SCIENCESPRINGDAY

FCT FACULDADE DE CIÊNCIAS E TECNOLOGIA UNIVERSIDADE NOVA DE LISBOA

DEPARTAMENTO DE INFORMÁTICA

View Typestate

SOFTWARE SYSTEMS / PLASTIC Team

CHI CENTER FOR INFORMATICS AND INFORMATION TECHNOLOGIES

Filipe Militão

(PhD Student)

Advised by: Luís Caires (FCT/UNL)

Jonathan Aldrich (Carnegie Mellon University)

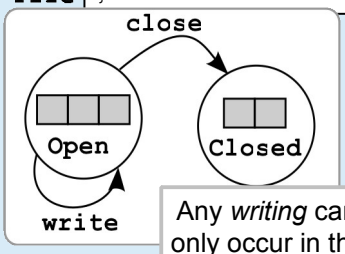
Current research interests include programming languages and models.

Problem:

- There is a pressing need to improve software quality and reduce defects (such as bugs) in programs.
- In object-oriented programs, the mutable state of objects may have different *internal invariants* during its life-time.
- Multiple references may point to the same object (**aliasing**) in non-obvious ways.
- It is hard to have both flexible and safe verification in the use of **state** in the presence of **aliasing** – it requires carefully managing the possibility of *interferences*.

```
void useFile( File file1, File file2 ){
  file1.close();
  file2.write("hello!");
}
```

Is this call safe? If `file1` and `file2` point to the same object this must be forbidden.



Any *writing* can only occur in the `Open` state.

Each reference to an object should work independently and its assumptions on the state should not be broken by other aliases.

Methodology:

- Assign types that encode state (**typestate**) to each reference/alias and track then throughout the program to ensure state is correct.
- To alias an object, break it down into **views** – small and independent permissions to partitions of an object that coordinate to avoid conflicting interferences.
- We consider 2 kinds of views: ***disjoint*** or ***overlapping***.

1. Disjoin views.

Each alias is the only reference to that part of the object. Thus, the alias only has permission to part of the state but that permission is exclusive/unique.

`p` is aliased in `pairInit` but in a safe way since the state usable by each alias is separate.

```
void init(){
  Pair p = new Pair();
  pairInit( p , p );
}
void pairInit(Left l, Right r){
  l.left = 1;
  r.right = 2;
}
```

`Pair` can be broken in `Left` and `Right`.

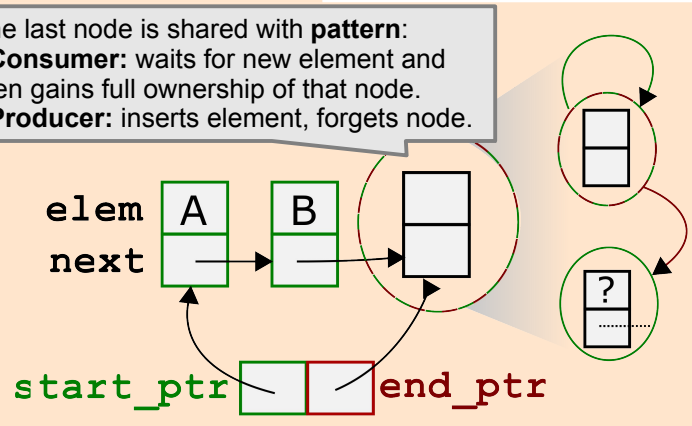
`Pair = Left * Right`

2. Overlapping views.

The state accessible by a view overlaps others. However, its representation must then encode an *access pattern* – a type describing the structured way in which the shared mutable state is usable (locally) by an alias. This pattern expresses the changes that the alias can do and how the other aliases may interfere with it.

A **Pipe** allows for *consumer-producer* style of sharing where the **Consumer** takes elements from a buffer (here modeled by a linked-list) and the **Producer** inserts more elements – each working independently of the other.

The last node is shared with **pattern**:
- **Consumer:** waits for new element and then gains full ownership of that node.
- **Producer:** inserts element, forgets node.



`Consumer` uniquely owns the elements already in the buffer.

`Pipe = Consumer * Producer`

Goals:

- Statically (i.e. without running the code) verify that a program does not have conflicting interferences, using a *type system*.
- Prove that its *typing rules* guarantee that only non-conflicting interferences can happen in a valid program.
- Develop a prototype for the core language.

Funding:

FCT Fundação para a Ciência e a Tecnologia
MINISTÉRIO DA EDUCAÇÃO E CIÊNCIA
SFRH / BD / 33765 / 2009 and INTERFACES NGN44-2009-2012

Information and Communication Technologies Institute
Carnegie Mellon | PORTUGAL
AN INTERNATIONAL PARTNERSHIP