

# 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 are:

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

## 1 First Order Logic

### 1.1 Grammar

- 1: *program*  $\rightarrow \langle \textit{sentence} \rangle^*$
- 2: *sentence*  $\rightarrow \textit{formula} .$
- 3: *formula*  $\rightarrow \text{true} \mid \text{false}$
- 4:  $\mid \textit{term } cOP \textit{ term}$   $\triangleright$  comparison
- 5:  $\mid \text{NAME} ( \textit{terms} )$   $\triangleright$  predicate
- 6:  $\mid \text{forall} \textit{ vars formula}$
- 7:  $\mid \text{exists} \textit{ vars formula}$
- 8:  $\mid \{ \textit{formula} \}$
- 9:  $\mid ( \textit{formula} )$
- 10:  $\mid \langle \sim \mid \text{not} \rangle \textit{formula}$
- 11:  $\mid \textit{formula } bOP \textit{ formula}$
- 12: *vars*  $\rightarrow \text{VARIABLE} \langle , \text{VARIABLE} \rangle^*$
- 13: *terms*  $\rightarrow \textit{term} \langle , \textit{term} \rangle^*$
- 14: *term*  $\rightarrow \textit{expr} \mid \textit{list}$   $\triangleright$  term
- 15: *expr*  $\rightarrow \text{INTEGER} \mid \text{FLOAT}$   $\triangleright$  numeric term
- 16:  $\mid \text{STRING}$   $\triangleright$  constant term
- 17:  $\mid \text{CONSTANT}$   $\triangleright$  constant term
- 18:  $\mid \text{NAME} ( \langle \textit{terms} \rangle? )$   $\triangleright$  function term
- 19:  $\mid \text{VARIABLE}$   $\triangleright$  variable term
- 20:  $\mid ( \textit{expr} )$   $\triangleright$  term
- 21:  $\mid - \textit{expr}$   $\triangleright$  negative term
- 22:  $\mid \textit{expr } aOP \textit{ expr}$   $\triangleright$  arithmetic term
- 23: *list*  $\rightarrow [ ]$   $\triangleright$  nil list
- 24:  $\mid [ \textit{term} \mid \langle \text{VARIABLE} \mid \textit{list} \rangle ]$   $\triangleright$  list
- 25: *cOP*  $\rightarrow \text{lt} \mid \text{eq} \mid \text{ge} \mid \text{gt} \mid \text{ne}$   $\triangleright$  comparison
- 26:  $\mid < \mid = \mid >= \mid > \mid !=$
- 27:  $\mid \text{le}$
- 28: *bOP*  $\rightarrow \text{iff} \mid <=>$   $\triangleright$  boolean connectives
- 29:  $\mid \text{implies} \mid => \mid \text{impliedby} \mid <=$
- 30:  $\mid \text{and} \mid \&\& \mid \& \mid \text{or} \mid || \mid |$
- 31: *aOP*  $\rightarrow * \mid / \mid \% \mid + \mid -$

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

### 1.3 Operators

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

Arity	Operators	Assoc.
unary	<code>-</code>	right
binary	<code>*, /, %</code>	left
binary	<code>+, -</code>	left
binary	<code>&lt;, =, &gt;=, &gt;, !=, &lt;&gt;, lt, le, eq, ge, gt, ne</code>	N/A
unary	<code>~, not, forall, exists</code>	right
binary	<code>and, &amp;&amp;, &amp;</code>	left
binary	<code>or,   ,  </code>	left
binary	<code>implies, ==&gt;, impliedby, &lt;=</code>	right
binary	<code>iff, &lt;=&gt;</code>	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:  $program \rightarrow \langle hornClause \rangle^*$
- 2:  $hornClause \rightarrow predicate \text{ :- } body \ .$  ▷ rule
- 3:  $\quad \quad \quad | \quad predicate \ .$  ▷ fact
- 4:  $\quad \quad \quad | \quad predicate \ ?$  ▷ query
- 5:  $predicate \rightarrow NAME \ ( \ terms \ )$
- 6:  $body \rightarrow subgoal \ ( \ , \ subgoal \ )^*$
- 7:  $subgoal \rightarrow !$  ▷ cut operator
- 8:  $\quad \quad \quad | \quad literal$
- 9:  $\quad \quad \quad | \quad \langle \sim \mid not \mid \backslash + \rangle \ literal$
- 10:  $literal \rightarrow true \mid false$
- 11:  $\quad \quad \quad | \quad term \ cOP \ term$  ▷ comparison
- 12:  $\quad \quad \quad | \quad predicate$
- 13:  $\quad \quad \quad | \quad ( \ literal \ )$
- 14:  $terms \rightarrow term \ ( \ , \ term \ )^*$
- 15:  $term \rightarrow expr \mid list$
- 16:  $expr \rightarrow INTEGER \mid FLOAT$  ▷ numeric term
- 17:  $\quad \quad \quad | \quad STRING$  ▷ constant term
- 18:  $\quad \quad \quad | \quad CONSTANT$  ▷ constant term
- 19:  $\quad \quad \quad | \quad NAME \ ( \ \langle terms \rangle? \ )$  ▷ function term
- 20:  $\quad \quad \quad | \quad VARIABLE$  ▷ variable term
- 21:  $\quad \quad \quad | \quad ( \ expr \ )$  ▷ term
- 22:  $\quad \quad \quad | \quad - \ expr$  ▷ negative term
- 23:  $\quad \quad \quad | \quad expr \ aOP \ expr$  ▷ arithmetic term
- 24:  $list \rightarrow [ \ ]$  ▷ nil list
- 25:  $\quad \quad \quad | \quad [ \ term \mid \langle VARIABLE \mid list \rangle \ ]$  ▷ list
- 26:  $cOP \rightarrow lt \mid le \mid eq \mid ge \mid gt \mid ne$
- 27:  $\quad \quad \quad | \quad < \mid <= \mid = \mid >= \mid > \mid !=$
- 28:  $aOP \rightarrow * \mid / \mid \% \mid + \mid -$

### 2.2 Symbols

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

User defined symbols:

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

### 2.3 Operators

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

Arity	Operators	Assoc.
unary	<code>-</code>	right
binary	<code>*</code> , <code>/</code> , <code>%</code>	left
binary	<code>+</code> , <code>-</code>	left
binary	<code>&lt;</code> , <code>&lt;=</code> , <code>=</code> , <code>&gt;=</code> , <code>&gt;</code> , <code>!=</code> , <code>&lt;&gt;</code> , <code>lt</code> , <code>le</code> , <code>eq</code> , <code>ge</code> , <code>gt</code> , <code>ne</code>	N/A
unary	<code>~</code> , <code>not</code> , <code>\+</code>	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:  $program \rightarrow \langle rule \mid wme \rangle^*$
- 2:  $rule \rightarrow \text{if} \langle condition \rangle + \text{then} \langle action \rangle +$
- 3:  $condition \rightarrow wme \mid - wme$
- 4:  $action \rightarrow \langle \text{add} \mid \text{insert} \rangle wme$
- 5:  $\quad \mid \text{remove INTEGER}$
- 6:  $\quad \mid \text{modify INTEGER ( ATTRIBUTE spec )}$
- 7:  $wme \rightarrow ( \text{TYPE} \langle attrSpec \rangle^* )$
- 8:  $attrSpec \rightarrow \text{ATTRIBUTE} : spec$
- 9:  $spec \rightarrow atom$
- 10:  $\quad \mid \{ testExpr \}$
- 11:  $\quad \mid [ evalExpr ]$
- 12:  $testExpr \rightarrow \text{true} \mid \text{false}$
- 13:  $\quad \mid atom \ cOP \quad \triangleright \text{comparison}$
- 14:  $\quad \mid cOP \ atom \quad \triangleright \text{comparison}$
- 15:  $\quad \mid [ evalExpr ] \ cOP \quad \triangleright \text{comparison}$
- 16:  $\quad \mid cOP \ [ evalExpr ] \quad \triangleright \text{comparison}$
- 17:  $\quad \mid ( testExpr )$
- 18:  $\quad \mid \langle \sim \mid \text{not} \rangle testExpr$
- 19:  $\quad \mid testExpr \ bOP \ testExpr$
- 20:  $evalExpr \rightarrow atom$
- 21:  $\quad \mid ( evalExpr )$
- 22:  $\quad \mid - evalExpr$
- 23:  $\quad \mid evalExpr \ aOP \ evalExpr$
- 24:  $atom \rightarrow \text{true} \mid \text{false}$
- 25:  $\quad \mid \text{INTEGER} \mid \text{FLOAT}$
- 26:  $\quad \mid \text{STRING} \quad \triangleright \text{constant: 'John'}$
- 27:  $\quad \mid \text{CONSTANT} \quad \triangleright \text{constant: john}$
- 28:  $\quad \mid \text{VARIABLE} \quad \triangleright \text{variable: X, Y}$
- 29:  $cOP \rightarrow < \mid <= \mid = \mid >= \mid > \mid != \mid <>$
- 30:  $bOP \rightarrow \text{or} \mid || \mid | \mid \text{and} \mid \&\& \mid \&$
- 31:  $aOP \rightarrow * \mid / \mid \% \mid + \mid -$

#### 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:  $program \rightarrow \langle sentence \rangle^*$
- 2:  $sentence \rightarrow ( concept \ bOP \ concept )$
- 3:  $\quad \quad \quad | ( constantList \rightarrow concept )$
- 4:  $concept \rightarrow NAME \quad \quad \quad \triangleright \text{atomic concept}$
- 5:  $\quad \quad \quad | [ fills \ role \ constant ]$
- 6:  $\quad \quad \quad | [ all \ role \ concept ]$
- 7:  $\quad \quad \quad | [ exists \ INTEGER \ role ]$
- 8:  $\quad \quad \quad | [ and \ concept \ concept^+ ]$
- 9:  $role \rightarrow : NAME$
- 10:  $constantList \rightarrow constant \langle , \ constant \rangle^*$
- 11:  $constant \rightarrow INTEGER \mid FLOAT$
- 12:  $\quad \quad \quad | STRING$
- 13:  $\quad \quad \quad | CONSTANT$
- 14:  $\quad \quad \quad | ( constant )$
- 15:  $\quad \quad \quad | - \ constant$
- 16:  $\quad \quad \quad | constant \ aOP \ constant$
- 17:  $bOP \rightarrow isa \mid << \quad \quad \quad \triangleright A \sqsubseteq B$
- 18:  $\quad \quad \quad | subsumes \mid >> \quad \quad \quad \triangleright A \sqsupseteq B$
- 19:  $\quad \quad \quad | equivalentto \mid == \quad \quad \quad \triangleright A \doteq B$
- 20:  $aOP \rightarrow * \mid / \mid \% \mid + \mid -$

### 4.2 Symbols

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

User defined symbols:

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

### 4.3 Operators

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

Arity	Operators	Assoc.
unary	<code>-</code>	right
binary	<code>*</code> , <code>/</code> , <code>%</code>	left
binary	<code>+</code> , <code>-</code>	left
binary	<code>is</code> , <code>isa</code> , <code>&lt;&lt;</code> , <code>subsumes</code> , <code>&gt;&gt;</code> , <code>equivalentto</code> , <code>==</code>	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<sup>1</sup> from “OWL 2 Web Ontology Language Manchester Syntax (Second Edition) 11 Dec. 2012”<sup>2</sup>.

In this grammar<sup>3</sup>, the meta rule **xyzLIST** denotes a comma separated sequence of one or more **xyz** values, and similarly, **xyz2LIST** denotes two or more values.

### 5.1 Grammar

```

1: kb → ⟨ frame ⟩*

2: frame → classFrame
3:   | roleFrame
4:   | individualFrame
5:   | EquivalentClasses: description2LIST
6:   | DisjointClasses: description2LIST
7:   | EquivalentProperties: role2LIST
8:   | DisjointProperties: role2LIST
9:   | SameIndividual: individual2LIST
10:  | DifferentIndividuals: individual2LIST

11: classFrame → Class: NAME
12:   ⟨ SubClassOf: descriptionLIST
13:   | EquivalentTo: descriptionLIST
14:   | DisjointWith: descriptionLIST
15:   | DisjointUnionOf: description2LIST
16:   ⟩*

17: roleFrame → ObjectProperty: NAME
18:   ⟨ Domain: descriptionLIST
19:   | Range: descriptionLIST
20:   | Characteristics: characteristicLIST
21:   | SubPropertyOf: roleExprLIST
22:   | EquivalentTo: roleExprLIST
23:   | DisjointWith: roleExprLIST
24:   | InverseOf: roleExprLIST
25:   | SubPropertyChain: roleChain
26:   ⟩*

27: roleExpr → ROLE | inverse ROLE

28: roleChain → roleExpr ⟨ o roleExpr ⟩+

29: characteristic → Functional
30:   | InverseFunctional
31:   | Reflexive
32:   | Irreflexive
33:   | Symmetric
34:   | Asymmetric
35:   | Transitive

```

<sup>1</sup>With suitable **Prefix:** and **Ontology:** entries, these files can be opened in Protege ontology management tool.

<sup>2</sup><https://www.w3.org/TR/owl2-manchester-syntax/>

<sup>3</sup>A frame is a block of statements, it differs from Frames discussed in Brachman and Levesque, but has similar syntax.

```

36: individualFrame → Individual: NAME
37:   ⟨ Types: descriptionLIST
38:   | Facts: factLIST
39:   | SameAs: individualLIST
40:   | DifferentFrom: individualLIST
41:   ⟩*

42: fact → ROLE INDIVIDUAL
43:   | not ROLE INDIVIDUAL

44: description → CONCEPT
45:   | roleExpr only description
46:   | roleExpr some description
47:   | roleExpr value description
48:   | roleExpr min INTEGER description
49:   | roleExpr max INTEGER description
50:   | roleExpr exactly INTEGER description
51:   | not description
52:   | ( description )
53:   | description and description
54:   | 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	<b>not</b>	N/A
binary	<b>and</b>	left
binary	<b>or</b>	left

## 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.

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

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.

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

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.

Mother(Lucy), Engineer(Lucy),

worksFor(Lucy, AcmeCo), hasBrother(Lucy, Jack),

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

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