

Representing Systems of Interacting Components in Euclid

K J Drylllerakis

M J Sergot

Department of Computing

Imperial College

{kd,mjs}@doc.ic.ac.uk

Jan 1996

Overview

- Modelling of Systems with Interacting Components
- Domain Logic Programming: a variant of the CLP scheme
- The language EUCLID: Syntax and Semantics
- Solving Electrical Circuits
- Conclusions

What is EUCLID

- a programming language of the LP paradigm
- syntax & operational semantics close to CLP(X) languages
- standard query mechanism with qualified answers (constraints)
- multiple (mathematical) domains: reals, intervals, functions, vectors
- working implementation: Prolog+C+CLP(R)+Mathematica

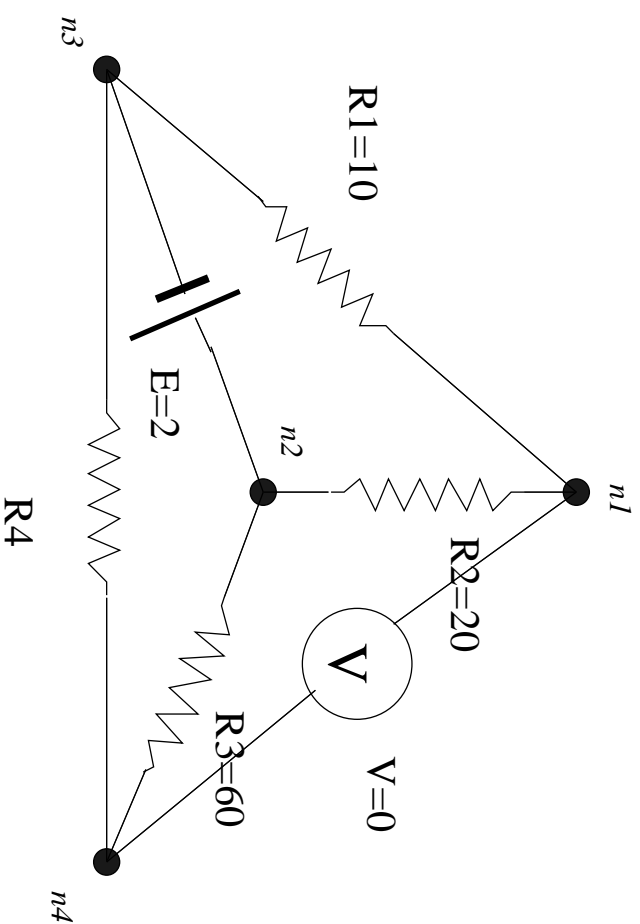
On EUCLID Syntax

- Programs are collections of *constraint clauses* representing a logical theory
- Constraint clauses have the form
Head :- [Typing], [Constraints], [Body]
- Domain and Classical Terms made from a first order alphabet
- Constraints and Predicates
- Variables are typed

EUCLID Domains

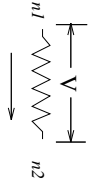
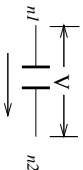
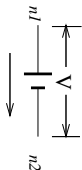
	Reals	Intervals	Formulae	Real Functions
Constants & Terms	0, 0.1, ... inf _{ty} , pi e, c	cc(<i>i</i> , <i>j</i>) oc(<i>i</i> , <i>j</i>) oo(<i>i</i> , <i>j</i>) co(<i>i</i> , <i>j</i>)	$x+3/x$ $f(x) = x + 3$ $R \rightarrow R$	{[cc(0,inf _{ty}):cos(X)//X]}
Operators	, +, -, pow, sqrt sin, ...		, +, - @, integral derivative	@, integral derivative, # limit, o
Relations	= <, >, ≠	= meets, >, <	=	= continuous

Electrical Circuits

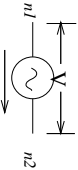
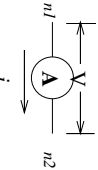
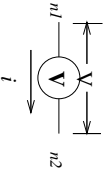


- 2-connector components
- nodes
- measurement techniques

Electrical Components I

 <p>Resistance R</p> <p>$V(t) = Ri(t)$</p>	<p>component (resistor, [R], Vt, It):- real (R), formulae ([Vt, It]), Vt=R*It</p>
 <p>Capacitance C</p> <p>$\frac{dV}{dt} = \frac{1}{C}i(t)$</p>	<p>component (capacitor, [C], Vt, It):- real (C), formulae ([Vt, It]), It=C*derivative(Vt).</p>
 <p>$\mathcal{EMF} \quad E$</p> <p>$V(t) = E$</p>	<p>component (dc_source, [E], Vt, It):- real (E), formulae ([Vt, It]), Vt=E//T</p>

Electrical Components II

 <p style="text-align: center;">$\mathcal{EMF}^i \mathcal{E}, \omega$</p> <p style="text-align: center;">$V(t) = \mathcal{E} \sin \omega t$</p>	<p>component (ac_source, [E, W], Vt, It):- reals ([E, W]), formulae ([Vt, It]), Vt=E*sin(W*T)//T</p>
	<p>component (ampere_meter, [A], Vt, It):- formulae ([A, Vt, It]), A=It, Vt=0//T</p>
	<p>component (volt_meter, [V], Vt, It):- formulae ([V, Vt, It]), V=Vt, It=0//T</p>

Solution Mechanism

- Give a description as a list of components [*type*, *chars*, [*n*₁, *n*₂]]
- Get instantiation/constraining on the variables
- Use formulæ to represent time dependencies

```
solve_circuit(CircuitDescription):-
  prepare_circuit(CircuitDescription,Circuit),
  calculate_loops(Circuit,Loops),
  Circuit=[Currents,Voltages],
  kirchoff_law1(Currents),
  kirchoff_law2(Voltages,Loops).
```

Governing Laws

Kirchoff's Law of currents: *the sum of all currents at a node is*

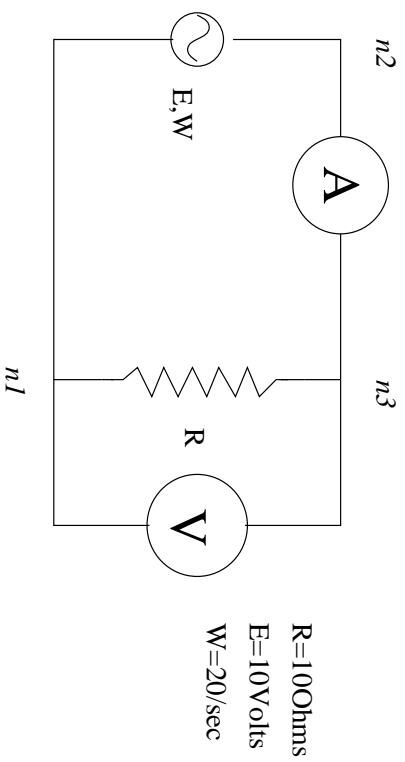
zero

```
kirchoff_law1 ([ ] ).  
kirchoff_law1 ([n(Node, Currents) | Nodes]):-  
  add_currents(Currents, TotalCurrent),  
  TotalCurrent=0//X,  
  kirchoff_law1(Nodes).
```

Kirchoff's Voltage Law: *the sum of all potential differences in a closed loop is zero*

```
kirchoff_law2(Voltages, [ ] ).  
kirchoff_law2(Voltages, [Loop|Loops]):-  
  add_voltages(Loop, Voltages, 0//X),  
  kirchoff_law2(Voltages, Loops).
```

Simple E-R circuit



```

euclid ?- solve_circuit( [
[ac_source, source1, [10, 20], [n1, n2]],
[ampere_meter, am1, [I], [n2, n3]],
[resistor, res1, [10], [n3, n1]],
[volt_meter, vol1, [V], [n3, n1]]]).

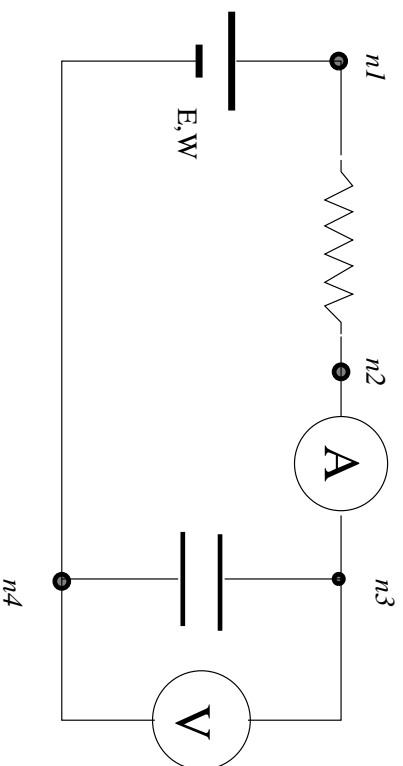
```

```
[B1, A1]:reals
```

```
I= -sin(20*B1)//B1
```

```
V= -10*sin(20*A1)//A1
```

Simple E-R-C circuit



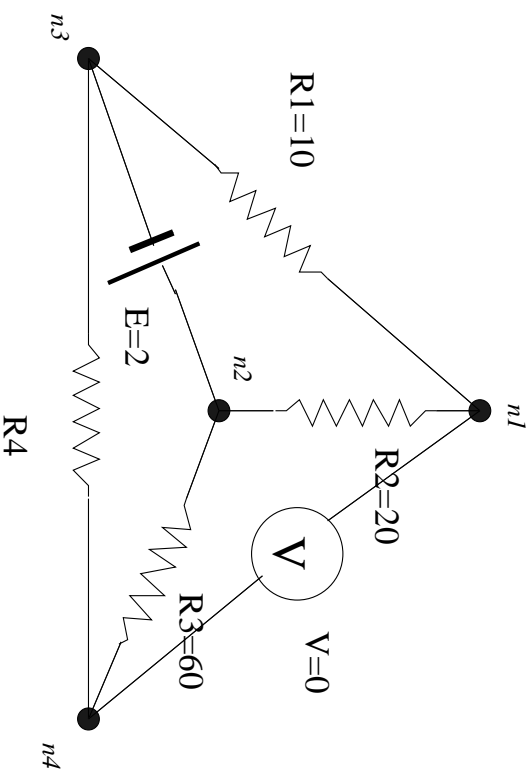
```
euclid ?- V00=0, solve_circuit( [
    [resistor, re1, [R], [n1,n2]],
    [ampere_meter, am1, [I], [n2,n3]],
    [capacitor, cap1, [C], [n3,n4]],
    [volt_meter, vo1, [V], [n3,n4]],
    [dc_source, source1, [E], [n4,n1]]]).
```

```
[A1,R,C,E]:reals
```

```
I= -E/ (R*e^(A1/ (R*C)))/A1
```

```
V= -E+E/e^(A1/ (R*C))/A1
```

Wheatstone Bridge



```

euclid ?- solve_circuit ( [
[volt_meter, v1, [0//X], [n1, n4]],
[dc_source, s1, [2], [n3, n2]],
[resistor, r1, [10], [n3, n1]],
[resistor, r2, [20], [n2, n1]],
[resistor, r3, [60], [n4, n2]],
[resistor, r4, [R], [n3, n4]]
]).

```

`[X]:reals`

`R=30`

Conclusions

- EUCLID offers a natural way to represent and solve physics and engineering problems
- Computed answers can be numeric or abstract-symbolic
- Formulae and function domains offer easy representation of time dependent quantities
- More examples in Scientific Knowledge Representation have been solved successfully with the same technique