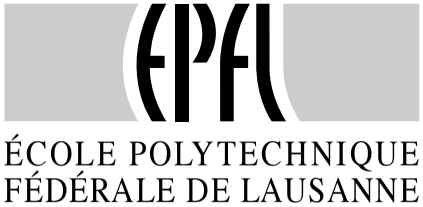


Testing in a World of Classes and Class Hierarchies

Stéphane Barbey

Stephane.Barbey @ di.epfl.ch

Swiss Federal Institute of Technology in
Lausanne
Software Engineering Laboratory

December 1995

URL http://lglwww.epfl.ch/Team/SB

© 1995 Stéphane Barbey

Overview

1. What is testing?

Testing Object-Oriented Software

2. Unit testing for object-oriented software
3. Advantages and drawbacks

The Theory of Testing

4. The Test Procedure
5. The Reduction Hypotheses

Conclusion

Testing is...

Definition

Testing is the process of finding programmers' errors or program faults in the behavior or in the code of a piece of software.

Goal

Testing is used to gain confidence that the implementation of a program meets (or does not meet) its specifications.

Testing is not...

- Testing cannot show that errors are not present.
- Testing is not only showing that a program does what it is intended to do.
- Testing is not the process of correcting errors.

Object-Oriented Software Testing vs. Traditional Testing

Issue

Some good methods of testing have already been developed with “procedure-oriented” software in mind.

Problem

This decomposition is ill-fitted for object-oriented software.

- The basic unit of test is not the function, but the class.
- An operation is not a testable component. (An instance of a class must pre-exist).

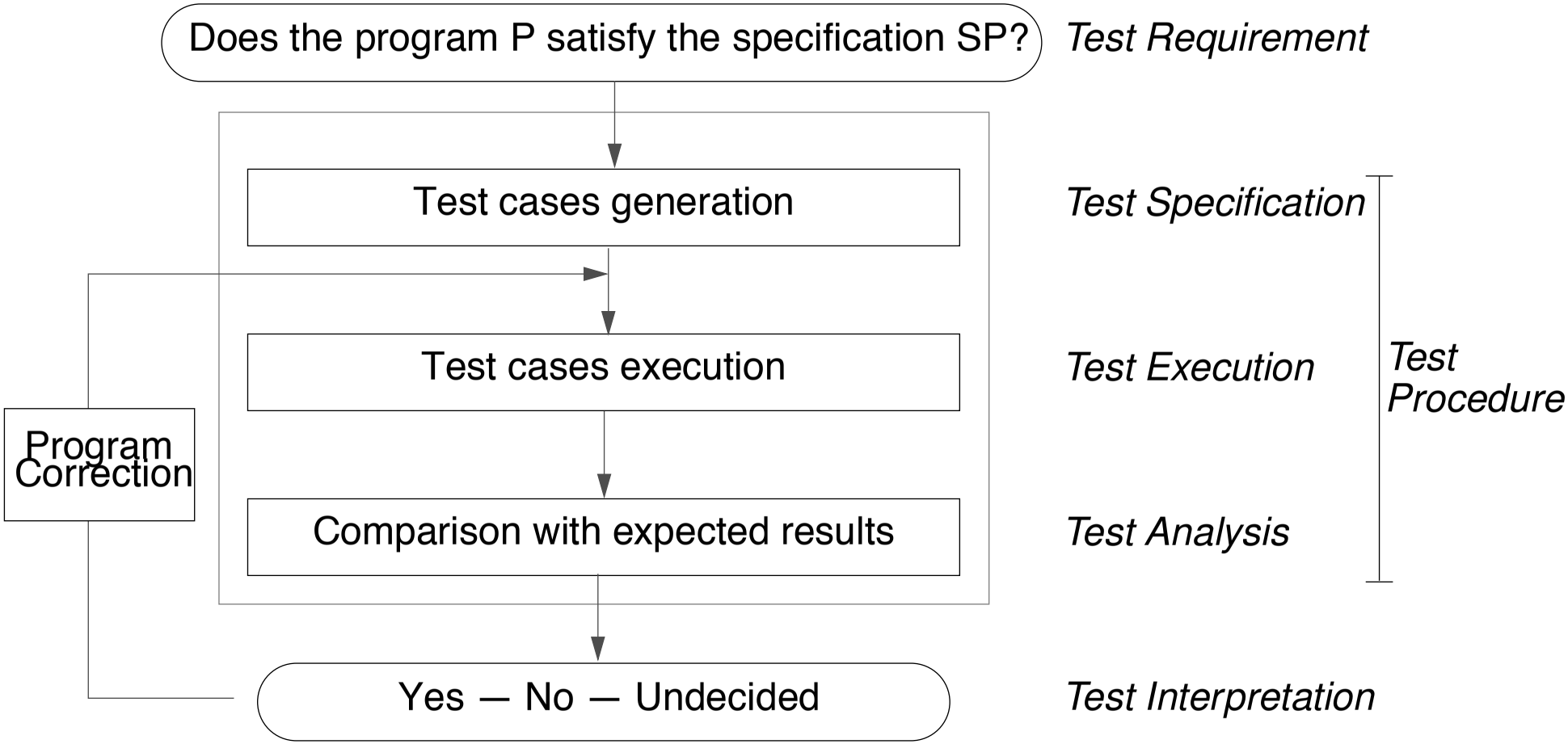
© 1995 Stéphane Barbey

Advantages and Drawbacks of Testing Object-Oriented Software

(compared to traditional software)

Advantages	Drawbacks
• encapsulation • operations are bound to types and not freely floating • the object-oriented paradigm unifies language constructs • operations are shorter (average 20 lines of code per operation in the Smalltalk library) • **incremental inheritance for derived types**	• use of state variables • use of references with other objects • equality defined in terms of identity and value • run-time binding (no static analysis) • substitutability

The Test Procedure



© 1995 Stéphane Barbey

The Theory of Testing

- **a theory of test selection**

"How to select test cases?"
finite - complete - valid - unbiased - decidable

- **a theory of test satisfaction**

"Verify the success of test cases?"
an equivalence relationship

© 1995 Stéphane Barbey

Structure of a Test

Result := Object.Operation (Arguments) ⟹

Final_State (Object) ≡ Expected_Final_State (Object) ∧
Result ≡ Expected_Result

In Ada

procedure Operation (Object: **in out** ...; Arguments: ...; Results: [**in**] **out** ...) ⟹

Controlling operand *Other operands* *Results*

Is_Equivalent (Final_State (Object), Expected_Final_State (Object)) **and**
Is_Equivalent (Result, Expected_Result)

© 1995 Stéphane Barbey

The Test Set Selection

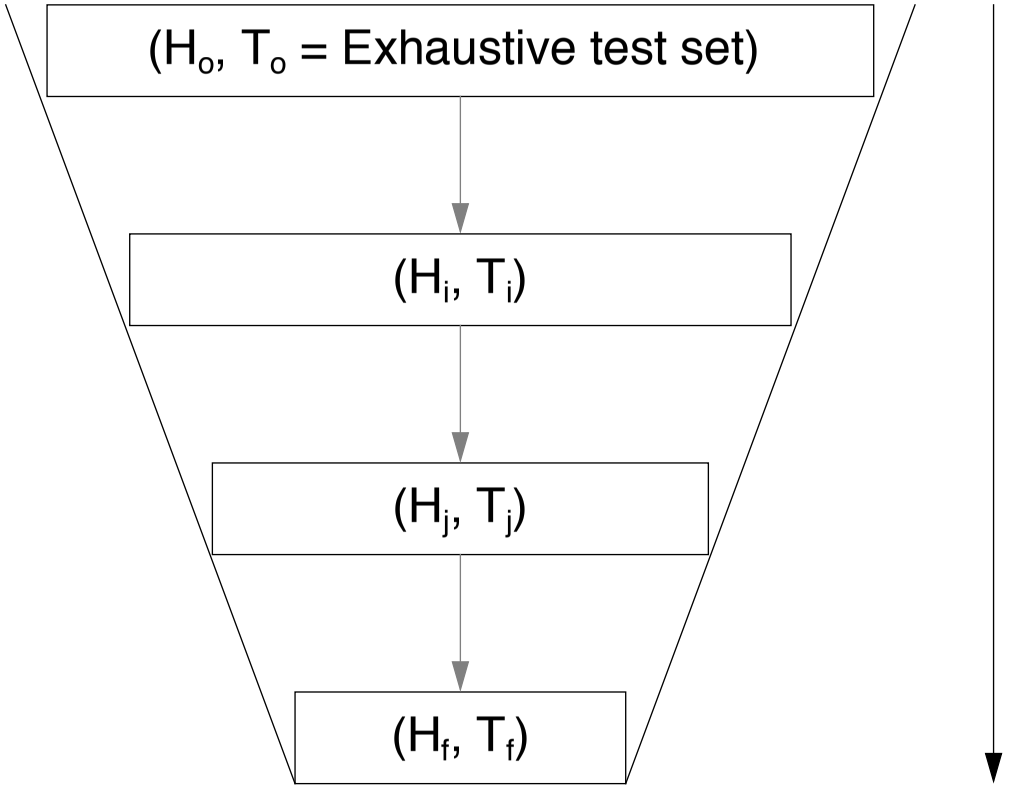
To select a finite test set which is sufficient under hypotheses to prove the property preservation of a program with respect to its specification.

Process

- To construct an exhaustive test set, and
- To reduce it by applying reduction hypotheses.

The goal of the selection is to get tests showing counter-examples to the program validity, i.e. to exhibit the presence of errors.

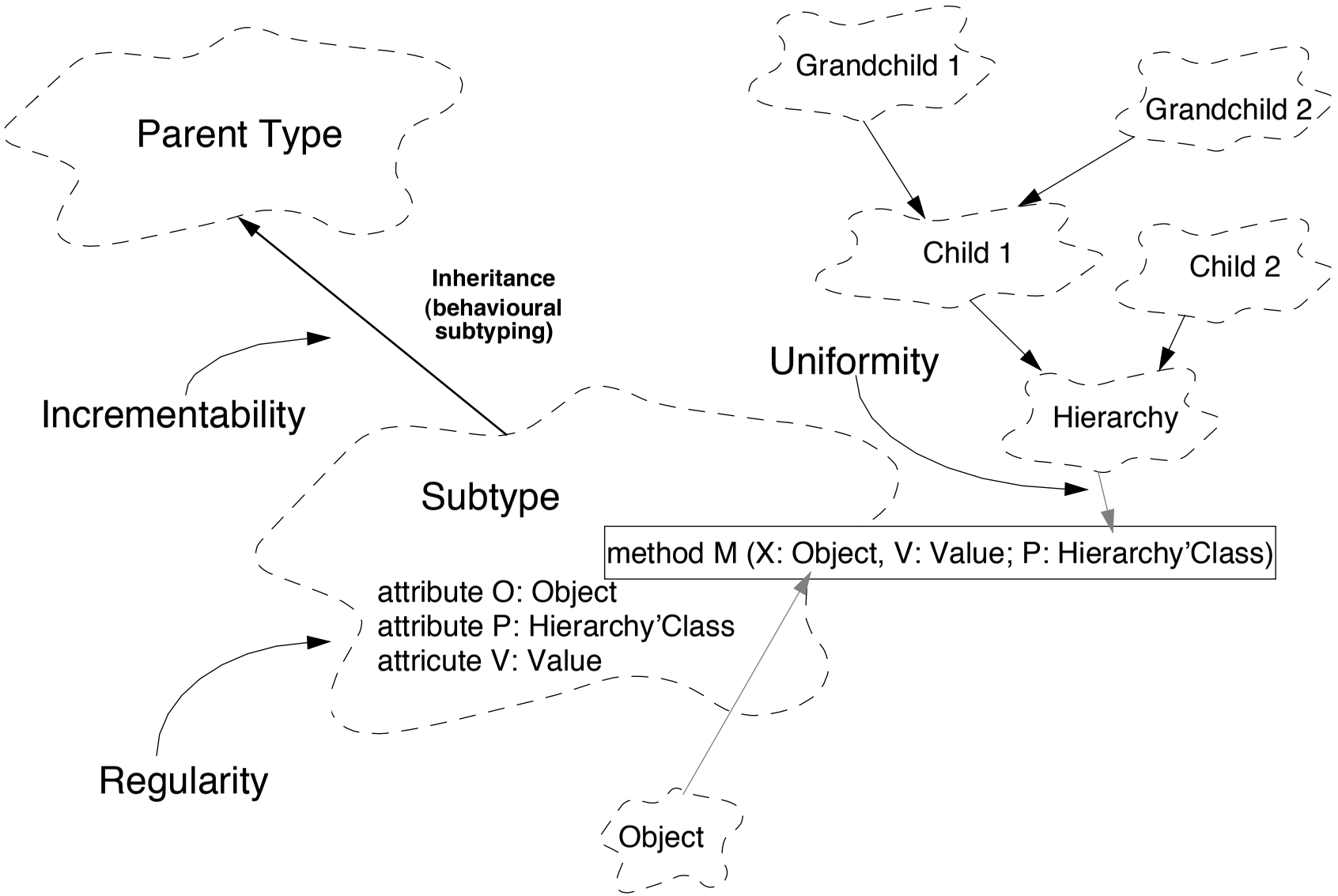
Iterative refinement of the test context



Reduction of the test set by application of the reduction hypothesis

© 1995 Stéphane Barbey

The Reduction Hypotheses



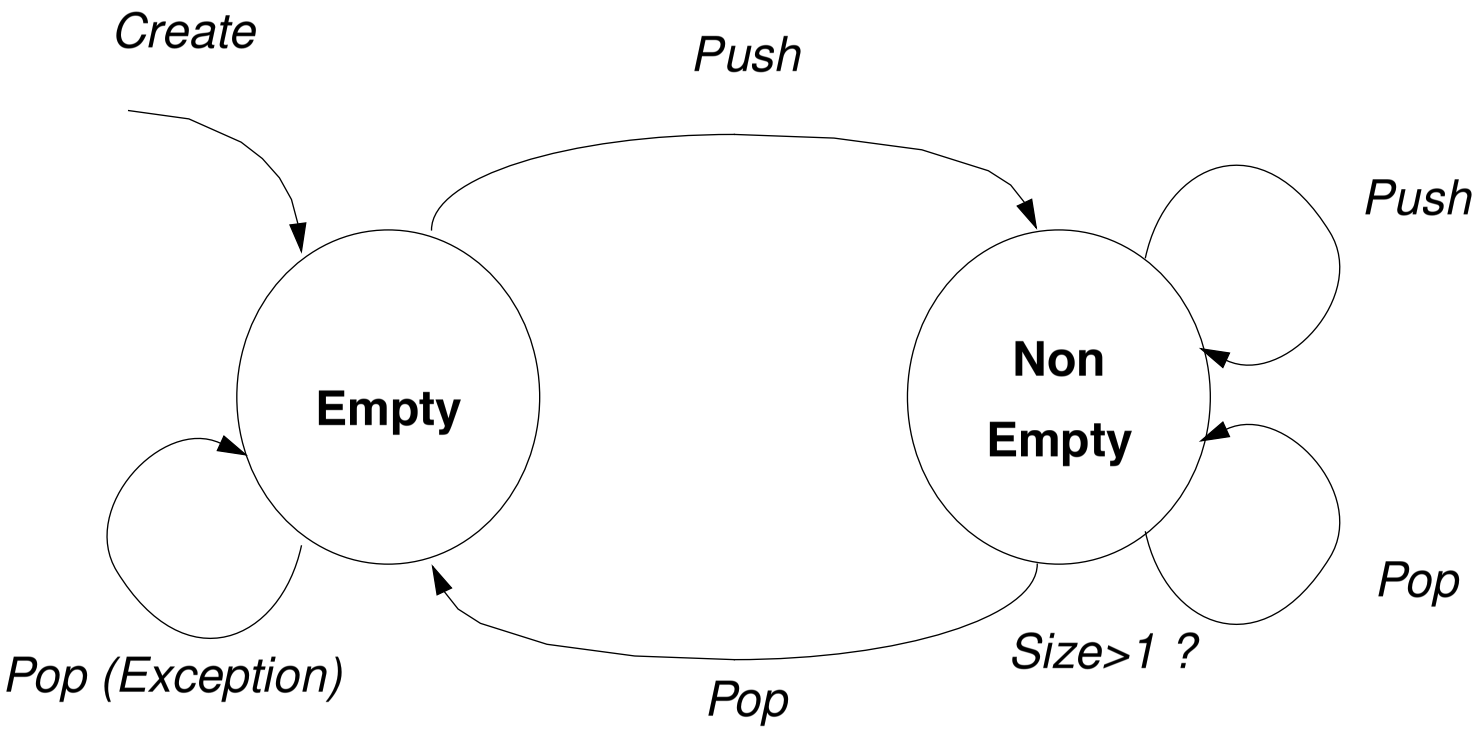
© 1995 Stéphane Barbey

The Regularity Hypothesis

If a program behaves correctly for a property A instantiated to a set of terms, then it should behave correctly for all possible instantiations of the same complexity.

- The possible states of an object can be grouped in state equivalence classes.
- Each member of an equivalence class should be behaviourally equivalent.
- Test sets can be drastically reduced by finding the minimum set of those equivalence classes.
- Then, each operation is tested with all possible starting states and all transactions are exercised.

The Regularity Hypothesis



© 1995 Stéphane Barbey

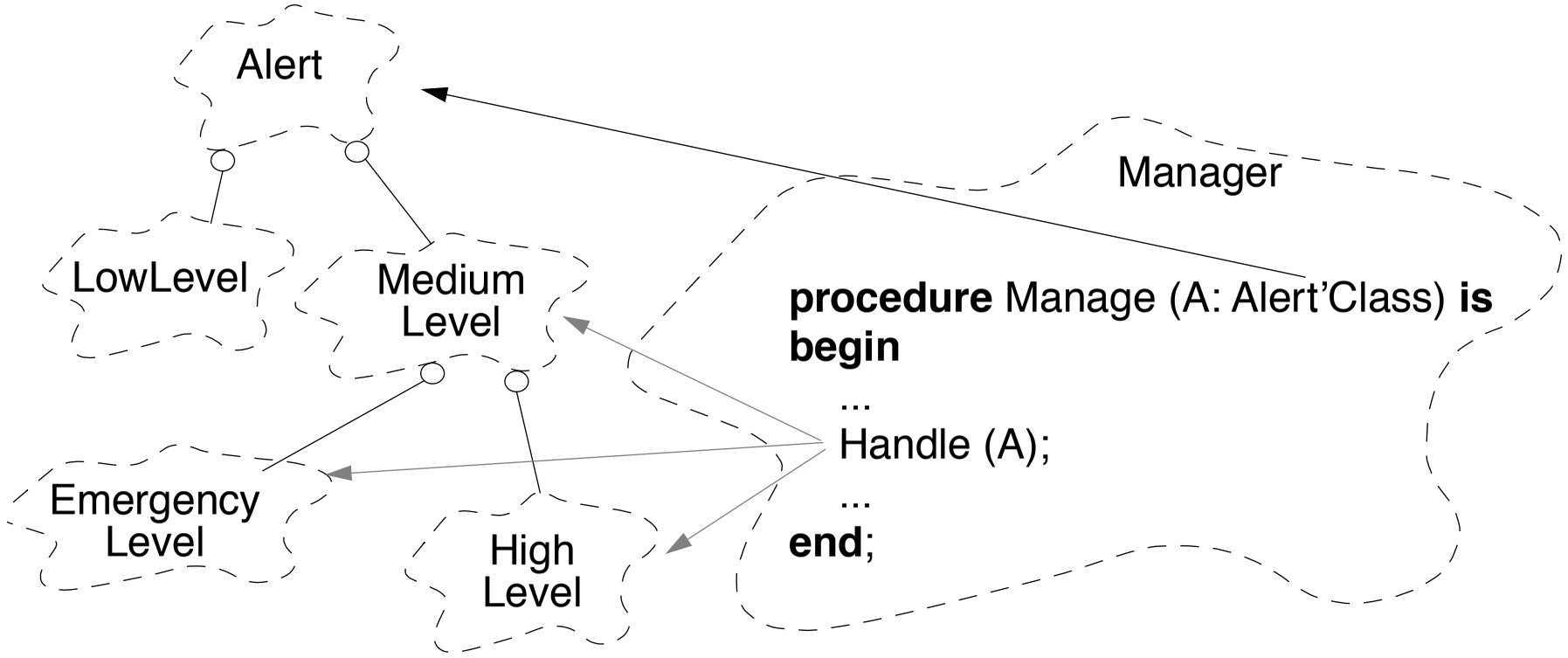
The Uniformity Hypothesis

If a program behaves correctly for a property A instantiated with a particular term T of a type S, then every instantiation of the property A with terms of the type S behaves correctly.

- As a consequence, if the property A is correct for one specific instantiation, it should be correct for all possible instantiations.
- This hypothesis corresponds to random testing.
- However, it is not applicable in the case of a (non-primitive) polymorphic term T belonging to a type S or one of its subtypes.

The Uniformity Hypothesis

- The items pushed on the stack have no incidence on the behaviour of the stack.
- However, there can be consequences if the behaviour of the tested method depends on the parameter.



© 1995 Stéphane Barbey

Subtyping

- **Conformance of behaviour**

  A type S is a *subtype* of T if S provides at least the functionality (c*onformance of behaviour*) of T.

- **Substitution**

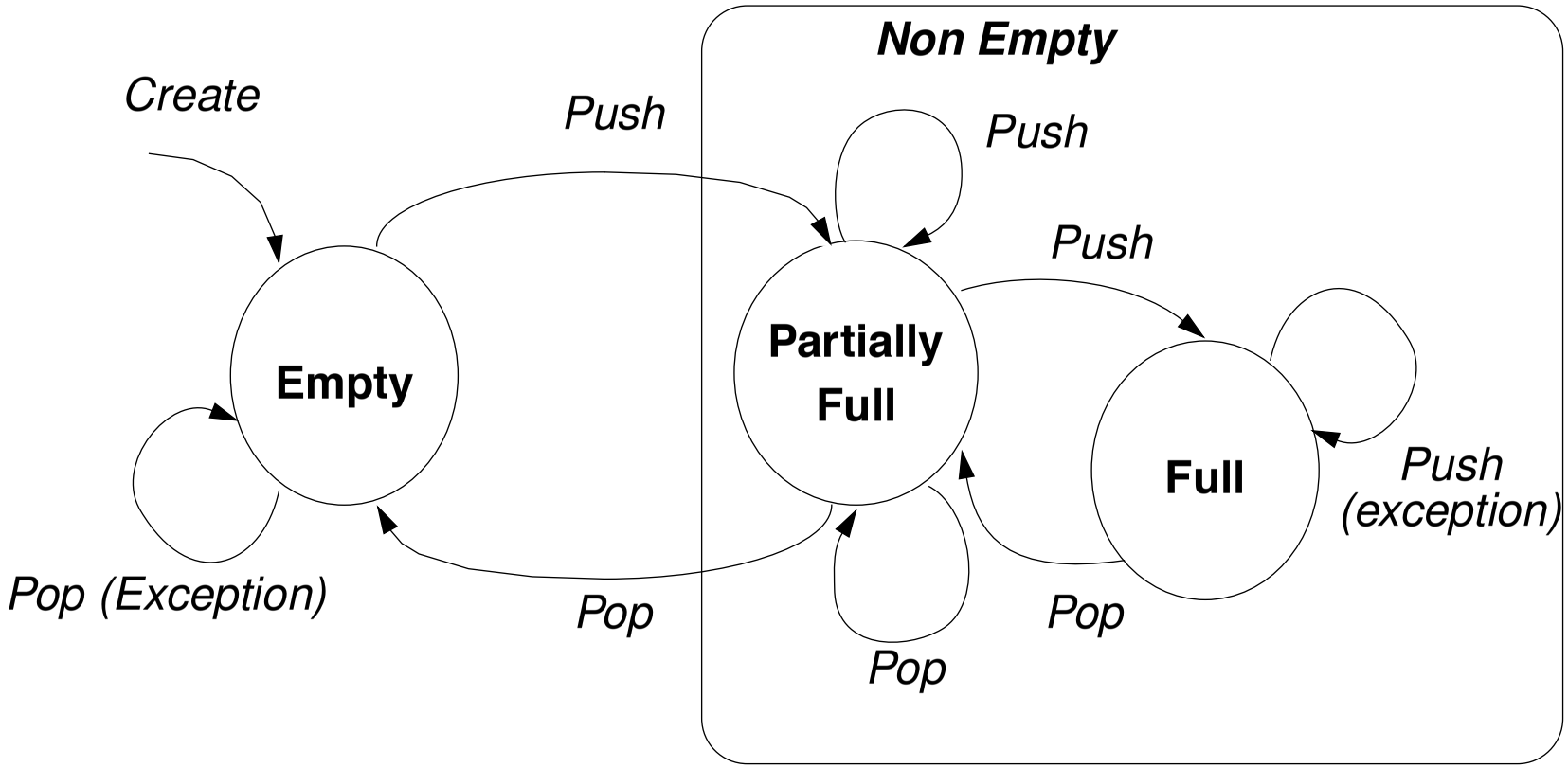
  Objects of type S can be used in any context in which an object of type T is appropriate.

If the subtype relationship has been carefully tested, it is not necessary to test the operation Manage with the states of all classes of the hierarchy, but just with the states of the root class.

The Incrementally Hypothesis

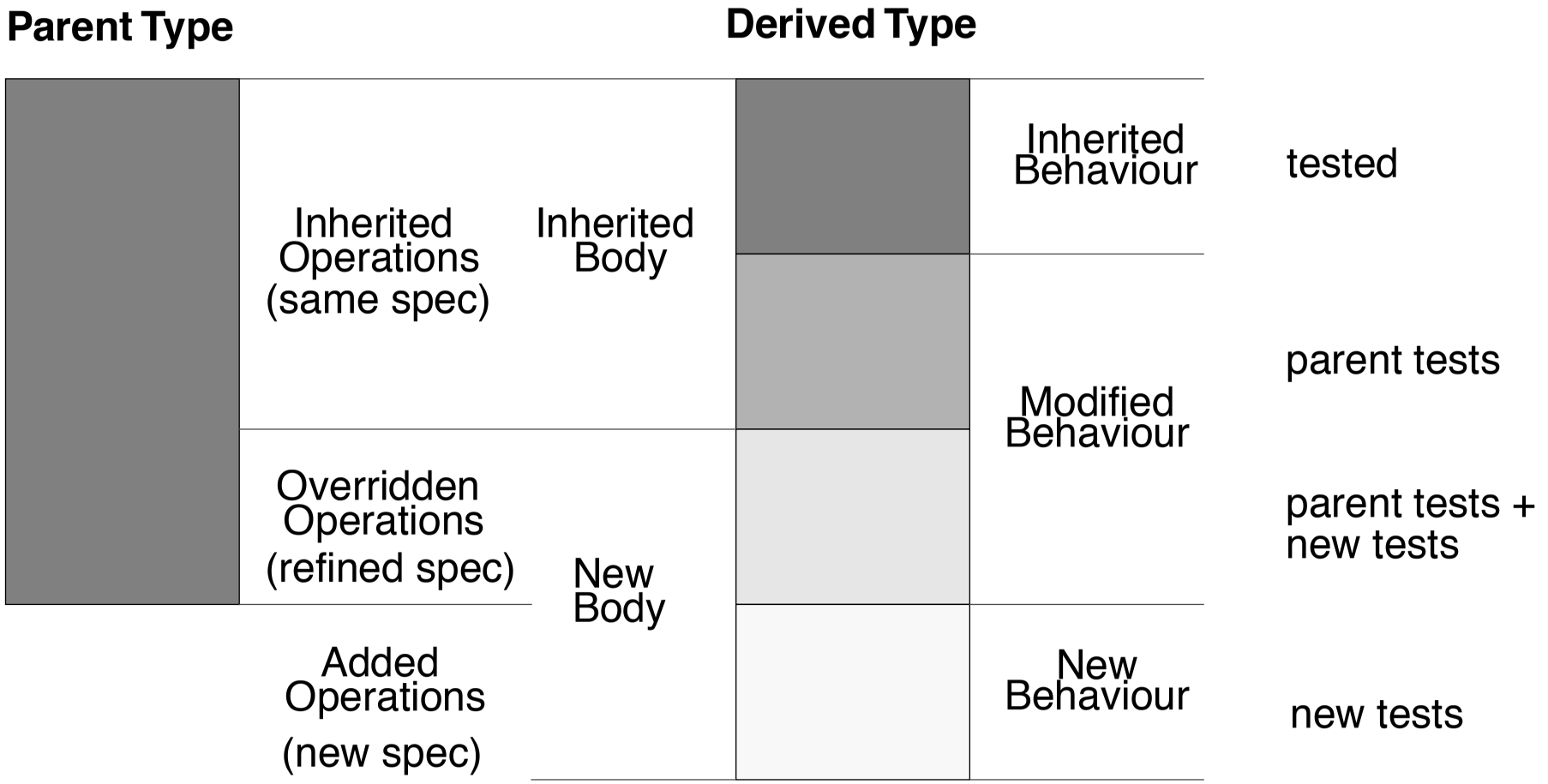
- There is a complex, but strong relation between the testing contexts of a supertype and its subtypes,
- We can take advantage of test sets defined for a supertype to reduce the new test sets to be selected to test the subtype.
- Inheritance can be very useful in minimizing the effort of testing a derived type given the test set developed for the base type.

The Incrementability Hypothesis



© 1995 Stéphane Barbey

Incremental Testing



© 1995 Stéphane Barbey

Inherited Body with New Behaviour

Redispatching

```
type Vehicle is tagged
record
        Passengers: Natural;
        ...;
    end record;
procedure Move (V: in out Vehicle;
                L: in Location);
function Speed_Of (V: Vehicle)
    return float;

type Car is new Vehicle with ...;
function Speed_Of (C: Car)
    return float;
    -- overridden
-- procedure Move (V: in out Car;
--                 L: in Location);
```

```
procedure Move (V: in out Vehicle;
                L: in Location) is
begin
    ... Speed_of (Vehicle'Class (V))...;
    -- dispatching
end Move;
```

Incremental Testing

Abstract Types

- Abstract types cannot have objects
  $\Rightarrow$ they cannot be tested!
- However, they have a specification
  $\Rightarrow$ test sets can be generated (for black-box testing)
- Those test sets are used for incremental testing

© 1995 Stéphane Barbey

Conclusion

- The theory of testing can be adapted to object-oriented programming by using the adequate reduction hypotheses.

Advantages of the proposed Approach

- This theory can be used with Ada 95.
- Theoretical soundness
- Possible automatization

Future Work

- Operationnal methods