

Sigma Knowledge Engineering Environment

Installation Instructions and User Guide for Sigma 2.01

10/14/2007

Written by Adam Pease and Nick Siegel

Please send questions or comments to:

apeace@articulatesoftware.com

nsiegel@articulatesoftware.com

License

This code and documentation are copyright Articulate Software (c) 2005, 2006, 2007. Some portions copyright Teknowledge (c) 2003 and reused under the terms of the GNU license. This software is released under the GNU Public License <http://www.gnu.org/copyleft/gpl.html> .

Users of this code also consent, by use of this code, to credit Articulate Software and Teknowledge in any writings, briefings, publications, presentations, or other representations of any software that incorporates, builds on, or uses this code. Please cite the following article in any publication with references:

Pease, A., (2003). The Sigma Ontology Development Environment. *In* Working Notes of the IJCAI-2003 Workshop on Ontology and Distributed Systems, August 9, Acapulco, Mexico.

Note that the current release of Sigma is "beta" code, and is not suitable for any mission-critical use. It is strictly a tool for research. Sigma comes with no warranty, expressed or implied. By using Sigma, you assume all responsibility for events that may result, and agree to hold Articulate Software blameless for any adverse consequences.

Bug reports, fixed code, and new, implemented features are welcome, however, and will be shared with the Sigma user community.

Table of Contents

Sigma Knowledge Engineering Environment Installation Instructions and User Guide for Sigma	
2.01 10/14/2007	0
Table of Contents.....	2
1 Introduction	3
2 Prerequisites.....	3
2.1 Directory Pathnames	3
2.1.1 Windows	4
2.1.2 Linux and Mac OS X.....	4
2.2 Memory Allocation	5
2.2.1 Windows	5
2.2.2 Linux and Mac OS X.....	5
3 Release Package Contents	6
4 Quick Start.....	8
5 Installation	11
6 Troubleshooting	13
7 User Guide.....	14
8 Reference Guide.....	21
9 References	27
10 Acknowledgments.....	27
11 Appendix: Natural Language Format.....	27
12 Appendix: Test Formats	28

1 Introduction

Sigma (Pease, 2003) is an environment for creating, testing, modifying, and performing inference with ontologies. This document explains how to install and use Sigma. Those new to Sigma are advised to read the entire document, but experienced users who simply want to install a newer version of Sigma might be able to get by with reading just the Quick Start section. Please bear in mind, however, that Sigma is a work in progress with a few rough edges, and that one purpose of this document is to help you achieve productive use of Sigma. Taking time to read this document now might save you time in the future.

2 Prerequisites

Sigma's core implementation comprises compiled Java classes, Java Server Pages, and various data files, including files of SUO-KIF statements that are loaded to build Sigma's knowledge base (KB), and files that contain mappings between concepts in the KB and WordNet synsets. To run Sigma, you will need the following hardware and software components:

- A host computer with at least 512 MB of main memory, running a Linux, Windows (NT, 2000, XP, 2003), or Mac OS X operating system, and on which are installed
- A web browser, such as Internet Explorer or Firefox;
- Version 5.0 or more recent of a Java Runtime Environment (JRE), which can be obtained from the relevant download area at <http://java.sun.com> ; and
- Version 5.5 or more recent of the Tomcat web application server, which can be obtained from the relevant download area at <http://tomcat.apache.org> .

The resources listed above constitute the basic computing infrastructure on which Sigma depends, and are not included in the Sigma release package. If you need help installing a JRE or Tomcat, please consult the pertinent documentation, or, if relevant, ask your system administrator or IT support staff for assistance.

After the required infrastructure is in place, you should perform a few configuration steps before proceeding to install and run Sigma. These steps involve setting a few environment variables so that Sigma can more easily find certain directory pathnames, and configuring Tomcat to allocate enough memory for Sigma to manage large KBs.

2.1 Directory Pathnames

The Sigma installation scripts and the Sigma program itself will try to use the values of the environment variables `SIGMA_HOME` and `CATALINA_HOME`, if the values are valid directory pathnames. `SIGMA_HOME` should be set to the absolute pathname of the directory in which you want Sigma's own working directory, named `KBs`, to be created. `CATALINA_HOME` should be

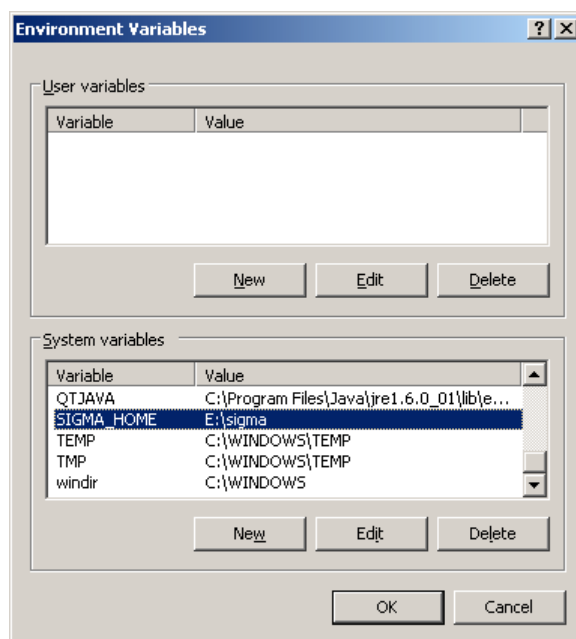


Figure 1: The Windows Environment Variables dialog

set to the absolute pathname of the Tomcat root directory, the directory in which the subdirectories `bin` and `webapps` are located.

2.1.1 Windows

To create and set the environment variables `SIGMA_HOME` and `CATALINA_HOME` in Windows, navigate to the user dialog named “Environment Variables”, which is part of the larger user dialog named “System Properties”. One way to reach this dialog is to follow the menu sequence `Start -> Control Panel -> System -> Advanced -> Environment Variables`. Click on the button labeled “New” to add a new variable and its value (Figure 1).

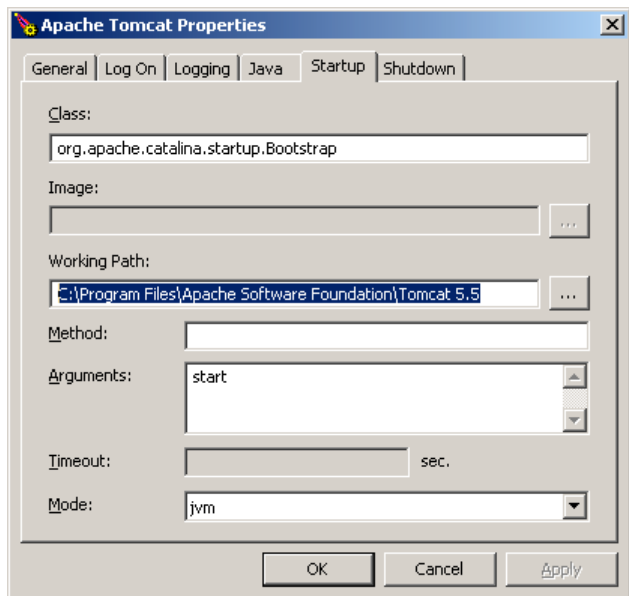


Figure 2: Tomcat's "Working Path" parameter

In most circumstances, the value of `CATALINA_HOME` should correspond to the value of Tomcat's “Working Path” parameter, which can be set by using the Configure Tomcat utility application accessible through the Windows menu sequence `Start -> All Programs -> Apache Tomcat m.n -> Configure Tomcat -> Startup`. If for some reason you are unable or choose not to set the variable `CATALINA_HOME`, then you should use Configure Tomcat to set the value labeled “Working Path” to be the pathname of Tomcat's root directory, if it does not already have this value (Figure 2).

2.1.2 Linux and Mac OS X

In Linux or other UNIX-like environments, the best way to declare and set the variables `SIGMA_HOME` and `CATALINA_HOME` is to add the necessary expressions to your login shell's initialization file. If your login shell is `bash`, for example, you can arrange for the values of `SIGMA_HOME` and `CATALINA_HOME` to be set whenever you log in by adding to your `.bashrc` file expressions like these:

```
export SIGMA_HOME=/home/username/sigma
export CATALINA_HOME=/opt/apache-tomcat-5.5.23
```

The actual pathnames to be used will depend on your local directory structure, the version of Tomcat installed, the location in which Tomcat is installed, and your own preferences. Note that these instructions assume you will be starting Tomcat from your login shell by invoking the shell script `startup.sh`, which is included in the Tomcat distribution in the subdirectory `bin`. If you typically start Tomcat by some other means, such as a server management utility, or if another user administers Tomcat, you might have to figure out a different way to make the values of `SIGMA_HOME` and `CATALINA_HOME` accessible to Sigma when it is running.

Setting the variables `SIGMA_HOME` and `CATALINA_HOME` will allow you to control where Sigma's working directory, `KBS`, is created, and will help running Sigma processes find the

working directory, but the variables are not absolutely necessary. The only datum absolutely required by Sigma's installation scripts in order to successfully install Sigma is the pathname of the Tomcat root directory. The Linux (UNIX) version of the installation script will prompt you for the information it requires. The Windows version of the installation script will try to guess the location of the Tomcat root directory, and will ask you to edit the script, supplying the correct value, if its guess is wrong. But it is better to set `SIGMA_HOME` and `CATALINA_HOME` before trying to install Sigma.

2.2 Memory Allocation

In order to build large KBs that comprise many constituent files, you will have to change

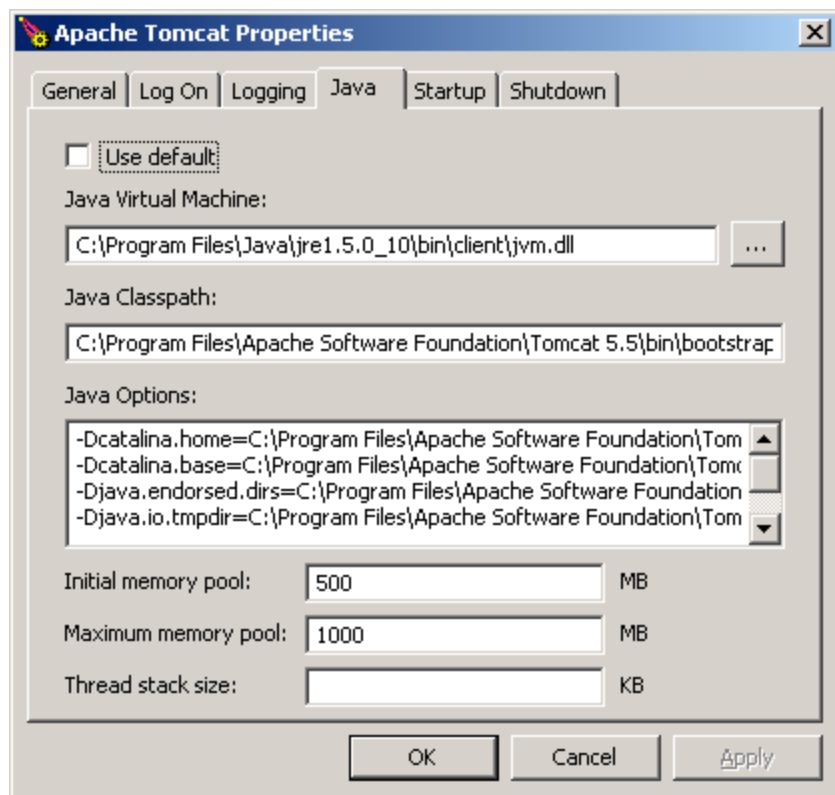


Figure 3: Setting Tomcat's memory allocation parameters with Configure Tomcat

Tomcat's default memory allocation parameters so that Sigma can access enough memory.

2.2.1 Windows

The easiest way to set Tomcat's memory allocation parameters in Windows is to use the Java tab of the Configure Tomcat utility, which is accessible via this menu sequence: Start -> All Programs -> Apache Tomcat m.n -> Configure Tomcat -> Java. Enter the value 500MB in the type-in area labeled "Initial memory pool" and the value 1000MB in the type-in area labeled "Maximum memory pool", and then click the button labeled "OK" (Figure 3).

2.2.2 Linux and Mac OS X

In Linux or other UNIX-like environments, you can arrange for Tomcat's memory allocation parameters to be determined at startup by setting the value of the environment variable `CATALINA_OPTS` in your login shell's initialization file. If your login shell is `bash`, for example, and you typically start Tomcat by invoking the script `startup.sh` that is included in the `bin` directory of the Tomcat distribution package, then you could add this expression to your `.bashrc` file:

```
export CATALINA_OPTS="$CATALINA_OPTS -Xms500M -Xmx1000M"
```

If you cannot or prefer not to add the expression above to your login shell's initialization file, another option is to edit the file `catalina.sh`, which, like `startup.sh`, is located in Tomcat's `bin` directory. You will want to add the following expression to `catalina.sh` somewhere before the point at which the command to start Java is invoked:

```
CATALINA_OPTS="$CATALINA_OPTS -Xms500M -Xmx1000M"
```

Note that the initial and maximum memory allocation parameter values recommended above are just educated guesses. You might have to experiment to determine how much memory is required to build or edit a given KB on your particular computer. If your computer has only, say, 512MB of RAM and 1.2GB of swap space, then probably you will not want to set Tomcat's maximum memory allocation value (*i.e.*, the value corresponding to the Java command line parameter `-Xmx`) to a number as high as 1000MB. If your computer has 4GB of RAM, 6GB of swap space, and you intend to build a KB comprising three million assertions, then you might want to set Tomcat's memory allocation parameters to values considerably higher than those recommended above.

3 Release Package Contents

The Sigma 2.01 release package is a ZIP archive file named `sigma-2-01.zip`. To install Sigma, you must first download the archive file to your computer's hard drive and unpack it. When unpacked, the archive will produce a directory named `sigma-2-01`, which contains several subdirectories and files. Displayed on a Windows desktop, the unpacked, opened directory (folder) `sigma-2-01` will look something like Figure 4. Before you unpack the

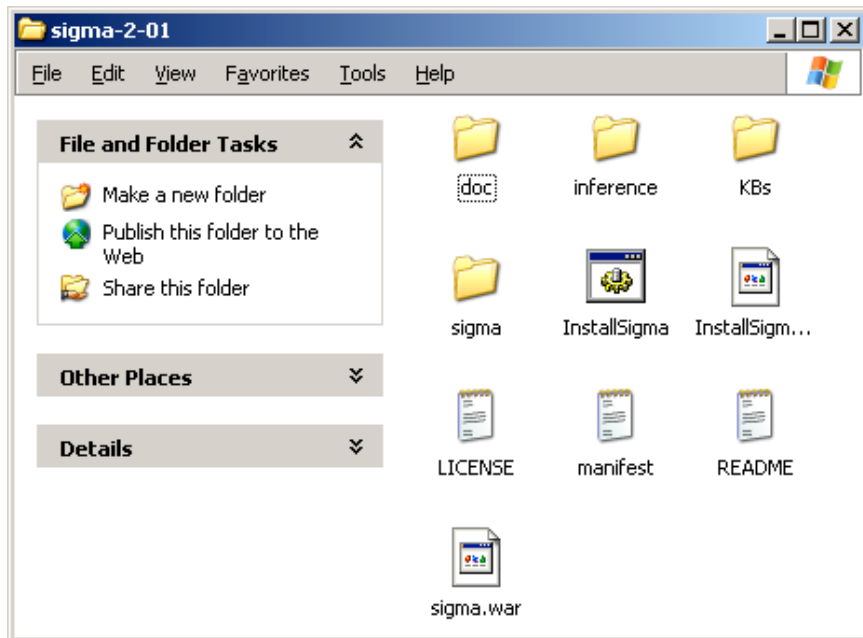


Figure 4: The directory `sigma-2-01`

archive file, be sure to move it to the location in your file system where you want the directory `sigma-2-01` to be created. In Windows XP and Server 2003, you can unpack the archive with Microsoft's Compressed Folders utility (Figure 5). In other versions of Windows, you can use any of several free or commercial archiving utilities, such as UltimateZip or WinZip. In Linux, you can unpack the archive with the `unzip` command.

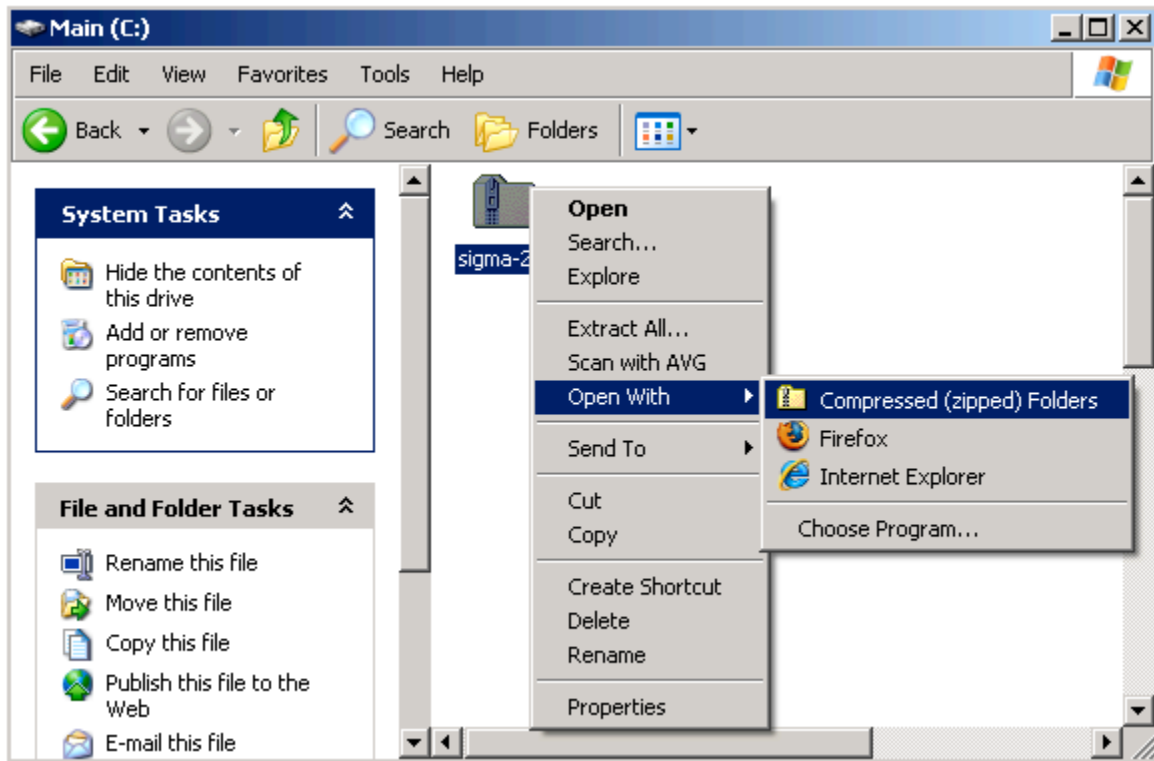


Figure 5: Using the Windows Compressed Folders utility to unpack sigma-2-01.zip

The unpacked sigma-2-01.zip archive file will yield the following hierarchy of directories and files:

```
sigma-2-01/
  doc/
    SigmaProgrammer.pdf
    SigmaUserManual-2-01.pdf
    suo-kif.pdf
  inference/
    kif.exe
    kif-linux
    SUMO-v.kif
  InstallSigma.bat
  InstallSigma.sh
  KBs/
    config.xml
    language.txt
    Merge.kif
    Mid-level-ontology.kif
    english_format.kif
    MILO-format.kif
    WordNetMappings-adj.txt
    WordNetMappings-adv.txt
    WordNetMappings-noun.txt
    WordNetMappings-verb.txt
```



```

index.sense
noun.exc
verb.exc
wordFrequencies.txt
stopwords.txt
tests/
    TQG1.kif.tq
    ...
    TQG50.kif.tq
    test_assertions.kif
    tq_notes.txt
manifest.txt
README.txt
sigma.war

```

4 Quick Start

This section is for users who wish to install and run Sigma before reading a more comprehensive overview of Sigma's components and functionality. Please understand that the quick install procedure described here cannot accommodate all possibly relevant variations in the computing environments of different users. If the steps described here fail for you, please try the more detailed installation instructions described in the section named Installation. If you wish to begin the quick start procedure, follow these steps:

1. Halt the Tomcat server if it is running. In Windows, you can halt Tomcat by using the Configure Tomcat utility application, as shown in Figure 6. In Linux or other UNIX-like environments, you can halt Tomcat by invoking the shell script `shutdown.sh`, which is located in Tomcat's `bin` directory.
2. Download and unpack the Sigma install package, which is available in the Sigma project area at SourceForge: <http://sourceforge.net/projects/sigma/>.

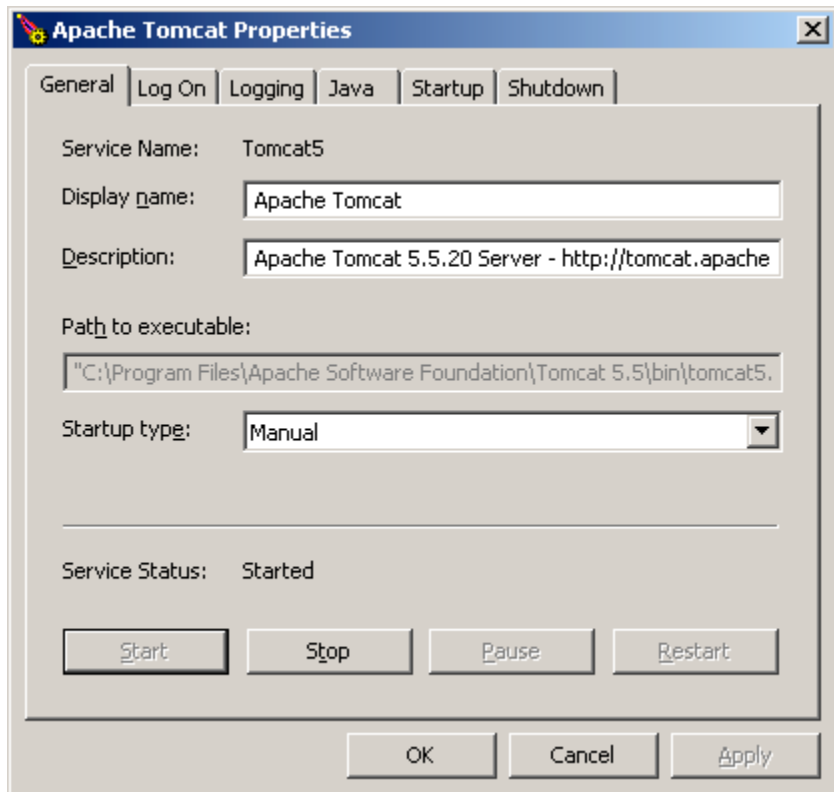


Figure 6: The Configure Tomcat utility, showing the “Stop” button

```
C:\WINDOWS\system32\cmd.exe

Installing Sigma files ...

Removed C:\Program Files\Apache Software Foundation\Tomcat 5.5\work\Catalina\localhost\sigma.
Removed C:\Program Files\Apache Software Foundation\Tomcat 5.5\webapps\sigma.
1 file(s) copied.
Wrote sigma.war.old.
1 file(s) copied.
Wrote sigma.war.
1 file(s) copied.
Wrote english_format.kif.
1 file(s) copied.
Wrote Merge.kif.
1 file(s) copied.
Wrote Mid-level-ontology.kif.
1 file(s) copied.
Wrote MIL0-format.kif.
1 file(s) copied.
Wrote language.txt.
1 file(s) copied.
Wrote stopwords.txt.
1 file(s) copied.
Wrote wordFrequencies.txt.
1 file(s) copied.
Wrote wordFrequencies_combined.txt.
1 file(s) copied.
Wrote WordNetMappings-adj.txt.
1 file(s) copied.
Wrote WordNetMappings-adv.txt.
1 file(s) copied.
Wrote WordNetMappings-noun.txt.
1 file(s) copied.
Wrote WordNetMappings-verb.txt.
1 file(s) copied.
Wrote noun.exc.
1 file(s) copied.
Wrote verb.exc.
1 file(s) copied.
Wrote index.sense.
1 file(s) copied.
Wrote test_assertions.kif.
1 file(s) copied.
Wrote TQG1.kif.tq.
1 file(s) copied.
Wrote TQG10.kif.tq.
1 file(s) copied.
Wrote TQG11.kif.tq.
1 file(s) copied.
. . .
Wrote TQG6.kif.tq.
1 file(s) copied.
Wrote TQG7.kif.tq.
1 file(s) copied.
Wrote TQG8.kif.tq.
1 file(s) copied.
Wrote TQG9.kif.tq.
1 file(s) copied.
Wrote tq_notes.txt.
Wrote config.xml.

Sigma files installed!
Press any key to continue . . .
```

Figure 7: Output displayed when the script InstallSigma.bat is run

3. Open the directory named `sigma-2-01`. It should contain several files and subdirectories, approximately like those shown in Figure 4. Please review the description of the release package's directories and files provided in the Release Package Contents section, above.

4. Run the version of the script `InstallSigma` that is intended for your operating system. In Windows, you will run the file named `InstallSigma.bat`, either by invoking it from a command prompt window or by double-clicking on the file icon with your mouse arrow. The file's icon will have a small gear in the center, as shown in Figure 4. In Linux or Mac OS X, you will run the shell script named `InstallSigma.sh` by invoking it from the shell prompt in a terminal window. `InstallSigma`'s main purpose is to copy files from the directory `sigma-2-01` to appropriate directories in your local file system. If `InstallSigma` encounters a problem that prevents it from copying a file or completing some other action, it will print a message noting the failure and then will quit, giving you a chance to diagnose and fix the problem. If `InstallSigma` successfully finishes its programmed tasks, it will print the message "Sigma files installed!" (Figure 7).
5. After you have successfully run `InstallSigma`, restart Tomcat and then direct your web browser to this URL: `http://localhost:8080/sigma/KBs.jsp` . You will experience a

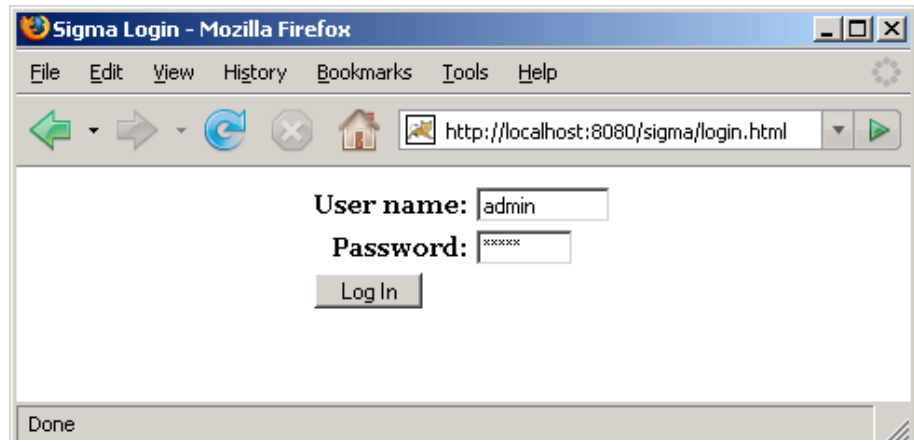


Figure 8: The Sigma Login page

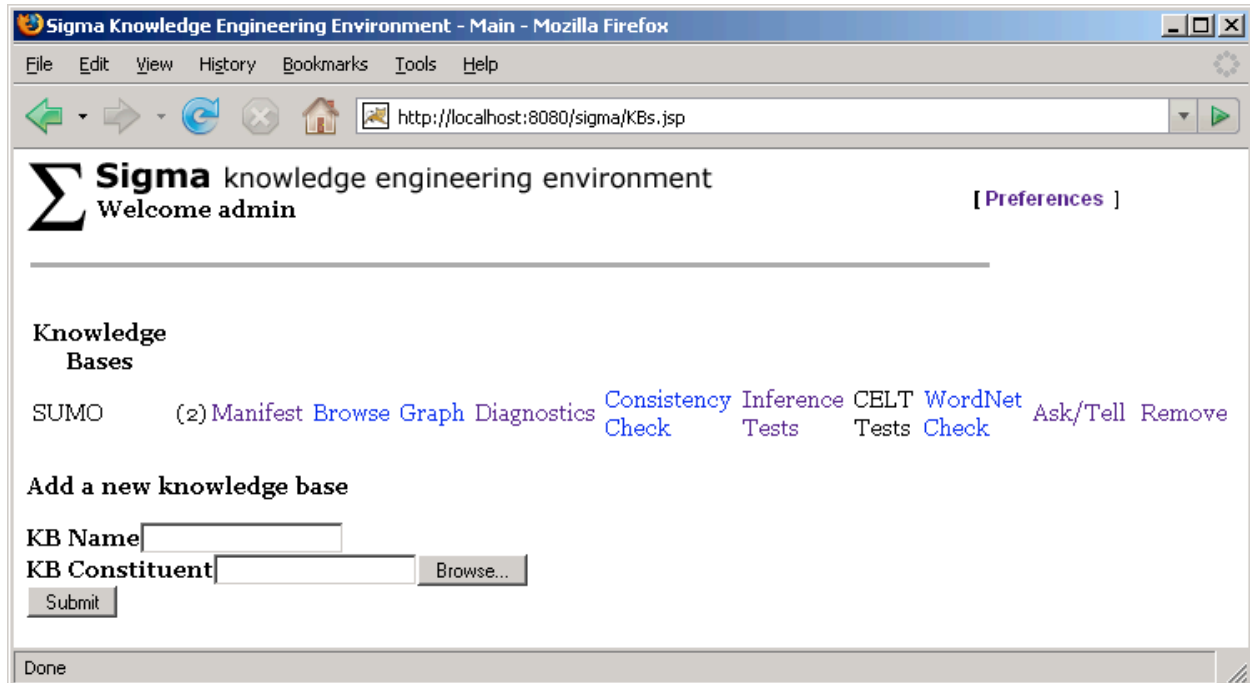


Figure 9: The Sigma Knowledge Bases page

brief delay as Tomcat unpacks and processes Sigma's web archive, but after a few seconds the Sigma login page should appear (Figure 8).

6. You should be able to log in by entering the user name `admin` and the password `admin`, and then clicking on the button labeled "Log In".
7. After you have logged in, Sigma will load constituent files to build a KB, and also will translate the resulting assertions into a format suitable for use by Sigma's local Vampire theorem prover. The translation process might last several seconds or a few minutes, depending on the speed of your computer's processor and the amount of memory available. As soon as the translation process has finished, the Knowledge Bases page will be displayed (Figure 9).
8. See the User Guide section of this document for instructions on how to create, modify, and work with knowledge bases. Note that you might have to change some of Sigma's default configuration settings to enable complete functionality, since some of the values cannot easily be set during the installation procedure, and others are based only on educated guesses. Follow the `[Preferences]` link to edit your configuration settings, and be sure to click on the button labeled "Save" in order to save your changes.

5 Installation

This section explains how to install Sigma if the Quick Start procedure described above fails, or you would like to have more control over the various steps of the installation process. For the most part, the instructions below correspond to the programmed actions of the installation script, `InstallSigma`.

1. Follow the first three steps of the Quick Start procedure: halt Tomcat, download and unpack the Sigma install package, and open (or `cd` into) the newly created directory `sigma-2-01`.
2. Sigma requires a "working" directory named `KBs`, and will create such a directory at startup if it cannot find one. Sigma consults the `KBs` directory to find its configuration files (`config.xml`, `language.txt`), the SUO-KIF files needed to build a knowledge base, and the data files that map between concepts in the KB and WordNet synsets. If the environment variable `SIGMA_HOME` has been set to a valid directory pathname, Sigma will expect to find (or will create) `KBs` as a subdirectory of the directory designated by `SIGMA_HOME`. If `SIGMA_HOME` has not been set but `CATALINA_HOME` has, then Sigma will expect to find (or will create) `KBs` under the directory designated by `CATALINA_HOME`. If neither `SIGMA_HOME` nor `CATALINA_HOME` has been set, then Sigma will expect to find (or will create) `KBs` in Tomcat's "current" directory, which, depending on how Tomcat was started, is likely to be either Tomcat's `bin` directory or the directory immediately above `bin`. Finally, if Sigma finds a `KBs` directory, it will load the file `config.xml` and will use the pathname parameters defined in that file to determine where its working directory is located, even if the parameters are wrong. Therefore, it is important to set `SIGMA_HOME` and `CATALINA_HOME` as described in the Prerequisites section of this document, and, if you are not using the script `InstallSigma` for installation, to edit the file `config.xml` so that its pathname parameters are consistent with the values of `SIGMA_HOME` and `CATALINA_HOME`. We recommend either of the following two options:

- Copy the entire `KBs` directory, including all of its contents, from `sigma-2-01` to a new location, and allow Sigma to use the new `KBs` directory as its working directory.
 - Simply allow Sigma to use the `KBs` directory in `sigma-2-01` as its working directory.
3. Whichever option you choose, be sure to set `SIGMA_HOME` to the pathname of the directory that contains the `KBs` directory, so that Sigma will be able to find the intended working directory when it starts up.
 4. Edit the file `config.xml` so that the relevant pathname parameter values are consistent with the values of `SIGMA_HOME` and `CATALINA_HOME`. The parameters to edit are as follows:
 - `"baseDir"` should be set to the same value as `SIGMA_HOME`.
 - `"kbDir"` should be set to the absolute pathname of Sigma's `KBs` directory.
 - `"inferenceTestDir"` should be set to the absolute pathname of the `tests` directory, which contains the files that define Sigma's inference tests. In the Sigma distribution package, the `tests` directory is located directly under the `KBs` directory, but it can be moved to a different location, if desired.
 - `"testOutputDir"` should be set to a directory under Tomcat's `webapps` directory in order for the HTML inference test result pages to be properly accessible. For example, if `[Tomcat]` is understood to denote Tomcat's root directory and the operating system is Windows, then the value of `"testOutputDir"` should be set to something like `[Tomcat]\\webapps\\sigma\\tests`.
 - `"inferenceEngine"` should be set to the absolute pathname of the Vampire executable file located in the directory `sigma-2-01/inference`. In Windows, the executable file named `kif.exe` should be used. In Linux, the executable file named `kif-linux` should be used. At the time of this release, there is no Vampire executable file available for Mac OS X.
 - Finally, the pathnames of any SUO-KIF constituent files included in `config.xml` (e.g., `Merge.kif` and `english_format.kif`) should begin with the value to which `"kbDir"` is set, since Sigma's constituent files will be stored in the `KBs` directory.
 5. In the `webapps` directory under the Tomcat root directory, delete the file `sigma.war` and also the directory `sigma` and its contents, if they exist.
 6. Delete any other files or directories under the Tomcat root directory that might be left over from previous Sigma sessions. For example, if `[Tomcat]` denotes the Tomcat root directory and the operating system is Windows, then you should delete the directory `[Tomcat]\\work\\Catalina\\localhost\\sigma`, if it exists.
 7. Copy the Sigma web application archive file, `sigma.war`, from the directory `sigma-2-01` to the `webapps` directory under Tomcat's root directory.
 8. Start Tomcat, and then direct your browser to `http://localhost:8080/sigma/KBs.jsp`. You will experience a brief delay as Tomcat unpacks and processes Sigma's web archive file, `sigma.war`, but after a few seconds the Sigma login page should appear (Figure 8).
 9. You should be able to log in by entering the user name `admin` and the password `admin`, and then clicking on the button labeled "Log In".

10. After you have logged in, Sigma will load constituent files to build a KB, and also will translate the resulting assertions into a format suitable for use by Sigma's local Vampire theorem prover. The translation process might last several seconds or a few minutes, depending on the speed of your computer's processor and the amount of memory available. As soon as the translation process has finished, the Knowledge Bases page will be displayed (Figure 9).
11. From the Knowledge Base page you can select SUO-KIF KB constituent files for loading in order to build or augment a KB, and can navigate to other pages that provide access to Sigma's various capabilities. Instructions for creating a knowledge base are provided in the User Guide section, below.
12. You might have to change some of Sigma's default configuration settings to enable complete functionality, since some of the values cannot easily be set during the installation procedure, and others are based only on educated guesses. Follow the [Preferences] link to edit your configuration settings, and be sure to click on the button labeled "Save" in order to save your changes.
13. Please note that if you intend to add constituent files to the default SUMO KB that is included in the install package, or if you intend to create a new KB from scratch, you should not copy SUO-KIF files into Sigma's working directory, KBs, and then direct Sigma, through the Knowledge Base or Manifest web pages, to load constituent files from that location. Instead, keep KB constituent files that you wish to load into Sigma via the web interface in some directory other than Sigma's working directory. When Sigma is directed to load a constituent file via the web interface, it copies the constituent file from the file's directory of origin to Sigma's working directory. If Sigma is told to load the file from its working directory and then tries to write the same file into its working directory, the resulting file might be corrupted, and the load operation might fail.

6 Troubleshooting

This section describes problems that some users might encounter when trying to install or use Sigma.

Limited size of constituent files: As currently configured, Sigma cannot load SUO-KIF constituent files larger than 2 MB in order to build a KB.

Tomcat port conflicts: Preexisting software may use some of the ports that Tomcat requires. In particular, some users have found a conflict with Tomcat's default "SHUTDOWN" port, 8005. If Tomcat does not start properly, it may be necessary to edit the file `server.xml`, in the directory `conf` under Tomcat's root directory, and change this to a different port, such as 8015.

Directory and file permissions: In Linux and other UNIX-like environments, users might encounter problems when trying to run `InstallSigma.sh`, trying to copy directories and files from `sigma-2-01` to other locations in the file system, or trying to delete old Sigma directories and files. Such problems occur most often because Tomcat was originally installed by a different user with administrative or "root" privileges, and the current user does not have permission to change directories or files under the Tomcat root directory. To address this problem, you might have to ask your system administrator or IT support person for help.

7 User Guide

In this section, we first run through a typical session illustrating Sigma's major capabilities, and then provide a reference to the remaining capabilities. Note that Sigma does not provide tools for editing knowledge bases (KBs). KBs in first order logic are similar to modern programming languages in complexity and expressiveness, and the most suitable tool for editing is a powerful text editor, such as open source versions of Emacs, or a commercial programming language editor such as Visual SlickEdit. The color-coding and formatting tools offered in such editors can be very helpful. Sigma serves the same purpose as a modern Integrated Development Environment (IDE), supporting structured examination, project management, and debugging.

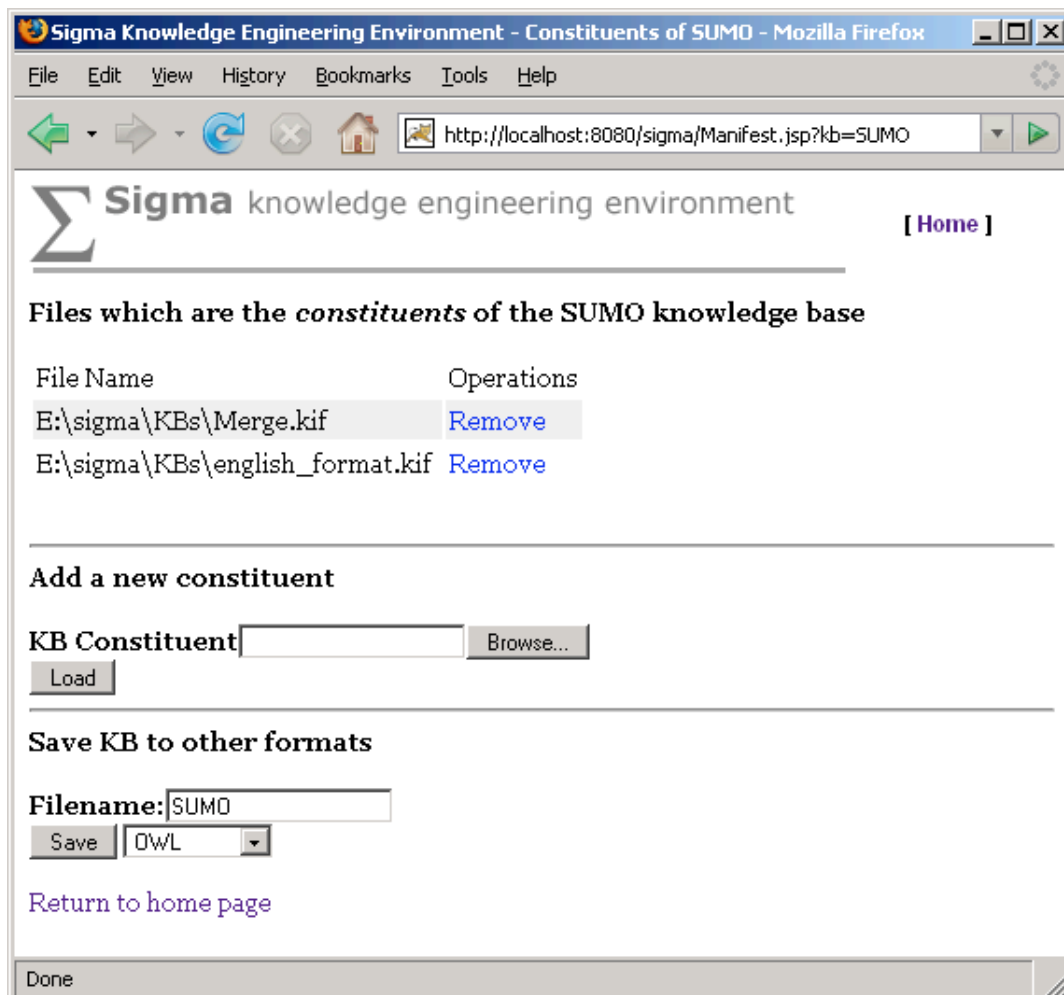


Figure 10: The Manifest page

The first page one sees when starting Sigma is the login page (Figure 8). In its present version, Sigma has only the most rudimentary login functionality, with one hard-wired password that allows access to all Sigma functions. If that password is not chosen, only read-only operations are allowed. If you do not have the administrative password to Sigma and wish to perform operations that are not read-only, log in with the user name `admin` and the password `admin`.

The next page displays a listing of all the composite KBs loaded, the operations allowed on those KBs, and a type-in area for creating a new composite KB. The standard Sigma release

package includes the Suggested Upper Merged Ontology (SUMO) (Niles & Pease, 2001), so you should see a page similar to that shown in Figure 9 if you have installed Sigma with the installation script, `InstallSigma`. If you are not using the standard release package, you probably will see a message indicating that no KBs are loaded. You will have to log in to Sigma with the administrator password and create a new knowledge base.

Sigma organizes ontologies in knowledge bases that are sets of assertions loaded from files selected by the user. Each knowledge base is created from one or more constituent files, which contain statements written in a simplified form of Knowledge Interchange Format (KIF) (Genesereth, 1991), a formal, logic-based representation language. Sigma’s dialect of KIF is called SUO-KIF (Pease, 2004). Sigma’s Manifest page shows the files contained in a knowledge base (Figure 10).

The fundamental interface component of Sigma is a statement browser that displays the SUO-KIF statements in which a given term appears. Clicking the link labeled “Browse” in the Knowledge Bases page will create a browser page for the selected KB. The initial browser page

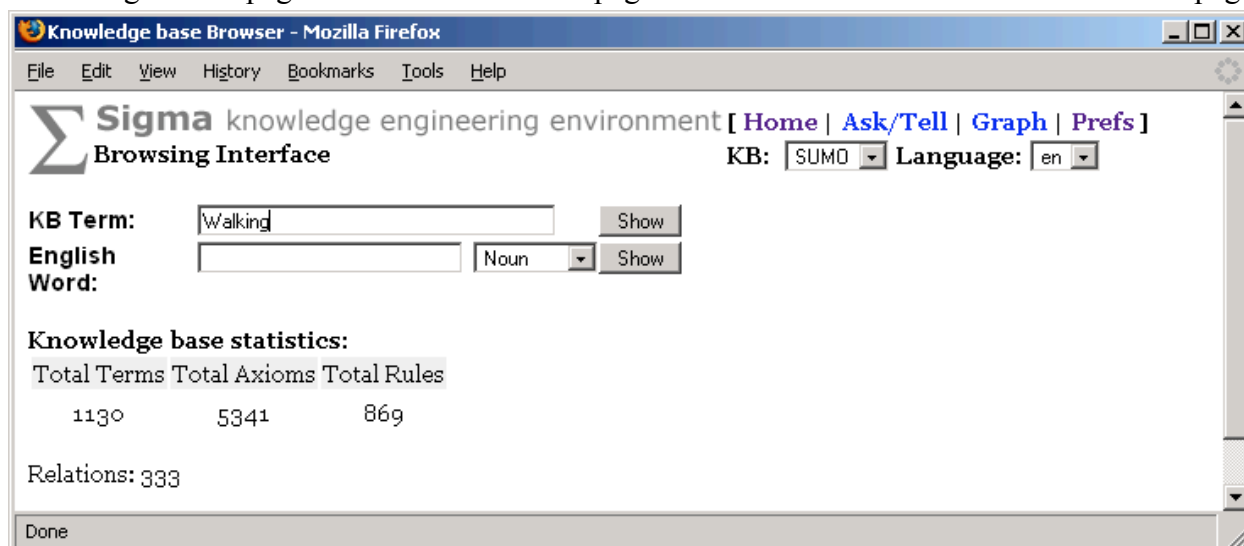



Figure 11: The Browsing Interface page in which the term “Walking” has been entered

just lists some statistics for the KB, and provides a type-in area for entering terms to be displayed. Figure 11 shows a browser page in which the user has typed the term Walking. Figure 12 shows a browser page listing all of the statements in the KB that contain the term Walking.

A mouse click on a hyperlinked term in a statement displays the browser page for that statement. Clicking on the term Running, for example, causes the creation of a page that shows all of the statements in which Running appears. In the blue-grey center column, the page shows the name of the SUO-KIF file from which each statement was loaded, and the line number at which it appears in its file. An English paraphrase of the statement is shown in the right-hand column. The paraphrases are generated automatically from statements composed with the SUO-KIF predicate `format`. On the Manifest page shown in Figure 10, we can see that the file `english_format.kif` has been loaded. `english_format.kif` contains `format` statements that serve as templates for paraphrasing SUO-KIF statements as English.

Knowledge base Browser - Mozilla Firefox

File Edit View History Bookmarks Tools Help


Sigma knowledge engineering environment
Browsing Interface

[\[Home \]](#)
[\[Ask/Tell \]](#)
[\[Graph \]](#)
[\[Prefs \]](#)

KB: Language:

KB Term: Show

English Word: Noun Show

Walking^(walking)

Rollerblade, afoot, amble, ambulate, ambulation, angry_walk, backpack, break, bumble, canter, careen, circumambulate, clamber, climb, climb_up, clomp, clump, cock, coggle, constitutional, constitutionalize, countermarch, crab, creep, curvet, dash, debouch, dodder, dogtrot, drag, dressage, drift, err, escalate, exhibit, falter, fast_break, file, file_in, file_out, fire_walking, flounce, flounder, foot, footer, footslog, footstep, forage, gait, gallop...

appearance as argument number 1

(documentation Walking "Ambulating relatively slowly, i.e. moving in such a way that at least one foot is always in contact with the ground.")

Merge.kif
8277-8278

(subclass Walking Ambulating)

Merge.kif
8276-8276

walking is a subclass of Ambulating

appearance as argument number 2

(partition Ambulating Walking Running)

Merge.kif
8271-8271

Ambulating is exhaustively partitioned into walking and Running

(termFormat en Walking "walking")

english_format.kif
793-793

termFormat en walking "walking"

antecedent

(=>
(
and
(instance ?WALK Walking)
(instance ?RUN Running)
(agent ?WALK ?AGENT)
(agent ?RUN ?AGENT)
(holdsDuring
(WhenFn ?WALK)
(measure ?AGENT
(SpeedFn ?LENGTH1 ?TIME)))
(holdsDuring
(WhenFn ?RUN)
(measure ?AGENT
(SpeedFn ?LENGTH2 ?TIME)))
(greaterThan ?LENGTH2 ?LENGTH1))
)
)

Merge.kif
8285-8293

- if ?WALK is an instance of walking and ?RUN is an instance of Running and ?AGENT is an agent of ?WALK and ?AGENT is an agent of ?RUN and the measure of ?AGENT is ?LENGTH1 per ?TIME holds during the time of existence of ?WALK and the measure of ?AGENT is ?LENGTH2 per ?TIME holds during the time of existence of ?RUN,
- then ?LENGTH2 is greater than ?LENGTH1

Done

Figure 12: The term "Walking" displayed in the Browser Interface page

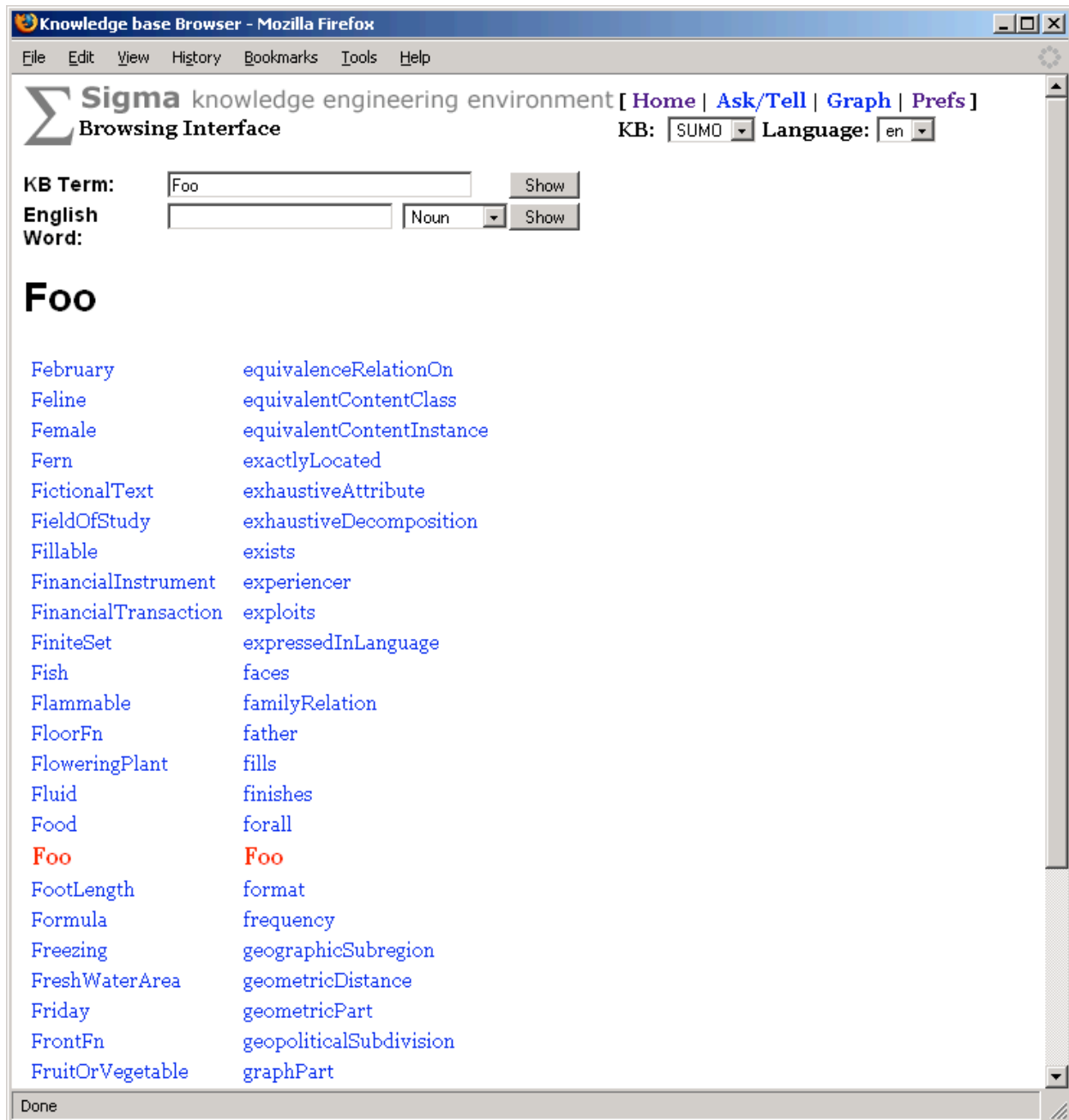


Figure 13: The Browsing Interface displaying the terms closest, alphabetically, to an unknown term

At the top right of the term browser page is a pull-down menu labeled “Language”. The menu is constructed automatically, based on the presence of language formatting statements in the knowledge base. Files containing `format` statements for the relations defined in SUMO have been created for French, Hindi, and a few other languages, and are available for download from Sigma’s CVS repository at this URL:

<http://sigmakee.cvs.sourceforge.net/sigmakee/KBs/Translations/> .

Note that the first time the user displays a term in the browser page, there will be a delay as the entire WordNet lexicon and SUMO mappings are loaded, and the `format` statements used for paraphrasing are processed.

If the user enters into the type-in area labeled “KB Term” a term that cannot be found, Sigma responds by displaying the set of terms that are closest, alphabetically, to the entered term, as shown in Figure 13. If the user enters a word into the type-in area labeled “English Word”, Sigma displays a list of all the matching English word senses in the WordNet lexicon and their mappings to terms in SUMO (Niles & Pease, 2003). The result of entering the word “buffalo”, for example, is the page shown in Figure 14.

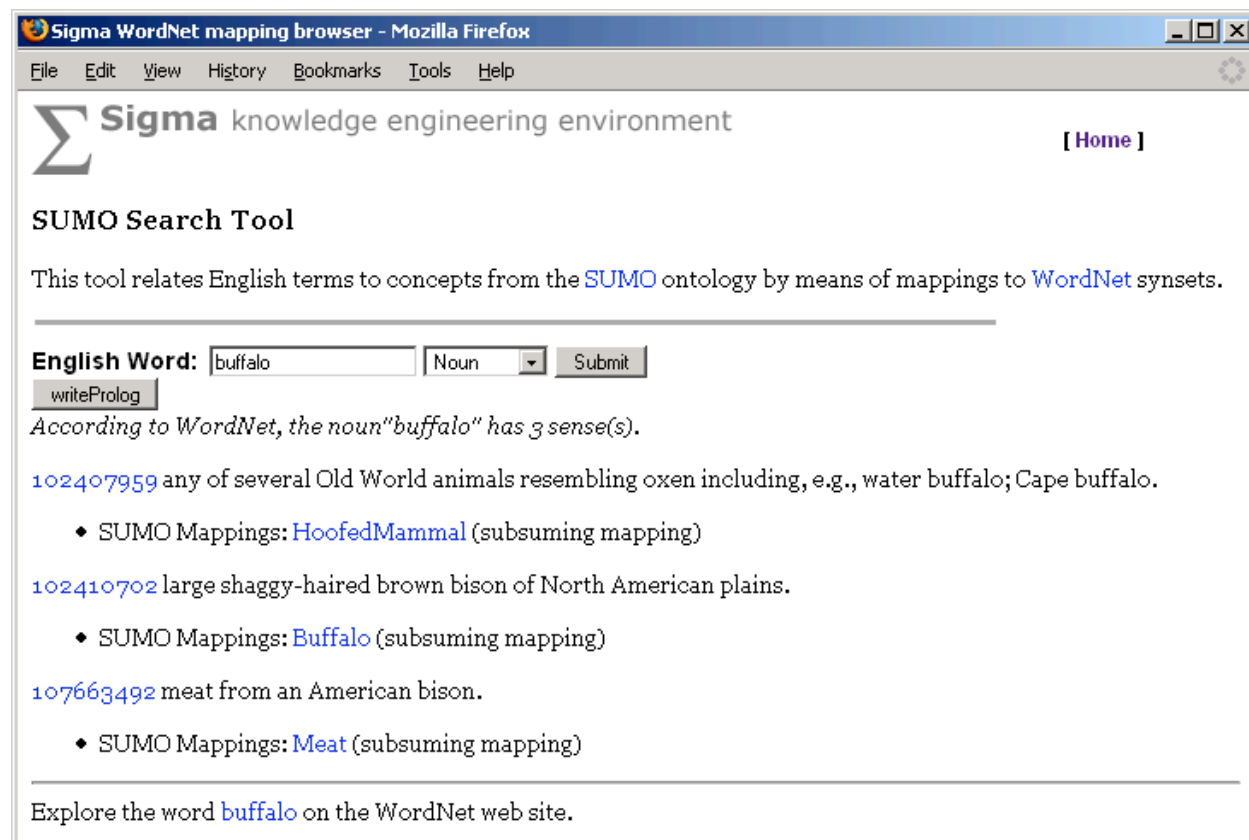


Figure 14: Sigma’s display of WordNet synsets for “buffalo”, and corresponding SUMO concepts

Another important capability in Sigma is logical inference over KB contents. Sigma is integrated with the Vampire theorem prover (Riazanov & Voronkov, 2002). To explore this functionality, go to the Knowledge Bases page and click on the link labeled “Ask/Tell”. To illustrate a trivial inference, we can ask for a subclass of the class Predicate by posing this query: `(subclass ?X Predicate)`. The theorem prover returns a very simple proof that the class TernaryPredicate is a subclass of Predicate, as shown in Figure 15.

Full discussion of the proofs resulting from a resolution theorem prover is beyond the scope of this manual, but we can point out a few items from the proof. Each step in the proof is numbered, and each step also has a justification for how it was derived. The justification can be a list of numbers. The value in the justification column for step 4 shows that it was derived from step 3. Steps can also be taken directly from the knowledge base, which is denoted by `[KB]`, or

from the query itself. Currently, no further justification about the inference rule applied is provided, although in this case, one can see that step 2 is derived from step 1 simply by renaming the variable. The proof method employed is proof by contradiction, where the query is negated, and the system tries to find a contradiction that results. So, we see the label [Negated Query] as the justification for step 1.

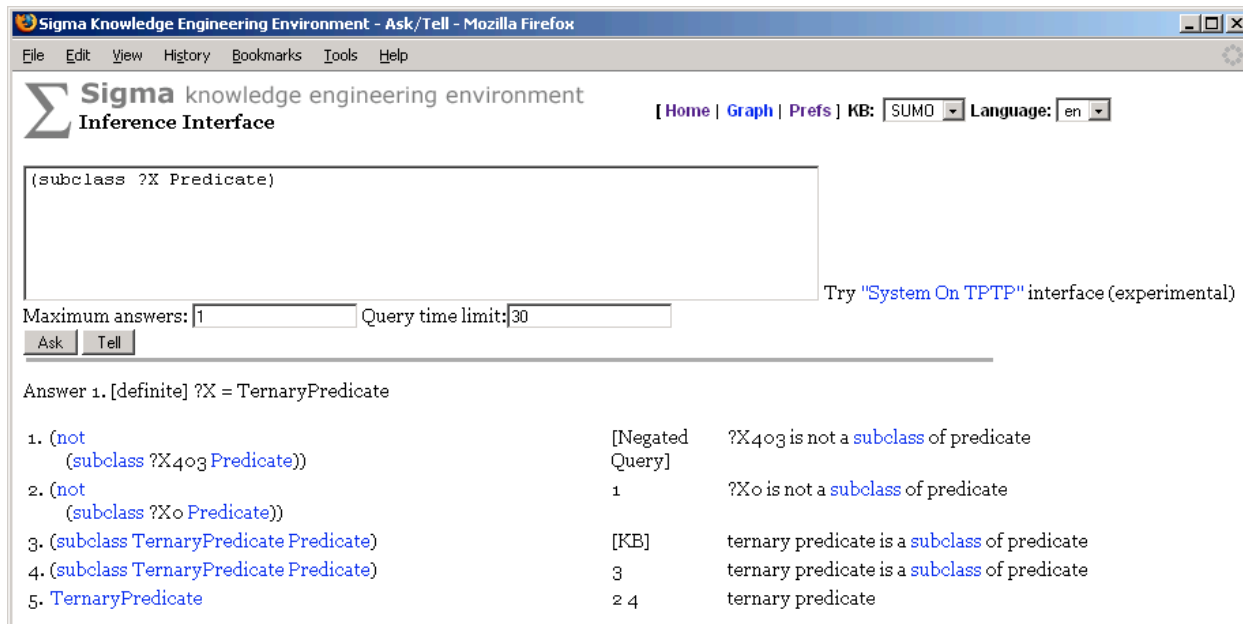


Figure 15: The Inference Interface page, displaying a query and a proof

The inference engine can be controlled by limiting the number of answers it is directed to find and by providing a time cutoff. In a large, interconnected knowledge base there are so many possible search paths that the inference engine would frequently continue to search indefinitely if such cutoffs were not provided.

One can also assert an individual formula to the knowledge base by entering a SUO-KIF statement and then clicking the button labeled “Tell”. The asserted formula then becomes accessible for inference and browsing. If the user has not previously done a “Tell” to the current KB, the system creates a new file named <KB-name>_UserAssertions.kif (e.g., SUMO_UserAssertions.kif), and adds the formula, as well as any subsequently entered formulas, to that file. The file will be loaded automatically when Sigma is restarted, and it can also be deleted from the manifest like any constituent, if desired.

Note that first order inference is computationally expensive and expected results may not be achieved, even if they logically follow from the knowledge base. Also, Vampire has no notion of what it means to return a commonsense answer, just a logically correct one, so general axioms in SUMO can occasionally give rise to answers that, although logically true, may not be useful or expected. A final caution on inference: although SUMO assumes a sorted logic, Vampire is currently unsorted, and therefore axioms occasionally can be employed in ways that are inconsistent with the argument types defined for SUMO relations, resulting in spurious answers. Our current remedy is to dynamically translate the KB into a special format that adds explicit “sortal” constraints to every rule (conditional or bi-conditional expression) before the KB is loaded into Vampire. This translation is performed only if the option labeled “Add “sortal”

antecedent to every axiom”, on the Preferences page, is set to “Yes” (Figure 16). The use of “sortals” with Sigma’s version of Vampire is still experimental, but the results are promising enough that “Yes” is now the default setting for this option. Note that if “sortal” constraints are used, statement caching (described in the next paragraph) should be used, too.

Sigma knowledge engineering environment [Home]

Fully qualified path and name of the inference engine

Directory in which the CELT system is located

Fully qualified path and name of SWI prolog

DNS address of the computer on which Sigma is hosted

Port number on which Tomcat responds

Name of the SUMO KB in Sigma

Directory in which tests for the inference engine are found

Command to invoke text editor

Command line option for text editor to set cursor at a particular line

☒ yes ☐ no : Show cached statements in the term browser

☐ yes ☒ no : Load CELT at startup

Formula Translation Options

- ☒ yes ☐ no : Add a "sortal" antecedent to every axiom
- ☐ yes ☒ no : Prefix all clauses with "holds" (otherwise instantiate all variables in predicate position)
- ☒ yes ☐ no : Employ statement caching
- ☒ yes ☐ no : Perform TPTP translation

Some options require a restart of Tomcat and Sigma. Changing a translation option will force the KB to be reloaded.

Figure 16: The Preferences page, showing the recommended settings for the Formula Translation Options

Sigma has a simple statement caching mechanism for hierarchical and subsumption relations that improves inference in many cases, since reasoning about class subsumption and other hierarchical relations is often necessary. Sigma can compute the transitive closure of subclass statements and assert them directly, so that Vampire can find them as ground assertions rather than repeatedly having to apply SUMO’s subclass reasoning axiom during inference. For example, if we ask whether Human is a subclass of Object, without statement caching Vampire

would have to apply the same axioms several times to determine the correct answer. It could, in fact, spend all of its allotted time searching unhelpful paths, since it does not know what commonsense answer we are looking for. By asserting directly that Human is a subclass of Object during a “preprocessing” step performed when KB constituent files are loaded, we allow the theorem prover to avoid a great deal of work, and get better and faster results. Caching is turned on from the Preferences page by setting the option labeled “Employ statement caching” to “Yes” (Figure 16).

The Preferences page also allows the user to set the directory in which the Vampire executable is found, the IP address of the server running Sigma, the name of the KB, the pathname of the directory in which inference tests are stored, whether automatically cached statements should be displayed in the browser, and whether the Controlled English to Logic Translation (CELT) (Pease & Murray, 2003) system should be loaded at startup. The CELT component is experimental, and is not included in the Sigma distribution.

8 Reference Guide

The section briefly describes each of the web/JSP pages that together constitute Sigma’s user interface.

login.html

The login page allows the user to type in a user name and password. At present, an administrative user name and password – `admin` – grants full access to Sigma functions, and all other user name and password combinations result in the availability of strictly read-only functionality.

KBs.jsp

The generated page lists all the knowledge bases currently loaded into Sigma. Each knowledge base can be examined or manipulated through functions that correspond to the following hyperlinks:

Manifest – takes the user to a page generated by `Manifest.jsp`, which lists all of the files that make up the knowledge base.

Browse – takes the user to a page generated by `Browse.jsp`, where all the statements for a given term are displayed, and where the user may search for the alphabetical neighbors of a term, as well as for English words and their links to SUMO.

Graph – takes the user to a page generated by `Graph.jsp`, which displays a graph of the terms connected a given (typically, transitive) relation.

Diagnostics – causes a set of pre-programmed KB diagnostic tests to be run, and then takes the user to a page of test results generated by `Diag.jsp`.

Consistency Check – takes the user to a page generated by `CCheck.jsp`, which displays contradictions discovered by the theorem prover. Note that this function might require several hours to run to completion, so use it with caution. If no theorem prover is currently available, this link will be deactivated.

Inference Tests – causes the inference tests defined by the files contained in `KBs/tests` to be run, and then displays a page of test results generated by

`InferenceTestSuite.jsp`. Note that it could take hours for the inference tests to run to completion. If no theorem prover is currently available, this link will be deactivated.

CELT Tests – causes CELT tests to be run, and then displays a page of test results generated by `InferenceTestSuite.jsp`. If the experimental CELT system is not loaded, this link will be deactivated.

WordNet Check – runs routines that check for correspondences between WordNet synsets and concepts in the KB, and then displays a page generated by `WNDiag.jsp` that shows missing correspondences.

Ask/Tell – takes the user to a page generated by `AskTell.jsp`, where one can make assertions to the KB or pose questions to the theorem prover. If the experimental CELT system is loaded, the user can enter assertions and queries in English. Otherwise, assertions and queries must be entered as SUO-KIF expressions.

Remove – Causes the corresponding KB to be deleted, and then displays a page generated by `KBs.jsp` that allows the user to start building a new KB (by loading constituent files).

WordNet.jsp

This page shows the WordNet synsets for the selected word and part of speech. Each synset shows the WordNet definition, and the SUMO term or terms that are linked to that synset. Clicking on a hyperlinked SUMO term takes the user to the term browser page for that term. The user can also type in another word and select a different part of speech, if desired. Sigma performs some simple processing of words that allows a word to be found even if the user types a plural or past tense, since WordNet stores only the root grammatical forms of words.

Browse.jsp

The Browsing Interface page is where the user concerned with building ontologies is likely to spend the most time. When no term is selected, three metrics are displayed. The “Total Terms” value is the number of constant names defined in the knowledge base. The constants may denote classes or instances. Note that variables do not count. The “Total Axioms” value is the number of ground or fully quantified statements in the knowledge base. This number is an approximation, since the statements are composed by users and are tallied according to their surface SUO-KIF appearance rather than their canonical clausal form. For example, one user might enter $(\Rightarrow A \text{ (and } B \text{ } C))$, which counts as one statement, while another user might enter $(\Rightarrow A \text{ } B)$ and $(\Rightarrow B \text{ } C)$, which count as two, even though the entries are logically equivalent. The “Total Rules” value is also somewhat of an approximation, because it is based on occurrences of the logical operators \Rightarrow and \Leftarrow . Since $(\Rightarrow A \text{ } B)$ is logically equivalent to $(\text{or } (\text{not } A) \text{ } B)$, two KBs could be logically equivalent despite having been constructed from different SUO-KIF expressions. Note that the number of rules is a subset of the number of axioms.

The Browsing Interface page has a number of controls. “KB Term” allows the user to search for a term in the knowledge base, and displays all statements that include the term. If the term is not found, the terms that are alphabetically closest to it are displayed. Terms occurring inside SUO-KIF expressions and their natural language paraphrases are hyperlinked, so that when the user clicks on a term, the Browsing Interface page for that term is displayed.

The “English Word” area functions as described in the WordNet.jsp page description in this section.

Links at the top of the Browsing Interface page take the user to the “home” Knowledge Bases page, the Ask/Tell page, the Graph page, and the Preferences page.

There are two menus at the top right of the page. One menu simply selects the knowledge base. This is an alternative to selecting “Browse” from a knowledge base on the Knowledge Bases page, since one can select the knowledge base of interest directly, without having to return to that page. The second menu is the “Language” menu.

Sigma is capable of loading SUO-KIF files that specify natural language formatting templates. These templates allow Sigma to paraphrase logical statements in a natural language. The formatting is quite simplistic, but can give some assistance to users who either are not comfortable in logic, or are not comfortable in the human language in which the KB’s constituent files been written. The syntax for the paraphrasing templates is described in the Appendix: Natural Language Format.

The user can load several natural language format files. The selection in the language menu tells Sigma in which language to format statements. The natural language paraphrases appear in the right hand column of the statement listing.

The center column of the Browser Interface page shows the name and line number of the file that contains the statement.

Manifest.jsp

The manifest page contains a listing of all the files that together compose the selected knowledge base. The full pathname of each constituent file is displayed. Next to each pathname is a link to remove the file. Below the list of pathnames are controls that allow to add new constituent files to the knowledge base.

Properties.jsp

This file generates the Preferences page, which allows the user to set and change a number of preference parameters, and is accessible via the [Preference] and [Prefs] links that appear on several other Sigma pages. Note that all of the parameters mentioned below, as well as others not mentioned here, can be set directly in the file `config.xml`, as described in the Installation section. The list of parameters the user can set includes the following:

- The user can set the directory in which the theorem prover executable is found.
- The user can set the IP address of the server on which Sigma is running. This allows hyperlinks to function properly when the user is working with a Sigma installation that is on a remote host, rather than on the user’s local machine.
- The user should set the name of the KB containing SUMO. This allows the hyperlinks on the WordNet page to function properly.
- The user can set the directory in which inference tests are found. The format for these tests is described in the section Appendix: Inference Test Format.
- The user can set whether the CELT (Pease & Murray, 2003) system should be loaded at startup. Starting this component can take several minutes, so generally it is best to turn this off when work in CELT is not being routinely performed. Also, since this component is experimental, it is not included in the general Sigma distribution.

- The user can set whether “sortal” constraints should be added to SUO-KIF axioms in the special translation of the KB generated for the theorem prover. For an example of the results, see the file `SUMO-v.kif`, which is included in the `inference` directory that is part of the Sigma download package. In general, this parameter should be set to “Yes”, and caching should also be enabled for best results during inference.
- The user can set whether caching should be performed, and whether automatically cached statements should be displayed in the browser. Caching results in storage of the transitive closure of all `subclass` statements, as well as statements computed for other transitive relations. Depending on the size of the KB, this can result in a huge number of new assertions that, although helpful for theorem prover performance, are not helpful when shown in the browser.
- The user can set whether predicates in the SUO-KIF translation prepared for the theorem prover should be rewritten as arguments to generalized “holds” predicates. The use of the “holds” predicate was introduced to Sigma in order to make it possible for rules that quantify over predicate names to be effectively used by first order theorem provers. An alternative, which appears to yield better results during inference, is to instantiate all relation variables with the names of the applicable relations. The default setting for this parameter is now “No”, and, in general, “holds” prefixing should be turned off for best inference performance.
- The user can set whether the SUO-KIF statements in the KB should be translated automatically to TPTP format, which is required for use of the theorem provers accessible via the experimental page generated by `SystemOnTPTP.jsp`.

Graph.jsp

This page allows the user to generate a hierarchical view of terms and relations in a knowledge base. Most typically, the user selects the `subclass` relation in order to see a portion of a class/subclass tree. The user selects a particular term and the number of “levels” above and below that term to be displayed. Only a text view is supported at this time. The user can select a different binary relation, such as `subrelation`, in order to see a hierarchy of predicates. This page also includes KB and language menus as described in the reference for the Browsing Interface page. Levels in the hierarchy are indicated by degree of indenting.

AskTell.jsp

The Inference Interface page supports asserting new statements to the knowledge base, and posing logical queries to the knowledge base. If the CELT system has been loaded, all non-SUO-KIF statements are assumed to be natural language statements, and are passed to CELT. CELT queries are terminated with a question mark, just like an English question. All other sentences are assumed to be statements. This page includes menus for the knowledge base, and the language in which natural language paraphrases are presented (see the description of the Browsing Interface page for more detail on these controls).

Pressing the button labeled “Tell” results in the statement being asserted to the knowledge base, and added to the file `<KB-name>_UserAssertions.kif`, where `<KB-name>` denotes the name of the current knowledge base. If the statement contains a syntax error, however, it is not asserted, and an error message will appear.

Pressing the button labeled “Ask” causes a query to be posed to the theorem prover. The theorem prover can be controlled by limiting the number of answers it is directed to find, as well as by providing a time cutoff. On a large, interconnected knowledge base there are so many possible search paths that the inference engine would frequently continue to search indefinitely if such cutoffs were not provided.

SystemOnTPTP.jsp

This experimental page is accessible from a link on the Inference Interface page (Figure 17). It allows the user to pose queries to any of several theorem provers accessible via a service

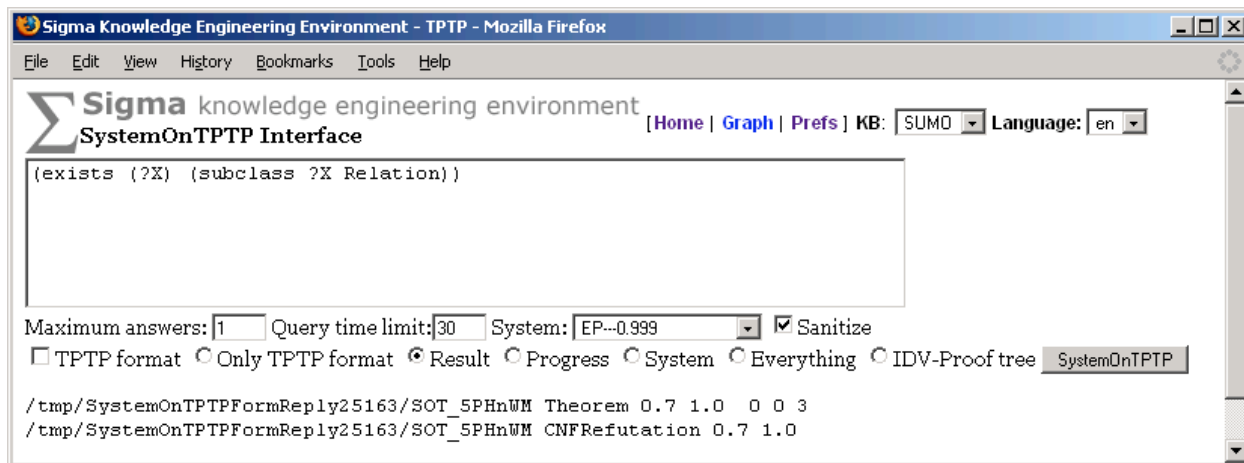


Figure 17: The SystemOnTPTP Interface page, showing the simple results of a query posed to the prover EP-0.999

provided by Geoff Sutcliffe and his students at the University of Miami. When a query is posed via the type-in area, both the query and the entire contents of the KB are translated to TPTP format and transmitted to the selected prover. The user can choose to see different parts of the query result data, and can view proofs in a graphical proof display interface developed by Steven Trac (Figure 18).

InferenceTestSuite.jsp

This code associated with this page will run inference tests, deposit the test results in the directory indicated in the Preferences page, and then automatically display an index to the test results when the tests have finished. Note that all tests will be completed before results are shown. This can take a long time if there are many tests. If no time limit is given in the definition of a particular test, a default of 60 seconds is used. It will be helpful to view the Tomcat log file (or the Tomcat message interface, if used) for messages showing how the tests are progressing. When the tests are complete, a list will be shown with links to each test source, its proof, if found, whether the test was successful as compared to the expected answer, and how long the test took. A total is provided for time, successes, and failures. The section Appendix: Test Formats described the format of inference test files.

This same body of code also runs CELT tests, if the user clicks on the “CELT Test” link from the Knowledge Bases page. CELT test results are shown in three columns with the English input first, then the expected logical form, the actual logical form, and whether the test failed or succeeded. The CELT test format is described in the section Appendix: Test Formats.

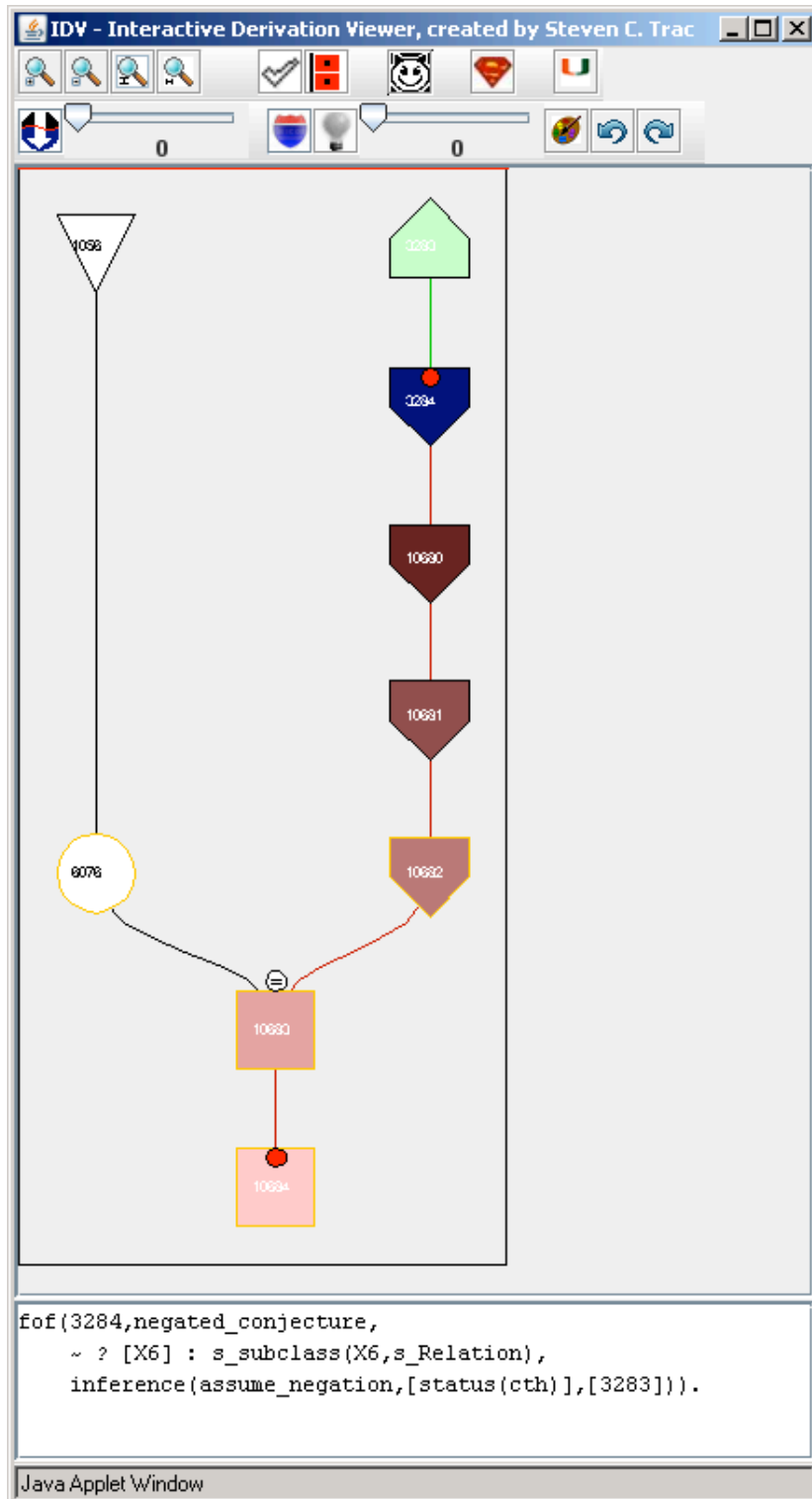


Figure 18: Steven Trac's graphical display tool for TPTP derivations

Diag.jsp

The Diagnostics page displays the results of three basic tests, which are run over the entire knowledge base when the "Diagnostics" link is clicked:

- Terms without documentation – tests whether each term occurs in a statement formed with the documentation predicate;
- Terms without a parent – tests whether each term occurs in a statement formed with the instance or subclass predicates; and
- Terms without a root at Entity – tests whether each term ultimately is an instance or subclass of Entity, which is the root term in SUMO.

This page might display the results of additional tests not mentioned above, and additional tests would be valuable. The current tests, while simple and requiring no inference (*i.e.*, no use of a theorem prover), do indicate some common problems entailed in building and maintaining large knowledge bases.

9 References

- Genesereth, M., (1991). “Knowledge Interchange Format”, in *Proceedings of the Second International Conference on the Principles of Knowledge Representation and Reasoning*, Allen, J., Fikes, R., Sandewall, E. (eds), Morgan Kaufman Publishers, pp. 238-249.
- Niles, I., & Pease, A., (2001). “Toward a Standard Upper Ontology”, in *Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS2001)*, Chris Welty and Barry Smith, eds.
- Niles, I., and Pease, A., (2003). “Linking Lexicons and Ontologies: Mapping WordNet to the Suggested Upper Merged Ontology”, in *Proceedings of the IEEE International Conference on Information and Knowledge Engineering. (IKE 2003)*, Las Vegas, Nevada, June 23-26, 2003.
- Pease, A., (2003). “The Sigma Ontology Development Environment”, in *Working Notes of the IJCAI-2003 Workshop on Ontology and Distributed Systems*, August 9, Acapulco, Mexico.
- Pease, A., (2004). *Standard Upper Ontology Knowledge Interchange Format*. Language manual, unpublished.
- Pease, A., and Murray, W., (2003). “An English to Logic Translator for Ontology-based Knowledge Representation Languages”, in *Proceedings of the 2003 IEEE International Conference on Natural Language Processing and Knowledge Engineering*, Beijing, China, pp. 777-783.
- Riazanov, A., & Voronkov, A., (2002). “The Design and Implementation of Vampire”, in *AI Communications*, Volume 15. Numbers 2-3.

10 Acknowledgments

Many thanks are due to Michal Sevcenko for defining the natural language format language, and to the Air Force and ARDA for their support for Sigma development.

11 Appendix: Natural Language Format

The predicate `format` associates a concept (either a relation or a function) with a string. `format` takes three arguments: the name or abbreviation of a natural language, the relation name, and the format string. When there is a need to visualize a concept in natural language, the associated string is used. The string contains a natural language description of the concept and special tags that are interpreted by the Sigma code to generate natural language paraphrases with proper HTML markup.

The meaning of these tags is as follows:

&%token – specifies a token that will be made into a hypertext link to the concept being visualized.

%1, %2, ... – this tag will be substituted with a natural language representation of the concept’s respective argument.

%n{text} – will be replaced either with an empty string, if a predicate is being rendered as positive, or “text” otherwise; the `%n` tag can be used as a shortcut for `%n{not}`.

%p{text} – will be replaced with “text” for positive rendering and with an empty string for negative rendering.

%*{range}[delim] – will be replaced with a list of natural-language representations of a subset of arguments; range specifies which arguments will be included; it is a comma separated list of numbers or ranges, for example, “1-4,6” denotes the first, second, third, fourth and sixth argument; the delim parameter specifies the delimiter which will be used to separate representations of arguments; both {range} and [delim] may be omitted, in which case {range} defaults to all arguments, and [delim] defaults to a single space.

%% – will be replaced with a single percent character.

The predicate `termFormat` relates a term to a natural language presentation of that term. It takes three arguments: the name or abbreviation of a natural language, the term name, and the format string.

12 Appendix: Test Formats

Inference test files are legal SUO-KIF files. The file name must end with the extension `.tq`. Inference tests support several special purpose predicates. `note` is a unary predicate that takes as its argument a term that will be the identifier for the test. `query` is a unary predicate that takes a SUO-KIF query as its only argument. `answer` is a unary predicate that takes as its argument either the symbols `yes` or `no`, or a list pair composed of a variable name (*e.g.*, `?X`) and an expected binding (value). `time` is a unary predicate whose argument specifies the number of seconds the system should wait for an answer.

There may also be a single CELT test file in the same test directory as the inference tests. It must be named `celtTest.txt`, and must contain legal SUO-KIF expressions. It should consist of pairs of statements. The first special unary predicate is `sentence`, which takes a string containing a CELT sentence as its argument. The second special unary predicate is `answer`, which takes a SUO-KIF formula as its argument. The output of CELT is compared against the `answer` formula to determine the success or failure of the test.