Overview

- Knowledge Representation (KR) Background
- Evolution and Issues
- PowerLoom® Language
  - Concept and Relation Language
  - Assertions
  - Rules
  - Example
- PowerLoom Application
- Conclusion
1) The idea of descriptive logics and how they differ from, say, systems like prolog.

2) A little on the evolution of PowerLoom.

3) Details of Powerloom: The Concept/Relation language, Assertions, retrievals, open/closed world semantics

4) Rules: Forward and Backward chaining, the many ways to express rules. How to invoke rules explicitly.

5) Classification: What it is and how it works in Powerloom. Do the rabies example (it's on my website at: http://www-scf.usc.edu/~csci561a/slides/rabies.plm

6) How a PowerLoom application looks (especially one written in Java)
Logic for Representation and Reasoning

300 B.C. 1800s

- All men are mortal
- Socrates is a man
- Therefore, Socrates is mortal

\[ \forall x \ (\text{Man}(x) \rightarrow \text{Mortal}(x)) \]
\[ \text{Man}(\text{Socrates}) \]
\[ \therefore \ \text{Mortal}(\text{Socrates}) \]

*Syllogism (Aristotle)*

*Predicate Calculus (Frege)*

Semantic Networks: Nodes and Links

```
animal
  ▼
   mammal
     ▼
       dog

  ▼
  sick animal
     ▼
       disease

  ▼
  medicine

  "A dog is a mammal"

  "A sick animal has a disease"

  "rabies is a disease"
```

```
is-a

"A dog is a mammal"

"A sick animal has a disease"

"rabies is a disease"
```
Semantic Networks: The Computer's View

Description Logic: Limited Understanding

• Subclass relations
  “A dog is a mammal”

• Structural description:
  • Cardinality, Fillers, Type restrictions
    “A sick animal has a disease”
Defining a “rabid dog”

animal

mammal

sick animal

has

disease

dog

rabies

has

rabid dog

Classification Concludes “sick animal”

animal

mammal

sick animal

has

disease

dog

rabies

has

rabid dog
Defining “rabid animal”

Classification Places Concept in Hierarchy
Description Logics

- Subsumption is the organizing and reasoning principle
  - Subset-of relation.
- Special language constructs for structural description
  - Classifier reasons about subsumption
  - Reasoning is based on structure of definitions
  - Limited language to allow tractable inference
    - (all R C)
    - (some R C)
    - (exactly n R)
    - ...

- Examples of description logics
  - KL-ONE, KRYPTON, Loom, Classic, OWL

Logic and Theorem Provers

- Reasoning based on logic
  - Theorem provers
  - Logic Programming (Prolog)

- PowerLoom combines logical reasoning with ideas from description logics
  - Prolog + additional logical inferences
  - Named concepts and definitions
  - First-order predicate calculus
### PowerLoom vs. Prolog

<table>
<thead>
<tr>
<th>Prolog</th>
<th>PowerLoom</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Horn clauses</td>
<td>• 1st order logic</td>
</tr>
<tr>
<td>• Closed world reasoning</td>
<td>• Open and closed world</td>
</tr>
<tr>
<td>• Backward chaining rules</td>
<td>• Backward and forward chaining rules</td>
</tr>
<tr>
<td>• Universal quantification</td>
<td>• Universal and existential</td>
</tr>
<tr>
<td>• Resolution theorem proving</td>
<td>• Deductive, specialist and other reasoning</td>
</tr>
<tr>
<td>• More efficient reasoner</td>
<td>• Relations are 1st class objects</td>
</tr>
</tbody>
</table>

### KR Issue: Expressivity and Tractability

- **Ideal Knowledge Representation System**
  1. Expressed language: You can say what you need to
  2. Sound reasoning: The reasoner doesn't make mistakes
  3. Complete reasoning: All allowed inferences are made
  4. Efficient: The answers are produced quickly (tractable algorithms)

- **Problem:** You can only have 3 of the above.
  - Two main schools of thought
    1. Sound, Complete & Tractable: Classic, OWL
    2. Expressive, Sound & Tractable: Loom, PowerLoom

- **PowerLoom is culmination of push for more expressivity**
KR Issue: Closed vs. Open World reasoning

- Closed World means the system knows all relevant facts
- Allows “negation as failure” reasoning
- Answers are either true or false
- Example: President Sample is in this lecture hall — false
- Database systems and Prolog are closed-world

- Open World means that there may be unknown facts
- Lack of proof does not mean false
- Answers are true, false or unknown
- Example: President Sample is on campus — unknown
- Many KR systems (including PowerLoom) are open world
  - PowerLoom can also do selectable closed world reasoning

PowerLoom Language
Logical Models 101

"Real" World

Terms represent entities:
- Joe
- car002

Predicates represent relations:
- owns

Sentences represent what is true in the world (facts):
- (Person Joe)
- (= (age Joe) 17)
- (Car car002)
- (owns Joe car002)
- (model car002 Ford)
- (not (rich Joe))

Rules define terms and represent domain regularities:
- (<=> (and (> (age ?x) 12) (< (age ?x) 20)) (Teenager ?x))
- (=> (and (Teenager ?x) (car ?y) (owns ?x ?y)) (happy ?x))

Facts + rules + inference derive concluded facts:
- (Teenager Joe)
- (happy Joe)

PowerLoom Representation Language

- First Order Logic base
- Syntax
- Declarative semantics
- Prefix notation

Example:

Facts:
- (person fred)
- (citizen-of fred germany)
- (national-language-of germany german)

Rules:
- (forall (?p ?c ?l)
  (=> (and (person ?p))
      (citizen-of ?p ?c)
      (national-language-of ?c ?l))
  (speaks-language ?p ?l)))
Definitions

- Terminology (relations, concepts) need to be defined before they are used via `defconcept`, `deffunction` & `defrelation`.

- Examples:
  ```
  (defconcept person)
  (defrelation married-to ((?p1 person) (?p2 person))
  (deffunction + ((?n1 number) (?n2 number)) :-> (?sum number))
  ```

- Advantage & Disadvantage
  - Allows certain amount of error checking (e.g., misspelled relations, argument type violations)
  - A bit more tedious and can sometime generate ordering problems

Logical Connectives & Rules

- Predicate logic uses *logical connectives* to construct complex sentences from simpler ones:
  - `and`, `or`, `not`, `<=`, `=>`, quantifiers `exists` and `forall`.

- Examples:
  - “Richard is not a crook”:
    ```
    (not (crook Richard))
    ```
  - “Every person has a mother”:
    ```
    (forall ?p
     (=> (person ?p)
     (exists ?m
      (has-mother ?p ?m))))
    ```
Using PowerLoom

- Starting PowerLoom using Java
  ```
  java -Xmx512m -jar AI.jar
  or
  powerloom
  ```
- Some useful interactive commands
  - Printing or changing modules (contexts)
    ```
    (cc)
    (cc "DOG")
    ```
  - Loading and saving work
    ```
    (load "my-work.plm")
    (save-module "DOG" "my-work.plm")
    ```
  - Getting help
    ```
    (help)
    (demo)
    ```
  - Stopping PowerLoom
    ```
    quit, bye, exit
    ```

An Example and Demo
Example Domain: Rabies

- animal
  - mammal
  - dog
  - sick animal
  - disease
  - rabies

“A dog is a mammal”
“A sick animal has a disease”
“rabies is a disease”

Defining “rabid dog” and “rabid animal”

- animal
  - mammal
  - dog
  - rabies
  - sick animal
  - disease
  - rabid dog
  - rabid animal

“rabid dog” and “rabid animal”
Classification in PowerLoom is not automatic

It must be invoked manually

- (classify-relations "MY-MODULE" true)
- (classify-instances "MY-MODULE" true)

Specific subset-of queries will still give the correct answer
- But value retrieval won’t find them
- Different effort expended – an example of PowerLoom incompleteness.
An Annotated Example

We define a separate BUSINESS module for our example
- Inherits built-in PowerLoom definitions from PL-KERNEL/PL-USER
- Sets up a separate name and assertion space to avoid unwanted interference with/from other loaded knowledge bases
- Allows easy experimentation (clearing/changing/editing/saving)
- All PowerLoom commands are interpreted relative to current module

(defvar "BUSINESS"
 :documentation "Module for the Business demo example."
 :includes ("PL-USER"))
(in-module "BUSINESS")
(clear-module "BUSINESS")
## Concepts

- Concepts define classes of entities
  - Defined via the `defconcept` command
  - Can have zero or more parent concepts (they all inherit `THING`)
  - Used to introduce typed instances

  ```lisp
  (defconcept company).
  (assert (company ACME-cleaners)).
  (retrieve all ?x (company ?x)).
  There are 2 solutions: #1: ?x=ACME-CLEANERS #2: ?x=MEGASOFT.
  ```

## Relations

- Relations define sets of relationships between entities
  - Defined via the `defrelation` command (& `deffunction` see later)
  - Can have one or more arguments (unary to n-ary)
  - Can be fixed or variable arity
  - Can be single or multi-valued
  - Usually specify types for each argument
  - Used to specify relationships between entities

  ```lisp
  (defrelation company-name ((?c company) (?name STRING))).
  (assert (company-name ACME-cleaners "ACME Cleaners, LTD").
  (assert (company-name megasoft "MegaSoft, Inc.")).
  ```
Relations /2

- Retrieve all relations asserted in the BUSINESS module:

  Number of solutions sought
  Retrieval variables specified implicitly

  (retrieve all (company-name ?x ?y))
  There are 2 solutions:
  #2: ?X=ACME-CLEANERS, ?Y="ACME Cleaners, LTD"

  (retrieve all (?y ?x) (company-name ?x ?y))
  There are 2 solutions:
  #1: ?Y="MegaSoft, Inc.", ?X=MEGASOFT
  #2: ?Y="ACME Cleaners, LTD", ?X=ACME-CLEANERS

Relation Hierarchies

- Hierarchies for concepts as well as relations are supported
  - PowerLoom represents a subconcept/subrelation relationship by asserting an "implication" relation (or an "implies" link)
  - Link is equivalent to a logic rule but allows more efficient inference
  - Various syntactic shortcuts are available to support often-used implication relations

  (defrelation fictitious-business-name ((?c company) (?name STRING))
   :=> (company-name ?c ?name))

  (forall (?c ?name)
   (=> (fictitious-business-name ?c ?name)
    (company-name ?c ?name)))

  (subset-of fictitious-business-name company-name)
Relation Hierarchies /2

- Retrieve all names of MegaSoft, fictitious or not
- Illustrates that company-name is a multi-valued relation

\[
\text{(assert (fictitious-business-name megasoft "MegaSoft"))}
\]

\[
\text{(retrieve all ?x (company-name megasoft ?x))}
\]

There are 2 solutions:
- #1: ?X="MegaSoft, Inc."
- #2: ?X="MegaSoft"

Functions

- Functions are term-producing, single-valued relations
- Defined via the deffunction command
- Very similar to relations defined via defrelation but:
  - Term producing: a function applied to its first n-1 input arguments specifies a unique, intensional term, e.g., "Fred's age"
  - Single-valued: each set of input arguments has at most one output argument (the last argument), e.g., "Fred's age is 42"
- By default, functions are assumed to be partial, i.e., could be undefined for some legal input values (e.g., 1/0)

\[
\text{(deffunction number-of-employees ((?c company)) :-} \rightarrow \text{ (?n INTEGER))}
\]

\[
\text{(assert (= (number-of-employees ACME-cleaners) 8))}
\]

\[
\text{(assert (= (number-of-employees megasoft) 10000))}
\]
Functions /2

- Functions syntax often results in shorter expressions than using similar relation syntax:

(retrieve all (and (company ?x) (< (number-of-employees ?x) 50)))

There is 1 solution:
#1: ?X=ACME-CLEANERS

- Compare to:

(retrieve all (and (company ?x) (exists ?n
                           (and (number-of-employees ?x ?n)
                                (< ?n 50)))))

There is 1 solution:
#1: ?X=ACME-CLEANERS

- Multiple function terms:

(retrieve all (> (number-of-employees ?x) (number-of-employees ?y)))

There is 1 solution:
#1: ?X=MEGASOFT, ?Y=ACME-CLEANERS

Defined Concepts

- Concepts (and functions/relations) can be defined completely in terms of rules
- Useful to name often-used queries or subexpressions and build up powerful vocabulary

(defconcept small-company (?c company):
  (<= (and (company ?c)
            (< (number-of-employees ?c) 50)))

New keyword

(forall ?c => (and (company ?c)
                     (< (number-of-employees ?c) 50))
                      (small-company ?c)))

(forall ?c => (small-company ?c)
              (and (company ?c)
                   (< (number-of-employees ?c) 50))))

Expands into these rules
Defined Concepts /2

- Retrieve small companies even if we don’t know exactly how many employees they have

```
(assert (and (company zz-productions)
             (< (number-of-employees zz-productions) 20)))

(retrieve all (small-company ?x))
```

There are 2 solutions:
- #1: ?X=ZZ-PRODUCTIONS
- #2: ?X=ACME-CLEANERS

All we know is that ZZ Productions has less than 20 employees

How is this derived
- Rule-based inference + transitivity of ‘<’

Negation & Open/Closed-World Semantics

- PowerLoom uses classical negation and an open-world assumption (OWA) by default
- KB is not assumed to be a complete model of the world: if something can’t be derived the answer is UNKNOWN, not FALSE
- Can distinguish between failure and falsity!
- Inference engine uses asymmetric effort to derive the truth or falsity of a query
  - Focuses effort on deriving truth, picks up falsity only via quick, shallow disproofs
  - Full effort for falsity available by asking for the negated query
  - Possible extension: 3-valued ask (similar to Loom)

```
(defun s-corporation ((?c corporation)))

(ask (s-corporation zz-productions)) \Rightarrow UNKNOWN
(ask (not (s-corporation zz-productions))) \Rightarrow UNKNOWN

(assert (not (s-corporation zz-productions)))

(ask (s-corporation zz-productions)) \Rightarrow FALSE
(ask (not (s-corporation zz-productions))) \Rightarrow TRUE
```

Due to open-world assumption
Quick disproof from assertion
Negation & Open/Closed-World Semantics /2

- Falsity can also come from sources other than explicit assertion
- Single-valued functions and relations
- Inequalities
- Disjoint types
- Negated rule heads, etc.

(ask (= (number-of-employees ACME-cleaners) 8)) ⇒ TRUE
(ask (= (number-of-employees ACME-cleaners) 10)) ⇒ FALSE
(ask (not (= (number-of-employees ACME-cleaners) 10))) ⇒ TRUE
(ask (= (number-of-employees zz-productions) 100)) ⇒ FALSE
(ask (= (number-of-employees zz-productions) 10)) ⇒ UNKNOWN

Quick disproof
since functions are single-valued

Quick disproof via inequality
constraints

Truly unknown
since there is not enough information

Negation & Open/Closed-World Semantics /3

- Selective closed-world semantics and negation-by-failure are also available (as used by Prolog, deductive databases, F-Logic, etc.)
- Useful in cases where we do have complete knowledge
- If something can’t be derived, it is assumed to be false
- Closed-world semantics specified by marking relations as closed
- Negation-by-failure via fail instead of not

(defrel (works-for (?p (?c company))))

(assert (works-for shirly ACME-cleaners))
(assert (works-for jerome zz-productions))

(ask (not (works-for jerome megasoft))) ⇒ UNKNOWN

Due to open world

Mark relation as closed

Via selective closed-world semantics

(retract (closed works-for))

(assert (closed works-for))

(ask (not (works-for jerome megasoft))) ⇒ TRUE

Via explicit negation-by-failure

(ask (fail (works-for jerome megasoft))) ⇒ TRUE
Retraction

- Retraction allows the erasure or change of a previously asserted truth-value of a proposition
- Useful for error correction or iterative "change of mind" during development
- Useful to change certain aspects of a scenario without having to reload the whole knowledge base
- Allows efficient, fine-grained change
  - Some cached information is lost and needs to be regenerated
  - Loss can be minimized by careful structuring of module hierarchy (put more stable knowledge higher up in the hierarchy)
- Allows the exploration of hypothetical conjectures
  - What would change if F were true or false?
  - Module system allows us to consider both possibilities at the same time

Some geographic terminology and information

```
(defconcept geographic-location)
(defconcept country ((?1 geographic-location)))
(defconcept state ((?1 geographic-location)))
(defconcept city ((?1 geographic-location)))
(defrelation contains ((?1 geographic-location) (?2 geographic-location)))

(assert (and
    (country united-states)
    (geographic-location eastern-us)
    (contains united-states eastern-us)
    (state georgia) (contains eastern-us georgia)
    (city atlanta) (contains georgia atlanta)
    (geographic-location southern-us)
    (contains united-states southern-us)
    (state texas) (contains eastern-us texas)
    (city dallas) (contains texas dallas)
    (city austin) (contains texas austin)))
```
Retraction /3

- Retraction to fix an incorrect assertion

(ask (contains eastern-us texas)) => TRUE
(retract (contains eastern-us texas))
(assert (contains southern-us texas))
(ask (contains eastern-us texas)) => UNKNOWN

Value Clipping

- Functions allow implicit retraction via *value clipping*
- Assertion of a function value automatically retracts a preexisting value
- Justified, since functions are single-valued

(deffunction headquarters ((?c company)) :-> (?city city))
(assert (= (headquarters zz-productions) atlanta))
(retrieve all (= ?x (headquarters zz-productions)))
There is 1 solution:
#1: ?X=ATLANTA

(assert (= (headquarters zz-productions) dallas))
(retrieve all (= ?x (headquarters zz-productions)))
There is 1 solution:
#1: ?X=DALLAS

- Assertion automatically clips previous value
- DALLAS value replaced ATLANTA
Value Clipping /2

- Clipping also works for single-valued relations

(defrelation headquartered-in ((?c company) (?city city)) :axioms (single-valued headquartered-in))

(assert (headquartered-in megasoft atlanta))
(retrieve all (headquartered-in megasoft ?x))
There is 1 solution:
 #1: ?X=ATLANTA

(assert (headquartered-in megasoft dallas))
(retrieve all (headquartered-in megasoft ?x))
There is 1 solution:
 #1: ?X=DALLAS

Contradictions

- Propositions that are both TRUE and FALSE are contradictory
- Contradictions can result from explicit assertions, during forward-chaining, or as the result of a refutation proof
- Contradictory propositions are treated as UNKNOWN to allow the system to continue to function

(assert (not (state texas)))

Derived both TRUE and FALSE for the proposition `|P#| (STATE TEXAS)'.
Clash occurred in module `|MDL|/FL-KERNEL-KB/BUSINESS'.

(ask (state texas)) ⇒ UNKNOWN
(ask (not (state texas))) ⇒ UNKNOWN
Rule-Based Inference

- Logic rules can be used to model complex relationships
- Rules can be unnamed or named via `defrule`
- Most definition commands expand into one or more rules
- Inference engines apply rules to derive conclusions

```
(retrieve all (contains southern-us ?x))
There is 1 solution:
#1: ?X=TEXAS
```

```
(defrule transitive-contains
(forall (?l1 ?l2 ?l3)
  => (and (contains ?l1 ?l2)
          (contains ?l2 ?l3))
  (contains ?l1 ?l3)))
```

```
(retrieve all (contains southern-us ?x))
There are 3 solutions:
#1: ?X=TEXAS
#2: ?X=AUSTIN
#3: ?X=DALLAS
```

```
(retract transitive-contains)
(retrieve all (contains southern-us ?x))
There is 1 solution:
#1: ?X=TEXAS
```

```
(assert (transitive contains))
(retrieve all (contains southern-us ?x))
There are 3 solutions:
#1: ?X=TEXAS
#2: ?X=AUSTIN
#3: ?X=DALLAS
```

Named Rules & Axiom Schemata

- Logic rules can be defined and named via `defrule`
- Rules are propositions which are in the domain of discourse
- Allows meta-annotations and reasoning
- Naming rules (or any proposition) provides extra level of convenience
- Axiom schemata allow simple definition of commonly used rule patterns

```
(defrelation transitive ((?r RELATION))
  =>> (binary-relation ?r)
  =>> (not (function ?r))
  =>> (=> (and (?r ?x ?y)
                 (?r ?y ?z))
           (=> (and (?r ?x ?y)
                    (?r ?y ?z))))
```

```
(retract transitive-contains)
```

```
(assert (transitive contains))
```

```
(assoc (transitive contains))
```

```
(defrelation transitive ((?r RELATION))
  =>> (binary-relation ?r)
  =>> (not (function ?r))
  =>> (=> (and (?r ?x ?y)
                 (?r ?y ?z))
           (=> (and (?r ?x ?y)
                    (?r ?y ?z))))
```
Justifications and Explanation

- Explanation of true/false queries
  - Backward inference can store proof trees that can be rendered into explanations
  - Simple built-in explanation mechanism
    - Various rendering possibilities, ASCII, HTML, XML
    - Eliminates explanation of duplicate and low-level goals
    - Explanation strings for different audiences (technical, lay)

(ask (contains southern-us dallas)) ⇒ TRUE

(why)
1 (CONTAINS SOUTHERN-US DALLAS)
  follows by Modus Ponens
  since 1.1 ! (FORALL (?l1 ?l3)
    (<= (CONTAINS ?l1 ?l3)
      (EXISTS (?l2)
        (AND (CONTAINS ?l1 ?l2)
          (CONTAINS ?l2 ?l3))))
    )
and 1.2 ! (CONTAINS SOUTHERN-US TEXAS)
and 1.3 ! (CONTAINS TEXAS DALLAS)

Explanation /2

- Explanation of retrieved results
- Separate explanation for each derived solution
- why explains most recently retrieved solution

(retrieve 3 (contains southern-us ?x))
There are 3 solutions so far:
#1: ?x=WASHINGTON
#2: ?x=TEXAS
#3: ?x=AUSTIN

(why)
1 (CONTAINS SOUTHERN-US AUSTIN)
  follows by Modus Ponens
  since 1.1 ! (FORALL (?l1 ?l3)
    (<= (CONTAINS ?l1 ?l3)
      (EXISTS (?l2)
        (AND (CONTAINS ?l1 ?l2)
          (CONTAINS ?l2 ?l3))))
    )
and 1.2 ! (CONTAINS SOUTHERN-US TEXAS)
and 1.3 ! (CONTAINS TEXAS AUSTIN)
Contexts & Modules

- Hypothetical or scenario reasoning can be achieved by
  - creating a new context which inherits existing set of facts and
  - allows the exploration of "assumptions".
- In this example, we show how certain inherited assertions can be retracted and changed

```
(defmodule "ALTERNATE-BUSINESS"
  :includes "BUSINESS")

(in-module "ALTERNATE-BUSINESS")

(assert (and (company web-phantoms)
              (company-name web-phantoms "Web Phantoms, Inc.")))

(retract (company-name megasoft "MegaSoft, Inc.")
(assert (company-name megasoft "MegaZorch, Inc.")
```

Contexts & Modules /2

- The ALTERNATE-BUSINESS module
  - inherits all of the information of its parent module
  - is subject to the specific changes made in the local module.

```
(in-module "BUSINESS")

(retrieve all (company-name ?x ?y))
There are 3 solutions:
#2: ?X=ACME-CLEANERS, ?Y="ACME Cleaners, LTD"
#3: ?X=MEGASOFT, ?Y="MegaSoft"

(in-module "ALTERNATE-BUSINESS")

(retrieve all (company-name ?x ?y))
There are 4 solutions:
#2: ?X=/PL-KERNEL-KB/PL-USER/BUSINESS/ALTERNATE-BUSINESS/WEB-
    PHANTOMS, ?Y="Web Phantoms, Inc."
#3: ?X=ACME-CLEANERS, ?Y="ACME Cleaners, LTD"
#4: ?X=MEGASOFT, ?Y="MegaSoft"
```

(From "fictional business name" assertion

New local assertion with qualification name — the lower name is not visible in the upper context

Changed local assertion
Cross-Contextual Reasoning

- Normally queries operate in the current module.
- The IST (IS-TRUE) relation (J. McCarthy) allows us to query about the state of knowledge in other modules.
- This also allows cross-module inference by binding variables across forms
- Example: “find all companies whose names differ in the two modules”

```
(in-module "BUSINESS")

(retrieve all (ist alternate-business (company-name ?x ?y)))
There are 4 solutions:
#3: ?X=ACME-CLEANERS, ?Y="ACME Cleaners, LTD"
#4: ?X=MEGASOFT, ?Y="MegaSoft"

(retrieve all (and (ist business (company-name ?x ?y))
(fail (ist alternate-business (company-name ?x ?y)))))
There is 1 solution:

Using PowerLoom from Java
Java Setup

- Details in the PowerLoom Manual
- Mapping PowerLoom names
  - Follows standard Java conventions
    - s-assert-proposition ⇒ sAssertProposition
  - "*" character maps to "$"
    - *module* ⇒ $MODULE$ — it's a global variable!
  - "?" character maps to "P" (for Predicate)
    - next? ⇒ nextP
- Java import statements

```java
import edu.isi.powerloom.*;
import edu.isi.powerloom.logic.*;
import edu.isi.stella.Module;
import edu.isi.stella.Stella_Object;
```

Initialization and Loading Files

- PowerLoom needs to be initialized before using. This can take a while. This form initializes basic PowerLoom
  - PLI.initialize();

- Other systems may also need initialization.
  - For example, PowerLoom extensions to get units and dimensions:
    - StartupPowerloomSystem.startupPowerloomSystem();

- PowerLoom files may need loading
  - PLI.load("mykb.plm", null);
Assertions, Retractions and Definitions

- Almost all needed interface methods are in the PLI class as static methods.
- Many have both object and String interfaces. Strings are generally easier to use.
- The general `sEvaluate` form can process any command that can be given at the interactive prompt.
- Most methods take a module and environment argument. The environment can be left as `null` to use the default.

```java
PLI.sAssertProposition("(Person Fred)", "PL-USER", null);
PLI.sAssertProposition("(name Fred "Frederick")", "PL-USER", null);
PLI.sRetractProposition("(Hungry Fred)", "PL-USER", null);
PLI.sCreateRelation("friend", 2, "PL-USER", null);
PLI.sEvaluate("(deffunction age ((?p Person) (?n INTEGER)))", "PL-USER", null);
```

- Ask queries return values of type `TruthValue`
- PLI has predicates to test the returned values.

```java
PLI.isTrue(PLI.sAsk("(> 8 7)", "PL-USER", null));
TruthValue tv = PLI.sAsk("(friend Jobs Eisner)", "PL-USER", null);
if (isTrue(tv)) System.out.println("Yes!");
if (isFalse(tv)) System.out.println("No.");
if (isUnknown(tv)) System.out.println("How should I know?");
if (isDefault(tv)) System.out.println("by default reasoning");
```
"Retrieve" Queries

- Retrieve queries return values of type PlIterator

String query = "all (and (Senator ?sen) (represents ?sen California)"
   + "(political-party ?sen ?party))";
PlIterator answer = PLI.sRetrieve(query, "POLITICS", null);

System.out.println("Answers to query " + query + ")
while (answer.nextP()) {
   // Iterate over the answers
   System.out.println(answer.value);
}

- Uses a different iterator protocol than Java
  - iterator.nextP() advances iteration and returns a boolean. This must be done first.
  - iterator.value gets the current value, and can safely be called more than once.
- Can be wrapped to use Java protocol
  - import edu.isi.stella.javalib.*;
  - javaIt = StellaIterator(PLI.sRetrieve(…));
- Values are of type Stella_Object and are tuples.
  Tuples can be decomposed using PLI.getNthValue(…)

Iterators for PowerLoom Answers
PowerLoom Datatypes in Java

- Literals are returned wrapped but can be coerced.
  - integer ⇒ int
  - float ⇒ double
  - string ⇒ String

- Logic Objects
  - type is edu.isi.powerloom.logic.LogicObject
  - PowerLoom objects like relations, instances, descriptions, skolems

- Stella Objects
  - type is edu.isi.stella.Stella_Object
  - Most general type. Usually wrapped literals, but may be modules.

Warning: You don’t always get what you expect!
- Skolems can appear when you expect, say, a number
- Best to test the type first!

```java
PLI.sAssertProposition("(and (age Fred 10) (> (weight Fred) 150))"...)

PLIterator answer;
answer = PLI.sRetrieve("1 (and (age Fred ?a) (weight Fred ?w))", ...)
answer.nextP();

// The next line works since age is 10, but is dangerous
int age = PLI.getNthInteger(answer.value, 0, "PL-USER", null);

// The next line blows up because the answer is a skolem!
int weight = PLI.getNthInteger(answer.value, 1, "PL-USER", null);

if (PLI.isInteger(PLI.getNthValue(answer.value, 1, "PL-USER", null))) {
    weight = PLI.getNthInteger(answer.value, 1, "PL-USER", null);
}
```
Additional Resources

- The interactive interface
  - Try things out before programming

- PowerLoom Manual
  - Has general information
  - Has information about Java-specific information

- Javadoc for PowerLoom
  - Caveat: For technical reasons almost all methods are public, but the intended API is contained mostly in the PLI class

- The example file PowerLoomExample.java

- PowerLoom website:
  http://www.isi.edu/isd/LOOM/PowerLoom/documentation

Ontosaurus:
Browsing PowerLoom
Relation BASIN-DEPTH-2.5

Textual Definition

Depth to the 2.5 km/s Vs boundary in a basin.

Structured Description

Formal logical encoding of one constraint implied by the textual definition

$\text{Basin-depth} = 0 \text{m} \iff \text{Vs30} > 2.5 \text{km/s}$

Ontosaurus Demo
Conclusion

How Does Logic Model the World?

- Terms correspond to entities in the (some) world
- Predicates model properties and relations between entities
- Domain rules define and constrain relations, for example, “If Joe is a teenager who owns a car then Joe is happy”
- Logical inference rules define the propagation of truth between logical sentences, for example: from X and X => Y it must be true that Y
- The more rules and sentences we add, the higher constrained their “interpretation” (what they could mean) becomes
- However, every consistent theory always has infinitely many (formal) interpretations
Advantages of Logic-based Models

- Tradition
  - Well-understood syntax and semantics
  - Very large amount of relevant research (> 2000 yrs.)
  - Many available logic-based tools
    - Provers, constraint reasoners, learners, planners, KR&R systems, etc.

- Representational power
  - Negation
  - Disjunction
  - Equality (object identity)
  - Logical connectives
  - Quantification
  - Rules, constraints
  - Abstraction
  - Definitions
  - Extendable vocabulary, ontologies
  - “If you can’t say it in logic, you probably don’t want to say it” 😊

Advantages of Logic-based Models

- General purpose, well-understood inference mechanisms
  - Deduction
  - Abduction
  - Induction
  - Constraint satisfaction
  - Automated reasoners
Advantages of Logic-based Models

- Formalizes reasoning and gives justification
  - Proofs provide justifications for derived facts
  - If one accepts the premises one must/should accept the conclusions

- Explanation and understandability
  - Proofs are a good starting point to provide explanations
  - Logical models are "easy" to understand and interpret (e.g., rules learned by an ILP method)
  - Logical models are easier to debug than other approaches

- Translatability
  - Different logical representations are (often) easily translatable into each other (e.g., this diffuses the attribute-vs.-collection distinction)

Disadvantages?

- Disadvantages
  - Difficult to handle uncertainty and probabilistic reasoning
    - But, various efforts to combine logical and probabilistic models (e.g., PRM’s)
  - Complexity of reasoning algorithms
  - Sometimes too expressive, too many different ways of saying the same thing
  - Hard to handle grey areas, but the world is grey
    - Have to make hard decisions (true, false)
    - Hard to say “many”, “few”, “nearly”, etc. (frustrates NLP people)