Connecting Effects and Uniqueness with Adoption

John Boyland
Connecting Effects and Uniqueness (1 of 2)

❖ Effects: [Greenhouse and others, 1999-2003]
  – Each method is annotated with state that it accesses.
  – Fields are aggregated into “data groups.”

❖ Uniqueness: [Boyland and others, 1998-2001]
  – (Active) unique fields have no (active) aliases;
  – Each method parameter, receiver and return value is:
    • UNIQUE - no (active) aliases exist;
    • BORROWED - no (lasting active) aliases will be made;
    • SHARED - no guarantees, no restrictions.

❖ [Related] Ownership
Effects depends on uniqueness (or ownership):
- Effects on unique (owned) sub-objected mapped into “this.”

Uniqueness depends on effects:
- How do we know if a unique field is active? (read effect):
  ```java
  Object[] contents = this.contents;
  int i = find(e, contents);
  ```
  ```java
  While find is looking at contents, it is possible that this object could be called upon to sort the elements?
  ```

Our solution:
- Express both effects and uniqueness with “permissions.”
Previous Solutions:

- ESC/Java [Leino and others, 1998-2003]
  - “Owner exclusion” + uniqueness not transferable.

- AliasJava [Aldrich and others, 2002]
  - Don’t worry about effects (aliases on stack tolerated).

- PRFJ [Boyapati and others, 2002-3]
  - Ownership is main metaphor (uniqueness partly supported);
  - Coarse-grain effects (object-level).

- External Uniqueness [Clarke and Wrigstad, 2003]
  - Ownership extended to fields + destructive reads.
Goal: Unify Effects and Uniqueness

Given the close connection:
- What are the unifying principles?
- Can we express our previous ideas using these principles?
- Does the expression have interesting implications?

Answers:
- Permissions (extended “adoption and focus”)
- Yes (see later)
- Yes (see later)
As with ownership, objects are nested in other objects:

(Non-linear pointers cannot be used to access data.)
Solution? Adoption & Focus (2 of 4)

- One can “focus” on non-linear ptr (and give up “adopter”)

(Now one can get at, and change, the (linear) history.)
When done, one gives up “adoptee” and regains “adopter”

The old history array can be gc’ed (or used elsewhere).
Effects are represented with permission keys:
- The bank cannot be accessed without key;
- A method needing to access the bank declares this need:
  - a method effect annotation.

Uniqueness is represented by (encapsulated) keys
- Only one key per object;
  - key to history array stored in account;
- Linearity is transferable.

“Free gifts”: ownership can be modeled by adoption.
Also a mutex can contain the key for protected state.
Problems with Adoption & Focus

1. Read and write effects aren’t distinguished.

2. Can’t focus on two adoptees at once:
   - Whole bank closed while examining one account;
   - To compare two accounts, you have to make copies.

3. Can only adopt whole objects:
   - History must be separate object (in addition to array);
   - Can’t independently work on “balance” and “owner”.

4. Null pointers break the type system.
   - the null pointer is non-linear, and no one has its key.
1. “Fractional Permissions” [Boyland, 2003]

2. Focus for multiple distinct adoptees permitted.

3. Adoption moved to apply to fields, not objects:
   – New unit-typed fields used to model “data groups”;
   – Default data group “All” used to model whole object’s state.

4. Conditional permissions:
   – A permission is available once a pointer is provably non-null.
   – Any pointer comparison can be used in condition:
     • e.g. circular lists
Contribution (1): Extending Adoption & Focus (2 of 4)

- Permissions separate from pointers
When we “focus,” we “carve out” permission:

Multiple adoptees can be carved out at once.
We can focus on a particular field:

Now we can access the history arrays.

University of Wisconsin-Milwaukee
Contributions (2): Expressing Annotations (1 of 2)

○ Data groups:
  – Fields (of unit type) that “adopt” (or “own”) other fields;
  – (Close correspondence to how ESC/Java represents them.)

○ Method effects:
  – Permissions that are passed to a method and returned;
  – (As before, effects not needed for objects to be created.)

○ Reference annotations:
  – unique = comes with own permissions;
  – shared = permission is adopted into global place;
  – borrowed = permissions only available in method effects.
Define a type “function” for each class:
- Gives type for every field and where adopted into;
  Established by constructor (partial = “raw”)

Extra reference annotations: (“free with purchase!”)
- owned-by X = permissions adopted into X;
- non-null = permission not conditional (should be default)

Annotations made possible with fractional permissions:
- immutable = comes with own fractional permission;
- read-only = fractional permission adopted into global place;
- unique-write = ditto plus remainder available here.
Implication: A Field Can Belong to Multiple Data Groups

However, as with “internet adopters”, you can’t satisfy both

Linearity (uniqueness) is preserved.
Implication: Aliases Don’t Need to be Buried.

- Alias burying:
  - when a unique field is read, all aliases must be dead;
  - no doubly-linked lists; no iterators; ...

- Better to distinguish the two concepts
  - pointer to state;
  - permission to access that state.

- Relaxation of alias burying:
  - when a unique field is read, one gets sole permission.

  We get “external uniqueness.” without destructive reads.
  “(some assembly req’d)”
Implication: Borrowed Parameters are not Aliases

- Method effects include state to access:
  - Uniqueness of permissions ensures separate state.
    E.g.: Hashtable.put(key, value) requires:
    - write access to hash table;
    - read access to key.
  - h.put(h, x) does not type-check.

- “Owner exclusion” [Leino and others, 2002]
  - “Cannot pass a unique field’s value to routine that may modify the owning container.”
  - Implied by effects:
    - bad calls simply do not type-check.
Implication: Iterators Adopt Content Permission (1 of 8)
Implication: Iterators Adopt Content Permission (2 of 8)
Implication: Iterators Adopt Content Permission (3 of 8)
Implication: Iterators Adopt Content Permission (4 of 8)
Implication: Iterators Adopt Content Permission (5 of 8)
Implication: Iterators Adopt Content Permission (6 of 8)
Implication: Iterators Adopt Content Permission (7 of 8)
Implication: Iterators Adopt Content Permission (8 of 8)
Caveats and Further Work

❍ Theory not done:
  – Even a couple of bugs in type rules in (submitted) paper;
  – Proofs not complete.

❍ Implementation just starting:
  – Uses alias types and flow-sensitive typing;
  – Probably undecidable in general;
  – Are implementable heuristics usable?

❍ Extensions to locking still being defined.
Conclusions

- Uniqueness and effects are interdependent
  - Checking either requires the other.

- Permissions (an extension to Adoption&Focus)
  - unifies uniqueness and effects;
  - simplifies reasoning;
  - has the potential to be proven correct;
  - enables interesting extensions.