Modeling in a Medical Domain

Basic Concepts
Using Recognition
More Sophisticated Relations
Methods, Actions and Production Rules
Basic Concepts—Personnel

Primitive Concepts

(defconcept person)
(defconcept official-responder :is-primitive person)

Closed World Concepts

(defconcept medical-person :is-primitive person
                          :characteristics :closed-world)

Defined Concepts

(defconcept medic :is (:and official-responder medical-person))
Basic Concepts—Personnel (alternate)

Primitive Concepts and Relations

(defconceput person)
(defrelation training)

Defined Concepts

(defconceput medical-person
 :is (:and person
     (:some training medical)))

(defconceput emergency-responder
 :is (:and person
     (:some training emergency)))

(defconceput medic
 :is (:and emergency-responder
     medical-person))
Basic Concepts—Injury

**Full set**

(defconcept injury
  :is (:one-of ‘airway ‘breathing ‘circulation ‘neurologic-disability ‘exposure ‘head ‘neck ‘chest ‘other))

**Subsets (subsumption calculated automatically)**

(defconcept primary-injury
  :is (:one-of ‘airway ‘breathing ‘circulation ‘neurologic-disability ‘exposure))

(defconcept secondary-injury
  :is (:one-of ‘head ‘neck ‘chest ‘other))
Basic Concepts—Injury (alternate)

Subsets

(defconcept primary-injury
  :is (:one-of 'airway 'breathing
       'circulation 'neurologic-disability
       'exposure))

(defconcept secondary-injury
  :is (:one-of 'head 'neck 'chest 'other))

Union specified by "or"

(defconcept injury
  :is (:or primary-injury
       secondary-injury))
Basic Relations—Injuries

Primitive

(defrelation injuries
    :characteristics  :closed-world)

Defined (range restricted)

(defrelation primary-injuries
    :is (:and injuries
        (:range primary-injury)))

(defrelation secondary-injuries
    :is (:and injuries
        (:range secondary-injury)))

Closed world by inheritance
Basic Concepts—Casualties

Defined by number restrictions

(defconcept casualty
 :is (:and person
 (:at-least 1 injuries)))

(defconcept critical-casualty
 :is (:and person
 (:at-least 1
 primary-injuries)))

Negated concepts can also be formed

(defconcept non-critical-casualty
 :is (:and casualty
 (:at-most 0
 primary-injuries)))

Implies \( \geq 1 \) injuries

Needs closed world!
Negated concepts can also be expressed

(defconcept helper
  :is (:and person
       (:at-most 0 injuries)
       (:not medical-person)))

Recall that “medical-person” was declared to be closed world

This is crucial to reasoning with “:not”

Without the closed world assumption, any individual not explicitly asserted to not be a medical-person could conceivably be one.

This uncertainty would inhibit recognition.
Using Recognition

Loom can recognize when assertions about individuals causes them to fulfill definitions

This allows information to be added as it becomes available

The logical consequences of the existing information is always maintained

Example:

```lisp
(tell (:about p2 Person
    (injuries 'airway)
    (injuries 'other)))
```

p2 is no longer a “Helper”
p2 is now a “Casualty” and a “Critical Casualty”
Some relations can involve sophisticated calculations

Loom provides a method for defining a relation that is the result of a calculation rather than an assertion

: predicate indicates a test for the relation

: function indicates a generator for the relation

Such relations are assumed to be single-valued.
Sophisticated Relations—Geography

**We need to associate a location with individuals**

(defrelation location
  :characteristics :single-valued)

**We want to calculate distance between locations**

(defrelation distance-from-locations
  :arity 3
  :function grid-distance)

**The auxiliary function does the calculation**

(defun grid-distance (loc1 loc2)
  (sqrt (* (- (loc-x loc2)
            (loc-x loc1)))
    ...
    )))
Sophisticated Relations—Geography

We also want to find the distance between individuals

(defrelation distance :arity 3
    :is (:satisfies (?x ?y ?d)
        (distance-from-location
            (location ?x)
            (location ?y)
            ?d)))

Direction can be handled analogously

Loom uses computed relations in backward chaining mode only—Information is not propagated forward.
Sophisticated Relations—Inference Direction

Concepts and relations can be defined in terms of computed relations:

\[
\text{(defrelation in-range :is (:satisfies (?x ?y) (< (distance ?x ?y) (range ?x))))}
\]

This relation can be queried, but it will not propagate information forward.

\[
\text{(ask (in-range helo-1 Hospital))}
\]

\[
\text{(retrieve ?c (:and (casualty ?c) (in-range helo-1 ?c)))}
\]
Sophisticated Relations—Alternate Inheritance

Problem: How can we automatically update the locations of individuals being transported by a vehicle?

• Each time the vehicle moves, update all passenger locations
• Determine the passenger location based on the vehicle location
Sophisticated Relations—Transitive Closures

Base relation “contained-in” is single-valued

(defrelation contained-in
  :characteristics :single-valued)

Transitive Closures

(defrelation contained-in* :is
  (:satisfies (?x ?y)
    (:exists (?z)
      (:and (contained-in ?x ?z)
        (contained-in* ?z ?y)))))

Note the recursive definition
Transitive Relation Idiom

Standard Definition of a Transitive Relation

$R^*$ Based on the Relation $R$:

\[
(\text{defrelation } R^* \text{ :is}
\begin{align*}
   & (\text{satisfies } (?x \ ?y) \\
   & (\text{exists } (?z) \\
   & (\text{and } (R \ ?x \ ?z) \\
   & (R^* \ ?z \ ?y )))
\end{align*}
\)
Sophisticated Relations—Following a Transitive Link

Base relation “position” is single-valued

(defrelation position
 :characteristics :single-valued)

Transitive Closures

(defrelation position*: is
 (:satisfies (?x ?y)
   (:exists (?z)
     (:and (contained-in* ?x ?z)
       (position ?z ?y))))))

The transitive link is followed in this relation to find a ?z with a position. Note that this will find ALL such ?z’s!
Sophisticated Relations—Alternate Inheritance Path

Base relation requires inverse

(defrelation contained-in)
(defrelation contains
  :is (:inverse contained-in))

“position” inherits via “contained-in”

(defrelation position
  :inheritance-link contained-in)

This allows the creation of meaningful “part-of” hierarchies, with inheritance of appropriate properties.
Methods, Actions and Production Rules

Methods specify procedures that are specialized by Loom queries.

Loom methods have a richer vocabulary than CLOS methods.

Actions specify properties of methods such as selection rules.

Production rules trigger on changes in the state of the knowledge base.

Production rules allow a reactive or event-driven style of programming.
Example Method

(defmethod assess-casualty (?medic ?casualty)
  :situation
    (:and (Medic ?medic) (casualty ?casualty))
  :response
    ((format t "~%Medic ~8A examines ~8A" ?medic ?casualty)
      (tell (examined ?casualty 'yes))))
Example Method

(defmethod assess-casualty (?medic ?casualty)
  :situation
  (:and (Medic ?medic) (casualty ?casualty))
  :response
  ((format t "~%Medic ~8A examines ~8A"
          ?medic ?casualty)
    (tell (examined ?casualty 'yes))))

Query determines applicability
Example Method

(defmethod assess-casualty (?medic ?casualty)
  :situation
  (:and (Medic ?medic) (casualty ?casualty))
  :response
  ((format t "~A Medic ~A examines ~A" ?medic ?casualty)
    (tell (examined ?casualty 'yes)))
)

Query determines applicability
Lisp code in the response
Example Method

(defmethod assess-casualty (?medic ?casualty)
  :situation
  (:and (Medic ?medic) (casualty ?casualty))
  :response
  ((format t "~%Medic ~8A examines ~8A"
           ?medic ?casualty)
   (tell (examined ?casualty 'yes))))

Query determines applicability
Lisp code in the response
Loom assertions in the response
Methods Can Be Performed Immediately or Scheduled

To call a method immediately use the “perform” function

To schedule a method for execution use the “schedule” function

Scheduled methods can be given a priority (the built-in priorities are :high and :low)

Methods are performed the next time there is a knowledge base update (ie, “tellm”)

Methods are executed in accordance with the priority

Within a priority methods are executed in the ordered they were scheduled
The :situation Determines Method Applicability

(defmethod treat-patient (?medic ?patient)
  :situation (:and (medic ?medic)
               (critical-casualty ?patient)
               (examined ?patient 'no))
  :response
  ((schedule (goto ?medic ?patient)
              :priority :high)
   (schedule (assess-casualty ?medic ?patient)
             :priority :high)))

(defmethod treat-patient (?medic ?patient)
  :situation (:and (medic ?medic)
               (non-critical-casualty ?patient)
               (examined ?patient 'no))
  :response
  ((schedule (goto ?medic ?patient)
              :priority :low)
   (schedule (assess-casualty ?medic ?patient)
             :priority :low)))
More on Choosing a Method

Often several methods are applicable to a particular situation. “defaction” forms can specify how to resolve ambiguities

- Choose all applicable methods
- Choose the most specific method
- Choose the last method defined
- Choose a method at random
- Issue a warning
- Cause an error

These resolution methods can be combined and are used in order
Example of Combined Resolution

If both secondary and primary injuries exist, :most-specific does not give a single result

Multiple selection criteria resolves the problem

(defaction treat-injury (?medic ?patient)
  :filters (:most-specific :select-all))

The criteria are prioritized

Avoids the need to define methods for all combinations of concepts

Method treat-injury has a definition for all three concepts
Methods Can Also Have Query-Based Iteration

Finding all casualties reported on Medic’s clipboard

(defmethod locate-casualties (?medic)
  :situation (medic ?medic)
  :with (casualties
    (clipboard ?medic) ?c)
  :response (...))

The response is executed once for each ?c that the query in the :with clause finds.

In the response ?medic is bound to the method argument and ?c to a particular casualty reported on the medic’s clipboard.
Production Rules Trigger on Changes in the Knowledge Base

The changes can be additions to the KB (:detects)
This applies to relation additions and concept additions

The changes can be deletions from the KB (:undetects)
This applies to relation deletions and concept deletions

The change can be in a relation value (:changes)
Noticing a New Injury

(defproduction notice-injury
 :when ((:and (:detects (injury ?self ?i))
               (phone ?i ?phone)))
 :do ((perform (report-injury ?phone ?i))))

The :detects clause triggers the production

The additional query (phone ?i ?phone) is a guard clause and also provides an additional variable binding

The variables from the :when clause are bound for the execution of the production body. In this example, the injury is reported using a phone by calling the method “report-injury”. A different method could be used if a radio were available.