Project 1
Using SNARK for an Ontology of Grocery-Store Products
CSE 4/563, Knowledge Representation
Due Friday, March 6, 2008

Professor Stuart C. Shapiro

February 9, 2008

1 Introduction

1.1 Ontologies

In philosophy, "ontology" is the study of what there is. In AI, there have been many definitions of "ontology", but simply put, an "ontology" is a taxonomy of the terms in some domain of knowledge, possibly along with other domain rules relating the terms. In this project you will create an ontology of grocery-store products, represent the ontology in the language of SNARK [1], and use SNARK to answer some questions about the ontology.

1.2 SNARK

SNARK (SRI's New Automated Reasoning Kit) [1] may be used as a KRR system by performing the following steps. (See also [2].)

1. Run Common Lisp on `nickelback.cse.buffalo.edu` or on `timberlake.cse.buffalo.edu`.
2. Load the `ask`/`query` SNARK interface and SNARK itself, by entering the Lisp command
   `:ld /projects/shapiro/CSE563/ask`
3. Change the Common Lisp listener's package to `snark-user` by entering the Lisp command
   `:pa snark-user`
4. Load a knowledge base.
5. Ask the system questions.

The first Lisp form in the knowledge base file must be `(initialize)`. This must be followed by assertions in the format `(assert` *`wfp`*`)`, where *`wfp`* is a quoted form in the syntax required by SNARK. This is a CLIF-like syntax using the connectives `not`, `and`, `or`, `implies`, and `iff`. "Various synonyms are accepted" [1], including `=>` for `implies`. Then `prove` may be called, or, as an alternative, the `query` or `ask` function from `/projects/shapiro/CSE563/ask.cl` may be used. Note that if you use `prove` you must evaluate `(new-row-context)` between successive calls to `prove`. For an example knowledge base (KB) file, see `/projects/shapiro/CSE563/Examples/SNARK/CarPoolWorld.cl`.

An example run of a propositional CarPoolWorld system using SNARK and the `query` front end follows. Most of the output produced by the loading of `snark` has been deleted.

```
cl-user(1): :ld /projects/shapiro/CSE563/Examples/SNARK/CarPoolWorld
...
To start using SNARK, change the package to snark-user.
Then use (initialize), (assert <wfp>), and (prove <wfp>).
; Running SNARK from ...

Ask if ''Tom is the driver or Betty is the driver''
    is implied by the KB.  (True.)
  (ask '(or TomIsTheDriver BettyIsTheDriver)) = True

Ask if ''Betty is the driver
        implies that Betty is not the passenger''
    is implied.  (True.)
  (ask '(=> BettyIsTheDriver (not BettyIsThePassenger))) = True

Ask if ''Tom drives Betty and Tom is the passenger'' is implied. (False.)
  (ask '(and TomDrivesBetty TomIsThePassenger)) = False

Ask if ''Betty drives Tom'' is implied by the KB. (Unknown.)
  (ask '(BettyDrivesTom)) = Unknown
```

Notice that `(ask` *`wfp`*`)` can return `True`, `False`, or `Unknown`. It returns `True` if *`wfp`* can be proved from the KB, `False` if `(not` *`wfp`*`)` can be proved from the KB, and `Unknown` if there are models that satisfy KB $\wedge$ *`wfp`* as well as models that satisfy KB $\wedge\neg$ *`wfp`*. We know that `ask` will terminate and return one of these answers because resolution refutation is a decision procedure for propositional logic.

If you evaluate `(query` *`comment wfp`*`)`, the comment is printed, `(ask` *`wfp`*`)` is called, and then the call to `ask` is printed along with the value it returns. The `query` function is the most convenient SNARK front-end to use at the end of a KB file, because you just load the file, and see what questions were posed to SNARK and how they were answered.

2 The Project

2.1 Project Statement

You are to use the SNARK to implement and demonstrate a propositional logic version of an ontology of products available in a large grocery store such as Wegmans or Tops.

2.2 Background

2.2.1 Category Hierarchies

Grocery products can be categorized in a taxonomy (in computer science, also called a hierarchy or tree). For example, if you go to Webman's web site [3], and click on "shopping=>Browse the store", you will find, under "Products", a list of "Departments", including "Bakery", "Cheese Shop", "Meat", "Produce", and "Seafood". "Meat", "Produce", and "Seafood" are categories of grocery-store products, but "Bakery" and "Cheese Shop" are clearly store departments. We could, however, reinterpret the latter as the product categories, "Bakery Products" and "Cheese". If you click on "Produce", you will see a list of food categories including "Apples", "Citrus Fruits", and "Lettuce and Greens". If you click on "Citrus Fruits", you will see more categories including "Oranges".

We will treat the subcategory relation as reflexive. That means that every category is a subcategory of itself. For example, the category of apples is a subcategory of the category of apples. If we want to talk about a subcategory of

A other than A itself, we will call it a **proper subcategory** of A. For example, the category of oranges is a proper subcategory of the category of citrus fruits.

An important property of hierarchies is that the subcategory relation is **transitive**. For example, since the category of oranges is a subcategory of the category of citrus fruits, and the category of citrus fruits is a subcategory of produce, then the category of oranges is a subcategory of the category of produce.

2.2.2 An Issue of Semantics

You will use propositional logic for this project. But this raises an issue of semantics. What is the intensional semantics of an atom like $Orange$? (It is traditional in KR to use the singular rather than the plural to name a category, that is $Orange$ rather than $Oranges$.) Above, we spoke about the category of oranges. So is $[Orange]$ the category of oranges? In propositional logic, the intensional semantics of an atomic proposition is a proposition, something that can be True or False. The category of oranges is not a proposition, and can't be True or False. So, if we're to retain the usual flavor of propositional logic, we'd have to imagine that we're talking about a particular thing, and $[Orange]$ would be *"It's an orange."* Then $(Orange \Rightarrow CitrusFruit)$ makes sense—*"If it's an orange then it's a citrus fruit."* The transitivity of the subcategory relation is captured by the transitivity of $\Rightarrow$; that is, $[(A \Rightarrow B) \wedge (B \Rightarrow C)] \Rightarrow (A \Rightarrow C)$. Similarly, the reflexivity of the subcategory relation is captured by the reflexivity of $\Rightarrow$; that is, $A \Rightarrow A$.

2.2.3 Instances

In our commonsense understanding of the world of grocery-store products, we not only have categories of products, we also have particular, individual products. For example, we not only have the category of oranges, we also have the particular, individual orange I ate for lunch today. We say that that orange is **an instance** of the category of oranges. Things can work out if we are careful in our intensional semantics. For example, we can declare the intensional semantics of some atomic proposition, say $orange31$, to be *"It's the orange Prof. Shapiro ate for lunch today."* Then, if $[Orange]$ is *"It's an orange"*, the wfp $(orange31 \Rightarrow Orange)$ makes sense, and accords with our understanding of the world.

Notice that an instance of some category is also an instance of all supercategories of that category. For example, orange31 is an instance of oranges, an instance of citrus fruits, and an instance of produce.

2.2.4 Partitions and Other Category Groupings

If B, C, and D are proper subcategories of A, we say that they **exhaust** A if every proper subcategory of A must be a subcategory of B, C or D, and every instance of A must be an instance of B, C, or D.

We say that a set of categories are **mutually exclusive** or **mutually disjoint** if any subcategory of one is not a subcategory of the others, and every instance of one is not an instance of the others.

We say that a set of categories, $B_1, \ldots, B_n$ **partition** a category A, if: B_1, and $\ldots$, and B_n are proper subcategories of A; B_1, and $\ldots$, and B_n exhaust A; and B_1, and $\ldots$, and B_n are mutually disjoint. For example, in the Wegmans produce hierarchy, Clementines, Honey Tangerines, and Mineola Tangerines partition the category of tangerines.

2.2.5 Multiple Inheritance

Although hierarchies usually form a tree (every category has at most one immediate supercategory), there may be multiple ways to categorize a domain, so that one category winds up being an immediate subcategory of several supercategories, one in each way of categorizing the domain. For example, besides the subcategories of produce discussed above, we can also categorize produce into organic and conventional. Additionally, Wegmans categorizes "Products" into "Department" categories, "Popular Brands" categories, and "Wegmans Wellness Keys" categories. It's possible for one subcategory of product to have supercategories in each of these three categorizations.

Multiple inheritance is indicated when an individual can be an instance of several categories, or a category can be a subcategory of several supercategories, none of which are subcategories of any of the others.

2.2.6 Negative Information

It is easy to imagine asking your system questions of the form *"if it's A, is it B?"* For example, to ask *"If it's an orange, is it a citrus fruit?"*, you might ask `(=> Orange CitrusFruit)` and you'd expect to get the answer `True`. In fact, unless we tell the system what individual we're considering (by saying something like *"It's orange31."*), all the questions will be conditionals—questions of the form `(=> A B)`, where `A` and `B` are not necessarily atomic. Could we ever expect to get an answer of `False` to such a question? If we do get an answer of `False,` that means that a wfp of the form `(not (=> A B))` is `True.` However, `(not (=> A B))` is logically equivalent to `(and A (not B))`, which we could not expect to be implied by the KB unless we've told it `A` (or something that implies `A`). An example of what this means is that to get the system to tell us that *If it's an orange, it's not a bakery product*, we should not expect to get an answer of `False` to the question `(=> Orange BakeryProduct)`, but an answer of `True` to the question `(=> Orange (not BakerProduct))`.

2.3 The Task

You are to create a knowledge base of an ontology of grocery-store products. You may collect your data from any reference sources, but you must cite your sources properly in your paper. Your knowledge base must include:

multiple inheritance: at least two categories that are each subcategories of at least three supercategories none of which is a subcategory of any of the other two;

disjoint categories: at least one category with at least three subcategories that are mutually disjoint but do not exhaust their supercategory;

partitions: at least two categories, each of which is partitioned by its subcategories.

In addition:

sufficient depth: no leaf category may be an immediate subcategory of a root category.

sufficient specificity: every leaf category must be one that is separately priced in the grocery store (For example, `Cereal` is not specific enough to be a leaf category, but `Honey Nut Cheerios` is. However, you needn't distinguish different sizes of the same category.);

multiple examples: every non-leaf category must have at least two immediate subcategories.

You are to include a demonstration of your system that uses the `query` function to ask SNARK questions that you devise. If A, B, and C are wfps, not necessarily atomic, your questions must include at least one of each of the following forms for which the answer is `True`:

- $A \Rightarrow B$ testing the transitivity of the subclass relation;
- $A_1 \vee \cdots \vee A_n \Rightarrow B$ also testing the transitivity of the subclass relation;
- $A \Rightarrow \neg B$ testing the mutual disjointness of some classes;
- $A \Rightarrow B_1 \vee \cdots \vee B_n$ testing the exhaustiveness of some subcategories;
- $A \wedge \neg B_1 \wedge \cdots \wedge \neg B_n \Rightarrow C$ also testing the exhaustiveness of some subcategories.

Do not use the same categories for all of these.

In addition,

- you must ask at least one question for which the answer is `Unknown`,

and

- every derivation answered `True` must require at least two reasoning steps.

2.4 Deliverables

As it says on the CSE 4/563 web page, you are to

> "hand in a paper, produced using a document formatting program such as LaTeX or Microsoft Word, and printed on 8.5 by 11 inch paper, stapled in the upper left-hand corner, with a title, your name, and other identifying information at the top of the first page (Do not use the header page automatically produced by the printer), plus a well-documented listing and run of your program. (Do not enclose your paper in a folder or cover.) The main product of your work is the paper, not the program!" [5]

The paper is due at the start of class, on the date specified in the title of this document.

In addition to the paper, you are to `submit` (using `submit_cse463` or `submit_cse563`) your program, so that it can be run and checked if the instructors choose. You are to submit your program by 10:30 AM on the date due, so that you can get to class by 11:00 AM to hand in the paper.

Note that "your program" is the KB file and the sequence of calls to `query`. In fact, for ease of testing and debugging, your KB file can end with the sequence of calls to `query`, as does `/projects/shapiro/CSE563/CarPoolWorld.cl`. Name your file `proj1KB.cl`.

2.4.1 The Paper

Your paper should have the following parts:

1. Descriptive title
2. Author identification
3. Introduction: general description of the project
4. Domain: English description of the domain your program works in
5. Formalism
   (a) Syntax and intensional semantics of atomic propositions, ordered alphabetically
   (b) English and formal presentation of information in the KB
6. Discussions and demonstrations of significant aspects of your program.
7. Acknowledgments as needed
8. References as needed

In §5b of your paper you must explicitly point out where you satisfy each of the points listed under "KB" in the following Grading Table, and in §6, you must separately demonstrate and discuss each of the required points listed under "Questions" in the Grading Table. The grader might run your submitted program to check that it really does what you claim in the paper, but **it will be assumed that you have not done anything you do not explicitly discuss in the paper.** In particular, you must accompany each query with an informal English explanation of the reasoning your system goes through to answer it, to show that at least two reasoning steps are required. For example, an explanation of the answer to the first query about CarPool World given on page 2 might be:

> Tom drives Betty or Betty drives Tom. If Tom drives Betty, then Tom is the driver. If Betty drives Tom, then Betty is the driver. Therefore, either Tom is the driver or Betty is the driver.

The fact that there are two "*if . . . then*" statements and one "*Therefore*" conclusion in this explanation shows that there are three reasoning steps involved.

2.5 Grading

Project grading will be according to the following table.

	CSE 463	CSE 563
KB		
Multiple inheritance	5	5
Disjoint categories	5	5
Partitions	5	5
Sufficient depth	5	5
Sufficient specificity	5	5
Multiple examples	5	5
KB Subtotal	**30**	**30**
Questions		
True question of form $A \Rightarrow B$	5	5
True question of form $A_1 \vee \cdots \vee A_n \Rightarrow B$	5	5
True question of form $A \Rightarrow \neg B$	5	5
True question of form $A \Rightarrow B_1 \vee \cdots \vee B_n$	5	5
True question of form $A \wedge \neg B_1 \wedge \cdots \wedge \neg B_n \Rightarrow C$	5	5
Not same categories	6	5
At least two reasoning steps for each True question	6	5
At least one Unknown question	5	5
Questions Subtotal	**42**	**40**
Paper		
Paper format	4	4
General project description	4	4
English description of domain	4	4
Syntax and semantics of atomic propositions	4	5
Description of formalization of KB	4	4
Discussions and demonstrations	4	4
Acknowledgments and References	4	5
Paper Subtotal	**28**	**30**
Project Total	**100**	**100**

Acknowledgments

Prof. Shapiro appreciates the advice of Richard Waldinger, SRI International, on the SNARK function `(new-row-context)` that flushes clauses generated after the assertions.

References

1. Mark E. Stickel, Richard J. Waldinger, & Vinay K. Chaudhri, “A Guide to SNARK”, `http://www.ai.sri.com/snark/tutorial/tutorial.html`.
2. Stuart C. Shapiro, “UB CSE2/563 Resources,” `http://www.cse.buffalo.edu/˜shapiro/Courses/CSE563/2009/resources.html#snark`
3. Wegmans, “Welcome to Wegmans Food Markets Inc.,” `http://www.wegmans.com/`.
4. Stuart C. Shapiro, “UB CSE2/563,” `http://www.cse.buffalo.edu/˜shapiro/Courses/CSE563/2009/`.