Dynamic Package Interface

Damien Zufferey

IST Austria

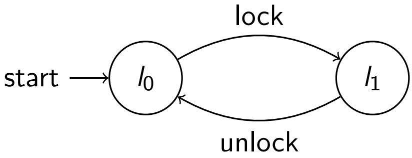
joint work with Shahram Esmaeilsabzali, Rupak Majumdar, and Thomas Wies.

State machine interface for single object

Example: Spin lock

```
void lock(spinlock *lock)
{
  while (1)
  {
    if (!xchg_32(lock, BUSY)) return;
    while (*lock) cpu_relax();
  }
}

void unlock(spinlock *lock)
{
  barrier();
  assert(*lock)
  *lock = 0;
}
```



Dynamic package interface

Extend state-machine interface to group of objects.

There can be an unbounded number of objects.
For the sake of simplicity, the example are shown in a non-concurrent setting.

Example: Sets and Iterators

```
class Set {
  protected int sver, size;
  public Set() {
    sver := size := 0;
  }
  public void add(int elem) {
    if (!duplicate) {
      sver++; size++;
    }
  }
  protected void delete(int pos) {
    sver++; size--;
  }
  public Iterator iterator() {
    return (new Iterator(this));
  }
}
```

```
class Iterator {
  int iver, pos;
  Set it_of;
  protected Iterator(Set s) {
    it_of := s; pos := 0;
    iver := s.sver;
  }
  public int next() {
    if (iver == S.sver) then pos++;
    else throw new Exception();
  }
  public void remove() {
    if (iver == S.sver) then {
      it_of.delete(pos);
      iver := S.sver;
    } else {
      throw new Exception();
} } }
```

Simple OO model: syntax

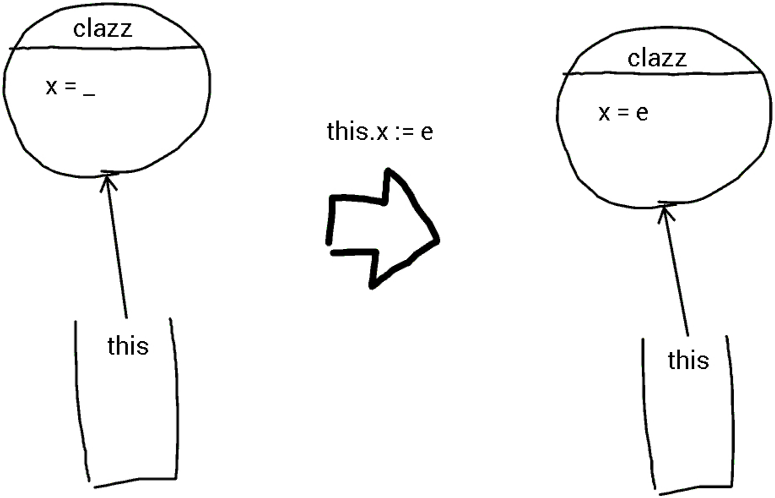
A state is $(O, this, q, v, st)$ where

- O is the set of objects
- q is the control-state
- v is the stack $(this + q)$
- st is the store

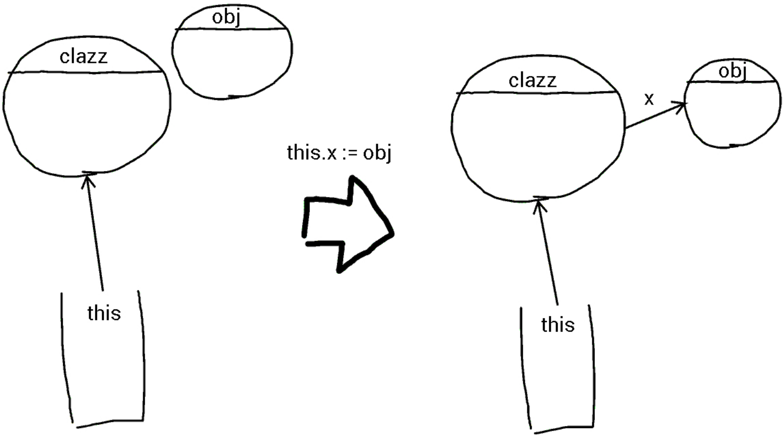
A method is a CFA with the operations on the edges:

- $this.x := \ldots$
- $this.m(\ldots)$
- **new** $O(\ldots)$
- **assume**$(\ldots)$
- **throw** $\ldots$

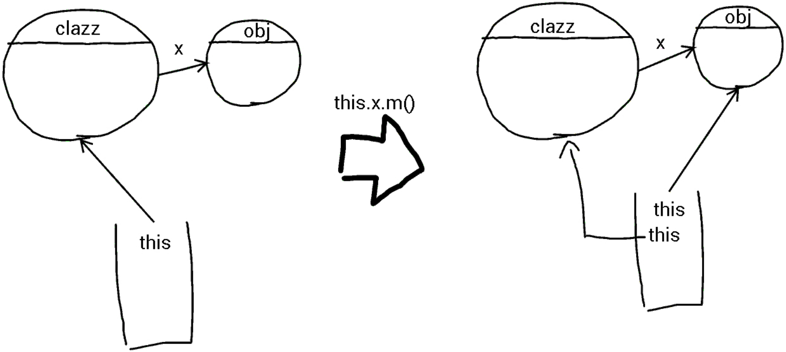
Simple OO model: semantics



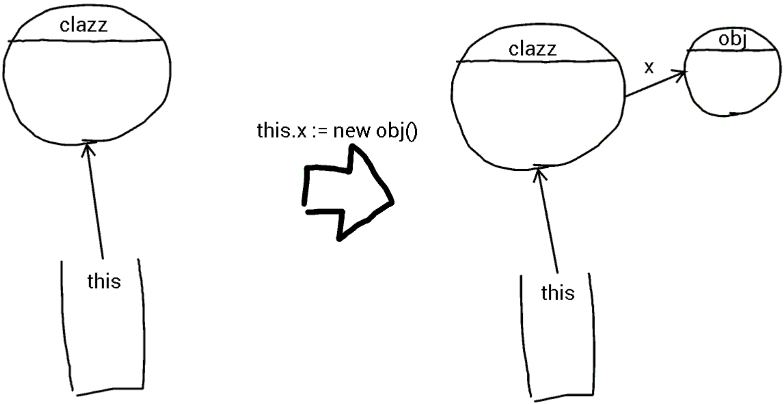
Simple OO model: semantics



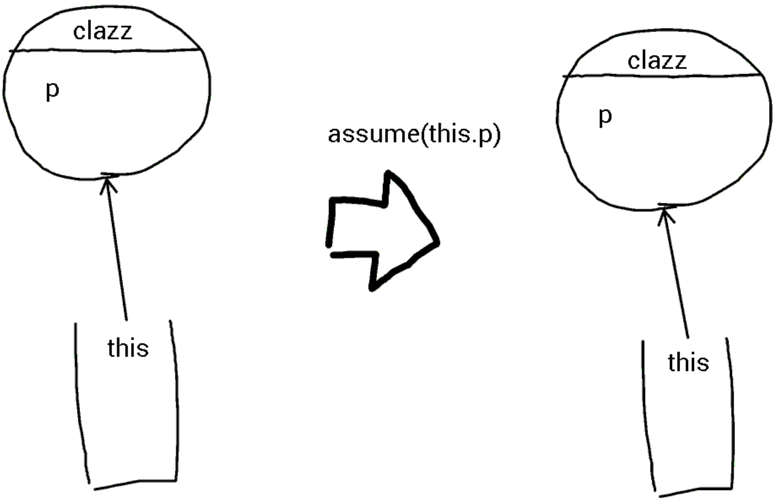
Simple OO model: semantics



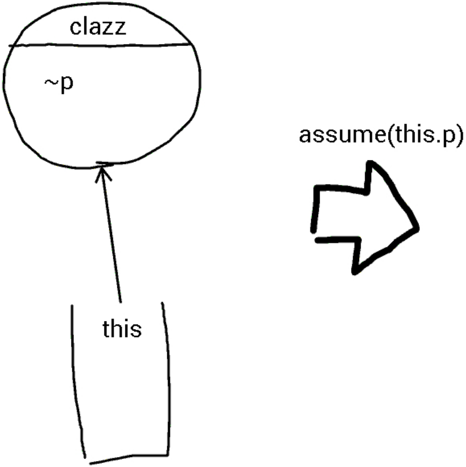
Simple OO model: semantics



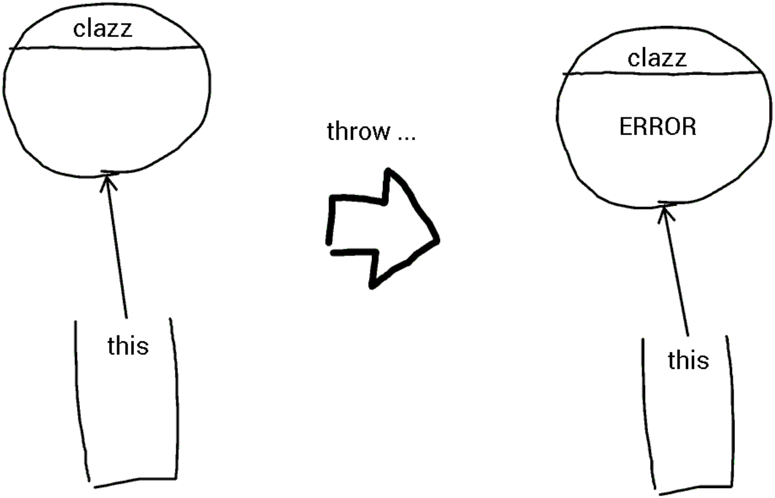
Simple OO model: semantics



Simple OO model: semantics



Simple OO model: semantics



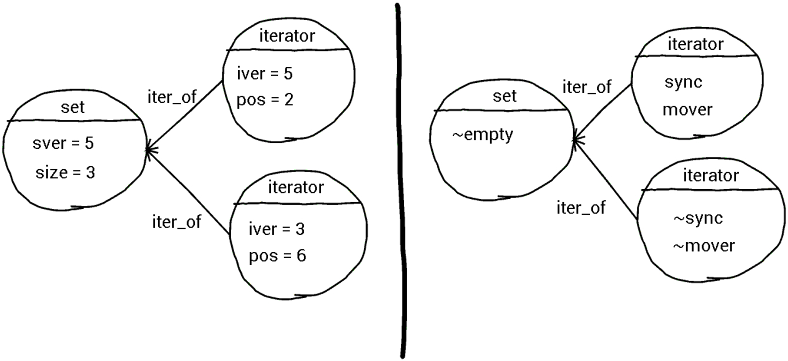
Predicate abstraction

We want interfaces with finite representation.
$\Rightarrow$ cannot remember everything.
$\Rightarrow$ predicate abstraction.

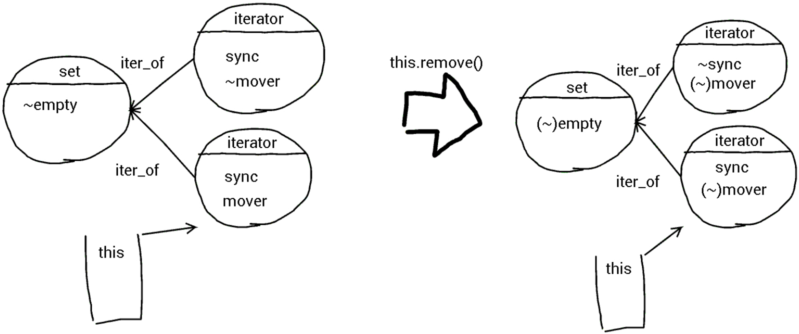
Two classes of predicates:

- unary predicate:
  empty for `Set` (`size = 0`)
- binary predicates:
  sync for `Iterator` (`iver = it_of.sver`)
  mover for `Iterator` (`pos < it_of.size`)

Predicate abstraction



Abstract post



The new predicates can be computed testing (for empty in *set*):

$$\varphi(set, this) \Rightarrow wp(this.\text{remove}, \text{empty}_{set'})$$
$$\varphi(set, this) \Rightarrow wp(this.\text{remove}, \neg\text{empty}_{set'})$$

φ encodes the graph structure as a formula.

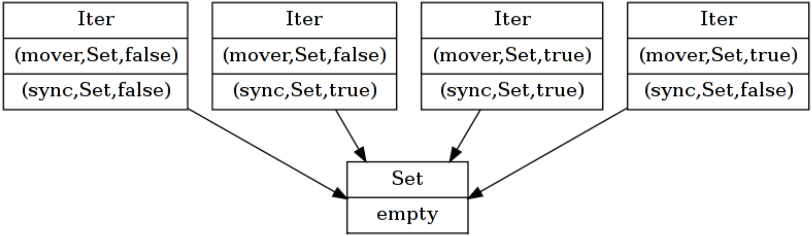
Representing a DPI

State-machine interfaces are automata. We try follow that idea with DPI.

- What are the states ?
- What is the alphabet ?

The states

States are graphs. Nodes represent equivalence classes of objects.

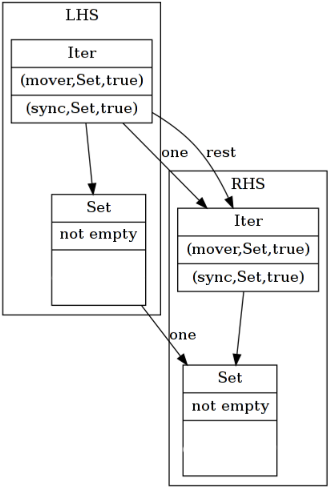


To have a finite number of states in the DPI we require that:

- longest acyclic path in any graph (undirected) is bounded (enforced by a sufficient syntactic restriction).
- there are a finite number of labels (predicate abstraction)
- stratified method call (bounded stack)

The transitions

t_3: Iter(-> Set(Set))[]{(mover,Set,true),(sync,Set,true)}.next



Similar to state-machine interfaces we need an object (*this*) + method name.

Additionally, the transition has a mapping of equivalence classes in the pre-state to the post-state. The systems are monotonic, so any equivalence classes which is not in the mapping is unchanged.

Under the hood: depth-bounded systems

Monotonicity $+$ the conditions on the graph $\Rightarrow$ DBS

Graphs of equivalence classes are ideals in the state space. By a strange coincidence we had a paper about "Ideal abstraction for WSTS". How convenient!

The last element needed to compute a DPI: the *covering set*.

Nice properties of the covering set:

- has a compact representation (finite union of ideals)
- is an inductive invariant (subsumes all the system's behaviors)

Extracting the DPI

- compute the covering set and the DBS
- apply the post operator once more
- track each transition to get the DPI

Show PICASSO output for the Set and Iterators example.

How to use a DPI

Unfortunately, a DPI cannot be used out of the box.
Why? the DPI is saturated, the initial state of the system is empty.

If we know in which equivalence classes the objects belongs, the DPI tells us how to update the state. Some bookkeeping is needed.

Conclusion

We presented:

- DPI as a generalisation of state-machine interfaces to groups of interacting objects.
- Abstraction to compute sound DPI.

Future work:

- Feasibility study for the use of DPI (bookkeeping part).