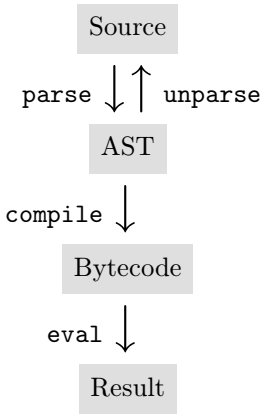
Computer Systems Modeling and Verification
(USEEN1)

Part I: Python `ast` module

Abstract Syntax Trees (AST) are tree representations of source codes. They are used in every compiler and bytecode interpreter (or virtual machine), as summarized in the following picture:



Recall that the *parse* function converts the source code into a tree structure, and that *unparse* is its inverse function.

The Python AST, together with the `parse`, `unparse` and `dump` functions, are defined in module `ast`[1] from the standard library (while the `compile` and `eval` functions are always available). More information about this module can also be found in *the missing Python AST docs*[2].

1 Arithmetic expressions without variables

```
>>> import ast
>>> e = ast.parse("(3␣+␣(6␣*␣5))␣*␣(2␣+␣7)", mode='eval')
>>> ast.dump(e, indent=4)
```

```
Expression(
    body=BinOp(
        left=BinOp(
            left=Constant(value=3),
            op=Add(),
            right=BinOp(
                left=Constant(value=6),
                op=Mult(),
                right=Constant(value=5))),
        op=Mult(),
        right=BinOp(
            left=Constant(value=2),
            op=Add(),
            right=Constant(value=7))))
```

```
>>> print(ast.unparse(e))
```

```
(3 + 6 * 5) * (2 + 7)
```

1. https://docs.python.org/3/library/ast.html
2. https://greentreesnakes.readthedocs.io

```
>>> bc = compile(e, filename='<string>', mode='eval')
>>> eval(bc)
   297
```

Remarks (*excerpt from the official documentation*)[3]

- The *filename* argument should give the file from which the code was read; pass some recognizable value if it wasn't read from a file (`'<string>'` is commonly used).
- The *mode* argument specifies what kind of code must be compiled; it can be `'exec'` if *source* consists of a sequence of statements, `'eval'` if it consists of a single expression, or `'single'` if it consists of a single interactive statement (in the latter case, expression statements that evaluate to something other than `None` will be printed).

2 Arithmetic expressions with variables

```
>>> e = ast.parse("(3 + (x * 5)) * (y + 7)", mode='eval')
>>> ast.dump(e, indent=4)
   Expression(
       body=BinOp(
           left=BinOp(
               left=Constant(value=3),
               op=Add(),
               right=BinOp(
                   left=Name(id='x', ctx=Load()),
                   op=Mult(),
                   right=Constant(value=5))),
           op=Mult(),
           right=BinOp(
               left=Name(id='y', ctx=Load()),
               op=Add(),
               right=Constant(value=7))))
>>> print(ast.unparse(e))
   (3 + x * 5) * (y + 7)
>>> bc = compile(e, filename='<string>', mode='eval')
>>> eval(bc, {'x': 5, 'y': 9})
   448
```

Remarks

- The context (field `ctx`) of a `Name` (an occurence of a variable) is always `Load` in the AST of an expression, so you can just ignore this field for the moment.
- Function `eval` can also invoked directly on strings or files (but not on an AST). Of course, in that case, the `parse` and `compile` functions are called implicitely. More details can be found in the documentation or using the interactive help command.

Exercises

1. Define a function `eval_expr(e: expr) -> int` which computes an `expr` and returns its integer value (this is the same function as in the previous assignment).

3. https://docs.python.org/3/library/functions.html#compile

2. Test function `eval_expr` on actual Python expressions using `ast.parse` and the following helper function:

```
def eval(e: Expression, m: Mapping[str, int] = {}) -> int:
    return eval_expr(e.body, m)
```

3 Locating expressions and statements

When the evaluation of an expression (or the execution of a program) fails, it is convenient to be able to locate the origin of the failure in the source code. For that purpose, the location of each statement and each expression is recorded in the AST.

A simple module `error` is provided with the code. This module contains the definition of the type of a `region`:

```
type region = tuple[int, int, int | None, int | None]
```

The first pair of integers corresponds to the beginning of the region: the first integer is the line number in the file, and the second integer is the column number within the line. Similarly, the second pair (which is optional) corresponds to the end of the region.

A helper function (called `fail`) can be used to display the region of some node from the AST, followed by the error message corresponding to a given exception and then exit. An example of usage of this function is also provided.