

Object-Oriented KRR, OO logical languages and Flora-2

CSE 505 – Computing with Logic

Stony Brook University

<http://www.cs.stonybrook.edu/~cse505>

What is Object-Oriented? Ontological Thinking

- Predicative logic:
 - Flat representation “has(car, wheels).”
 - Each sentence is self-contained “has(car, engine).”
 - Information about an entity is scattered in multiple sentences “is(prius, car).”

VS.

- Object-oriented:
 - Sentences are grouped
 - Structured and organized
 - Usually a correspondence with the user interface
 - Translatable to logic
- “prius isA car (a car has an engine and wheels)”

Frames

- When one encounters a new situation, one selects from memory a structure called a Frame. This is a remembered framework to be adapted to fit reality by changing details as necessary – Marvin Minsky, 1974
- Example: a Birthday Party

DRESS ——— SUNDAY BEST.

PRESENT ——— MUST PLEASE HOST. MUST BE BOUGHT AND GIFT-WRAPPED.

GAMES ——— HIDE AND SEEK. PIN TAIL ON DONKEY.

DECOR ——— BALLOONS. FAVORS. CREPE-PAPER.

PARTY-MEAL—CAKE. ICE-CREAM. SODA. HOT DOGS.

CAKE ——— CANDLES. BLOW-OUT. WISH. SING BIRTHDAY SONG.

ICE-CREAM ——— STANDARD THREE-FLAVOR.

Reasoning Operations on Frames

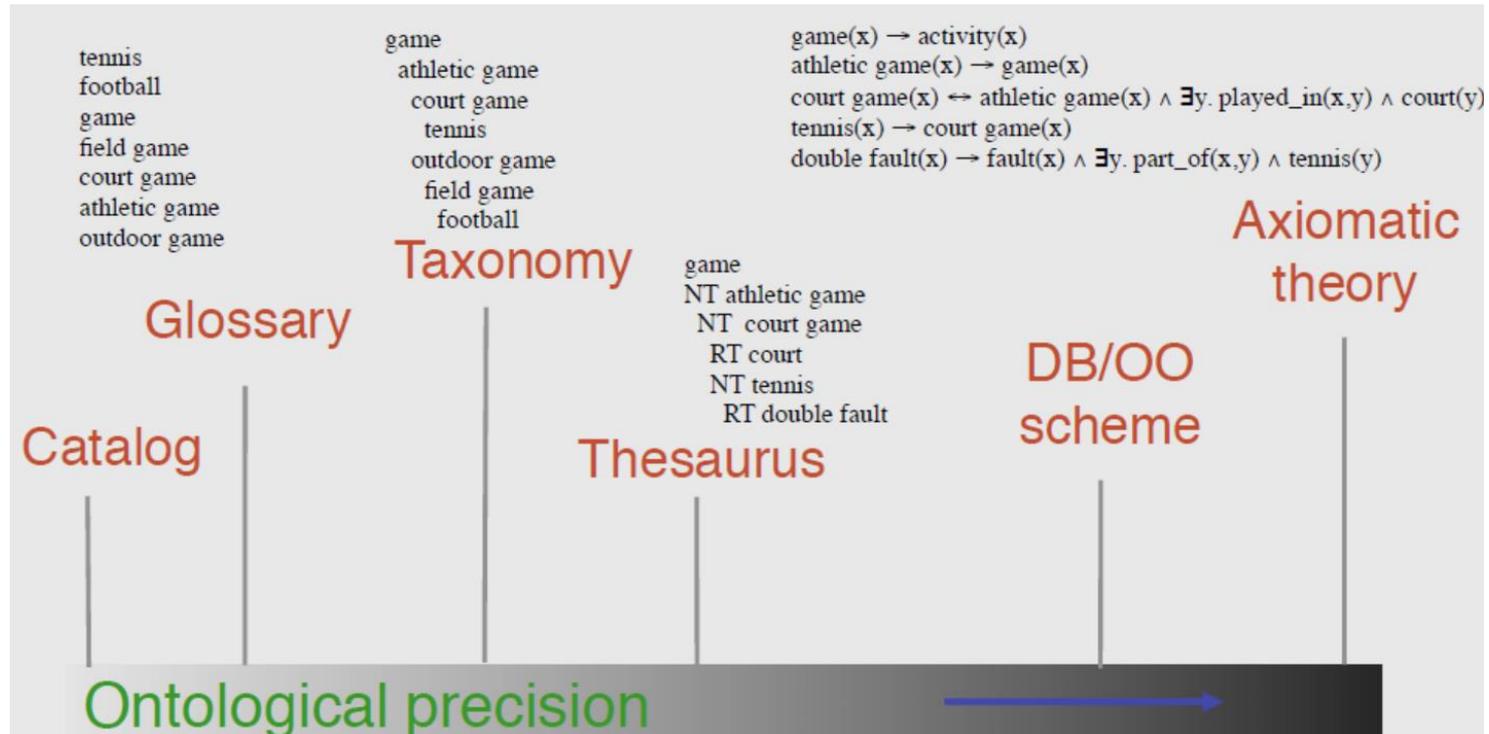
- Expectation: How to select an initial frame to meet some conditions
 - Child's birthday party
- Elaboration: How to select and assign sub-frames to represent additional details
 - North American birthday party vs an Asian Party
- Alteration: How to find a frame to replace one that does not fit well
 - No gifts allowed
- Novelty: What to do if no acceptable frame can be found?
- Learning: What frames should be stored or modified as a result of experience?

Object-Oriented Languages

- Many modern programming languages support features such as data abstraction and inheritance: Java
- Object databases represent information as objects and support object-oriented programming
- Graphical representations have several features in common
 - A hierarchy of classes
 - Classes have properties that can inherit
 - Facets provide further descriptors of values

Object-Oriented KR Languages

- An *ontology* defines a set of representational primitives with which to model a domain of knowledge or discourse
 - The representational primitives are classes and relationships
 - Their definitions include information about their meaning and constraints on their logically consistent application



Nicola Guarino

Object-Oriented KR Languages

- The Knowledge Machine <http://www.cs.utexas.edu/users/mfkb/km.html> provides classes, individuals, instance-of, subclass-of, Template Slots, Domains, Ranges, Inheritance, Logical Interface, ...
- Description Logic:

```
 $\mathcal{T} = \{\text{Doctor} \sqsubseteq \text{Person}, \text{Parent} \equiv \text{Person} \sqcap \exists \text{hasChild}.\text{Person},$   
 $\text{HappyParent} \equiv \text{Parent} \sqcap \forall \text{hasChild}.\text{(Doctor} \sqcup \exists \text{hasChild}.\text{Doctor)}\}$   
 $\mathcal{A} = \{\text{John}:\text{HappyParent}, \text{John hasChild Mary}, \text{Mary}:\forall \text{hasChild}.\perp$ 
```

$\models \text{Mary}:\text{Doctor}$? 

John:HappyParent, John hasChild Mary, Mary: \forall hasChild. \perp
Mary: \neg Doctor
John:Parent, John: \forall hasChild.(Doctor \sqcup \exists hasChild.Doctor)
John:Person, John: \exists hasChild.Person
Mary:(Doctor \sqcup \exists hasChild.Doctor)
John hasChild a, a:Person, a:(Doctor \sqcup \exists hasChild.Doctor)
Mary: \exists hasChild.Doctor
Mary hasChild b, b:Doctor, b:Person
 b: \perp

Ian Horrocks

- A commonly used language that uses description logic is OWL <http://www.w3.org/TR/owl2-overview/>
- An API for accessing OWL knowledge: <http://owlapi.sourceforge.net>

Flora-2

- Combines Logic Programs with Object-Oriented Representation:
 - Flora-2 is an advanced object-oriented knowledge base management system, including meta-programming and defeasible reasoning as an extension of Prolog
 - The Flora-2 project sourceforge Web site:
<http://flora.sourceforge.net>
 - In particular, from that main page view.
[FLORA-2 Documentation](#) (user's manual, documentation of packages and technical tutorial),
[FLORA-2 Mailing lists](#) (for general users and development).

The Syntax of Flora-2

- The alphabet of Flora-2 consists of:
 - . ends every statement in Flora-2, including queries.
 - Constant symbols: `123`, `a`, `John`, `"12345"^^xsd:integer` (constant in a lexical space)
 - Variable symbols as alphanumeric symbols prefixed by the character “?”: `?x`, `?Var1`, `?ABC`, `?_ABC`, `?` (the last two are anonymous variables)
 - Terms: `f(1,a,?Var)` , `?p(b(?x),?q(c,d))` (Hilog and first-order terms); `[a,b,c]`, `[1,2,3 | [4,5]]` (list terms)
 - Negation symbols: `\naf` (negation-as-failure, a.k.a. default negation) and `\neg` (strong negation).
 - Unification and equality symbols: `=` (unifies), `:=` (logically equal), `:=` (user-defined function operator).
 - Frame construction symbols: `->` (has value), `=>` (has type), `:` (class membership), `::` (subclass-of).
 - Connectives: `:-` (directional implication, i.e., “if” between head and body); `“,”` (comma) and `\and` for conjunction; `“;”` (semi-colon) and `\or` for disjunction.
 - Comments: `//` (rest of line), `/*...*/` (enclosed, possibly multi-line).
 - Aggregation operators: `\collectset`, `min`, `max`, `count`, `sum`, `avg`.
 - Comprehension operators: `\setof` , `\bagof`.
 - Other/auxiliary symbols: `(,)` , `[,]` , `,` (comma), `<` , `>` , `|` , `{, }` , `$` (used for reification), `@` (used for meta-info annotation) , `'` (single quote, used for symbols).
 - Delay pragmas (goal reordering): `must`, `wish`.

Facts, Rules and Queries

- Flora-2 is a logic programming reasoning engine – it combines facts and rules to infer new facts

A fact:

Socrates is a man.

`man(Socrates).`

A rule:

?X is mortal if ?X is a man.

`mortal(?X) :- man(?X).`

A query:

Is Socrates mortal?

`?- mortal(Socrates).`

The answer:

Yes

TRUE

Facts, Rules and Queries

// socrates.flr

man(Socrates).

mortal(?X) :- man(?X).

age(Socrates,56) and home(Socrates,Athens).

student(Socrates,Plato) and

student(Socrates,Xenophon).

man(Plato) and man(Xenophon).

age(Plato,27). age(Xenophon,27).

philosopher(Xenophon).

talksAbout(Xenophon,Xenophon).

home(father(Socrates),Athens).

home(father(father(Socrates)),Athens).

avgAge(?AvgAge) :-

?AvgAge = avg{ ?Y | man(?E), age(?E,?Y) }.

?- mortal(?X). /* ?X = Plato; ?X = Socrates;
?X = Xenophon */

?- age(?X,?Y). /* ?X = Plato, ?Y = 27;
?X = Socrates, ?Y = 56;
?X = Xenophon, ?Y = 27 */

?- home(?X,?Y). /* ?X = Socrates, ?Y = Athens;
?X = father(Socrates), ?Y = Athens
?X = father(father(Socrates)),?Y = Athens*/

?- avgAge(?AvgAge). // ?AvgAge = 36.6667

HiLog

- HiLog (higher-order) - a predicate can be any kind of term
(in particular, a variable)

Which facts are about Socrates?

?- ?X(Socrates). // predicate is a variable

- HiLog will be especially useful for Frame syntax (next slide).

Frame Syntax

- Flora-2 offers an alternative, object-oriented syntax for many expressions: [Frame syntax \(F-logic\)](#).
- The regular syntax is called, by contrast, **Predicate syntax**.
- Frame syntax is more concise than Predicate syntax.
 - Especially for users familiar with object-oriented programming or RDF, frame syntax is sometimes more intuitive or familiar than predicate syntax.

In Predicate syntax:

In Frame syntax:

Frame syntax rearranges the components
of Flora-2 facts having 2 arguments:

Socrates is 56 years old.

age(Socrates,56).

Socrates[age -> 56].

More generally, it rearranges the subject, predicate
(a.k.a. property), and object (a.k.a. value) of the fact:

predicate(subject , object).

subject[predicate -> object].

There is a special syntax for expressing several values
for the same property:

Socrates has Plato and Xenophon as students.

student(Socrates,Plato) and
student(Socrates,Xenophon).

Socrates[student -> {Plato, Xenophon}].

If desired, we can write a frame containing
several properties of the same subject:

Socrates is 56 years old and lives in Athens.

age(Socrates,56) and
home(Socrates,Athens).

Socrates[age -> 56, home -> Athens].

Frame Syntax

In Predicate Syntax

We can also make statements
asserting what class, or kind
of thing, an object is:

Socrates is a man.

man(Socrates).

We can also express subclass relationships
between classes:

All Athenians are people.

subclass(Athenian, Person).

We can also write rules in the frame syntax:

All immortal men are gods.

god(?X):- man(X) and
immortal(?X).

In Frame Syntax:

Socrates : man.

Athenian :: Person.

?X : God :- ?X: Man
and ?X: Immortal.

Frame Syntax

// socrates_frames.flr

Socrates : man.

?X:mortal :- ?X:man.

Socrates[age->56] and Socrates[home->Athens].

Socrates[student->{Plato,Xenophon}].

Plato:man and Xenophon:man.

Plato[age->27]. Xenophon[age->27].

Xenophon:philosopher.

Xenophon[talksAbout->Xenophon].

father(Socrates)[home->Athens].

father(father(Socrates))[home->Athens].

avgAge(?AvgAge) :-

 ?AvgAge = avg{ ?Y | ?E:man, ?E[age->?Y] }.

?X:Athenian :- ?X[home->Athens].

Athenian :: Person.

?X : God :- ?X: Man , ?X: Immortal.

?- ?X : mortal. /* ?X = Plato; ?X = Socrates;
 ?X = Xenophon */

?- ?X[age->?Y]. /* ?X = Plato, ?Y = 27;
 ?X = Socrates, ?Y = 56;

 ?X = Xenophon, ?Y = 27 */

?- ?X[home->?Y]./*?X = Socrates, ?Y = Athens;
 ?X = father(Socrates), ?Y = Athens

 ?X = father(father(Socrates)),?Y = Athens*/

?- avgAge(?AvgAge). // ?AvgAge = 36.6667

?- ?X : Person. // ?X = Socrates;

 ?X = father(Socrates);

 ?X = father(father(Socrates))*/

Rule Annotations

- Flora-2 uses annotations to record key meta-information about a rule
- Annotations are always at the beginning of the rule, in a frame that looks like this: $@!\{\text{ruleId}\}$ and/or $@\{\text{tag}\}$ and/or $@@\{\text{strict}\}$ and/or $@@\{\text{defeasible}\}$

An annotation assigning a unique identifier, 'rule24', to a rule and making it defeasible:

$@!\{\text{rule24}\} @@\{\text{defeasible}\} p(?X) :- q(?X).$

An annotation assigning a 'tag' `newRules` to a rule (for the purpose of prioritizing it with respect to defeasibility):

$@\{\text{newRules}\} p(?X) :- q(?X).$

Rule Conflict and Overriding

- What if two rules make contradictory conclusions?

For example, in the following rule set, two different rules conclude $p(a)$ and $\text{neg } p(a)$:

In this situation, neither $p(a)$ nor $\text{neg } p(a)$ is inferred – both inferences are “defeated” – as we see in these two queries:

We can change that by stating that one rule tag “overrides” the other:

The addition of the override info causes us to infer $p(a)$ but still defeat $\text{neg } p(a)$

$@\{\text{foo}\} \quad p(a) \text{ :- } q(a).$

$@\{\text{bar}\} \quad \text{neg } p(a) \text{ :- } q(a).$

$q(a).$

$?- p(a).$

No

$?- \text{neg } p(a).$

No

$\backslash\text{overrides}(\text{foo}, \text{bar}).$

$?- p(a).$

Yes

Other Key Features

- Reification – use Flora-2 sentences, or other Flora-2 complex expressions, as terms

- allows us to use rules as data

Socrates believes that students are pleasant if stimulated.

`believes(Socrates, $ {pleasant(?x):- student(?x) and stimulated(?x)}).`

- Aggregation – summarizing many facts

What is the average age of employees?

`?- ?AvgAge = \avg{ ?Y | man(?E), age(?E,?Y)].`

Frames and Rules Reasoning Example

@!{fam1}

Family1 : Family[husband->Mike,
 wife->Nancy,
 son->{Jason, Noah}].

@!{fam2}

Family2 : Family[husband->Tim,
 wife->Karen,
 daughter->{Katherine, Caroline}].

...

@!{son1} @s1 ?f[child->c] :- ?f[son->c].

@!{daughter1} @d1 ?f[child->c] :- ?f[daughter->c].

@!{husband_child1} @hc1

?f[parent->p] :- ?f[husband->p, child->c].

@!{wife_child1} @wc1

?f[parent->p] :- ?f[wife->p, child->c].

@!{father1} @f1

?child[father->father] :- ?family[husband->father, child->child].

@!{grandfather1} @gf1

?c[grandfather->gf] :- ?f1[parent->p, child->c], ?f2[husband->gf, child->p].

1. Load the file.
2. Ask the query:
“?- ?Child{father->?Father}.”

Defeasible Rules Reasoning Example

```
@{rep}    neg pacifist(?X) :- republican(?X).
@{qua}    pacifist(?X) :- quaker(?X).
@{pri1}   \overrides(rep, qua).
@{fac1}   republican(nixon).
@{fac2}   quaker(nixon).
```

1. Ask the query:

```
?- neg pacifist(nixon).
```

TRUE

2. Another interesting query is:

```
?- pacifist(nixon).
```

FALSE

Aggregates Example

```
John : Employee[salary(2000)->10,  
              salary(2001)->11,  
              salary(2002)->12].
```

```
Ed : Employee[salary(2000)->20,  
              salary(2001)->22,  
              salary(2002)->24].
```

```
// count employees
```

```
?- ?employeeCount = \count{?who | ?who : Employee}.
```

```
// average salary of all employees
```

```
?- ?avgSalary = \avg{?salary | ?who : Employee[salary(?year)->?salary]}.
```

```
// each employee's average salary along with the value of the grouping variable ?who
```

```
?- ?avgSalary = \avg{?salary{?who} |  
                  ?who : Employee[salary(?year)->?salary]}.
```

```
// total salary by year
```

```
?- ?yearlyPayroll = \sum{?salary{?year} | ?who : Employee[salary(?year)-  
>?salary]}.
```

```
// minimum salary
```

```
?- ?min = \min{?salary | ?who{salary(?year)->?salary}}.
```

```
// years for which salary information is available
```

```
?- ?years = \collectbag{?year | ?who{salary(?year)->?salary}}.
```

```
// unique years for which salary information is available
```

```
?- ?years = \collectset{?year | ?who{salary(?year)->?salary}}.
```

1. Load salary.flr
2. Ask queries in the Flora-2 Console to count employees, compute average salary, etc.

Trust Policy example

Use higher order logic to implement trust policies

```
2007 : times. 2008 : times. 2009 : times.
print : privileges. webPage : privileges.
Bob : admins. John : admins. Cara : admins.
Al : users.
Bob[controls -> print].           //Bob controls printing.
neg John[controls -> print].     // John explicitly does not administer printing
Cara[controls -> ?priv} :- ?priv : privileges. // Cara is the most senior admin.
//privileges are enforced based upon the statements admins make
@{grantOrDeny(?admin,?t,?X)} ?priv(?user) :- ?priv : privileges, ?admin : admins,
?admin{states(?t) -> ?X}, ?X = ${?priv(?user)}.
@{grantOrDeny(?admin,?t,?X)} neg ?priv(?user) :- ?priv : privileges and ?admin : admins and
?admin[states(?t) -> ?X] and ?X = ${neg ?priv(?user)}.
//metastatements about privilege statements
?X[atom->${?priv(?user)}] :- ?X = ${?priv(?user)}.
?X[negated->>false] :- ?X = ${?priv(?user)}.
?X[atom->${?priv(?user)}] :- ?X = ${neg ?priv(?user)}.
?X[negated->>true] :- ?X = ${neg ?priv(?user)}.
// More recent statements have higher priority, in case of conflict.
@{recency} \overrides(grantOrDeny(?admin1,?t1,?X), grantOrDeny(?admin2,?t2,?Y)) :-
  ?admin1 : admins and ?admin2 : admins and ?t1 : times and ?t2 : times and
  ?admin1{states(?t1) -> ?X} and ?admin2{states(?t2) -> ?Y} and ?X.atom = ?Y.atom and
  naf ?X.negated == ?Y.negated and ?t2 < ?t1.
```

Continued on next slide.

(c) Paul Fodor (CS Stony Brook)

Trust Policy example

```
// Statements made by admins that control a particular privilege have priority over
statements made by admins that do not control that privilege.
```

```
@{control} \overrides(grantOrDeny(?admin1,?t1,?X), grantOrDeny(?admin2,?t2,?Y)) :-
  ?admin1 : admins and ?admin2 : admins and ?t1 : times and ?t2 : times and
  ?admin1{states(?t1) -> ?X} and ?admin2{states(?t2) -> ?Y} and ?X.atom = ?Y.atom and
  naf ?X.negated ==: ?Y.negated and ?X.atom = ${?priv(?user)} and
  ?admin1{controls -> ?priv} and naf ?admin2{controls -> ?priv}.
```

```
//Two overrides statements cannot override each other and both be true.
```

```
!- \overrides(grantOrDeny(?admin2,?t2,?X), grantOrDeny(?admin1,?t1,?Y)) and
  \overrides(grantOrDeny(?admin1,?t1,?Y), grantOrDeny(?admin2,?t2,?X)).
```

```
//Statements concerning control trump those concerning recency.
```

```
\overrides(control,recency).
```

```
// Admins Bob and Cara make conflicting statements over time about Al's printing. Both
administer printing. Bob's statement is more recent, so prevails.
```

```
// Admins Bob and John make conflicting statements over time about Al's printing. John's is
more recent, but John does not administer printing, so Bob's statement prevails.
```

```
// Admins Cara and John make conflicting statements over time about Al's webPage. John's is
more recent, but John does not administer webPages, so Cara's statement prevails.
```

```
Cara[states(2007) -> ${print(Al)}].
```

```
Cara[states(2007) -> ${webPage(Al)}].
```

```
Bob[states(2008) -> ${neg print(Al)}].
```

```
John[states(2009) -> ${neg webPage(Al)}].
```

```
John[states(2009) -> ${print(Al)}].
```

1. Load the file in Flora-2.

2. Queries:

?- neg print(Al)

?- webPage(Al)

**Al is permitted to have a webPage
but not to print.**

Extra lecture notes

- The following lecture notes describe:
 - How to install Flora-2, and
 - How to use Flora-2.

Installing Flora-2

- Prerequisite:
 - FLORA-2 relies on the XSB inference engine to run (<http://xsb.sourceforge.net>).
 - Instructions for compiling XSB can be found in the XSB manual:
 - <http://xsb.sourceforge.net/manual1/manual1.pdf>
 - XSB binaries are available for Windows here: <http://xsb.sourceforge.net/downloads>
 - Development Tools to compile XSB: To compile XSB, you'll need a C compiler:
 - For the Mac, you'll likely want the XTools package
 - <http://developer.apple.com/technologies/tools/xcode.html>
 - For Windows, Microsoft Visual C++ Express Edition will work
 - <http://www.microsoft.com/visualstudio>
 - For Windows, you also need Microsoft's *nmake* (downloaded as part of Visual C++ Express above).

Installing Flora-2

- Configuring FLORA-2 under Windows (8, 7 or XP, 64bit or 32bit) with Microsoft Visual C++:

- Execute the following commands (we assume Flora-2 is downloaded in *C:\flora2*):

```
cd C:\flora2
```

```
makeflora.bat
```

- The last command will work only if an XSB executable is on the system search path specified by the environment variable PATH. If it is not, then additional parameters need to be supplied. If we assume that XSB is installed in the directory *C:\XSB*, then we run:

```
makeflora.bat C:\XSB\bin\xsb.bat
```

- If you configured a 64-bit version of XSB, then configure FLORA-2 using

```
makeflora C:\XSB\bin\xsb64.bat
```

- You will be able to run FLORA-2 by typing

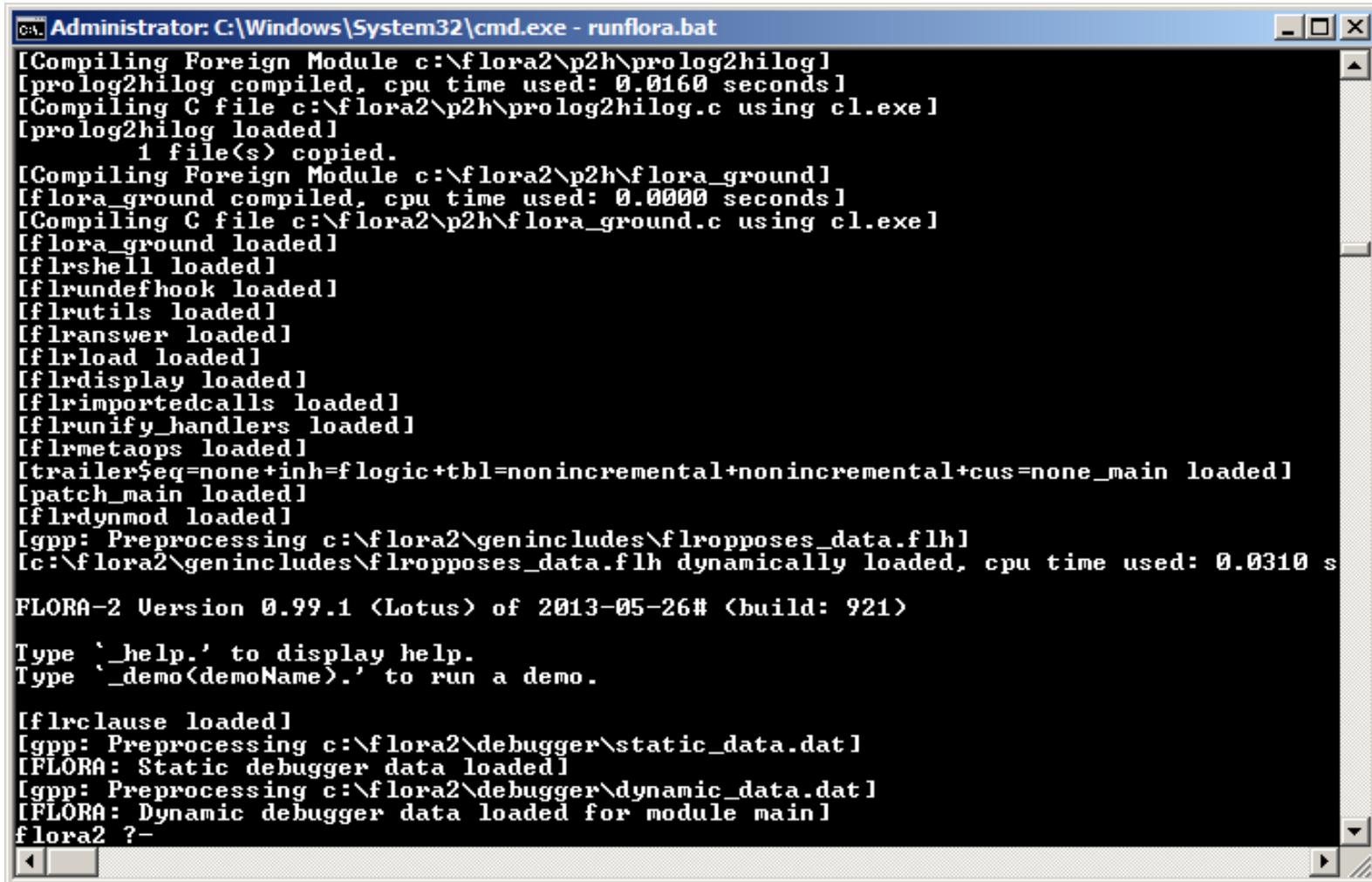
```
C:\flora2\runflora.bat
```

Using Flora-2 – The Basics

- to start the Flora-2 engine:
 - It is recommended that you add the flora2 directory to the PATH environment variable
 - In Windows, click right on “Computer” in Start, select “Properties”, select “Advanced” and change Environment Variable PATH to include the flora2 path – for example, add “.C:\flora2” if your Flora-2 directory is C:\flora2
 - Execute the command:
runflora
- file editing:
 - Flora-2 files have the extension *.flr* and can be edited in your preferred editor (Notepad++, Emacs, etc.)
- to load a file into the engine:
 - Execute the following command in Flora-2 console (assume that your file is *file.flr*):
\load('file').
(the suffix *.flr* can be added, but is not necessary)
 - **Note this is cumulative. previously loaded rules remain in the engine’s rule base until that is cleared out.**
- to ask a query:
 - type the query into the Flora-2 console and hit enter,
 - results are immediately displayed below.

Starting Flora-2 the First Time

- If you have just installed Flora-2, start *runflora* into the current directory:



```
Administrator: C:\Windows\System32\cmd.exe - runflora.bat
[Compiling Foreign Module c:\flora2\p2h\prolog2hilog]
[prolog2hilog compiled, cpu time used: 0.0160 seconds]
[Compiling C file c:\flora2\p2h\prolog2hilog.c using cl.exe]
[prolog2hilog loaded]
    1 file(s) copied.
[Compiling Foreign Module c:\flora2\p2h\flora_ground]
[flora_ground compiled, cpu time used: 0.0000 seconds]
[Compiling C file c:\flora2\p2h\flora_ground.c using cl.exe]
[flora_ground loaded]
[flrshell loaded]
[flrundefhook loaded]
[flrutils loaded]
[flranswer loaded]
[flrload loaded]
[flrdisplay loaded]
[flrimportedcalls loaded]
[flrunify_handlers loaded]
[flrmetaops loaded]
[trailer$eq=none+inh=flogic+tbl=nonincremental+nonincremental+cus=none_main loaded]
[patch_main loaded]
[flrdynmod loaded]
[gpp: Preprocessing c:\flora2\genincludes\flropposes_data.flh]
[c:\flora2\genincludes\flropposes_data.flh dynamically loaded, cpu time used: 0.0310 s]

FLORA-2 Version 0.99.1 <Lotus> of 2013-05-26# <build: 921>

Type '_help.' to display help.
Type '_demo(demoName).' to run a demo.

[flrclause loaded]
[gpp: Preprocessing c:\flora2\debugger\static_data.dat]
[FLORA: Static debugger data loaded]
[gpp: Preprocessing c:\flora2\debugger\dynamic_data.dat]
[FLORA: Dynamic debugger data loaded for module main]
flora2 ?-
```

- The Flora-2 console waits for Flora-2 queries.