The Syntax of Essence′

November 27, 2007

Essence′ is a solver-independent constraint modelling language, which is a subset of the abstract specification language Essence [1]. Hence Essence′ can be used to

- formulate constraint problem models
- specify parameter values
- summarise problem solutions

In this document we give the grammar specification of each of these different levels that is supported by the Essence′ translator [2]. Please note that this is not a formal specification of the Essence′ language.

1 Grammar Specification

An Essence′ problem instance consists of two separate specifications: a problem model defining decision variables, domains and constraints, and a parameter specification giving parameter values to specify the problem instance. The solution(s) of a problem instance can then be summarised by a solution specification. Hence we have three different types of specifications:

1. problem model
2. parameter specification
3. solution specification

Before giving a concise context-free grammar for each part, we want to give an overview of the notation that is used.

1.1 Notation

- Terms written in *italic font* are non-terminals and terms written in `typewriter font` or special characters that are underlined (such as :) are terminals.
- Comments are preceded by `$`, which can be placed everywhere in the grammar.

- A *letter* is an alphabetic character. An *identifier* is a string whose first character is a *letter* and the rest of its characters are alphanumeric or "_". Identifier recognition is case sensitive.
- A *number* is any string whose elements are the numeric characters.
- {*a*} stands for a non-empty list of *a*s.
- {*a*}' stands for a non-empty list of *a*s separated by commas.
- {*a*}* stands for a non-empty list of *a*s separated by the symbol "*".
- [*a*] stands for one or zero occurences of *a*.

1.2 Grammar: Problem Model

1.2.1 Model

Model	::=	*Header* [{*Declaration*}'] [*Objective*] [`such that` { *Expression* }']
Header	::=	`ESSENCE'` *Number* . *Number*
Declaration	::=	`given` { *Parameter* }'| `where` { *Expression* }' | `letting` { *Constant* }' | `find` { *Variable* }'
Objective	::=	`maximising` *Expression* | `minimising` *Expression*
DomainIdentifiers	::=	{ *Identifier* }' : *Domain*
Constant	::=	*Identifier* `be domain` *Domain* | *Identifier* [: *Domain*] `be` *Expression*
Parameter	::=	*DomainIdentifiers*
Variable	::=	*DomainIdentifiers*

1.2.2 Domains

SimpleDomain	::	`bool` | `int` ({ *RangeAtom* }') | *Identifier*
Domain	::=	*SimpleDomain* | (*Domain*) | `matrix indexed by` [{ *Domain* }'] `of` *SimpleDomain*
RangeAtom	::=	*Expression* | *Expression* .. *Expression*
IndexRangeAtom	::=	*Expression* | *Expression* .. *Expression* | ..*Expression* | *Expression* ..

1.2.3 Expressions

Expression	::=	(*Expression*) |
		AtomExpression |
		DeRefExpression |
		UnaryOpExpression |
		BinaryOpExpression |
		GlobalConstraint |
		QuantifierOpExpression |
AtomExpression	::=	*Number* | `true` | `false` | *Identifier*
DeRefExpression	::=	*Identifier* [{ *IndexRangeAtom* }']
UnaryOpExpression	::=	- *Expression* |
		| *Expression* | |
		! *Expression*
BinaryOpExpression	::=	*Expression BiOp Expression*
BiOp	::=	`+` | `-` | `/` | `*` | `^` |
		`\/` | `/\` | `=>` | `<=>` |
		`=` | `!=` | `<=` | `<` | `>=` | `>` |
		`<lex` | `<=lex` | `>lex` | `>=lex`
GlobalConstraint	::=	`alldiff` (*Expression*) |
		`element` (*Expression* , *AtomExpression* , *AtomExpression*) |
		`table` ([`VariableList`] , [`TupleList`]) |
		`atleast` (`Expression` , `ConstantList` , `ConstantList`) |
		`atmost` (`Expression` , `ConstantList` , `ConstantList`)
QuantifierOpExpression	::=	*Quantifier BindingExpression* "." *Expression*
Quantifier	::=	`forall` | `exists` | `sum`
BindingExpression	::=	{ *Identifier* }' : *SimpleDomain*

1.2.4 Further Restrictions

- Quantifications may not range over decision variables, i.e. *BindingExpressions* may not contain decision variables

1.3 Grammar: Parameter Specification

ParameterSpecification	::=	*Header*
		[{*Declaration*}']
Header	::=	`ESSENCE'` *Number* . *Number* . *Number*
Declaration	::=	`param` { *Constant* }'
Constant	::=	*Identifier* `be domain` *Domain* |
		Identifier `be` *ConstantExpression*
ConstantExpression	::=	*Number* | `true` | `false`
ConstantDomain	::	`bool` |
		`int` ({ *RangeAtom* }')
RangeAtom	::=	*ConstantExpression* |
		ConstantExpression .. *ConstantExpression*

Operator	Functionality	Associativity
,	comma	Left
:	colon	Left
( )	left and right parenthesis	Left
`[ ]`	left and right brackets	Left
!	not	Right
`/\`	and	Left
`\/`	or	Left
`=>`	if (implication)	Left
`<=>`	iff (logical equality)	Left
`-`	unary minus	Right
`^`	power	Left
* /	multiplication, integer division	Left
`+ -`	addition, substraction	Left
`< <= > >=` `<lex <=lex >lex >=lex`	(lex)less, (lex)less or equal, (lex)greater, (lex)greater or equal	none
`= !=`	equality, disequality	none
.	dot	Right

Table 1: Operator precedence in ESSENCE′

1.4 Grammar: Solution Specification

SolutionSpecification ::= *Header*
[{*Solution*}']
Header ::= `ESSENCE′` *Number* . *Number*
Solution ::= `variable` *Identifier* `is` { *SolutionExpression*}'
SolutionExpression ::= *Number* |
ConstantArray |
ConstantArray ::= [{ *Number* }'] |
[{ *ConstantArray* }']

2 Operator Precedence

Table 1 describes the precedence of the operators that are arranged by decreasing order of precedence (the operators on top have highest precedence)

3 Examples

To illustrate the grammar specified above, we give some examples. These examples can be found with the ESSENCE′ translator which is available with Minion[1].

[1] http://minion.sourceforge.net/

3.1 N-Queens

We start with the well-known n-queens problem which is to place n queens on an $n \times n$ chessboard. The problem model states the problem class, as given as follows:

***N-Queens* problem model**

```
ESSENCE' 1.0

given      n :  int(1...)
letting    INDEX be domain int(1..n)
find       f :  matrix indexed by [ INDEX ] of INDEX

such that  alldifferent(f),
           forall i,j :  INDEX .
             (i > j) =>
               ((f[i] - i != f[j] - j)
               /\ (f[i] + i != f[j] + j))
```

To solve a problem instance of the n-queens problem, we need to give a value for the parameter n. We do this in the parameter specification:

***N-Queens* parameter specification**

```
ESSENCE' 1.0
param      n be 8
```

After giving both problem model and parameter specification to a solver, for instance using the ESSENCE′ translator, you will be given a solution, such as

***N-Queens* solution specification**

```
ESSENCE' 1.0
var      f is [0, 2, 1, 4, 3, 6, 7, 5]
```

References

[1] A.M. Frisch, M. Grum, C. Jefferson, B. Martínez Hernández, and I. Miguel. The design of essence: A constraint language for specifying combinatorial problems. In *IJCAI*, pp 80–87, 2007.

[2] I.P. Gent, I. Miguel and A. Rendl. Tailoring Solver-independet Constraint Models: A Case Study with Essence′ and Minion In *SARA*, 2007.

[3] I.P. Gent, C. Jefferson, and I. Miguel. Minion: A fast scalable constraint solver. In *ECAI*, pp 98–102, 2006.