KRR: Language Quick Reference.

S. Baskaran

Introduction

The grammar (EBNF rules), operator precedence and operator associativity are specified for five languages. In ENBF, the *non-terminals are in italics font*, the `terminals are in blue monospace font`, and the meta characters and escape sequences are:

$\rightarrow$	defines a rule	?	zero or one
|	alternation	$*$	zero or more
ϵ	empty string	$+$	one or more
$\langle\cdots\rangle$	grouping	$\triangleright$	line comment

1 First Order Logic

1.1 Grammar

1: *program* $\rightarrow$ $\langle$ *sentence* $\rangle*$
2: *sentence* $\rightarrow$ *formula* `.`
3: *formula* $\rightarrow$ `true` | `false`
4: | *term cOP term* $\triangleright$ comparison
5: | `NAME (` *termList* `)` $\triangleright$ predicate
6: | `forall` *varList formula*
7: | `exists` *varList formula*
8: | `{` *formula* `}` | `(` *formula* `)`
9: | $\langle\sim$ | `not` $\rangle$ *formula*
10: | *formula bOP formula*

11: *varList* $\rightarrow$ `VARIABLE` $\langle$ `,` `VARIABLE` $\rangle*$
12: *termList* $\rightarrow$ *term* $\langle$ `,` *term* $\rangle*$
13: *term* $\rightarrow$ *expr* | *list* $\triangleright$ term
14: *expr* $\rightarrow$ `INTEGER` | `FLOAT` $\triangleright$ numeric term
15: | `STRING` $\triangleright$ constant term
16: | `CONSTANT` $\triangleright$ constant term
17: | `NAME (` $\langle$ *termList* $\rangle?$ `)` $\triangleright$ function term
18: | `VARIABLE` $\triangleright$ variable term
19: | `(` *expr* `)` $\triangleright$ term
20: | `-` *expr* $\triangleright$ negative term
21: | *expr aOP expr* $\triangleright$ arithmetic term

22: *list* $\rightarrow$ `[ ]` $\triangleright$ nil list
23: | `[` *term* `|` $\langle$ `VARIABLE` | *list* $\rangle$ `]` $\triangleright$ list

24: *cOP* $\rightarrow$ `lt` | `eq` | `ge` | `gt` | `ne` $\triangleright$ comparison
25: | `<` | `=` | `>=` | `>` | `!=`
26: | `le`

27: *bOP* $\rightarrow$ `iff` | `<=>` $\triangleright$ boolean connectives
28: | `implies` | `=>` | `impliedby` | `<=`
29: | `and` | `&&` | `&` | `or` | `||` | `|`

30: *aOP* $\rightarrow$ `*` | `/` | `%` | `+` | `-`

1.2 Symbols

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

User defined symbols:

Symbol	Syntax	Example
`NAME`	`[a-z][A-Za-z0-9_]*`	`likes, age`
`CONSTANT`	`[a-z][A-Za-z0-9_]*`	`anna, elsa`
`STRING`	`'...'`	`'WALL-E'`
`VARIABLE`	`[A-Z][A-Za-z0-9_]*`	`X, Y, Z`

1.3 Operators

Listed from highest to lowest precedence and ones with equal precedence are grouped together.

Arity	Operators	Assoc.
unary	`-`	right
binary	`*`, `/`, `%`	left
binary	`+`, `-`	left
binary	`<`, `=`, `>=`, `>`, `!=`, `<>`, `lt`, `le`, `eq`, `ge`, `gt`, `ne`	N/A
unary	`~`, `not`, `forall`, `exists`	right
binary	`and`, `&&`, `&`	left
binary	`or`, `||`, `|`	left
binary	`implies`, `=>`, `impliedby`, `<=`	right
binary	`iff`, `<=>`	right

1.4 Example

```
# 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)
}.
```

2 Horn Clauses

Horn clause syntax is borrowed from SWI Prolog, but with some differences. We added some operators for convenience, and the queries are expressed using "?".

The syntax is so similar that it is easy to convert horn clause programs into SWI Prolog programs.

2.1 Grammar

1: *program* → ⟨ *hornClause* ⟩∗

2: *hornClause* → *predicate* `:-` *body* `.` ▷ rule
3: | *predicate* `.` ▷ fact
4: | *predicate* `?` ▷ query

5: *predicate* → `NAME` `(` *termList* `)`

6: *body* → *subgoal* ⟨ `,` *subgoal* ⟩∗
7: *subgoal* → `!` ▷ cut operator
8: | *literal*
9: | ⟨ ∼ | `not` | `\+` ⟩ *literal*

10: *literal* → `true` | `false`
11: | *term cOP term* ▷ comparison
12: | *predicate*
13: | `(` *literal* `)`

14: *termList* → *term* ⟨ `,` *term* ⟩∗
15: *term* → *expr* | *list*
16: *expr* → `INTEGER` | `FLOAT` ▷ numeric term
17: | `STRING` ▷ constant term
18: | `CONSTANT` ▷ constant term
19: | `NAME` `(` *termList*? `)` ▷ function term
20: | `VARIABLE` ▷ variable term
21: | `(` *expr* `)` ▷ term
22: | `-` *expr* ▷ negative term
23: | *expr aOP expr* ▷ arithmetic term

24: *list* → `[` `]` ▷ nil list
25: | `[` *term* `|` ⟨ `VARIABLE` | *list* ⟩ `]` ▷ list

26: *cOP* → `lt` | `le` | `eq` | `ge` | `gt` | `ne`
27: | `<` | `<=` | `=` | `>=` | `>` | `!=`

28: *aOP* → `*` | `/` | `%` | `+` | `-`

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_]*	likes, age
CONSTANT	[a-z][A-Za-z0-9_]*	anna, elsa
STRING	'...'	'WALL-E'
VARIABLE	[A-Z][A-Za-z0-9_]*	A, B, X, Y

2.3 Operators

Listed from highest to lowest precedence and ones with equal precedence are grouped together.

Arity	Operators	Assoc.
unary	-	right
binary	*, /, %	left
binary	+, -	left
binary	<, <=, =, >=, >, !=, <>, lt, le, eq, ge, gt, ne	N/A
unary	∼, not, \+	right

2.4 Example

```
# This is a line comment

### append(A,B,C) computes 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)).
```

3 Production Systems

The working-memory-elements (WMEs) and rules follow the syntax given in "Knowledge Representation and Reasoning" by Brachman and Levesque. In addition, it allows `insert` as an alias for `add` action.

3.1 Grammar

1: *program* → ⟨ *rule* | *wme* ⟩∗

2: *rule* → `if` ⟨ *condition* ⟩+ `then` ⟨ *action* ⟩+

3: *condition* → *wme* | `-` *wme*

4: *action* → `add` *wme*
5: | `insert` *wme* ▷ same as `add`
6: | `remove INTEGER`
7: | `modify INTEGER (` `ATTRIBUTE` *spec* `)`

8: *wme* → `(` `TYPE` ⟨ *attrSpec* ⟩∗ `)`

9: *attrSpec* → `ATTRIBUTE :` *spec*

10: *spec* → *atom*
11: | `{` *testExpr* `}`
12: | `[` *evalExpr* `]`

13: *testExpr* → `true` | `false`
14: | *atom cOP* ▷ comparison
15: | *cOP atom* ▷ comparison
16: | `[` *evalExpr* `]` *cOP* ▷ comparison
17: | *cOP* `[` *evalExpr* `]` ▷ comparison
18: | `(` *testExpr* `)`
19: | ⟨ ∼ | `not` ⟩ *testExpr*
20: | *testExpr bOP testExpr*

21: *evalExpr* → *atom*
22: | `(` *evalExpr* `)`
23: | `-` *evalExpr*
24: | *evalExpr aOP evalExpr*

25: *atom* → `true` | `false`
26: | `INTEGER` | `FLOAT`
27: | `STRING` ▷ constant: 'John'
28: | `CONSTANT` ▷ constant: john
29: | `VARIABLE` ▷ variable: X, Y

30: *cOP* → `<` | `<=` | `=` | `>=` | `>` | `!=` | `<>`

31: *bOP* → `or` | `||` | `|` | `and` | `&&` | `&`

32: *aOP* → `*` | `/` | `%` | `+` | `-`

3.2 Symbols

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

User defined symbols:

Symbol	Syntax	Example
TYPE	[a-z][A-Za-z0-9_]*	car, bus
ATTRIBUTE	[a-z][A-Za-z0-9_]*	hue, size
CONSTANT	[a-z][A-Za-z0-9_]*	red, big
STRING	'...'	'WALL-E'
VARIABLE	[A-Z][A-Za-z0-9_]*	A, X, Y

3.3 Operators

Listed from highest to lowest precedence and ones with equal precedence are grouped together.

Arity	Operators	Assoc.
unary	-	right
binary	*, /, %	left
binary	+, -	left
binary	<, <=, =, >=, >, !=, <>	N/A
unary	∼, not	right
binary	and, &&, &	left
binary	or, ||,		left

3.4 Example

```
# This is a line comment

### WORKING MEMORY ELEMENTS (WMEs)

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

### RULES

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])
```

4 Description Logic: $\mathcal{DL}$

This is an implementation of the description logic language $\mathcal{DL}$ from "Knowledge Representation and Reasoning" by Brachman and Levesque.

4.1 Grammar

1: *program* → ⟨ *sentence* ⟩∗

2: *sentence* → `(` *concept bOP concept* `)`
3: | `(` *constantList* `->` *concept* `)`

4: *concept* → `NAME` ▷ atomic concept
5: | `[` `fills` *role constant* `]`
6: | `[` `all` *role concept* `]`
7: | `[` `exists` `INTEGER` *role* `]`
8: | `[` `and` *concept concept+* `]`

9: *role* → `:` `NAME`

10: *constantList* → *constant* ⟨ `,` *constant* ⟩∗
11: *constant* → `INTEGER` | `FLOAT`
12: | `STRING`
13: | `CONSTANT`
14: | `(` *constant* `)`
15: | `-` *constant*
16: | *constant aOP constant*

17: *bOP* → `isa` | `<<` ▷ A ⊑ B
18: | `subsumes` | `>>` ▷ A ⊒ B
19: | `equivalentto` | `==` ▷ A ≐ B

20: *aOP* → `*` | `/` | `%` | `+` | `-`

4.2 Symbols

Keywords: `fills`, `all`, `exists`, `and`, `isa`, `subsumes`, `equivalentto`.

User defined symbols:

Symbol	Syntax	Example
NAME	[A-Z][A-Za-z0-9_]*	Man, Mortal
CONSTANT	[a-z][A-Za-z0-9_]*	anna, elsa
STRING	'...'	'WALL-E'

4.3 Operators

Listed from highest to lowest precedence and ones with equal precedence are grouped together.

Arity	Operators	Assoc.
unary	-	right
binary	*, /, %	left
binary	+, -	left
binary	is, isa, <<, subsumes, >>, equivalentto, ==	N/A

4.4 Example

```
# 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]
  ]
)
```

5 Subset of OWL 2

This is a subset of OWL 2, it follows OWL Manchester Syntax[1] from "OWL 2 Web Ontology Language Manchester Syntax (Second Edition) 11 Dec. 2012"[2].

In this grammar[3], the meta rule `xyzLIST` denotes a comma separated sequence of `xyz` values, similarly, `xyz2LIST` denotes a sequence of two or more values.

5.1 Grammar

kb → ⟨*frame*⟩∗

frame → *classFrame*
| *roleFrame*
| *individualFrame*
| `EquivalentClasses:` *description2LIST*
| `DisjointClasses:` *description2LIST*
| `EquivalentProperties:` *role2LIST*
| `DisjointProperties:` *role2LIST*
| `SameIndividual:` *individual2LIST*
| `DifferentIndividuals:` *individual2LIST*

classFrame → `Class: NAME`
⟨ `SubClassOf:` *descriptionLIST*
| `EquivalentTo:` *descriptionLIST*
| `DisjointWith:` *descriptionLIST*
| `DisjointUnionOf:` *description2LIST*
⟩∗

roleFrame → `ObjectProperty: NAME`
⟨ `Domain:` *descriptionLIST*
| `Range:` *descriptionLIST*
| `Characteristics:` *characteristicLIST*
| `SubPropertyOf:` *roleExprLIST*
| `EquivalentTo:` *roleExprLIST*
| `DisjointWith:` *roleExprLIST*
| `InverseOf:` *roleExprLIST*
| `SubPropertyChain:` *roleChain*
⟩∗

roleExpr → `ROLE` | `inverse ROLE`

roleChain → *roleExpr* ⟨`o` *roleExpr*⟩+

characteristic → `Functional`
| `InverseFunctional`
| `Reflexive`
| `Irreflexive`
| `Symmetric`
| `Asymmetric`
| `Transitive`

individualFrame → `Individual: NAME`
⟨ `Types:` *descriptionLIST*
| `Facts:` *factLIST*
| `SameAs:` *individualLIST*
| `DifferentFrom:` *individualLIST*
⟩∗

fact → `ROLE INDIVIDUAL`
| `not ROLE INDIVIDUAL`

description → `CONCEPT`
| *roleExpr* `only` *description*
| *roleExpr* `some` *description*
| *roleExpr* `value` *description*
| *roleExpr* `min INTEGER` *description*
| *roleExpr* `max INTEGER` *description*
| *roleExpr* `exactly INTEGER` *description*
| `not` *description*
| `(` *description* `)`
| *description* `and` *description*
| *description* `or` *description*

5.2 Symbols

OWL Manchester Syntax uses **case sensitive keywords**: `not`, `and`, `or`, `inverse`, `only`, `some`, `min`, `max`, `exactly`, `value`, `o`, `Functional`, `InverseFunctional`, `Reflexive`, `Irreflexive`, `Symmetric`, `Asymmetric`, `Transitive`, `Prefix:`, `Ontology:`, `Class:`, `SubClassOf:`, `EquivalentTo:`, `DisjointWith:`, `DisjointUnionOf:`, `ObjectProperty:`, `Characteristics:`, `Domain:`, `Range:`, `SubPropertyOf:`, `InverseOf:`, `SubPropertyChain:`, `Individual:`, `Types:`, `Facts:`, `SameAs:`, `DifferentFrom:`, `EquivalentClasses:`, `DisjointClasses:`, `EquivalentProperties:`, `DisjointProperties:`, `SameIndividual:`, `DifferentIndividuals:`

User defined symbols:

Symbol	Syntax	Example
`NAME`	`[a-zA-Z][A-Za-z0-9_]*`	`owns, Car`
`CONCEPT`	`[a-zA-Z][A-Za-z0-9_]*`	`Car, Bus`
`ROLE`	`[a-zA-Z][A-Za-z0-9_]*`	`owns, eats`
`INDIVIDUAL`	`[a-zA-Z][A-Za-z0-9_]*`	`lucy, jack`

5.3 Operators

Operators from highest to lowest precedence.

Arity	Operators	Assoc.
unary	`not`	N/A
binary	`and`	left
binary	`or`	left

[1] With suitable `Prefix:` and `Ontology:` entries, these files can be opened in Protege ontology management tool.

[2] `https://www.w3.org/TR/owl2-manchester-syntax/`

[3] A frame is a block of statements, it differs from Frames discussed in Brachman and Levesque, but has similar syntax.

5.4 Example

```
# This is a line comment

Class: Person
    SubClassOf:      eats some Fruit
    EquivalentTo:    Human
    DisjointWith:    Fruit, Meat
    DisjointUnionOf: Man, Woman

Class: TOAD
    EquivalentTo: Teen and owns some Apple
    SubClassOf:   Happy

ObjectProperty: hasChild
  Domain: Person
  Range:  Person
  InverseOf:  hasParent

ObjectProperty: hasSibling
  Characteristics: Symmetric
  Domain: Person
  Range:  Person

ObjectProperty: hasBrother
  Domain: Person
  Range:  Man
  SubPropertyOf: hasSibling

ObjectProperty: hasSister
  Domain: Person
  Range:  Woman
  SubPropertyOf: hasSibling

Individual: Lucy
  Types:  Woman, hasChild only Woman
  Facts:  hasHusband Manny, not owns Car157
  SameAs: SmartLucy
  DifferentFrom: Manny, Car157

EquivalentClasses: Dead, not Alive
DisjointClasses:   Fruit, Meat

EquivalentProperties: owns, hasOwnershipOf
DisjointProperties: hasBrother, hasSister

SameIndividual: Manny, LazyManny
DifferentIndividuals: Manny, Diego, Sid, Lucy
```

5.5 Note

We have implemented a subset of OWL 2, this subset is a superset of $\mathcal{ALC}$, for example, this subset supports min and max cardinality restrictions, inverse roles, etc., that are not supported in $\mathcal{ALC}$. Therefore, while testing $\mathcal{ALC}$ Tableau use only $\mathcal{ALC}$ constructors and statements discussed in the lecture.

5.6 $\mathcal{ALC}$ Examples

Persons who do not own a car.

$\text{Person} \sqcap \neg\exists\text{owns.Car}$

```
Person and not (owns some Car)
```

Those who do not travel by bus or train.

$\neg\exists\text{travelsBy.}(\text{Bus} \sqcup \text{Car})$

```
not (travelsBy some (Bus or Train))
```

$\forall\text{travelsBy.}\neg(\text{Bus} \sqcup \text{Car})$

```
travelsBy only not (Bus or Train)
```

Owns a thing that has battery.

$\exists\text{owns.}(\exists\text{hasPart.Battery})$

```
owns some (hasPart some Battery)
```

Those with friends who own only electric cars.

$\exists\text{hasFriend.}(\forall\text{owns.}(\text{Electric} \sqcap \text{Car}))$

```
hasFriend some (owns only (Electric and Car))
```

Lucy is a mother and an engineer, she works for Acme Co. Her brother Jack is a doctor and he owns a car.

$\text{Mother(Lucy)}, \quad \text{Engineer(Lucy)},$

$\text{worksFor(Lucy, AcmeCo)}, \quad \text{hasBrother(Lucy, Jack)},$

$\text{Doctor(Jack)}, \quad (\exists\text{owns.Car})\text{(Jack)}$

```
Individual: Lucy
  Types: Mother, Engineer
  Facts: worksFor AcmeCo, hasBrother Jack

Individual: Jack
  Types: Doctor, owns some Car
```

Domain of hasBrother is Person and range is Man.

$\exists\text{hasBrother.}\top \sqsubseteq \text{Person}$ (domain axiom)

$\top \sqsubseteq \forall\text{hasBrother.Man}$ (range axiom)

```
ObjectProperty: hasBrother
  Domain: Person
  Range:  Man
```