

Basic Introduction to Loom

Thomas A. Russ

USC

Information Sciences Institute



What Is Loom?

- *Loom is a Knowledge Representation Language*
- *Loom is a Description Logic*
- *Loom is in the KL-ONE Family of Languages*
- *Loom is a Programming Framework*



Concepts, Relations and Instances

- *Concepts are types. Logically they are unary predicates.*

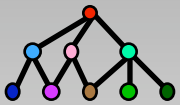
Dog, Mailman, Theory

- *Relations are tuples. Logically they are n-ary predicates. (Most relations can also be used as logical functions)*

owned-by, employer-of, proof-for

- *Instances are individuals in a domain. They may belong to one or more concepts and participate in relations.*

Fido, Fred, Evolution



Subsumption



- *The main organizing principle behind a description logic is the computation of subsumption.*
- *Concept C1 subsumes another concept C2 when all members of C2 must be members of C1.*
 - Mammal subsumes Dog*
- *C1 is more general and is a super-concept
C2 is more specific and is a sub-concept*
- *Concepts are not related by subsumption are called siblings.*

Subsumption Calculation



- *Loom computes structural subsumption. That means that the subsumption test is based on the structure, or definition, of concepts and relations.*
- *Description logics are derivatives of predicate calculus enhanced by additional combination operators. Subsumption is defined in terms of these additional operators.*
- *Subsumption calculations are done automatically and allow Loom to maintain and organize the knowledge base as it evolves.*

Loom Has Two Main Parts to a Knowledge Base



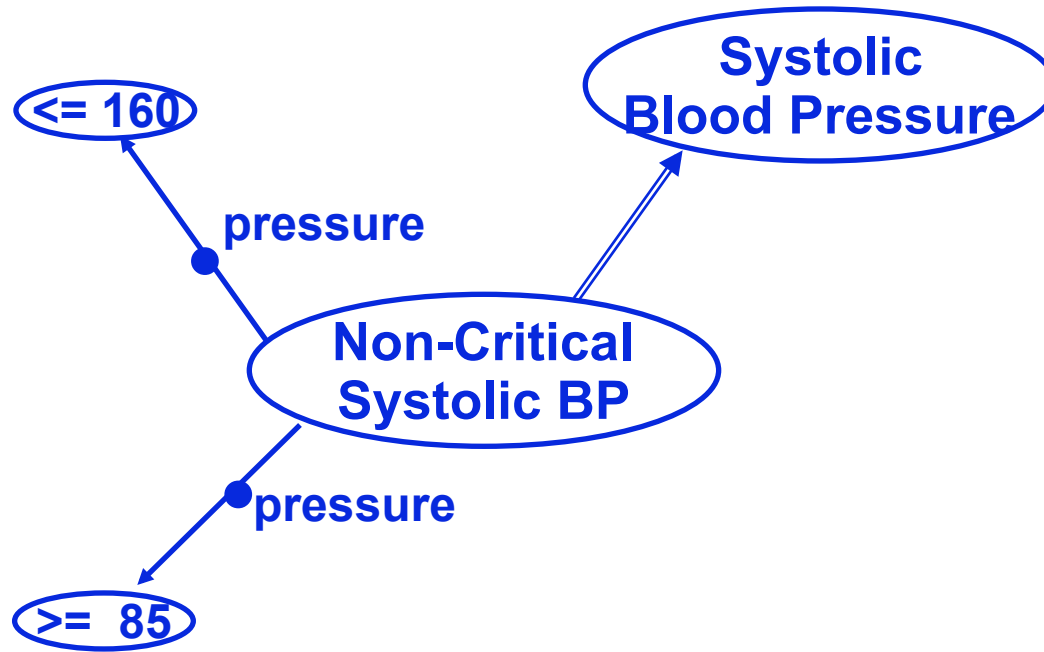
- *The concepts and relations form the terminology. It is the domain-specific language. This is often called the TBox.*
- *Assertions are domain facts. They are made about individuals called instances. This is often called the ABox.*
Assertions use the terminology of the TBox.
- *Instances can belong to concepts and participate in relations.*

Loom Supports Two Languages

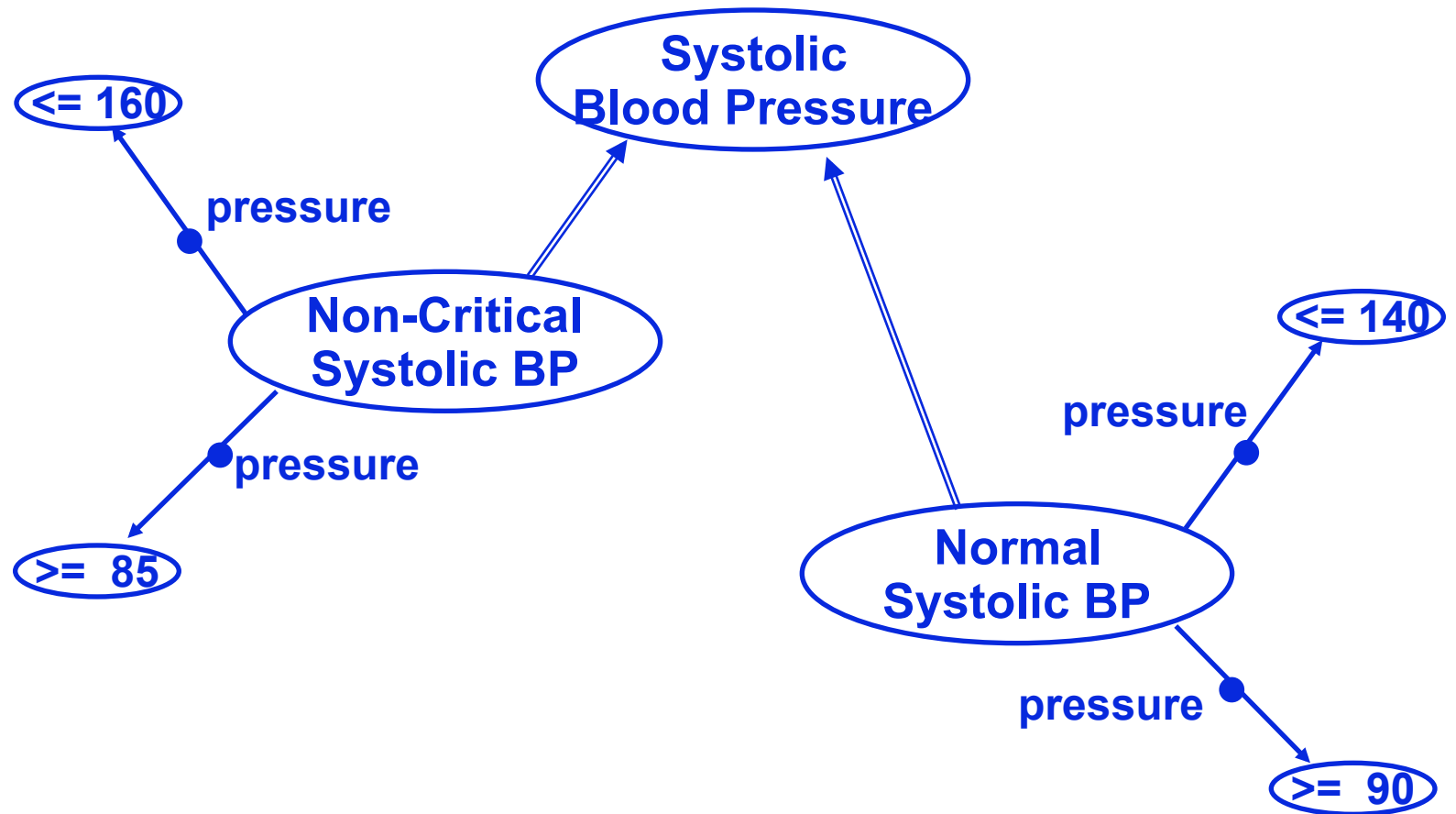


- *The definition language is used to define terminology. The definitions of concepts and relations are written using this language. The definition language is variable-free.*
- *The query language is used to write questions that are matched against the knowledge base. Queries can be yes/no questions or can request the retrieval of matching instances. The query language uses variables identified with a leading question mark (?).*
- *Assertions are made using the query language (but all variables must be bound)*

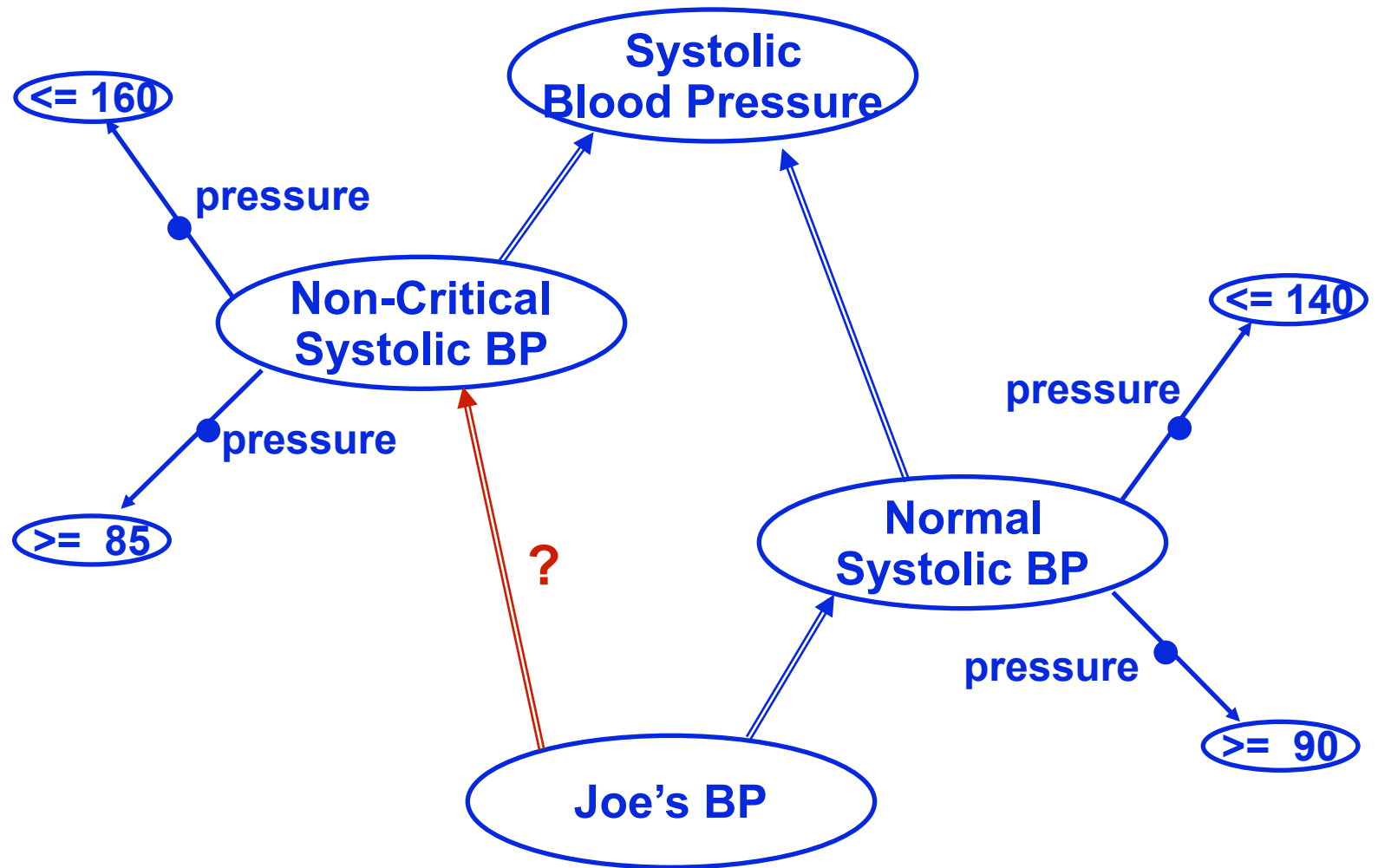
A Non-Critical Blood Pressure is “a Systolic B.P. between 85 and 160.”



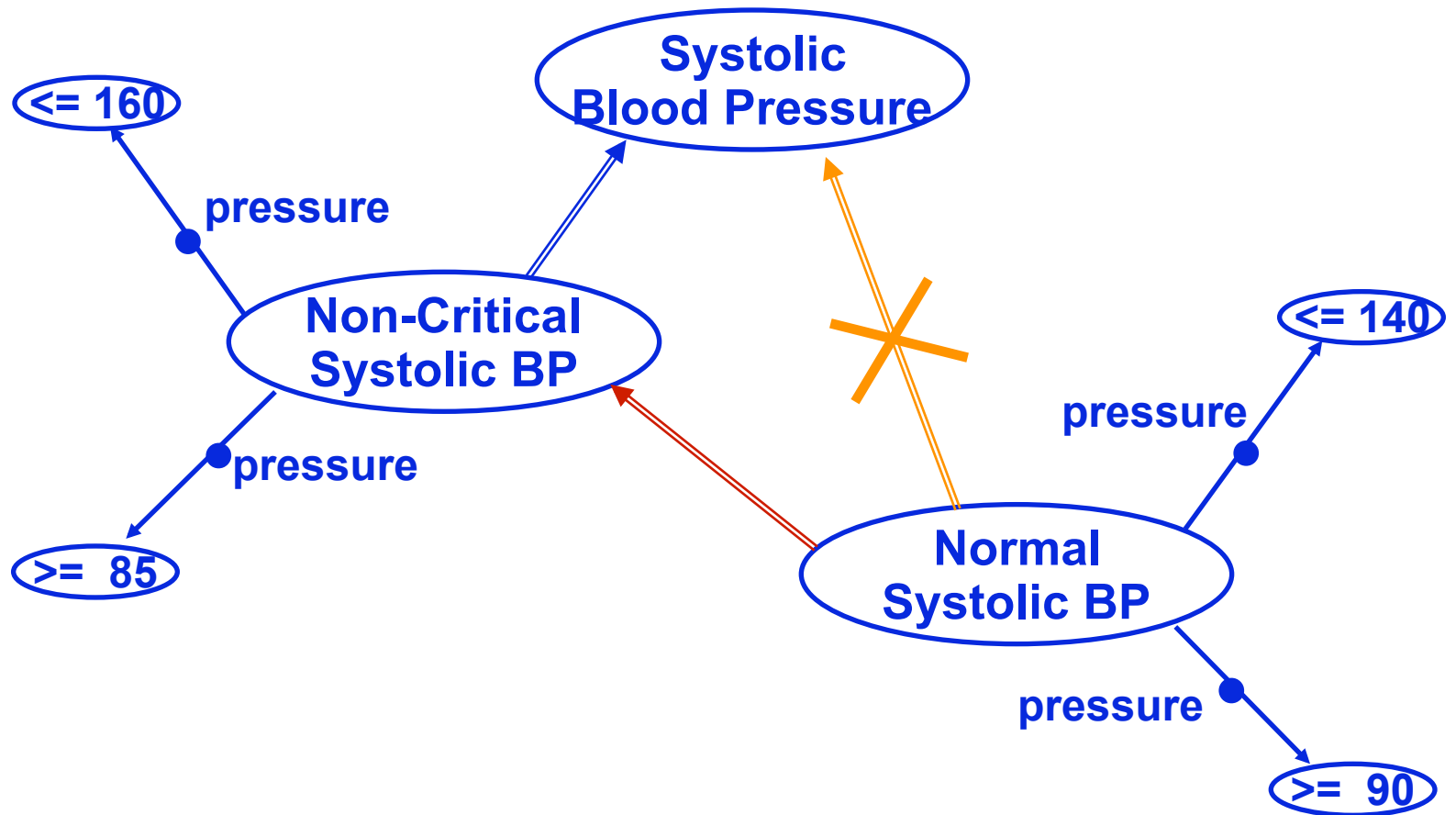
Normal Systolic B.P. is “a Systolic B.P. between 90 and 140.”



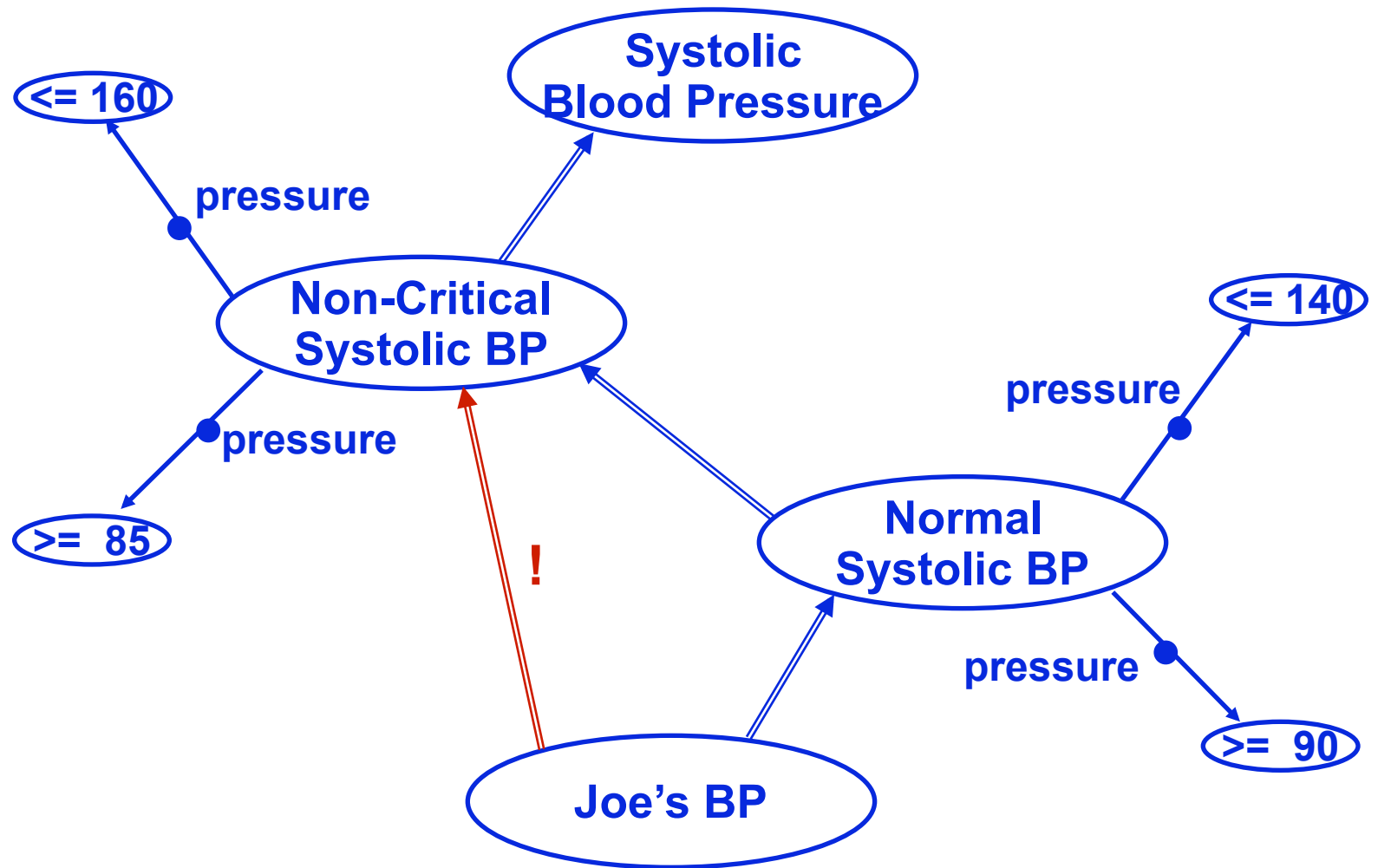
If Joe's BP is Normal is it also Non-Critical?



Concept Classification Infers Normal BP is Subsumed by Non-Critical BP



With Classified Concepts the Answer is Easy to Compute



TBox: *Syntax for Definitions*

Concept Definitions

```
(defconcept <name>  
  :is <definition>)
```

```
(defconcept <name>  
  :is-primitive <definition>)
```

Relation Definitions

```
(defrelation <name>  
  :is <definition>  
  :domain <domain>  
  :range <range>  
  :arity <integer>)
```



Defconcept *:and, :or, :not*

Concepts Defined in Terms of Others

(:and <concept> <concept> ...)

*(defconcept Slave-Boy
 :is (:and Slave Boy))*

*(defconcept Major
 :is (:or Engineering
 Liberal-Arts
 Sciences))*

(defconcept Male :is (:not Female))

*Note that “slave” subsumes “slave-boy” but that
“major” subsumes “sciences”*



Defconcept

:at-least, :at-most, :exactly

Number Restrictions on Relations

```
(:at-least <number> <relation>)
```

```
(defconcept 4-Door  
  :is (:and Car  
        (:exactly 4 has-door)))
```

```
(defconcept Parent  
  :is (:and Person  
        (:at-least 1 has-child)))
```

```
(defconcept Parent2  
  :is (:and Person  
        (:at-least 2 has-child)))
```

“Parent” subsumes “Parent2”



Defconcept *:all, :some*

Range Restrictions on Relations

```
(:all <relation> <concept>)
```

```
(defconcept Parent-of-Girls
  :is (:and Person
        (:at-least 1 has-child)
        (:all has-child Female)))
```

```
(defconcept Parent-with-Son
  :is (:and Person
        (:some has-child Male)))
```

“:some” implies “:at-least 1”

“:all” does not imply “:at-least 1”

Booby Trap!



Defconcept *:the*

Combination of “:exactly 1” and “:all”

(:the <relation> <concept>)

```
(defconcept Exclusive-Ford-Dealer
  :is (:and Business
        (:the sells Ford)))
```



```
(defconcept Exclusive-Ford-Dealer
  :is (:and Business
        (:exactly 1 sells)
        (:all sells Ford)))
```



Defconcept *:filled-by, :not-filled-by*

*Restricts relations to have specific instance fillers
(or non-fillers)*

```
(:filled-by <relation> <instance> ...)
```

```
(defconcept USC-Employee  
  :is (:and Person  
       (:filled-by employer USC)))
```

*“USC” is an instance, which will be created by
Loom if necessary.*

```
(defconcept Upperclassman  
  :is (:and Person  
       (:not-filled-by  
        college-year 1 2)))
```



Defconcept

:same-as, :subset

Restrictions on the values of relations

```
(:same-as <relation> <relation>)
```

```
(defconcept In-Town-Worker
  :is (:and Person
        (:same-as work-location
                   residence)))
```

An “in-town-worker” has a work location that is the same value as the residence.

```
(defconcept Contented-Worker
  :is (:and Person
        (:subset work-assignments
                  interests)))
```



Defconcept *:relates, comparisons*

Arbitrary relations between role fillers

```
(:relates <relation> <relation> ...)
```

```
(defconcept Socially-Linked-to-Boss
  :is (:and Person
        (:relates Brother
                   Best-Friend
                   Boss)))
```

Special cases for numeric comparisons

```
(defconcept Oversubscribed-Course
  :is (:and Course
        (> course-participants
           course-size)))
```



Defconcept :satisfies

More expressive escape. “:satisfies” introduces variables and allows more expressive statements



```
(:satisfies <variable> <query>)
```

```
(defconcept Lender  
  :is (:satisfies (?x)  
          (:exists (?z)  
                (owes-money-to ?z ?x))))
```

Drawback: Loom can't do as much reasoning about subsumption.

Tip: Rewrite to Use Specialized Forms

Defconcept Qualified Restrictions :at-least, :at-most, :exactly

Qualified Number Restrictions on Relations

`(:at-least <number> <relation> <concept>)`

```
(defconcept corporation
  :is (:and Business-Entity
        (:exactly 1 employee
                 President)))
```

```
(defconcept Parent-of-son
  :is (:and Person
        (:at-least 1 child Male)))
```

```
(defconcept Parent-of-son2
  :is (:and Person (:some child Male)))
```

“Parent of son” is the same as “Parent of son 2”



Defconcept Qualified Restrictions :all

Qualified Range Restrictions on Relations

(:all <relation> <concept> <concept>)

```
(defconcept Unenlightened-Company
  :is (:and Company
        (:all employee Male
              Supervisor)))
```

A company, all of whose employees who are supervisors are Male.

“Supervisor” qualifies “employee” and limits the set to which the “Male” restriction applies.

Nothing is said about non-Supervisor employees



Defconcept Qualified Restrictions

:all

Contrast

```
[1] (defconcept Unenlightened-Company
      :is (:and Company
             (:all employee Male
                    Supervisor))))
```

```
[2] (defconcept Unenlightened-Company2
      :is (:and Company
             (:all employee
                    (:and Male Supervisor))))
```

[1] can have female employees who are not supervisors. [2] has no female employees

[1] can have employees who are not supervisors. All of [2]'s employees must be supervisors.



Defconcept Keywords

:partitions

The name of a partition, the members of which divide the concept into disjoint sub-concepts.

:exhaustive-partitions

A partition that is collectively exhaustive.

:in-partition

Member of a partition.



Defconcept Characteristics

■ *Values of the keyword :characteristics*

:open-world, :closed-world

Declares the concept to use open or closed world semantics. Closed world semantics implies failure-as-negation. In other words, closed world means all concept members are known.

:monotonic, :perfect

Assertions won't be retracted. For :perfect, no subsequent assertions either.



Defrelation

:and

Combination of relations. Fillers must satisfy all relations in the conjunction.

```
(:and <relation> <relation> ...)
```

```
(defrelation co-worker-friend  
  :is (:and friend co-worker))
```

Note: Relations cannot use “or”.



Defrelation

:domain, :range

Restrictions on the domain or range of a relation

(:domain <concept>)

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

```
(defrelation mother-of
  :is (:and child
        (:domain female)))
```



Defrelation

:inverse

Defines the inverse relation

```
(:inverse <relation>)
```

```
(defrelation parent) ; primitive
```

```
(defrelation child  
  :is (:inverse parent))
```

A relation cannot be defined as its own inverse using this syntax, since that would be a circular definition. (See :symmetric later)



Defrelation

:compose

Composition is used to chain relations

```
(:compose <relation> <relation> ...)
```

```
(defrelation grandfather  
  :is (:compose parent father))
```

```
(defrelation great-grandfather  
  :is (:compose parent  
          parent  
          father))
```



Defrelation

:satisfies

Escape to allow more complicated descriptions of relations by introducing variables

```
(:satisfies <variables> <query>)
```

```
(defrelation owns-same-stock  
  :is (:satisfies (?x ?y)  
          (:exists (?z)  
                (:and (Stock ?z)  
                       (owns ?x ?z)  
                       (owns ?y ?z))))))
```



Defrelation

:domain and :range constraints

The domain and range of a relation can be specified as constraints instead of definitions.

```
(defrelation owns-stock
  :domain Person
  :is (:and owns (:range stock)))
```

Non-definitional
constraint

Part of Definition

Using constraints can be a way of avoiding circular definitions.

N-Ary Relations

Loom relations do not need to be binary, but must have a fixed arity

`(defrelation love-triangle :arity 3)`

`(defrelation square :arity 4)`



A “love-triangle” is a relationship among three persons. A “square” is a relation between four geometric points.

Relation Characteristics

:single-valued, :multiple-valued

Determines how many fillers allowed.

:symmetric

The relation is its own inverse.

:commutative

The order of the first N-1 arguments doesn't matter.

:open-world, :closed-world

:monotonic, :perfect

Assertions won't be retracted. For :perfect, no subsequent assertions either.



Using `:function` and `:predicate`

The `:function` and `:predicate` arguments to `defconcept` and `defrelation` allow the specification of arbitrary decision parameters

```
(defconcept odd-Number
  :is-primitive Number
  :predicate oddp)
```

*Definition used
for subsumption*

*Predicate determines
membership*

```
(defrelation plus
  :arity 3
  :function ((x y) (+ x y)))
```

*Function
arguments*

*Function
body*



Using :function and :predicate

Either function names or lambda expressions (without the “lambda”) can be used.

The function or predicate must be a complete test for the concept or relation. The definition is used only for subsumption computation.

Lisp functions used as :predicates have the same arity as the concept (1) or relation.

Lisp functions used as :functions have take one fewer arguments than the arity of the concept or function.

For example, concept :functions take no arguments.



Assertions

Assertions can be for concept membership:

```
(tell (Dog Fido) (Man Jim)
      (Politician Bush))
```



Assertions can be relation (role) fillers:

```
(tell (owner Fido Jim)
      (supports Jim Bush))
```

Assertions with :about

Many assertions about a single individual gets repetitious:

```
(tell (Man Jim) (Professor Jim)
      (Republican Jim) (age Jim 45)
      (department Jim Biology))
```

:about syntax shortens this by

```
(tell (:about Jim
        Man Professor Republican
        (age 45)
        (department Biology))
```

The subject of the :about clause is not present in any of the assertion forms.



Additional Assertions with :about

Certain assertions can only be made using :about syntax. These are descriptive assertions rather than ground facts:

```
(tell (:about Jim
      (:at-least 2 brother)
      (:at-most 1 job)))
```



Queries

Queries can be Yes/No questions:

```
(ask (Professor Jim))
```

```
(ask (:and (Professor Jim)
           (brother Jim Fred)
           (:not (brother Jim Bob))))
```

Queries can retrieve matching instances:

```
(retrieve ?prof (Professor ?prof))
```

```
==> (professor-1 prof-2 ...)
```

```
(retrieve (?prof) (Professor ?prof))
```

```
==> ((professor-1) (prof-2) ...)
```



Queries (cont.)

Multiple variables can also be used:

```
(retrieve (?x ?y)
  (:and (married ?x ?y) (Happy ?y)))
```

```
(retrieve (?x ?y ?z)
  (:and (friend ?x ?z)
    (friend ?y ?z)))
```



Query Language

:and, :or

:not

The negation can be proven.

:fail

The positive cannot be proven.

:exists, :for-all

Introduces an existential or universal variable.

:same-as

:collect, :set-of

Introduce sub-query that returns a set.



Query Language Examples

Retrieve friends of friends:

```
(retrieve (?x ?y)
  (:exists (?z)
    (:and (friend ?x ?z)
           (friend ?z ?y))))
```

Retrieve people with a mutual friend

```
(retrieve (?x ?y)
  (:exists (?z)
    (:and (friend ?x ?z)
           (friend ?y ?z))))
```



Query Language Examples (cont.)

Retrieve people all of whose brothers are older

```
(retrieve (?x)
  (:for-all (?z)
    (:implies (brother ?x ?z)
      (> (age ?z) (age ?x)))))
```

Retrieve people with more than 4 siblings:

```
(retrieve ?x
  (:about ?x (:at-least 5 sibling)))
```



Query Language Examples (cont.)

Retrieve the number of people with happy spouses:

```
(retrieve ?n
  (count (:collect (?sp)
    (:exists (?y)
      (:and (person ?y)
        (spouse ?y ?sp)
        (happy ?sp))))
    ?n))
```



Getting Started

Start the Loom system.

- *Create a new context (theory) and establish a Lisp package:*

```
(loom:use-loom "PROJECT" )
```

- *Creates a package named “PROJECT”*
- *Creates a theory context named “PROJECT-THEORY”*



Examining the Knowledge Base

Printing concepts, instances, relations

```
(pc <conceptName>), (pi ...), (pr ...)
```

```
(pc parent) ==>
```

```
(defconcept Parent
```

```
  :is (:and Person
```

```
    (:at-least 1 child)))
```

Finding concepts, instances, relations

```
(fc <conceptName>), (fi ...), (fr ...)
```

```
(fc parent) ==> |C|Parent
```

```
(fr child) ==> |R|child
```

```
(fi Jim) ==> |I|JIM
```



Starting Over

Commands that clear the state of the knowledge base and allow a new start:

■ *Clearing the current workspace:*

`(clear-context)`

■ *Clearing all workspaces. Restoring Loom to its initial state:*

`(initialize-network)`

■ *Clearing instances in all contexts*

`(initialize-instances)`

