XML Representation of Constraint Networks
Version 2.0

Organising Committee of the
Second International Competition of CSP Solvers

Abstract

In this document, we present a new extended format to represent constraint networks using XML. This format allows representing constraints defined either in extension or in intention. It also allows to reference global constraints. It will be used for the second international competition of CSP solvers which will be held during summer and winter 2006.

Revisions of this document

May 23, 2006

Two new attributes (*maxConstraintArity* and *maxSatisfiableConstraints*) of element <presentation> have been introduced. They are optional (and will not be considered for the 2006 competition).

Impact on the 2006 competition: none as such attributes will be removed from instances under canonical form.

August 2, 2006

The use of operators *eq* and *ne* with Boolean arguments is deprecated (see subsection 2.5). Instead of using *ne*, you can use *xor* and instead of using *eq*, you can use *not* with *xor*. Certainly, new operators (iff, if, ...) will be introduced in the future. The interest of removing (in the medium term) the use of *eq* and *ne* with Boolean arguments is that it simplifies typing control of expressions and parsing.

Impact on the 2006 competition: none as instances involving predicates which use operators *eq* and *ne* with Boolean arguments will be discarded.

1 Introduction

For the second international competition of CSP solvers which will be held during summer 2006, it has been decided to deal with constraint networks involving finite domains and constraints defined in extension or in intention. It

means that, in such networks, domains (associated with variables) correspond to finite sets of values, and constraints are either explicitly defined by sets of tuples, or implicitly defined by predicates.

In order to avoid any ambiguity, we briefly introduce constraint networks. A constraint network consists of a finite set of variables such that each variable X has an associated domain $\mathrm{dom}(X)$ denoting the set of values allowed for X, and a finite set of constraints such that each constraint C has an associated relation $\mathrm{rel}(C)$ denoting the set of tuples allowed for the variables $\mathrm{vars}(C)$ involved in C. A solution to a constraint network is an assignment of values to all the variables such that all the constraints are satisfied. A constraint network is said to be satisfiable if it admits at least a solution. The Constraint Satisfaction Problem (CSP), whose task is to determine whether or not a given constraint network is satisfiable, is NP-complete. A constraint network is also called CSP instance.

To propose an XML representation of constraint networks, one has to address the main issue of representing constraints. When a constraint is given in intention, a predicate has to be introduced and when a constraint is given in extension, a set of tuples has to be introduced. More precisely, constraints can be represented by giving:

- a predicate that determines whether a given tuple is allowed or not,
- or a set of allowed tuples, called supports,
- or a set of unallowed (or forbidden) tuples, called conflicts.

Important Remark 1 *The description given by this document exploits both BNF (Backus Naur Form) and XML structures, and may sometimes appear as approximative. However, note that the objective of this document is to be as precise as possible while preserving human-readability. For a more rigourous description, one can see the DTD (Document Type Definition) or XML schema (available in April, 2006) that is associated with format 2.0. Also, a tool (called checker) that will validate instances (it is not possible to perform all verifications using a DTD or a schema) in format 2.0 will be furnished.*

Important Remark 2 *The format proposed in this document represents a compromise between structuration and readibility. However, we keep some freedom by taking into account several representations of predicate expressions.*

Important Remark 3 *This format is accompanied by :*

- *a XML schema,*
- *three parsers, one written in C++ (which provides an interface suitable for solvers written in C) and the two other ones written in Java (using DOM and SAX, respectively),*
- *a tool called* `solutionChecker` *that allows to compute the number of constraints violated by a full variable assignment (of course, a number of violated constraints equal to* 0 *corresponds to a solution),*

- *a tool called* `instanceChecker` *that allows to validate instances and to convert constraints defined in intention into constraints defined in extension.*
- *a tool called* `instanceShuffler` *that allows to shuffle variables and constraints of a given instance.*

For more information about such tools, see `http://www.cril.univ-artois.fr/~lecoutre/research/tools/tools.html`

2 XML Representation

Each CSP instance is represented following the format given in Figure 1 where q, n, r, p and e respectively denote the number of distinct domains, the number of variables, the number of distinct relations, the number of distinct predicates and the number of constraints. Note that $q \leq n$ as the same domain definition can be used for different variables, $r \leq e$ and $p \leq e$ as the same relation or predicate definition can be used for different constraints. Thus, each instance is defined by an XML element which is called *instance* and which contains four, five or six elements. Indeed, it is possible to have one instance defined without any reference to a relation or/and to a predicate. Then, the elements <relations> and <predicates> may be missing (in this case, it means that only global constraints are referenced).

Each basic element (<presentation>, <domain>, <variable>, <relation>, <predicate> and <constraint>) of the representation admits an attribute called *name*. The value of the attribute *name* must be a valid identifier according to the most common rules (start with a letter or underscore and further contain letters, digits or underscores). More precisely, an identifier is defined (in BNF notation) as follows:

```
<identifier> ::= <letter> | "_" { <letter> | <digit> | "_" }
<letter> ::= "a".."z" | "A".."Z"
<digit> ::= "0".."9"
```

Identifiers are case-sensitive. Useful in the rest of the document, separators and integers are defined as follows:

```
<whitespace> ::= " " | "\t" | "\n" | "\r"
<separator> ::= <whitespace> | { <whitespace> }
<integer>  ::= [ "+" | "-"]  <digit> {<digit>}
```

Remark 1 *The format 2.0 described in this paper is not an extension of the format 1.1 proposed for the first competition of CSP solvers. It means that an instance represented using the format 1.1 does not respect the new format 2.0.*

Remark 2 *In the representation of any instance, it is not possible to find several attributes "name" using the same identifier.*

Remark 3 *In the body of any element of the document, one can insert an <extension> element in order to put any information specific to a solver.*

```
<instance>
   <presentation
       name = 'put here the instance name'
       ...
       format = 'XCSP 2.0' >
     Put here the description of the instance
   </presentation>

   <domains nbDomains='q'>
      <domain
         name = 'put here the domain name'
         nbValues = 'put here the number of values' >
        Put here the list of values
      </domain>
      ...
   </domains>

   <variables nbVariables='n'>
      <variable
         name = 'put here the variable name'
         domain = 'put here the name of a domain'
      />
      ...
   </variables>

   <relations nbRelations='r'>
       <relation
           name = 'put here the name of the relation'
           arity = 'put here the arity of the relation'
           nbTuples = 'put here the number of tuples'
           semantics = 'put here either supports or conflicts' >
         Put here the list of tuples
       </relation>
        ...
   </relations>

   <predicates nbPredicates='p'>
       <predicate
           name = 'put here the name of the predicate' >
         <parameters>
           put here a list of formal parameters
         </parameters>
         <expression>
            Put here one (or more) representation of the predicate expression
         </expression>
       </predicate>
       ...
   </predicates>

   <constraints nbConstraints='e'>
      <constraint
         name = 'put here the name of the constraint'
         arity = 'put here the arity of the constraint'
         scope = 'put here the scope of the constraint'
         reference = 'put here the name of a relation, a
                      predicate or a global constraint'>
         ...
      </constraint>
      ...
   </constraints>
</instance>
```

Figure 1: XML representation of a CSP instance

2.1 Presentation

The XML element called <presentation> admits a set of attributes and may contain a description (a string) of the instance :

```
<presentation
    name = 'put here the instance name'
    maxConstraintArity = 'put here the greatest constraint arity'
    nbSolutions = 'put here the number of solutions'
    solution = 'put here a solution'
    maxSatisfiableConstraints = 'the maximum number of satisfied constraints'
    format = 'XCSP 2.0' >
  Put here the description of the instance
</presentation>
```

The attribute *name* must be a valid identifier while the attribute $format$ must be given the value 'XCSP 2.0'. All other attributes of <description> are optional as they only provide human-readable information. The attribute $maxConstraintArity$ is of type integer and denotes the greatest arity of all constraints involved in the instance. The attribute *nbSolutions* can be given an integer value denoting the total number of solutions of the instance, an expression of the form 'at least k' with k being a positive integer or '?'.

For example,

- `nbSolutions = '0'` indicates that the instance is unsatisfiable,
- `nbSolutions = '3'` indicates that the instance has exactly 3 solutions,
- `nbSolutions = 'at least 1'` indicates that the instance has at least 1 solution (and, hence, is satisfiable),
- `nbSolutions = '?'` indicates that it is unknown whether or not the instance is satisfiable,

The attribute *solution* indicates a solution if one exists and has been found. The attribute $maxSatisfiableConstraints$ can be given an integer value denoting the maximum number of constraints that can be satisfied, an expression of the form 'at least k' with k being a positive integer or '?'.

2.2 Domains

The XML element called *domains* admits an attribute which is called *nbDomains* and contains some occurrences (at least, one) of an element called *domain*, one for each domain associated with at least one variable of the instance. The attribute *nbDomains* is of type integer and its value is equal to the number of occurrences of the element *domain*. Each element *domain* admits two attributes, called *name* and $nbValues$ and contains a list of values, as follows:

```
<domain
    name = 'put here the domain name'
    nbValues = 'put here the number of values' >
  Put here the list of values
</domain>
```

The attribute *name* corresponds to the name of the domain and its value must be a valid identifier. The attribute $nbValues$ is of type integer and its value is equal to the number of values of the domain. The content of the element <domain> gives the list (set) of integer values included in the domain. The description of (the content of) a domain takes the form (in BNF notation):

```
<domainDescription> ::= <domainPiece> {<separator> <domainPiece>}
<domainPiece> ::= <integer> | <integer> ".." <integer>
```

To summarize, a domain is defined from some pieces that correspond to single values and ranges of integer values. For example,

- 1 5 10 corresponds to the set $\{1, 5, 10\}$.
- 1..3 10..14 corresponds to the set $\{1, 2, 3, 10, 11, 12, 13, 14\}$.

Note that $nbValues$ gives the number of values of the domain (i.e. the domain size), and not, the number of domain pieces. Note also that one (or more) space character is used as a separator of domain pieces.

2.3 Variables

The XML element called *variables* admits an attribute which is called $nbVariables$ and contains some occurrences of an element called *variable*, one for each variable of the instance. The attribute $nbVariables$ is of type integer and its value is equal to the number of occurrences of the element *variable*. Each element *variable* is empty but admits two attributes, called *name* and *domain*, as follows:

```
<variable
   name = 'put here the variable name'
   domain = 'put here the name of a domain'
/>
```

The attribute *name* corresponds to the name of the variable and its value must be a valid identifier. The value of the attribute *domain* gives the name of the associated domain. It must correspond to the value of the *name* attribute of a *domain* element.

2.4 Relations

If present, the XML element called *relations* admits an attribute which is called *nbRelations* and contains some occurrences (at least, one) of an element called *relation*, one for each relation associated with at least a constraint of the instance. The attribute *nbRelations* is of type integer and its value is equal to the number of occurrences of the element *relation*.

Each element *relation* admits four attributes, called *name*, *arity*, $nbTuples$ and *semantics*, and contains a list of tuples that represents either allowed tuples (supports) or unallowed tuples (conflicts). It is defined as follows:

```
<relation
    name = 'put here the name of the relation'
    arity = 'put here the arity of the relation'
    nbTuples = 'put here the number of tuples'
    semantics = 'put here either supports or conflicts' >
  Put here the list of tuples
</relation>
```

The attribute *name* corresponds to the name of the relation and its value must be a valid identifier. The attribute *arity* is of type integer and its value is equal to the arity of the relation. The attribute *nbTuples* is of type integer and its value is equal to the number of tuples of the relation. The attribute *semantics* can only be given two values: 'supports' and 'conflicts'. Of course, if the value of *semantics* is 'supports' (resp. 'conflicts'), then it means that the list of tuples correspond to allowed (resp. unallowed) tuples. The content of the element <relation> gives the list (set) of tuples of the relation. For a binary relation, the description of (the content of) a relation takes the form (in BNF notation):

```
<binaryRelationDescription> ::= [<binaryTupleList>]
<binaryTupleList> ::= <binaryTuple> | <binaryTuple> "|" <binaryTupleList>
<binaryTuple> ::= <integer> <separator> <integer>
```

For ternary relations, one has just to consider tuples formed from 3 values, etc. For example, a list of binary tuples is:

```
0 1|0 3|1 2|1 3|2 0|2 1|3 1
```

while a list of ternary tuples is:

```
0 0 1|0 2 1|1 0 1|1 2 0|2 1 1|2 2 2
```

Note that an empty list of tuples is authorized by the syntax. Also, remark that one (or more) space character is used as separator of tuple values, and that '|' is used as separator of tuples.

Remark 4 *Strictly speaking, an element <relation> does not correspond to a well-defined relation. Indeed, the Cartesian product on which the relation should be defined is not precised. However, it allows to associate an element <relation> with constraints whose domains (i.e. Cartesian products corresponding to their scopes) are different.*

2.5 Predicates

If present, the XML element called *predicates* admits an attribute which is called *nbPredicates* and contains some occurrences (at least, one) of an element called *predicate*, one for each predicate associated with at least a constraint of the instance. The attribute *nbPredicates* is of type integer and its value is equal to the number of occurrences of the element *predicate*.

Each element *predicate* admits one attribute, called *name*, and contains two elements, called <parameters> and <expression>. It is defined as follows:

```
<predicate
    name = 'put here the name of the predicate' >
  <parameters>
     put here a list of formal parameters
  </parameters>
  <expression>
     Put here one (or several) representation(s) of the predicate expression
  </expression>
</predicate>
```

The attribute *name* corresponds to the name of the predicate and its value must be a valid identifier. The element <parameters> contains a list of formal parameters as follows (in BNF notation):

```
<formalParameters> ::= [<formalParametersList>]
<formalParametersList> ::= <formalParameter>
                         | <formalParameter> <separator> <formalParametersList>
<formalParameter> ::= <type> <separator> <identifier>
<type>  ::= "int"
```

Each parameter is then defined by a pair composed of its name and the name of its type. For the moment, the only authorized type is 'int' (denoting integer values). However, in the future, we project to take into account other types: "bool", "string", etc. Note that one (or more) space character is used between formal parameters and between the type and the name of a parameter.

The element <expression> contains one (or more) representation of the predicate expression.

Remark 5 *For the 2006 competition, only the functional representation described below will be considered.*

2.5.1 Functional Representation

It is possible to insert a functional representation of the predicate expression by inserting in <expression> an element <functional> which contains any boolean expression defined as follows:

```
<integerExpression> ::=
      <integer> | <identifier>
    | "neg(" <integerExpression> ")"
    | "abs(" <integerExpression> ")"
    | "add(" <integerExpression> "," <integerExpression> ")"
    | "sub(" <integerExpression> "," <integerExpression> ")"
    | "mul(" <integerExpression> "," <integerExpression> ")"
    | "div(" <integerExpression> "," <integerExpression> ")"
    | "mod(" <integerExpression> "," <integerExpression> ")"
    | "pow(" <integerExpression> "," <integerExpression> ")"
    | "min(" <integerExpression> "," <integerExpression> ")"
    | "max(" <integerExpression> "," <integerExpression> ")"
```

```
<booleanExpression> ::=
      "false" | "true"
    | "not(" <booleanExpression> ")"
    | "and(" <booleanExpression> "," <booleanExpression> ")"
    | "or(" <booleanExpression> "," <booleanExpression> ")"
    | "xor(" <booleanExpression> "," <booleanExpression> ")"
    | "eq(" <booleanExpression> "," <booleanExpression> ")"          <DEPRECATED>
    | "eq(" <integerExpression> "," <integerExpression> ")"
    | "ne(" <booleanExpression> "," <booleanExpression> ")"          <DEPRECATED>
    | "ne(" <integerExpression> "," <integerExpression> ")"
    | "ge(" <integerExpression> "," <integerExpression> ")"
    | "gt(" <integerExpression> "," <integerExpression> ")"
    | "le(" <integerExpression> "," <integerExpression> ")"
    | "lt(" <integerExpression> "," <integerExpression> ")"
```

Hence, any constraint in intention can be defined by a predicate which corresponds to an expression built from (boolean and integer) constants and the introduced set of functions (operators). The semantics of operators is given by Table 1. Overflows, divisions by zero and integer divisions are discussed in the document dealing with the format rules for the competition.

Operation	Arity	Syntax	Semantics	MathML
Arithmetic (operands are integers)				
Opposite	1	neg(x)	-x	<minus>
Absolute Value	1	abs(x)	| x |	<abs>
Addition	2	add(x,y)	x + y	<plus>
Substraction	2	sub(x,y)	x - y	<minus>
multiplication	2	mul(x,y)	x * y	<times>
Integer Division	2	div(x,y)	x div y	<quotient>
Remainder	2	mod(x,y)	x mod y	<rem>
Power	2	pow(x,y)	x^y	<power>
Minimum	2	min(x,y)	min(x,y)	<min>
Maximum	2	max(x,y)	max(x,y)	<max>
Relationnal (operands are integers)				
Equal to	2	eq(x,y)	$x = y$	<eq>
Different from	2	ne(x,y)	$x \neq y$	<neq>
Greater than or equal	2	ge(x,y)	$x \geq y$	<geq>
Greater than	2	gt(x,y)	$x > y$	<gt>
Less than or equal	2	le(x,y)	$x \leq y$	<leq>
Less than	2	lt(x,y)	$x < y$	<lt>
Logic (operands are Booleans)				
Logical not	1	not(x)	not x	<not>
Logical and	2	and(x,y)	x and y	<and>
Logical or	2	or(x,y)	x or y	<or>
Logical xor	2	xor(x,y)	x xor y	<xor>
Logical iff	2	iff(x,y)	x iff y	<iff>

Table 1: Operators used to build predicate expressions

Remark that variables can occur ; their names must correspond to the names of formal parameters. To illustrate this, let us consider the predicate that allows

defining constraints involved in any instance of the queens problem. It corresponds to: $X \neq Y \wedge |X-Y| \neq Z$. We obtain using the functional representation:

```
<predicate name="P0">
  <parameters>
    int X int Y int Z
  </parameters>
  <expression>
    <functional>
      and(ne(X,Y),ne(abs(sub(X,Y)),Z))
    </functional>
  </expression>
</predicate>
```

2.5.2 MathML Representation

MathML is a language dedicated to represent mathematical expressions. We can represent predicate expressions using a subset of this language. It is possible to insert an XML representation of the predicate expression by inserting in <expression> an element called <math> which contains any boolean expression defined (in BNF notation) as follows:

```
<integerExpression> ::=
     "<cn>" <integer> "</cn>"
   | "<ci>" <identifier> "</ci>"
   | "<apply> <minus/>"  <integerExpression> "</apply>"
   | "<apply> <abs/>"  <integerExpression> "</apply>"
   | "<apply> <plus/>" <integerExpression> <integerExpression> "</apply>"
   | "<apply> <minus/>" <integerExpression> <integerExpression> "</apply>"
   | "<apply> <times/>" <integerExpression> <integerExpression> "</apply>"
   | "<apply> <quotient/>" <integerExpression> <integerExpression> "</apply>"
   | "<apply> <rem/>" <integerExpression> <integerExpression> "</apply>"
   | "<apply> <power/>" <integerExpression> <integerExpression> "</apply>"
   | "<apply> <min/>" <integerExpression> <integerExpression> "</apply>"
   | "<apply> <max/>" <integerExpression> <integerExpression> "</apply>"

<booleanExpression> ::=
     "<false/>"
   | "<true/>"
   | "<apply> <not/>" <booleanExpression> "</apply>"
   | "<apply> <and/>" <booleanExpression> <booleanExpression> "</apply>"
   | "<apply> <or/> <booleanExpression> <booleanExpression> "</apply>"
   | "<apply> <xor/> <booleanExpression> <booleanExpression> "</apply>"
   | "<apply> <eq/>" <integerExpression> <integerExpression> "</apply>"
   | "<apply> <neq/>" <integerExpression> <integerExpression> "</apply>"
   | "<apply> <gt/>" <integerExpression> <integerExpression> "</apply>"
   | "<apply> <geq/>" <integerExpression> <integerExpression> "</apply>"
   | "<apply> <lt/>" <integerExpression> <integerExpression> "</apply>"
   | "<apply> <leq/>" <integerExpression> <integerExpression> "</apply>"
```

For more information, see `http://www.w3.org/Math` or `http://www.dessci.com/en/support/tutorials/mathml/content.htm`. For example, to represent $X \neq Y \wedge |X - Y| \neq Z$, we can write:

```
<predicate name="P0">
  <parameters> int X int Y int Z </parameters>
  <expression>
    <math>
      <apply>
         <neq/> <ci> X </ci> <ci> Y </ci>
      </apply>
      <apply>
         <neq/>
         <apply>
           <abs/>
           <apply> <minus/ > <ci> X </ci> <ci> Y </ci> </apply>
         </apply>
         <ci> Z </ci>
      </apply>
    </math>
  </expression>
</predicate>
```

2.5.3 Postfix Representation

It is possible to insert a postfix representation (not fully described in this document) of the predicate expression by inserting in <expression> an element <postfix>. For example, to represent $X \neq Y \wedge |X - Y| \neq Z$, we can write:

```
<predicate name="P0">
  <parameters>
    int X int Y int Z
  </parameters>
  <expression>
    <postfix>
      X Y ne X Y sub abs Z ne and
    </postfix>
  </expression>
</predicate>
```

2.5.4 Infix Representation

It is possible to insert an infix representation (not fully described in this document) of the predicate expression by inserting in <expression> an element <infix>. For example, to represent $X \neq Y \wedge |X - Y| \neq Z$, we can write (using a C syntax for the boolean expression):

```
<predicate name="P0">
  <parameters>
    int X int Y int Z
  </parameters>
  <expression>
    <infix syntax="C">
      X != Y && abs(X-Y) != Z
    </infix>
  </expression>
</predicate>
```

2.6 Constraints

The XML element called <constraints> admits an attribute which is called *nbConstraints* and contains some occurrences (at least, one) of an element called <constraint>, one for each constraint of the instance. The attribute *nbConstraints* is of type integer and its value is equal to the number of occurrences of the element <constraint>.

Each element <constraint> admits four attributes, called *name*, *arity*, *scope* and *reference*, and potentially contains some elements:

```
<constraint
   name = 'put here the name of the constraint'
   arity = 'put here the arity of the constraint'
   scope = 'put here the scope of the constraint'
   reference = 'put here the name of a relation, of
               predicate or a global constraint'>
   ...
</constraint>
```

The attribute *name* corresponds to the name of the constraint and its value must be a valid identifier. The attribute *arity* is of type integer and its value is equal to the arity of the constraint (that is to say, the number of variables in its scope). The value of the attribute *scope* denotes the set of variables involved in the constraint. It must correspond to a list of variable names where each name corresponds to the value of the *name* attribute of a *variable* element. One (or more) space character is used as a separator.

There are three alternatives to represent constraints. Indeed, it is possible to introduce:

- constraints in extension
- constraints in intention
- global constraints

2.6.1 Constraints in extension

The value of the attribute *reference* must be the name of a relation. It means that it must correspond the value of the *name* attribute of a <relation> element.

The element <constraint> is empty when it represents a constraint defined in extension.

For example,

```
<constraint name="C0" scope="X0 X1" reference="rel0" />
```

2.6.2 Constraints in intention

The value of the attribute *reference* must be the name of a predicate. It means that it must correspond to the value of the *name* attribute of a <predicate> element.

The element <constraint> contains an element <parameters> when it represents a constraint defined in intention. The element <parameters> contains a list of effective parameters which corresponds to either integers or names of variables (which must occur in the scope of the constraint). One (or more) space character is used as a separator.

For example:

```
<constraint name="C0" scope="X0 X1" reference="P0">
   <parameters>
      X0 X1 1
   </parameters>
</constraint>
```

The semantics is the following. Given a tuple built by assigning a value to each variable belonging to the scope of the constraint, the predicate expression is evaluated after replacing each occurrence of a formal parameter corresponding to an effective parameter denoting a variable with the assigned value. The tuple is allowed iff the expression evaluates to *true*.

Note that it is possible for an effective parameter to be any kind of expressions. However, it will not be considered for the 2006 competition.

2.6.3 Global constraints

The value of the attribute $reference$ must be the name of a global constraint, prefixed by "global:". As the character ':' cannot occur in any valid identifier, it avoids some potential collision with other identifiers.

The element <constraint> may contain an element <parameters> when it represents a global constraint. If present, the element <parameters> contains a sequence of parameters specific to the global constraint. As a consequence, the description of such parameters must be given for each global constraint. It is then clear that, for each global constraint, we have to indicate in a separate document, its name (the one to be referenced), its parameters (and the way they are structured in XML) and its semantics. Below, we provide such information for two global constraints.

Constraint allDifferent

Semantics all variables must take different values.

Parameters none

Example

```
<constraint name="C0" scope="X0 X1 X2 X3" reference="global:allDifferent" />
```

Alternatives This constraint can be represented in intention by introducing a predicate that represents a conjunction of inequalities. It can also be converted into a clique of binary notEqual constraints.

References For some information, e.g. see [2, 4, 1].

Constraint weightedSum

Semantics $\sum_{i=1}^{k} k_i * X_i \ op \ k_{i+1}$ where k_i denotes an integer, X_i the i^{th} variable occurring in the scope of the constraint, *op* a relational operator in $\{=, \neq, >, \geq, <, \leq\}$, and k_{i+1} an integer.

Parameters There is a first parameter (an element <list>) containing k pairs composed of one integer (coefficient) and one variable identifier. Space is used as a separator between pairs and between integers and variable identifiers. All coefficients must be non null and all variable identifiers must be different and must occur in the scope of the constraint. There is a second parameter (an element *op*) containing a string denoting the relational operator. This string must belong to $\{eq, ne, ge, gt, le, lt\}$ (see Table 1). There is a third parameter which is an integer (and which is not enclosed in any element).

Example $X0 + 2X1 - 3X2 > 12$

```
<constraint name="C2" scope="X0 X1 X2" reference="global:weightedSum">
  <parameters>
     <list> 1 X0 2 X1 -3 X2 </list>
     <op> gt </op>
     12
  </parameters>
</constraint>
```

Alternatives This arithmetic constraint can be represented in intention (using the grammar described earlier in the paper). It is interesting to note that:

- $\sum_{i=1}^{k} k_i * X_i = k_{i+1} \Leftrightarrow \sum_{i=1}^{k} k_i * X_i \geq k_{i+1} \wedge \sum_{i=1}^{k} k_i * X_i \leq k_{i+1}$
- $\sum_{i=1}^{k} k_i * X_i \neq k_{i+1} \Leftrightarrow \sum_{i=1}^{k} k_i * X_i > k_{i+1} \vee \sum_{i=1}^{k} k_i * X_i < k_{i+1}$
- $\sum_{i=1}^{k} k_i * X_i > k_{i+1} \Leftrightarrow \sum_{i=1}^{k} k_i * X_i \geq k_{i+1} - 1$
- $\sum_{i=1}^{k} k_i * X_i < k_{i+1} \Leftrightarrow \sum_{i=1}^{k} -k_i * X_i > -k_{i+1}$

References This arithmetic constraint is related to the constraint called sum_ctr in [1]. Some information can also be found in [3].

3 Validity of Instances

See Section 6 of the following document: `http://www.cril.univ-artois.fr/~lecoutre/research/tools/tools.pdf`.

4 Some examples

In Figures 2 and 3, one can see the XML representation of the 4 queens instance. In Figures 4, 5 and 6, one can see the XML representation of a CSP instance involving 5 variables and 5 constraints.

- $C0$: $X0 \neq X1$
- $C1$: $X3 - X0 \geq 2$
- $C2$: $X2 - X0 = 2$
- $C3$: $X1 + 2 = |X2 - X3|$
- $C4$: $X1 \neq X4$

Finally, in Figures 7 and 8, one can see the XML representation of the 3 magic square instance. The global constraints $weightedSum$ and $allDifferent$ are used.

References

[1] N. Beldiceanu, M. Carlsson, and J. Rampon. Global constraint catalog. Technical Report T2005-08, Swedish Institute of Computer Science, 2005.

[2] J.C. Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of AAAI'94*, pages 362–367, 1994.

[3] J.C. Régin and M. Rueher. A global constraint combining a sum constraint and difference constraints. In *Proceedings of CP'00*, pages 384–395, 2000.

[4] W.J. van Hoeve. The alldifferent constraint: a survey. In *Proceedings of the Sixth Annual Workshop of the ERCIM Working Group on Constraints*, 2001.

```
<instance>
  <presentation  name="Queens"  nbSolutions="at least 1"  format="XCSP 2.0">
     This is the 4-queens instance represnted in extension.
  </presentation>

  <domains nbDomains="1">
    <domain name="dom0" nbValues="4">
      1..4
    </domain>
  </domains>

  <variables nbVariables="4">
    <variable name="X0" domain="dom0"/>
    <variable name="X1" domain="dom0"/>
    <variable name="X2" domain="dom0"/>
    <variable name="X3" domain="dom0"/>
  </variables>

  <relations nbRelations="3">
    <relation name="rel0" arity="2" nbTuples="10" semantics="conflicts">
       1 1|1 2|2 1|2 2|2 3|3 2|3 3|3 4|4 3|4 4
    </relation>
    <relation name="rel1" arity="2" nbTuples="8" semantics="conflicts">
       1 1|1 3|2 2|2 4|3 1|3 3|4 2|4 4
    </relation>
    <relation name="rel2" arity="2" nbTuples="6" semantics="conflicts">
       1 1|1 4|2 2|3 3|4 1|4 4
    </relation>
  </relations>

  <constraints nbConstraints="6">
    <constraint name="C0" arity="2" scope="X0 X1" reference="rel0"/>
    <constraint name="C1" arity="2" scope="X0 X2" reference="rel1"/>
    <constraint name="C2" arity="2" scope="X0 X3" reference="rel2"/>
    <constraint name="C3" arity="2" scope="X1 X2" reference="rel0"/>
    <constraint name="C4" arity="2" scope="X1 X3" reference="rel1"/>
    <constraint name="C5" arity="2" scope="X2 X3" reference="rel0"/>
  </constraints>
</instance>
```

Figure 2: The 4-queens instance in extension

```
<instance>
   <presentation  name="Queens"  nbSolutions="at least 1"  format="XCSP 2.0">
     This is the 4-queens instance represented in intention.
  </presentation>

  <domains nbDomains="1">
    <domain name="dom0" nbValues="4">
       1..4
    </domain>
  </domains>

  <variables nbVariables="4">
    <variable name="X0" domain="dom0"/>
    <variable name="X1" domain="dom0"/>
    <variable name="X2" domain="dom0"/>
    <variable name="X3" domain="dom0"/>
  </variables>

  <predicates nbPredicates="1">
    <predicate name="P0">
      <parameters> int X int Y int Z  </parameters>
      <expression>
        <functional> and(ne(X,Y),ne(abs(sub(X,Y)),Z)) </functional>
      </expression>
    </predicate>
  </predicates>

  <constraints nbConstraints="6">
    <constraint name="C0" arity="2" scope="X0 X1" reference="P0">
      <parameters> X0 X1 1 </parameters>
    </constraint>
    <constraint name="C1" arity="2" scope="X0 X2" reference="P0">
      <parameters> X0 X2 2 </parameters>
    </constraint>
    <constraint name="C2" arity="2" scope="X0 X3" reference="P0">
      <parameters> X0 X3 2 </parameters>
    </constraint>
    <constraint name="C3" arity="2" scope="X1 X2" reference="P0">
      <parameters> X1 X2 1 </parameters>
    </constraint>
    <constraint name="C4" arity="2" scope="X1 X3" reference="P0">
      <parameters> X1 X3 2 </parameters>
    </constraint>
    <constraint name="C5" arity="2" scope="X2 X3" reference="P0">
      <parameters> X2 X3 1 </parameters>
    </constraint>
  </constraints>
</instance>
```

Figure 3: The 4-queens instance in intention

```
<instance>
  <presentation  name="Test"  format="XCSP 2.0">
     This is another instance represented in extension.
  </presentation>

  <domains nbDomains="3">
    <domain name="dom0" nbValues="7">
       0..6
    </domain>
    <domain name="dom1" nbValues="3">
      1 5 10
    </domain>
    <domain name="dom2" nbValues="10">
      1..5 11..15
    </domain>
  </domains>

  <variables nbVariables="5">
    <variable name="X0" domain="dom0"/>
    <variable name="X1" domain="dom0"/>
    <variable name="X2" domain="dom1"/>
    <variable name="X3" domain="dom2"/>
    <variable name="X4" domain="dom0"/>
  </variables>

  <relations nbRelations="4">
    <relation name="rel0" arity="2" nbTuples="7" semantics="conflicts">
       0 0|1 1|2 2|3 3|4 4|5 5|6 6
    </relation>
    <relation name="rel1" arity="2" nbTuples="25" semantics="conflicts">
       1 0|1 1|1 2|1 3|1 4|1 5|1 6|2 1|2 2|2 3|2 4|2 5|2 6|3 2|3 3|
       3 4|3 5|3 6|4 3|4 4|4 5|4 6|5 4|5 5|5 6
    </relation>
    <relation name="rel2" arity="2" nbTuples="1" semantics="supports">
       5 3
    </relation>
    <relation name="rel3" arity="3" nbTuples="17" semantics="supports">
       0 1 3|0 5 3|0 10 12|1 1 4|1 5 2|1 10 13|2 1 5|2 5 1|2 10 14|
       3 10 5|3 10 15|4 5 11|4 10 4|5 5 12|5 10 3|6 5 13|6 10 2
    </relation>
  </relations>

  <constraints nbConstraints="5">
    <constraint name="C0" arity="2" scope="X0 X1" reference="rel0"/>
    <constraint name="C1" arity="2" scope="X3 X0" reference="rel1"/>
    <constraint name="C2" arity="2" scope="X2 X0" reference="rel2"/>
    <constraint name="C3" arity="3" scope="X1 X2 X3" reference="rel3"/>
    <constraint name="C4" arity="2" scope="X1 X4" reference="rel0"/>
  </constraints>
</instance>
```

Figure 4: Test Instance in extension

```
<instance>
   <presentation  name="Test"  format="XCSP 2.0">
     This is another instance represented in intention.
  </presentation>

  <domains nbDomains="3">
    <domain name="dom0" nbValues="7">
       0..6
    </domain>
    <domain name="dom1" nbValues="3">
      1 5 10
    </domain>
    <domain name="dom2" nbValues="10">
      1..5 11..15
    </domain>
  </domains>

  <variables nbVariables="5">
    <variable name="X0" domain="dom0"/>
    <variable name="X1" domain="dom0"/>
    <variable name="X2" domain="dom1"/>
    <variable name="X3" domain="dom2"/>
    <variable name="X4" domain="dom0"/>
  </variables>

  <predicates nbPredicates="4">
    <predicate name="P0">
      <parameters> int X int Y </parameters>
      <expression>
        <functional> ne(X,Y) </functional>
      </expression>
    </predicate>
    <predicate name="P1">
      <parameters> int X int Y int Z </parameters>
      <expression>
        <functional> ge(sub(X,Y),Z) </functional>
      </expression>
    </predicate>
    <predicate name="P2">
      <parameters> int X int Y int Z </parameters>
      <expression>
        <functional> eq(sub(X,Y),Z) </functional>
      </expression>
    </predicate>
    <predicate name="P3">
      <parameters> int X int Y int Z int T </parameters>
      <expression>
        <functional> eq(add(X,Y),abs(sub(Z,T))) </functional>
      </expression>
    </predicate>
  </predicates>
  ...
```

Figure 5: Test Instance in intention (to be continued)

```
  ...
 <constraints nbConstraints="5">
   <constraint name="C0" arity="2" scope="X0 X1" reference="P0">
     <parameters> X0 X1 </parameters>
   </constraint>
   <constraint name="C1" arity="2" scope="X0 X3" reference="P1">
     <parameters> X3 X0 2 </parameters>
   </constraint>
   <constraint name="C2" arity="2" scope="X0 X2" reference="P2">
     <parameters> X2 X0 2 </parameters>
   </constraint>
   <constraint name="C3" arity="3" scope="X1 X2 X3" reference="P3">
     <parameters> X1 2 X2 X3 </parameters>
   </constraint>
   <constraint name="C4" arity="2" scope="X1 X4" reference="P0">
     <parameters> X1 X4 </parameters>
   </constraint>
 </constraints>
</instance>
```

Figure 6: Test Instance in intention (continued)

```
<instance>
  <presentation  name="Magic Square"  format="XCSP 2.0">
     This is the magic square of order 3.
  </presentation>

  <domains nbDomains="1">
    <domain name="dom0" nbValues="9">
      1..9
    </domain>
  </domains>

  <variables nbVariables="9">
    <variable name="X0" domain="dom0"/>
    <variable name="X1" domain="dom0"/>
    <variable name="X2" domain="dom0"/>
    <variable name="X3" domain="dom0"/>
    <variable name="X4" domain="dom0"/>
    <variable name="X5" domain="dom0"/>
    <variable name="X6" domain="dom0"/>
    <variable name="X7" domain="dom0"/>
    <variable name="X8" domain="dom0"/>
  </variables>

  <constraints nbConstraints="8">
    <constraint name="C0" arity="3" scope="X0 X1 X2" reference="global:weightedSum">
     <parameters>
        <list> 1 X0 1 X1 1 X2 </list>
        <op> eq </op>
        15
     </parameters>
    </constraint>
    <constraint name="C1" arity="3" scope="X3 X4 X5" reference="global:weightedSum">
     <parameters>
        <list> 1 X3 1 X4 1 X5 </list>
        <op> eq </op>
        15
     </parameters>
    </constraint>
    <constraint name="C2" arity="3" scope="X6 X7 X8" reference="global:weightedSum">
     <parameters>
        <list> 1 X6 1 X7 1 X8 </list>
        <op> eq </op>
        15
     </parameters>
    </constraint>
    <constraint name="C3" arity="3" scope="X0 X3 X6" reference="global:weightedSum">
     <parameters>
        <list> 1 X0 1 X3 1 X6 </list>
        <op> eq </op>
        15
     </parameters>
    </constraint>
    ...
```

Figure 7: The 3-magic square (to be continued)

```
    ...
   <constraint name="C4" arity="3" scope="X1 X4 X7" reference="global:weightedSum">
    <parameters>
       <list> 1 X1 1 X4 1 X7 </list>
       <op> eq </op>
       15
    </parameters>
   </constraint>
   <constraint name="C5" arity="3" scope="X2 X5 X8" reference="global:weightedSum">
    <parameters>
       <list> 1 X2 1 X5 1 X8 </list>
       <op> eq </op>
       15
    </parameters>
   </constraint>
   <constraint name="C6" arity="3" scope="X0 X4 X8" reference="global:weightedSum">
    <parameters>
       <list> 1 X0 1 X4 1 X8 </list>
       <op> eq </op>
       15
    </parameters>
   </constraint>
   <constraint name="C7" arity="3" scope="X2 X4 X6" reference="global:weightedSum">
    <parameters>
       <list> 1 X2 1 X4 1 X6 </list>
       <op> eq </op>
       15
    </parameters>
   </constraint>
   <constraint name="C8" arity="9" scope="X0 X1 X2 X3 X4 X5 X6 X7 X8"
               reference="global:allDifferent" />
  </constraints>
</instance>
```

Figure 8: The 3-magic square (continued)