Structural Lock Correlation with Ownership Types

Yi Lu, John Potter, Jingling Xue
University of New South Wales, Sydney

ESOP'13 Proceedings of the 22nd European conference of Programming Languages and Systems

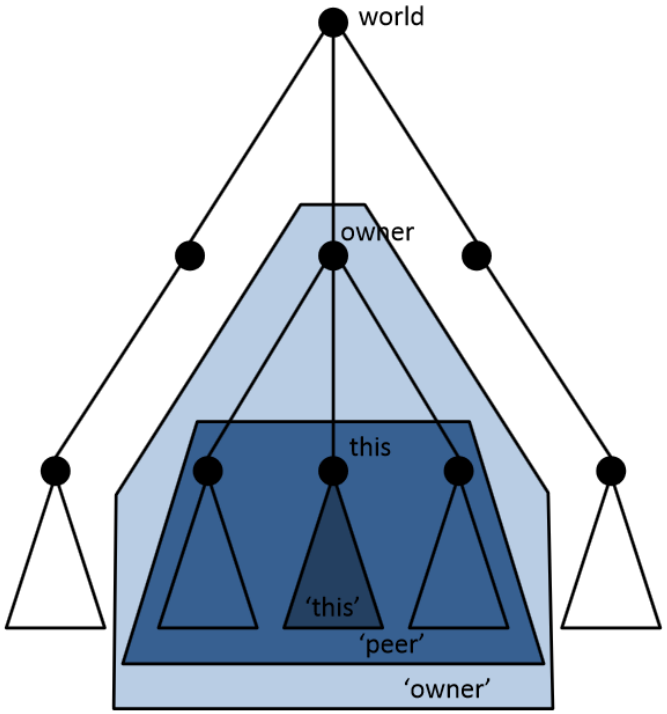
Motivation

- Concurrent object-oriented programming is hard
- Locks used to coordinate conflicting memory accesses
- Locking behaviour of libraries not always formally specified
- Use of (arbitrary) locks:
  - Hard to enforce lock specification
  - Information hiding vs. Fine grained locking

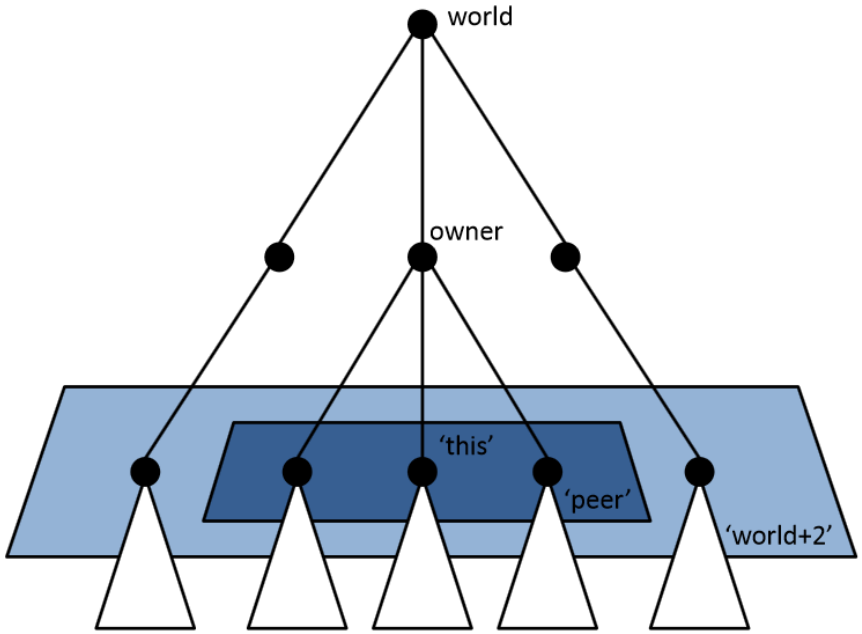
- We want to be able to abstract locks in a useful way

Ownership

Containment



Context



Source: Structural Lock Correlation with Ownership Types, Yi Lu, John Potter, Jingling Xue

Locks and Effects

- Lock correlation: Relationship between lock and the guarded memory
- Notation: $\langle L :: \varepsilon \rangle$

- Effects: Objects on which reads/writes occur
  - Writes more important in this context

Example: Lock correlation

```
par {
      sync (l1) { o1.f = … }; // A
      sync (l2) { o2.f = … }; // B
}
```

- Corresponding Lock Effects:

  `<l1::o1>` and `<l2::o2>`

Example: Lock correlation cont.

Lock Effects: `<l1::o1>` and `<l2::o2>`

- Safe when:
  - `o1` and `o2` are not aliased
  - `l1` and `l2` are aliased

Example: Lock correlation cont.

Lock Effects: `<l1::o1>` and `<l2::o2>`

Assumption: `l1` owns `o1` and `l2` owns `o2`

- If `l1` and `l2` are not aliased `o1` and `o2` must be different objects
- If `l1` and `l2` are aliased the tasks are correctly synchronized

Structural Lock Correlation

- In structural lock correlations the lock must own all of the associated side effects

- Notation: $<[\omega]::\varepsilon>$
  - ω is held when side-effect ε occurs and ω contains ε

- Two structural locks at the same rank are correctly synchronized:
  - Two locks are either aliased
  - Or the effects cannot overlap

Lock Abstraction

- Cannot precisely name actual locks and side-effects in larger scopes
- Abstraction of locks and side-effects possible
  - Precise information lost
  - But: Structural correlation information retained

→ Modularity

Example

```
class Account { int balance = 0; }

class Customer {

private final Account[] accounts;
…
void depositA(int i, int x) {
        Account acct = account[i];
        acct.balance += x;
}
void depositB(int i, int x) {
        Account acct = account[i];
        sync (this) acct.balance += x;
}
void depositC(int i, int x) {
        Account acct = account[i];
        sync (acct) acct.balance += x;
}
}
```

```
Customer c, d; int i, j, x, y;


// case ParA
par {   c.depositA(i, x);
        d.depositA(j, y); }

// case ParB
par {   c.depositB(i, x);
        d.depositB(j, y); }

// case ParC
par {   c.depositC(i, x);
        d.depositC(j, y); }
```

Example

- Structural Lock Correlation without Ownership:

  depositA: <acct> → <peer>

  depositB: <this::acct> → <this::peer>

  depositC: <[acct]::acct> → <[peer]::peer>

  - ParA

    Conflicting effects

  - ParB

    Conflicting effects (different customers might change the same account at the same time)

  - ParC

    Accepted

Example

```
class Account { int balance = 0; }

class Customer {

private final Account<this>[] accounts;
…
void depositA(int i, int x) {
          Account acct = account[i];
          acct.balance += x;
}
void depositB(int i, int x) {
          Account acct = account[i];
          sync (this) acct.balance += x;
}
void depositC(int i, int x) {
          Account acct = account[i];
          sync (acct) acct.balance += x;
}
}
```

```
Customer c, d; int i, j, x, y;


// case ParA
par {    c.depositA(i, x);
         d.depositA(j, y); }

// case ParB
par {    c.depositB(i, x);
         d.depositB(j, y); }

// case ParC
par {    c.depositC(i, x);
         d.depositC(j, y); }
```

Example

- Structural Lock Correlation with Ownership:
  depositA: <acct> → <this+1>
  depositB: <this::acct> → <[this]::this+1>
  depositC: <[acct]::acct> → <[this+1]::this+1>

  - ParA
    Conflicting effects
  - ParB
    Accepted
  - ParC
    Accepted

Conclusion

- Structural lock correlation is preserved through abstraction
- Possible to enforce locking specification at interface boundaries
- Model supports modular checking of lock usage

Related Work and Citations

- Ownership- and Universe type systems
- SafeJava
  - Fields implicitely guarded
- Locksmith
  - Static race checker for C programs
- Chord
  - Static race checker for java programs

- No citations yet (but quite recent: ‘13)

Review

- Very hard to distinguish the new ideas from previous work
- Formal part of the paper unpresentable

- Good examples