Frequently Asked Questions

- Why Don’t Instances Get Recognized?
- Use of “:all”
- Use of “:for-all”
- Compiling Loom Code
- Combining Number Restrictions
- Why Doesn’t (:exactly 1 R) Clip?
- Why Aren’t Concepts Disjoint?
- My Concept Name Changed!
- Multiple Value Roles & Defaults
- Inverse Relations
- Loom vs. CLOS
FAQ: Why Aren’t Instances Recognized?

- Why don’t instances get recognized as belonging to a concept when I assert them?
  - Time needs to be advanced: Use (tellm) or (new-time-point)
  - Lite instances are being used instead of classified instances: Use (creation-policy :classified-instance)

- How do I tell if I have classified or lite instances?
  - Use the function (creation-policy).
  - Subtle: Look at the printed representation:

Note case of letter “i”

- |i|Fred is a lite instance;
- |I|Barney is classified
**FAQ: Use of “:all”**

- **Value restrictions using :all**
  - `(defrelation R)`
  - `(defconcept C)`
  - `(defconcept C-all :is (:and C (:all R C)))`

- **Assertions**
  - `(tell (C c1) (C c2) (R c2 c1))`

- **Query**
  - `(retrieve ?x (C-all ?x))`
FAQ: Use of “:all”

- **Value restrictions using :all**
  - `(defrelation R)`
  - `(defconcept C)`
  - `(defconcept C-all :is (:and C (:all R C)))`

- **Assertions**
  - `(tell (C c1) (C c2) (R c2 c1))`

- **Query**
  - `(retrieve ?x (C-all ?x)) ==> NIL`

- **Why NIL? Because R is not closed, therefore other unknown R fillers could exist which are not Cs.**
FAQ:
Use of “:all”

- **Value restrictions using :all**
  - (defrelation R :characteristics :closed-world)
  - (defconcept C)
  - (defconcept C-all :is (:and C (:all R C))

- **Assertions**
  - (tell (C c1) (C c2) (R c2 c1))

- **Query**
  - (retrieve ?x (C-all ?x)) => (c1 c2)

- **Why both of them?** How can all of c1’s R fillers be Cs if c1 doesn’t have any Rs? Since there are no such fillers, it is trivially fulfilled.
FAQ: Use of “:all”

- **Value restrictions using :all**
  - `(defrelation R :characteristics :closed-world)
  - `(defconcept C)
  - `(defconcept C-all :is (:and C (:all R C) (:at-least 1 R))

- **Assertions**
  - `(tell (C c1) (C c2) (R c2 c1))

- **Query**
  - `(retrieve ?x (C-all ?x))⇒(c2)

- **The :at-least 1 restriction expresses what we really mean!**
FAQ:
Proper use of “:for-all”

- Loom’s universal quantification is does not have a type restriction built in. The consequence is that special syntax is needed inside :for-all constructs
  - (defconcept C)
    (defrelation R)
  - (retrieve ?x
    (:for-all (?z)
      (:and (R ?x ?z) (C ?z)))))

- This will produce an error message
  - To successfully evaluate a universally quantified clause, the clause must contain at least one negated term. In this case, the clause
    (|R|R ?X ?Z)
    does not.
FAQ: Proper use of “:for-all”

Logically speaking, the query

• (retrieve ?x
  (:for-all (?z)
    (:and (R ?x ?z) (C ?z))))

is extremely unlikely to be satisfied if ?z ranges over all individuals in the knowledge base. The query must be formulated to restrict the value of ?z

• (retrieve ?x
  (:for-all (?z)
    (:implies (R ?x ?z) (C ?z))))

or equivalently

• (retrieve ?x
  (:for-all (?z)
    (:or (:not (R ?x ?z)) (C ?z)))))
FAQ: Compiling Loom Code

- Loom performs code generation and optimization during macro-expansion of the forms “tell”, “forget”, “ask” and “retrieve”
- The proper expansion of the code requires that all definitions referenced in the form be available
  - Definition files must therefore be loaded before assertion or query files are compiled
  - If definitions are in the same file, then they must be enclosed by an “eval-when” form specifying compile time evaluation. The last form in the eval-when should be a call to “finalize-definitions”
FAQ: Compiling Loom Code

- Certain redefinitions (such as changing a relation from single to multipleneral rule, all code which uses definitions should be recompiled when those definitions change.)
FAQ: Combining Number Restrictions

- **Loom has a (limited) ability to reason about number restrictions and their combinations**

- **Example**
  - (defrelation R)
  - (defconcept C)
  - (defconcept -C :is (:not C))
  - (defconcept A :is (:and C (:at-least 2 R C) (:at-least 2 R -C)))

  - (tell (A a1))
  - (ask (:about a1 (:at-least 2 R))) ==> T
  - (ask (:about a1 (:at-least 4 R))) ==> T
  - (ask (:about a1 (:at-least 5 R))) ==> NIL

- **Loom knows C and -C are disjoint, so there must be at least 4 fillers of R on any A.**
FAQ: Combining Number Restrictions

- Inference is not complete in all cases

Example

• (defrelation R)
  (defconcept C)
  (defconcept -C :is (:not C))
  (defconcept A
    :is (:and C (:at-least 2 R C)
         (:at-most 3 R)))

• (tell (A a1))

• (ask (:about a1 (:at-most 3 R)))  ==> T
  (ask (:about a1 (:at-most 3 R C)))  ==> T
  (ask (:about a1 (:at-most 1 R -C)))  ==> NIL
  (ask (:about a1 (:at-most 3 R -C)))  ==> T

- Loom cannot infer the upper limit on -C fillers based on the upper limit on R and the lower limit on fillers of type C
FAQ: Why Doesn’t (:exactly 1 R) Clip?

- **Number restriction in concept definition**
  - (defrelation R)
  - (defconcept C :is-primitive (:exactly 1 R))

- **Assertions**
  - (tell (C c1) (R c1 3))
  - (tell (R c1 4))

- **Query**
  - (retrieve ?x (R c1 ?x)) ==> (3 4)

- The assertion of C and of the two role fillers have equal weight. There is no logical preference for one over the other.
FAQ:
Why Doesn’t (:exactly 1 R) Clip?

- To get clipping the relation itself must be asserted to be single-valued:
  - (defrelation R :characteristics :single-valued)

- Or Loom must be able to infer that R must be single-valued:
  - (defrelation R :domain C)
  - (defconcept C :is-primitive (:exactly 1 R))
  - Since the domain of R is C all instances that have R fillers must also be of type C. Since C only has 1 R, R must be single-valued.
FAQ:
Why Aren’t Concepts Disjoint?

Example

• (defrelation R :attributes :closed-world)
  (defconcept A)
  (defconcept B)
  (defconcept C :is (:and A (:all R A)))
• (tell (A a1) (B b1) (R a1 b1))
• (ask (C a1) ==> NIL (Good!))
  (ask (:not (C a1)) ==> NIL (Huh?)

Why can’t Loom conclude that a1 is not a C?

Because concepts are not disjoint by default.

Just because b1 is a B, it doesn’t preclude it from being an A as well.
FAQ: Why Aren’t Concepts Disjoint?

Example

- (defrelation R :attributes :closed-world)
  (defconcept A)
  (defconcept B)
  (defconcept C :is (:and A (:all R A)))

Alternate Fixes

- (defconcept A :implies (:not B))
- (defconcept A :characteristics :closed-world)
  - Make A and B members of a partition.

Solution

- (tell (A a1) (B b1) (R a1 b1))
- (ask (C a1) ==> NIL
  (ask (:not (C a1)) ==> T
FAQ:
My Concept Name Changed!

Consider these definitions
• (defrelation R)
  (defconcept A)
  (defconcept B :is (:and A (:some R A)))
  (defconcept C :is (:and A (:some R A)))

Note identical definitions of B and C.
• (tell (C c1))
  (get-types `c1) ==> (|C|B |C|A |C|THING)

What happened to the concept C?
FAQ: My Concept Name Changed!

- Consider these definitions
  - (defrelation R)
  - (defconcept A)
  - (defconcept B :is (:and A (:some R A)))
  - (defconcept C :is (:and A (:some R A)))

- Note identical definitions of B and C.
  - (tell (C c1))
    (get-types 'c1) ==> (|C|B |C|A |C|THING)

- What happened to the concept C? It merged!
  - (ask (C c1)) ==> T
    (find-concept 'c) ==> |C|C

- Loom can find and use it under either name, but only one name is used for display.
FAQ: Multiple Value Roles & Defaults

- Definitions
  - (defrelation R)
  - (defconcept C :defaults (:filled-by R 5))

- Assertions
  - (tell (C c1) (C c2) (R c2 4))

- Queries
  - (retrieve ?x (R c1 ?x))
  - (retrieve ?x (R c2 ?x))
FAQ: Multiple Value Roles & Defaults

- **Definitions**
  - (defrelation R)
  - (defconcept C :defaults (:filled-by R 5))

- **Assertions**
  - (tell (C c1) (C c2) (R c2 4))

- **Queries**
  - (retrieve ?x (R c1 ?x)) => (5)
  - (retrieve ?x (R c2 ?x)) => (5 4)
FAQ:
Multiple Value Roles & Defaults

- Problem: You can’t easily get rid of default fillers on multiple-value roles
- Solution: Consider only using them on single-value roles
FAQ: Multiple Value Roles & Defaults

- Problem: You can’t easily get rid of default fillers on multiple-value roles

- Solution: Consider only using them on single-value roles

- Non-solution: Use forget to get rid of default value. Doesn’t work because forget just withdraws support for assertions. Loom can prove the value a different way (default inference)

- Solution 2: Assert the negation. Note that this is very clumsy and not our first choice recommendation
  • (tell (:not (R c2 5)))
FAQ:
My New Inverse Doesn’t Work!

If I define a new inverse relation, the old assertions don’t work properly:

(defrelation R)
(tell (R Fred Sue) (R Bill Sue))
(defrelation R-1 :is (:inverse R))
(retrieve ?x (R-1 Sue ?x)) => NIL

Why weren’t Bill and Fred returned?
FAQ: My New Inverse Doesn’t Work!

- If I define a new inverse relation, the old assertions don’t work properly:

  (defrelation R)
  (tell (R Fred Sue) (R Bill Sue))
  (defrelation R-1 :is (:inverse R))
  (retrieve ?x (R-1 Sue ?x)) => NIL

- Why weren’t Bill and Fred returned?
  - Loom implements inverse relations by explicitly asserting the inverse relation
  - Since R-1 did not exist when “R Fred Sue” was asserted, the inverse assertion was not made
FAQ:
Can’t Strings Have Inverses?

Why doesn’t the following work?
(defrelation Name)
(defrelation Name-of
  :is (:inverse Name))
(tell (R Sue "Sue Jones")) => Error
FAQ: Can’t Strings Have Inverses?

- Why doesn’t the following work?
  
  (defrelation Name)
  
  (defrelation Name-of :is (:inverse Name))
  
  (tell (R Sue “Sue Jones”)) => Error

- The inverse assertion can’t be made!
  
  • Built-in types (such as numbers, strings and symbols) cannot have assertions made about them.
  
  • The objects are too primitive to support assertions
  
  • Inverses are implemented as assertions
FAQ:
Loom vs. CLOS

- Loom has multiple slots with the same name
- Loom “type” hierarchies are determined structurally
- Loom relation names have significance in determining the type of objects
- Loom instances can have slots added on the fly without redefinition
- Loom has a query language