CLASSIC

May 21st, 2002 Michael Minock

Classic

CLASSIC is:

the description logic that we will be using in this class

was developed at AT&T research labs

CLASSIC has been used in:

The Information Manifold Project

A big Lucent/AT&T configuration tool

Etc,...

NeoClassic

CLASSIC was originally in LISP.

NEOCLASSIC written in C++

Operates under NT and Unix.

It has a character based interface This is what we will use!

And a C++, programmers interface

Overview

- The 'DDL' Commands
- The Assertion Commands
- 'Querying'
- Rules
- Explanations and Error Handling
- Limitations

Classic

CLASSIC represents information in:

- Concepts
- Roles
- Individuals
- Rules

Basic Model

Classic Realm vs. The Host Realm

Classic Realm are the 'objects' being modeled

Host Realm are the 'literals' - in the programming language sense.

Descriptions (1-place predicates) apply to either CLASSIC or HOST individuals returning a Boolean result.

CLASSIC:

```
(and Person
  (all age (and (minimum 30) (maximum 40)))
  (atMost 5 friend)
  (all friend Male))
```

HOST:

```
(and (minimun 30) (maximum 40))
```


Classic Descriptions

```
ClassicDescription :=
  'ClassicThing' |
  ClassicConcept |
  (and ClassicDescription+) |
  (oneOf ClassicIndividual+) |
  (atLeast PositiveInteger Role) |
  (atMost NonNegativeInteger Role) |
  (fills Role ClassicIndividual) |
  (fills Role Description) |
  (all Role Description) |
  (testC ClassicTestGenerate Parameter*)
```

Role Restrictions: atLeast, atMost, fills, all.

```
(all child (and Male Female)) = (atMost 0 child)
```


Host Descriptions

```
HostDescription :=
  'HostThing' |
   Number | Integer | Float | String |
   HostConcept |
   (and HostDescription+) |
   (oneOf HostIndividual+) |
   (minimum Number) |
   (maximum Number) |
   (testH HostTestGenerate Parameter*)
```

Universal Descriptions

Top: `(and)`

Bottom: `(oneOf)`

The 'DDL' Commands

Primitive and Defined Concepts

Primitive Concepts are special. There is some quality that is beyond fully modeling.

Hence to be a member of a primitive concept an individual must be asserted to be so - and still match the other conditions of the concept.

Defined Concept - non primitive.

Creating Roles

Actually before creating concepts, it is useful to have some roles defined to use in the concept definitions.

```
(createRole Symbol Boolean)
```

If boolean is `true`, then the role may only have one filler.

Creating Concepts

```
(createConcept Symbol HostDescription)
(createConcept Symbol ClassicDescription)
```

Primitive Concepts Defined by:

```
(createConcept Symbol ClassicDescription true)
```

Disjoint Primitive Concepts By:

```
(createConcept Symbol ClassicDescription symbol+)
```

```
(createConcept Night TimePeriod dn)
(createConcept Day TimePeriod dn)
```

Individuals

Host Individuals are Constants - they may not be created, just mentioned

Classic Individuals

Syntactically the same as concepts

The assumption will be an open-world assumption until roles are explicitly closed.

Asserting Individuals

```
(createIndividual Symbol ClassicDescription)
```

```
(createIndiviidual Sam
  (and Person (all parent Doctor)
    (fills brother Fred)
    (atLeast 1 Sister)))
```

```
(closeRole Individual Role)
```

roles may become full, but they only close, when YOU close them

Updates

- Adding Information

  `(addToldInformation Individual ClassicDescription)`

- Retraction

  `(removeToldInformation Individual ClassicDescription)`

  `(uncloseRole Individual Role)`

Querying

The traditional notion of querying doesn't quite apply.

In this case, turn your query into a concept and see which individuals turn up.

(printInfo queryConcept)

(printInfo individual)

Rules

Simple Rules:

Antecedent Concept and Consequent Description:

```
(createRule Symbol ClassicConcept ClassicDescription)
```

Computed Description Rule:

```
(createRule Symbol ClassicConcept
  (computeDescription Fn Parameter*))
```

Computed Filler Rule:

```
(createRule Symbol ClassicConcept
   (computeFillers Fn Role Parameter*))
```


Incompleteness of Subsumption

Classic has several sources of incompleteness with respect to calculating subsumption:

`testH` and `testC` are treated as black-boxes.

CLASSIC individuals are treated as having no properties with respect to concept subsumption:

If Mary is known to be an `Athlete`, then the description: `(and (fills child Mary) (atMost 1 child)` is not subsumed by `(all child Athlete)`.

Rules are only forward chaining, no backward chaining.

Error Processing

Aside from simple syntax errors, there are three principle errors that you may make:

- incorrect use of names
- creation of incoherent concepts
- attempts to make the KB inconsistent