ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Uniformity Hypotheses and Subdomain Decomposition in Testing Object-Oriented Software

Stéphane Barbey

Stephane.Barbey @ di.epfl.ch

Swiss Federal Institute of Technology in Lausanne
Software Engineering Laboratory

January 1997
URL http://lglwww.epfl.ch/~barbey

Overview

Uniformity Hypotheses

1. Goal, definition and application
2. Kinds of uniformities
3. Uniformity on objects

Subdomain Decomposition

4. Why, where and how to find subdomains
5. COOPN/2 constructs
6. Conjunction of Subdomains

7. Conclusion

Uniformity Hypotheses

Goal

Reduction for variables in a HML formula. (*Var*(ϕ) is the set of variables in the formula ϕ.)

Uniformity Hypothesis

If a test of a formula ϕ, with $v \in Var(\phi)$, is successful for a given value of *v*, then it is successful for all possible values of *v*.

Application

Selecting a unique instance of each variable *v* in *Var* (ϕ).

Uniformity Hypotheses

<<V1> <card.create V3> <telephone.insert V4> V2, Result>

- uniformity on events (V1)
- uniformity on HML formulae (V2)
- uniformity on algebraic values (V3)
  - applying a random conjunction of the functions defined in the corresponding ADT
  - selecting generator functions only.
- uniformity on objects (V4)

Uniformity on Objects

- An object in any state can be generated.
- Generating randomly a sequence of events including first a creation method, and then a (possibly empty) sequence of methods.
- This sequence should include modifiers only.
- This sequence must be injected in formula in construction at a place before the event containing the object to which the uniformity is applied
- This strategy can be further restricted to creation operations.

Uniformity on Objects

Example

Given

<<t.create><t.insert C>T, Result>

where C is an object of type PhoneCard.

Applying a uniformity hypothesis on C results in:

<<t.create>*<seq. of events on C>*<t.insert C>T, true>

<<t.create>*<c.create 1234>*<t.insert c>T, true>
<<t.create>*<c.create 1234><c.withdraw 20>*<t.insert c>T, true>
<<t.create>*<c.create 1234><c.withdraw 12><c.withdraw 4>*<t.insert c>T, true>

Uniformity Hypotheses

- Uniformity hypothesis are very strong.
  - The coverage of the tested unit could be very low.
  - When a uniformity hypothesis selecting an unique instance of the code does not cover all cases described in the specification.

```
(pp = p) = true => enter p With c.get-pin pp :
              wait-for-pin c s -> wait-for-buy c s;
(pp = p) = false => enter p With c.get-pin pp :
              wait-for-pin c s -> ready-to-eject c s;
```

- Use subdomain Decomposition

Subdomain Decomposition

- Whenever constraints exist on the domain of any free variable v of ϕ, it is needful to decompose the domain $D(v)$ of each free variable v in subdomains $D_i(v)$, such that $D(v) = \cup_i D_i(v)$.

- Subdomains may be disjoint or overlap.

- The subdomains that occur are handled in conjunction to derive a set of constraints on which uniformities are then applied to select ground formulae, by applying uniformity hypothesis on each subdomain D_i.

- For hybrid object-oriented software, variables with domain are algebraic values (of ADTs) and objects (of classes),

Where to Find Subdomains

Axiom$_i$: Condition $\Rightarrow$ Event({Parameter$_j$}) **With** SynchroExpression: Precondition $\rightarrow$ Postcondition

- `Axiom`$_i$
  Distinct axioms
- `Condition`
  Algebraic conditions on behavioral axioms
- `Event ({Parameter`$_j$`})`
  Method parameters
- **`With`** `SynchroExpression`
  Synchronization expressions on other objects
- `Precondition` $\rightarrow$ `Postcondition`
  Pre- and postconditions by means of the algebraic semantics

How to Find Subdomains

$$\langle\langle C.\text{Create (V0)}\rangle\langle C.\text{Withdraw (V1)}\rangle\langle C.\text{Get-Balance (V2)}\rangle T, Result\rangle$$

$$R1: \frac{\dfrac{\dfrac{}{\textsc{Class-Sem}\ \langle loid,\bot\rangle \xrightarrow{\text{C.Create (V0)}} \langle loid',(\text{id V0, balance 20})_C\rangle}}{\textsc{Stab}\ \langle loid,\bot\rangle^* \xrightarrow{\text{C.Create (V0)}} \langle loid',(\text{id V0, balance 20})_C\rangle} \quad \dfrac{\dfrac{}{\textsc{Object-Sem}\ \langle loid'',(\text{balance B1})_{C'}\rangle \xrightarrow[(\gamma_1)]{C'.\text{Withdraw (V1)}} \langle loid'',(\text{balance B1 - V1})_{C'}\rangle}}{\textsc{Monotonicity}\ \langle loid'',(\text{id I1, balance B1})_{C'}\rangle \xrightarrow[(\gamma_1)]{C'.\text{Withdraw (V1)}} \langle loid'',(\text{id I1, balance B1-V1})_{C'}\rangle}}{\dfrac{\textsc{Beh-Seq}\ \langle loid,\bot\rangle \xrightarrow[(\gamma_1,\sigma_1)]{\text{C.Create (V0) .. C.Withdraw (V1)}} \langle loid',(\text{id V0, balance 20-V1})_C\rangle}{\textsc{Stab}\ \langle loid,\bot\rangle^* \xrightarrow[(\gamma_1,\sigma_1)]{\text{C.Create (V0) .. C.Withdraw (V1)}} \langle loid',(\text{id V0, balance 20-V1})_C\rangle}}$$

$$\frac{R1 \quad \dfrac{\dfrac{}{\textsc{Object-Sem}\ \langle loid''',(\text{balance V2})_{C''}\rangle \xrightarrow{C''.\text{Get-Balance (V2)}} \langle loid''',(\text{balance V2})_{C''}\rangle}}{\textsc{Monotonicity}\ \langle loid''',(\text{id I2, balance V2})_{C''}\rangle \xrightarrow{C''.\text{Get-Balance (V2)}} \langle loid''',(\text{id I2, balance V2})_{C''}\rangle}}{\dfrac{\textsc{Beh-Seq}\ \langle loid,\bot\rangle \xrightarrow[(\gamma_1,\sigma_1,\sigma_2)]{\text{C.Create (V0) .. C.Withdraw (V1) .. C.Get-Balance (V2)}} \langle loid',(\text{id V0, balance 20-V1})_C\rangle}{\textsc{Stab}\ \langle loid,\bot\rangle^* \xrightarrow[(\gamma_1,\sigma_1,\sigma_2)]{\text{C.Create (V0) .. C.Withdraw (V1) .. C.Get-Balance (V2)}} \langle loid',(\text{id V0, balance 20-V1})_C\rangle}}$$

$$\sigma_1 = \begin{pmatrix} loid' = loid'' \\ C = C' \\ V0 = I1 \\ 20 = B1 \end{pmatrix} \qquad \sigma_2 = \begin{pmatrix} loid' = loid''' \\ C = C'' \\ V0 = I2 \\ V2 = 20 - V1 \end{pmatrix} \qquad \gamma_1 = (B1 \geq V1)$$

Algebraic Conditions on Behavioral Axioms

- Each algebraic condition on an axiom will usually draw up two different subdomains, one for each possible result of the condition.

(b >= m) = true => withdraw m : balance b -> balance b - m;

two cases: (b >= m) = true, (b >= m) = false

- If there are distinct axioms for one event, subdomains for each axiom can possibly be overlapping.

(pp = p) = true => enter p **With** c.get-pin pp :
wait-for-pin c s -> wait-for-buy c s;
(pp = p) = false => enter p **With** c.get-pin pp :
wait-for-pin c s -> ready-to-eject c s;

two cases: (pp = p) = true, (pp = p) = false

Algebraic Conditions on Behavioral Axioms

- According to the way algebraic conditions are written, it can possibly lead to more cases.

(b >= m) = true => withdraw m : balance b -> balance b - m;

Three cases: (b >= m) = true ⇒ b = m, b > m; (b >= m) = false ⇒ b < m

- Thus, it does only rely on the semantics of the condition, but also on the way it is written.

- Unfolding complex conditions will also lead to more cases,

((b >= m) and (pp=p)) = true => withdraw m p:
balance b, id pp -> balance b - m, id pp;

Six cases:

$(b=m\ ,\ b>m,\ b<m) \times (pp=p,\ pp\neq p)$

Method Parameters

- Method parameters are handled like algebraic conditions.

- For example, the axiom

  event (3) **With** ... : ...

  is the same as

  (v=3) = true => event(v) **With** ... : ...

Synchronization Expressions on Other Objects

- Many cases
  - State of the connected object

    ax_insert_1:(b >0) = true => insert c **With** c.get-balance b :
            idle s -> wait-for-pin c s;
    ax_insert_2:(b >0) = false => insert c **With** c.get-balance b :
            idle s -> ready-to-eject c s;

    <<t.create>*<seq. of statements of C>*<t.insert C>T, true>

  - Possibility of triggering, e.g.

    ax_next_1: next_step **With** o.meth1 + o.meth2: ...

    ax_last_1: last_step **With** o1.meth1: ...
    ax_last_2: last_step **With** o2.meth2: ...

- Open issue

Pre- and Postconditions

- Preconditions give conditions on the marking before triggering.

- Pre- and postconditions are *cumulative*, and allow to find constraints on sequences of events.

$$m \xrightarrow[m \to m - pre_e + post_e]{e\ (v)} m' \xrightarrow[m' \to m' - pre_f + post_f]{f\ (v)} m''$$

$$m'' = m - pre_e + post_e - pre_f + post_f$$

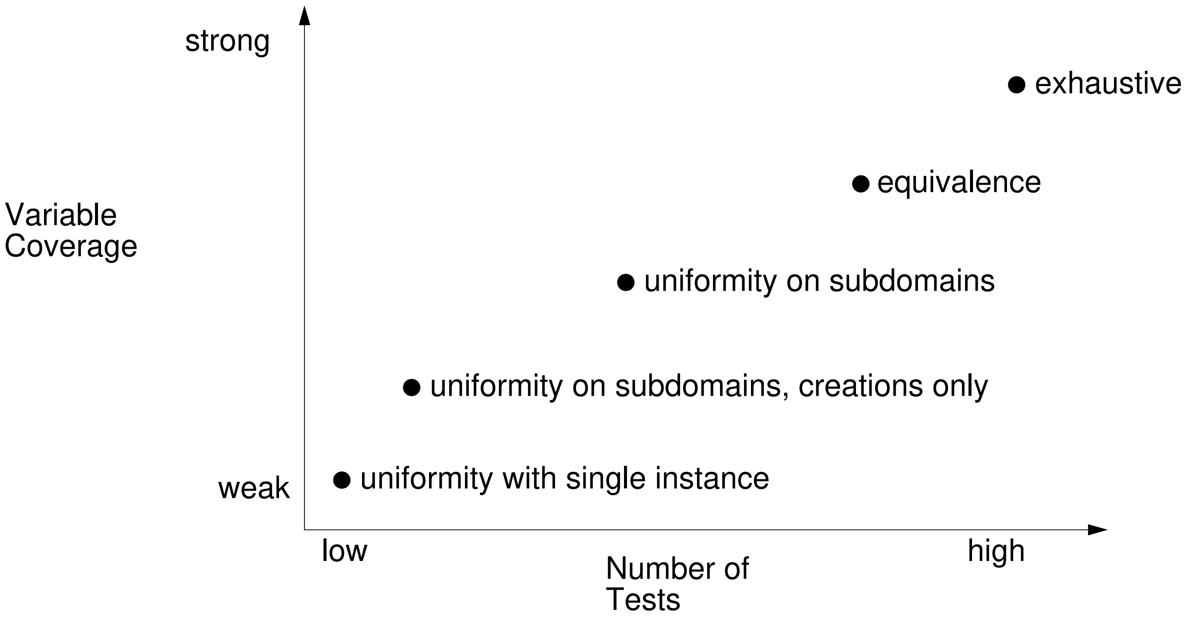
Conjunction of Subdomains

<<c.create V0><c.withdraw V1><c.get-pin V2><c.get-balance V3>T, Result>

- no constraint on V0
- two cases for V1 (condition: 20 ≥ V1), possibly three
- two cases for V2 (condition: V0 =V2)
- two cases for V3 (condition: V3 = 20 - V1)

1: (V0, 20≥V1, V0=V2, 20–V1=V3,
<<c.create 1234><c.withdraw 12><c.get-pin 1234><c.get-balance 8>T, true>)
2: (V0, 20≥V1, V0=V2, 20-V1≠V3,
<<c.create 4123><c.withdraw 6><c.get-pin 4123><c.get-balance 5>T, false>)
3: (V0, 20≥V1, V0≠V2, V3,
<<c.create 1111><c.withdraw 17><c.get-pin 1234><c.get-balance 3>T, false>)
4: (V0, 20<V1, V2, V3,
<<c.create 5873><c.withdraw 50><c.get-pin 5873><c.get-balance 6>T, false>)

Uniformity and Subdomains



Conclusion

- Uniformity hypotheses drastically reduce a test set, but they usually have a low coverage of the code
- Subdomain decomposition can be applied to give a better coverage, but requires an analysis of all constructs in a behavioral axiom
- Thus subdomains are complex to find.

Future work

- Synchronization
- Observability (oracle)
- Systematization/Automatization
- Quality of the produced tests