KRR: Language Quick Reference.

Baskaran Sankaranarayanan

Introduction

The grammar, operator precedence and associativity rules are specified for four languages. The grammar uses EBNF where *non-terminals symbols are shown in italics*, **terminals symbols are in bold font**.

→	defines a rule	?	zero or one
|	alternation	∗	zero or more
ϵ	empty string	+	one or more
⟨· · ·⟩	a group	'meta char'	**terminal**
▷	line comment	''	' (single quote)

1 First Order Logic

1.1 Grammar

1: *program* → ⟨ *sentence* ⟩∗

2: *sentence* → *formula* **.**
3: *formula* → **true** | **false**
4: | *term copr term* ▷ comparison
5: | **name (** *termList* **)** ▷ predicate
6: | **forall** *varList formula*
7: | **exists** *varList formula*
8: | **{** *formula* **}** | **(** *formula* **)**
9: | ⟨ **∼** | **not** ⟩ *formula*
10: | *formula bopr formula*

11: *varList* → **variable** ⟨ **,** **variable** ⟩∗
12: *termList* → *term* ⟨ **,** *term* ⟩∗
13: *term* → *expr* | *list* ▷ term
14: *expr* → **integer** | **float** ▷ numeric term
15: | **string** ▷ constant term
16: | **constant** ▷ constant term
17: | **name (** *termList?* **)** ▷ function term
18: | **variable** ▷ variable term
19: | **(** *expr* **)** ▷ term
20: | **−** *expr* ▷ negative term
21: | *expr aopr expr* ▷ arithmetic term

22: *list* → **[ ]** ▷ Nil list
23: | **[** *expr* **]** ▷ list term
24: | **[** *expr* '**|**' ⟨ *expr* | *list* ⟩ **]** ▷ list term

25: *copr* → **lt** | **eq** | **ge** | **gt** | **ne** ▷ comparison
26: | **<** | **=** | **>=** | **>** | **! =**
27: | **le**

28: *bopr* → **iff** | **<=>** ▷ boolean connectives
29: | **implies** | **=>** | **impliedby** | **<=**
30: | **and** | **&&** | **&** | **or** | '**||**' | '**|**'

31: *aopr* → '**∗**' | **/** | **%** | '**+**' | **−**

1.2 Symbols

Keywords: **forall**, **exists**, **iff**, **implied**, **impliedby**, **lt**, **le**, **eq**, **ge**, **gt**, **ne**, **true**, **false**, **not**.

User defined symbols:

Symbol	Syntax	Example
name	`[a-z][A-Za-z0-9_]*`	man, mortal
constant	`[a-z][A-Za-z0-9_]*`	john, mary
string	`'...'`	'Woody Allen'
variable	`[A-Z][A-Za-z0-9_]*`	X, Y, Z

1.3 Operators

Operator precedence levels: highest first and operators with same precedence listed together.

Arity	Operators	Associativity
unary	**−**	Right–Left
binary	**∗**, **/**, **%**	Left–Right
binary	**+**, **−**	Left–Right
binary	**<**, **=**, **>=**, **>**, **! =**, **<>**, **lt**, **le**, **eq**, **ge**, **gt**, **ne**	N/A
unary	**not**, **∼**, **forall**, **exists**	Right–Left
binary	**and**, **&&**, **&**	Left–Right
binary	**or**, **||**, **|**	Left–Right
binary	**implies**, **=>**, **impliedby**, **<=**	Right–Left
binary	**iff**, **<=>**	Right–Left

1.4 Program Example

```
/* This is a block comment */
// This is a line comment

forall X ( man(X) implies mortal(X) ).
forall X ( man(X) => mortal(X) ).

forall X ( mortal(X) impliedby man(X) ).
forall X ( mortal(X) <= man(X) ).

exists X ( p(X) and q(x) && r(X) & t(X) ) .
exists X ( p(X) or  q(x) || r(X) | t(X) ) .

forall X,Y,Z ( p(X,Y) & p(Y,Z) => p(X,Z) ).
forall X,Y   ( p(X,Y) & p(Y,X) => X = Y  ).

forall X,Y   ( p(a,Y) => p(X,b) ).

forall A,B,C { append(A,B,C) & C=B <= A=[] }.
forall X,A,B,C {
       append([X|A],B,[X|C]) <= append(A,B,C)
}.

// End of program
```

2 Horn Clauses

2.1 Grammar

```
program → ⟨ hornClause ⟩*

hornClause → predicate :- body .                 ▷ rule
           | predicate .                         ▷ fact
           | predicate '?'                       ▷ query

predicate → name ( termList )

body → subgoal ⟨ , subgoal ⟩*
subgoal → !                                      ▷ cut operator
        | literal
        | ⟨ ~ | \+ | not ⟩ literal

literal → true | false
        | term copr term                         ▷ comparison
        | predicate
        | ( literal )

termList → term ⟨ , term ⟩*
term → expr | list
expr → integer | float                           ▷ numeric term
     | string                                    ▷ constant term
     | constant                                  ▷ constant term
     | name ( termList? )                        ▷ function term
     | variable                                  ▷ variable term
     | ( expr )                                  ▷ term
     | – expr                                    ▷ negative term
     | expr aopr expr                            ▷ arithmetic term

list → [ ]                                       ▷ Nil list
     | [ expr ]                                  ▷ list term
     | [ expr '|' ⟨ expr | list ⟩ ]              ▷ list term

copr → lt | le | eq | ge | gt | ne
     | < | <= | = | >= | > | ! =

aopr → '*' | / | % | '+' | –
```

2.2 Symbols

Keywords: **lt**, **le**, **eq**, **ge**, **gt**, **ne**, **true**, **false**, **not**.

User defined symbols:

Symbol	Syntax	Example
name	`[a-z][A-Za-z0-9_]*`	person, father
constant	`[a-z][A-Za-z0-9_]*`	john, mary
string	`'...'`	'Woody Allen'
variable	`[A-Z][A-Za-z0-9_]*`	A, B, P, X

2.3 Operators

Operator precedence levels: highest first and operators with same precedence listed together.

Arity	Operators	Associativity
unary	–	Right–Left
binary	*, /, %	Left–Right
binary	+, –	Left–Right
binary	<, <=, =, >=, >, ! =, <>, **lt**, **le**, **eq**, **ge**, **gt**, **ne**	N/A
unary	**not**, ~, \+	Right–Left

2.4 Program Example

```
/* This is a block comment */
// This is a line comment

// append(A,B,C) means C = A + B.

append([], B, B).
append([X|A], B, [X|C]) :- append(A, B, C).

append([1|2],[3|4],C)?

parent(P,X)          :- mother(P,X).
parent(P,X)          :- father(P,X).
grandparent(G,X)     :- parent(G,P), parent(P,X).

cousin(X,Y)          :- X != Y,
                        not sibling(X,Y),
                        grandparent(Z,X),
                        grandparent(Z,Y).

americanCousin(X,Y) :- cousin(X,Y), !,
                        american(X).

composite(N) :- N > 1, ~ prime(N).
composite(N) :- N > 1, \+ prime(N).
composite(N) :- N > 1, not (prime(N)).

// End of program
```

3 Production Systems

3.1 Grammar

```
 1:  program → ⟨ sentence ⟩*

 2:  sentence → rule | wme

 3:  rule → if condition+ then action+

 4:  condition → wme
 5:            | −wme

 6:  action → add wme
 7:         | remove integer
 8:         | modify integer ( attribute spec )

 9:  wme → ( type attrSpec* )

10:  attrSpec → attribute : spec

11:  spec → atom
12:       | { testExpr }
13:       | [ evalExpr ]

14:  testExpr → true | false
15:           | atom copr                    ▷ comparison
16:           | copr atom                    ▷ comparison
17:           | [ evalExpr ] copr            ▷ comparison
18:           | copr [ evalExpr ]            ▷ comparison
19:           | ( testExpr )
20:           | ⟨ ~ | not ⟩ testExpr
21:           | testExpr bopr testExpr

22:  evalExpr → atom
23:           | ( evalExpr )
24:           | − evalExpr
25:           | evalExpr aopr evalExpr

26:  atom → true | false
27:       | integer | float
28:       | string                   ▷ constant: 'John'
29:       | constant                  ▷ constant: john
30:       | variable                  ▷ variable: X, Y

31:  copr → < | <= | = | >= | > | ! = | <>

32:  bopr → or | '||' | '|' | and | && | &

33:  aopr → '*' | / | % | '+' | −
```

3.2 Symbols

Keywords: **if**, **then**, **add**, **remove**, **modify**, **true**, **false**, **not**, **or**, **and**.

User defined symbols:

Symbol	Syntax	Example
type	`[a-z][A-Za-z0-9_]*`	brick, counter
attribute	`[a-z][A-Za-z0-9_]*`	name, size
constant	`[a-z][A-Za-z0-9_]*`	heap, hand
string	`'...'`	'A', 'B', 'C'
variable	`[A-Z][A-Za-z0-9_]*`	S, N, I

3.3 Operators

Operator precedence levels: highest first and operators with same precedence listed together.

Arity	Operators	Associativity
unary	−	Right–Left
binary	*, /, %	Left–Right
binary	+, −	Left–Right
binary	<, <=, =, >=, >, ! =, <>	N/A
unary	**not**, ~	Right–Left
binary	**and**, &&, &	Left–Right
binary	**or**, ||, |	Left–Right

3.4 Program Example

```
/* This is a block comment */
// This is a line comment

(counter value: 1)
(brick name: 'A' size: 10 position: heap)
(brick name: 'B' size: 30 position: heap)
(brick name: 'C' size: 20 position: heap)

IF (brick position: heap name: N size: S)
  -(brick position: heap size: {> S})
  -(brick position: hand)
THEN
   MODIFY 1 (position hand)

IF (brick position: hand)
   (counter value: I)
THEN
   MODIFY 1 (position I)
   MODIFY 2 (value [I + 1])

// End of program
```

4 Description Logic

4.1 Grammar

```
program → ⟨ sentence ⟩*

sentence → ( concept bopr concept )
         | ( constantList -> concept )

concept → name                              ▷ atomic concept
        | [ fills role constant ]
        | [ all role concept ]
        | [ exists integer role ]
        | [ and concept concept+ ]

role → : name

constantList → constant ⟨ , constant ⟩*
constant → integer | float
         | string
         | constant
         | ( constant )
         | − constant
         | constant aopr constant

bopr → isa | <<                             ▷ A ⊑ B
     | subsumes | >>                        ▷ A ⊒ B
     | equivalentto | ==                    ▷ A ≐ B

aopr → '*' | / | % | '+' | −
```

4.2 Symbols

Keywords: **fills**, **all**, **exists**, **and**, **is**, **isa**, **subsumes**, **equivalentto**.

User defined symbols:

Symbol	Syntax	Example
name	`[A-Z][A-Za-z0-9_]*`	Person, Man
constant	`[a-z][A-Za-z0-9_]*`	john, mary
string	`'...'`	'Woody Allen'

4.3 Operators

Operator precedence levels: highest first and operators with same precedence listed together.

Arity	Operators	Associativity
unary	−	Right–Left
binary	*, /, %	Left–Right
binary	+, −	Left–Right
binary	is, isa, <<, subsumes, >>, equivalentto, ==	Left–Right

4.4 Program Example

```
/* This is a block comment */
// This is a line comment

( BlendedRedWine
  ==
  [AND Wine
       [FILLS :Color red]
       [EXISTS 2 :GrapeType]
  ]
)

( ProgressiveCompany
  ==
  [AND Company
       [EXISTS 7 :Director]
       [ALL :Manager
            [AND Woman [FILLS :Degree phD] ]
       ]
       [FILLS :MinSalary 24.00/hour]
  ]
)

// End of program
```