Objects, Re-use, and Causality

\[ V_1 - V_2 = R \cdot I \]

\[ I = \frac{V_1 - V_2}{R} \]

\[ V_2 = V_1 - R \cdot I \]

\[ V_1 = V_2 + R \cdot I \]
Causality Assignment

\[
\begin{align*}
  x + y + z &= 0 \quad \text{Equation 1} \\
  x + 3z + u^2 &= 0 \quad \text{Equation 2} \\
  z - u - 16 &= 0 \quad \text{Equation 4} \\
  u - 5 &= 0 \quad \text{Equation 4}
\end{align*}
\]
Causality Assignment = bipartite (dependency) graph maximum cardinality matching
Causality Assignment: causality assigned

\[
\begin{align*}
  x + y + z &= 0 \quad \text{Equation 1} \\
  x + 3z + u^2 &= 0 \quad \text{Equation 2} \\
  z - u - 16 &= 0 \quad \text{Equation 4} \\
  u - 5 &= 0 \quad \text{Equation 4} \\
\end{align*}
\]

\[
\begin{align*}
  y &= -x - z \\
  x &= -3z - u^2 \\
  z &= u + 16 \\
  u &= 5
\end{align*}
\]
causality assignment: network flow

source

Equation 1
variable "x"

Equation 2
variable "y"

Equation 3
variable "z"

Equation 4
variable "u"

sink
Network Flow Problems

\[ G = [V, E] \]

Directed graph \( G \) with source \( s \) and sink \( t \)
Network Flow: definitions

Positive capacity $\text{cap}(v, w)$ on every edge $[v, w]$.
$\text{cap}(v, w) = 0$ if $[v, w]$ is not an edge.

A flow on $G$ is any real-valued function $f$ with properties:

1. *skew symmetry.* $f(v, w) = -f(w, v)$.
   
   $f(v, w) > 0$ is called a flow from $v$ to $w$.

2. *capacity constraint.* $f(v, w) \leq \text{cap}(v, w)$. If $[v, w]$ is an edge such that $f(v, w) = \text{cap}(v, w)$, the flow is said to saturate $[v, w]$.

3. *flow conservation.* For every vertex $v$ other than $s$ and $t$

\[
\sum_w f(v, w) = 0
\]
Network Flow: *maximum flow*

The value $|f|$ of a flow $f$ is the net flow out of the source

$$
\sum_v f(s, v)
$$

*Maximum Flow Problem* (Ford and Fulkerson).
Network Flow: cut

A cut: partition $X, \overline{X}$ of the vertex set $V$ into two parts $X$ and $\overline{X} = V - X$ such that $X$ contains $s$ and $\overline{X}$ contains $t$. The capacity of a cut $X, \overline{X}$ is

$$cap(X, \overline{X}) = \sum_{v \in X, w \in \overline{X}} cap(v, w)$$

Flow across a cut is

$$f(X, \overline{X}) = \sum_{v \in X, w \in \overline{X}} f(v, w)$$
Network Flow: *max-flow min-cut theorem*

For any flow $f$, the flow across any cut $X, \bar{X}$ is equal to the flow value. Capacity constraint → flow across cut cannot exceed capacity of the cut. *Maximum flow* is not greater than the capacity of a *minimum cut*.

*max-flow min-cut theorem: maximum flow = minimum cut*
Network Flow: *residual graph*

*Residual capacity for flow* $f$

\[ res(v, w) = cap(v, w) - f(v, w) \]

Up to $res(v, w)$ additional flow can be pushed along $[v, w]$.

*Residual graph* $R$ is graph with edges $res(v, w)$.

*Augmenting path* from $s$ to $t$.

*Residual capacity* is minimum $res(v, w)$. 
Network Flow: *residual graph*
Network Flow: *augmenting path*
Ford Fulkerson

- Augmenting step
  1. Find an augmenting path $p$ for the current flow.
  2. Increase the value of the flow by pushing $\text{res}(p)$ units of flow along $p$.

- Pathfinding step
  1. Find a path $p_i$ from $s$ to $t$ in $G^*$.
  2. Let $\Delta_i$ be the minimum of $f^*(v, w)$ for $[v, w]$ an edge of $p_i$.
     For every edge $[v, w]$ on $p_i$, decrease $f^*(v, w)$ by $\Delta_i$ and delete $[v, w]$ from $G^*$ if its flow is now zero.
  3. Increment $i$ by one.
Path Finding: which path?

- Edmonds and Karp:
  augmentation along path with maximum residual capacity.

- Dinic:
  augmentation along shortest augmenting path.
Length: number of edges a path contains.
Dinic’s algorithm: find blocking flows to saturate edges

1. Begin with zero flow.

2. Find a blocking flow $f'$ on the level graph for the current flow $f$.
   Blocking flow: every path from the source $s$ to the sink $t$ contains a saturated edge.

3. Replace $f$ by the flow $f + f'$ defined by:

   $$(f + f')(v, w) = f(v, w) + f'(v, w).$$

4. Repeat until the sink $t$ is not in the level graph for the current flow.
Level Graph

- $R$: the residual graph for a flow $f$.
- $level$ of $v = \text{the length of the shortest path from } s \text{ to any vertex } v \text{ in } R$.
- $Level \ graph \ L$ for $f = \text{the subgraph of } R \text{ containing}$
  - only the vertices reachable from $s$
  - only the edges $[v, w]$ such that

$$level(w) = level(v) + 1.$$ 

$L$ contains every shortest augmenting path and can be constructed in $O(m)$ time by breadth-first search.
Finding a Blocking Flow (DFS)

- **Initialize**: Let $p = [s]$ and $v = s$. Go to Advance.

- **Advance**: If there is no edge out of $v$, go to Retreat. Otherwise, let $[v, w]$ be an edge out of $v$. Replace $p$ by $p & [w]$ and $v$ by $w$. If $w \neq t$ repeat Advance; if $w = t$ go to Augment.

- **Augment**: Let $\Delta$ be the minimum of $(\text{cap}(v, w) - f(v, w))$ for $[v, w]$ an edge of $p$. Add $\Delta$ to the flow of every edge on $p$, delete from $G$ all newly saturated edges, and go to Initialize.

- **Retreat**: If $v = s$ halt. Otherwise, let $[u, v]$ be the last edge on $p$. Delete $v$ from $p$ and $[u, v]$ from $G$, replace $v$ by $u$, and go to Advance.
Dinic Performance

\(m\) is number of nodes, \(n\) is number of edges

- Finds a blocking flow in \(O(nm)\) time, and a maximum flow in \(O(n^2m)\) time.

- On a unit network, Dinic's algorithm finds a blocking flow in \(O(m)\) time, and a maximum flow in \(O(n^{1/2}m)\) time. Unit network: edge capacities integer, each vertex \(v\) other than the source and the sink has either a single entering edge of capacity one, or a single outgoing edge of capacity one.

- On a network whose edge capacities are all one, Dinic's algorithm finds a maximum flow in \(O(\min\{n^{2/3}m, m^{3/2}\})\) time.
Example
Symbolic Manipulation (Computer Algebra)

Simplification of expressions, re-writing of equations, symbolic solving, . . .

- Mathematica
- REDUCE
- AXIOM
- MACSYMA
- MuPAD (http://www.mupad.de/)
(muPAD) examples

```
>> 100!;
93326215443944152681699238856266700490715968264381621468592963895217599993\n229915608941463976156518286253697920827223758251185210916864000000000000000
000000000

>> (x+1)^4;

        4
    (x + 1)

>> expand(%);

2            3            4
4 x + 6 x + 4 x + x + 1

>> x^2+2*x+1;

        2
    2 x + x + 1
```
(muPAD) examples

>> factor(%%);

[1, x + 1, 2]

>> diff(x^2+2*x+1,x);

2 x + 2

>> int(x^2+2*x+1,x);

3

2 x

x + x + --

3

>> 2+3+4+x+4;

x + 13

>> 2+3+x+y+x*y+x^2+y^3+4;

2 3

x + y + x y + x + y + 9
(muPAD) examples

```
>> solve({x+a*y-2,x-b*y+4},{x,y});

{ { 2 b - 4 a   6 } }
{ { x = -------, y = ----- } }
{ { a + b       a + b } }

>> subs(%a=3,b=4);

{ { x = -4/7, y = 6/7 } }

>> generate::C(x^2+4-sin(y));

"  " t4 = -sin(y) + x*x + 4.0 ;"

>> generate::TeX(x^2+4-sin(y));

" - \sin\left(y\right) + x^2 + 4"
```
Canonical Form

\[
\begin{align*}
\text{\texttt{>> (y+2)+3 + x;}} & \quad \text{\texttt{x + y + 5}} \\
\text{\texttt{>> 5+y+x;}} & \quad \text{\texttt{x + y + 5}} \\
\text{\texttt{>> 2+3+2*y+x-y;}} & \quad \text{\texttt{x + y + 5}} \\
\text{\texttt{>> 2+x -y -x;}} & \quad \text{\texttt{2 - y}} 
\end{align*}
\]
Canonical Form

(Davenport)

A representation of a mathematical object (e.g., polynomial) is canonical if two different representations always correspond to two different objects.

A correspondence $f$ between a class $O$ of objects and a class $R$ of representations is a representation of $O$ by $R$ if each element of $O$ corresponds to one or more elements of $R$ (otherwise it is not represented) and each element of $R$ corresponds to one and only one element of $O$ (otherwise we do not know which element of $O$ is represented).

The representation is canonical if $f$ is bijective. With a canonical representation it is possible to check equality of objects by verifying that their representations are equal.
Normal Form

If $O$ has the structure of a monoid, a weaker concept may be defined. A representation is called *normal* if zero has only one representation. Every canonical representation is normal, but the converse is false.

Having a unique representation for zero is important to be able to test for division by zero.

A normal representation over a group also gives us an algorithm to determine whether two elements $a$ and $b$ of $O$ are equal. It is sufficient to check whether $a - b = 0$. In a canonical representation, it suffices to check whether $a$’s and $b$’s representations are identical.
Regular and Natural form

A representation should be *regular*

\[ A = x^2 + x, \]
\[ A + 1 = (x^3 - 1)/(x - 1), \]
\[ A - x = x^2, \ldots \text{is not regular}. \]

Representations must be *natural*. Some form of simplification should occur. For polynomials in one variable, every power of \( x \) should appear at most once, and powers should be sorted in ascending or descending order.
Polynomials

Representations of polynomials: *dense* and *sparse*.

- **Dense**: vector of coefficients
- **Sparse**: list of (coeff, degree) tuples

\[
(x^{1000} + 1)(x^{1000} - 1) = x^{2000} - 1
\]

Polynomial *in* a particular variable: \( \sin(x) + 3 \sin^2(x) - 2 \)
is polynomial in \( \sin(x) \)

Increasing or decreasing powers.
Polynomials in multiple variables

canonical, natural representation

Different types of ordering:

- **lexicographic**: alphabetically ordered. Within one variable name, ordered by powers. If the powers of that variable are the same, look at the next (lexicographic) variable. \( x^2 + 2xy + x + y^2 + y + 1 \)

- **total degree, then lexicographic**: lexicographic distinction between same total degree, ordered by total degree. \( x^2 + 2xy + y^2 + x + y + 1 \)

- **total degree, then inverse lexicographic**: \( y^2 + 2xy + x^2 + y + x + 1 \)
Types, Domains, Algebraic Structures

Used to define \textit{generic} operations

Definitions (Birkhoff & McLane)

1. Semigroup $S, +$
   - closure
   - associativity

2. Monoid
   - Semigroup with unit 0

3. Group
   - Identity 1
   - Inverse
4. Commutative (Abelian)
   - Semigroup
   - Monoid
   - Group

5. Ring $R$, $+$, $\ast$
   - $R$, $+$ Abelian Group
   - $R$, $\ast$ Monoid with unit 1
   - $\ast$ is distributive on both sides over $+$

6. Commutative Ring
   - Ring and $R$, $\ast$ is commutative

7. Field $=$ Commutative Ring
   - each non-zero element has multiplicative inverse
Computer Algebra ~ Compilers

1. lexical analysis

2. syntactic analysis (grammar parsing)

3. intermediate representation: Abstract Syntax Tree (AST) and Symbol Table (ST)

4. operations (symbolic manipulation) on AST+ST

5. compiler compilers
   - Gentle http://www.first.gmd.de/gentle
   - TRAP http://www.first.gmd.de/smile/ Personally
   - ANTLR http://www.antlr.org
   - PCCTS http://www.ocnus.com/pccts.html
• Catalog of Compiler Construction Tools
  
  http://www.first.gmd.de/cogent/catalog
Internal model representation

- Abstract Syntax Tree + Symbol Table

\[ b + 2 - (a + 3) = x \]

\[ = [ +[b, +[2, -[a, 3]], x] \]

- From the AST + ST, a dependency graph can be built.
  1. for causality assignment,
  2. for equation re-write,
  3. for loop detection and sorting,
  4. for constant folding,
  5. for parameter expression lifting \((-K/g)\),
  6. for output equation selection
Constant Folding Graph Grammar

\[ c_1 + c_2 \]

Rule 1.

```
OP: +
CONST: c1
CONST: c2
::= CONST: c1+c2
```
Constant Folding Transformation

\[ v = 2 + (3 + 4) \]
Canonical Representation

To encode associativity and commutativity of operators.
A representation of a mathematical object is canonical if two different representations always correspond to two different objects.

1. $n$-ary operators
2. $inv$ for each operator
3. lexicographic ordering

\[ +[2, \ inv[3], \ a, \ inv[b]] \]
\[ +[inv[1], \ a, \ inv[b]] \]

\[ 2 - 3 + a - b \Rightarrow -1 + a - b \]

\[ \rightarrow \text{ symbolic operations (simplify, analyze, \ldots) } \]
\[ \rightarrow \text{ re-use AND performance! } \]
Object-Oriented Modelling of Physical Systems

1. Encapsulation, objects, classes, …

2. Types (Software vs. Dynamical systems)
   - Subtypes
   - Contravariance
   - Semantics of composition

3. Inheritance

4. Different levels of abstraction
Based on . . .

- WEST (bioactivated sludge waste water treatment)
  www.hemmiswest.com

- Modelica (ESPRIT Basic Research, now Association)
  www.modelica.org

Aims:
- Standard language for model exchange and re-use
- Support non-causal, hybrid, hierarchical modelling
- Semantics based on Hybrid DAEs
- Separate model (goal: re-use, exchange)
  from its numerical solution (goal: accuracy, speed)
- Library of basic models
Electrical example: Modelica vs. Matlab/Simulink
Electrical Types

type Time = Real (final quantity="Time", final unit="s");
type ElectricPotential = Real (final quantity="ElectricPotential", final unit="V");
type Voltage = ElectricPotential;
type ElectricCurrent = Real (final quantity="ElectricCurrent", final unit="A");
type Current = ElectricCurrent;
Electrical Pin Interface

connector PositivePin "Positive pin of an electric component"
  Voltage v "Potential at the pin";
  flow Current i "Current flowing into the pin";
end PositivePin;
Electrical Port

partial model OnePort

  "Component with two electrical pins p and n
  and current i from p to n"
Voltage v "Voltage drop between the two pins (= p.v - n.v)"
Current i "Current flowing from pin p to pin n"
  PositivePin p;
  NegativePin n;
equation
  v = p.v - n.v;
  0 = p.i + n.i;
  i = p.i;
end OnePort;
Electrical Resistor

model Resistor "Ideal linear electrical resistor"
extends OnePort;
parameter Resistance R=1 "Resistance";
equation
    R*i = v;
end Resistor;
The circuit

model circuit
   Resistor R1(R=10);
   Capacitor C(C=0.01);
   Resistor R2(R=100);
   Inductor L(L=0.1);
   VsourceAC AC;
   Ground G;

equation
   connect(AC.p, R1.p);
   connect(R1.n, C.p);
   connect(C.n, AC.n);
   connect(R1.p, R2.p);
   connect(R2.n, L.p);
   connect(L.n, C.n);
   connect(AC.n, G.p);
end circuit;
Dynasim Modelica demo
Future

- multi-formalism
- multi-abstraction
- meta-modelling (AToM³)