

Weak equivalence for constraint sets

Sieger van Denneheuvel & Karen L. Kwast
Department of Mathematics and Computer Science
University of Amsterdam
Plantage Muidergracht 24, 1018 TV, Amsterdam

Abstract

We describe a generalization of equivalence between constraint sets, called *weak equivalence*. This new equivalence relation takes into account that not all variables have the same function in a constraint set and therefore distinguishes between *restriction variables* and *intermediate variables*. We explore the properties of weak equivalence and its underlying notion of weak implication with an axiomatic approach. In addition a complete set of axioms for weak implication is presented. With examples derived from the declarative rule language RL we show the applicability of weak equivalence to constraint solving.

1 Introduction

Recently, constraint solving systems have emerged that allow declarative constraint processing (see for example Mathematica [Wolfram, 1988], Bertrand [Leler, 1988] and CLP(R) [Jaffar and Michaylov, 1987]) whereas more traditional systems required user intervention to direct the solver towards a desired solution. From a general point of view, a typical input specification for these systems consists of a set of *constraints* and a set of *restriction variables* which are "interesting" variables as far as the solver is concerned. The remaining variables in the set of constraints are *intermediate variables*. For constraint solving, restriction variables can be further divided in two mutually disjunct sets, namely *known variables* and *wanted variables*.

Given the above input specification the objective of the constraint solver is to express all wanted variables symbolically in terms of known variables (i.e. without supplying actual values for the known variables). In such a symbolic solution only restriction variables are allowed and as a consequence all intermediate variables must be eliminated from the input constraint set. As a simple example consider the input constraint set $\{x = y + 2, y = z + 2\}$ where z is known and x is wanted, so x and z together are the restriction variables. A constraint solver set to solve this problem will probably return $\{x = z + 4\}$ as the solution, thereby eliminating the intermediate variable y . For such a solver output, we would like to say that it is in some way equivalent with

the input constraint set. Unfortunately the standard definition for equivalence among constraint sets would classify the example input and output as inequivalent, since the output set does not impose any restriction to the intermediate variable y .

Therefore we define a more general equivalence, called *weak equivalence* which distinguishes between restriction variables and intermediate variables.

2 The rule language family RL

Our primary motivation for investigation of weak equivalence lies in its role in the integration of relational databases and constraint solving. This integration is one of the aims of the rule language family RL (see [Van Emde Boas, 1986a, Van Emde Boas, 1986b]). In the RL languages, knowledge can be represented in three different types of rules: *tabular rules*, *clauses* and *constraints*. Corresponding to these types of rules there are three areas of technology that support that style of knowledge processing in isolation: Database Systems, Logic Programming Systems, and Spreadsheets. A main goal of RL is to integrate these three technologies in one knowledge base system. Query processing should be executed with the help of an existing relational database system; knowledge and queries expressed in RL are to be pre-processed by a constraint solver, and to be compiled into a database query language so that large amounts of data can be processed effectively.

In the current prototype of RL/1 we focus on the integration of a constraint solver with a relational database system; RL/1 has been presented elsewhere [Denneheuvel, 1990]. The RL/1 prototype system has been implemented in Prolog which includes a language parser, a database system and a constraint solver. The examples presented in this paper were processed successfully by this prototype.

Terms in RL/1 are constructed inductively in the usual way from constants, variables and functions. Both numeric and string domains are supported. For use in the database application we include binary functions such as $+$, $-$, $*$ and $/$. *Constraints* include expressions of the form $t_1 \text{ op } t_2$ with $\text{op} \in \{=, >, <, \neq, \leq, \geq\}$ and two special constraints false and true. Null values are not available in the language itself but may be present in the tables of the underlying database (cf. [Kwast, 1991]).

In **RL/1** a distinction is made between *extensional*

objects and *intensional* objects. Extensional objects correspond to base tables in the underlying relational database. Intensional objects on the other hand are objects whose relations can be materialized by evaluation of their definition. An intensional object is defined by one or more *rules*, alternatively called clauses. A rule typically consists of a *rule head* and a *rule expression* separated by the keyword WHEN. The rule expression contains constraints or invocations of intensional and extensional objects, separated by the conjunctive AND operator. Defining an intensional object with several rules expresses *disjunction* between the rules. The question whether a rule expression as a whole denotes a finite or infinite relation is in general undecidable; partial information about this question is obtained from the constraint solver.

2.1 Query commands

Query commands in RL/1 result in an *answer relation* which consists of attributes and a (possibly empty) set of tuples. Optionally the result of a query command can be stored in an extensional table object. There are the following types of query commands:

```
INFER ( attribute-list ) WHEN
      rule-exp [ TO table ]
SHOW rule-exp [ TO table ]
```

The INFER command yields an answer relation with attributes equal to the attribute list between the INFER and WHEN keywords. The SHOW command yields an answer relation whose attributes are the variables occurring in the rule expression.

Processing the above queries requires one or more invocations of the constraint solver. The solver output is used to compile the query into a database request. Symbolic queries on the other hand work by presenting this intermediate solver output to the user directly:

```
SYMINFER ( attribute-list ) WHEN
          rule-exp [ TO file ]
SYMSHOW rule-exp [ TO file ]
```

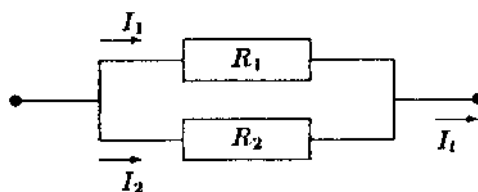
The 'TO file' option allows the computed symbolic answer to be stored in a text file. Symbolic query commands produce one or more of the following answers:

```
condition= Condition set
solutions Solution set
```

The solution set contains elements of the form $x = t$ with x a wanted variable and t a term. The terms t in the solution set contain only known variables. The condition part consists of constraints over the known variables; it states under what additional restrictions the obtained solution is valid. Intermediate variables are eliminated from both the condition and the solution set. Known variables are determined by the rule expression of the query and wanted variables by the attribute list of the query.

3 Symbolic computation in RL/1

Intensional objects are declarative representations of knowledge; therefore it is not known beforehand how the



```
module resistors(number:vt:it:rt:
  r1:r2:i1:i2:v:i:r).
table experimentdata(vt,r1,r2)=
  [[9,1200,400],[10,1200,400],
  [11,1200,400],[12,1200,400]].
clause ohmlaw(v,i,r) when v=i*r.
clause maxwatt(v,i) when v*i<=0.25.
clause cstr(vt,rt,it,i1,i2,r1,r2) when
  ohmlaw(vt,i1,r1) and ohmlaw(vt,i2,r2)
  and ohmlaw(vt,it,rt) and it=i1+i2.
clause circ(vt,it,rt,r1,r2) when
  cstr(vt,rt,it,i1,i2,r1,r2)
  and maxwatt(i1,vt) and maxwatt(i2,vt).
```

Figure 1: Parallel resistors

attributes of an object are invoked. For instance consider the module in Figure 1 for a parallel circuit of two resistors. In the representation of the circuit both the physical laws of basic electricity theory and the practical condition that the resistors should not be overloaded are expressed in terms of constraints. We know from elementary physics that the values given in the table *experimentdata* fully determine all quantities. How should the RL/1 knowledge base determine that this is indeed the case? The constraint solver is capable to figure out the determinedness.

Suppose voltage is applied to the example circuit then the current through the circuit and the total resistance can be calculated:

```
show circ(9,it,rt,1200,400)
  rt  it
  ---  ---
  300  0.03
```

With other variables given, an answer is also computed:

```
show circ(vt,0.03,rt,1200,400)
  vt  rt
  ---  ---
  9   300
```

In the constraints for the circuit the law for parallel resistors is not given directly, but the total resistance r_t can nevertheless be computed. Now suppose we apply various voltages to the two resistors as specified in the *experimentdata* table in Figure 1. In the experiment the voltage is slowly increased and the resistors stay the same. Surely at some high voltage the circuit will break down. Let's run the experiment (the '(*)' notation expands to all attributes of the involved object):

```
show experimentdata(*) and circ(*)
  vt  r1  r2  rt  it
  ---  ---  ---  ---  ---
  9   1200  400  300  0.03
  10  1200  400  300  0.033
```

In the answer relation the rows are corresponding to the rows from the experiment data table but obviously some rows are missing. In these missing rows the constraints were not satisfiable (i.e. the non-smoking condition $maxwatt(i2, vt)$ was violated) so they do not appear in the answer to the query.

The capability to produce symbolic output is an important feature of the RL/1 knowledge base. It allows the user to analyze relationships between variables and to infer relationships that hold in special circumstances. A general relationship between the resistance of the individual components and the total resistance can be found with the following symbolic query:

```

syminfer(rt) when
  known(r1,r2) and cstr(*)
solution= rt=r1*r2/(r1+r2)

```

The constraint solver has inferred the law for parallel resistors.

The classification of variables in the three disjoint partitions of wanted, known and intermediate variables is crucial for derivation of relationships like the previous example. By making r_j and r_2 known and r_i wanted the constraint solver of the RL/1 system was directed towards a solution that expresses r_i in terms of r_1 and r_2 . The constraint solver actively applies this information in its inference process to yield the desired result.

On the other hand suppose that variables of a constraint set were classified in only two partitions of wanted and intermediate variables as is the case in many existing constraint solving systems. The constraint solver would correctly establish that the equation system is undetermined (there are too many variables and not enough equations to solve them).

In the next sections we formalize the notion of equivalence that is used by the solver.

4 Weak equivalence and implication

First we introduce some definitions and notations. Variable names are chosen from non capitals (x, y, z, \dots) and sets of variables from capitals (A, Y, Z, \dots). V is the set of all variables. Constants are denoted by c and functions by f . Terms are denoted by t . Sets of constraints are represented as A, B, C or D . To denote the cardinality of a set I we use the notation $\|I\|$.

Solutions are constraints of the form $x = t$ with $x \notin \text{var}(t)$. The variable x is called the *head variable* and the variables in the term t the *tail variables*.

Definition 1 A solution set is a finite set

$\{x_1 = t_1, \dots, x_n = t_n\}$, satisfying:

1. $\|\{x_1, \dots, x_n\}\| = n$, (the variables are distinct)
2. $\{x_1, \dots, x_n\} \cap \text{var}(\{t_1, \dots, t_n\}) = \emptyset$

Definition 2 A tuple is a solution set of the form $\{x_1 = c_1, \dots, x_n = c_n\}$ with c_1, \dots, c_n constants.

Solution sets are denoted by Φ, Ψ and Θ and tuples by ϕ, ψ and θ . For a solution set $\Phi = \{x_1 = t_1, \dots, x_n = t_n\}$ the head variables x_1, \dots, x_n are denoted as $h\text{v}(\Phi)$.

The operator *restrict* retains those solutions in Φ that have a head in X :

Definition 3 $\Phi[X] = \{x = t \mid x \in X, x = t \in \Phi\}$

Definition 4 $\Phi =_X \Psi$ iff $\Phi[X] = \Psi[X]$

Solution sets $\Phi = \{x_1 = t_1, \dots, x_n = t_n\}$ can be interpreted as substitutions $[x_1 := t_1, \dots, x_n := t_n]$ which can be applied to (collections of) items:

Definition 5 $\Phi(A) = A[x_1 := t_1, \dots, x_n := t_n]$
with $\Phi = \{x_1 = t_1, \dots, x_n = t_n\}$

A tuple ϕ satisfies constraints in A , $\phi \models A$, if and only if ϕ satisfies all constraints in A . A set of constraints A implies a set of constraints B , $A \models B$, if ϕ satisfies B whenever ϕ satisfies A , for all tuples ϕ .

Definition 6 (Strong equivalence)

1. $\phi \models A$ iff $\forall a \in A (\phi \models a)$
2. $A \models B$ iff $\forall \phi (\phi \models A \Rightarrow \phi \models B)$
3. $A \equiv B$ iff $A \models B \ \& \ B \models A$

In the case of weak equivalence these definitions are subscripted with the restriction variables X . The notion of strong satisfaction is changed to weak satisfaction by existentially quantifying the variables not in X (i.e. the intermediate variables). As a consequence the values of these variables in the tuple ϕ are no longer involved in the satisfaction of A .

Definition 7 (Weak equivalence)

1. $\phi \models_X A$ iff $\exists \psi (\psi \models A \ \& \ \phi =_X \psi)$
2. $A \models_X B$ iff $\forall \phi (\phi \models A \Rightarrow \phi \models_X B)$
3. $A \equiv_X B$ iff $A \models_X B \ \& \ B \models_X A$

Example 1 (Equivalence)

$$\begin{aligned} \{x = 1\} \not\models_{x,y} \{y = 1\} & \quad \{x < x + y, y > 0\} \equiv_x \text{true} \\ \{x = 1\} \not\models_x \{y = 1\} & \quad \{x < x + y, y > 0\} \equiv_y \{y > 0\} \\ \{x = y\} \equiv_x \{x = z\} & \quad \{x < x + y, y > 0\} \equiv \{y > 0\} \end{aligned}$$

In the above definitions we presented strong implication between constraint sets as an implication using strong satisfaction in both the premise and the consequent. Weak implication was constructed from this definition by weakening the consequent of the implication to weak satisfaction. It may be conceivable that other interesting definitions arise if instead only the premise of the implication is weakened (*single implication*) or if both the premise and the consequent are weakened (*double implication*):

Definition 8 (Single and double implication)

1. $A \models'_X B$ iff $\forall \phi (\phi \models_X A \Rightarrow \phi \models B)$
2. $A \models''_X B$ iff $\forall \phi (\phi \models_X A \Rightarrow \phi \models_X B)$

In the next proposition we compare these four possible definitions of implication.

Proposition 9 (Relating implication definitions)

1. $A \models'_X B \Rightarrow A \models B$ ($A \models B \not\Rightarrow A \models'_X B$)
2. $A \models B \Rightarrow A \models_X B$ ($A \models_X B \not\Rightarrow A \models B$)
3. $A \models''_X B \Leftrightarrow A \models_X B$

The relationships between the four definitions are summarized in Figure 2 (arrows denote implication). Single implication is too strong and only applicable when $\text{var}(B) \subseteq X$, in which case it is equivalent to strong implication. Double implication is equivalent to weak implication and therefore superfluous.

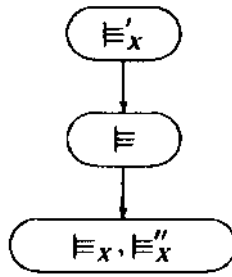


Figure 2: Overview of the implication definitions

4.1 Inconsistency, tautology and redundancy

Inconsistency is normally defined as strong equivalence with the special constraint **false**. As a consequence tautology and redundancy can be defined in terms of strong and weak equivalence. The strong definitions are obtained from the weak definitions by dropping the subscript X .

Definition 10 (Strong and weak inconsistency)

1. $Inc_X(A)$ iff $A \equiv_X \text{false}$
2. $Taut_X(A)$ iff $A \equiv_X \text{true}$
3. $Red_X(A)$ iff $\exists B \subset A (\|B\| < \|A\| \ \& \ A \equiv_X B)$

It turns out that weak and strong inconsistency are actually equivalent:

Proposition 11

1. $A \equiv_X \text{false} \Leftrightarrow A \equiv \text{false}$
2. $A \equiv_X \text{true} \Leftrightarrow A \equiv \text{true}$
3. $Inc(A) \Leftrightarrow Inc_X(A)$

However, contrary to inconsistency, weak tautology is truly weaker than strong tautology:

Proposition 12

1. $Taut(A) \Rightarrow Taut_X(A)$
2. $Taut_X(A) \not\Rightarrow Taut(A)$

It is easy to construct a counter example such that $Taut_X(A)$ but not $Taut(A)$. Let $A = \{x < x + y, y > 0\}$ then it holds that $\{x < x + y, y > 0\} \equiv_{\{x\}} \text{true}$ so $Taut_X(A)$. On the other hand $\neg Taut(A)$ because $\{x < x + y, y > 0\} \equiv \{y > 0\} \not\equiv \text{true}$.

As a consequence weak redundancy is not equivalent to strong redundancy:

Proposition 13

1. $Red(A) \Rightarrow Red_X(A)$
2. $Red_X(A) \not\Rightarrow Red(A)$

Again we can construct a counter example such that $Red_X(A)$ but not $Red(A)$. Let $A = \{x = y + 2, y = z + 2\}$ then it holds that $A \equiv_{\{x,y\}} \{x = y + 2\}$ so $Red_X(A)$. On the other hand $\neg Red(A)$ because both $\{x = y + 2, y = z + 2\} \not\equiv \{x = y + 2\}$ and $\{x = y + 2, y = z + 2\} \not\equiv \{y = z + 2\}$.

5 Weak axioms

In this section we enumerate axioms that hold for implication and equivalence. We start with some properties of strong implication that are maintained for weak implication.

Proposition 14 (The equivalence relation \equiv_X)

1. $A \equiv_X A$
2. $A \equiv_X B \ \& \ B \equiv_X C \Rightarrow A \equiv_X C$
3. $A \equiv_X A$
4. $A \equiv_X B \Rightarrow B \equiv_X A$
5. $A \equiv_X B \ \& \ B \equiv_X C \Rightarrow A \equiv_X C$

Now we explore some axioms that are specifically concerned with the restriction variables of weak implication. As was to be expected, restriction variables can be harmlessly deleted:

Proposition 15 (Removal of restriction variables)

1. $A \equiv_{X \cup Y} B \Rightarrow A \equiv_X B$
2. $A \equiv_X B \Rightarrow A \equiv_{X \cup Y} B$

A variable can be added to the set of restriction variables X in $A \equiv_X B$ if it does not occur in B :

Proposition 16 (Addition of restriction variables)

1. $A \equiv_X B \Rightarrow A \equiv_{X \cup Y} B$ if $\text{var}(B) \cap Y \subset X$
2. $A \equiv_X B \Rightarrow A \equiv_{X \cup Y} B$ if $\text{var}(A \cup B) \cap Y \subset X$

For strong implication we have, among others, the following axioms:

Proposition 17 (Axioms for strong implication)

1. $A \cup B \equiv A$
2. $A \equiv B \ \& \ A \equiv C \Rightarrow A \equiv B \cup C$
3. $A \equiv B \Rightarrow A \cup C \equiv B \cup C$
4. $A \equiv B \cup C \Rightarrow A \equiv B$
5. $A \equiv B \ \& \ B \cup C \equiv D \Rightarrow A \cup C \equiv D$
6. $A \equiv B \cup C \ \& \ B \equiv D \Rightarrow A \equiv C \cup D$
7. $A \equiv B \ \& \ C \equiv D \Rightarrow A \cup C \equiv B \cup D$

The axioms (1) and (2) are known in Functional Dependency Theory [Vardi, 1988] as *decomposition* and *union* respectively. From these axioms and transitivity (axiom (2) of Proposition 14) all other strong axioms (3)-(7) can be derived. For weak implication however the situation is different, since only (1) and (4) hold unconditionally:

Proposition 18 (Axioms for weak implication)

1. $A \cup B \equiv_X A$
2. $A \equiv_X B \ \& \ A \equiv_X C \Rightarrow A \equiv_X B \cup C$
if $\text{var}(B) \cap \text{var}(C) \subset X$
3. $A \equiv_X B \Rightarrow A \cup C \equiv_X B \cup C$
if $\text{var}(B) \cap \text{var}(C) \subset X$
4. $A \equiv_X B \cup C \Rightarrow A \equiv_X B$
5. $A \equiv_X B \ \& \ B \cup C \equiv_X D \Rightarrow A \cup C \equiv_X D$
if $\text{var}(B) \cap \text{var}(C) \subset X$
6. $A \equiv_X B \cup C \ \& \ B \equiv_X D \Rightarrow A \equiv_X C \cup D$
if $\text{var}(C) \cap \text{var}(D) \subset X$
7. $A \equiv_X B \ \& \ C \equiv_X D \Rightarrow A \cup C \equiv_X B \cup D$
if $\text{var}(B) \cap \text{var}(D) \subset X$

The conditional axioms of Proposition 18 are more general than their counterparts in Proposition 17 since from the conditional axioms the unconditional axioms can be derived. For instance axiom (2) in Proposition 17 can be derived from axiom (2) in Proposition 18 by assigning to X the set of all variables V in which case the condition becomes trivially true.

Name	Axiom	Restriction
True	$A \models_x \text{true}$	
False	$\text{false} \models_x A$	
Weakening	$A \cup B \models_x A$	
Transitivity	$A \models_x B \ \& \ B \models_x C \Rightarrow A \models_x C$	
Union	$A \models_x B \ \& \ A \models_x C \Rightarrow A \models_x B \cup C$	$\text{var}(B) \cap \text{var}(C) \subset X$
Substitution	$A \cup \Phi \models_x \Phi(A) \cup \Phi$	
Generalization	$\Phi(A) \cup \Phi \models_x A \cup \Phi$	
Abstraction	$\Phi(A) \models_x A \cup \Phi$	$h\nu(\Phi) \cap X = \emptyset$
Removal	$A \models_{X \cup Y} B \Rightarrow A \models_x B$	
Irrelevance	$A \models_x B \Rightarrow A \models_{X \cup Y} B$	$\text{var}(B) \cap Y \subset X$

Figure 3: Axioms for weak implication

6 Completeness

A natural question that arises in the context of an axiomatic approach is whether a sound and complete axiomatization is feasible. As a first step we proved the soundness of all axioms in Figure 3.

Proposition 10 *For general constraints, the axioms in Figure 3 are sound.*

Both for weak and strong implication, a completeness result can be derived. We proved completeness with respect to *primitive* constraints, that is constraints of the format 'x = y' or 'x = c':

Proposition 20 *For primitive constraints, the axioms in Figure 8 are sound and complete.*

For completeness the axioms concerned with substitution have to be added to the ones we gave before. The most interesting axiom is Abstraction that allows you to add solutions for intermediate variables.

From the complete set for weak implication a complete set of strong axioms can be easily inferred. Abstraction is never applicable for strong implication so it is replaced by the derived axiom Instantiation:

$$A \models B \Rightarrow \Phi(A) \models \Phi(B)$$

7 Constraint elimination

Intuitively more constraints can be eliminated if weak equivalence is used instead of strong equivalence. In this section we provide a proposition that specifically exploits weak equivalence for constraint elimination. A subset B of a set of constraints $A \cup B$ is redundant with respect to the restriction variables if there exists a solution set Φ for intermediate variables such that $\Phi(B)$, the effect of substituting Φ on B , is equivalent with true:

Proposition 21

$A \cup B \models_x A$ if there exists some Φ such that:

1. $\Phi(B) \equiv \text{true}$
2. $h\nu(\Phi) \cap X = \emptyset$
3. $h\nu(\Phi) \cap \text{var}(A) = \emptyset$

Proof: i) $A \cup B \models_x A$: Weakening.

ii) $A \models_x A \cup B$:

$$A \models_x A \Rightarrow A \models_x \Phi(A) \dots \dots \dots (3)$$

$$\begin{aligned} &\Rightarrow A \models_x \Phi(A) \cup \text{true} \\ &\Rightarrow A \models_x \Phi(A) \cup \Phi(B) \dots \dots \dots (1) \\ &\Rightarrow A \models_x \Phi(A \cup B) \\ &\Rightarrow A \models_x A \cup B \cup \Phi \dots \dots \dots (2, \text{Abstraction}) \\ &\Rightarrow A \models_x A \cup B \dots \dots \dots (\text{Weakening}) \end{aligned}$$

Example 2 (Constraint elimination)

$$\begin{aligned} \{x = y + 2, y = z + 2\} &\equiv_{x,y} \{x = y + 2\} \\ &\text{with } \Phi = \{z = y - 2\} \\ \{u > x, x > v, u > v\} &\equiv_{u,v} \{u > v\} \\ &\text{with } \Phi = \{x = (u + v)/2\} \\ \{v = u + 3, x > u\} &\equiv_{u,v} \{v = u + 3\} \\ &\text{with } \Phi = \{x = u + 1\} \\ \{v = u + 3, x > u\} &\not\equiv_{u,v,x} \{v = u + 3\} \\ &\text{by (2) } x \text{ may not be substituted} \\ \{x + y = 0, y > 0\} &\not\equiv_x \{x + y = 0\} \\ &\text{by (3) } \Phi = \{y = 1\} \text{ is illegal} \end{aligned}$$

The proof of the proposition is added to illustrate the convenience of reasoning by means of weak equivalence.

8 An elaborated example

As an illustrative example of application of weak equivalence consider the circuit of Figure 4 derived from the domain of electronics. The corresponding RL/1 representation is listed in Figure 5. In ancient history the bridge of Wheatstone was used to measure ohm resistance. In the setup, r_1 (or r_2) is a variable resistor and r_4 (or r_3) the unknown resistor. The component in the center is an ampere meter with resistance r_5 . For measurement the variable resistor is adjusted so that no current flows through r_5 . In this balanced state the following relationship holds:

$$r_1/r_2 = r_3/r_4$$

To see if theory matches reality we apply 10 volt at the clamps of the circuit. In case we choose the values of the resistors all equal to 200 ohm then the circuit is in balanced state and the RL/1 knowledge base can compute the values for v_b , v_c and i_5 :

```
show circ(10,vb,vc,0,15,200,200,200,200,200)
vb      vc      i5
5.00000 5.00000 0
```

To obtain this answer, the solver infers the following weak equivalence:

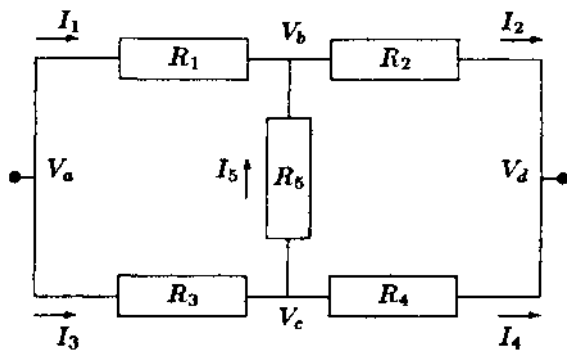


Figure 4: Circuit overview of the bridge of Wheatstone

```

module wheatstone(number:x:v:i:r:
  va:vb:vc:vd:ve:
  i1:i2:i3:i4:i5:r1:r2:r3:r4:r5).
clause ohmlaw(v,i,r) when v=i*r.
clause circ(va,vb,vc,vd,i5,r1,r2,r3,r4,r5)
when ohmlaw(va-vb,i1,r1)
  and ohmlaw(va-vc,i3,r3)
  and ohmlaw(vc-vd,i4,r4)
  and ohmlaw(vb-vd,i2,r2)
  and ohmlaw(vc-vb,i5,r5)
  and i1+i5=i2 and i4+i5=i3.

```

Figure 5: The bridge of Wheatstone

$$\{v_a - v_b = i_1 * r_1, v_a - v_c = i_3 * r_3, v_c - v_d = i_4 * r_4, \\ v_b - v_d = i_2 * r_2, v_c - v_b = i_5 * r_5, \\ i_1 + i_5 = i_2, i_4 + i_5 = i_3, \\ r_1 = 200, r_2 = 200, r_3 = 200, r_4 = 200, r_5 = 200, \\ v_a = 10, v_d = 0, \} \equiv_{\{v_b, v_c, i_5\}} \{v_b = 5, v_c = 5, i_5 = 0\}$$

If one of the resistors, say r_1 , is taken larger than the others, a small positive current flows through the center resistor:

```

show circ(10,vb,vc,0,15,400,200,200,200,200)
vb      vc      i5
-----
3.84615 4.61538 0.00384

```

It might be quite interesting to see if the relationship for r_4 can be inferred symbolically from the represented knowledge under the assumption that the circuit is in balanced state. We want to express r_4 in terms of r_1 , r_2 and r_3 so these latter variables are specified as known variables. As an extra constraint we know that since the circuit is in balanced state no current flows through r_5 :

```

syminfer(r4) when
  circ(*) and i5=0 and known(r1,r2,r3).
solution= r4=r2*r3/r1

```

To obtain this answer, the solver had to infer the following weak equivalence:

$$\{v_a - v_b = i_1 * r_1, v_a - v_c = i_3 * r_3, v_c - v_d = i_4 * r_4, \\ v_b - v_d = i_2 * r_2, v_c - v_b = i_5 * r_5, \\ i_1 + i_5 = i_2, i_4 + i_5 = i_3, \\ i_5 = 0\} \equiv_{\{r_1, r_2, r_3, r_4\}} \{r_4 = r_2 * r_3 / r_1\}$$

These equations are nonlinear by the occurrences of the terms $i_1 * r_1$, $i_2 * r_2$, $i_3 * r_3$, $i_4 * r_4$ and $i_5 * r_5$ (note that the known variables r_1 , r_2 and r_3 are in fact not given as a value and therefore contribute to the nonlinearity of the constraint system).

9 Conclusion

In this paper we used weak equivalence to express, explicitly, the functional difference between wanted, known and intermediate variables in a set of constraints. By a complete set of axioms we established the main properties of weak equivalence. Using these axioms we proved an application independent tool to simplify constraint sets using this new form of equivalence.

With examples run on a prototype implementation we showed practical applicability of weak equivalence for constraint solving purposes. By distinguishing wanted, known and intermediate variables in a constraint set the solver can be guided towards a solution that discards intermediate variables and expresses wanted variables in terms of known variables. Derivation of symbolic relationships is facilitated by the theoretical concepts introduced in this paper in two ways. Firstly, weak equivalence allows you to express formally the correspondence between the derived relationship and the original constraint set. Secondly the reasoning necessary for inference of symbolic relationships can be represented in terms of weak equivalence.

References

- [Denneheuvel, 1990] S. van Denneheuvel & P. van Emde Boas, *The rule language RL/1*, A. M. Tjoa & R. Wagner (eds.), Database and Expert Systems Applications, Proc. of the Int. Conf., Vienna, Austria, Aug, Springer Verlag Wien, 381-387.
- [Jaffar and Michaylov, 1987] J. Jaffar & S. Michaylov, *Methodology and Implementation of a CLP System*, Logic Programming, Proc. of the Fourth Int. Conf., Ed. Jean-Louis Lassez, 196-218.
- [Kwast, 1991] K. L. Kwast, *The incomplete database*, IJCAI'91, Morgan Kaufmann Publishers.
- [Leler, 1988] Wm. Leler, *Constraint Programming Languages: Their Specification and Generation*, Addison-Wesley Series in Computer Science.
- [Vardi, 1988] M. Y. Vardi, *Fundamentals of Dependency Theory*, In: Trends in Theoretical Computer Science, Egon Borger (ed.), Computer Science Press, 171-225.
- [Van Emde Boas, 1986a] P. van Emde Boas, *RL, a Language for Enhanced Rule Bases Database Processing*, Working Document, Rep IBM Research, RJ 4869 (51299).
- [Van Emde Boas, 1986b] P. van Emde Boas, *A semantical model for the integration and modularization of rules*, Proc. MFCS 12, Bratislava, Springer LNCS 233, 78-92.
- [Wolfram, 1988] S. Wolfram, *Mathematica, A System for Doing Mathematics by Computer*, Addison-Wesley.