SAARLAND UNIVERSITY ARTIFICIAL INTELLIGENCE

Prof. Dr. Jörg Hoffmann, Dr. Daniel Fišer, Daniel Höller, Dr. Sophia Saller

Practical Exercise Sheet 3

Deadline Friday, July 24, 23:59

About the submission of this sheet.

- You might submit the solutions to exercises in groups of up to 3 students.
- All students of a group need to be in the same tutorial.
- Hand in the solution **in CMS** and using "Team Groupings".
- Solutions need to be packaged into a `.zip` file that contains a single folder with name: `AI2022_PE3_mat1_mat2_mat3`, where `mat1, mat2, mat3` are the matriculation numbers of the students who submit together.
- This folder must contain the following files:
  - `authors.txt` listing names and matriculation numbers of all students of your group. Use one line per student and no spaces: Name;Matriculation number.
  - The Z3 files containing your solutions. **Do not rename** the Z3 template files.
  - Your `sudoku.log` file with the results.
- Do not add any other folder or sub folder, this means place all files directly into `AI2022_PE3_mat1_mat2_mat3`. Do not place any file outside of this folder.

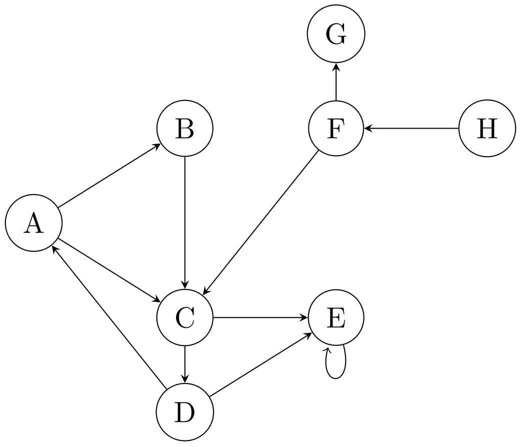
This exercise sheet is accompanied with several source files, which we provide through a separate `zip` archive. You can download this archive from the CMS under the Materials category.

In this exercise, your task is to model different problems in predicate logic, and to solve those models using `Z3`. Please refer to the previous exercise sheet for an introduction and installation instructions. Table 1 shows an overview of the `Z3` language fragments that are relevant for this sheet. **You must not use any statement that is not included in this table**.

Exercise 1: Graphs. (6 Points)

In this exercise, we want to verify basic properties of the directed graph below:
To model the structure and the properties of the graph, we use the following predicates:

- Edge(x, y) holds if there is a (directed) edge from x to y
- Reachable(x, y) holds if there is a path from x to y, i.e. y is reachable from x
- Isolated(x) holds if x is an isolated node, i.e. x has no incoming or outgoing edges



Solve the following tasks in Z3. Start with the provided template file (`exercise01.z3`). Do not use additional predicates or sorts.

1. Fully specify the graph given above by providing the definition of the Edge relation.
2. Define the following axiom.
   - $\forall x\,[\text{Reachable}(x, x)]$ – Every node can reach itself.
3. Define the following axiom.
   - $\forall x\,\forall y\,\forall z\,[\text{Edge}(x, y) \land \text{Reachable}(y, z) \Rightarrow \text{Reachable}(x, z)]$ – If there is an edge from x to y and y can reach z then x can reach z.
4. Define the following axiom.
   - $\forall x\ \ \text{Isolated}(x) \Leftrightarrow (\neg\exists y\ \ [\text{Edge}(x, y) \lor \text{Edge}(y, x)])$ – A node is isolated if and only if there is no incoming or outgoing edge
5. Show that vertex E is reachable from vertex A and that no vertex in the graph is isolated.

Tip: Introduce several test cases (e.g. *(assert (Reachable A A))*) and see what is the result for your model. These will not be part of your assignment, they are just for you to see if everything is fine with the model. Note that the model is not able to derive if something is not reachable, as the axioms are not strong enough to prove the absence of a property.

Exercise 2. (14 Points)

There are four stores which sell three types of items: tools (screwdrivers, saws and hammers), snacks (chips, sweets and cookies) as well as beverages (water, juice, coke).
Given that we know the facts given in the list below, use Z3 to prove that (a) Store 1 sells coke, (b) there is (at least) one snack sold by **both** Store 2 and 3, and (c) (at least) one store sells both juice and water. Formulate each of the facts (1.-10.) and the statements to be proven (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. The `distinct` statement may be useful in this exercise.

1. Any item is either a tool, a snack, or a beverage.
2. Store 1 and 4 sell saws and Store 3 sells sweets.
3. Every item is sold by at least one store.
4. Exactly one store has all 3 types of snacks.
5. There is no store that sells both water and saws.
6. Any store that sells juice also sells another beverage.
7. Store 1 sells (at least) one tool, one snack and one beverage.
8. Store 2 has no beverages to offer.
9. Every store sells exactly 3 different items.
10. Exactly 3 items are sold by 2 or more (possibly different) stores and they are one tool and two snacks. All others items are sold by a single store.

We provide a stub (`exercise02.z3`) that you must complete, 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 stub also includes some constraints that must be specified, because false statements must be explicitly negated. For example, when specifying which objects are tools, snacks, etc, it must also be indicated that all other objects are not tools (we do this via =).

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

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.
Mathematical Expressions	
`(declare-const var Int)`	Defines integer variable `var`.
`var`	Evaluates to the value of variable `var`.
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 expressions.
`(=>` E E'`)`	Implication, is true if E implies E'.
`(distinct` E_1 `...` E_n`)`	Is true iff every expression E_1 to E_n evaluates to a different value.
Predicate Logic	
`(declare-fun P (`$T_1 \ldots T_n$`) 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. $x_1, \ldots, x_n$ 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. $x_1, \ldots, x_n$ can be used in the expression `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.