# Session 3: Inferences in IDP

Given the following statements: I am one year older than the double of my sons age. The sum of our ages lies between 70 and 80. My son is an adult if he is at least 21 years old. What problems can I solve with this information? A few examples.

- Derive our possible ages
- Prove that my son is an adult
- Given: my actual age is the solution where I am the oldest. How old are we?
- Given: I am 53. How old are we?

```
vocabulary V {
  type age isa int
  myAge:age
  sonAge:age

  SonAdult
}

theory T : V {
  myAge = 2 * sonAge + 1.
  myAge + sonAge >= 70.
  myAge + sonAge =< 80.

  SonAdult <=> sonAge >= 21.
}

structure S : V {
  age = {1..100}
}
```

We can solve all these problems within IDP. We introduce a vocabulary consisting of a type representing our age and two constants representing the ages of the father and his son. We define a theory that expresses the necessary constraints. Lastly, we initialize the type 'age'. Now we can solve the various problems:

# 1 Overview of inferences

## 1.1 Model expansion

Given a (partial) structure and a theory, model expansion returns models of the theory expanding the given structure. If you want to find and print all models of a theory that extends that a given structure, we do the following in IDP:

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

If we want to find and print just a single model, we do

```
procedure main() {
  print(onemodel(T,S))
}
```

If you want to find and print a custom number of models you do the following

```
procedure main() {
  stdoptions.nbmodels = n
  printmodels(modelexpand(T,S))
}
```

where n is any number greater or equal 0 (0 means find all models). Applying model expansion to the age problem above gives models that represent possible ages.

**Output vocabulary** If you are only interested in parts of your models, it is possible to provide a subvocabulary containing only the symbols you are interested in.

```
vocabulary Vson {
  extern V::sonAge/0:1
}
```

Then we can supply this subvocabulary to the model expansion functions:

```
procedure main() {
  printmodels(modelexpand(T,S,Vson))
  printmodels(allmodesl(T,S,Vson))
  print(onemodel(T,S,Vson))
}
```

This will print structures looking like

```
structure : Vson {
  age = {1..100}
  sonAge = 23
}
```

## 1.2 Model iteration

Sometimes you want to dynamically decide whether another model should be computed. This can be done by the model iterator. Instead of a list of models, a call to `modelIterator` returns an iterator object, which returns a new model every time you execute it. `modelIterator` takes the same arguments as the model expansion functions, so it is also possible to include an output vocabulary if you are only interested in parts of the models.

```
procedure main() {
  iterator = modelIterator(T,S)
  // iterator = modelIterator(T,S,Vson)

  print("Print another model?")
  while io.stdin:read() == "yes" do
    model = iterator()
    print(model)
  end
}
```

## 1.3 Satisfiability checking

An inference related to model expansion is satisfiability checking. This inference takes as input a structure and a theory, and return a boolean stating whether the given structure satisfies the theory. For example, given the following structures

```
structure Scorrect : V {
  sonAge = 23
  myAge = 47
}

structure Sincorrect : V {
  sonAge = 23
  myAge = 50
}
```

we can print if they satisfy the theory.

```
procedure main() {
  print(sat(T,Scorrect))   // true
  print(sat(T,Sincorrect)) // false
}
```

## 1.4 Optimal model expansion

Sometimes you are interested in a particular model that is minimal or maximal in some sense. IDP provides optimal model expansion in the form of

two methods: `minimize` and `maximize`. Both these procedures need something to minimze or maximize. IDP introduces term blocks for this. The following is age of the son as a term:

```
term sonTerm : V {
  sonAge
}
```

This term then can be used to find an optimal model:

```
procedure main() {
  printmodels(maximize(T,S,sonTerm))
  printmodels(minimize(T,S,sonTerm))
}
```

It is also possible to supply an output vocabulary as a fourth argument:

```
procedure main() {
  printmodels(maximize(T,S,sonTerm,Vson))
  printmodels(minimize(T,S,sonTerm,Vson))
}
```

### 1.5 Optimal propagation

If you want to prove that the son in our problem is always an adult, this can be done with optimal propagation. This inference will expand a structure with information which is true in every model. Since we are only interested in the predicate `SonAdult` we provide an output vocabulary

```
vocabulary Vadult {
  extern V::SonAdult/0
}
```

However, `optimalpropagate` does not take an output vocabulary as input. So we have to manually recast the vocabulary.

```
procedure main() {
  sol = optimalpropagate(T,S)
  setvocabulary(sol,Vadult)
  print(sol)
}
```

### 1.6 entailment

To check that a fact is a logical consequence of theory you can use optimal propagate. However, sometimes it is useful to check if a given theory is a logical consequence of another theory. If you want check if a theory is a logical consequence of another theory, you can use the procedure `entail`.

## 1.7 Explaining unsatisfiability

Sometimes when searching for models you encounter an `unsat`, meaning that no models can be found. This can be intended, but most of the time there is a mistake in your theory (or sometimes in your structure you want to expand). The inference `explainunsat` will provide an explanation as follows: Given a structure $S$, a theory $T$, and vocabulary $V$, it will search for a structure $S'$ and theory $T'$ such that

- $S$ is more precise than $S'$ and $T$ entails $T'$.
- $S$ is equal to $S'$ on the symbols outside $V$.
- $S$ is maximally imprecise with respect to the above criteria.
- $T'$ is maximally restrictive with respect to the above criteria.

Intuitively, this means that `explainunsat` searches for a smallest subset of $S$ and smallest subset of $T$ for which the unsatisfiability remains. Investigating this new structure and theory will give pointers to what caused the unsatisfiability. As an example add the theory

```
theory Told : V {
  myAge = 81.
}
```

Then we can combine the two theories and run `explainunsat`.

```
procedure main() {
  Tcombined = merge(T,Told)
  explainunsat(Tcombined,S)
}
```

If you only want the structure, the procedure `unsatstructure` does the job.

## 1.8 Query

Sometimes it is useful to do more with a model than just print it. IDP supports querying to ask specific questions about a structure. Recall the map coloring problem from previous exercise session. One could be interested in knowing the colors which were not used for coloring a country. For this purpose, IDP uses a query block, which represents a set expression. In the following example, the user is interested in the colors, for which there does not exist an area, that is colored in that color.

```
query Q : V {
  {c[Color] : ~(? a[Area]: Coloring(a) = c)}
}
```

Now, after calculating a model of the theory, we can execute this query over the resulting structure.

```
procedure main() {
  sol = onemodel(T,S)
  print(sol)
  print("Unused colors:")
  print(query(Q,sol))
}
```

The above queries return a set of domain elements. However, sometimes a boolean query is also useful. Returning back to the age puzzle, we can query a model if the son is an adult. The following syntax is used for the query.

```
query SonIsAdult : V {
  {: SonAdult}
}
```

Printing the result of this query will give either `true` or `false`.

## 2 Using inferences to solve problems

Using multiple inferences on a single piece of knowledge is the essence of the knowledge-base paradigm.

**Exercise 1** Determine how to solve the following problem by stating which inferences are used (and in which order). Provide any extra vocabulary, structure, theory, term, query, or procedure you need to solve the problem.

1. Determine if a partially filled in sudoku is correct. (this includes that the puzzle can be filled in further without violating the rule)
2. Given a map of countries, determine the minimal amount of colors to color the map so that neighboring countries are colored differently.
3. A sudoku maker has some partially filled in sudoku and wants to know if there exists a unique solution.
4. The same sudoku maker has a partially filled in puzzle with more than one solution. Find all solutions that have a 5 in the cell with row 1 and column 1.
5. A train company has their time table converted to IDP. Determine the shortest (in time) route between two given stations.
6. The same train company wants to identify the routes on which more than 5 trains are driving in a single hour.
7. Given a partial solution to a Sudoku. Determine all the consequences using only the rule that row $r$ cannot contain multiple 5s.

8. For each of the consequences found in the previous exercise, determine the minimal amount of facts from the initial partial solution that derive the consequence. This serves as an explanation to the consequence. HINT: negate the consequence and add to the original partial solution.

9. Suppose the rules for sudoku are ordered from easy to hard such as

   (a) A row contains at least one $n$ for each $1 \leq n \leq 9$.

   (b) A column contains at least one $n$ for each $1 \leq n \leq 9$.

   (c) A block contains at least one $n$ for each $1 \leq n \leq 9$.

   (d) A row cannot contain multiple $n$ for each $1 \leq n \leq 9$.

   (e) A column cannot contain multiple $n$ for each $1 \leq n \leq 9$.

   (f) A block cannot contain multiple $n$ for each $1 \leq n \leq 9$.

   Putting each constraint in a separate theory gives 54 theories. How to provide a step-by-step explanation of a solution given above theories such that each step is as small as possible? HINT: Use the above two exercises. Start by using a single theory. If that fails try combining two theories, etc.

10. Generalise previous exercise to any logical puzzle with a single solution.

## 2.1 Databases and logic

Queries originate from a database context. It is possible to represent a (relational) database into a vocabulary and structure. For example, consider the next database: From a logical perspective we can directly see this database as a structure $\mathcal{I}$ over a vocabulary $\Sigma$.

$$\Sigma = \{Instructor/2, Grade/3, Enrolled/2, Prerequ/2, PassingGrade/1, \\ Ray, Hec, \ldots, CS230, \ldots, AAA, \ldots, Jill, \ldots\}$$

The domain of $\mathcal{I}$ is

$$Domain_{\mathcal{I}} = \{Ray, Hec, \ldots, CS230, \ldots, AAA, \ldots, Jill, \ldots\}$$

The interpretation of $\mathcal{I}$ consists of

- The interpretation of all constants: $Ray^{\mathcal{I}} = Ray, \ldots$, $CS230^{\mathcal{I}} = CS230, \ldots, AAA^{\mathcal{I}} = AAA, \ldots$
- The relation of tuples of domain elements $Instructor^{\mathcal{I}}, Grade^{\mathcal{I}}, \ldots$ as in Figure 2.1.

Instructor

| | |
|---|---|
| Ray | CS230 |
| Hec | CS230 |
| Sue | M100 |
| Sue | M200 |
| Pat | CS238 |

Prerequ[a]

| | |
|---|---|
| CS230 | CS238 |
| CS148 | CS230 |
| M100 | M200 |

[a]Prerequ(a,b) means a is a prerequisite for b

Enrolled

| | |
|---|---|
| Jill | CS230 |
| Jack | CS230 |
| Sam | CS230 |
| Bill | CS230 |
| May | CS238 |
| Ann | CS238 |
| Tom | M100 |
| Ann | M100 |
| Jill | M200 |
| Sam | M200 |
| Flo | M200 |

Grade

| | | |
|---|---|---|
| Sam | CS148 | AAA |
| Bill | CS148 | D |
| Jill | CS148 | A |
| Jack | CS148 | C |
| Flo | CS230 | AA |
| May | CS230 | AA |
| Bill | CS230 | F |
| Ann | CS230 | C |
| Jill | M100 | B |
| Sam | M100 | AA |
| Flo | M100 | D |
| Flo | M100 | B |

PassGrade

| |
|---|
| AAA |
| AA |
| A |
| B |
| C |

Figure 1: The student database

**Exercise 2.** Formulate the next queries in $\Sigma$ en evaluate them in structure $\mathcal{I}$. All queries here are boolean: the answer is *true* or $false$. On Toledo you can find an example file in which you can fill the queries and use the IDP system to answer them.

1. *Is there a prerequisite for CS238?*
2. *Has May passed for CS230?*
3. *Are all the courses which are (direct) prerequisite for M100 instructed by Ray?*
4. *Has everyone that is enrolled in CS230 passed for at least one course instructed by Sue?*
5. *Did John do an exam for CS230?*
6. *Are there students taking a course from every instructor?*
7. *Is there are student who is enrolled in exactly 1 course?*

**Exercise 3** On Toledo you can find an example file in which you can write the queries and use the IDP system to answer them.
Construct the following queries:

1. *All tuples (x,y) such that student x has passed for the course y.*
2. *All students who follow exactly one course*
3. *All students who are enrolled in CS230 and have passed for all the courses they did up till now (a course has been done by a student if the student has a grade for it).*

**Exercise 4. (extra)** An essential part of the database are the integrity constraints. The database is subject to these constraints. These need to be checked when the database is changed and need to prevent the creation of invalid relations. Express the following integrity constraints in $\Sigma$ (on Toledo you can find an example file in which you can fill in the constraints and call the IDP system to test if they are satisfied in the given database).

1. No subject is a (direct) prerequisite for itself
2. Nobody can have more than one grade for a course.
3. No student can be an instructor. (A student is someone who is enrolled in a course)
4. Everyone who is enrolled for a course needs to have passed all courses which are (direct) needed prerequisites for that course.

# 3 UNA and DCA

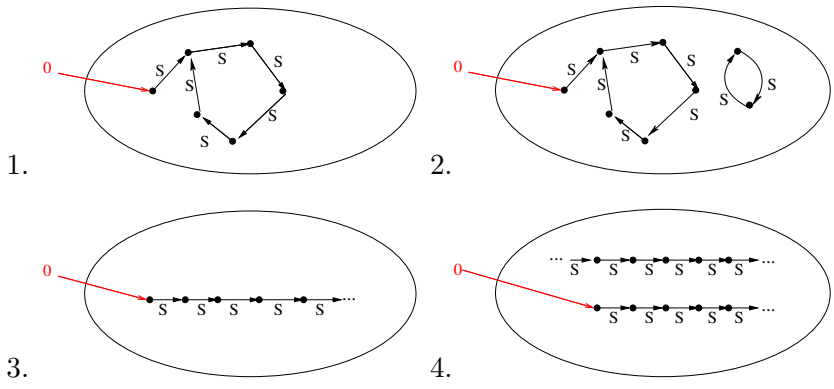
If the constants in the database vocabulary are not given with constructed types, there exist models in which for example $Ray = Jill$. This is of course not intended. Therefore, for correct modelling, two sort of axioms are needed: Unique Name Axiom (UNA) and Domain Closure Axiom (DCA).

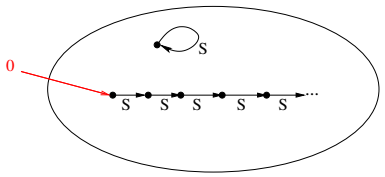
UNA is an axiom that expresses that different constants denote different objects. (see p. 58 of course notes)

So if we have a function $F/1 : 1$ and a constant $C/0 : 1$, then $F(a) \neq c$ for all domain elements $a$. UNA is very similar to the injectivity of a mathematical function.

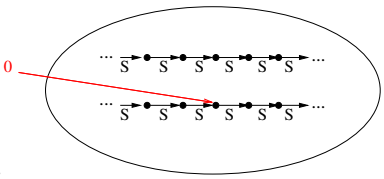
DCA intuitively: every domain element is equal to a term that is defined only by constants and function symbols in the vocabulary.

**Exercise 5.** In which of these images of {0, S}-structures is the domain closure axiom or the unique name axiom satisfied? (see course notes p. 90-91)






5.




6.

**Exercise 6.** Define UNA($\tau$) and DCA($\tau$) for $\tau$ = { Nil : *list*, Cons: A$\times$ *list* $\rightarrow$ *list* }. Here *list* is the type of lists over A.

**Exercise 7.** Define the UNA and DCA in IDP for a type consisting of the four wind directions (North, West, South, East). This means you can not use constructed types.