Loom: Basic Concepts

Thomas A. Russ

USC
Information Sciences Institute

Outline of Tutorial

- *LOOM Terminology*
- *Definition Language*
- *Classifier Examples*
- *Assertion Language*
- *Query Language*
- *Additional Inferences*

LOOM Terminology

Two Compartments

TBox for Definitions

ABox for Assertions (Facts)

TBox

- Term Forming Language
  - Concepts
  - Relations
- Subsumption Is Reasoning Method
- Defines “Vocabulary” of Domain

Defconcept

```
(defconcept name
  [:is | :is-primitive]description)
```

Definition Options:

- **Primitive/Non-primitive**
  - **:is   :is-primitive**
- **Combination of Other Concepts**
  - *(:and A B) (:or C D)*
- **Role Number Restrictions**
  - **(:at-least 2 arms)**
- **Role Type Restrictions**
  - **(:some child male)**

Defconcept Examples

```
(defconcept Soldier)

(defconcept Medic
    :is (:and Soldier Medical-Personnel))

(defconcept Casualty
   :is (:and Person (:at-least 1 injuries)))
```

Defconcept

```
(defconcept name
  [:is | :is-primitive]descr options)
```

Additional Options:

Characteristics

:closed-world :monotonic

Roles of the concept

(:roles R1 R2 R3)

roles are relations that are closely associated with a particular concept

Defconcept with roles

```
(defconcept Helicopter
   :roles (range payload))
```

Defrelation

```
(defrelation name
  [:is | :is-primitive]description)
```

Definition Options:

- Primitive/Non-primitive
  - `:is     :is-primitive`
- Relation to Other Concepts
  - `(:compose R S)`
- Domain and Range Restrictions
  - `(:domain person)`
- Characteristics
  - `:symmetric  :closed-world`

Necessary vs. Sufficient

Necessary and Sufficient

```
    (defconcept A
  :is (:and B C))
```

Necessary

```
    (implies A (:and B C))
```

Sufficient

```
    (implies (:and B C) A)
```

Observations About Definitions

The Loom language is “variable-free”

Requires special constructs and implicit bindings

```
    (:at-least 2 Child Male)
```

Sometimes this isn’t sufficiently expressive

Adding Expressivity (:satisfies)

- Loom definitions can be made more expressive with the “:satisfies” construct
- :satisfies is used to introduce variables.
- Example—Transitive closure

```
(defrelation R*
  :is (:satisfies (?x ?y)
         (:or (R ?x ?y)
              (:exists ?z
                 (:and (R ?x ?z)
                       (R* ?z ?y))))))
```

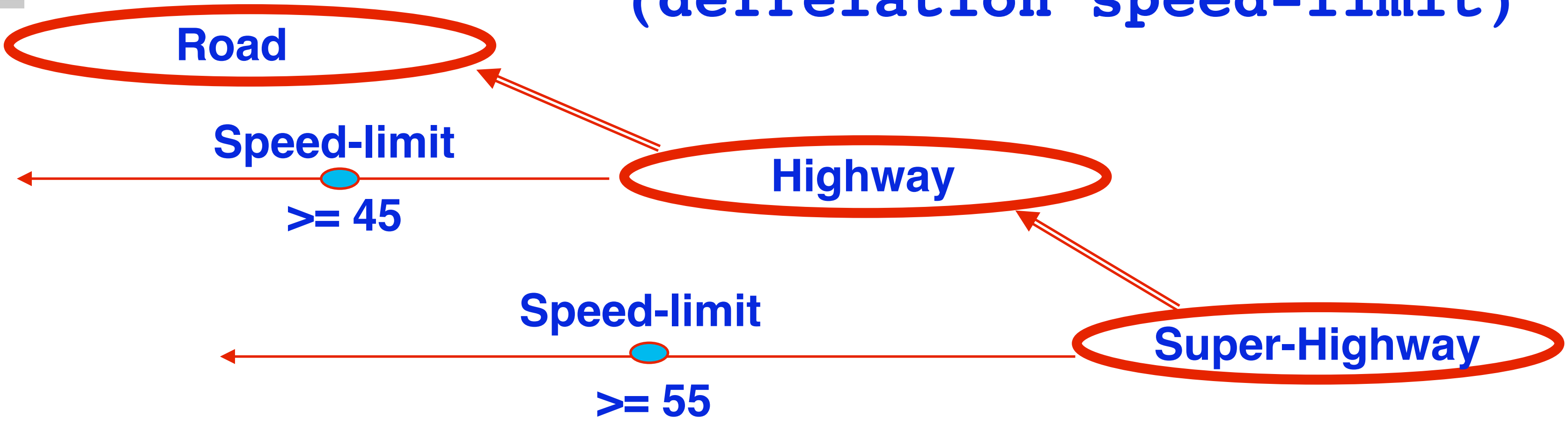
- Expressivity is higher, but Loom cannot do as much inference with :satisfies clauses

Subsumption

```
(defconcept road)
(defconcept highway
   :is (:and road
             (>= speed-limit 45)))

(defconcept super-highway
   :is (:and road
             (>= speed-limit 55)))

       (defrelation speed-limit)
```



Road

Speed-limit

>= 45

Highway

Speed-limit

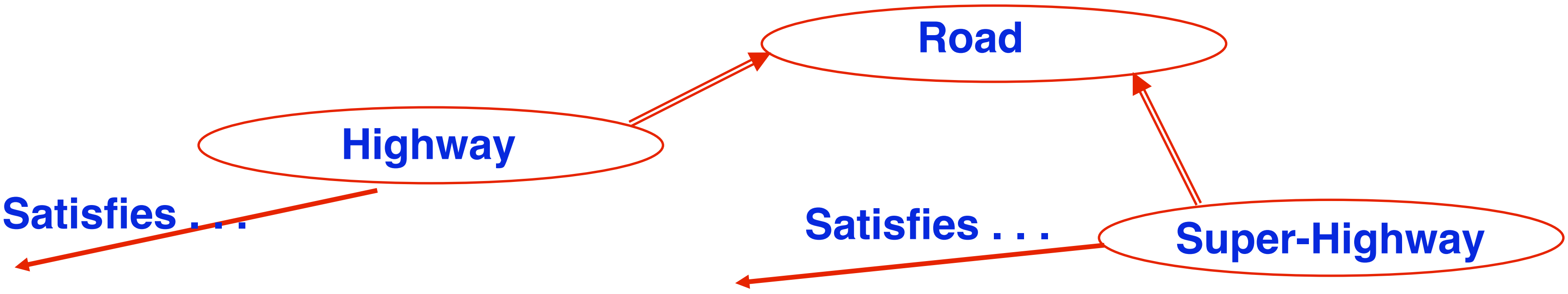
>= 55

Super-Highway

No Subsumption

```
(defconcept road)
(defrelation speed-limit)
(defconcept highway
  :is (:and road
            (:satisfies (?x)
              (>= (speed-limit ?x) 45))))

(defconcept super-highway
  :is (:and road
            (:satisfies (?x)
              (>= (speed-limit ?x) 55))))
```



Road

Highway

Super-Highway

Satisfies . . .

Satisfies . . .

Relation Hierarchies

- In Loom, relations can also be defined in hierarchies

```
(defrelation child)
  (defrelation son
        :is (:and child (:range Male)))
```

- Assertions and queries don’t have to match syntactically, only semantically
  - *If one asserts Joe is Tom’s son, then asking for Tom’s children will return Joe*
  - *Similarly, asserting that Joe is a male and Tom’s child will let Joe be retrieved by asking for Tom’s son*

ABox

- *Uses TBox Vocabulary*
- *Assertions About “Individuals”*
  - Is-a
  - Role Values
  - Restrictions

Assertions

Basic Forms:

- `tell`—Adds assertions to the knowledge base
- `forget`—Removes assertions from the knowledge base

Assertions

Basic Syntax

```
Assert is-a concept
(tell (A Joe) (B Joe))
```

Concept Name

Instance Identifier

Assertions

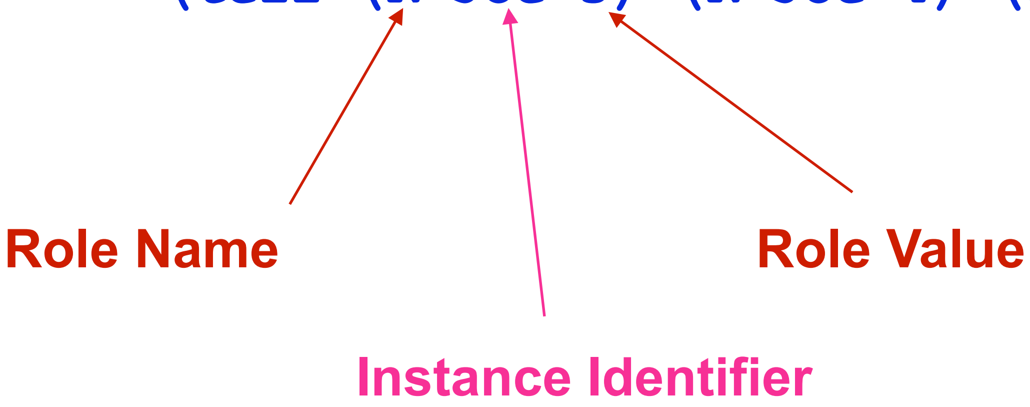
Basic Syntax

Assert is-a concept

```
(tell (A Joe) (B Joe))
```

Assert role values

```
(tell (R Joe 3) (R Joe 4) (S Joe 2))
```



Role Name

Role Value

Instance Identifier

USC ISI

Assertions

Basic Syntax

Assert is-a concept

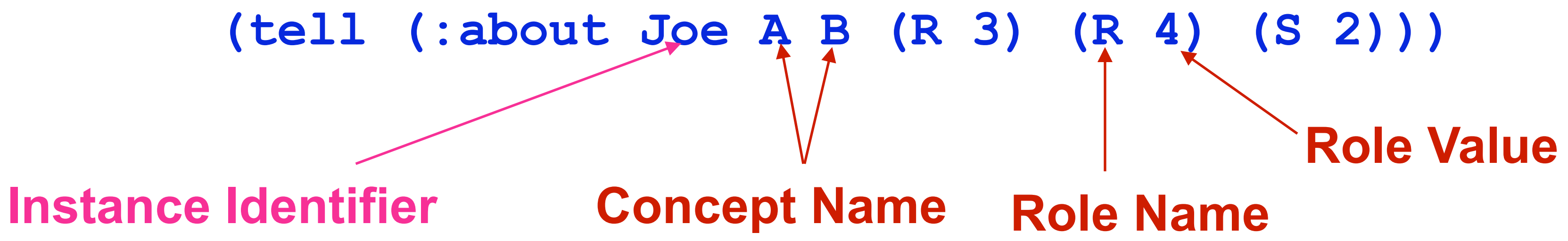
```
(tell (A Joe) (B Joe))
```

Assert role values

```
(tell (R Joe 3) (R Joe 4) (S Joe 2))
```

:about Syntax

Used for multiple assertions about a single individual:



```
(tell (:about Joe A B (R 3) (R 4) (S 2)))
```

Instance Identifier

Concept Name

Role Name

Role Value

Assertions

Basic Syntax

Assert is-a concept

```
(tell (A Joe) (B Joe))
```

Assert role values

```
(tell (R Joe 3) (R Joe 4) (S Joe 2))
```

:about Syntax

Used for multiple assertions about a single individual:

```
(tell (:about Joe A B (R 3) (R 4) (S 2)))
```

Allows assertion of restrictions

```
(tell (:about Jim (:at-least 3 R) (R 2)))
```

Queries

- *Ask About Grounded Facts*

- *Retrieve Individuals Matching Query Schema*

USC ISI

Query Language

(ask statement)

Is fido a dog?:

```
(ask (dog fido))
```

Query Language

(ask *statement*)

```
    Is fido a dog?:
       (ask (dog fido))
```

(retrieve *var-list query*)

```
    Return all dogs in the KB:
       (retrieve ?d (dog ?d))
```

USC ISI

Query Language

(ask *statement*)

Is fido a dog?:

```
(ask (dog fido))
```

(retrieve *var-list query*)

Return all dogs in the KB:

```
(retrieve ?d (dog ?d))
```

Return list of dogs and their owners:

```
(retrieve (?d ?o)
        (:and (dog ?d)
              (owner ?d ?o)))
```

Note: Ownerless dogs are not returned.

Different Decompositions

Two Axes:

- `Cover`
- `Partition`

Enable different reasoning strategies.

Cover

```
(defconcept a)
(defconcept b)
(defconcept c)
(defconcept or-abc :is (:or a b c))
```

Cover

```
(defrelation r)                 ; A common primitive parent
(defrelation s)                 ;  (ie, “x”)  is required for
                                ; this inference to be made

(defconcept x)
(defconcept a
   :is-primitive (:and x (:at-most 1 r)))
(defconcept b
   :is-primitive (:and x (:at-most 0 s)))
(defconcept c :is-primitive x)
(defconcept or-abc :is (:or a b c))

(tell (or-abc Joe))
  ;Joe is one-of A, B, or C
(tell (R Joe 1) (R Joe 2) (S Joe 1))
(ask (C Joe))    ==> T
  ;because we can rule out A and B
```

Partition

```
(defconcept p :partitions $p$)

(defconcept x :is-primitive p
              :in-partition $p$)
(defconcept y :is-primitive p
              :in-partition $p$)
(defconcept z :is-primitive p
              :in-partition $p$)


(tell (x i2))   ==>   |C|X

(tell (z i2))   ==>   INCOHERENT

(forget (x i2)) ==>   |C|Z
```

Mapping from Logic to an Object Framework

- *Loom’s language provides a logical description of instances in terms of properties and restrictions*
- *CLOS classes provide a physical description in terms of slots*
- *Loom concept descriptions can be mapped into CLOS class definitions*

Mapping from Logic to an Object Framework

Superclasses can come from

- The superconcepts (subsumption) of the concept definition
- Explicit specification via :mixin-classes

Slots can be determined multiple ways

- All :roles become slots
- All restricted relations (:at-least, etc.) in the concept definition become slots
- (Optional) All :domain restricted relations become slots.

Mapping from Logic to an Object Framework—Example

```
(defconcept C
     :is (:and A B X
               (:at-least 2 R)
               (:at-most 1 S))
     :roles (P Q)
     :mixin-classes (browser-item))
```

↓

```
(defclass C (A B X browser-item)
      ((R :accessor R :initarg :R
                      :initform nil)
       (S :accessor S ...)
       (P :accessor P ...)
       (Q :accessor Q ...)))
```

Summary

TBox Determines Domain Vocabulary

- Definitions
- Subsumption
- Disjointness

ABox Describes Specific Domain

- Instances
- Facts

Queries Retrieve Information from the ABox

- Yes/No Questions
- Find Matching Instances