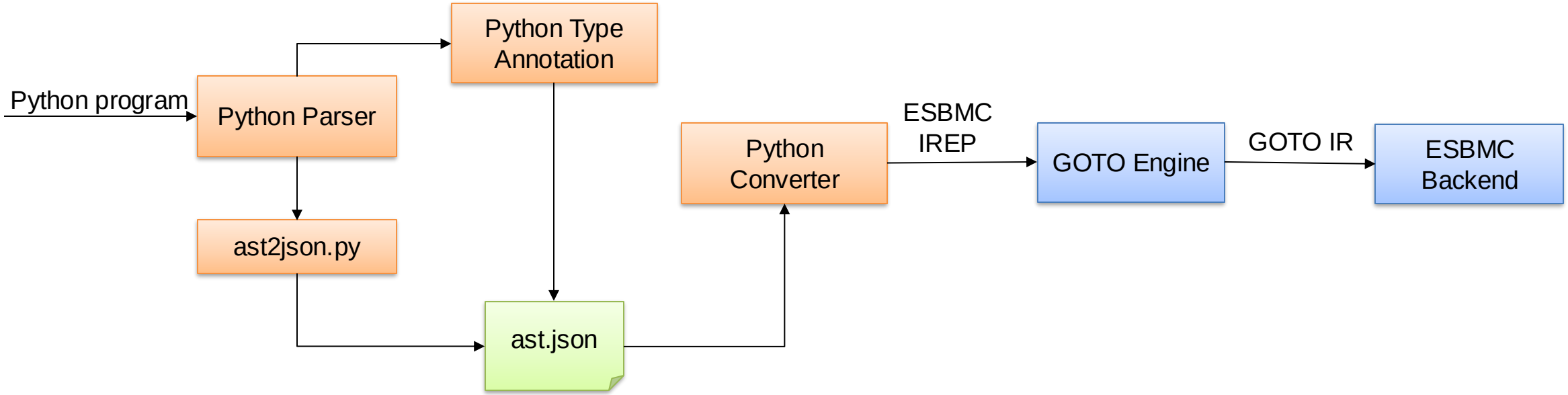
MANCHESTER 1824
The University of Manchester

Python Model Checking

Bruno Farias, Youcheng Sun, and Lucas C. Cordeiro
{bruno.farias, youcheng.sun, lucas.cordeiro}@manchester.ac.uk
University of Manchester
6th February 2024

MANCHESTER 1824
The University of Manchester

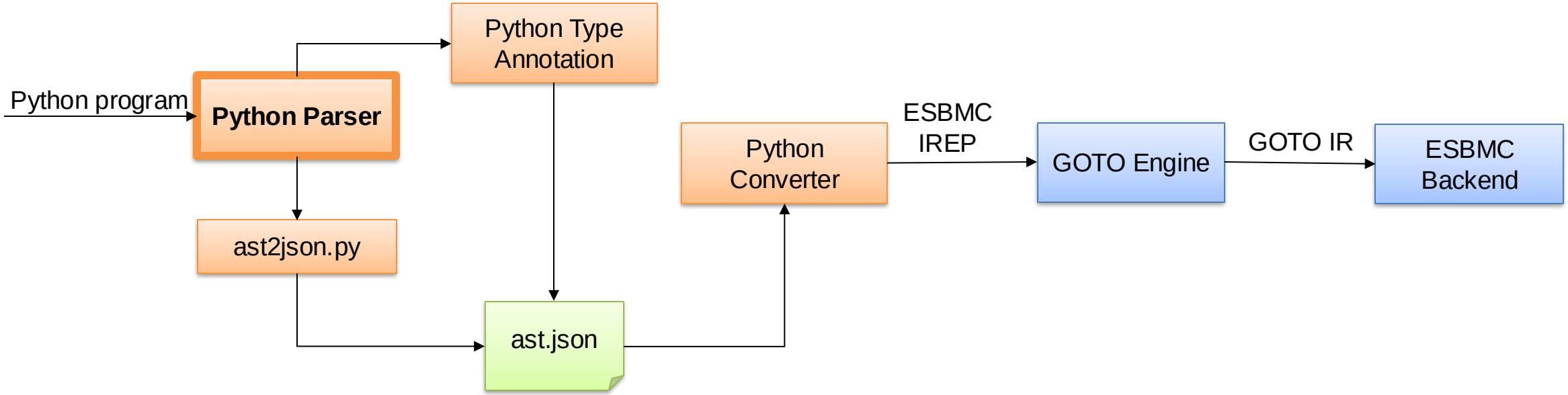
ESBMC Python Frontend



- Translate Python **features** and **behaviour** into ESBMC intermediation representation model
- Generate and export an Abstract Syntax Tree (AST) in JSON format
- Add **type annotation** within function bodies

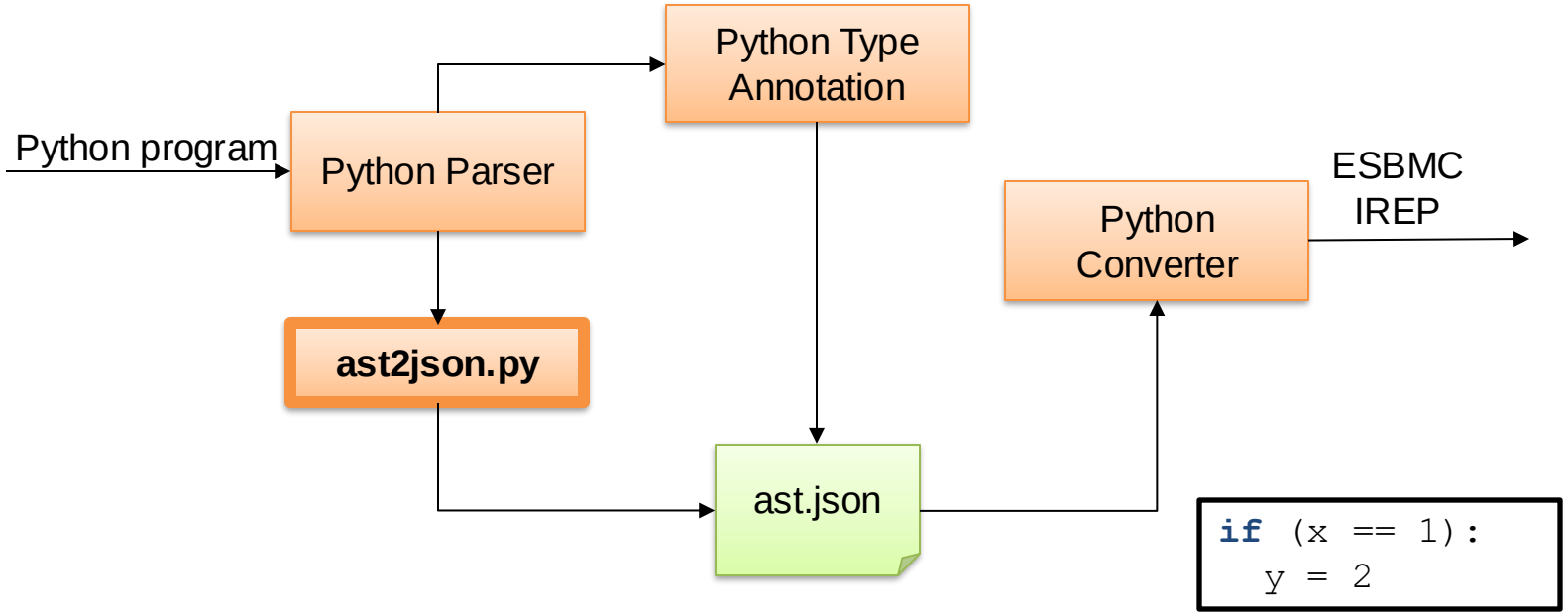
MANCHESTER 1824
The University of Manchester

ESBMC Python Frontend



- Creates a child process for ast2json.py execution
- Dump ast2json.py content from ESBMC binary
- Manage flags for AST annotation and print

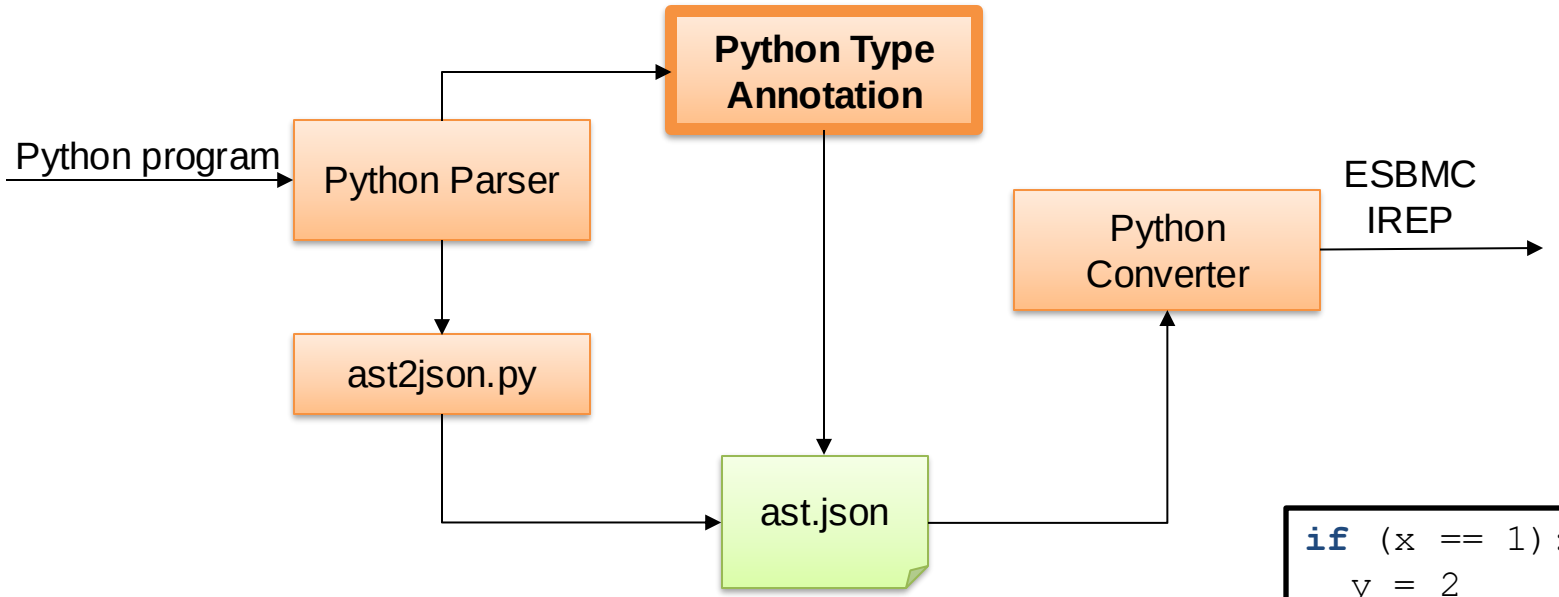
ESBMC Python Frontend




```
{
    "_type": "If",
    "body": [
        {
            "_type": "Assign",
            "targets": [
                {
                    "_type": "Name",
                    "id": "y",
                    "lineno": 5
                }
            ],
            "value": {
                "_type": "Constant",
                "value": 2
            }
        }
    ],
    "orelse": [],
    "test": {
        "_type": "Compare",
        "comparators": [
            {
                "_type": "Constant",
                "value": 1
            }
        ],
        "left": {
            "_type": "Name",
            "id": "x",
            "lineno": 4
        },
        "ops": [
            {
                "_type": "Eq"
            }
        ]
    }
}
```


- Manages python package dependencies
- Generates an AST from Python program
- Dump a JSON file with AST content

ESBMC Python Frontend




```
if (x == 1):
  y = 2
```

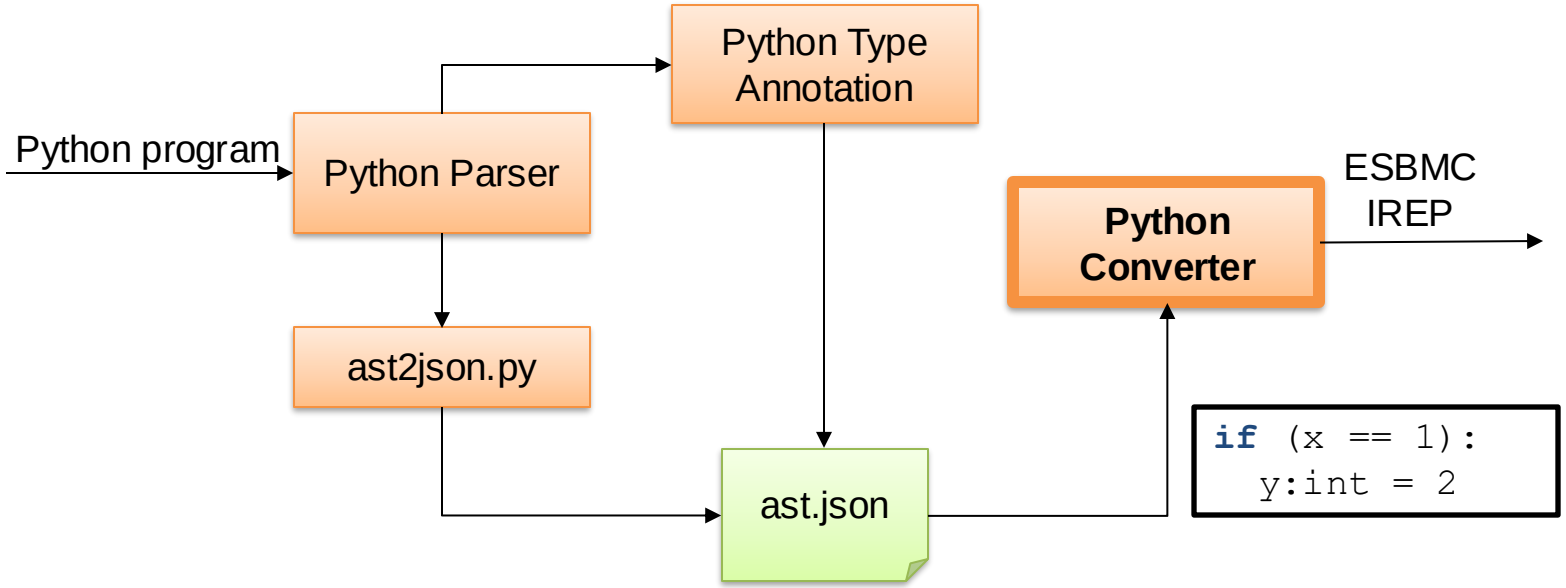
```
if (x == 1):
  y:int = 2
```

- Parse JSON generated by ast2json.py
- Add type annotation for different types of variable assignments


```
{
    "_type": "AnnAssign",
    "annotation": {
        "_type": "Name",
        "col_offset": 2,
        "ctx": {
            "_type": "Load"
        },
        "end_col_offset": 5,
        "end_lineno": 2,
        "id": "int",
        "lineno": 2
    },
    "col_offset": 0,
    "end_col_offset": 9,
    "end_lineno": 2,
    "lineno": 2,
    "simple": 1,
    "target": {
        "_type": "Name",
        "col_offset": 0,
        "ctx": {
            "_type": "Store"
        },
        "end_col_offset": 1,
        "end_lineno": 2,
        "id": "y",
        "lineno": 2
    },
    "value": {
        "_type": "Constant",
        "value": 2
    }
}
```

MANCHESTER 1824
The University of Manchester

ESBMC Python Frontend



- Read **AST** with type annotation
- Generate **ESBMC IREP** for all program statements
- Build a **symbols table** with all declared variables and defined functions

```
code
  * type: code
  * operands:
    0: =
      * type: bool
      * operands:
        0: symbol
          * type: signedbv
            * width: 32
          * name: x
          * identifier: py:test.py@x
        1: constant
          * type: signedbv
            * width: 32
          * value: 00000000000000000000000000000001
      * #location:
        * file: test.py
        * line: 4
        * function:
        * column: 4
    1: code
      * type: code
      * operands:
        0: code
          * type: code
          * operands:
            0: symbol
              * type: signedbv
                * width: 32
              * name: y
              * identifier: py:test.py@y
            1: constant
              * type: signedbv
                * width: 32
              * value: 00000000000000000000000000000010
          * statement: assign
          * #location:
      * statement: block
      * #location:
        * file: test.py
        * line: 4
        * function:
        * column: 0
  * statement: ifthenelse
```

MANCHESTER 1824
The University of Manchester

ESBMC Python Frontend

Supported features

- Boolean operations (and, or, not)
- Bitwise operations
- Conditional statements (if/ while)
- Assignent statements
- Assert statement
- Pass statement
- Function definitions
- Recursive functions
- Class definitions
- Class and Instance attributes
- Non-determinism

```
1   def add(x:int, y:int) -> int:
2     return x+y
3
4   class MyClass:
5     def __init__(self, x:int):
6       self.data:int = x
7
8   a = True
9   b = False
10
11  assert a == True and b == False
12
13  c = 2
14  d = c << 1
15  assert d == 4
16
17  assert add(c, d) == 6
18
19  obj = MyClass(5)
20  assert obj.data == 5
```

MANCHESTER 1824
The University of Manchester

ESBMC Python

Verification Demo

MANCHESTER 1824
The University of Manchester

Conditional Statements

Python program

```
x = 1
y = 2
result = 0

if (x == 1 and y == 2):
  result = 1

assert result == 1
```

ESBMC IRep

Function Definitions

Python program

```
def add(a:int, b:int) -> int:
    return a+b
```

ESBMC IRep

```
Symbol......: py:test.py@F@add
Base name...: add
Module......: test
Mode........: Python (Python)
Type........: code
      * arguments:
        0: argument
            * type: signedbv
              * width: 32
        1: argument
            * type: signedbv
              * width: 32
      * return_type: signedbv
        * width: 32
Value.......: code
      * type: code
      * operands:
        0: code
            * type: code
            * operands:
              0: +
                  * type: signedbv
                    * width: 32
                  * operands:
                    0: symbol
                        * type: signedbv
                          * width: 32
                        * name: a
                        * identifier: py:test.py@F@add@a
                    1: symbol
                        * type: signedbv
                          * width: 32
                        * name: b
                        * identifier: py:test.py@F@add@b
            * statement: return
            * #location:
              * file: test.py
              * line: 2
              * function: add
              * column: 2
      * statement: block
```

MANCHESTER 1824
The University of Manchester

Class Definitions

Python program

```
class MyClass:
  def __init__(self, x:int):
    self.data:int = x
```

```
class MyClass {
  MyClass(MyClass* self, int);
}
```

ESBMC IRep

```
Symbol......: tag-MyClass
Base name...: MyClass
Module......: test.py
Mode........: Python (Python)
Type........: struct
      * tag: MyClass
      * components:
        0: component
            * type: signedbv
                * width: 32
                * #member_name: tag-MyClass
            * name: data
            * access: public
            * pretty_name: data
      * methods:
        0: component
            * type: code
                * arguments:
                  0: argument
                      * type: pointer
                          * subtype: symbol
                              * identifier: tag-MyClass
                      * #location:
                        * file: test.py
                        * line: 2
                        * function: MyClass
                        * column: 17
                      * #base_name: self
                      * #identifier: py:test.py@C@MyClass@F@MyClass@self
                  1: argument
                      * type: signedbv
                          * width: 32
                      * #location:
                        * file: test.py
                        * line: 2
                        * function: MyClass
                        * column: 23
                      * #base_name: x
                      * #identifier: py:test.py@C@MyClass@F@MyClass@x
                * return_type: constructor
            * name: MyClass
Flags.......: type
Location....: file test.py line 1 column 0
```

Non-determinism

- **nondet_X()** functions with X in {bool, int, float} allows to simulate non-deterministic values
- Initialize variables with all possible values

Successful

```
x: int = nondet_int()
y: int = x

if (nondet_bool()):
    x = x + 1
else:
    x = x + 2

assert(x != y and x == y+1 or x == y+2)
```

Fail

```
def div(num:int, den:int) -> int:
    return num/den

x:int = nondet_int()
y:int = nondet_int()

div(x, y)
```

```
[Counterexample]


State 1 file main.py line 5 column 0 thread 0
------------------------------------------------------
  y = 0 (00000000 00000000 00000000 00000000)

State 2 file main.py line 2 column 4 function div thread 0
------------------------------------------------------
Violated property:
  file main.py line 2 column 4 function div
  division by zero
  den != 0


VERIFICATION FAILED
```

MANCHESTER 1824
The University of Manchester

ESBMC Python Frontend

Type hints

- All functions signatures must be pre-annotated

```
def saturating_sub(a: int, b: int) -> int:
    return a - b if a > b else 0
```

- Variables type annotation
  - Constant values: x = 10
  - Referred variables: y = x
  - Class instances: obj = MyClass()

MANCHESTER 1824
The University of Manchester

ESBMC Python Frontend

Type annotation

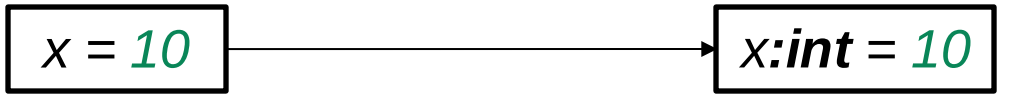
- JSON defines value types
  - Integer: {“age”: 27}
  - Fraction: {“size”: 120.5}
  - Exponent of 10: {“distance”: 3.7e+23}
  - String: {“name”: “John”}
  - Boolean: {“isCold”: true}
  - Array: {“colors”: [“red”, “blue”, “green”]}
  - Null

- Add JSON nodes with type annotation (compatible with **ast2json)** from JSON value information
- ast and ast2json are standard libraries

ESBMC Python Frontend

Type annotation

- Constant Values



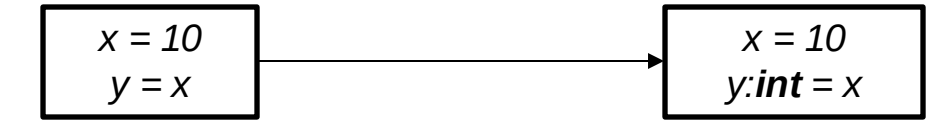

```
{
    "_type": "Assign",
    "col_offset": 0,
    "end_col_offset": 6,
    "end_lineno": 1,
    "lineno": 1,
    "targets": [
        {
            "_type": "Name",
            "col_offset": 0,
            "ctx": {
                "_type": "Store"
            },
            "end_col_offset": 1,
            "end_lineno": 1,
            "id": "x",
            "lineno": 1
        }
    ],
    "type_comment": null,
    "value": {
        "_type": "Constant",
        "col_offset": 4,
        "end_col_offset": 6,
        "end_lineno": 1,
        "kind": null,
        "lineno": 1,
        "n": 10,
        "s": 10,
        "value": 10
    }
}
```

```
{
    "_type": "AnnAssign",
    "annotation": {
        "_type": "Name",
        "col_offset": 2,
        "ctx": {
            "_type": "Load"
        },
        "end_col_offset": 5,
        "end_lineno": 1,
        "id": "int",
        "lineno": 1
    },
    "col_offset": 0,
    "end_col_offset": 10,
    "end_lineno": 1,
    "lineno": 1,
    "simple": 1,
    "target": {
        "_type": "Name",
        "col_offset": 0,
        "ctx": {
            "_type": "Store"
        },
        "end_col_offset": 1,
        "end_lineno": 1,
        "id": "x",
        "lineno": 1
    },
    "value": {
        "_type": "Constant",
        "col_offset": 8,
        "end_col_offset": 10,
        "end_lineno": 1,
        "kind": null,
        "lineno": 1,
        "n": 10,
        "s": 10,
        "value": 10
    }
}
```

ESBMC Python Frontend

Type annotation

- Referred Variables




```
{
    "_type": "Assign",
    "col_offset": 0,
    "end_col_offset": 5,
    "end_lineno": 2,
    "lineno": 2,
    "targets": [
        {
            "_type": "Name",
            "col_offset": 0,
            "ctx": {
                "_type": "Store"
            },
            "end_col_offset": 1,
            "end_lineno": 2,
            "id": "y",
            "lineno": 2
        }
    ],
    "type_comment": null,
    "value": {
        "_type": "Name",
        "col_offset": 4,
        "ctx": {
            "_type": "Load"
        },
        "end_col_offset": 5,
        "end_lineno": 2,
        "id": "x",
        "lineno": 2
    }
}
```

```
{
    "_type": "AnnAssign",
    "annotation": {
        "_type": "Name",
        "col_offset": 2,
        "ctx": {
            "_type": "Load"
        },
        "end_col_offset": 5,
        "end_lineno": 2,
        "id": "int",
        "lineno": 2
    },
    "col_offset": 0,
    "end_col_offset": 9,
    "end_lineno": 2,
    "lineno": 2,
    "simple": 1,
    "target": {
        "_type": "Name",
        "col_offset": 0,
        "ctx": {
            "_type": "Store"
        },
        "end_col_offset": 1,
        "end_lineno": 2,
        "id": "y",
        "lineno": 2
    },
    "value": {
        "_type": "Name",
        "col_offset": 8,
        "ctx": {
            "_type": "Load"
        },
        "end_col_offset": 9,
        "end_lineno": 2,
        "id": "x",
        "lineno": 2
    }
}
```

ESBMC Python Frontend

Type annotation

- Class instances

```
class MyClass:
  pass

obj = MyClass()
```

```
class MyClass:
  pass

obj:MyClass = MyClass()
```

```
{
    "_type": "Assign",
    "targets": [
        {
            "_type": "Name",
            "col_offset": 0,
            "ctx": {
                "_type": "Store"
            },
            "id": "obj"
        }
    ],
    "value": {
        "_type": "Call",
        "args": [],
        "func": {
            "_type": "Name",
            "col_offset": 6,
            "ctx": {
                "_type": "Load"
            },
            "end_col_offset": 13,
            "end_lineno": 4,
            "id": "MyClass",
            "lineno": 4
        }
    }
}
```

```
{
    "_type": "AnnAssign",
    "annotation": {
        "_type": "Name",
        "ctx": {
            "_type": "Load"
        },
        "id": "MyClass"
    },
    "target": {
        "_type": "Name",
        "col_offset": 0,
        "ctx": {
            "_type": "Store"
        },
        "id": "obj"
    },
    "value": {
        "_type": "Call",
        "args": [],
        "func": {
            "_type": "Name",
            "col_offset": 14,
            "ctx": {
                "_type": "Load"
            },
            "end_col_offset": 21,
            "end_lineno": 4,
            "id": "MyClass"
        }
    }
}
```

MANCHESTER 1824
The University of Manchester

ESBMC Python Frontend

Language Translation Process

Python program

```
1  def add(a:int, b:int) -> int:
2      return a+b
3
4  n1 = 1
5  n2 = 2
6  result = add(n1,n2)
7  assert result == 3
```

Abstract Syntax Tree

```
{
    "_type": "Assert",
    "test": {
        "_type": "Compare",
        "comparators": [
            {
                "_type": "Constant",
                "value": 3
            }
        ],
        "left": {
            "id": "result",
            "lineno": 7
        },
        "ops": [
            {
                "_type": "Eq"
            }
        ]
    }
}
```

ESBMC IREP

```
code
  * type: code
  * operands:
    0: =
        * type: bool
        * operands:
          0: symbol
              * type: signedbv
                * width: 32
              * name: result
              * identifier: py:test.py@result
          1: constant
              * type: signedbv
                * width: 32
              * value: 00000000000000000000000000000011
  * statement: assert
```

ESBMC Python: Benchmark

Consensus Specifications

- Consensus protocol dictate how the participants in Ethereum agree on the validity of transactions, and the state of the system.
- Git repository with **Markdown** documents describing specifications.
- Infrastructure to generate **Python** libraries from Markdown

ESBMC Python: Benchmark

Ethereum Consensus Specification

Markdown

consensus-specs / specs / phase0 / **beacon-chain.md**

Preview | Code | Blame | 1939 lines (1617 loc) · 71.4 KB

Math

`integer_squareroot`

```python
def integer_squareroot(n: uint64) -> uint64:
    """
    Return the largest integer ``x`` such that ``x**2 <= n``.
    """
    x = n
    y = (x + 1) // 2
    while y < x:
        x = y
        y = (x + n // x) // 2
    return x
```

`xor`

```python
def xor(bytes_1: Bytes32, bytes_2: Bytes32) -> Bytes32:
    """
    Return the exclusive-or of two 32-byte strings.
    """
    return Bytes32(a ^ b for a, b in zip(bytes_1, bytes_2))
```

eth2spec Python Library

mainnet.py

lib > python3.10 > site-packages > eth2spec-1.4.0b4-py3.10.egg > eth2spec > bellatrix > mainnet.py > integer_squareroot

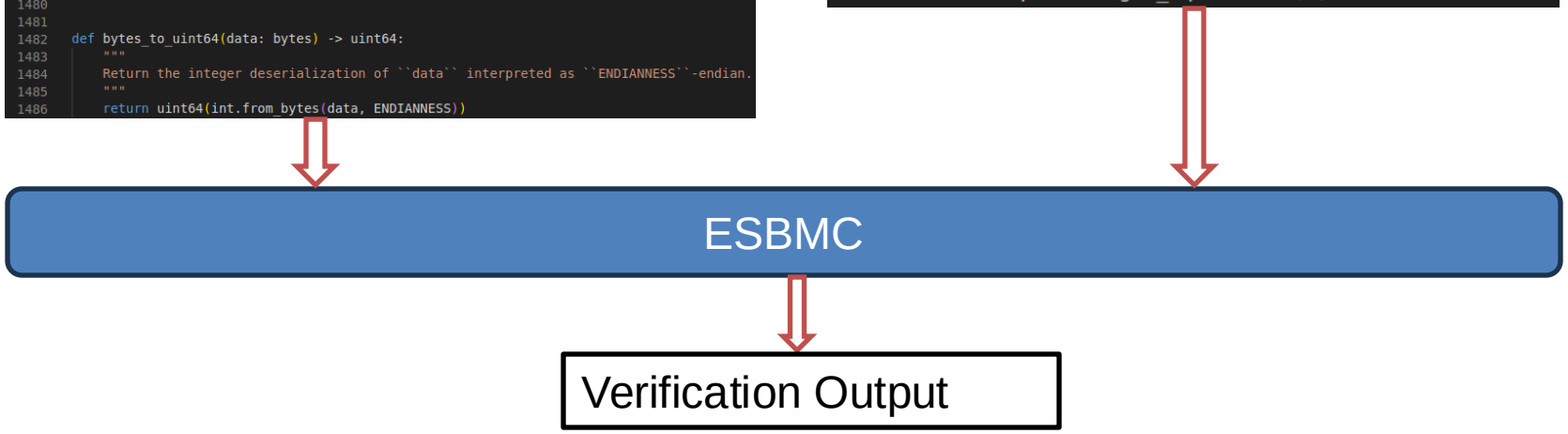
```python
1461
1462
1463  def integer_squareroot(n: uint64) -> uint64:
1464      """
1465      Return the largest integer ``x`` such that ``x**2 <= n``.
1466      """
1467      x = n
1468      y = (x + 1) // 2
1469      while y < x:
1470          x = y
1471          y = (x + n // x) // 2
1472      return x
1473
1474
1475  def xor(bytes_1: Bytes32, bytes_2: Bytes32) -> Bytes32:
1476      """
1477      Return the exclusive-or of two 32-byte strings.
1478      """
1479      return Bytes32(a ^ b for a, b in zip(bytes_1, bytes_2))
1480
1481
1482  def bytes_to_uint64(data: bytes) -> uint64:
1483      """
1484      Return the integer deserialization of ``data`` interpreted as ``ENDIANNESS``-endian.
1485      """
1486      return uint64(int.from_bytes(data, ENDIANNESS))
```

Python Application

integer_squareroot.py

eth2bmc > samples > helpers > math > integer_squareroot.py > ...

```python
1  from eth2spec.bellatrix import mainnet as spec
2  from eth2spec.utils.ssz.ssz_typing import (uint64)
3
4  x = uint64(16)
5  assert spec.integer_squareroot(x) == 4
6
7  x = uint64(25)
8  assert spec.integer_squareroot(x) == 5
```



MANCHESTER 1824
The University of Manchester

Ethereum Consensus Specification

Constants

```
# Constant vars
TARGET_AGGREGATORS_PER_COMMITTEE = 2**4
INTERVALS_PER_SLOT = uint64(3)
ETH_TO_GWEI = uint64(10**9)
SAFETY_DECAY = uint64(10)
NODE_ID_BITS = 256
GENESIS_SLOT = Slot(0)
GENESIS_EPOCH = Epoch(0)
FAR_FUTURE_EPOCH = Epoch(2**64 - 1)
BASE_REWARDS_PER_EPOCH = uint64(4)
DEPOSIT_CONTRACT_TREE_DEPTH = uint64(2**5)
JUSTIFICATION_BITS_LENGTH = uint64(4)
ENDIANNESS: Final = 'little'
BLS_WITHDRAWAL_PREFIX = Bytes1('0x00')
ETH1_ADDRESS_WITHDRAWAL_PREFIX = Bytes1('0x01')
DOMAIN_BEACON_PROPOSER = DomainType('0x00000000')
DOMAIN_BEACON_ATTESTER = DomainType('0x01000000')
DOMAIN_RANDAO = DomainType('0x02000000')
DOMAIN_DEPOSIT = DomainType('0x03000000')
DOMAIN_VOLUNTARY_EXIT = DomainType('0x04000000')
DOMAIN_SELECTION_PROOF = DomainType('0x05000000')
DOMAIN_AGGREGATE_AND_PROOF = DomainType('0x06000000')
DOMAIN_APPLICATION_MASK = DomainType('0x00000001')
TARGET_AGGREGATORS_PER_SYNC_SUBCOMMITTEE = 2**4
SYNC_COMMITTEE_SUBNET_COUNT = 4
G2_POINT_AT_INFINITY = BLSSignature(b'\xc0' + b'\x00' * 95)
TIMELY_SOURCE_FLAG_INDEX = 0
TIMELY_TARGET_FLAG_INDEX = 1
TIMELY_HEAD_FLAG_INDEX = 2
TIMELY_SOURCE_WEIGHT = uint64(14)
TIMELY_TARGET_WEIGHT = uint64(26)
TIMELY_HEAD_WEIGHT = uint64(14)
SYNC_REWARD_WEIGHT = uint64(2)
PROPOSER_WEIGHT = uint64(8)
WEIGHT_DENOMINATOR = uint64(64)
DOMAIN_SYNC_COMMITTEE = DomainType('0x07000000')
DOMAIN_SYNC_COMMITTEE_SELECTION_PROOF = DomainType('0x08000000')
DOMAIN_CONTRIBUTION_AND_PROOF = DomainType('0x09000000')
PARTICIPATION_FLAG_WEIGHTS = [TIMELY_SOURCE_WEIGHT, TIMELY_TARGET_WEIGHT, TIMELY_HEAD_WEIGHT]
MAX_REQUEST_LIGHT_CLIENT_UPDATES = 2**7
SAFE_SLOTS_TO_IMPORT_OPTIMISTICALLY = 128
```

MANCHESTER 1824
The University of Manchester

Ethereum Consensus Specification

Containers

```python
class Container(_ContainerBase):
    _field_indices: Dict[str, int]
    __slots__ = '_field_indices'

    def __new__(cls, *args, backing: Optional[Node] = None, hook: Optional[ViewHook] = None,
                append_nodes: Optional[PyList[Node]] = None, **kwargs):
        if backing is not None:
            if len(args) != 0 or append_nodes is not None:
                raise Exception("cannot have both a backing and elements to init fields")
            return super().__new__(cls, backing=backing, hook=hook, **kwargs)

        input_nodes = []
        for fkey, ftyp in cls.fields().items():
            fnode: Node
            if fkey in kwargs:
                finput = kwargs.pop(fkey)
                if isinstance(finput, View):
                    fnode = finput.get_backing()
                else:
                    fnode = ftyp.coerce_view(finput).get_backing()
            else:
                fnode = ftyp.default_node()
            input_nodes.append(fnode)
        # if this is the base of some container subclass, add the subclass nodes to the backing we are building.
        if append_nodes is not None:
            input_nodes.extend(append_nodes)

        # check if any keys are remaining to catch unrecognized keys
        if len(kwargs) > 0:
            raise AttributeError(f'The field names [{"".join(kwargs.keys())}] are not defined in {cls}')

        out = super().__new__(cls, hook=hook, append_nodes=input_nodes)
        return out

    def __init_subclass__(cls, *args, **kwargs):
        super().__init_subclass__(*args, **kwargs)
        cls._field_indices = {fkey: i for i, fkey in enumerate(cls.fields())}
        if len(cls._field_indices) == 0:
            raise Exception(f"Container {cls.__name__} must have at least one field!")
```

```python
class _ContainerBase(ComplexView):
    __slots__ = ()

    def __new__(cls, *args, backing: Optional[Node] = None, hook: Optional[ViewHook] = None,
                append_nodes: Optional[PyList[Node]] = None, **kwargs):
        if backing is not None:
            if len(args) != 0 or append_nodes is not None:
                raise Exception("cannot have both a backing and elements to init fields")
            return super().__new__(cls, backing=backing, hook=hook, **kwargs)
        if append_nodes is None:
            raise Exception("cannot init container without fields")

        backing = subtree_fill_to_contents(append_nodes, cls.tree_depth())
        out = super().__new__(cls, backing=backing, hook=hook)
        return out

    @classmethod
    def fields(cls) -> Fields:  # base condition for the subclasses deriving the fields
        return {}
```

MANCHESTER 1824
The University of Manchester

Ethereum Consensus Specification

Byte Vectors

```python
class ByteVector(RawBytesView, FixedByteLengthViewHelper, View):
    def __new__(cls, *args, **kwargs):
        byte_len = cls.vector_length()
        out = super().__new__(cls, *args, **kwargs)
        if len(out) != byte_len:
            raise Exception(f"incorrect byte length: {len(out)}, expected {byte_len}")
        return out

    def __class_getitem__(cls, length) -> Type["ByteVector"]:
        chunk_count = (length + 31) // 32
        tree_depth = get_depth(chunk_count)

        class SpecialByteVectorView(ByteVector):
            @classmethod
            def default_node(cls) -> Node:
                return subtree_fill_to_length(zero_node(0), tree_depth, chunk_count)

            @classmethod
            def tree_depth(cls) -> int:
                return tree_depth

            @classmethod
            def type_byte_length(cls) -> int:
                return length

        return SpecialByteVectorView

    @classmethod
    def vector_length(cls):
        return cls.type_byte_length()

    @classmethod
    def default_bytes(cls) -> bytes:
        return b"\x00" * cls.vector_length()

    @classmethod
    def type_repr(cls) -> str:
        return f"ByteVector[{cls.vector_length()}]"
```

MANCHESTER 1824
The University of Manchester

Ethereum Consensus Specification

Bounded Integers

```python
class uint32(uint):
    __slots__ = ()

    @classmethod
    def type_byte_length(cls) -> int:
        return 4


class uint64(uint):
    __slots__ = ()

    @classmethod
    def type_byte_length(cls) -> int:
        return 8

    # JSON encoder should be able to handle uint64, converting it to a string if necessary.
    # no "to_obj" here.


class uint128(uint):
    __slots__ = ()

    @classmethod
    def type_byte_length(cls) -> int:
        return 16

    def to_obj(self) -> ObjType:
        return "0x" + self.encode_bytes().hex()
```

```python
class uint(int, BasicView):
    __slots__ = ()

    def __new__(cls, value: int):
        if value < 0:
            raise ValueError(f"unsigned type {cls} must not be negative")
        byte_len = cls.type_byte_length()
        if value.bit_length() > (byte_len << 3):
            raise ValueError(f"value out of bounds for {cls}")
        return super().__new__(cls, value)  # type: ignore

    def __add__(self: T, other: int) -> T:
        return self.__class__(super().__add__(self.__class__.coerce_view(other)))

    def __radd__(self: T, other: int) -> T:
        return self.__add__(other)

    def __sub__(self: T, other: int) -> T:
        return self.__class__(super().__sub__(self.__class__.coerce_view(other)))

    def __rsub__(self: T, other: int) -> T:
        return self.__class__(self.__class__.coerce_view(other).__sub__(self))

    def __mul__(self, other):
        if not isinstance(other, int):
            return super().__mul__(other)
        return self.__class__(super().__mul__(self.__class__.coerce_view(other)))

    def __rmul__(self, other):
        return self.__mul__(other)

    def __mod__(self: T, other: int) -> T:
        return self.__class__(super().__mod__(self.__class__.coerce_view(other)))

    def __rmod__(self: T, other: int) -> T:
        return self.__class__(self.__class__.coerce_view(other).__mod__(self))

    def __floordiv__(self: T, other: int) -> T:  # Better known as "//"
        return self.__class__(super().__floordiv__(self.__class__.coerce_view(other)))
```

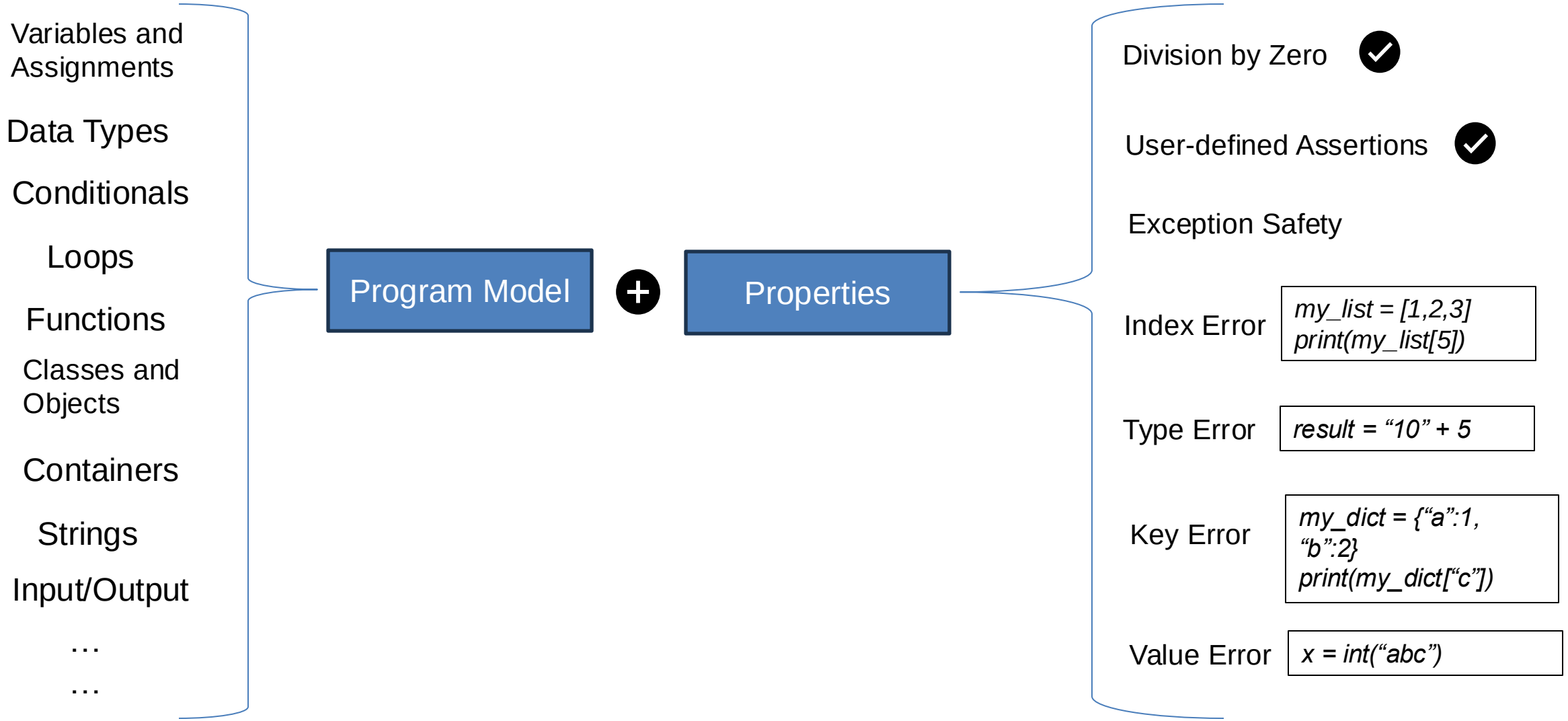
MANCHESTER 1824
The University of Manchester

Ethereum Consensus Specification

Verification Demo

MANCHESTER 1824
The University of Manchester

Python Model Checking



MANCHESTER 1824
The University of Manchester

ESBMC Python Frontend: Next steps

- Refine **type annotation** to incorporate additional features
- **Complete modeling** for remaining language features
- Add verification for specific **Python properties**
- Python **benchmark suite**
- Integration with **Large Language Models**

Python Model Checking

Thank you

Bruno Farias, Youcheng Sun, Lucas C. Cordeiro
bruno.farias@manchester.ac.uk
University of Manchester
29th January 2024