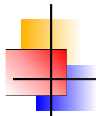


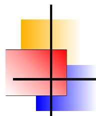
Quantified Abstractions of Distributed Systems

GUILLERMO ROMÁN DÍEZ

UNIVERSIDAD POLITÉCNICA DE MADRID, SPAIN

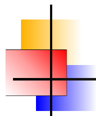
Joint work with
Elvira Albert, Jesús Correas and Germán Puebla

XIII JORNADAS SOBRE PROGRAMACIÓN Y LENGUAJES
(PROLE'13)



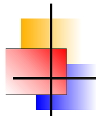
Introduction

- Distribution is currently mainstream in software development
- The configuration (topology) of the system changes dynamically (e.g. cloud computing) and without fixed limits
- When reasoning about Distributed Systems it is essential to have information:
  - Regarding the topology of the distributed system:
    - What are the kinds of nodes that compose the system?
    - How many instances of each kind of node can be created
  - Regarding communication among nodes:
    - With which nodes can each node communicate?
    - What kind of communication is used?
    - How often do nodes communicate?
    - How large is the data sent in each communication?



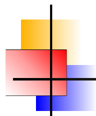
Our Approach Integrates

- A Static Analysis for OO programs called Points-to Analysis to abstract distributed systems:
  - good for inferring qualitative information (shapes)
  - to infer information about nodes (*abstract nodes*)
  - to infer information about communication (*abstract interactions*)
  - which is valid for an unlimited number of elements in the system
- Resource Analysis (a.k.a. Cost Analysis) to infer upper-bounds
  - good for inferring quantitative information
  - the number of nodes each abstract node may represent
  - the number of interactions represented by each abstract interaction
  - the resource consumption of each abstract node.



The Distribution Model: Concurrent Objects

- Concurrent objects form a well established model for distributed concurrent systems
- Concurrent objects live in a distributed environment
- Objects communicate via asynchronous method calls of the form o.m().
  - The o object is responsible for executing the call m()
  - Each method call creates a new task for the corresponding object o



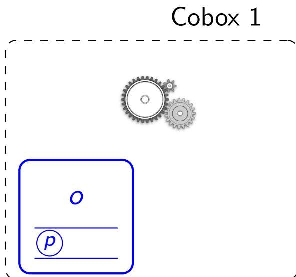
The Distribution Model: Coboxes

- For further flexibility, we allow multiple objects to be grouped into distributed nodes (Coboxes)
  - Each cobox has its own execution engine
  - Such engine executes the tasks of all objects that belong to the cobox
  - Task execution may be interleaved using cooperative concurrency
  - Each Object belongs to a cobox for its entire lifetime
- The topology of the system is determined by the program points where objects are created
  - `new` creates an object that belongs to the current cobox
  - `newcog` creates a new cobox

```
p() {
  a = new Ob();
  b = newcog Ob();
  a.q();
  b.q();
  a.m();
  m();
}
```

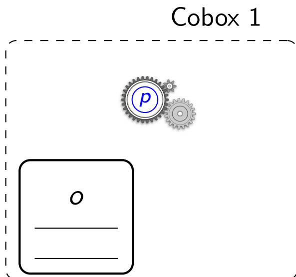
Example of Distributed Execution with Coboxes

```
p() {
  a = new Ob();
  b = newcog Ob();
  a.q();
  b.q();
  a.m();
  m();
}
```

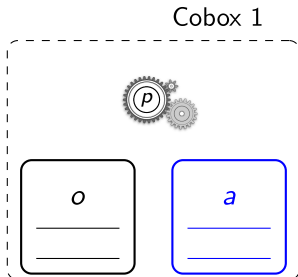


Example of Distributed Execution with Coboxes

```
p() {
   a = new Ob();
   b = newcog Ob();
   a.q();
   b.q();
   a.m();
   m();
}
```

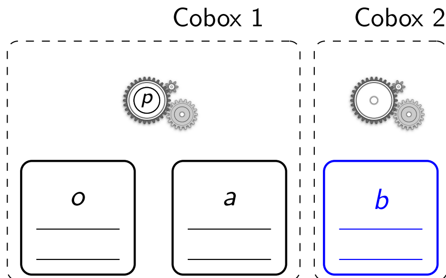


```
p() {
    a = new Ob();
    b = newcog Ob();
    a.q();
    b.q();
    a.m();
    m();
}
```



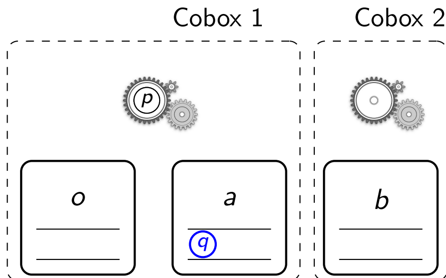
Example of Distributed Execution with Coboxes

```
p() {
    a = new Ob();
    b = newcog Ob();
    a.q();
    b.q();
    a.m();
    m();
}
```



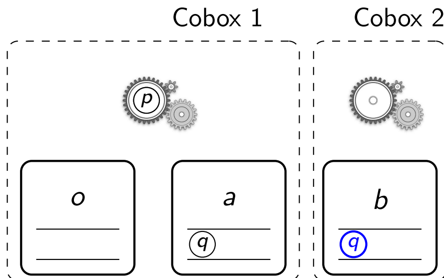
Example of Distributed Execution with Coboxes

```
p() {
  a = new Ob();
  b = newcog Ob();
  a.q();
  b.q();
  a.m();
  m();
}
```



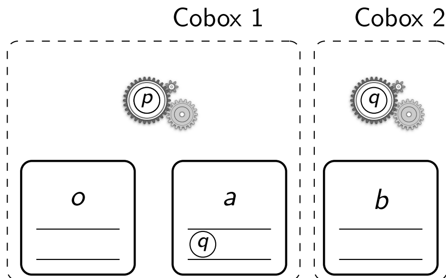
Example of Distributed Execution with Coboxes

```
p() {
  a = new Ob();
  b = newcog Ob();
  a.q();
  b.q();
  a.m();
  m();
}
```



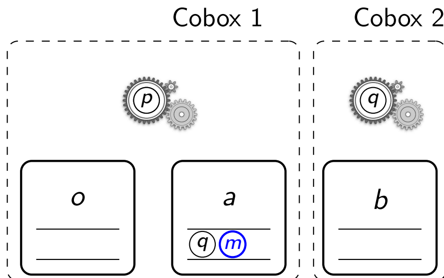
Example of Distributed Execution with Coboxes

```
p() {
  a = new Ob();
  b = newcog Ob();
  a.q();
  b.q();
  a.m();
  m();
}
```



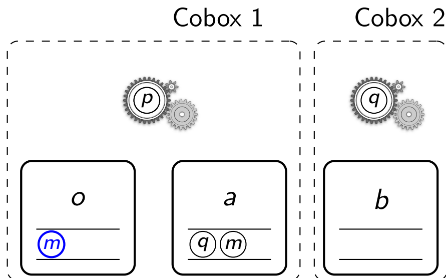
Example of Distributed Execution with Coboxes

```
p() {
  a = new Ob();
  b = newcog Ob();
  a.q();
  b.q();
  a.m();
  m();
}
```



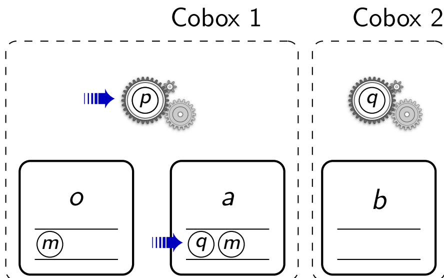
Example of Distributed Execution with Coboxes

```
p() {
  a = new Ob();
  b = newcog Ob();
  a.q();
  b.q();
  a.m();
  m();
}
```



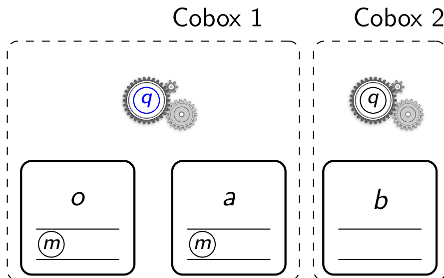
Example of Distributed Execution with Coboxes

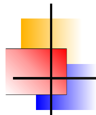
```
p() {
  a = new Ob();
  b = newcog Ob();
  a.q();
  b.q();
  a.m();
  m();
}
```



Example of Distributed Execution with Coboxes

```
p() {
  a = new Ob();
  b = newcog Ob();
  a.q();
  b.q();
  a.m();
  m();
}
```





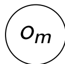
Configuration and Communication

- Before abstracting systems, we introduce the concrete notions of configuration and communication
- The Configuration of a system consists of two elements
  - The set of coboxes created along the execution
  - The set of objects created within each cobox
- The configuration is determined by the program points where objects and coboxes are created
  - `new` or `newcog` instructions
- The Communication is the set of method calls between pairs of objects

Configuration Example

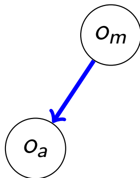
```
void m (int n) {
  Obj a = new Obj();
  Obj b = new Obj();
  b.p(n,a);
}

void p (int n, Obj a) {
  while (n > 0) {
    Obj c = new Obj();
    c.q(a);
    n--;
  }
}

void q (Obj a) {
  a.foo();
}

void foo () {2 inst}
```

```
void m (int n) {
  Obj a = new Obj();
  Obj b = new Obj();
  b.p(n,a);
}

void p (int n, Obj a) {
  while (n > 0) {
    Obj c = new Obj();
    c.q(a);
    n--;
  }
}

void q (Obj a) {
  a.foo();
}

void foo () {2 inst}
```



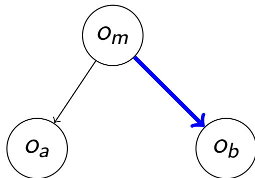
Configuration Example

```
void m (int n) {
  Obj a = new Obj();
  Obj b = new Obj();
  b.p(n,a);
}

void p (int n, Obj a) {
  while (n > 0) {
    Obj c = new Obj();
    c.q(a);
    n--;
  }
}

void q (Obj a) {
  a.foo();
}

void foo () {2 inst}
```



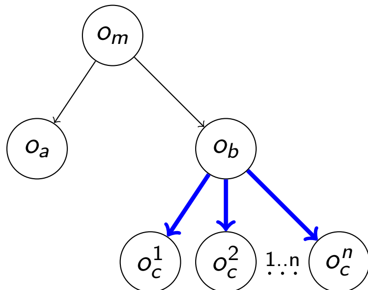
Configuration Example

```
void m (int n) {
  Obj a = new Obj();
  Obj b = new Obj();
  b.p(n,a);
}

void p (int n, Obj a) {
  while (n > 0) {
    Obj c = new Obj();
    c.q(a);
    n--;
  }
}

void q (Obj a) {
  a.foo();
}

void foo () {2 inst}
```



Configuration Example

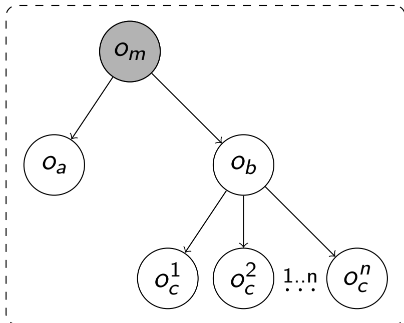
```
void m (int n) {
  Obj a = new Obj();
  Obj b = new Obj();
  b.p(n,a);
}

void p (int n, Obj a) {
  while (n > 0) {
    Obj c = new Obj();
    c.q(a);
    n--;
  }
}

void q (Obj a) {
  a.foo();
}

void foo () {2 inst}
```



Configuration Example

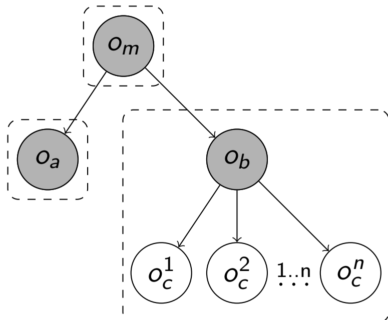
```
void m (int n) {
  Obj a = new Obj();
  Obj b = new Obj();
  b.p(n,a);
}

void p (int n, Obj a) {
  while (n > 0) {
    Obj c = new Obj();
    c.q(a);
    n--;
  }
}

void q (Obj a) {
  a.foo();
}

void foo () {2 inst}
```

Configuration 1



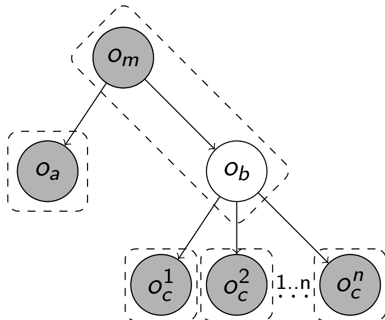
Configuration Example

```
void m (int n) {
  Obj a = newcog Obj();
  Obj b = newcog Obj();
  b.p(n,a);
}

void p (int n, Obj a) {
  while (n > 0) {
    Obj c = new Obj();
    c.q(a);
    n--;
  }
}

void q (Obj a) {
  a.foo();
}

void foo () {2 inst}
```



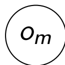
Configuration Example

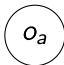
```
void m (int n) {
  Obj a = newcog Obj();
  Obj b = new Obj();
  b.p(n,a);
}

void p (int n, Obj a) {
  while (n > 0) {
    Obj c = newcog Obj();
    c.q(a);
    n--;
  }
}

void q (Obj a) {
  a.foo();
}

void foo () {2 inst}
```

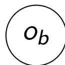


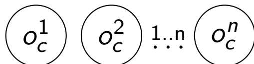
Communication Example

```
void m (int n) {
  Obj a = new Obj();
  Obj b = new Obj();
  b.p(n,a);
}

void p (int n, Obj a) {
  while (n > 0) {
    Obj c = new Obj();
    c.q(a);
    n--;
  }
}

void q (Obj a) {
  a.foo();
}

void foo () {2 inst}
```



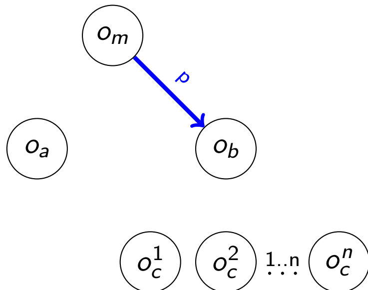






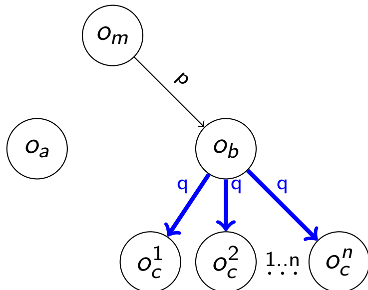
Communication Example

```
void m (int n) {
  Obj a = new Obj();
  Obj b = new Obj();
  b.p(n,a);
}

void p (int n, Obj a) {
  while (n > 0) {
    Obj c = new Obj();
    c.q(a);
    n--;
  }
}

void q (Obj a) {
  a.foo();
}

void foo () {2 inst}
```



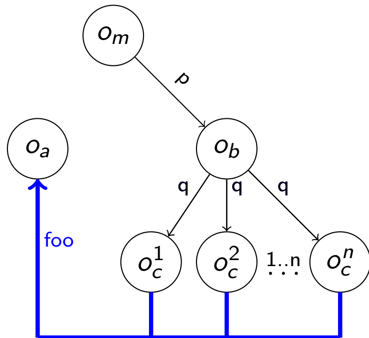
Communication Example

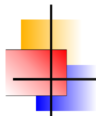
```
void m (int n) {
  Obj a = new Obj();
  Obj b = new Obj();
  b.p(n,a);
}

void p (int n, Obj a) {
  while (n > 0) {
    Obj c = new Obj();
    c.q(a);
    n--;
  }
}

void q (Obj a) {
  a.foo();
}

void foo () {2 inst}
```



Communication Example

```
void m (int n) {
  Obj a = new Obj();
  Obj b = new Obj();
  b.p(n,a);
}

void p (int n, Obj a) {
  while (n > 0) {
    Obj c = new Obj();
    c.q(a);
    n--;
  }
}

void q (Obj a) {
  a.foo();
}

void foo () {2 inst}
```





Abstracting Distributed Systems

Distributed systems are abstracted following the following steps:

1. Identify the nodes
   - Points-to analysis provides an over-approximation the set of objects each reference may point to
   - Allocation sites are used to represent (abstractions of) objects
2. Abstract the system
   - The Abstract Configuration is obtained from the points-to graph
   - The Abstract Communication is obtained from the interactions graph

Abstract Configuration Example

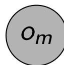
```
(m) void m (int n) {
  (1) Obj a = newcog Obj();
  (2) Obj b = newcog Obj();
      b.p(n,a);
    }

    void p (int n, Obj a) {
      while (n > 0) {
  (3)   Obj c = new Obj();
        c.q(a);
        n--;
      }
    }

    void q (Obj a) {
      a.foo();
    }

    void foo () {2 inst}
```

Points-to Graph

Abstract Configuration Example

```
(m) void m (int n) {
  (1) Obj a = newcog Obj();
  (2) Obj b = newcog Obj();
      b.p(n,a);
    }

    void p (int n, Obj a) {
      while (n > 0) {
  (3)   Obj c = new Obj();
        c.q(a);
        n--;
      }
    }

    void q (Obj a) {
      a.foo();
    }

    void foo () {2 inst}
```

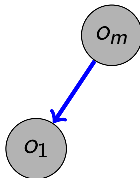
Points-to Graph



Abstract Configuration Example

```
(m) void m (int n) {
(1)   Obj a = newcog Obj();
(2)   Obj b = newcog Obj();
      b.p(n,a);
    }

    void p (int n, Obj a) {
      while (n > 0) {
(3)     Obj c = new Obj();
        c.q(a);
        n--;
      }
    }

    void q (Obj a) {
      a.foo();
    }

    void foo () {2 inst}
```

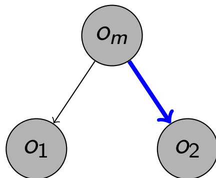
Points-to Graph



Abstract Configuration Example

```
(m) void m (int n) {
    (1) Obj a = newcog Obj();
    (2) Obj b = newcog Obj();
        b.p(n,a);
    }

    void p (int n, Obj a) {
        while (n > 0) {
    (3)     Obj c = new Obj();
            c.q(a);
            n--;
        }
    }

    void q (Obj a) {
        a.foo();
    }

    void foo () {2 inst}
```

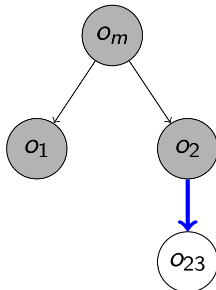
Points-to Graph



Abstract Configuration Example

```
(m) void m (int n) {
    (1) Obj a = newcog Obj();
    (2) Obj b = newcog Obj();
    b.p(n,a);
    }

    void p (int n, Obj a) {
      while (n > 0) {
(3)      Obj c = new Obj();
         c.q(a);
         n--;
      }
    }

    void q (Obj a) {
      a.foo();
    }

    void foo () {2 inst}
```

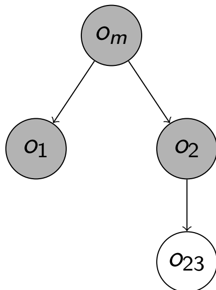
Points-to Graph



Abstract Configuration Example

```
(m) void m (int n) {
    (1) Obj a = newcog Obj();
    (2) Obj b = newcog Obj();
    b.p(n,a);
  }

  void p (int n, Obj a) {
    while (n > 0) {
(3)   Obj c = new Obj();
      c.q(a);
      n--;
    }
  }

  void q (Obj a) {
    a.foo();
  }

  void foo () {2 inst}
```

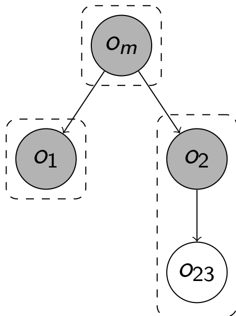
Points-to Graph



Abstract Configuration Example

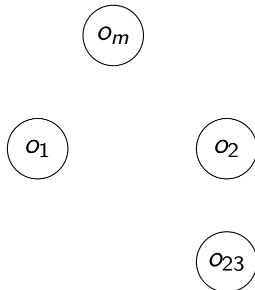
```
ⓜ void m (int n) {
  ① Obj a = newcog Obj();
  ② Obj b = newcog Obj();
    b.p(n,a);
  }

  void p (int n, Obj a) {
    while (n > 0) {
  ③   Obj c = new Obj();
      c.q(a);
      n--;
    }
  }

  void q (Obj a) {
    a.foo();
  }

  void foo () {2 inst}
```

Abstract Configuration



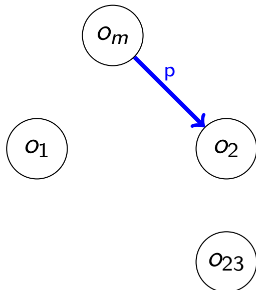
```
(m) void m (int n) {
(1)   Obj a = new Obj();
(2)   Obj b = new Obj();
      b.p(n,a);
    }

    void p (int n, Obj a) {
      while (n > 0) {
(3)     Obj c = new Obj();
        c.q(a);
        n--;
      }
    }

    void q (Obj a) {
      a.foo();
    }

    void foo () {2 inst}
```

Interaction Graph



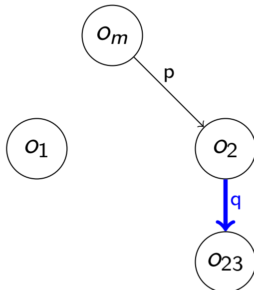
Abstract Communication Example

```
(m) void m (int n) {
  (1) Obj a = new Obj();
  (2) Obj b = new Obj();
      b.p(n,a);
    }

    void p (int n, Obj a) {
      while (n > 0) {
  (3)   Obj c = new Obj();
        c.q(a);
        n--;
      }
    }

    void q (Obj a) {
      a.foo();
    }

    void foo () {2 inst}
```



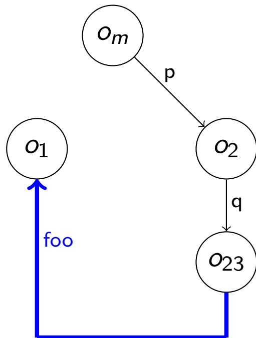
Abstract Communication Example

```
(m) void m (int n) {
(1)   Obj a = new Obj();
(2)   Obj b = new Obj();
      b.p(n,a);
    }

    void p (int n, Obj a) {
      while (n > 0) {
(3)     Obj c = new Obj();
        c.q(a);
        n--;
      }
    }

    void q (Obj a) {
      a.foo();
    }

    void foo () {2 inst}
```

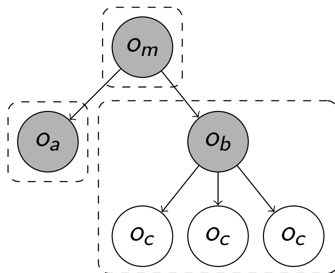


Abstract Communication Example

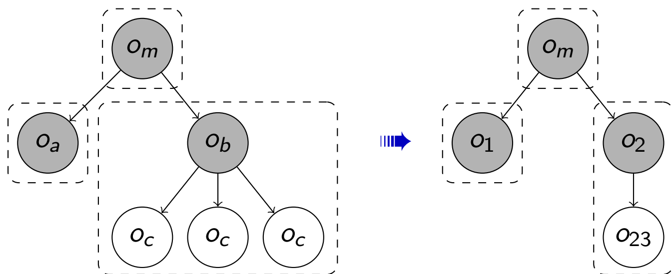
```
(m) void m (int n) {
(1)   Obj a = new Obj();
(2)   Obj b = new Obj();
      b.p(n,a);
    }

    void p (int n, Obj a) {
      while (n > 0) {
(3)     Obj c = new Obj();
        c.q(a);
        n--;
      }
    }

    void q (Obj a) {
      a.foo();
    }

    void foo () {2 inst}
```



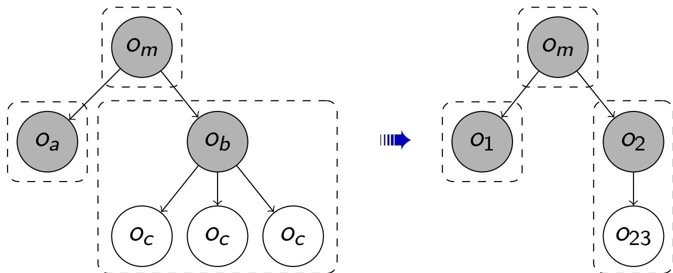
Concrete Vs. Abstract

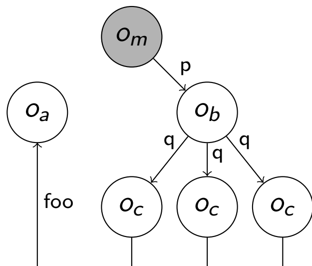


Concrete Vs. Abstract

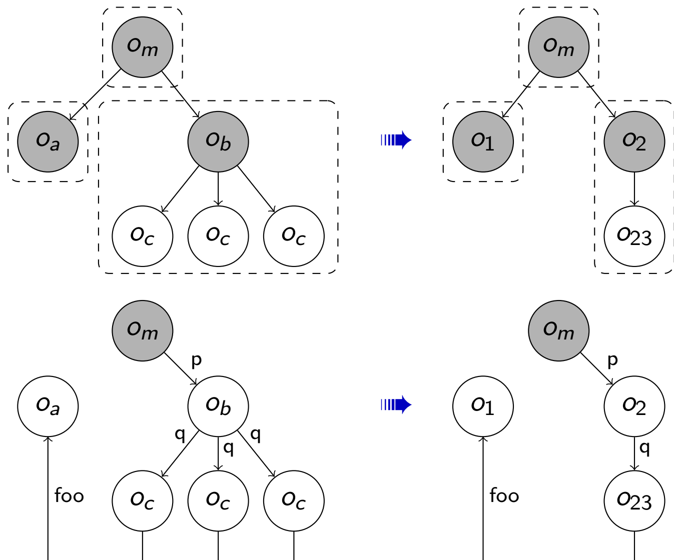


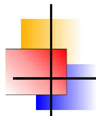
Concrete Vs. Abstract





Concrete Vs. Abstract

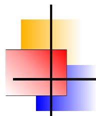




Quantifying Abstractions of Distributed Systems

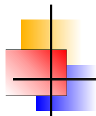
Abstractions are quantified by leveraging Resource-analysis

- Quantified Abstract Configuration: Bound the number of instances created of each abstract node
- Quantified Communication: Bound the number of interactions between abstract nodes



Background on Resource Analysis

- The aim of Resource Analysis is to bound the resource consumption (cost) of executing a given program P on any input data without actually executing P
- Resource Analysis aims at obtaining an Upper-Bound (UB) on the cost of the program
- Cost Models determine the type of resource we are measuring
  - A cost model $\mathcal{M}(b)$ is a function which, for each instruction, returns the cost of executing the instruction b
  - As an example, for counting the number of instructions executed by a program: $\mathcal{M}(b) = 1$



Cost Centers

- UB accumulates the cost of the whole program
- For distributed programs we have to split the global cost among the coboxes which compose the system
  - We are not only interested in counting the cost consumed by each instruction,
  - also in knowing to which cobox the cost must be attributed
- We add Cost Centers to determine the object responsible of executing each instruction
  - E.g., the cost model with cost centers for counting instructions is $\mathcal{M}(b, o) = 1 * c(o)$
- Note that the UB is parametric w.r.t. the objects inferred in the points-to analysis

Cost Centers Example

```
void m (int n) {
  Obj a = newcog Obj();
  Obj b = newcog Obj();
  b.p(n,a);
}

void p (int n, Obj a) {
  while (n > 0) {
    Obj c = new Obj();
    c.q(a);
    n--;
  }
}

void q (Obj a) {
  a.foo();
}

void foo () {2 inst}
```

Cost Centers Example

```
void m (int n) {
  Obj a = newcog Obj();
  Obj b = newcog Obj();
  b.p(n,a);
}

void p (int n, Obj a) {
  while (n > 0) {
    Obj c = new Obj();
    c.q(a);
    n--;
  }
}

void q (Obj a) {
  a.foo();
}

void foo () {2 inst}
```

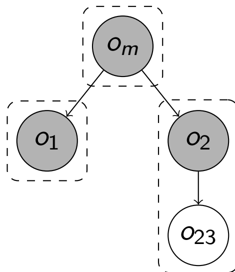
$$UB_m^I(n) = \underbrace{3}_{m} + n * (\underbrace{3}_{p} + \underbrace{1}_{q} + \underbrace{2}_{foo})$$

Cost Centers Example

```
void m (int n) {
  Obj a = newcog Obj();
  Obj b = newcog Obj();
  b.p(n,a);
}

void p (int n, Obj a) {
  while (n > 0) {
    Obj c = new Obj();
    c.q(a);
    n--;
  }
}

void q (Obj a) {
  a.foo();
}

void foo () {2 inst}
```

$$UB^{l}_{m}(n) = \underbrace{3}_{m} + n * (\underbrace{3}_{p} + \underbrace{1}_{q} + \underbrace{2}_{foo})$$

$$UB^{l}_{m}(n) = \underbrace{3 * c(o_m)}_{m} + n * (\underbrace{3 * c(o_2)}_{p} + \underbrace{1 * c(o_{23})}_{q} + \underbrace{2 * c(o_1)}_{foo})$$

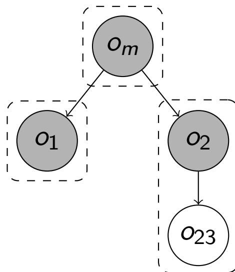


Cost Centers Example

```
void m (int n) {
  Obj a = newcog Obj();
  Obj b = newcog Obj();
  b.p(n,a);
}

void p (int n, Obj a) {
  while (n > 0) {
    Obj c = new Obj();
    c.q(a);
    n--;
  }
}

void q (Obj a) {
  a.foo();
}

void foo () {2 inst}
```

$$UB^{l}_{m}(n) = \underbrace{3}_{m} + n * (\underbrace{3}_{p} + \underbrace{1}_{q} + \underbrace{2}_{foo})$$

$$UB^{l}_{m}(n) = \underbrace{3*c(o_m)}_{m} + n * (\underbrace{3*c(o_2)}_{p} + \underbrace{1*c(o_{23})}_{q} + \underbrace{2*c(o_1)}_{foo})$$

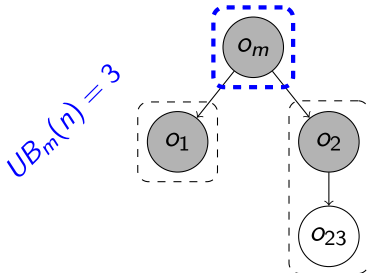


Cost Centers Example

```
void m (int n) {
  Obj a = newcog Obj();
  Obj b = newcog Obj();
  b.p(n,a);
}

void p (int n, Obj a) {
  while (n > 0) {
    Obj c = new Obj();
    c.q(a);
    n--;
  }
}

void q (Obj a) {
  a.foo();
}

void foo () {2 inst}
```

$$UB^{I}_{m}(n) = \underbrace{3}_{m} + n * (\underbrace{3}_{p} + \underbrace{1}_{q} + \underbrace{2}_{foo})$$

$$UB^{I}_{m}(n) = \underbrace{3*c(o_m)}_{m} + n * (\underbrace{3*c(o_2)}_{p} + \underbrace{1*c(o_{23})}_{q} + \underbrace{2*c(o_1)}_{foo})$$

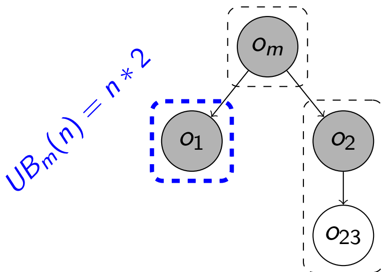


Cost Centers Example

```
void m (int n) {
  Obj a = newcog Obj();
  Obj b = newcog Obj();
  b.p(n,a);
}

void p (int n, Obj a) {
  while (n > 0) {
    Obj c = new Obj();
    c.q(a);
    n--;
  }
}

void q (Obj a) {
  a.foo();
}

void foo () {2 inst}
```

$$UB^{I}_{m}(n) = \underbrace{3}_{m} + n*(\underbrace{3}_{p} + \underbrace{1}_{q} + \underbrace{2}_{foo})$$

$$UB^{I}_{m}(n) = \cancel{\underbrace{3*c(o_m)}_{m}} + n*(\cancel{\underbrace{3*c(o_2)}_{p}} + \cancel{\underbrace{1*c(o_{23})}_{q}} + \underbrace{2*c(o_1)}_{foo})$$

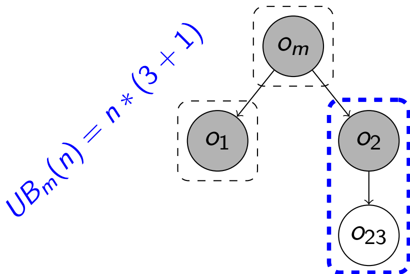


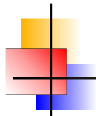
Cost Centers Example

```
void m (int n) {
  Obj a = newcog Obj();
  Obj b = newcog Obj();
  b.p(n,a);
}

void p (int n, Obj a) {
  while (n > 0) {
    Obj c = new Obj();
    c.q(a);
    n--;
  }
}

void q (Obj a) {
  a.foo();
}

void foo () {2 inst}
```

$$UB_m^I(n) = \underbrace{3}_{m} + n * (\underbrace{3}_{p} + \underbrace{1}_{q} + \underbrace{2}_{foo})$$

$$UB_m^I(n) = \underbrace{3{*}c(o_m)}_{m} + n * (\underbrace{3{*}c(o_2)}_{p} + \underbrace{1{*}c(o_{23})}_{q} + \underbrace{2{*}c(o_1)}_{foo})$$





Quantified Configuration

- We now aim at quantifying abstract configurations to over-approximate the number of objects (and coboxes)
- The cost model to count the number of instances is

$$\mathcal{M}^{C}(b, o_l) = \begin{cases} c(o_l) & \text{if } b \equiv \textit{new} \text{ or } \textit{newcog} \\ 0 & \textit{otherwise} \end{cases}$$

- This cost model counts the elements for each particular instance considered by the points-to analysis
- By quantifying the number of coboxes we can estimate the maximum level of parallelism of the program

Quantified Abstract Configuration Example

```
void m (int n) {
  Obj a = newcog Obj();
  Obj b = newcog Obj();
  b.p(n,a);
}

void p (int n, Obj a) {
  while (n > 0) {
    Obj c = new Obj();
    c.q(a);
    n--;
  }
}

void q (Obj a) {
  a.foo();
}

void foo () {2 inst}
```

Quantified Abstract Configuration Example

```
void m (int n) {
  Obj a = newcog Obj();
  Obj b = newcog Obj();
  b.p(n,a);
}

void p (int n, Obj a) {
  while (n > 0) {
    Obj c = new Obj();
    c.q(a);
    n--;
  }
}

void q (Obj a) {
  a.foo();
}

void foo () {2 inst}
```

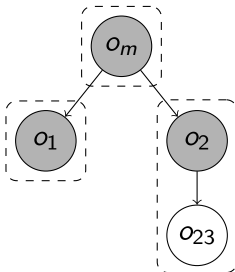
$$UB_m^C(n) = \underbrace{1 + 1}_{o_1 \;\; o_2} + n * \underbrace{1}_{o_{23}}$$

Quantified Abstract Configuration Example

```
void m (int n) {
  Obj a = newcog Obj();
  Obj b = newcog Obj();
  b.p(n,a);
}

void p (int n, Obj a) {
  while (n > 0) {
    Obj c = new Obj();
    c.q(a);
    n--;
  }
}

void q (Obj a) {
  a.foo();
}

void foo () {2 inst}
```

$$UB_m^C(n) = \underbrace{1+1}_{o_1\ \ o_2} \ + \ n * \underbrace{1}_{o_{23}}$$

$$UB_m^C(n) = c(o_1) + c(o_2) + n * c(o_{23})$$

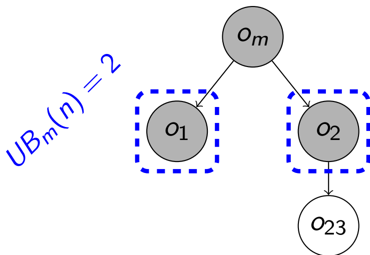


Quantified Abstract Configuration Example

```
void m (int n) {
  Obj a = newcog Obj();
  Obj b = newcog Obj();
  b.p(n,a);
}

void p (int n, Obj a) {
  while (n > 0) {
    Obj c = new Obj();
    c.q(a);
    n--;
  }
}

void q (Obj a) {
  a.foo();
}

void foo () {2 inst}
```

$$UB^{C}_{m}(n) = \underbrace{1+1}_{o_1\ \ o_2} + n * \underbrace{1}_{o_{23}}$$

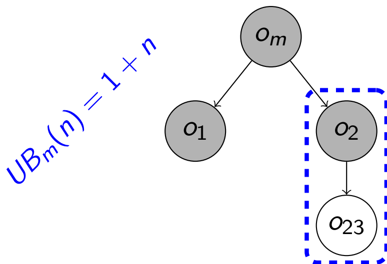
$$UB^{C}_{m}(n) = c(o_1) + c(o_2) + \cancel{n * c(o_{23})}$$



```
void m (int n) {
  Obj a = newcog Obj();
  Obj b = newcog Obj();
  b.p(n,a);
}

void p (int n, Obj a) {
  while (n > 0) {
    Obj c = new Obj();
    c.q(a);
    n--;
  }
}

void q (Obj a) {
  a.foo();
}

void foo () {2 inst}
```

$$UB_m^C(n) = \underbrace{1+1}_{o_1 \ o_2} + n * \underbrace{1}_{o_{23}}$$

$$UB_m^C(n) = \cancel{c(o_1)} + c(o_2) + n * c(o_{23})$$



Quantified Communication

- For capturing interactions between objects we pass as parameters the considered caller and callee objects

$$\mathcal{M}^K(b, o, p) = \begin{cases} c(m)\cdot c(o, p) & \text{if } b \equiv p.m(_) \\ 0 & \textit{otherwise} \end{cases}$$

- This cost model also includes the method used in the call
- The resulting UBs contain cost centers made up of pairs of abstractions c(o, p)
  - o is the object that is executing
  - p is the object responsible for executing the call
- By processing the cost center we can quantify:
  - Interaction between objects ($c(o, p)$)
  - Interactions between coboxes
  - The amount of data transferred between objects ($c(m)$)

Quantified Communication Example

```
void m (int n) {
  Obj a = newcog Obj();
  Obj b = newcog Obj();
  b.p(n,a);
}

void p (int n, Obj a) {
  while (n > 0) {
    Obj c = new Obj();
    c.q(a);
    n--;
  }
}

void q (Obj a) {
  a.foo();
}

void foo () {2 inst}
```

Quantified Communication Example

```
void m (int n) {
  Obj a = newcog Obj();
  Obj b = newcog Obj();
  b.p(n,a);
}

void p (int n, Obj a) {
  while (n > 0) {
    Obj c = new Obj();
    c.q(a);
    n--;
  }
}

void q (Obj a) {
  a.foo();
}

void foo () {2 inst}
```

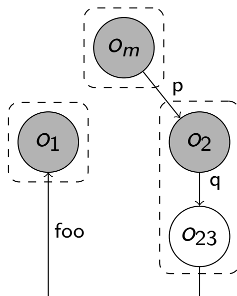
$$UB_m^K(n) = \underbrace{1}_{p} + n * (\underbrace{1}_{q} + \underbrace{1}_{foo})$$

Quantified Communication Example

```
void m (int n) {
  Obj a = newcog Obj();
  Obj b = newcog Obj();
  b.p(n,a);
}

void p (int n, Obj a) {
  while (n > 0) {
    Obj c = new Obj();
    c.q(a);
    n--;
  }
}

void q (Obj a) {
  a.foo();
}

void foo () {2 inst}
```

$$UB_m^K(n) = \underbrace{1}_{p} + n * (\underbrace{1}_{q} + \underbrace{1}_{foo})$$

$$UB_m^K(n) = c(o_m, o_2)*c(p) + n * (c(o_2, o_{23})*c(q) + c(o_{23}, o_1)*c(foo))$$

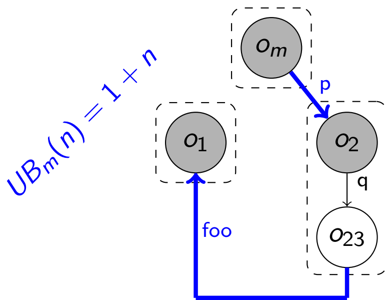


Quantified Communication Example

```
void m (int n) {
  Obj a = newcog Obj();
  Obj b = newcog Obj();
  b.p(n,a);
}

void p (int n, Obj a) {
  while (n > 0) {
    Obj c = new Obj();
    c.q(a);
    n--;
  }
}

void q (Obj a) {
  a.foo();
}

void foo () {2 inst}
```

$$UB_m^K(n) = \underbrace{1}_{p} + n * (\underbrace{1}_{q} + \underbrace{1}_{foo})$$

$$UB_m^K(n) = c(o_m, o_2) * c(p) \; +$$
$$n * (\cancel{c(o_2, o_{23}) * c(q)} + c(o_{23}, o_1) * c(foo))$$

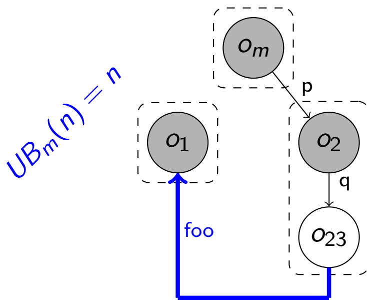


Quantified Communication Example

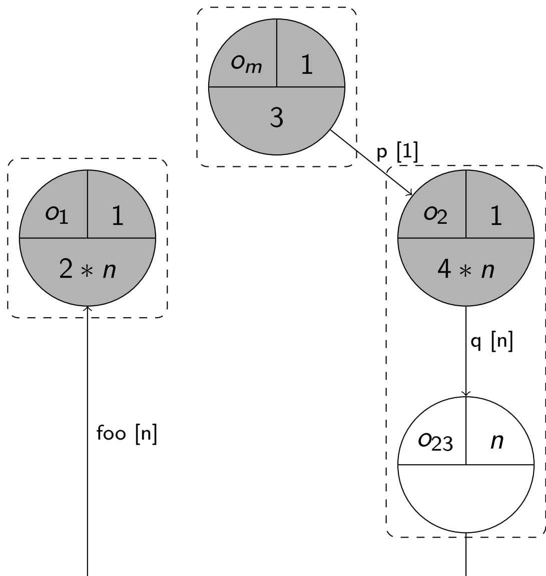
```
void m (int n) {
  Obj a = newcog Obj();
  Obj b = newcog Obj();
  b.p(n,a);
}

void p (int n, Obj a) {
  while (n > 0) {
    Obj c = new Obj();
    c.q(a);
    n--;
  }
}

void q (Obj a) {
  a.foo();
}

void foo () {2 inst}
```

$$UB_m^K(n) = \underbrace{1}_{p} + n * (\underbrace{1}_{q} + \underbrace{1}_{foo})$$

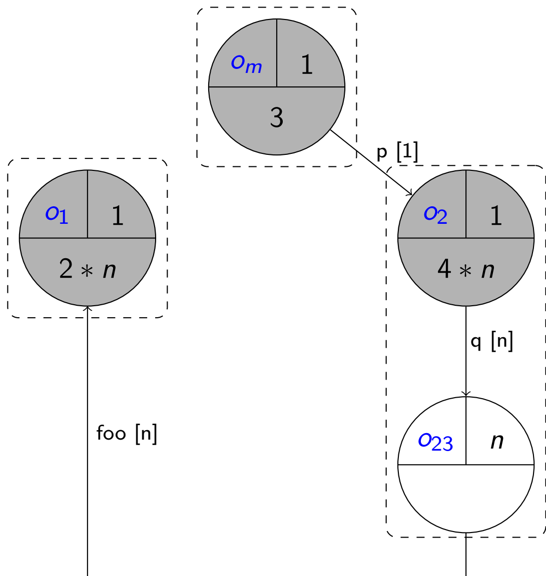
$$UB_m^K(n) = c(o_m, o_2) * c(p) +$$
$$n * (c(o_2, o_{23}) * c(q) + c(o_{23}, o_1) * c(foo))$$



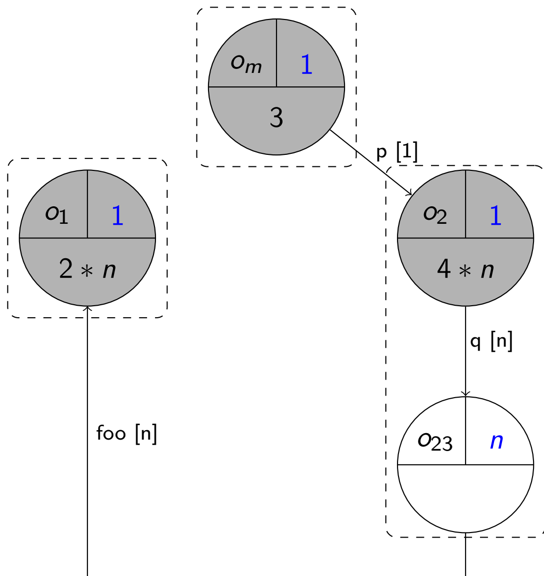
Quantified Abstraction



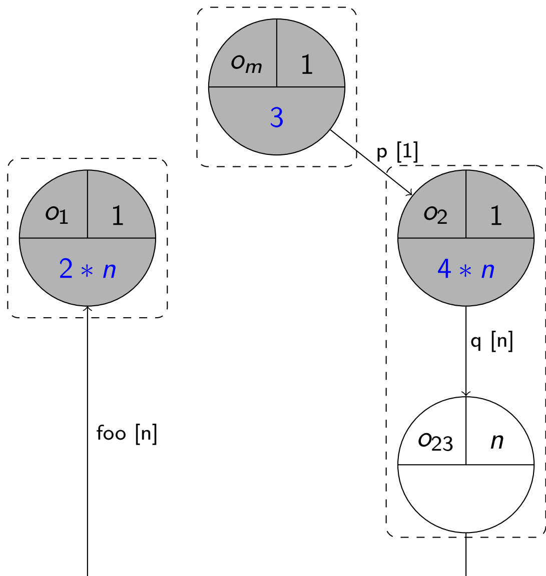
Quantified Abstraction



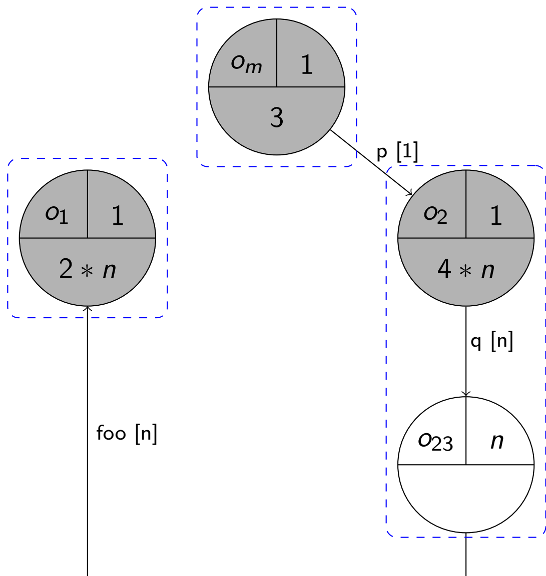
Quantified Abstraction



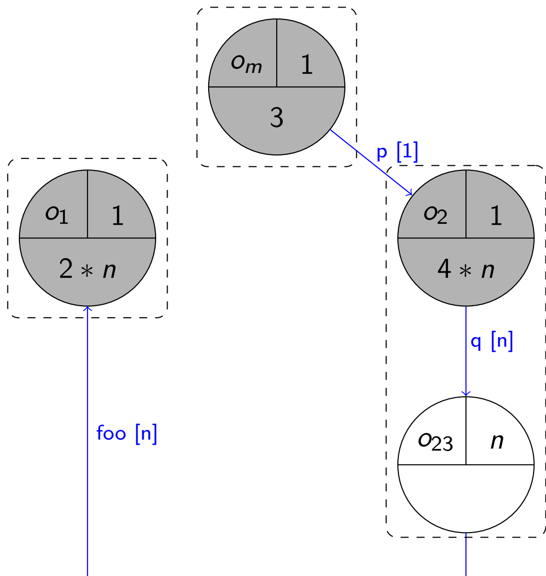
Quantified Abstraction



Quantified Abstraction



Quantified Abstraction



Conclusions

- Quantified Abstractions integrate two well established static analyses:
  - Points-to analysis
  - Resource analysis
- Quantified Abstractions have many applications for distributed applications
  - Provide a global view of the distributed application
  - Allow us to identify nodes that execute a too large number of processes
  - Are useful to perform meaningful resource analysis of distributed systems
  - Allow us to detect components that have many interactions
  - Provide a further step towards static bandwidth analysis.
- Future work
  - Apply it in order to determine optimal configurations
  - Provide resource-usage guarantees regarding both load of different nodes and amount of communication required