Jylog

Jython + Prolog/RDF

by Brian Burns and Adam Keys

Overview

Adding Prolog and RDF to Jython.
Building on DCJython (Jython with SQL and SIM).

The name Jylog = Jython + Prolog, though already used for a Java logging library.

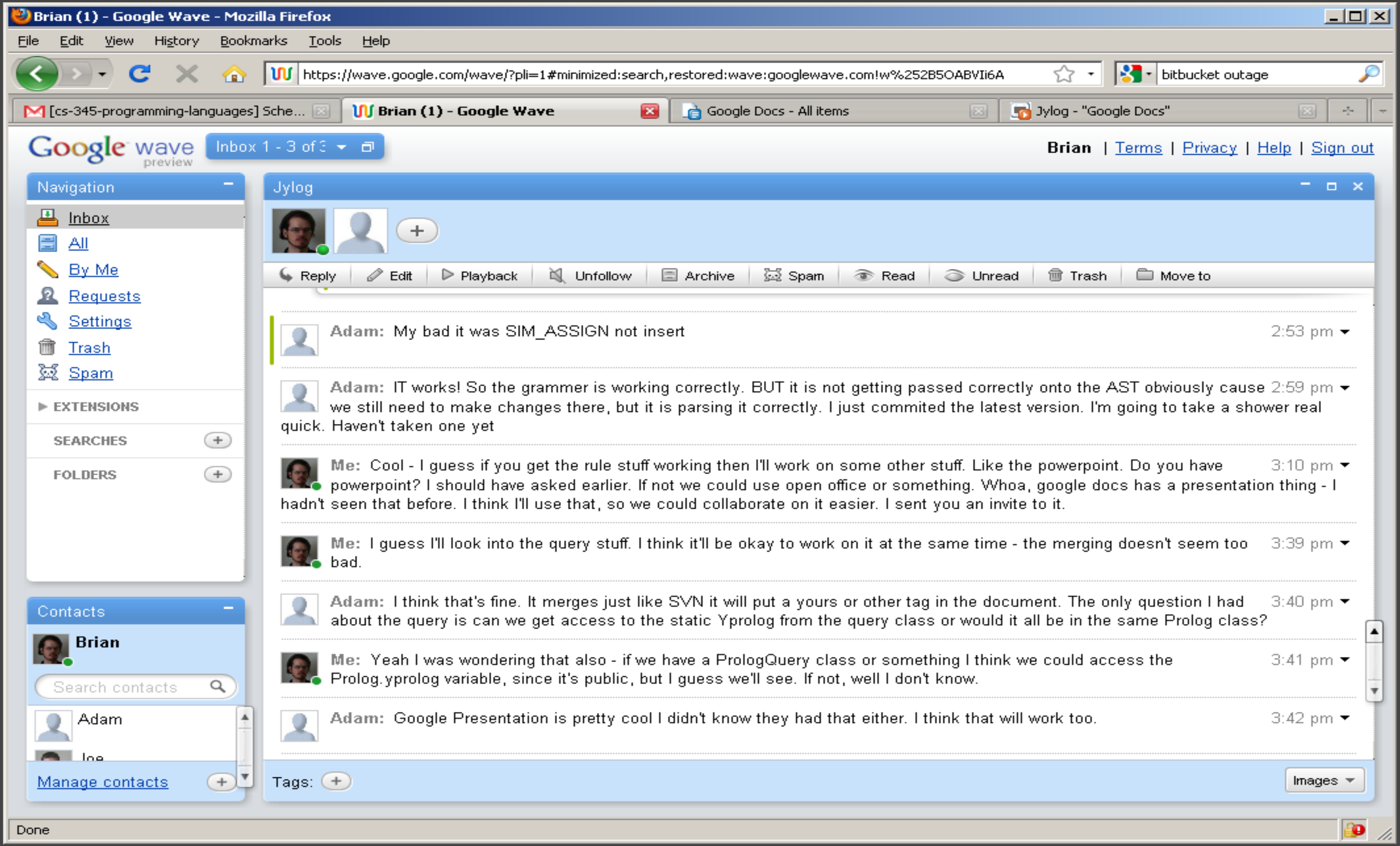
RDF is a subset of a Prolog database - just binary predicates instead of n-ary predicates. So Jylog adds binary predicates to the TDB triple store.

Tools Used

- Google Wave
- Bitbucket
- Mercurial
- Google Docs Presentation
- YProlog

Google Wave

Google Wave was used as our calibration tool.



Adam: My bad it was SIM_ASSIGN not insert 2:53 pm

Adam: IT works! So the grammer is working correctly. BUT it is not getting passed correctly onto the AST obviously cause we still need to make changes there, but it is parsing it correctly. I just commited the latest version. I'm going to take a shower real quick. Haven't taken one yet 2:59 pm

Me: Cool - I guess if you get the rule stuff working then I'll work on some other stuff. Like the powerpoint. Do you have powerpoint? I should have asked earlier. If not we could use open office or something. Whoa, google docs has a presentation thing - I hadn't seen that before. I think I'll use that, so we could collaborate on it easier. I sent you an invite to it. 3:10 pm

Me: I guess I'll look into the query stuff. I think it'll be okay to work on it at the same time - the merging doesn't seem too bad. 3:39 pm

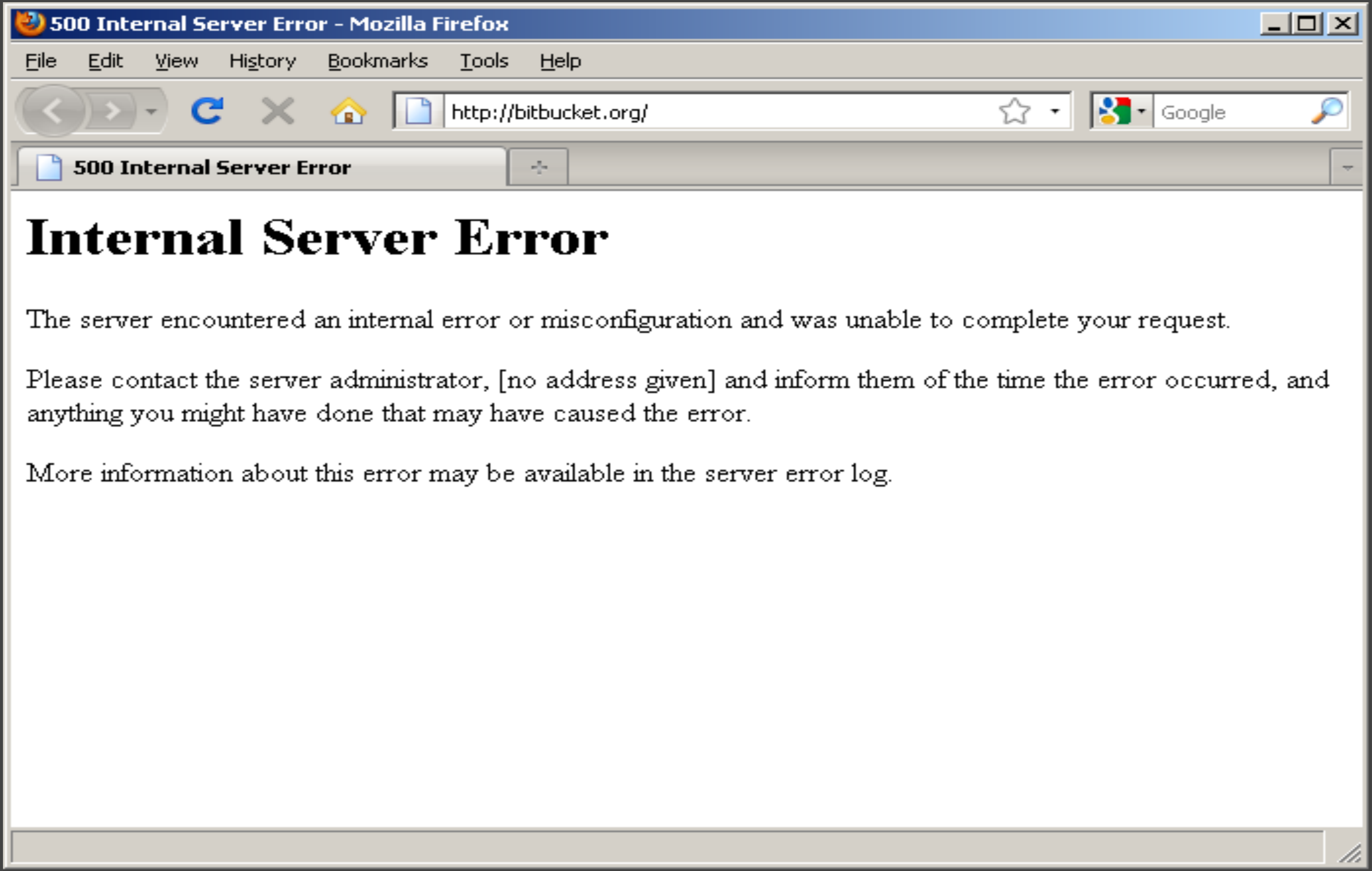
Adam: I think that's fine. It merges just like SVN it will put a yours or other tag in the document. The only question I had about the query is can we get access to the static Yprolog from the query class or would it all be in the same Prolog class? 3:40 pm

Me: Yeah I was wondering that also - if we have a PrologQuery class or something I think we could access the Prolog.yprolog variable, since it's public, but I guess we'll see. If not, well I don't know. 3:41 pm

Adam: Google Presentation is pretty cool I didn't know they had that either. I think that will work too. 3:42 pm

Bitbucket.org

We used BitBucket for our source control host using Mercurial.


500 Internal Server Error - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://bitbucket.org/
Google
500 Internal Server Error

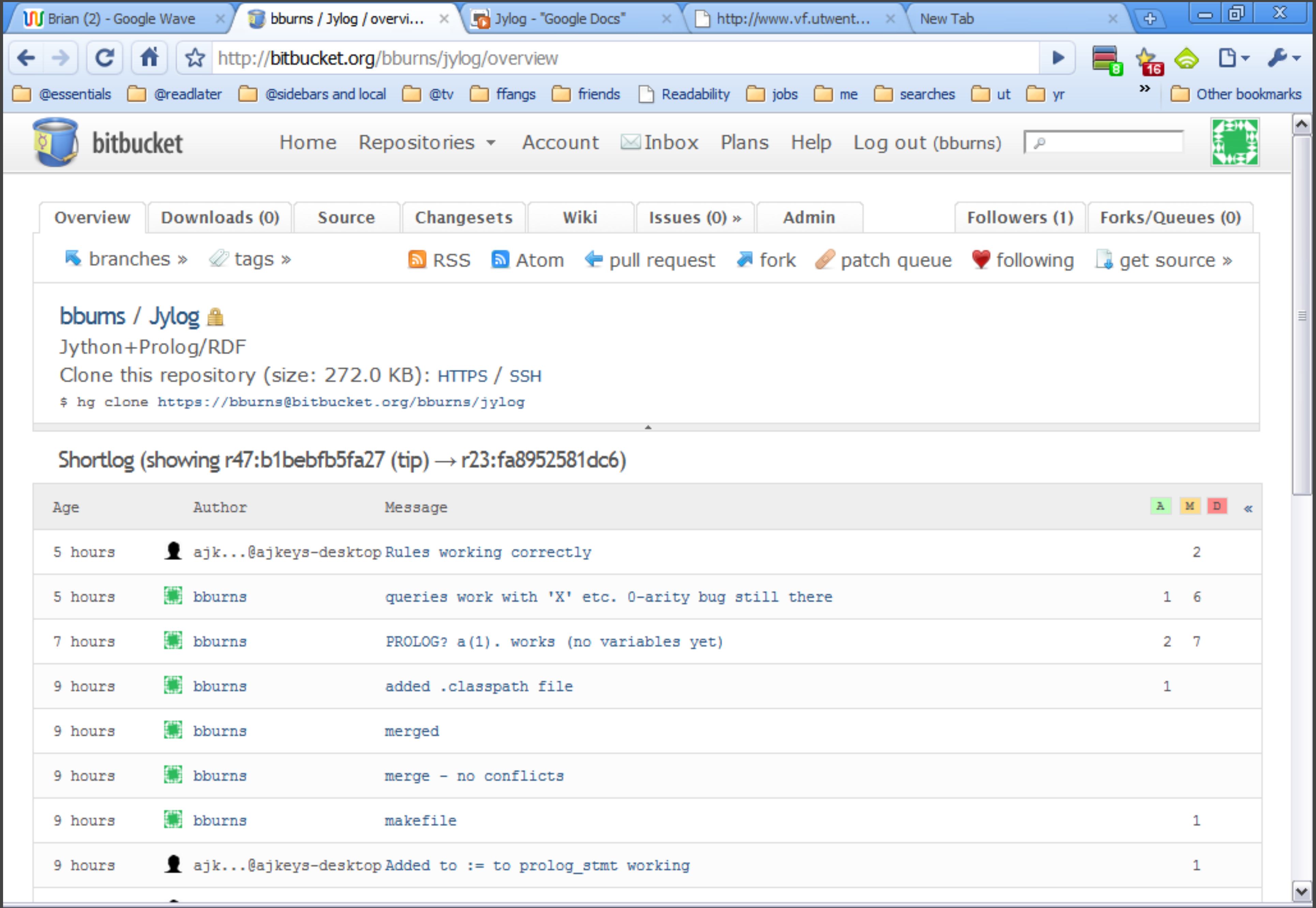
Internal Server Error

The server encountered an internal error or misconfiguration and was unable to complete your request.

Please contact the server administrator, [no address given] and inform them of the time the error occurred, and anything you might have done that may have caused the error.

More information about this error may be available in the server error log.

Bitbucket.org (when it's working)



bburns / Jylog

Jython+Prolog/RDF

Clone this repository (size: 272.0 KB): HTTPS / SSH

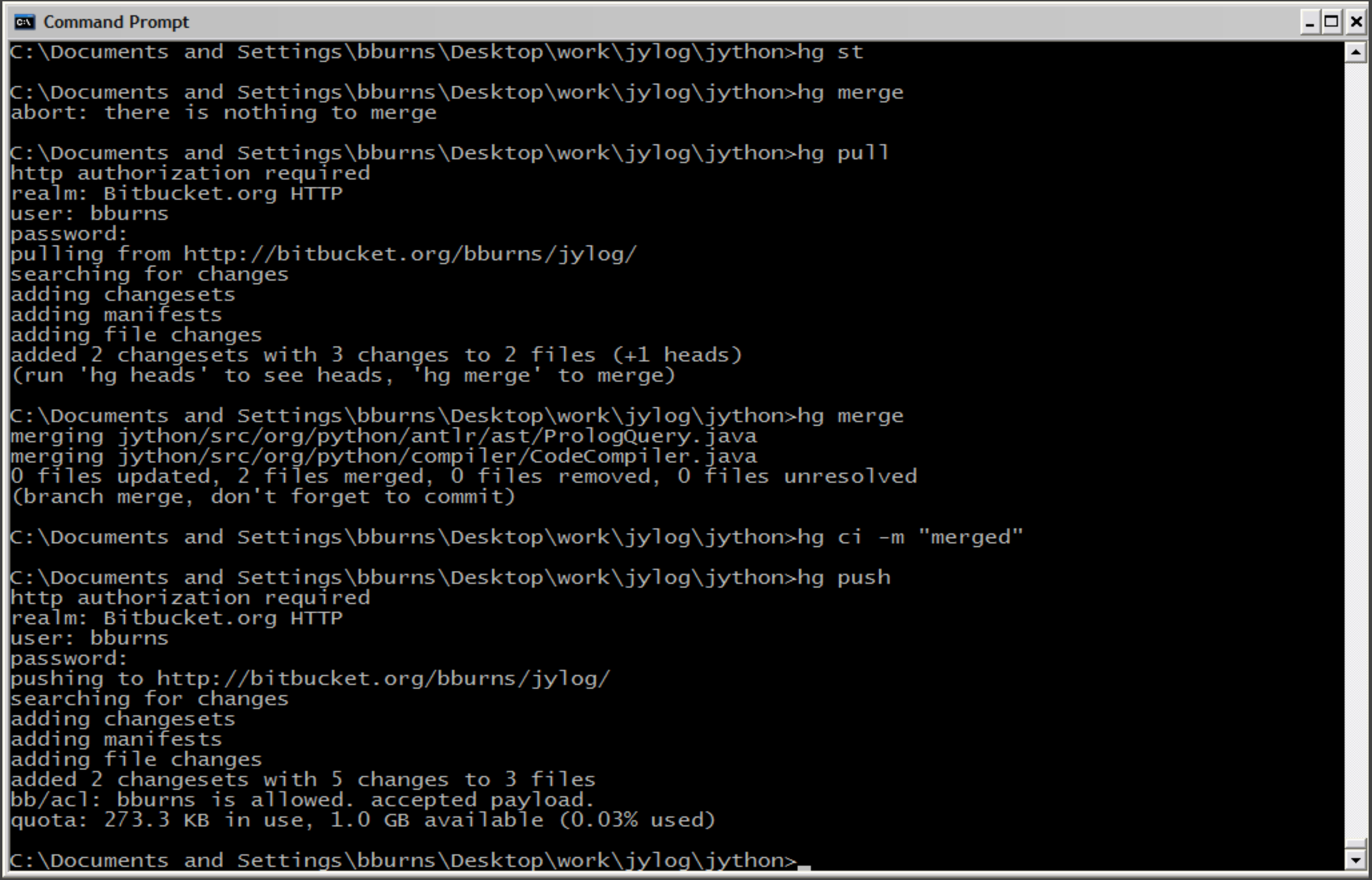
```
$ hg clone https://bburns@bitbucket.org/bburns/jylog
```

Shortlog (showing r47:b1bebfb5fa27 (tip) → r23:fa8952581dc6)

Age	Author	Message	A	M	D
5 hours	ajk...@ajkeys-desktop	Rules working correctly		2	
5 hours	bburns	queries work with 'X' etc. 0-arity bug still there	1	6	
7 hours	bburns	PROLOG? a(1). works (no variables yet)	2	7	
9 hours	bburns	added .classpath file	1		
9 hours	bburns	merged			
9 hours	bburns	merge - no conflicts			
9 hours	bburns	makefile		1	
9 hours	ajk...@ajkeys-desktop	Added to := to prolog_stmt working		1	

Mercurial

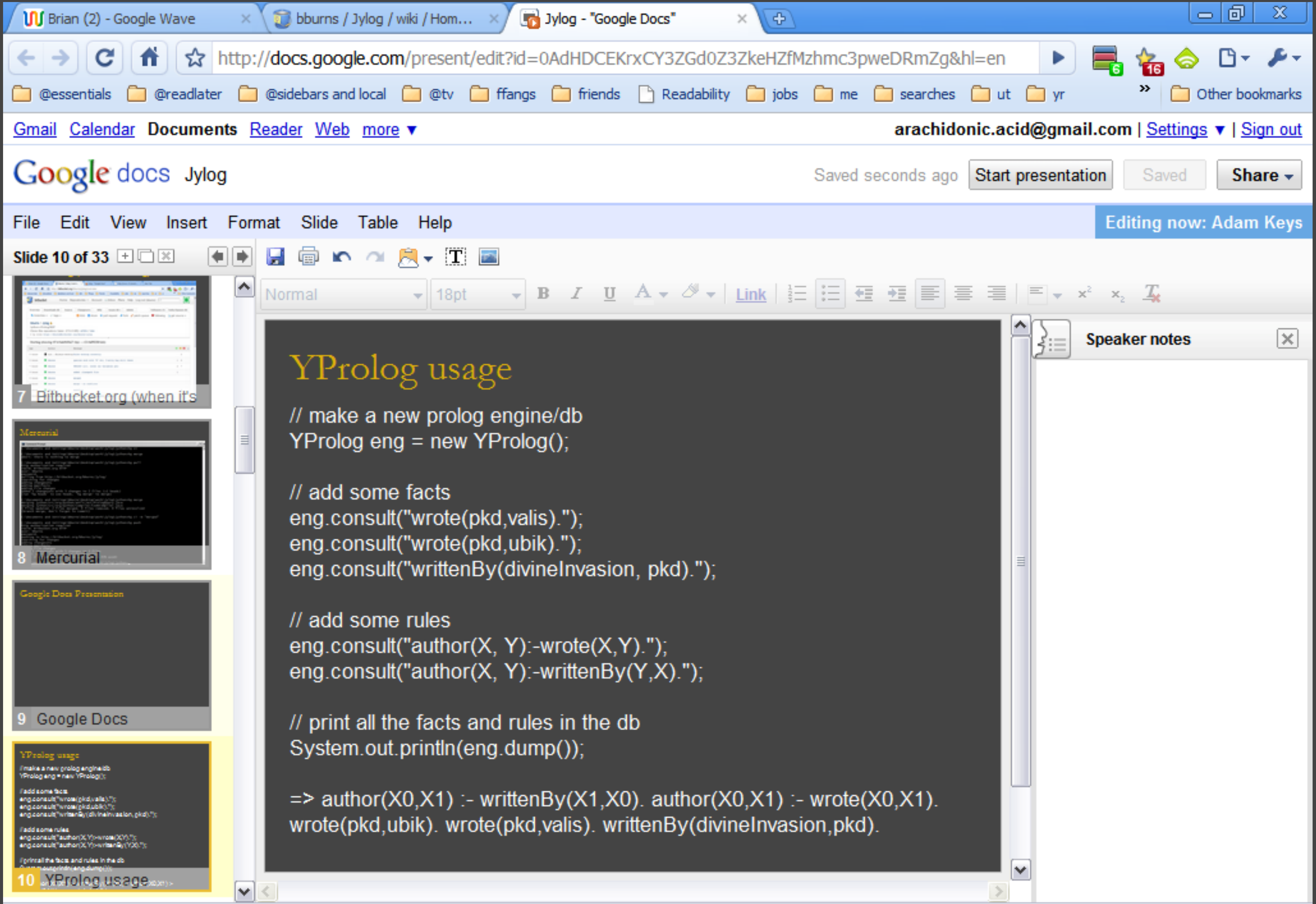
Distributed revision control system



Command Prompt

```
C:\Documents and Settings\bburns\Desktop\work\jylog\jython>hg st

C:\Documents and Settings\bburns\Desktop\work\jylog\jython>hg merge
abort: there is nothing to merge

C:\Documents and Settings\bburns\Desktop\work\jylog\jython>hg pull
http authorization required
realm: Bitbucket.org HTTP
user: bburns
password:
pulling from http://bitbucket.org/bburns/jylog/
searching for changes
adding changesets
adding manifests
adding file changes
added 2 changesets with 3 changes to 2 files (+1 heads)
(run 'hg heads' to see heads, 'hg merge' to merge)

C:\Documents and Settings\bburns\Desktop\work\jylog\jython>hg merge
merging jython/src/org/python/antlr/ast/PrologQuery.java
merging jython/src/org/python/compiler/CodeCompiler.java
0 files updated, 2 files merged, 0 files removed, 0 files unresolved
(branch merge, don't forget to commit)

C:\Documents and Settings\bburns\Desktop\work\jylog\jython>hg ci -m "merged"

C:\Documents and Settings\bburns\Desktop\work\jylog\jython>hg push
http authorization required
realm: Bitbucket.org HTTP
user: bburns
password:
pushing to http://bitbucket.org/bburns/jylog/
searching for changes
adding changesets
adding manifests
adding file changes
added 2 changesets with 5 changes to 3 files
bb/acl: bburns is allowed. accepted payload.
quota: 273.3 KB in use, 1.0 GB available (0.03% used)

C:\Documents and Settings\bburns\Desktop\work\jylog\jython>_
```

Google Docs Presentation



Brian (2) - Google Wave
bburns / Jylog / wiki / Hom...
Jylog - "Google Docs"
http://docs.google.com/present/edit?id=0AdHDCEKrxCY3ZGd0Z3ZkeHZfMzhmc3pweDRmZg&hl=en
@essentials @readlater @sidebars and local @tv ffangs friends Readability jobs me searches ut yr Other bookmarks
Gmail Calendar Documents Reader Web more
arachidonic.acid@gmail.com | Settings | Sign out
Google docs Jylog
Saved seconds ago
Start presentation
Saved
Share
File Edit View Insert Format Slide Table Help
Editing now: Adam Keys
Slide 10 of 33
Normal
18pt
Link
Speaker notes
7 Bitbucket.org (when it's
8 Mercurial
9 Google Docs
10 YProlog usage

YProlog usage

```
// make a new prolog engine/db
YProlog eng = new YProlog();

// add some facts
eng.consult("wrote(pkd,valis).");
eng.consult("wrote(pkd,ubik).");
eng.consult("writtenBy(divineInvasion, pkd).");

// add some rules
eng.consult("author(X, Y):-wrote(X,Y).");
eng.consult("author(X, Y):-writtenBy(Y,X).");

// print all the facts and rules in the db
System.out.println(eng.dump());

=> author(X0,X1) :- writtenBy(X1,X0). author(X0,X1) :- wrote(X0,X1).
wrote(pkd,ubik). wrote(pkd,valis). writtenBy(divineInvasion,pkd).
```

Prolog Engines

Some of the freely available Prolog Engines

- tuProlog
- WProlog -> XProlog -> YProlog
- gnuprolog
- hobstick

YProlog usage

```
// make a new prolog engine/db
YProlog eng = new YProlog();

// add some facts
eng.consult("wrote(pkd,valis).");
eng.consult("wrote(pkd,ubik).");
eng.consult("writtenBy(divineInvasion, pkd).");

// add some rules
eng.consult("author(X, Y):-wrote(X,Y).");
eng.consult("author(X, Y):-writtenBy(Y,X).");

// print all the facts and rules in the db
System.out.println(eng.dump());

=> author(X0,X1) :- writtenBy(X1,X0). author(X0,X1) :- wrote(X0,X1).
wrote(pkd,ubik). wrote(pkd,valis). writtenBy(divineInvasion,pkd).
```

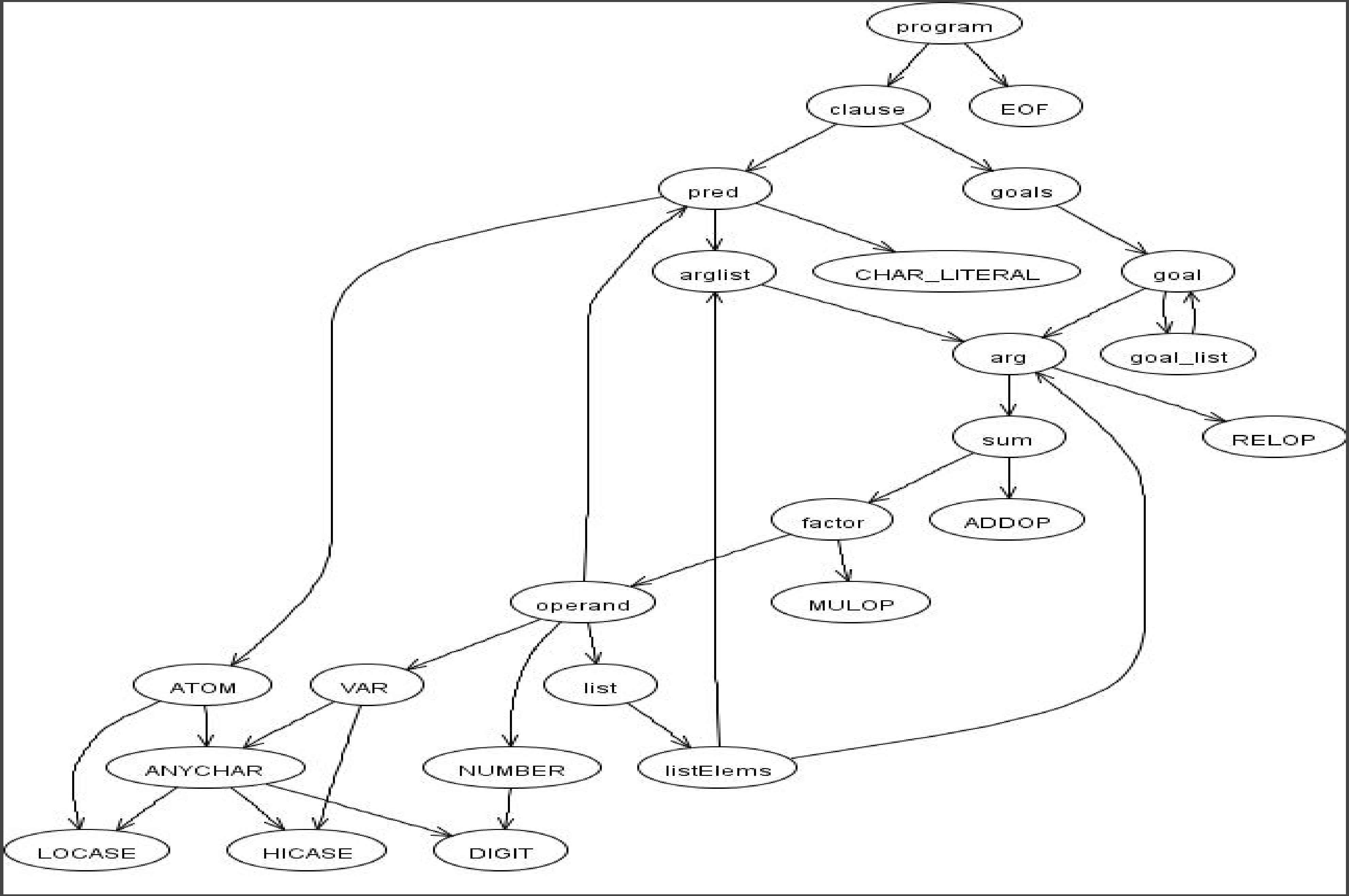
YProlog usage, ctd.

```
// a query can return just one result, like this
System.out.println( eng.queryToString("author(pkd,valis).") );
=> author(pkd,valis)

// or a given max number of results
System.out.println( eng.queryToString("author(pkd,X).", 10, " ")
);
=> author(pkd,divineInvasion). author(pkd,ubik). author(pkd,
valis).

// or ALL the results at once
System.out.println( eng.queryToString("author(pkd,X).", 0, " ") );
```

XProlog grammar

```
program    : clause+ EOF
clause     : pred [':-' goals] '.'
primitives : primitive+ EOF
primitive  : goal [':=' NUMBER | ':-' goals] '.'
goals      : goal (',' goal)*
pred       : ATOM [ '(' arglist ')' ]
           | CHAR_LITERAL
arglist    : arg (',' arg)*
goal       : '(' goal_list [ '->' goal_list ] [ ';' goal_list ] ')'
           | arg
           | '!'
goal_list  : goal (',' goal)*
arg        : sum (RELOP sum)*
sum        : factor (ADDOP factor)*
factor     : operand (MULOP operand)*
operand    : pred
           | VAR
           | NUMBER
           | list
list       : '[' listElems? ']'
listElems  : arglist [ '|' arg ]
```

XProlog grammar



Jylog Features

Jylog includes all of Jython's features plus the addition of Prolog statements. Jylog can accept a fact, rule or query.

To add a fact the syntax is:

```
PROLOG border('sussex','kent').
```

To add a rule the syntax is:

```
PROLOG adjacent('X', 'Y') := border('X', 'Y').
```

Finally, to send a query to Jylog the syntax is:

```
PROLOG? adjacent('X', 'kent').
```

Jylog Rule Syntax

You may have noticed the peculiar syntax of the rule statment. We ended up having to use ':=' as part of the rule syntax instead of ':-' because Jython can have statements that are of the form:

```
myList[1:-3]
```

Declaring ':-' as a token caused Jython to no longer correctly recognize the above line of code. Because of this, we chose to use ':=' because of the similar look and its similar location on the keyboard.

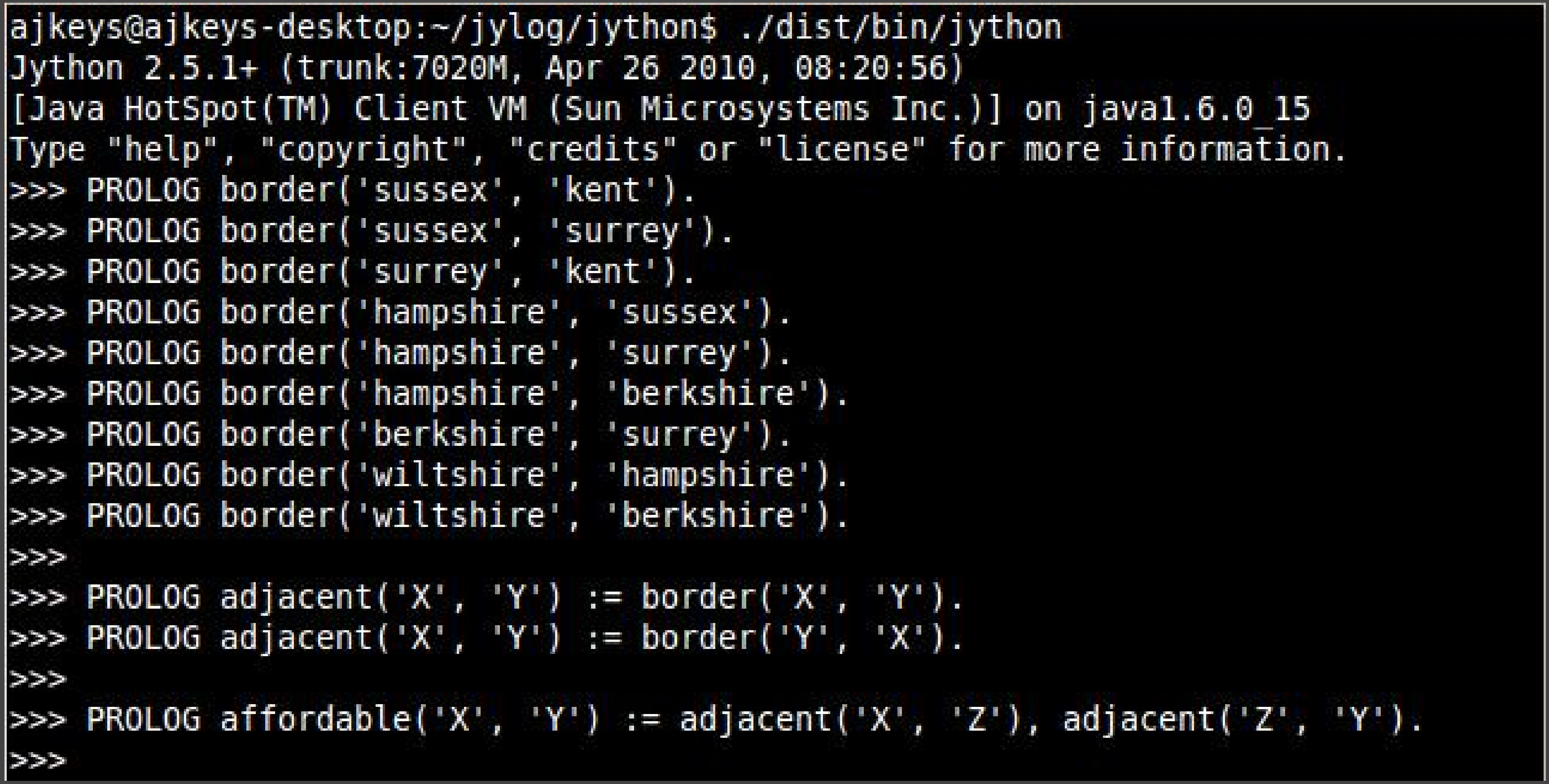
Jylog Example

This code is from Homework 7 so it should look familiar.

```
PROLOG border('sussex', 'kent').
PROLOG border('sussex', 'surrey').
PROLOG border('surrey', 'kent').
PROLOG border('hampshire', 'sussex').
PROLOG border('hampshire', 'surrey').
PROLOG border('hampshire', 'berkshire').
PROLOG border('berkshire', 'surrey').
PROLOG border('wiltshire', 'hampshire').
PROLOG border('wiltshire', 'berkshire').

PROLOG adjacent('X', 'Y') := border('X', 'Y').
PROLOG adjacent('X', 'Y') := border('Y', 'X').

PROLOG affordable('X', 'Y') := adjacent('X', 'Z'), adjacent('Z', 'Y').
```

We can easily load this code into Jylog through copy and paste or a python script.

Jylog Example (continued)

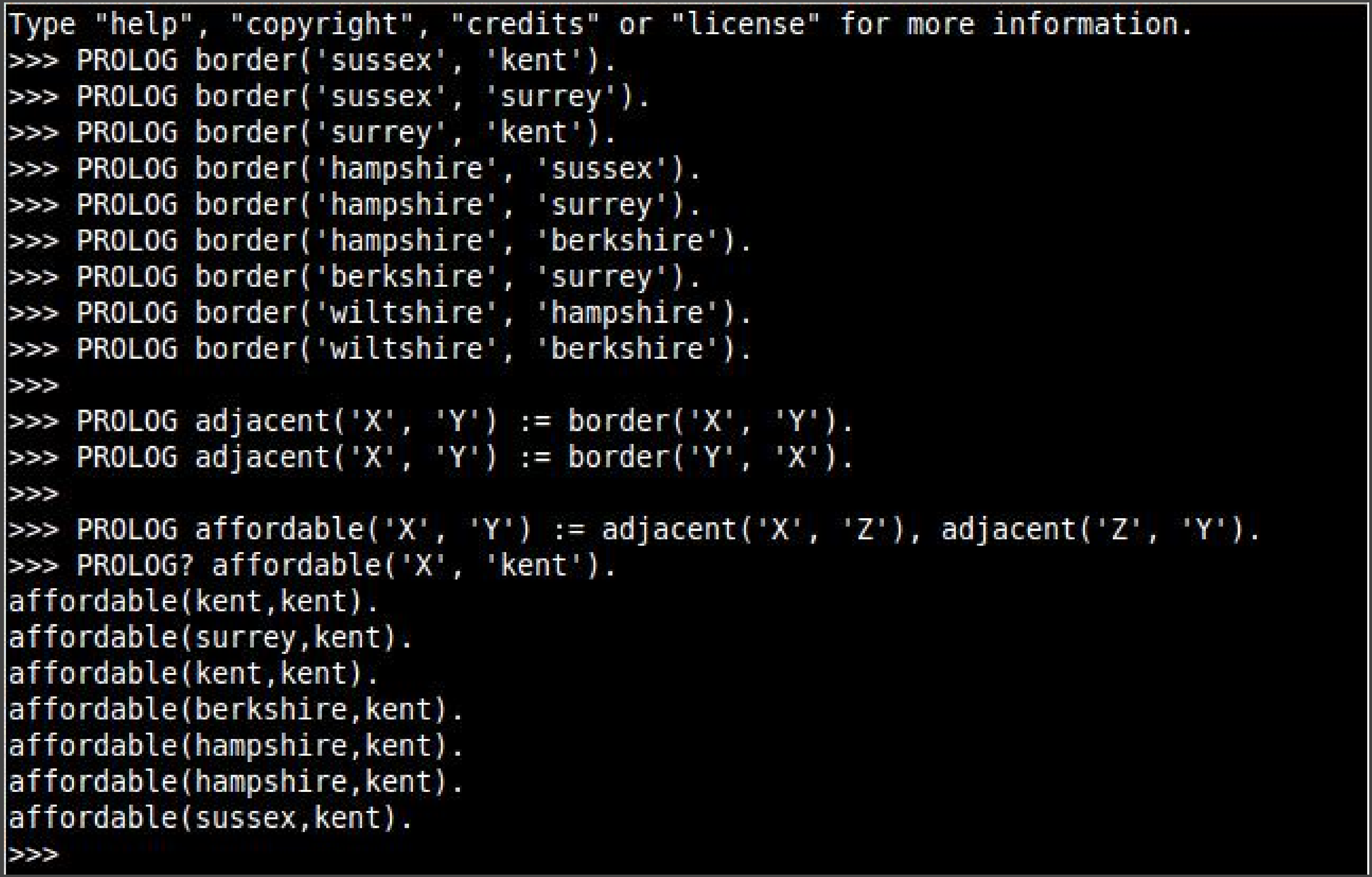


```
ajkeys@ajkeys-desktop:~/jylog/jython$ ./dist/bin/jython
Jython 2.5.1+ (trunk:7020M, Apr 26 2010, 08:20:56)
[Java HotSpot(TM) Client VM (Sun Microsystems Inc.)] on java1.6.0_15
Type "help", "copyright", "credits" or "license" for more information.
>>> PROLOG border('sussex', 'kent').
>>> PROLOG border('sussex', 'surrey').
>>> PROLOG border('surrey', 'kent').
>>> PROLOG border('hampshire', 'sussex').
>>> PROLOG border('hampshire', 'surrey').
>>> PROLOG border('hampshire', 'berkshire').
>>> PROLOG border('berkshire', 'surrey').
>>> PROLOG border('wiltshire', 'hampshire').
>>> PROLOG border('wiltshire', 'berkshire').
>>>
>>> PROLOG adjacent('X', 'Y') := border('X', 'Y').
>>> PROLOG adjacent('X', 'Y') := border('Y', 'X').
>>>
>>> PROLOG affordable('X', 'Y') := adjacent('X', 'Z'), adjacent('Z', 'Y').
>>>
```

We can now send queries to Jylog of the form:

PROLOG? affordable('X', 'kent').

Jylog Example (continued)



```
Type "help", "copyright", "credits" or "license" for more information.
>>> PROLOG border('sussex', 'kent').
>>> PROLOG border('sussex', 'surrey').
>>> PROLOG border('surrey', 'kent').
>>> PROLOG border('hampshire', 'sussex').
>>> PROLOG border('hampshire', 'surrey').
>>> PROLOG border('hampshire', 'berkshire').
>>> PROLOG border('berkshire', 'surrey').
>>> PROLOG border('wiltshire', 'hampshire').
>>> PROLOG border('wiltshire', 'berkshire').
>>>
>>> PROLOG adjacent('X', 'Y') := border('X', 'Y').
>>> PROLOG adjacent('X', 'Y') := border('Y', 'X').
>>>
>>> PROLOG affordable('X', 'Y') := adjacent('X', 'Z'), adjacent('Z', 'Y').
>>> PROLOG? affordable('X', 'kent').
affordable(kent,kent).
affordable(surrey,kent).
affordable(kent,kent).
affordable(berkshire,kent).
affordable(hampshire,kent).
affordable(hampshire,kent).
affordable(sussex,kent).
>>>
```

As you can see the results of the query are returned to us.

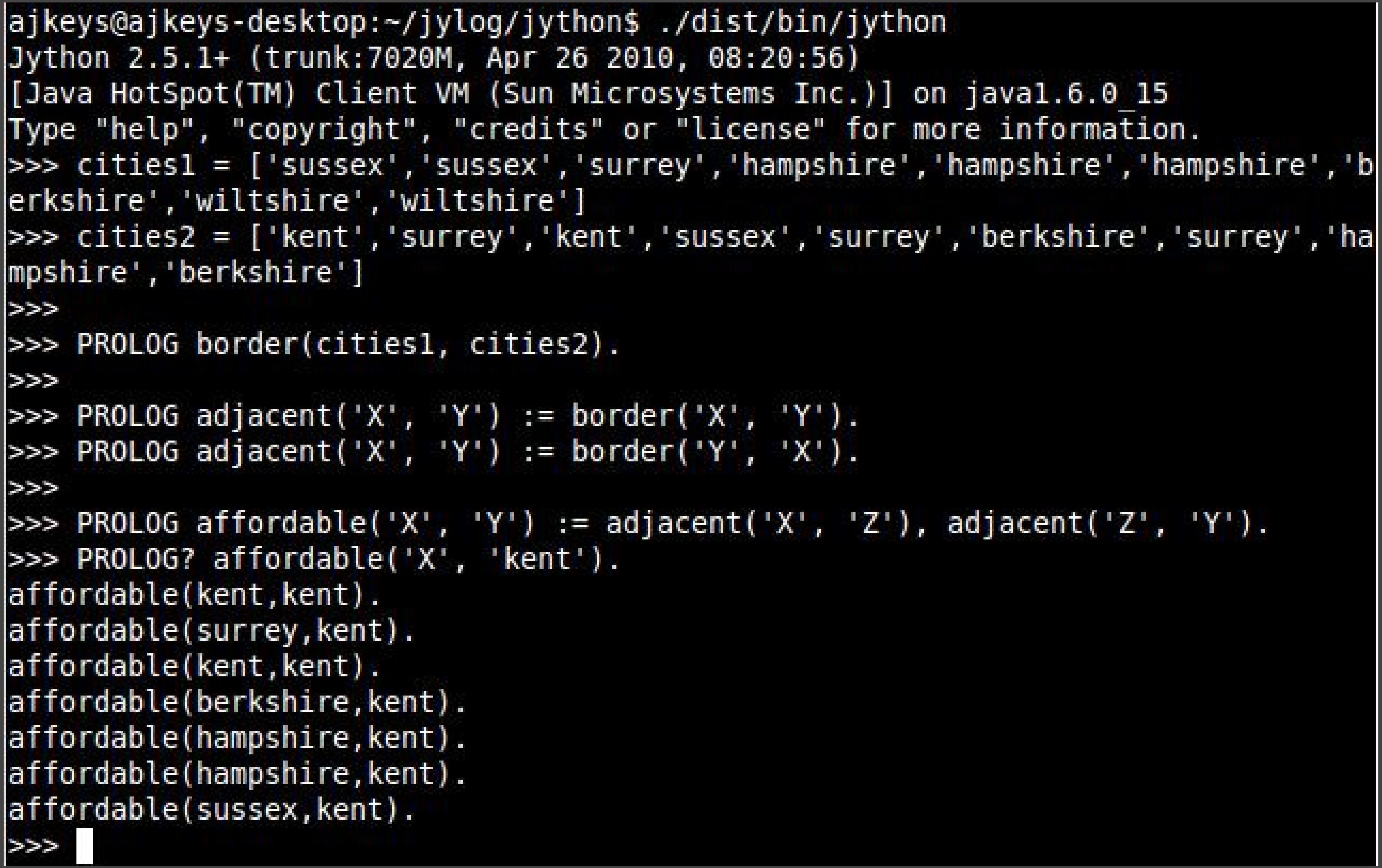
Jylog Example (A better way!)

Jylog also allows us to use lists. A key thing to know when using lists is that they can only be used in a fact and when using multiple lists they MUST be of the same length. This syntax allows you to relate two or more lists with the same predicate. So the problem from the previous slide can be rewritten as:

```
cities1 = ['sussex','sussex','surrey','hampshire','hampshire','hampshire','berkshire','wiltshire','wiltshire']
cities2 = ['kent','surrey','kent','sussex','surrey','berkshire','surrey','hampshire','berkshire']

PROLOG border(cities1, cities2).

PROLOG adjacent('X', 'Y') := border('X', 'Y').
PROLOG adjacent('X', 'Y') := border('Y', 'X').

PROLOG affordable('X', 'Y') := adjacent('X', 'Z'), adjacent('Z', 'Y').
```

Loading this code into Jylog and sending it the same query as before gives us...

Jylog Example (A better way!)



```
ajkeys@ajkeys-desktop:~/jylog/jython$ ./dist/bin/jython
Jython 2.5.1+ (trunk:7020M, Apr 26 2010, 08:20:56)
[Java HotSpot(TM) Client VM (Sun Microsystems Inc.)] on java1.6.0_15
Type "help", "copyright", "credits" or "license" for more information.
>>> cities1 = ['sussex','sussex','surrey','hampshire','hampshire','hampshire','b
erkshire','wiltshire','wiltshire']
>>> cities2 = ['kent','surrey','kent','sussex','surrey','berkshire','surrey','ha
mpshire','berkshire']
>>>
>>> PROLOG border(cities1, cities2).
>>>
>>> PROLOG adjacent('X', 'Y') := border('X', 'Y').
>>> PROLOG adjacent('X', 'Y') := border('Y', 'X').
>>>
>>> PROLOG affordable('X', 'Y') := adjacent('X', 'Z'), adjacent('Z', 'Y').
>>> PROLOG? affordable('X', 'kent').
affordable(kent,kent).
affordable(surrey,kent).
affordable(kent,kent).
affordable(berkshire,kent).
affordable(hampshire,kent).
affordable(hampshire,kent).
affordable(sussex,kent).
>>> █
```

As you can see this gives us the same result set and is easier to write. With lists the possibilities are endless! You could even declare 100 vertex facts easily like this "PROLOG vertex(range(100))."!

Adding to the triple-store

A triple store just stores binary predicates, so it is a subset of a Prolog database. ie

```
  wrote(pkd, valis).
```

is the triple

```
  (pkd, wrote, valis)
```

```
// add binary predicates to triple store also
if (exprs.size() == 2) {
    String predicateName = this.predicateNames.get(0);
    String value1 = exprs.get(0).toString();
    String value2 = exprs.get(1).toString();

    Resource r = model.createResource(objectUri + value1);
    Property p = model.createProperty(propertyUri + predicateName);
    r.addProperty(p, value2);
    ...
}
```

Adding to the triple-store, ctd

But then you have to get the data out somehow...

```
>>> PROLOG wrote('pkd','valis').
Adding binary predicate to triple store: pkd, wrote, valis.
 WARN [main] (SetupTDB.java:702) - No BGP optimizer

>>> PROLOG wrote('pkd','ubik').
Adding binary predicate to triple store: pkd, wrote, ubik.

>>> SELECT wrote FROM emp;

[nothing]
```

So, it's in there, but didn't have time to get SQL query working.

A better prolog syntax?

For interactive prolog sessions...

```
> wrote(pkd,valis).
> wrote(pkd,ubik).
> wrote(pkd,ubik)?
true
> wrote(pkd,X)?
[valis,ubik]
```

The problem with normal prolog syntax is that there's no difference between a fact and a query -
ie wrote(pkd,valis). could be a fact that you're declaring, or a query that you're entering at the prompt.
Putting a question mark at the end for queries distinguishes the two cases.

LL(*) grammars

ANTLR can handle LL(*) grammars - this means indefinite lookahead.

This is the default setting - to override it you'd need to set

```
options { k=3; }
```

to make it just handle LL(3) grammars.

So it should be able to distinguish a Python function call from an n-ary predicate with a period or question mark at the end -

```
prolog_query : NAME (LPAREN expr (COMMA expr)* RPAREN)?  '?';
```

But...

```
[java] error(201): grammar\Python.g: 580:12: The following alternatives can never be matched: 13
```

[referring to the this rule]

LL(*) grammars

Turn on backtrack option -

```
grammar Python;
options {
    ASTLabelType=PythonTree;
    output=AST;
    backtrack=true; // if LL(*) analysis fails, pick first alternative that matches
}
```

And... success! (?)

```
> ant antlr_gen | grep error
>
```

But... no...

```
> ant
    [javac] build\gensrc\...\PythonParser.java:17325: cannot find symbol
    [javac] symbol  : variable ctype
    [javac] location: class org.python.antlr.PythonParser
    [javac]         t=test(ctype);
    [javac]                ^
```

So backtracking seems to mess up the Jython grammar somewhere.

An OO-Prolog system?

Object-oriented and relational views of data are orthogonal:

id	title	author	date
valis	VALIS	pkd	1978
ubik	Ubik	pkd	1966
divineInvasion	The Divine Invasion	pkd	1980

```
> author(valis,pkd).
> date(valis,1978).
> wrote(Author,Book):-author(Book,Author).
> wrote(pkd,X)?
[valis,ubik,divineInvasion]

> divineInvasion.title
'The Divine Invasion'
> ubik.date = 1966
> wrote(pkd,valis,1978).        # but what about n-ary predicates like this?
> pkd.(wrote,valis)=1978       # or use accessor names for the different args?
```

How it works

In Prolog.g:

```
// put a node on the AST
-> ^(PROLOGQ<PrologQuery>[$PROLOGQ, $pred.text, (java.
util.List<expr>) $prolog_query::exprs])
```

translates to (in PythonParser.java)

```
new PrologQuery(PROLOGQ, PROLOGQ88, (pred!=null?
pred.getText():null), (java.util.List<expr>)
((prolog_query_scope)prolog_query_stack.peek()).exprs)
```

ie

```
new PrologQuery(PROLOGQ, PROLOGQ88, pred.getText(),
exprs)
```

How it works

Constructor for the PrologQuery node (in PrologQuery.java):

```
    public PrologQuery(int dummy, Token token, String
predicateName, java.util.List<expr> arguments) {
        super(token);
        PrologQueries.put(this.toString(), this);
        this.predicateName = predicateName;
        this.values = arguments;
        if (arguments == null) {
            this.values = new ArrayList<expr>();
        }
        for (PythonTree t : this.values) {
            addChild(t);
        }
    }
```

How it works

Jython then walks over the abstract syntax tree (AST), visiting the nodes -

in CodeCompiler.java:

```
public Object visitPrologQuery(PrologQuery node) throws Exception {
    setline(node);
    if (node.getInternalValues().size() > 0) { // put exprs on stack
        for (int i = 0; i < node.getInternalValues().size(); i++) {
            visit(node.getInternalValues().get(i)); // put expr on stack
            code.invokestatic(p(Py.class), "prologQueryClause",
                              sig(Void.TYPE, PyObject.class)); // evaluate it
        }
        visit(node.getInternalValues().get(0)); // put dummy on stack
        code.ldc(node.toString());
        code.invokestatic(p(Py.class), "prologQueryClauseFinal",
                          sig(Void.TYPE, PyObject.class, String.class));
    }
    else { // handle 0-arity predicates also
        PyObject po = new PyInteger(1);
        Num n = new Num(po); visit(n); // compiles
        code.ldc(node.toString());
        code.invokestatic(p(Py.class), "prologClauseFinal",
                          sig(Void.TYPE, PyObject.class, String.class));
    }
```

How it works

In Py.java:

```
    static private List<PyObject> prologQueryExprs = new
ArrayList<PyObject>();

    public static void prologQueryClause(PyObject o) {
        //System.out.println("prologQueryClause");
        prologQueryExprs.add(o);
    }
    public static void prologQueryClauseFinal(PyObject o, String s) {
        //System.out.println("prologQueryClauseFinal");
        //System.out.println(prologQueryExprs);
        PrologQuery.prologQueryGetInsertProcessor(prologQueryExprs, s);
        prologQueryExprs.clear();
    }
```

How it works

back in PrologQuery.java:

```
    public static void prologQueryGetInsertProcessor(java.util.List<PyObject> exprs, String node_name) {
     //System.out.println("prologQueryGetInsertProcessor");
     PrologQueries.get(node_name).prologQueryProcessInsert(exprs);
 }


 public void prologQueryProcessInsert(java.util.List<PyObject> exprs) {

     // build the prolog clause to pass to the prolog engine
     StringBuilder sb = new StringBuilder();
     sb.append(predicateName);
     if (!exprs.isEmpty()) {
         sb.append("(");
         String sep = "";
         for (PyObject po : exprs) {
             sb.append(sep).append(po);
             sep = ", ";
         }
         sb.append(")");
         sb.append(".");
     }
     String query = sb.toString();

     // ask the prolog engine
     //System.out.printf("query prolog with %s\n", query);
     // String answer = Prolog.yprolog.queryToString(query); // return first result only
     String answer = Prolog.yprolog.queryToString(query, 0, "\n"); // return ALL results, separated by linefeeds
     System.out.println(answer);
```

Dummy values

Putting a dummy value on the stack -

```
visit(node.getInternalValues().get(0));
```

For the 0-arity predicates, there were no internal values available, so did this -

```
visit(new Num(new PyInteger(1)));
```

Semantic web

It would have been nice to be able to tie into the semantic web, which is basically a gigantic triple store.

So could do Prolog queries and inference on the data -

```
> wrote = http://predicates.com/wrote
> pkd = http://authors.com/philip_k_dick

> wrote(pkd, X)?
[valis, ubik, the_divine_invasion, ...]
```

But that's in version 2.0...