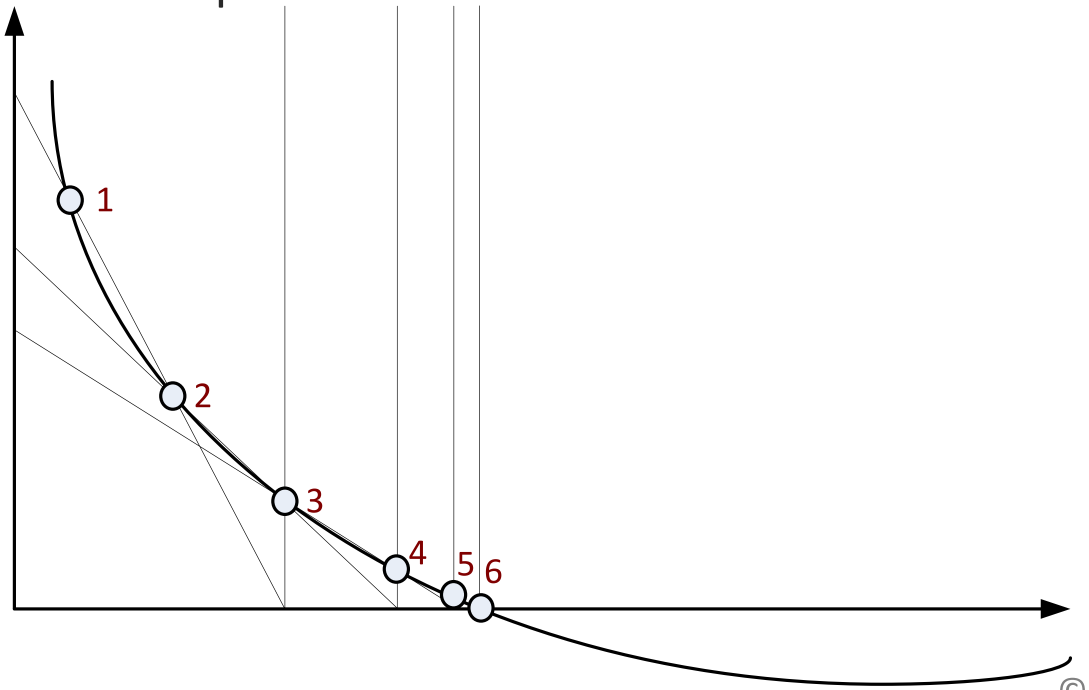
- Newton's Method can be extended to the multi-dimensional case $\mathbf{r} = f(\mathbf{x})$ where $\mathbf{r}$ and $\mathbf{x}$ are vectors of size n.
- To this end, we need to compute the Jacobian: $\mathbf{J} = Df(\mathbf{x})$
  Example for n = 3:

$$\mathbf{J} = \begin{bmatrix} \partial f(x)_1/\partial x_1 & \partial f(x)_1/\partial x_2 & \partial f(x)_1/\partial x_3 \\ \partial f(x)_2/\partial x_1 & \partial f(x)_2/\partial x_2 & \partial f(x)_2/\partial x_3 \\ \partial f(x)_3/\partial x_1 & \partial f(x)_3/\partial x_2 & \partial f(x)_3/\partial x_3 \end{bmatrix}$$

- Iteration formula: $\mathbf{x}_{\text{new}} = \mathbf{x} - \Delta\mathbf{x}$
- Increment: $\mathbf{J}\Delta\mathbf{x} = f(\mathbf{x})$
- In order to compute the increment, we need to solve a linear system of equations. This is of effort $O(n^3)$

Secant Method

- Newton requires the computation of the (partial) derivatives within the Jacobian Matrix.
- When these derivatives cannot be computed, we can take use of methods that approximate the derivates during the process of convergence.
- One example is the Secant Method:



Selection of Tearing Variables

Robotics and Mechatronics Centre

- There is one open question: How do we select the tearing-variables?
- Let us look at an example:

- E2: $e = c+d+f$ (e is known)
  E3: $b = d$
  E4: $c = d$
  E5: $b = f$

- If we select c and b as tearing variables, we get two residuals:
  E2: $f := e-c-d$
  E3: $d := b$
  E4: residual1 := $c-d$
  E5: residual2 := $b-f$

Selection of Tearing Variables

- There is one open question: How do we select the tearing-variables?
- Let us look at an example:

- E2: $e = c+d+f$ (*e* is known)
  E3: $b = d$
  E4: $c = d$
  E5: $b = f$

- But selecting *d* as tearing variable, turns out to be sufficient:
  E3: $b := d$
  E4: $c := d$
  E5: $f := b$
  E2: residual := $c+d+f-e$

Selection of Tearing Variables

- By choosing a good set of tearing variables, we try to minimize the number of required tearing variables and residuals.
- This minimizes the effort of each Newton iteration and minimizes the amount of code that needs to be generated.
- Technically, we want to minimize the number of rows in the BLT-form that have non-zero entries above the diagonal.
- Unfortunately, this optimization problem is NP-hard.
- Hence, heuristics are applied. (For instance, choose the equation that has the lowest number of unknowns. From this equation, pick the variable that occurs in most other equations.)

Some Remarks on Tearing

- Some remarks: Applying an iterative solver on tearing variables is just one of many methods to solve a system of non-linear equations.

- For band-matrices, tearing might be very tempting, since only a very small number of tearing variables is sufficient. But frequently, it is numerically highly unstable.

- In addition to tearing, Dymola transforms small linear systems into explicit form by symbolic manipulations.

Some Remarks on Blocks

Robotics and Mechatronics Centre

- So far, we looked at the BLT-Transformation from a purely structural viewpoint, assuming an equation can be solved for all its variables.
- In reality this is too simple. Let us consider:

$$a = \sin(\varphi)$$

- By all means, we want to avoid causalizing for phi:

$$\cancel{\varphi = \mathrm{asin}(a)}$$

- Since this just picks one out of infinitely many solutions and might introduce discontinuities, when φ crosses $n \cdot \pi$ during simulation time.

Some Remarks on Blocks

- Usually, one solves an equation only for those variables that are linear extractable. This means that x is only part of linear terms.
- Hence in reality, there might be more blocks than necessary from a pure structural viewpoint.
- One can also apply the tearing first and identify the blocks later. However, also this procedure may introduce extra blocks.
- Blocks are not nested!

Summary

- The goal is to bring the flat system of DAEs into a form suited for numerical ODE solvers.

- To this end, the BLT-form is desired.
  - The blocks can be identified using the Dulmage-Mendelsohn Permutation
    - Perfect Matching
    - Tarjan’s Algorithm

- The individual Blocks are solved iteratively by using the tearing method.

Questions?