Representing Objects

Eelco Visser

TU Delft

CS4200 | Compiler Construction | December 16, 2021

This Lecture

Objects

- classes and methods in ChocoPy
- object layout

Objects

Classes and Methods in ChocoPy

class definition

inheritance

```
class animal(object):
  makes_noise:bool = False

  def make_noise(self: "animal") → object:
    if (self.makes_noise):
      print(self.sound())

  def sound(self: "animal") → str:
    return "???"

class cow(animal):
  def __init__(self: "cow"):
    self.makes_noise = True

  def sound(self: "cow") → str:
    return "moo"

c:animal = None
c = cow()
c.make_noise()
```

attribute

method definition

method call

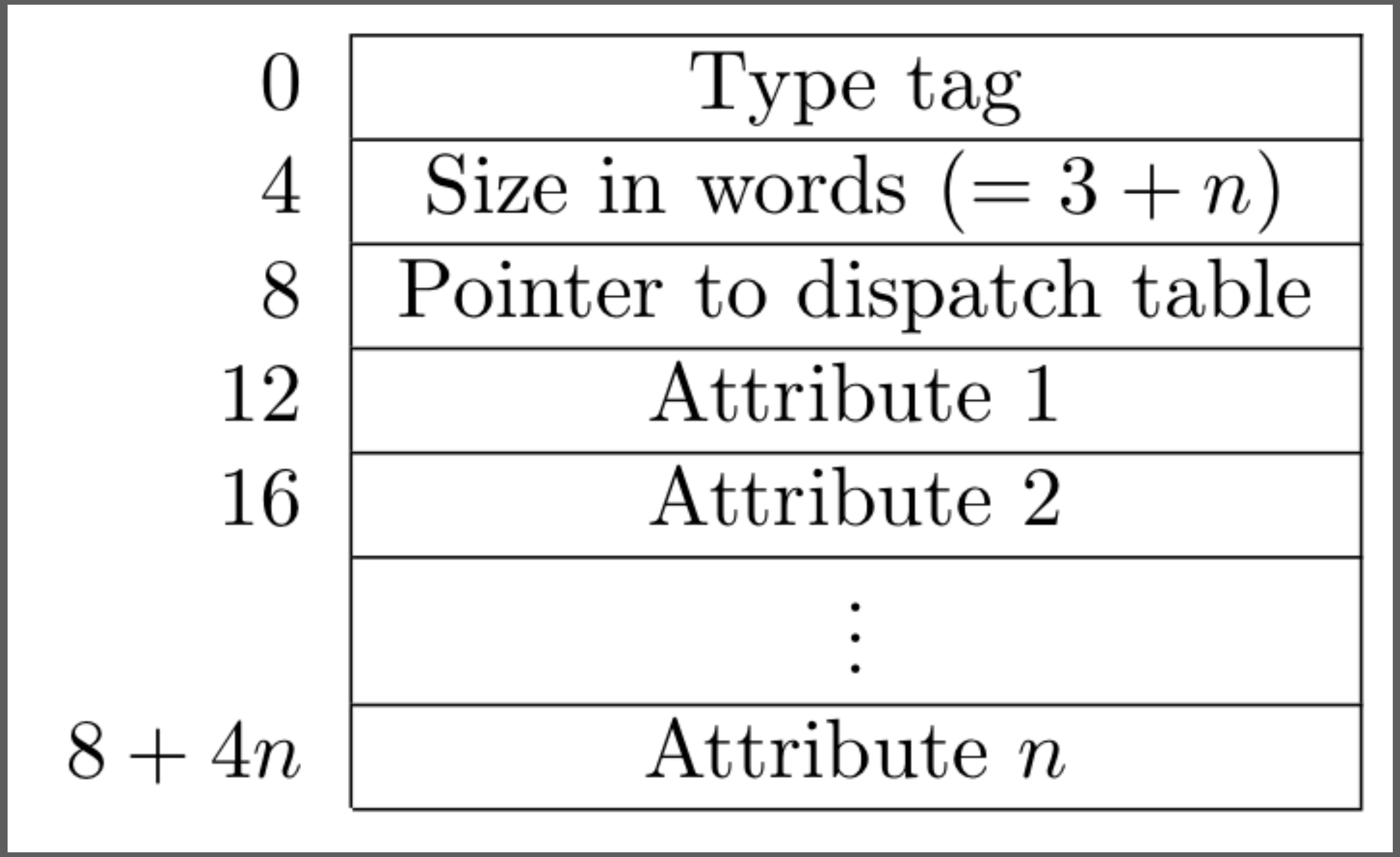
object initialization

attribute reference

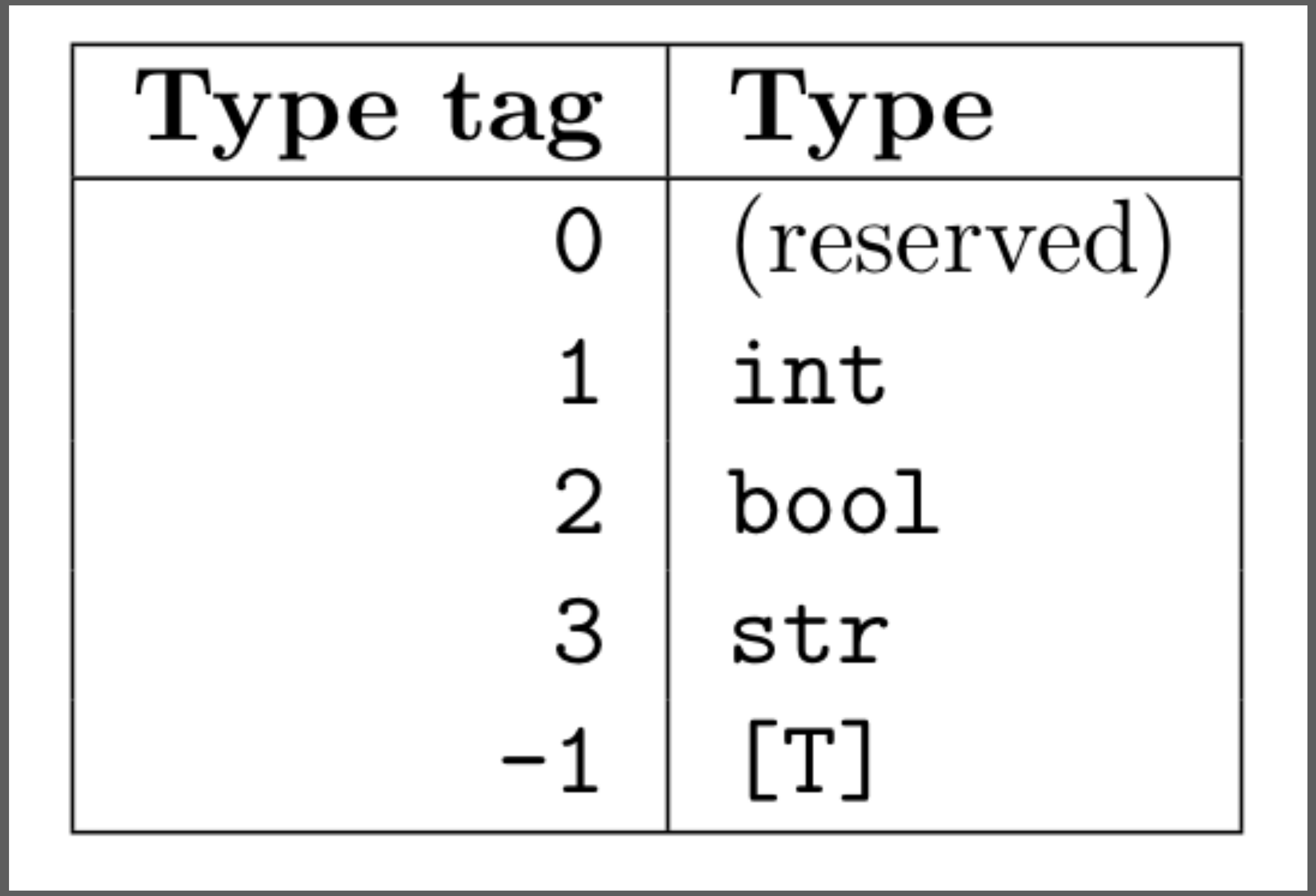
object construction

method call

Object Layout



0	Type tag
4	Size in words ($= 3 + n$)
8	Pointer to dispatch table
12	Attribute 1
16	Attribute 2
	$\vdots$
$8 + 4n$	Attribute n



Type tag	**Type**
`0`	(reserved)
`1`	`int`
`2`	`bool`
`3`	`str`
`-1`	`[T]`

Prototypes

```
class animal(object):
  makes_noise:bool = False

  def make_noise(self: "animal") -> object:
    if (self.makes_noise):
      print(self.sound())

  def sound(self: "animal") -> str:
    return "???"

class cow(animal):
  def __init__(self: "cow"):
    self.makes_noise = True

  def sound(self: "cow") -> str:
    return "moo"

c:animal = None
c = cow()
c.make_noise()
```

0	Type tag
4	Size in words $(= 3 + n)$
8	Pointer to dispatch table
12	Attribute 1
16	Attribute 2
	$\vdots$
$8 + 4n$	Attribute n

```
.globl $object$prototype
$object$prototype:
  .word 0                     # Type tag for class: object
  .word 3                     # Object size
  .word $object$dispatchTable # Pointer to dispatch table
  .align 2

.globl $int$prototype
$int$prototype:
  .word 1                    # Type tag for class: int
  .word 4                    # Object size
  .word $int$dispatchTable   # Pointer to dispatch table
  .word 0                    # Initial value of attribute: __int__
  .align 2

.globl $animal$prototype
$animal$prototype:
  .word 4                     # Type tag for class: animal
  .word 4                     # Object size
  .word $animal$dispatchTable # Pointer to dispatch table
  .word 0                     # Initial value of attribute: makes_noise
  .align 2

.globl $cow$prototype
$cow$prototype:
  .word 5                     # Type tag for class: cow
  .word 4                     # Object size
  .word $cow$dispatchTable    # Pointer to dispatch table
  .word 0                     # Initial value of attribute: makes_noise
  .align 2
```

Prototypes & Dispatch Tables

```
class animal(object):
  makes_noise:bool = False

  def make_noise(self: "animal") → object:
    if (self.makes_noise):
      print(self.sound())

  def sound(self: "animal") → str:
    return "???"

class cow(animal):
  def __init__(self: "cow"):
    self.makes_noise = True

  def sound(self: "cow") → str:
    return "moo"

c:animal = None
c = cow()
c.make_noise()
```

```
.globl $animal$prototype
$animal$prototype:
  .word 4                       # Type tag for class: animal
  .word 4                       # Object size
  .word $animal$dispatchTable # Pointer to dispatch table
  .word 0                       # Initial value of attribute: makes_noise
  .align 2

.globl $cow$prototype
$cow$prototype:
  .word 5                       # Type tag for class: cow
  .word 4                       # Object size
  .word $cow$dispatchTable    # Pointer to dispatch table
  .word 0                       # Initial value of attribute: makes_noise
  .align 2
```

```
.globl $animal$dispatchTable
$animal$dispatchTable:
  .word $object.__init__     # Implementation for method: animal.__init__
  .word $animal.make_noise # Implementation for method: animal.make_noise
  .word $animal.sound        # Implementation for method: animal.sound

.globl $cow$dispatchTable
$cow$dispatchTable:
  .word $cow.__init__        # Implementation for method: cow.__init__
  .word $animal.make_noise # Implementation for method: cow.make_noise
  .word $cow.sound           # Implementation for method: cow.sound
```

same interface as super class

include inherited methods

override methods

Object Creation & Initialization

```
class animal(object):
  makes_noise:bool = False

  def make_noise(self: "animal") → object:
    if (self.makes_noise):
      print(self.sound())

  def sound(self: "animal") → str:
    return "???"

class cow(animal):
  def __init__(self: "cow"):
    self.makes_noise = True

  def sound(self: "cow") → str:
    return "moo"

c:animal = None
c = cow()
c.make_noise()
```

alloc copies prototype

```
  la a0, $cow$prototype       # Load pointer to prototype of: cow
  jal alloc                   # Allocate new object in A0

  sw a0, -12(fp)              # Push on stack slot 3
  sw a0, -16(fp)              # Push argument 0 from last.
  addi sp, fp, -16            # Set SP to last argument.
  lw a1, 8(a0)                # Load address of object's dispatch table
  lw a1, 0(a1)                # Load address of method: cow.__init__
  jalr a1                     # Invoke method: cow.__init__

  addi sp, fp, -@..main.size # Set SP to stack frame top.
  lw a0, -12(fp)              # Pop stack slot 3
  sw a0, $c, t0               # Assign global: c (using tmp register)
```

constructor calls __init_ method

```
.globl $cow.__init__
$cow.__init__:
  …
  li a0, 1                          # Load boolean literal: true
  sw a0, -12(fp)                    # Push on stack slot 3
  lw a0, 0(fp)                      # Load var: cow.__init__.self
  mv a1, a0                         # Move object
  lw a0, -12(fp)                    # Pop stack slot 3
  bnez a1, label_11                 # Ensure not None
  j error.None                      # Go to error handler
label_11:
  sw a0, 12(a1)                     # Set attribute: cow.makes_noise
  …
  jr ra                             # Return to caller
```

0	Type tag
4	Size in words $(= 3 + n)$
8	Pointer to dispatch table
12	Attribute 1
16	Attribute 2
	$\vdots$
$8 + 4n$	Attribute n

Method Call: Dynamic Dispatch

```
class animal(object):
  makes_noise:bool = False

  def make_noise(self: "animal") → object:
    if (self.makes_noise):
      print(self.sound())

  def sound(self: "animal") → str:
    return "???"

class cow(animal):
  def __init__(self: "cow"):
    self.makes_noise = True

  def sound(self: "cow") → str:
    return "moo"

c:animal = None
c = cow()
c.make_noise()
```

not null check

do not invoke function label directly

```
  lw a0, $c          # Load global: c
  bnez a0, label_1 # Ensure not None
  j error.None     # Go to error handler

label_1:
  sw a0, -16(fp)   # Push argument 0 from last.
  lw a0, -16(fp)   # Peek stack slot 3
  lw a1, 8(a0)     # Load address of object's dispatch table
  lw a1, 4(a1)     # Load address of method: animal.make_noise
  addi sp, fp, -16 # Set SP to last argument.
  jalr a1          # Invoke method: animal.make_noise
```

look up address of actual method in dispatch table

```
.globl $animal$dispatchTable
$animal$dispatchTable:
  .word $object.__init__     # Implementation for method: animal.__init__
  .word $animal.make_noise # Implementation for method: animal.make_noise
  .word $animal.sound        # Implementation for method: animal.sound

.globl $cow$dispatchTable
$cow$dispatchTable:
  .word $cow.__init__        # Implementation for method: cow.__init__
  .word $animal.make_noise # Implementation for method: cow.make_noise
  .word $cow.sound           # Implementation for method: cow.sound
```

0	Type tag
4	Size in words ($= 3 + n$)
8	Pointer to dispatch table
12	Attribute 1
16	Attribute 2
	⋮
$8 + 4n$	Attribute n

Accessing Attributes

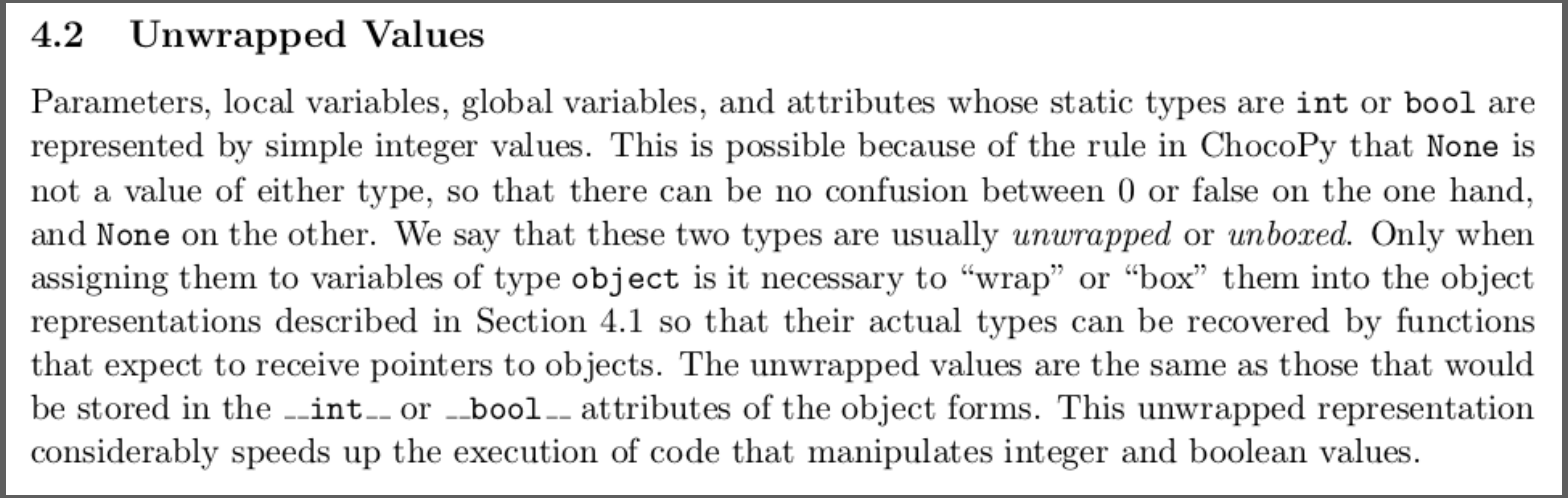
```
class animal(object):
  makes_noise:bool = False

  def make_noise(self: "animal") → object:
    if (self.makes_noise):
      print(self.sound())

  def sound(self: "animal") → str:
    return "???"

class cow(animal):
  def __init__(self: "cow"):
    self.makes_noise = True

  def sound(self: "cow") → str:
    return "moo"

c:animal = None
c = cow()
c.make_noise()
```

offset in object in memory

```
.globl $animal.make_noise
$animal.make_noise:

  …

  lw a0, 0(fp)              # Load var: animal.make_noise.self
  bnez a0, label_5          # Ensure not None
  j error.None              # Go to error handler
label_5:
  lw a0, 12(a0)             # Get attribute: animal.makes_noise
  beqz a0, label_4          # Branch on false.

  …

label_4:
  mv a0, zero               # Load None
  j label_3                 # Jump to function epilogue
label_3:
  …
  jr ra                     # Return to caller
```

0	Type tag
4	Size in words $(= 3 + n)$
8	Pointer to dispatch table
12	Attribute 1
16	Attribute 2
	$\vdots$
$8 + 4n$	Attribute n

Boxed vs Unboxed Values



4.2 Unwrapped Values

Parameters, local variables, global variables, and attributes whose static types are `int` or `bool` are represented by simple integer values. This is possible because of the rule in ChocoPy that `None` is not a value of either type, so that there can be no confusion between 0 or false on the one hand, and `None` on the other. We say that these two types are usually *unwrapped* or *unboxed*. Only when assigning them to variables of type `object` is it necessary to "wrap" or "box" them into the object representations described in Section 4.1 so that their actual types can be recovered by functions that expect to receive pointers to objects. The unwrapped values are the same as those that would be stored in the `__int__` or `__bool__` attributes of the object forms. This unwrapped representation considerably speeds up the execution of code that manipulates integer and boolean values.