Knowledge Representation and Reasoning

Logics for Artificial Intelligence

Stuart C. Shapiro

Department of Computer Science and Engineering
and Center for Cognitive Science
University at Buffalo, The State University of New York
Buffalo, NY 14260-2000

shapiro@cse.buffalo.edu

## Contents

### Part I

1. Introduction ................................................................. 4  
2. Propositional Logic .......................................................... 19  
3. Predicate Logic Over Finite Models .................................... 173  
4. Full First-Order Predicate Logic ........................................ 224  
5. Summary of Part I ........................................................... 362  

### Part II

6. Prolog ................................................................................. 375  
7. A Potpourri of Subdomains .................................................. 411  
8. SNePS .............................................................................. 429  
9. Belief Revision/Truth Maintenance ....................................... 516  
10. The Situation Calculus ....................................................... 570  
11. Summary ........................................................................... 589
Part III

12. Production Systems ................................................................. 602
13. Description Logic ................................................................. 611
14. Abduction ................................................................. 628
13 Description Logics

Main reference:
DL: Main Ideas

- Terminological Box or T-Box.
  **Definition** of *Concepts* (“Classes”) and *Roles* (“Properties”).

- Assertional Box or A-Box.
  Assertions about individuals (instances)
  - Unary predicates = concepts
  - Binary predicates = roles

- Necessary and Sufficient conditions on classes.

- Subsumption Hierarchy
Syntax of a Simple DL\textsuperscript{a}

Atomic Symbols

- Positive integers: 1, 2, 3
- Atomic concepts: Thing, Pizza, PizzaTopping, PizzaBase
  Thing is the top of the hierarchy.
- Roles: hasTopping, hasBase
- Constants: item1, item2

Syntax of a Simple DL

Concepts

• Every atomic concept is a concept

• If \( r \) is a role and \( d \) is a concept, \([\text{ALL } r \ d]\) is a concept.
  The concept of individuals all of whose \( r \)'s are \( d \)'s.
  E.g., \([\text{ALL hasTopping VegetarianTopping}]\)

• If \( r \) is a role and \( n \) is a positive integer, \([\text{EXISTS } n \ r]\) is a concept.
  The concept of individuals that have at least \( n \) \( r \)'s.
  E.g., \([\text{EXISTS 1 hasTopping}]\)

• If \( r \) is a role and \( c \) is a constant, \([\text{FILLS } r \ c]\) is a concept.
  The concept of individuals one of whose \( r \)'s is \( c \).
  E.g., \([\text{FILLS hasTopping item2}]\)

• If \( d_1, \ldots, d_n \) are concepts, \([\text{AND } d_1, \ldots, d_n]\) is a concept.
  The concept that is the intersection of \( d_1, \ldots, d_n \).
  E.g., \([\text{AND Pizza [EXISTS 1 hasTopping] [ALL hasTopping VegetarianTopping}]\]

Page 614
Syntax of a Simple DL

Sentences

- If $d_1$ and $d_2$ are concepts, $(d_1 \sqsubseteq d_2)$ is a sentence.
  $d_1$ is subsumed by $d_2$
  E.g., VegetarianPizza $\sqsubseteq$ Pizza

- If $d_1$ and $d_2$ are concepts, $(d_1 \equiv d_2)$ is a sentence.
  $d_1$ and $d_2$ are equivalent
  E.g., VegetarianPizza $\equiv$ [AND Pizza [EXISTS 1 hasTopping] [ALL hasTopping VegetarianTopping]]

- If $c$ is a constant and $d$ is a concept, $(c \rightarrow d)$ is a sentence.
  The individual $c$ satisfies the description expressed by $d$.
  E.g., item1 $\rightarrow$ Pizza
Necessary and Sufficient Conditions

A **necessary** condition on a class, $d$, is a property, $p$, such that if an individual, $c$, is an instance of $d$, it is **necessary** that $c$ satisfy $p$. A **sufficient** condition on a class, $d$, is a property, $p$, such that if an individual, $c$, satisfies $p$, then that is a **sufficient** reason to decide that it is an instance of $d$.

A **defined** concept has both necessary and sufficient conditions.

A **primitive** concept has only necessary conditions.
Subsumption Hierarchy

\[(d_1 \sqsubseteq d_2)\]

\(d_1\) is subsumed by \(d_2\)

E.g., VegetarianPizza \(\sqsubseteq\) Pizza

means that every instance of \(d_1\) is an instance of \(d_2\).

Every DL concept is subsumed by Thing, the top of the hierarchy.
Classification Algorithm

Decision procedure for placing every defined concept correctly in the subsumption hierarchy.

Note: Two concepts that subsume each other are the same.

Note: No concept can be computed as being subsumed by a primitive concept.
Examples Using Classic Defined and Primitive Concepts

: (cl-startup)
t

: (cl-define-concept 'PizzaTopping 'Classic-Thing)
*WARNING*: The new concept PizzaTopping is identical to the existing concept @c{Classic-Thing}.

@c{Classic-Thing}

: (cl-define-primitive-concept 'PizzaBase 'Classic-Thing)
@c{PizzaBase}
Creating An Individual

: (cl-create-ind 'base1 'PizzaBase)
@i{base1}

: (cl-instance? @base1 @PizzaBase)
t

: (cl-print-ind @base1)
Base1 ->
  Derived Information:
    Primitive ancestors: PizzaBase Classic-Thing
    Parents: PizzaBase
    Ancestors: Thing Classic-Thing
@i{base1}
Defining Some Roles

: (cl-define-primitive-role 'hasIngredient
  :inverse 'isIngredientOf)
@r{hasIngredient}

: (cl-define-primitive-role 'hasBase :parent 'hasIngredient
  :inverse 'isBaseOf)
@r{hasBase}

: (cl-define-primitive-role 'hasTopping :parent 'hasIngredient
  :inverse 'isToppingOf)
@r{hasTopping}
Necessary and Sufficient Conditions

: (cl-define-concept 'Pizza '(and Classic-Thing (at-least 1 hasBase)
(at-least 1 hasTopping)))

@c{Pizza}
: (cl-create-ind 'pizza1 'Pizza)
@i{pizza1}
: (cl-print-ind @pizza1)
Pizza1 ->
Derived Information:
Parents: Pizza
Ancestors: Thing Classic-Thing
Role Fillers and Restrictions:
  Hasingredient[1 ; INF]
  Hastopping[1 ; INF]
  Hasbase[1 ; INF]
@i{pizza1}
: (cl-create-ind 'item3 '(and (fills hasBase base3) (fills hasTopping topping3)))
@i{item3}
: (cl-print-ind @item3)
Item3 ->
Derived Information:
Parents: Pizza
Ancestors: Thing Classic-Thing
Role Fillers and Restrictions:
  Hasingredient[2 ; INF] -> Base3 Topping3
  Hastopping[1 ; INF] -> Topping3
  Hasbase[1 ; INF] -> Base3
@i{item3}
Classification

: (cl-define-concept 'PreparedFood '(and Classic-Thing (at-least 1 hasIngredient)))
@c{PreparedFood}

: (cl-print-concept @PreparedFood)
PreparedFood ->
  Derived Information:
    Parents: Classic-Thing
    Ancestors: Thing
    Children: Pizza
    Role Restrictions:
    Hasingredient[1 ; INF]
@c{PreparedFood}

: (cl-print-concept @Pizza)
Pizza ->
  Derived Information:
    Parents: PreparedFood
    Ancestors: Thing Classic-Thing
    Role Restrictions:
    Hasingredient[1 ; INF]
    Hastopping[1 ; INF]
    Hasbase[1 ; INF]
@c{Pizza}

: (cl-instance? @pizza1 @PreparedFood)
t
Disjoint Concepts

: (cl-startup)
t
: (cl-define-primitive-concept 'PizzaTopping 'Classic-Thing)
  @c{PizzaTopping}
: (cl-define-disjoint-primitive-concept 'CheeseTopping 'PizzaTopping 'pizzaToppings)
  @c{CheeseTopping}
: (cl-define-disjoint-primitive-concept 'MeatTopping 'PizzaTopping 'pizzaToppings)
  @c{MeatTopping}
: (cl-define-disjoint-primitive-concept 'SeafoodTopping 'PizzaTopping 'pizzaToppings)
  @c{SeafoodTopping}
: (cl-define-disjoint-primitive-concept 'VegetableTopping 'PizzaTopping 'pizzaToppings)
  @c{VegetableTopping}
classic(56): (cl-define-primitive-concept 'ProbeInconsistentTopping
  '(and CheeseTopping VegetableTopping))
*WARNING*: Disjoint primitives: @tc{CheeseTopping}, @tc{VegetableTopping}.
*CLASSIC ERROR* while processing
  (cl-define-primitive-concept ProbeInconsistentTopping (and CheeseTopping
VegetableTopping))
  occurred on object @c{ProbeInconsistentTopping--*INCOHERENT*}:
  Trying to combine disjoint primitives: @tc{CheeseTopping} and
  @tc{VegetableTopping}.
classic-error
(disjoint-prims-conflict @tc{CheeseTopping} @tc{VegetableTopping})
nil
@c{ProbeInconsistentTopping--*INCOHERENT*}
Open World

: (cl-define-primitive-concept 'MushroomTopping 'VegetableTopping)
@c{MushroomTopping}
: (cl-define-primitive-concept 'OnionTopping 'VegetableTopping)
@c{OnionTopping}
: (cl-define-concept 'VegetarianPizza '(and Pizza (all hasTopping VegetableTopping)))
@c{VegetarianPizza}

: (cl-create-ind 'mt1 'MushroomTopping)
@i{mt1}
: (cl-create-ind 'ot1 'OnionTopping)
@i{ot1}
: (cl-create-ind 'pizza2 '(and Pizza (fills hasTopping mt1) (fills hasTopping ot1)))
@i{pizza2}

: (cl-instance? @pizza2 @VegetarianPizza)
nil
: (cl-ind-close-role @pizza2 @hasTopping)
@i{pizza2}
: (cl-instance? @pizza2 @VegetarianPizza)
t
## Typology of DL Languages

<table>
<thead>
<tr>
<th>Construct</th>
<th>Syntax</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concept</td>
<td>A</td>
<td>FL&lt;sub&gt;0&lt;/sub&gt;</td>
</tr>
<tr>
<td>Role name</td>
<td>R</td>
<td>FL&lt;sup&gt;-&lt;/sup&gt;</td>
</tr>
<tr>
<td>Intersection</td>
<td>C ∩ D</td>
<td>AL</td>
</tr>
<tr>
<td>Value Restriction</td>
<td>∀ R.C</td>
<td></td>
</tr>
<tr>
<td>Limited existential quantification</td>
<td>∃R.⊤</td>
<td>S</td>
</tr>
<tr>
<td>Top or Universal</td>
<td>⊤</td>
<td></td>
</tr>
<tr>
<td>Bottom</td>
<td>⊥</td>
<td></td>
</tr>
<tr>
<td>Atomic negation</td>
<td>¬A</td>
<td></td>
</tr>
<tr>
<td>Negation</td>
<td>¬C</td>
<td>C</td>
</tr>
<tr>
<td>Union</td>
<td>C ∪ D</td>
<td>U</td>
</tr>
<tr>
<td>Existential restriction</td>
<td>∃R.C</td>
<td>E</td>
</tr>
</tbody>
</table>

Language $S = \text{ALC}_{R+} = \text{ALC}$ plus transitive roles.

Typology, continued

<table>
<thead>
<tr>
<th>Construct</th>
<th>Syntax</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number restrictions</td>
<td>$(\geq n , R) , (\leq n , R)$</td>
<td>N</td>
</tr>
<tr>
<td>Nominals</td>
<td>{a_1 \ldots a_n}</td>
<td>O</td>
</tr>
<tr>
<td>Role hierarchy</td>
<td>$R \subseteq S$</td>
<td>H</td>
</tr>
<tr>
<td>Inverse role</td>
<td>$R'$</td>
<td>I</td>
</tr>
<tr>
<td>Qualified number restriction</td>
<td>$(\geq n , R.C) , (\leq n , R.C)$</td>
<td>Q</td>
</tr>
</tbody>
</table>

Key to abbreviations under “Syntax”:
A: atomic concept
C, D: concept definitions
R: atomic role
S: role definition