Problem: linked data structures

- What about this case?
- Can we reason about them modularly?

class Person {
    int freeDay;
    Meeting next;
    invariant this.next != null
    => this.next.day != freeDay;
}

class Meeting {
    int day;
    invariant 0 ≤ day<7;

    void Reschedule(int d)
        requires 0 ≤ d <7;
    {
        expose(this){
            day = d;
        }
    }
}
Threats to Person’s object invariant

Another object might invoke to a method of class Meeting breaking Person’s invariant:

Call reschedule(4)

next.day != freeDay

By modifying $m$ I might break the invariant of other class (Foo)

$\text{elem.day} \geq 5$
Threats to Person’s object invariant

next.day != freeDay

restrict Meeting’s aliasing
Expose+ownership

next.day != freeDay

p1:Person

m:Meeting

inv = { Mutable, Valid, Committed }
Expos^ownership

\textbf{Exposure Ownership}

\textbf{call reschedule(4)}

\textbf{next.day} \neq \textbf{freeDay}

\textbf{requires this.inv=Valid}

\textbf{m:Meeting}

\textbf{p1:Person}

\textbf{next}

\textbf{owner}

\textbf{inv =}

\begin{align*}
\text{Mutable} \\
\text{Valid} \\
\text{Committed}
\end{align*}
Exposé+ownership

```plaintext
call reschedule(4)

next.day != freeDay

p1:Person

next

owner

m:Meeting

inv = {Mutable, Valid, Committed}
```
next.day != freeDay

requires this.inv=Valid

inv = \{Mutable, Valid, Committed\}
Expose+ownership

next.day != freeDay

call reschedule(4)

call reschedule(4)

call reschedule(4)

call reschedule(4)

m:Meeting

p1:Person

owner

next

inv =

Mutable

Valid

Committed
Expose+ownership

next.day != freeDay

call reschedule(4)

call reschedule(4)

m:Meeting

p1:Person

owner

next

inv = {Mutable, Valid, Committed}
Exposé+ownership

```plaintext
next.day != freeDay

call reschedule(4)

p1:Person

m:Meeting

inv = { Valid, Committed }

next

owner
```
**Expose+ownership**

Next day != freeDay

**call reschedule(4)**

\[ p1:Person \]

\[ m:Meeting \]

Next.

Owner.

*p1:Person* will return *m:Meeting* to Committed status.

inv = [Mutable, Valid, Committed]
Expose+ownership

next.day != freeDay

In case it exists, P1’s owner will return this object to Committed status

inv = 
{Mutable, Valid, Committed}
Which objects must be in Mutable status if Room is Mutable?

From Person’s perspective, what fields can I access?
Ownership is an **acyclic** relation

- I can not own my owner
- Each object has at most one owner

**Ownership rule:**

- If o.inv = Mutable, then owner(o), owner(owner(o)), ...
  are Mutable.

The object invariant of o can only depend on:

- The fields of o
- Any field of any other object which o **owns** (recursively)
A new ghost field is added:
- **owner**: reference to the “owner” of the object
- Field `inv` values are ∈ {Committed, Valid, Mutable}

An object status is Committed if:
- The object invariant holds
- Its `owner` is not in Mutable status

Committed: acts as a lock to guarantee validity
The `rep` (representation) modifier introduces implicitly *ownership* invariants

```java
class Person {
    int freeDay;
    [rep] Meeting next;

    /* implicit invariant
     * next ≠ null ⇒ next.owner = this;
    */
    ...
}
```
pack/unpack is extended to support this new protocol

unpack o:

assert o.inv = Valid;
o.inv := Mutable;
foreach (c | c.owner = o)
  { c.inv := Valid; }

pack o:

assert o.inv = Mutable;
assert ∀c: c.owner = o ⇒ c.inv = Valid;
foreach (c | c.owner = o)
  { c.inv := Committed; }
assert Invariant(o);
o.inv := Valid
Memory state:
\[ \forall o: o.inv \neq \text{mutable} \Rightarrow \]
\[ \text{Inv}(o) \land \]
\[ (\forall c: c.owner = o \Rightarrow c.inv = \text{Committed}) \]

Admissible Invariants:

Only accesses to fields
- this.f₁. … .fₙ, where f₁ … .fₙ₋₁ are fields of “rep” references
class Person {
    int freeDay;
    rep Meeting next;

    invariant next ≠ null ⇒
        next.day ≠ freeDay;

    int doTravel(int td)
        requires inv==valid;
        modifies this.*;
    {
        expose(this) {
            freeDay = td;
            if (next!=null) {
                next.reschedule((td+1)%7);
            }
        }
    }
}

class Meeting {
    int day;
    void reschedule( int d )
        requires inv==valid; {
            expose(this) {
                day = d;
            }
        }
    }
}

Person person = ... ;
Meeting meeting = ... ;
person.next := meeting ;

The only owner of meeting is person
• \textbf{[Rep] defines} an object hierarchy
• What happens to other (recursive) structures?
Example: cyclic list?

```cpp
class CyclicList {
  [rep] Node header;
  //implicit invariant header.owner == this
}

class Node {
  [rep] Node next;
  //implicit invariant next.owner == this
}
```
Problem with cyclic lists

class CyclicList {
    [rep] Node header;
    //header.owner == this
}

class Node {
    [rep] Node next;
    //next.owner == this
}
Peer references

- The `[rep]` modifier states **I am** the owner of the reference

- The `[peer]` modifier states the reference and **I** share the **same** owner
class CyclicList {

    [rep] Node header;
    //header.owner == this
}

class Node {

    [peer] Node next;
    // next.owner == this.owner
}
This methodology deals with

- Re-entrancy (using the “inv” field value)
- Nested structures (using ownership)

It handles:

- Recursive linked structures (lists)
- Recursion, ownership transference (not seen today)

It allows a modular verification

- Check only the invariant of the class under analysis
- Access protocol (inv field)
- Aliasing is not restricted
Some references

- Tutorial Spec#

- Paper: