Session 1 and 2 : FO($\cdot$) and IDP

1 FO

1.1 Translation of natural language to first-order logic

Pick a vocabulary and translate the following sentences:

1. John and Peter are both not happy.
2. John and Peter are not both happy.
3. A necessary condition for John to be happy is that Peter is happy.
4. All men like all vegetarians.
5. All men except butchers like all vegetarians.
6. Some man is a butcher and a vegetarian.
7. No man is both a butcher and a vegetarian.
8. No man likes a woman who is vegetarian.
9. Not all men that are vegetarians are happy.
10. Only men like butchers.
11. Some butcher likes all vegetarians.
12. John likes all vegetarians.
13. John does not like any vegetarian.
14. All butchers are not vegetarians.
15. For every butcher there is a vegetarian who likes him.

1.2 Intermezzo

In the previous exercise, it is possible to use both `Happy(John)` and `John(happy)`. However, one of them is the better choice. Try to answer the following question to get to the conclusion.

1. Unary predicates denote sets. Should `John` or `Happy` denote a set?
2. If `John` denotes a set of properties such as `happy`, how would you describe that John has an age of 43 years and a shoe size of 43?
3. Two predicates are equal if they denote the same set. Suppose `Bob` and `John` have the same properties, then which possibility (`John` and `Bob` as predicates or as domain elements) allows to still make a distinction between them?
4. If `John` and `Bob` are predicates, how would you translate that they are friends?
5. Suppose we want to translate the following sentence: "All rich people are unhappy"? Why is the following sentence not a valid FO formula?

$$\forall X : X(\texttt{rich}) \Rightarrow \neg X(\texttt{happy})$$

1.3 Quantors

Which of the following sentences are true in the natural numbers?

1. $\forall x : \exists y : x \geq y$
2. $\forall x : \exists y : x \leq y$
3. $\exists y : \forall x : x \geq y$
4. $\exists y : \forall x : x \leq y$
5. $\forall x : \exists y : \neg(x \leq y)$
6. $\forall x : \exists y : \neg(x \geq y)$
7. $\exists y : \forall x : \neg(x \leq y)$
8. $\exists y : \forall x : \neg(x \geq y)$
9. $\exists y : \neg\forall x : x \leq y$
10. $\exists y : \neg\forall x : x \geq y$
11. $\forall x : \neg\exists y : x \geq y$
12. $\forall x : \neg\exists y : x \leq y$

2 Structures

For a given vocabulary, a structure over that vocabulary is an assignment of values to the symbols in the vocabulary. This is a mathematical abstraction of the state of affairs. For the state of affairs implied by the statements given below, write a structure, abstracting this state of affairs, over the following vocabulary:

- Person/1
- Age/2
- Friends/2

- Oldest/0:

An example of a structure over this vocabulary:

- Person = {Bert; Ernie}
- Age = {(Bert, 20); (Ernie, 21)}
- Friends = {}
- Oldest = Ernie

Create a structure satisfying all the following statements at once:

- An is 16 years old and friends with Pete, who is older.
- Everyone who has friends is a person
- Fred, who is 14, does not have any friends, he is still a person though.
- Betty has two friends, she is younger than both of them.
- Every person has an age.
- The earth is older than any person.
- No one is friends with someone who is not friends with them.
- The objects discussed here are the only ones we know anything about.
- When deciding who is the oldest, only people are compared.

3 Model Expansion

- What? Inference task
- Given:
  - An interpretation for certain predicates and functions (e.g. Parent/2)
  - A number of other predicates and functions for which we don't have (complete) data (e.g. Ancestor/2)
  - A number of constraints that limit/determine the possible values for the unknown predicates
- Goal: find all the possible (according to the constraints) values for the unknown predicates and functions.

3.1 A guideline in solving model expansion problems

1. Determine which types you will need.
2. Determine for which predicates/functions we have complete data: they are the *given* predicates/functions.
3. Determine for which predicates/functions we need to find the possible values: they are the predicates/functions we want to *find*.
4. Write down the necessary constraints. In this process it might be the case that we want to use some additional new predicates/functions to make the modelling easier.
5. Write down the data for the given predicates

3.2 The IDP-framework: Getting started

There is a document on Toledo describing how to get the IDP framework. Follow these instructions.

When solving Model Expansion (MX) problems with IDP, your specification will probably consist of three components:

- A *vocabulary*. This is where you declare your ontology. It looks for example like this:

```
vocabulary V {
  type Food
  type Fruit isa Food
  type Label
  Labeling(Fruit,Label)
  LabelOf(Fruit): Label
}
```

  This declares a vocabulary with three types (Food, Fruit, and Label), a predicate symbol Labeling (with arity two, where the first argument should be a Fruit and the second a Label) and a function sending a Fruit to its Label. Functions are by default total: this means that every fruit has to have a label, if we use the above vocabulary.

- The second component you will need is a *theory* (over the above vocabulary) expressing your constraints, for example

```
theory T : V {
  ! f[Fruit]: ? n[Label]: Labeling(f,n).
}
```

  expressing that for all ($\forall$ is `!`) objects f of type Fruit, there is ($\exists$ is `?`) an object n of type Label such that `Labeling(f,n)` holds. You can use all FO-connectives in such constraints ($\wedge$ is &, $\vee$ is |, $\Rightarrow$ is =>, $\Leftrightarrow$ is <=> and negation is $\sim$) as well as aggregates, inductive definitions, arithmetic,.... (these are also listed in the help function of the IDE, for details, check the IDP manual [1]).

- The last of the required components is a (partial) *structure*: it contains your data. For example

  ```
  structure S : V {
    Fruit = {Apple;Pear}
    Label = {1..7}
    LabelOf = {Apple->1; Pear->5}
  }
  ```

  declares a structure with two Fruit: Apple and Pear, and 7 labels, numbered 1 to 7. It also specifies the LabelOf function. In a predicate interpretation, elements within a tuple are separated by commas, different tuples are separated by semicolons.

When you have your ontology (`V`), your description of the problem domain (`T`) and your data (`S`), you are not yet solving a problem. These things can be used to solve various problems. Now, concerning the MX problem, you probably want to write a procedure (e.g.):

```
procedure main() {
  stdoptions.nbmodels = 5
  sols = modelexpand(T,S)
  print(sols[3])
  printmodels(sols)
}
```

This commands used in this procedure say: “I want 5 models (0 = all models)”, “execute MX with T and S, store the list of solutions in sols” and “print the third model”, and “print all found models”.

HINT: When modelling, especially in the begin, you will make mistakes: common mistakes (and ways to solve them) are:

- Confusing $\Rightarrow$ with $\land$, for example stating $\forall x : Man(x) \land Person(x)$ if you want to say “all men are persons”. **Rule of thumb**: $\forall$ always comes with a $\Rightarrow$, $\exists$ always comes with a $\land$.

- Declaring a function, while actually it is not (for example the successor function on the natural numbers) is only a *partial* function on the interval 0..7 as 7 has no successor. (check your function constraints)

HINT: When modelling, you will make mistakes (also others than the ones above). Finding them is not always easy. Basically there are two kinds of mistakes:

[1] `http://dtai.cs.kuleuven.be/krr/files/bib/manuals/idp3-manual.pdf`

- You make your constraints too weak, and get unintended models. Finding these kinds of mistakes is often easy. You check *why* the unintended models should not be models. The constraint excluding that one, is probably wrong.

- You make your constraints too strong and get UNSAT: thus, your mistake can be anywhere in the theory... In that case, it might be useful to execute the procedure `explainunsat(T,S)`. This procedure tries to find a minimal set of constraints that are already UNSAT together, thus giving you better directions on where you can find the bug. This procedure is not always complete. It can be executed as follows:

  ```
  procedure main() {
    explainunsat(T,S)
  }
  ```

  Expand the previous example so that the theory states that each label belongs to a fruit and no fruit has two labels:

  ```
  theory T : V {
    ! f[Fruit]: ? n[Label]: Labeling(f,n).

    ! n[Label]: ? f[Fruit]: Labeling(f,n).
    ! f[Fruit] n1[Label] n2[Label]: Labeling(f, n1) &
        Labeling(f, n2) => n1 = n2.
  }
  ```

  Executing MX with T and S now results in unsat. Run explainunsat as shown above and IDP will show you why the combination of this theory and structure is unsatisfiable. It achieves this by showing you that to satisfy the second sentence, there must be a fruit labeled by more than one label.

3.3 Optimal model expansion

Model expansion is a useful task, but sometimes, you are not just interested in any model, but only in models that are minimal with respect to a certain cost. You can do this in IDP by making a term block, for example

```
term t : V {
  sum{x y: P(x,y): f(x)}
}
```

is a term expressing the sum over all tuples x, y such that $P(x, y)$ holds, for all these tuples, we sum the value $f(x)$. As you see, this is an aggregate: sum is a function working on a set. You can also use aggregates in FO$(\cdot)$ theories. For example:

```
! f[Fruit]: #{n:Labeling(f,n)} < 7.
```

expresses that every fruit can only be labeled with less than seven labels. As you see, the first of the aggregates (sum) needs a third argument, a term to sum, but cardinality does not.

In order to find minimal models, you can use the minimize procedure.

Now suppose a graph is given (see file `ShortestPath.idp` on Toledo) and your task is to find a shortest path between two given nodes (Start and End). Make an IDP theory such that models of the theory are exactly loop-free paths from Start to End. Such a path should satisfy:

- For each node, there is at most one outgoing edge on the path and at most one incoming node on the path
- It starts in Start and ends in End
- Not just any edge can be on the path: all edges on the path should be connected for it to be a real path.

Make a term equal to the length of the path.

Search for minimal models.

3.4 Getting used to writing theories

Download the `scienceweek.zip` file from toledo. Extract it to your IDP working folder (∼`/idp/` is the default) . We will use these demos throughout the exercise session.

3.4.1 Graph coloring

In the IDE, pick the file `coloring` from the `Scienceweek/graphColoring` folder. This demo shows how to model the graph coloring problem in IDP.

The constraint describing that *two adjacent nodes can not have the same color* is missing. Specify this constraint in the theory `T` and check your solution by pressing `Run`.

The solution should use four colors, with no two same colored nodes connected by an edge. Edges will be colored red if they connect two nodes with the same color.

3.4.2 Sudoku

Pick the `sudoku.idp` file from the `Scienceweek/sudoku` folder. This demo shows how to model Sudoku in IDP.

The block to which each cell belongs is defined. However the essential constraints, modelling correctly solved sudoku puzzles, are missing. Add these constraint and press Run to verify your solution.

3.4.3 Hanoi

Pick the `hanoi.idp` file from the `Scienceweek/hanoi` folder. This demo shows how to model the towers of Hanoi puzzle in IDP.

The goal of the puzzle is to move a tower consisting of three blocks from left to right, by moving only one block at a time. The blocks have three different sizes. While moving the blocks it is only allowed to place a small block on top of a larger block, never the other way around. Blocks can only be moved when there is no other block on top of them.

Fill in the missing constraints in the theory T. Do not change the definitions.

3.4.4 N-Queens

Pick the `nqueens.idp` file from the `Scienceweek/nqueens` folder. This demo shows how to model the N-Queens problem in IDP.

The problem is to put N queens on a chess-board of size N×N so that they cannot attack each other. A queen can attack another queen if it is in the same column, row or diagonal.

3.4.5 Roster

Pick the `roster.idp` file from the `Scienceweek/roster` folder. This demo shows how a course roster can be modelled in IDP.

As before add the constraints, as they are explained by the comments, to the theory. You will need **aggregates** and a **definition**. Your mistakes will be highlighted with colors in the visualisation.

3.5 Schur numbers

The Schur number problem is to partition n (positive) integers into m sets such that all of the sets are sum-free. By sum-free, we mean that, if x and y ($x = y$ is possible) are assigned to the same set, then $x + y$ is not in the set. In our version of the Schur number problem, we permit an extra input that specifies the assignment of some numbers to sets (a partial solution). The task then is to check whether the given partial assignment can be extended to a full sum-free assignment. Write a theory in FO which describes a solution to the above problem. Given the set of integers from 1 to 70, check whether it is possible to partition them into 5 partitions such that they satisfy the Schur criteria, extending the following partial assignment:

- {1,8,12,14} are in partition 1
- {3,5,7,11,13,15} are in partition 3
- {2,6,10} are in partition 4
- {4} is in partition 5

3.6 Inductive definitions

Given a graph G.

- Write down a pure FO theory expressing the transitive closure of the graph, and show that this theory has unintended models (take a very simple graph, e.g. a graph with two nodes A and B, and one edge (A, A), and ask IDP to find all models). **Hint:** to find and print all models, replace your main procedure by

  ```
  procedure main() {
    printmodels(allmodels(T,S))
  }
  ```

- Now write down the correct definition using inductive definitions, and verify that this theory does not have the unintended models of the first theory.

3.7 Constructors

As seen in the course, sometimes we want to have a type that consists of a set of constants with Unique Names and Domain Closure Axioms (UNA and DCA). In IDP, this kind of types are called constructed types. A constructed type is declared in the vocabulary as follows

```
vocabulary V {
  type T constructed from {a,b,c}
}
```

This declares a type `T`, with three constants `a,b` and `c` such that

- `a`, `b` and `c` are different (unique names)
- Every structure over `V` interprets `T` as a type with exactly three elements (domain closure)

As an exercise, please solve the following simplified variant of the “Einstein puzzle”. A Belgian guy, an American guy and a German guy live in the same street. They live in three houses, a red, a green and a blue. In every house, of course, one of them lives. Everyone has a different pet: a dog, cat or a fish. Try to find out who lives in which house, and has which pet using the following hints. (Please, use constructed types!!!)

- The person who owns the fish lives in the blue house
- The American does not live in the blue house
- The Belgian guy dislikes dogs and the color red
- The person who lives in the green house has a dog
- If the German has a cat, then he lives in the blue house.

Also verify that this puzzle has a unique solution.

3.8 Scheduling problem

Consider the following situation regarding a scheduling problem at a university:

- Courses consist of lectures which must be taught in rooms during time slots.
- Some courses have many students, while others have few students.
- Rooms may be large or small. Only large rooms may be used for courses with many students. Some rooms have multimedia, others do not.
- Each course has a given number of lectures. There is no fixed order of the lectures, but some lectures have to precede other lectures. Some lectures need multimedia, others do not.
- There a are a number of time slots available.

Problems:

- Develop your own ontology and vocabulary to represent this domain.
- Access the file `scheduling_vocanddata.idp` (see toledo under exercise session 1-2). This file specifies a vocabulary, plus the data of a simple scheduling problem. Compare your ontology with the one here.
- Specify the laws that a correct schedule will satisfy using this ontology in your theory.
- Combine your theory together with the given structure and run IDP to generate all solutions to the scheduling problem.
- Check if this theory and the given data entail that lecture C11 does not take place in room R1.