Scenario-based Ontology Evaluation (S Cone) 
User Guide

Fabian Neuhaus

April 5, 2015

Contents

1 Purpose of this document

This document describes S Cone, a tool for Scenario-based Ontology Evaluation. “S Cone” is the name for a methodology, namely an approach to ontology engineering that relies on requirements-based tests to drive the development process. “S Cone” is also the name of a language that was developed for writing down requirements of ontologies. Lastly, “S Cone” is a name for a tool that translates requirements for ontologies into behavioral tests and executes them. Thus, S Cone enables the evaluation of the behavior of ontologies.

2 Introduction

In the Ontology Summit 2013 Communique ontology evaluation is defined as follows:

Ontology evaluation consists of gathering information about some properties of an ontology, comparing the results with a set of requirements, and assessing the ontology’s suitability for some specified purpose [1].

Hence, according to this definition, requirements play a central role in the evaluation of ontologies. They are used to evaluate whether an ontology meets its expectations and whether it is able to play its intended role within a given information system. In short, requirements provide the measurement yard for the fitness of an ontology.

In order to evaluate the fitness of ontologies Grüninger and Fox have suggested to capture requirements as combination of scenarios and competency questions. Their approach can be summarized as follows:

- A requirement is captured within scenario, which is a narrative text in a natural language. The scenario describes a possible state of the world and raises a set of competency questions. The answers to these competency questions should follow logically from the scenario – provided the knowledge that is supposed to be represented by the ontology.

- Both the scenarios and the questions are formalized within the same knowledge representation language.

- A theorem prover is used to try to prove the answer to the formalized competency questions based on the ontology under test and the formalized scenario.

- If the theorem prover is able to answer the competency questions correctly, the ontology meets the requirement.

This methodology of scenario-based ontology evaluation has several major advantages.

Firstly, because the scenarios and the competency questions are written in a narrative text in a natural language, they can be written, reviewed, and maintained by the people who are not professional ontologists. This means that it is possible to specify the requirements together with a stakeholder in a language the stakeholder understands, and it allows to generate executable tests, which show whether these requirements are met.

Secondly, since, typically, the requirements are specified in many scenarios, and any scenario is linked to multiple competency questions, the ratio of competency questions that are answered correctly compared to all competency questions provides a good metric for progress during development. Often new ontology developers are not sure when to stop to add new axioms to an ontology. Using competency questions provides a natural answer: assuming that the requirements are specified properly, then an ontology that answers all competency questions correctly covers the domain appropriately to meet the requirements – thus, no additional information needs to be represented.

Thirdly, because the competency questions allow to track progress and can be evaluated automatically, it is easy to use them for regression testing. Hence, it is possible to automatically detect whether an changes to an ontology broke some existing feature and, thus, catch errors before a new version of an ontology is released.

Unfortunately, at this stage there is no convenient ways for ontology engineers to evaluate their ontologies with this methodology. One barrier is the formalization step from the natural language text to the formalized scenarios.

---

and formalized competency questions, which puts an extra burden on the ontology engineers. Another reason is that currently there are no tools that integrate the ontology evaluation step within the work flow of ontology development. For this reason, the Ontology Summit 2013 Communique recommends:

> Evaluation should be conducted against carefully identified requirements; these requirements depend on the intended use of the ontology and its operational environment. For this reason, we recommend the development of integrated ontology development and management environments that support the tracking of requirements for, and the evaluation of, ontologies across all phases of their development and use.

The purpose of Scone is provide an important building block for developing such an integrated ontology development and evaluation environment. Scone is a form of restricted English that allows to express requirements for ontologies. These requirements can be automatically translated into Distributed Ontology, Model, and Specification (DOL) libraries. The DOL libraries capture the content of the scenarios and the affiliated competency questions in a formal way. These DOL files can be executed by proof management systems such as Het5 which allows their use as behavioral tests. Since we plan to integrate Scone into the ontology repository Ontohub6, it will be convenient for ontology developers to integrate the use of these behavioral tests within their development workflow.

3 Overview and Example

The Scone language is an ontology evaluation specific extension of Gherkin7. Gherkin is the language of Cucumber8, which is a widely used tool that supports behavior-driven development (BDD) of software by executing plain-text functional descriptions as automated tests.

A typical example for a Scone file is presented in Figure ???. The file contains a description of the feature of the ontology that is tested. This section contains free text and is not utilized by the tool for testing purpose. The background section contains information that is shared across all scenarios. In this example, the language is set to OWL and the ontology geneology.owl is

---

4DOL is a response to the Object Management Group (OMG) Ontology, Model and Specification Integration and Interoperability Request for Proposal and is currently in the finalization phase. More information and the latest version of the draft standard is available at https://github.com/tillmo/DOL
5http://www.informatik.uni-bremen.de/agbkb/forschung/formal_methods/CoFI/bets/index_e.htm
6www.ontohub.org
7https://github.com/cucumber/cucumber/wiki/Gherkin
8https://github.com/cucumber/cucumber/wiki
Feature: Family relationships
In order to support my research the user should be able query my data for family relationships using "male", "female", "parent of", "grandparent of", "father of", "mother of", "older than"

Background:
* Language OWL
* Test the ontology <https://example.org/geneology.owl>

Scenario: Relative age between family members
The parenthood relation entail an ordering of age.
Given Chris is a parent of Dora.
And Amy is a parent of Chris.
And Amy is a parent of Berta.
Then infer Chris is older than Dora.
And infer Amy is older than Dora.
And don’t infer Berta is older than Dora.
And don’t infer Dora is older than Dora.

Scenario: Inferring various family relationships
In this scenario we test the definitions of "mother of", "grandparent", and "father".
Given John is a parent of Mary.
And Sue is a mother of John.
Then infer that Sue is a grandparent of Mary.
Given John is male.
Then infer that John is a father.
Given a mother is defined as a female, who is a parent of some thing.
Then infer that Sue is a mother.
And the scenario is consistent.

Scenario: Children aren’t older than their parents
Given Jeff is the father of Chris.
And Chris is older than Jeff.
Then the scenario is inconsistent.

Figure 1: SCONE specification example
loaded. The background is followed by three different scenarios. Each scenario has a title and may contain a descriptive text. Further, these scenarios may contain assumptions, which are marked by the "Given" keyword. In the first scenario the assumptions introduce Dora, her father Chris, her grandmother Amy, and her aunt Berta. (The keyword "And" is used instead of "Given" in order to improve the readability of the text, there is no functional difference.) The purpose of this scenario is to test the relationship between "parent of" and "older than". Thus, the scenario contains four competency questions about the age of Dora compared to the people in the scenario. These competency questions are introduced by the "Then" keyword (or, afterward, "And"). They are describing the intended behavior of the ontology given the scenario. E.g., one is supposed to be able to infer that Amy is older than her granddaughter, but we should not be able to logically infer that Berta is older than her niece.

The order of assumptions and competency questions within a scenario is not important. This makes it possible to state the competency questions after all relevant assumptions have been introduced. For example, in the second scenario the two first assumptions are sufficient for inferring that Sue is a grandparent of Mary. By adding a new assumption about John’s gender, we can add a new competency question that tests the definition of the father relationship. This example shows how one can build a scenario step-by-step by alternating between adding new assumptions and new competency questions.

Most of the assumptions are declarations of facts about individuals; e.g., “John is male” or “John is the father of Mary”. However, the second example in Fig. ?? contains a definition of “mother”. Definitions are only one kind of many types of complex assumptions that SCONE supports (see section ??). However, since SCONE is a controlled English it is less expressive than, for example, OWL. For the rare cases where its expressivity is not sufficient, SCONE allows to directly include axioms in the given ontology language.

There are four different kinds of competency questions support: (1) a piece of information should be inferable from the scenario, (2) a piece of information should not be inferable, (3) the scenario is consistent, and (4) the scenario is inconsistent.

4 Architecture of SCONE

Figure ?? provides an overview over the implementation of SCONE. The inputs for SCONE are one or more scone files and, typically, one ontology that is under test. (Theoretically, more than one ontology can be tested.) Each SCONE file tests a feature with the help of several scenarios. Since SCONE is an extension of Cucumber, Cucumber runs the testing process. While Gherkin provides the general structuring of a SCONE file into features, background, scenarios, it does not contain any keywords that are particularly relevant to ontology development. Thus, Cucumber hands over SCONE sentences to the SCONE parser, which analyses SCONE sentences and competency questions. The result is used to generated a DOL-file, which contains
Figure 2: System Architecture of SCONET
some embedded text in the chosen ontology language (e.g., OWL). After the DOL file is generated, Cucumber calls Hets in order to execute the proof obligations within the DOL-file. The result is reported back to Cucumber, which generates a report to the user.

5 Syntax of the Scone Language

5.1 Main Structure

\[ \langle scone-spec \rangle = \langle feature \rangle, \langle background \rangle, \{ \langle scenario \rangle \}; \]

\[ \langle feature \rangle = \{ \langle tag \rangle \}, \text{Feature:}, \langle title \rangle, \text{EOL, } \langle narrative \rangle; \]

\[ \langle title \rangle = \text{string} \]

\[ \langle narrative \rangle = \text{string} \]

\[ \langle background \rangle = \text{Background:}, \[ \langle language setting \rangle \], \{ \langle import \rangle \}; \]

\[ \langle tag \rangle = \text{a space delimited string starting with '@';} \]

\[ \langle language setting \rangle = \text{Language, } \langle language ID \rangle; \]

\[ \langle language ID \rangle = \%\text{OWL} | \%\text{CommonLogic} \]

\[ \langle import \rangle = \text{* Test, [the ontology], } \langle uriref \rangle; \]

5.2 Scenarios

Scenarios are built from a set of assumptions and a set of tests. Each assumption involves a sentence. A test is either a consistency test or an inference test. Any inference test involve a sentence.\[\]

\[ \langle scenario \rangle = \{ \langle tag \rangle \}, \langle scenario head \rangle, \langle scenario body \rangle; \]

\[ \langle scenario head \rangle = \text{Scenario:}, \langle title \rangle, \text{EOL, } \langle narrative \rangle, \[ \langle language setting \rangle \]; \]

\[ \langle scenario body \rangle = \{ \langle assumption \rangle \}, \{ \langle test \rangle \}; \]

\[ \langle assumption \rangle = \text{Given, } \langle sentence \rangle, \[.\]; \]

\[ \langle test \rangle = \text{Then, } \langle inference test \rangle | \langle consistency test \rangle, \[.\]; \]

\[ \langle inference test \rangle = \text{[don’t], } \text{infer, [that], } \langle infer sentence \rangle; \]

\[ \langle consistency test \rangle = \text{the scenario is, consistent | inconsistent;} \]

---

\[\]

9To enable a correct mapping to OWL particular sentences are not allowed to be part of inference tests, this is why we introduced \text{infer sentence} as a syntactical subcategory. This limitation can be omitted for CLIF or other languages that are expressive enough to enable the required form of existential quantification.
5.3 Sentences

\(<\text{sentence}> = <\text{definition}> | <\text{proposition}> | <\text{fact}> | <\text{source}>;</n\)

\(<\text{infer sentence}> = <\text{definition}> | <\text{universal}> | <\text{fact}> | <\text{source}>;</n\)

\(<\text{definition}> = ((\text{pos class}), \text{is defined as}, <\text{class expression}>)
         | ((\text{class name}), \text{is enumerated as}, <\text{indiv}>, \{\text{and}, <\text{indiv}>\});\n\)

\(<\text{proposition}> = <\text{universal}> | <\text{particular}>;\n\)

\(<\text{universal}> = <\text{universal positive}> | <\text{universal negative}>;\n\)

\(<\text{universal positive}> = (\text{every}, <\text{class name}>, \text{is}, <\text{class expression}>)
         | ((\text{class name}), \text{is a subclass of}, <\text{class expression}>);\n\)

\(<\text{universal negative}> = (\text{no}, <\text{class name}>, \text{is a}, <\text{class expression}>)
         | ((\text{class name}), \text{and}, <\text{class name}>, \{\text{and}, <\text{class name}>\}, \text{are disjoint});\n\)

\(<\text{particular}> = \text{some}, <\text{class name}>, (\text{are} | \text{aren't}), <\text{class expression}>;\n\)

\(<\text{fact}> = <\text{instance}> | <\text{relation}> | <\text{equation}> | <\text{different}>;\n\)

\(<\text{instance}> = <\text{indiv name}>, \text{is} | \text{isn't}, <\text{class expression}>;\n\)

\(<\text{relation}> = <\text{indiv name}>, <\text{predicate phrase}>, <\text{indiv name}>;\n\)

\(<\text{equation}> = <\text{indiv name}>, \text{is} | \text{isn't}, \text{the same as}, <\text{indiv name}>;\n\)

\(<\text{different}> = <\text{indiv name}>, \text{and}, <\text{indiv name}>, \{\text{and}, <\text{indiv name}>\}, \text{are different};\n\)

\(<\text{source}> = <\text{language ID}>, <\text{delimiter}>, <\text{source body}>, <\text{delimiter}>;\n\)

\(<\text{source body}> = \text{some syntactically valid expressions in the identified ontology language}\n\)

\(<\text{delimiter}> = \text{EOL}, "\"", \text{EOL};\n\)

5.4 Terms

\(<\text{class expression}> = <\text{class atom}> | <\text{conjunction}> | <\text{disjunction}> | <\text{qualified class}>\n\)

\(<\text{pos class}> = [\text{a} \mid \text{an}], <\text{class name}>;\n\)

\(<\text{class atom}> = [\text{not}], <\text{pos class}>;\n\)

\(<\text{conjunction}> = <\text{pos class}>, \text{and}, <\text{class atom}>, \text{and}, <\text{class atom}>
         | \text{neither}, <\text{pos class}>, \text{nor}, <\text{pos class}>, \{\text{nor}, <\text{pos class}>\};\n\)
\(\langle \text{disjunction} \rangle = \langle \text{class atom} \rangle, \text{or}, \langle \text{class atom} \rangle, \{\text{or}, \langle \text{class atom} \rangle\};\)

\(\langle \text{qualified class} \rangle = \langle \text{pos class} \rangle, [,], \text{who} | \text{which} | \text{that}, \langle \text{qualifier} \rangle;\)

\(\langle \text{qualifier} \rangle = (\text{is}, \langle \text{class expression} \rangle)\)
\[\text{or} (\langle \text{predicate phrase} \rangle, \langle \text{quantifier} \rangle, \langle \text{class expression} \rangle);\]

\(\langle \text{quantifier} \rangle = \text{some} | \text{only}\)

\(\langle \text{predicate phrase} \rangle = (\langle \text{predicate open} \rangle, \langle \text{predicate name} \rangle)\)
\[\text{or} (\text{is} | \text{isn’t}, [\text{a} | \text{an} | \text{the}], \langle \text{predicate fragment} \rangle, \langle \text{predicate end} \rangle);\]

\(\langle \text{predicate open} \rangle = \text{does} | \text{doesn’t} | (\text{has} | \text{hasn’t}, \text{as});\)

\(\langle \text{predicate end} \rangle = \text{of} | \text{than} | \text{to} | \text{on} | \text{in};\)

\(\langle \text{class name} \rangle = \langle \text{qname} \rangle \ (\text{see below})\)

\(\langle \text{predicate name} \rangle = \langle \text{qname} \rangle \ (\text{see below})\)

\(\langle \text{predicate fragment} \rangle = \langle \text{qname} \rangle \ (\text{see below})\)

\(\langle \text{indiv name} \rangle = \langle \text{qname} \rangle \ (\text{see below})\)

\(\langle \text{uriref} \rangle = <, \langle \text{relativeURI} \rangle, > \ (\text{see below})\)

The grammar for the Scone language does not explicitly show white spaces. White space is required between two terminals or non-terminals. The syntactic categories \textit{qname} and \textit{uriref} are defined as strings matching the \textit{qname} and \textit{uriref} production within the W3C Turtle Specification.\footnote{See \url{http://www.w3.org/TeamSubmission/turtle/}} The prefixes and the local parts of qnames must not match any of the keywords. Further, the prefix of a qname must not be ‘scn’.

A special case are predicate expressions like ‘father of’ or ‘larger than’. Here the predicate is divided up in two different words. Of course, this is a more general phenomenon in the English language, which cannot be easily supported in a controlled English like Scone. However, to provide at least limited support we introduced predicate end as syntactic category. The words in this category (e.g., ‘of’ and ‘than’) serve as keyword. The predicate is generated by concatenating the word that precedes the keyword with the keyword by an underscore. (E.g., ‘man who is father of some child’ is turned into ‘man and father _ of some child’.)
6 Semantics of the Scone Language

6.1 Scone and Gherkin

Scone extends Gherkin by providing a set of step definitions, which are relevant for ontology evaluation. This enables us to use Cucumber for the execution of Scone tests. Since writing new step definitions requires only some basic knowledge of Ruby and regular expressions\footnote{https://github.com/cucumber/cucumber/wiki/Step-Definitions} Scone is easily extendable and customizable.

Since Scone is an extension of Gherkin, the main structure of a Scone document (i.e., feature description, background, scenarios) is inherited from Gherkin. Further, the ability to tag scenarios and features as well as the keywords "*", "Given", "Then", "And", which are used to separate individual steps, are provided by Gherkin. Since Gherkin is well documented\footnote{https://github.com/cucumber/cucumber/wiki/Gherkin} we won’t explain the semantics of these aspects of Scone in this document in detail, but only to the degree to which it is necessary to understand the rest of Scone.

6.2 Mapping to DOL

The semantics of a Scone file is provided by its mapping to a DOL file. To consider this mapping, let’s reconsider the structure of a Scone file: it consists of (i) a description of the feature of the ontology that is tested (2) a setup of the backup, and (3) a number of scenarios, which consists of assumptions and competency questions.

The feature description is intended to document the purpose of the tests for humans, but for the formal semantics of the Scone file it is irrelevant. The background captures information that is shared across all scenarios in the file. This is convenient, but, from a theoretical perspective, does not add anything to the expressivity of the language, because it would be possible to add the directly to each scenario in the file. Thus, without any loss of generality we can ignore the background and focus on the scenarios.

Since the order of the statements within a scenario does not matter, any scenario is represented by the pattern in Fig ??\footnote{Note that we assume here exactly one language setting. The grammar actually does not require that. If no language setting is present, the default language is OWL. In the case of more than one language setting, we consider only the last one.}.
Scenario:
* Language language_ID
* Load the ontology <uriref_1>
  ...
* Load the ontology <uriref_k>

Given assumption_1
  ...
Given assumption_n

Then comp_question_1
  ...
Then comp_question_m

Figure 3: Generic schema of a SONE scenario

The semantics of the scenario is provided by a translation to a DOL file (see Fig. ??). Its first part defines an ontology importsAndAssumptions that aggregates all information from the imported ontologies and the formalization of the assumptions. The formalization-function that maps SONE sentences to sentences in the ontology language obviously depends on the chosen language. In the case of languages like OWL and Common Logic, the definition of the corresponding formalization-function is straightforward.
logic language_ID

ontology ont_1 = <uriref_1>
...
ontology ont_k = <uriref_k>

ontology combinedImports = ont_1 and ... and ont_k

ontology importsAndAssumptions = combinedImports then
  formalization[assumption_1]
  ...
  formalization[assumption_n]
end

ontology cq1 = importsAndAssumptions then
  %implies formalization[comp_question_1] end
...
ontology cqk = importsAndAssumptions then
  %implies formalization[comp_question_k] end

Figure 4: Translation to a DOL file

The second part of the file states the proof obligations between the ontology importsAndAssumptions and the competency questions. (Each competency question yields one proof obligation.) The formalization of the competency question depends on its kind.

- If the competency question is a test of the form "Then infer <sentence>",
  then the formalization[comp_question_i] equals formalization[<sentence>].

- If the competency question is of the form "Then the ontology is inconsistent",
  then formalization[comp_question_i] equals a representation of falsum in the chosen language.\(^\text{14}\)

- The representation of the two other cases is difficult in the current version of the syntax of DOL. We are in discussion with the DOL working group, which intends to add the necessary features to DOL.

Until this happens, we can evaluate competency questions of the type "Then the ontology is consistent" operationally by calling a consistency

---

\(^{14}\)E.g., "(and (p) (not (p))))" in Common Logic and "Class: owl:Thing SubClassOf owl:Nothing" in OWL.
checker on the ontology importsAndAssumptions. In the case of OWL ontologies, competency questions of the type "Then don’t infer <sentence>” can be evaluated operationally by attempting to infer the <sentence>” from the ontology. Because OWL is decidable, a failed attempt shows that the <sentence>” cannot be inferred, and, thus, the competency questions is passed successfully. In the case of Common Logic ontologies, competency questions of the type "Then don’t infer <sentence>” can be evaluated operationally by calling a consistency checker on the ontology that is the result of extending importsAndAssumptions by (not formalization[<sentence>]).

7 Mapping of Sentences and Terms to OWL

The following table maps Scone expressions to OWL expressions (in Manchester syntax). This mapping provides the semantics of Scone sentences. We will use ‘c’, ‘p’, ‘i’ (possibly with indices) as symbols for class names, predicate names, and individual names, respectively. ‘X’ and ‘Y’ are used for class expressions. ‘PredicatePhrase’ for predicate phrases. We simplify the mapping by ignoring grammatical variants, such as the use of articles (e.g., ‘a’, ‘the’) or the use of plurals.

Table 1: Definition of the formalization mapping for OWL

<table>
<thead>
<tr>
<th>formalization[&lt;scone&gt;]</th>
<th>OWL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Definitions and propositions</strong></td>
<td></td>
</tr>
<tr>
<td>[c is defined as Y]</td>
<td>Class:</td>
</tr>
<tr>
<td>[c is enumerated as i₁ and i₂ and . . . iₙ]</td>
<td>[c] EquivalentTo: {i₁, . . . , iₙ}</td>
</tr>
<tr>
<td>[Every c is Y]</td>
<td>Class:</td>
</tr>
<tr>
<td>[c is a subclass of Y]</td>
<td>Class:</td>
</tr>
<tr>
<td>[No c is a Y]</td>
<td>Class:</td>
</tr>
<tr>
<td>[c₁ and c₂ and . . . cₙ are disjoint]</td>
<td>DisjointClasses:</td>
</tr>
<tr>
<td>[Some X are Y]</td>
<td>Individual:</td>
</tr>
<tr>
<td>[Some X aren’t Y]</td>
<td>Individual:</td>
</tr>
<tr>
<td><strong>Facts</strong></td>
<td></td>
</tr>
<tr>
<td>[i is a Y]</td>
<td>Individual:</td>
</tr>
<tr>
<td>[i isn’t a Y]</td>
<td>Individual:</td>
</tr>
<tr>
<td>[i PredicatePhrase j]</td>
<td>Individual:</td>
</tr>
<tr>
<td>[i is the same as j]</td>
<td>SameIndividual:</td>
</tr>
<tr>
<td>[i₁ and i₂ and . . . iₙ are different]</td>
<td>DifferentIndividuals:</td>
</tr>
<tr>
<td><strong>Class expressions</strong></td>
<td></td>
</tr>
<tr>
<td>[not c]</td>
<td>not [c]</td>
</tr>
<tr>
<td>[c and X₁ and . . . and Xₙ]</td>
<td>([c] and [X₁] and . . . and [Xₙ])</td>
</tr>
</tbody>
</table>

Continued on next page
Table 1 – *Continued from previous page*

<table>
<thead>
<tr>
<th>formalization</th>
<th>OWL</th>
</tr>
</thead>
<tbody>
<tr>
<td>neither $X_1$ nor ... nor $X_n$</td>
<td>(not $[X_1]$ and ... and not $[X_n]$)</td>
</tr>
<tr>
<td>$X_1$ or ... or $X_n$</td>
<td>$([X_1]$ or ... or $[X_n]$)</td>
</tr>
<tr>
<td>$c$, which is $X$</td>
<td>$[c]$ and $[X]$</td>
</tr>
<tr>
<td>$c_1$, which PredicatePhrase only $c_2$</td>
<td>$(c_1$ and [PredicatePhrase] only $[c_2]$)</td>
</tr>
<tr>
<td>$c_1$, which PredicatePhrase some $c_2$</td>
<td>$(c_1$ and [PredicatePhrase] some $[c_2]$)</td>
</tr>
<tr>
<td>$c_1$, which PredicatePhrase some $c_2$</td>
<td>$(c_1$ and [PredicatePhrase] some $[c_2]$)</td>
</tr>
</tbody>
</table>

**Predicate phrases**

| [does p] | [p] |
| [doesn’t p] | not [p] |
| [is predFragment predEnd] | [p] where $p = \text{predFragment}_1 \text{predEnd}$ |
| [isn’t predFragment predEnd] | not [p] where $p = \text{predFragment}_2 \text{predEnd}$ |

**Names**

| class name: $[c]$ | $c$ |
| predicate name: $[p]$ | $p$ |
| indiv name: $[i]$ | $i$ |

† w is a new individual

8 Installation

To run Scone, you need an environment configured properly. We use Vagrant to build a Virtual Machine that contains everything you need, see the Scone-Vagrant repository. This project can be find at: [https://bitbucket.org/malefort/scone-vagrant](https://bitbucket.org/malefort/scone-vagrant)

The installation of SCON-E-vagrant is explained in detail on the homepage of this project. Additional information can be found on the wiki of the scone-vagrant project.


After the installation of the vagrant environment, one can retrieve Scone from the following HG repository: [https://bitbucket.org/malefort/scone](https://bitbucket.org/malefort/scone)

The installation of SCON-E is explained in detail on the homepage of this project. Additional information can be found on the wiki of the SCON-E project.


This project contains a bug-tracker, which can be used to report bugs and ask for feature requests.

---

15 As discussed in section ??, in these cases the predicate name is the result of concatenating the predicate fragment with the predicate end. E.g., the predicate fragment “father” together with the predicate end “of” yields “father_of”.

---

14