Symbolic Model Checking with Z3

Problem Symbolic Model Checking. Consider the transition system $\mathcal{M} = (S, R, I)$:

- $S = \{0, 1, 2, 3\}$ is the set of states,
- $I = \{0\}$ is the set of initial states,
- $R = \{(0, 1), (0, 2), (1, 3), (2, 3), (3, 3)\}$ defines the transition relation,
- $AP = \{p\}$ is the set of atomic propositions, and
- L is the labeling function with $L(0) = L(1) = L(2) = \emptyset$, and $L(3) = \{p\}$.

Let's use two boolean variables x and y to encode the states of this transition system.

- State 0: $\neg x \wedge \neg y$
- State 1: $x \wedge \neg y$
- State 2: $\neg x \wedge y$
- State 3: $x \wedge y$

Complete the following tasks:

i. Draw the transition diagram for $\mathcal{M}$.

ii. Write a Boolean logic formula that represents the set of initial states, I.

iii. Write a Boolean logic formula for the set of states that correspond to property p.

iv. Write a Boolean logic formula that represents the transition relation, R.

v. Draw the BDD for the transition relation R. Use variable ordering x, x', y, y'.

vi. Compute $EX\ p$ using the boolean encoding described above by writing a python-Z3 program that encodes the logical definition of $EX\ p$. Take a look at the provided example code from the lecture for inspiration.

vii. What is the resulting expression returned by running Z3 on your query?

viii. The result of your query should be an expression that corresponds to the set of states satisfying $EX\ p$. Which states (i.e. state numbers) from the original transtion system does this expression encode? Does this match the answer you would get if you performed the explicit EX algorithm for CTL model checking that we learned earlier in the semester? (Hint: it should!)

ix. **Bonus Exploration**. We showed how all of the other CTL operations can be expressed recursively using EX. For example $EG\ p \equiv p \wedge EX\ EG\ p$. In the slides, we showed (via example only) that repeatedly applying EX and simplifying in the right way would converge to $EG\ \phi$ (or whatver other operator we were computing). Can you write a python function for EG, EF and possibly other operators that just uses a function that computes EX?