Introduction

In this mini-project , your task is to model different problems and solve them using Z3[1]. An introduction can be found in `https://www.philipzucker.com/z3-rise4fun/guide.html`, and you can find additional material at `http://theory.stanford.edu/~nikolaj/programmingz3.html`. To solve the exercises, you have to create a text file in Z3's input language (based on SMT-lib; Z3 also has an interface with several popular programming languages but we will use the declarative SMT-lib language), which you can then pass as command line argument to the Z3 executable.
The Z3 executable is publicly available and it is recomended to download and install it before the lecture. On Ubuntu/Debian, you can simply use "apt install z3". On Windows you can download the release exe file fromn the official repository. Otherwise, you can also use the online solver at `https://compsys-tools.ens-lyon.fr/z3/`.
The result of Z3 (`sat` or `unsat`) will be printed to the console. Please, use the `-st` option, in order to print additional information about the solving process, e.g., `z3 -st filename.z3`
Table 1 shows an overview of the Z3 language fragments that are relevant for this sheet.

Note: If you write contradictory axioms, anything becomes provable. To test that this is not the case, try to prove something false (like formula(false)) and see whether the solver also returns Proof found. If that is the case, there must be a contradiction somewhere in your premises.

Submission: Master solutions won't be provided for the mini-project. We have enabled a submission in Moodle, where you can submit your `.z3` files for feedback. Also, feel free to send any questions by email (`alto@cs.aau.dk`).

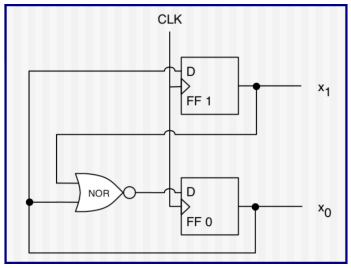
[1] `https://github.com/Z3Prover/z3`

Statement	Description
General	
`(check-sat)`	Checks whether the predicate logic problem defined up to this point is satisfiable.
`(assert E)`	Adds the boolean / predicate logic expression `E` to the list of formulas to be verified in `check-sat`.
`(get-value (E))`	Prints the value of `E`, where `E` can be an arbitrary expression such as constant, propositions, or boolean combination thereof (must occur after `(check-sat)`).
`(get-model)`	Prints all variable assignments (must occur after `(check-sat)`).
`(echo "message")`	Prints `message` to the console.
`; This is a comment`	Commenting.
Boolean Expressions	
`true`	Constant for true.
`false`	Constant for false.
`(declare-const p Bool)`	Declares a new proposition with name `p`.
`p`	Evaluates to the truth assignment of `p`
`(not E)`	Evaluates to the negation of `E`.
`(and` E_1 `...` E_n`)`	Conjunction over the expressions E_1 to E_n.
`(or` E_1 `...` E_n`)`	Disjunction over the expressions E_1 to E_n.
`(=` E_1 `...` E_n`)`	Is true if and only if the expressions E_1 to E_n evaluate to the same value. Can be used e.g. to compare variables, or to compute the equivalence over boolean expresions.
`(=>` E E'`)`	Implication, is true if E implies E'.
`(distinct` E_1 `...` E_n`)`	Is true iff every expresion E_1 to E_n evaluates to a different value.
Predicate Logic	
`(declare-fun P (`$T_1 \dots T_n$`) Bool)`	Declares an n-ary predicate with name `P` and parameter types T_1 to T_n.
`(P` t_1 `...` t_n`)`	Evaluates to the value of the n-ary predicate `P` with arguments t_1 to t_n.
`(forall ((`x_1 T_1`) ... (`x_n T_n`)) (E))`	All quantification over n variables: x_1 with type T_1, ..., x_n with type T_n. $x_1, \dots, x_n$ can be used in the expression `E`.
`(exists ((`x_1 T_1`) ... (`x_n T_n`)) (E))`	Existential quantification over n variables: x_1 with type T_1, ..., x_n with type T_n. $x_1, \dots, x_n$ can be used in the expression `E`.
Mathematical Expressions	
`(declare-const var Int)`	Defines integer variable `var`.
`var`	Evaluates to the value of variable `var`.
`(+` E_1 `...` E_n`)`	Evaluates to $\sum_{i=1}^{n} E_i$.
`(*` E_1 `...` E_n`)`	Evaluates to $\prod_{i=1}^{n} E_i$.
`(=` E E'`)`	Is true iff E and E' evaluate to the same number.
`(<=` E E'`)`	Is true iff $E \leq E'$.

Table 1: Z3 input language fragment you may use for the predicate logic modeling exercises. You may use the data types `Int`, `Bool`, or the ones specified in the template files we provide.

Exercise 1 Circuit Verification (SAT)

Encode the example in the slides of Chapter 7 as a logical formula.



- Counter, repeatedly from $c = 0$ to $c = 2$.
- 2 bits x_1 and x_0; $c = 2 * x_1 + x_0$.
- ("FF" Flip-Flop, "D" Data IN, "CLK" Clock)

→ The circuit simply repeats the operations: $x_0 \leftarrow NOR(x_1, x_2) = \neg(x_1 \vee x_2)$; and $x_1 \leftarrow x_0$. The clock is there so that both operations happen simultaneusly, so x_0 and x_1 are updated at the same time.

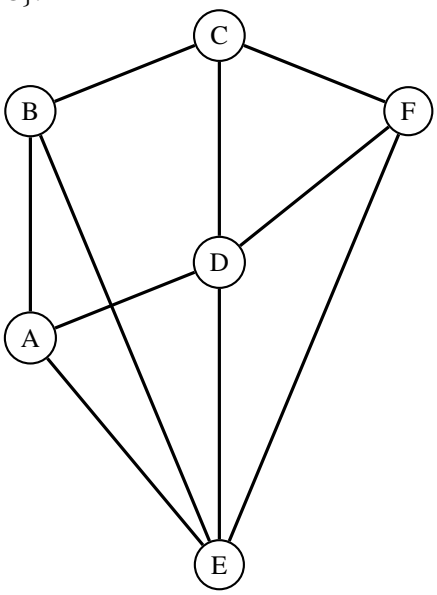
Then, use Z3 to verify if the following two properties hold:

1. If $c < 3$ in current clock cycle, then $c < 3$ in next clock cycle.
2. If $c < 2$ in current clock cycle, then $c < 2$ in next clock cycle.

Otherwise, ask Z3 to provide a counter example, i.e., an assignment to x_0, x_1, x'_0, x'_1 that violates the property.

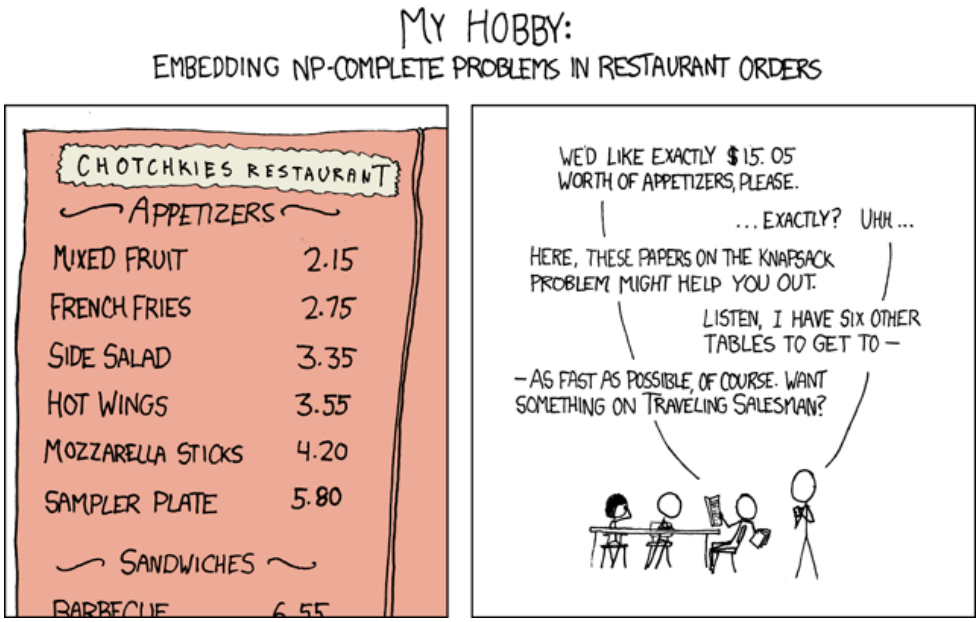
Exercise 2 Surveillance

Given a graph, select a minimal set of nodes that need to be surveilled, so that every node is surveilled or is connected to a surveilled node. For example, encoding the following graph can be solved by choosing {A, C}.



Exercise 3 XKCD (Arithmetic Theories)

Consider the problem suggested in the following comic strip (`https://xkcd.com/287/`):



1. Encode the exact question in the strip. What combination of appetizers satisfies the constraint?
2. Modify your encoding to find a different combination that satisfies the constraint.

Exercise 4 Gardens (Predicate Logic)

There are four gardens that cultivate different kinds of flowers (rose, tulip, and lily), vegetables (onion, carrot, and pepper), and fruits (apple, banana, and cherry).
Given that we know the facts given in the list below, use Z3 to prove that there must be (a) a lily in garden1, (b) a fruit in gardens 2 and 3 (the same kind of fruit in both), and (c) tulips and roses are in the same garden. Formulate each of the facts (1.-10.) and the statements to be proved (a–c) as a separate statement in Z3. Write all statements in general form (i.e. without referring to particular objects), unless the description refers to particular objects.

1. Any plant is either a fruit, a flower, or a vegetable.
2. Every plant is in at least one garden.
3. There is no garden that grows both rose and carrots.
4. Garden2 has no flowers.
5. Gardens 1 and 4 have carrots and Garden3 has bananas.
6. Any garden with tulip has another flower.
7. Garden1 has one fruit, one vegetable and one flower.
8. Everybody grows exactly 3 different plants.
9. Exactly one garden has all 3 kinds of fruits.
10. Exactly 3 plants are in 2 or more gardens and they are one vegetable and two fruits. All others plants are in a single garden.

Complete the provided template (`template-garden.z3`) in which some functions and predicates have already been declared. You are allowed to define more functions or predicates if you need it to complete the solution. The template also includes some constraints that must be specified, because false statements must be explicitly negated. For example, when specifying which objects are fruits, vegetables, etc, it must also be indicated that all other objects are not fruits (we do this via $=$).

Exercise 5 Free Modelling (Optional)

Model another problem of your choice!
It is likely that some sub-problem in your current (or perhaps some past) semester project, can be modelled and solved using Z3. Feel free to use this opportunity to try out how this technology can help and feel free to ask for advice and/or feedback on how to do that.