Let us consider the following abstract syntax model of a Java-like programming language with exceptions.

abstract sig Instruction {}
sig Block extends Instruction {
    stmts: set Instruction
}
sig TryCatch extends Instruction {
    tryBlock: Instruction,
    catchBlock: Instruction
}

// represents a throw new Exception()
sig Throw extends Instruction {}

1/ Can a catch block throw an exception? Explain why.

2/ What does "set" mean in the context of Block?

3/ Can one have instances of the signature Instruction? Why?

4/ Not all instances of this model represent a valid AST. Represent graphically an invalid AST that Alloy could have generated.

5/ Give two missing constraints in natural language that participate to representing valid ASTs.

6/ Translate those two missing constraints in Alloy.

7/ Explain in natural language what the following fact means.

fact {
    all i: Instruction | lone b: Block | i in b.stmts
}

8/ We now want to identify the instructions that throw exceptions for sure.

one sig State {
    // contains all instructions
    throws: set Instruction
}

The following specifies what instructions must be in State.throws:
- all throw statements throw an exception
- all blocks containing an instruction that throws an exception throw themselves an exception
- try catch blocks throw an exception if and only if the try block and the catch blocks throw an exception
Translate in Alloy those facts (we remind you that ran[throws] returns the set of elements tagged as throwing an exception)

9/ Write a predicate called “atLeastOneNoThrowsAndOneThrows” to find a code example with one block throwing an exception and one block NOT throwing an exception. The predicate will be run as follows

```
run atLeastOneNoThrowsAndOneThrows for 3
```

10/ Let us now model a source code transformation in Alloy as follows. Explain in natural language the content of this transformation.

```
pred transf_post_condition[]
{
  all t : Throw | one try_added:TryCatch | t in try_added.tryBlock
}
```