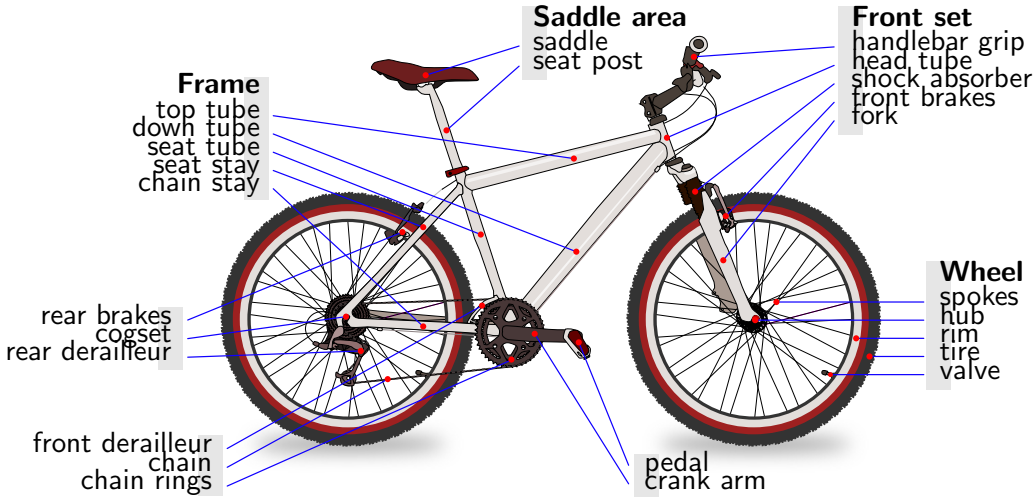
Invariant slicing by accessibility

Ilgiz Mustafin

Constructor Institute of Technology, Schaffhausen, Switzerland

16th Alpine Verification Meeting, September 2024

Full bicycle



Model of a bicycle

Bicycle:

- Front wheel
- Back wheel

Model of a bicycle

Bicycle:

- Front wheel
- Back wheel
- Valid (or usable) when the wheels are attached

Code of a bicycle

```
class
  BICYCLE
```

Code of a bicycle

```
class
  BICYCLE

feature
```

Code of a bicycle

```
class
  BICYCLE

feature

  front_wheel: detachable WHEEL -- detachable means ‘possibly Void’
```

Code of a bicycle

```
class
  BICYCLE

feature

  front_wheel: detachable WHEEL -- detachable means ‘possibly Void’
  back_wheel: detachable WHEEL
```

Code of a bicycle

```
class
  BICYCLE

feature

  front_wheel: detachable WHEEL -- detachable means ‘possibly Void’
  back_wheel: detachable WHEEL

-- Valid when:
```

Code of a bicycle

```
class
  BICYCLE

feature

  front_wheel: detachable WHEEL -- detachable means 'possibly Void'
  back_wheel: detachable WHEEL

-- Valid when:

  front_wheel ≠ Void
```

Code of a bicycle

```
class
  BICYCLE

feature

  front_wheel: detachable WHEEL -- detachable means 'possibly Void'
  back_wheel: detachable WHEEL

-- Valid when:

  front_wheel ≠ Void
  back_wheel ≠ Void
end
```

Code of a bicycle

```
class
  BICYCLE

feature

  front_wheel: detachable WHEEL -- detachable means 'possibly Void'
  back_wheel: detachable WHEEL

invariant -- Valid when:

  front_wheel ≠ Void
  back_wheel ≠ Void
end
```

What is a class invariant?

Class invariant is the validity condition for every object of this class

Code of bicycle's operations

An operation that can be done on a bicycle is to change its front wheel:

```
class
  BICYCLE

feature

  change_front_wheel (new_wheel: WHEEL)
    do -- Invariant holds
      remove_front_wheel
      -- now has no front wheel. Invariant does not hold
      install_front_wheel (new_wheel)
      -- now has a front wheel
    end -- Invariant holds

end
```

Dynamic and static

What to do with invariants?

- Check invariants during the program execution
- Verify (prove) statically that invariants hold at all points where they must hold

Our research is focused on the static verification aspect.

Example: Marriage

When invariant depends on other objects, things get more complicated:

```
class
  PERSON
feature
  spouse: detachable PERSON
  is_married: BOOLEAN
  marry (other: PERSON)
    do
      set_married ; other.set_married
      set_spouse (other) ; other.set_spouse (Current)
    end
  divorce
    do
      spouse := Void
      is_married := False
    end
invariant
  married_iff_has_spouse: is_married = (spouse ≠ Void)
  reciprocal: is_married implies (spouse.spouse = Current)
end
```

Example: Reference leak

We can break the invariant of an object even without touching it:

```
Alice.marry(Bob)
Alice.divorce
Alice.marry(Charles)
```

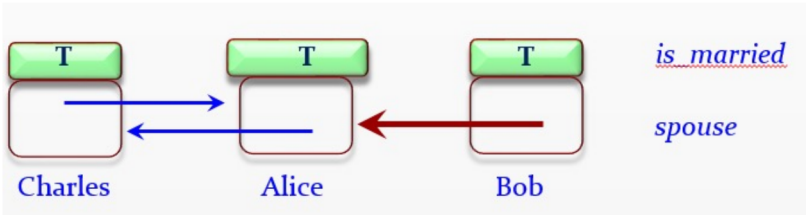


Image from [MAK24]

Current solution. Semantic collaboration

Reference leak is solved by keeping track of two collections of objects:

- *observers* — objects whose invariants depend on this object
- *subjects* — other objects that are used in the invariant of this object

```
set_spouse (other: PERSON)
  require
    inv_only ([]) -- don't expect invariant to hold
  do
    spouse := other
  end

invariant
  spouse_subject: is_married implies subjects = { spouse }
    -- reading 'spouse.spouse' is now allowed

  spouse_observer: is_married implies spouse.observers.has (Current)
  reciprocal: is_married implies (spouse.spouse = Current)
```

Example: Furtive access

```
marry (other: PERSON)
  do
    set_married
    other.set_married
    set_spouse (other)

      -- Here other.is_married = True but other.spouse = Void
      -- so other's invariant does not hold
      -- but it must hold before the call
    other.set_spouse (Current)
  end
```

New solution. Invariant slicing

The recent paper from our chair proposes a new solution without the annotation burden [MAK24].

Invariant slicing is based on using the feature export status.

When making a call `x.r` on an object `x` of the routine `r`, only the part of the invariant which has the same or lower visibility (export status) can be relied on and only that part of the invariant must be reestablished.

New solution. Invariant slicing for furtive access

```
class
  PERSON
feature
  spouse: detachable PERSON
  is_married: BOOLEAN

  marry (other: PERSON)
    do
      set_married ; other.set_married
      set_spouse (other) ; other.set_spouse (Current)
    end

feature {PERSON}
  set_married (m: BOOLEAN) do married := m end
  set_spouse (o: PERSON) do spouse := o end

invariant
  married_iff_has_spouse: is_married = (spouse ≠ Void)
  reciprocal: is_married implies (spouse.spouse = Current)
end
```

References I

Bertrand Meyer, Alisa Arkadova, and Alexander Kogtenkov, *The concept of class invariant in object-oriented programming*, Formal Aspects of Computing **36** (2024), no. 1, 1–38.

arXiv link: `https://arxiv.org/abs/2109.06557`

These slides are based on our previous presentation with Alessandro Schena at the CIRCUS workshop in June 2024 in Schaffhausen.

Currently we are working on the integration into EiffelStudio. Stay tuned!