

## Assignment 11

due 6pm Thursday, December 2, 2021

This is the last assignment of the semester. It is due after Thanksgiving, on the last day of class. The material needed for problem 2 will be added to the textbook and repository by Tuesday, November 23.

These problems rely on the latest version of the github lamr repository, so be sure to use a new Gitpod image or update your repository following the instructions in the README. They require you to use an SMT solver (Z3, CVC4, or CVC5) and a first-order theorem prover (Vampire). The lamr repository contains Linux binaries in the folder `LAMR/bin`, but if you want to use the software on Windows or macOS, you need to replace these with the appropriate binaries, which you can find online.

### Problem 1

The almost square of order  $n$  is a rectangle of size  $n \times (n + 1)$ . The almost squares of orders 1 to 3 can fully cover the almost square of order 4. A solution is shown below.

```
1 1 3 3 3
2 2 3 3 3
2 2 3 3 3
2 2 3 3 3
```

In this exercise, we are going to use an SMT solver to determine whether the almost squares of order 1 to  $n$  can fully cover the almost square of order  $m$ . The encoding uses the theory `QF_LIA`. The encoding uses  $4n$  variables: for the almost square of order  $i$ , we use variables `xmin_i`, `xmax_i`, `ymin_i`, and `ymax_i`. The variable `xmin_i` (`xmax_i`) denotes the first (last, respectively) row in which the almost square of order  $i$  is placed. Similarly, the variable `ymin_i` (`ymax_i`) denotes the first (last, respectively) column in which the almost square of order  $i$  is placed.

The covering of the almost square of order 4 shown above can be expressed using the following assignment to these variables

- `xmin_1 = 1, xmax_1 = 2, ymin_1 = 4, ymax_1 = 4`
- `xmin_2 = 1, xmax_2 = 2, ymin_2 = 1, ymax_2 = 3`
- `xmin_3 = 3, xmax_3 = 5, ymin_3 = 1, ymax_3 = 4`

The code fragment below shows the first part of the encoding used to compute the covering. It shows the declaration of the first variables and the first constraints on those variables.

```
(set-logic QF_LIA)
(declare-const xmin_1 Int)
(declare-const xmax_1 Int)
(declare-const ymin_1 Int)
(declare-const ymax_1 Int)
...
(assert (and (>= xmin_1 1) (<= xmax_1 5)))
(assert (and (>= ymin_1 1) (<= ymax_1 4)))
...
```

Finish the encoding use the following steps:

**a) (3 points)**

Express constraints that ensure that the almost square of order  $i$  covers exactly a subgrid of  $n \times (n + 1)$  or  $(n + 1) \times n$ . The only variables that you can use are `xmin_i`, `xmax_i`, `ymin_i`, and `ymax_i`. Hint: Split the constraint into three parts with one part that enforces the relation between `xmin_i` and `xmax_i`, one part that enforces the relation between `ymin_i` and `ymax_i`, and one part that enforces the relation between all four variables. You will get better results if you express the last part as a conjunction of linear constraints, without using disjunction.

**b) (3 points)**

For each pair of almost squares, express the constraint that they cannot overlap each other, i.e., there is no cell that is covered by multiple almost squares.

**c) (3 points)**

Determine a grid assignment showing that the almost squares of orders 1 to 8 can fully cover the almost square of order 15. SMT solvers should be able to quickly solve the intended encoding. Print the grid assignment in a format similar to the grid assignment shown above.

**d) (3 points)**

Encode the same problem using the theory `QF_BV` and compare the runtimes between both theories. It is important to use signed bitvectors. The signed bitvector operations for `<`, `≤`, `>`, and `≥` are `bvslt`, `bvslsle`, `bvsgt`, and `bvsge`, respectively. Addition is `bvadd` and subtraction `bvsub`. The declaration of the variables and the initial bounds for these variables are shown below for  $n = 3$  and  $m = 4$ . Note that for bitvectors, CVC4/5 are slow, Z3 a bit faster and Boolector is fastest.

```
(set-logic QF_BV)
(declare-const xmin_1 (_ BitVec 16))
(declare-const xmax_1 (_ BitVec 16))
(declare-const ymin_1 (_ BitVec 16))
(declare-const ymax_1 (_ BitVec 16))
...
(assert (and (bvsge xmin_1 #x0001) (bvslsle xmin_1 #x0005)))
(assert (and (bvsge xmax_1 #x0001) (bvslsle xmax_1 #x0005)))
(assert (and (bvsge ymin_1 #x0001) (bvslsle ymin_1 #x0004)))
(assert (and (bvsge ymax_1 #x0001) (bvslsle ymax_1 #x0004)))
...
```

**e) Bonus point!**

The same encoding can also be used to cover the almost square of order 55 with the almost squares of order 1 to 20. Solving this formula can take minutes. Give it a try.

**Problem 2 (6 points)**

Recall from the textbook Smullyan's asylum puzzles, where every inhabitant is a patient or a doctor, and every inhabitant is either sane or insane. Everything a sane inhabitant believes is true, and everything an insane inhabitant believes is false. So " $x$  believes  $P$ " can be represented as  $\text{Sane}(x) \leftrightarrow P$ .

Smullyan's final asylum puzzle runs as follows:

The last asylum Craig visited he found to be the most bizarre of all. This asylum was run by two doctors named Doctor Tarr and Professor Fether. There were other doctors on the staff as well. Now, an inhabitant was called *peculiar* if he believed that he was a patient. An inhabitant was called *special* if all patients believed he was peculiar and no doctor believed he was peculiar. Inspector Craig found out that at least one inhabitant was sane and that the following condition held:

*Condition C:* Each inhabitant had a best friend in the asylum. Moreover, given any two inhabitants,  $A$  and  $B$ , if  $A$  believed that  $B$  was special, then  $A$ 's best friend believed that  $B$  was a patient.

Shortly after this discovery, Inspector Craig had private interviews with Doctor Tarr and Professor Fether. Here is the interview with Doctor Tarr:

*Craig:* Tell me, Doctor Tarr, are all the doctors in this asylum sane?

*Tarr:* Of course they are!

*Craig:* What about the patients? Are they all insane?

*Tarr:* At least one of them is.

The second answer struck Craig as a surprisingly modest claim! Of course, if all the patients are insane, then it certainly is true that at least one is. But why was Doctor Tarr being so cautious? Craig then had his interview with Professor Fether, which went as follows:

*Craig:* Doctor Tarr said that at least one patient here is insane. Surely that is true, isn't it?

*Professor Fether:* Of course it is true! All the patients in this asylum are insane! What kind of asylum do you think we are running?

*Craig:* What about Doctor Tarr? Is he sane?

*Professor Fether:* Of course he is! How dare you ask me such a question?

At this point, Craig realized the full horror of the situation! What was it?

Smullyan's solution shows that, under these hypotheses, all the patients in the asylum are sane and all the doctors are insane. Using Vampire, however, we were able to prove something stronger, namely, that the hypotheses themselves are inconsistent. In other words, no such asylum can possibly exist!

Try doing the same. You can start with these hypotheses:

```
def last_asylum_hypotheses : List FOFoForm := [
  fo!{Doctor(Tarr)},
  fo!{Doctor(Fether)},
  fo!{ $\forall x. \text{Peculiar}(x) \leftrightarrow (\text{Sane}(x) \leftrightarrow \neg \text{Doctor}(x))$ },
  fo!{ $\forall x. \text{Special}(x) \leftrightarrow \forall y. \neg \text{Doctor}(y) \leftrightarrow (\text{Sane}(y) \leftrightarrow \text{Peculiar}(x))$ },
  fo!{ $\exists x. \text{Sane}(x)$ },
  ...
]
```

You can also introduce a function `bf(x)` to denote the best friend of  $x$ , and use constants `Tarr` and `Fether` for those two characters.

(This puzzle is an homage to a story by Edgar Allen Poe called "The System of Doctor Tarr and Professor Fether.")