The Syntax of ESSENCE′

Alan M. Frisch, Chris Jefferson, Bernadette Martínez Hernández,
Ian Miguel, Andrea Rendl

September 13, 2007

This document describes the syntax of ESSENCE′ version 0.1, a solver-independent constraint modelling language, which is a subset of ESSENCE [1]. The development of ESSENCE is incremental, so features are added to both ESSENCE and ESSENCE′, as we go along.

1 ESSENCE' grammar

An ESSENCE′ problem model consists of a problem specification defining decision variables, domains and constraints, and a parameter specification giving parameter values to specify the problem instance. Comments are preceded by `$`, which can be placed everywhere in the grammar.

1.1 Notation

- 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 as.
- $\{a\}$' stands for a non-empty list of as separated by commas.
- $\{a\}$* stands for a non-empty list of as separated by the symbol "*".
- $[a]$ stands for a nil or one occurence of a.

1.2 CF Grammar

1.2.1 Model

model	::=	*header*
		[{*declaration*}']
		[*objective*]
		[`such that` { *expression* }']
header	::=	`ESSENCE` *number* "." *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*

1.2.3 Expressions

expression	::=	“(“ *expression* “)” |
		atomExpression |
		deRefExpression |
		unitOpExpression |
		binaryOpExpression |
		functionOpExpression |
		quantifierOpExpression |
atomExpression	::=	*number* | `true` | `false` | *identifier*
deRefExpression	::=	*identifier* “[“ { *expression* }’ “]”
unitOpExpression	::=	“-” *expression* |
		“|” *expression* “|” |
		“!” *expression*
binaryOpExpression	::=	*expression* *biOp* *expression*
biOp	::=	`+` | `-` | `/` | `*` | `^` |
		`\/` | `/\` | `=>` | `<=>` |
		`=` | `!=` | `<=` | `<` | `>=` | `>` |
		`<lex` | `<=lex` | `>lex` | `>=lex`
functionOpExpression	::=	`alldiff` “(“ *expression* “)” |
		`element` “(“ *expression*, *atomExpression*, *atomExpression* “)” |
		`min` “(“ *expression*, “)” |
		`max` “(“ *expression*, “)”
quantifierOpExpression	::=	*quantifier* *bindingExpression* “.” *expression*
quantifier	::=	`forall` | `exists` | `sum`
bindingExpression	::=	{ *identifier* }’ “:” *simpleDomain*

2 Operator Precedence

Table 2 describes the precedence of the operators that are arranged by decreasing order of precedence (the operators on top have highest precedence)

3 ESSENCE′ → MINION Translator

A subset of ESSENCE′ can be tailored to a MINION [2] problem instance by the ESSENCE′ → MINION translator. The translator is still under development, so the following grammar parts are currently **not supported**.

- arrays of decision variables with 3 or more dimensions
- arrays of parameters with 4 or more dimensions
- absolute value
- modulo
- power with a decision variable as exponent

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′

- boolean negation
- element constraint on 2-dimensional arrays
- sparse domains

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