

# Reasoning about pure methods using effects

**Adam Darvas**

ETH Zurich

Joint work with **Rustan Leino**, **Manuel Fähndrich** and **Peter Müller**

Microsoft Research

ETH Zurich

**Work in progress...**

# Pure methods

```
class Account {
  long balance;

  [Pure] bool PremiumAccount()
    ensures result == balance > 100000;
  { ... }

  void Debit(long amount)
    ensures balance == (PremiumAccount() ? old(balance) - amount :
                        old(balance) - amount - DEBIT_FEE);
  { ... }
}
```

- Convenient and natural to use pure method calls in specifications
- Have to be translated to logic of underlying theorem prover

---

# Pure methods – A verification challenge?

## Encoding of (weakly) pure methods

- [ESC/Java](#)
  - no support for pure methods
- [ESC/Java2](#)
  - handles pure methods since 2004,
  - in an unsound way
- [\[Darvas & Müller, 2005 and 2006\]](#)
  - describes a sound and general handling,
  - clutters resulting formulas significantly
- [Spec#](#)
  - implemented a variation of that which yields simpler formulas (2006)

## Reasoning about pure methods

- Inspector methods [Jacobs & Piessens, 2006]

# Motivation

```
class List {  
  [Pure] bool Has( object o )  
  { ... }  
  
  void Remove( object o )  
  requires Has( o );  
  { ... }  
  ...  
}
```

```
T Foo(List list, string o)  
  requires list.Has( o );  
{  
  ...  
  ...  
  list.Remove( o );  
}
```

Does precondition of call `list.Has(o)` hold?

Well, depends on preceding statements.

# Standard way to reason about it

Use [method specifications](#)

- Pre- and postconditions, and modifies clauses

Use [model fields](#) to prevent breach of information hiding

```
class List {  
  model Set set;  
  [Pure] bool Has( object o )  
    ensures result == o ∈ set;  
  { ... }  
  
  void Remove( object o )  
    requires Has( o ); // o ∈ set  
    modifies set;  
    ensures set == old(set) \ o;  
  { ... }  
  ...  
}
```

```
T Foo(List list, string o)  
  requires list.Has( o );  
{  
  ...  
  ...  
  list.Remove( o );  
}
```

# Problems

- Reasoning with model classes not fully understood
  - [usability of JML](#) model library (complex, circular, sound?)
  - [current research](#) topic: [Schoeller *et al.*, 2006]  
[Julien Charles, 2006]
- Sometimes use of pure methods is more natural than model fields

```
void Debit(long amount)
  ensures balance == (PremiumAccount\(\) ...
```

```
void Foo()
  requires o.M(p) != null;
{
  ...
  ...
  o.M(p).f = 5;
}
```

---

# Our proposal

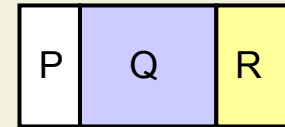
- Use **read** and **write effects** of methods
- Use **data groups** to prevent breach of information hiding
- Keep **track** of possible **changes** of data-group members
- Use **knowledge from previous program points** if known to be unchanged

---

# Data groups [Leino, 1998]

- Set of locations and other data groups
- Used in modifies-clause: permission to modify a group gives the permission to modify its members

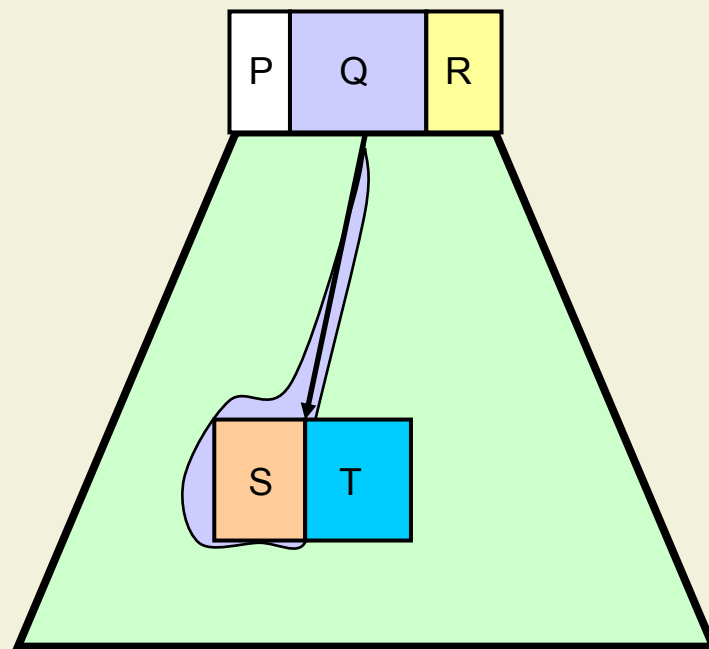
```
class A {  
  DataGroup P, Q, R;  
  int value in P;  
  ...  
}
```



# Data groups [Leino, 1998]

- Set of locations and other data groups
- Used in modifies-clause: permission to modify a group gives the permission to modify its members

```
class A {  
  DataGroup P, Q, R;  
  int value in P;  
  rep B b maps S into Q;  
  ...  
}  
  
class B {  
  DataGroup S, T;  
  ...  
}
```

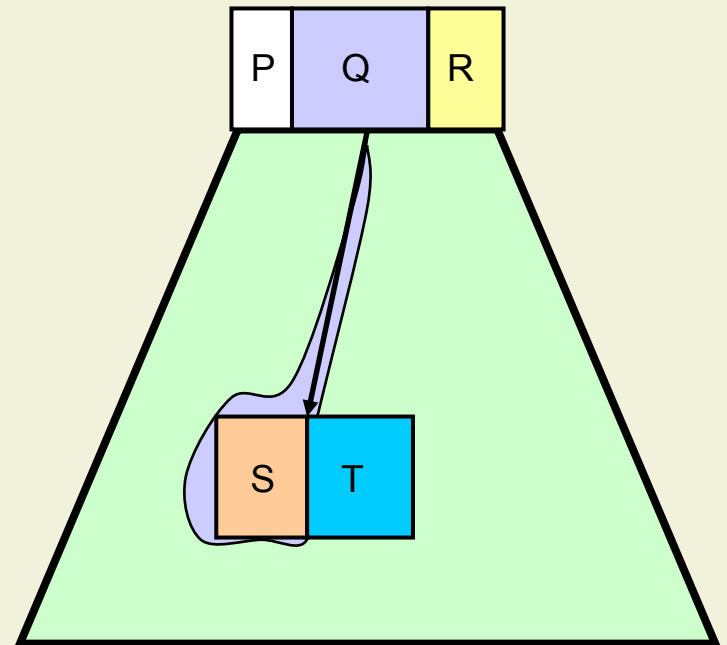


# Read and write effects

- In public specifications use data groups to express read and write effects

```
class A {  
  ...  
  rep B b maps S into Q;  
  ...  
  void Foo()  
    modifies this.R, this.Q;  
  
  [Pure] bool Bar()  
    reads this.P, b.T;  
}
```

No overlap of effects thus can deduce that Foo has no effect on the return value of Bar



---

# Snapshots

Keep track whether members of a data-group might have changed

- Associate a digest, **snapshot**, with every data-group
- **Take new snapshot** whenever a member of data-group might change (field update and method call)
  
- **Theorem:** if the snapshot of a data-group  $o.DG$  is the same at two program points,  $P$  and  $Q$ , then no members of the data-group changed between  $P$  and  $Q$ .
  
- **Consequence:** if the snapshots of all data-groups mentioned in the read-effect of a pure method  $M$  are the same in  $P$  and  $Q$ , then  $M$  yields the same return value in  $P$  and  $Q$ .

Consequence expressed as an axiom for each pure method  $M$ .

# List example revisited

- Annotating List with read and write-effects

```
class List {  
  DataGroup CONTENTS;  
  
  [Pure] bool Has( object o )  
    reads this.CONTENTS;  
  { ... }  
  
  void Remove( object o )  
    requires Has(o);  
    modifies this.CONTENTS;  
  { ... }  
  ...  
}
```

```
T Foo(List list, string o)  
  requires list.Has(o);  
{  
  ...  
  ...  
  list.Remove( o );  
}
```

**Does precondition of call list.Has(o) hold?**

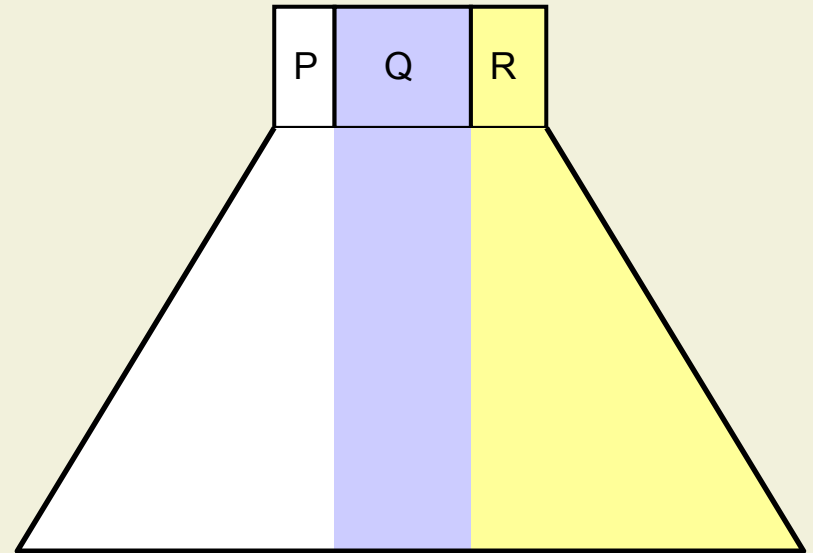
Need to prove that preceding statements do not modify snapshots of list.CONTENTS

# Result

```
[Pure] int ReadP()  
reads this.P;
```

```
void WriteQ()  
modifies this.Q;
```

```
assume o.ReadP() == 5;  
o.WriteQ();  
o.f = 10; // f ∈ R  
assert o.ReadP() == 5;
```



# Result – implemented version

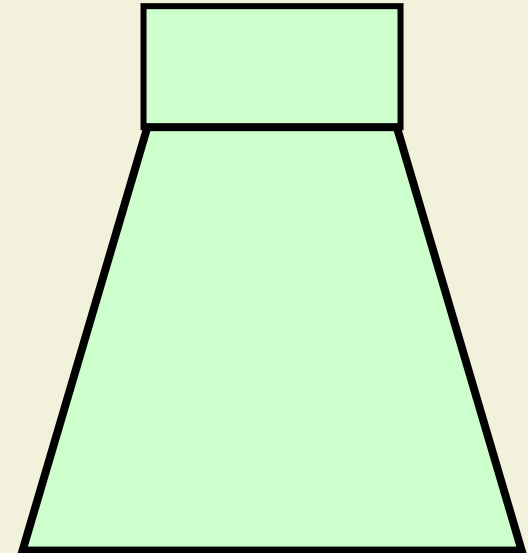
Less fine-grained: one snapshot per object (+ownership cone)

Needs no language extension

```
[Confined] int ReadFG()
```

```
void WriteH()  
  modifies this.h;
```

```
assume o.ReadFG() == 5;  
p.WriteH();  
assert o.ReadP() == 5;
```



Verifies only if  $o \neq p$

---

# Conclusions

- A **reasoning technique** for pure methods
  - data-groups and read-effects vs model fields
  - leads to reduction in completeness
  - works “under the hood”
    - in particular if effects could be inferred
- Compared to **related work** [Jacobs & Piessens, 2006]
  - more **fine-grained** due to data-groups,
  - applicable to Spec#'s heap model,
  - handles (pure) methods
    - in contrast to “*parameter read-confined*” pure methods

---

# Future work

- Extend approach to peers
  - Exact formalization and soundness proof
  - Implementation in Spec#
  - Experiment usability (power of Simplify)
- 
- See if approach can be realized outside Boogie methodology
- 
- **Acknowledgement:** idea of snapshot from Bart Jacobs