ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

A Theory of Specification-Based Testing for Object-Oriented Software

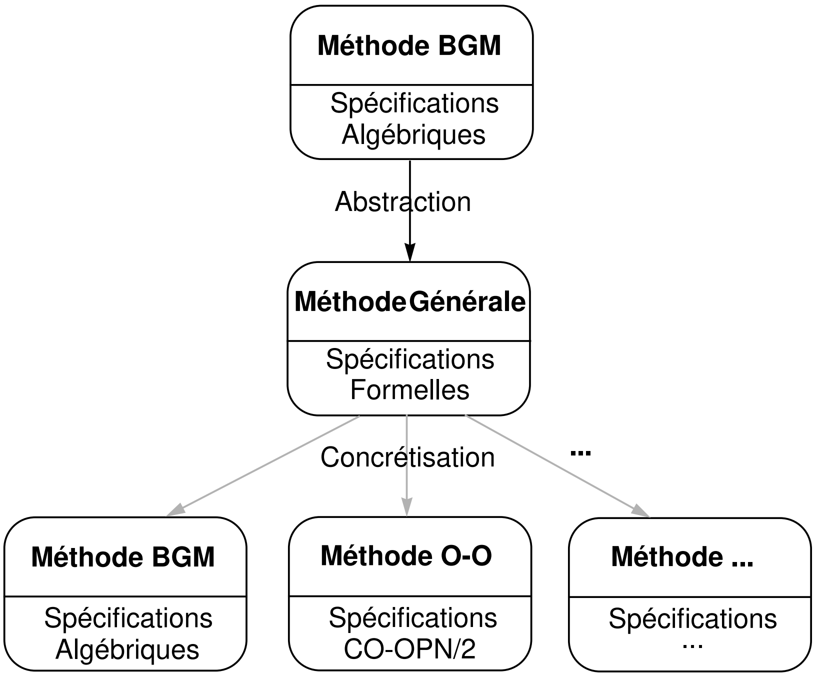
Cécile Péraire

Département d'informatique
Laboratoire de Génie Logiciel
Juin 1996

Plan de la présentation

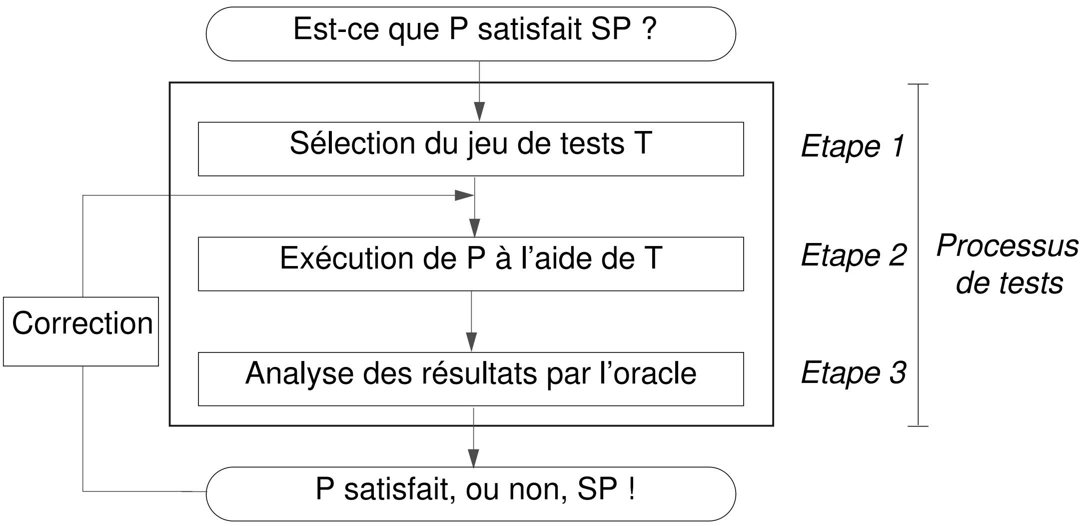
- Objectifs
- Test fonctionnel - Rappels
- Test fonctionnel pour applications orientées-objets
- Exemple: Agence de voyage
- Conclusion
- Référence

Objectifs



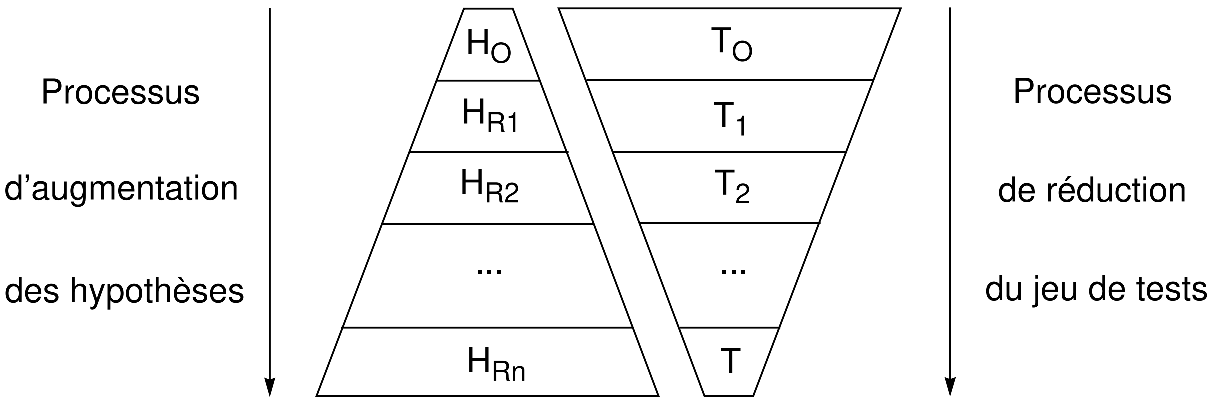
Test fonctionnel - Rappels

But du test: mettre en évidence les erreurs d'un logiciel.



Test fonctionnel - Rappels (suite)

Sélection du jeu de tests - Hypothèses de réduction



© 1996 Cécile Péraire

Test fonctionnel - Rappels (suite)

Hypothèses de réduction: Uniformité

Si le test d'une formule ϕ contenant une variable v,
est positif pour une valeur quelconque de v,
alors il est positif quelque soit la valeur prise par v.

Exemple: Spécification algébrique de la pile d'entiers non bornées

Axiome:

$$\mathtt{haut}\left(\begin{array}{|c|}\hline N\\\hline P\\\hline\end{array}\right) = \mathtt{suc}(\mathtt{haut}(\boxed{P})) \qquad \texttt{-- Variables: N et P}$$

Hypothèse: Uniformité sur la variable N

Tests:

$$\mathtt{haut}\left(\begin{array}{|c|}\hline 0\\\hline P\\\hline\end{array}\right) = \mathtt{suc}(\mathtt{haut}(\boxed{P})) \qquad \texttt{-- Variable: P}$$

© 1996 Cécile Péraire

Test fonctionnel - Rappels (suite)

Hypothèses de réduction: Régularité

Si le test d'une formule ϕ contenant un terme t,
est positif pour tout t de complexité inférieure ou égale à une borne,
alors il est positif quelque soit la complexité de t.

Exemple: Spécification algébrique des piles d'entiers non bornées

Tests:

$$\mathtt{haut}\left(\begin{array}{|c|}\hline \mathtt{0} \\ \hline \mathtt{P} \\ \hline \end{array}\right) = \mathtt{suc}\left(\mathtt{haut}\left(\begin{array}{|c|}\hline \mathtt{P} \\ \hline \end{array}\right)\right) \qquad \texttt{-- Variable: P}$$

Hypothèse: Régularité de borne 2 sur la variable P
(complexité: hauteur de P)

$$\mathtt{P} = \begin{array}{|c|}\hline \mathtt{vide} \\ \hline \end{array} \qquad \mathtt{P} = \begin{array}{|c|}\hline \mathtt{N1} \\ \hline \mathtt{vide} \\ \hline \end{array} \qquad \mathtt{P} = \begin{array}{|c|}\hline \mathtt{N2} \\ \hline \mathtt{N1} \\ \hline \mathtt{vide} \\ \hline \end{array} \qquad \texttt{-- Variables: N1 et N2}$$

© 1996 Cécile Péraire

Test fonctionnel pour applications orientées-objets

Choix des formalismes

- Formalisme utilisé pour décrire la spécification *SP*: *CO-OPN/2*
  Relation d'équivalence: bisimulation ($\underline{\leftrightarrow}$)
- Formalisme utilisé pour décrire le jeu de tests *T*: Logique *HML*
  Relation d'équivalence: équivalence *HML* ($\sim_{HML}$)

Il existe une **adéquation** entre *CO-OPN/2* et la logique *HML*:

$$\begin{array}{rcccl} & \multicolumn{3}{c}{\text{Théorème d'adéquation}} & \\ & \sim_{HML} & \Leftrightarrow & \underline{\leftrightarrow} & \\ \text{Définition} & \Updownarrow & & \Updownarrow & \text{Définition} \\ & \models_O & \Leftrightarrow & \models & \\ & \multicolumn{3}{c}{\text{Corollaire d'adéquation}} & \end{array}$$

➥ (P satisfait H) $\Rightarrow$ ($P \models_O T \Leftrightarrow P \models SP$)

© 1996 Cécile Péraire

Test fonctionnel pour applications orientées-objets (suite)

La logique *HML* [Hennessy-Milner]

Définition: Syntaxe de *HML*

Pour un ensemble d'événements *E*, le langage *HML* est défini comme suit:

- $T \in HML$
- $f \in HML \Rightarrow (\neg f) \in HML$
- $f, g \in HML \Rightarrow (f \wedge g) \in HML$
- $f \in HML \Rightarrow (\langle e \rangle\, f) \in HML$ avec $e \in E$

Exemples de formules *HML*:

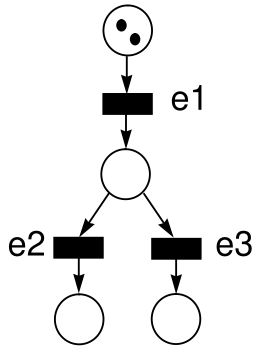
$\langle e1 \rangle \langle e2 \rangle\, T$

$\langle e1 \rangle \langle e3 \rangle\, T$

$(\langle e1 \rangle \langle e2 \rangle\, T) \wedge (\langle e1 \rangle \langle e3 \rangle\, T)$

$\neg \langle e1 \rangle \langle e3 \rangle\, T$

...



© 1996 Cécile Péraire

Test fonctionnel pour applications orientées-objets (suite)

Génération des jeux de tests

Un test est défini comme un doublet $\langle$*Formule, Résultat*$\rangle$ avec:

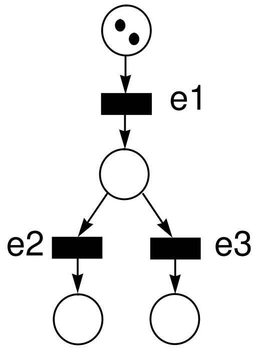
- $Formule \in HML$
- $Résultat \in \{true, false\}$
  Résultat indique si le résultat attendu de l'évaluation de *Formule* (à partir d'un état initial donné) est vrai ou faux.

Exemples:

$\langle \langle e1 \rangle \langle e2 \rangle\, T, true \rangle$

$\langle \neg \langle e1 \rangle \langle e3 \rangle\, T, false \rangle$

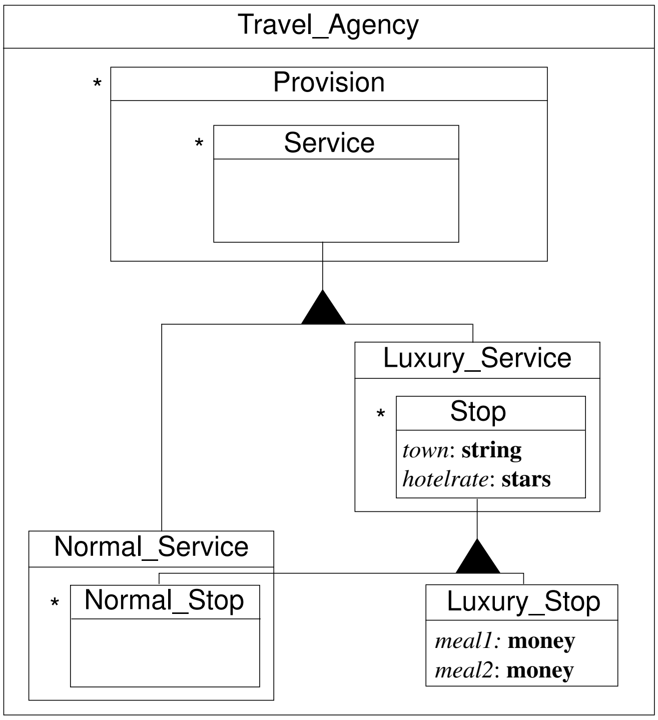
Remarque:
Un test va permettre de vérifier
que des scénarios d'utilisation non désirés,
ne soient pas jouables par le programme.



© 1996 Cécile Péraire

Exemple: Agence de voyage

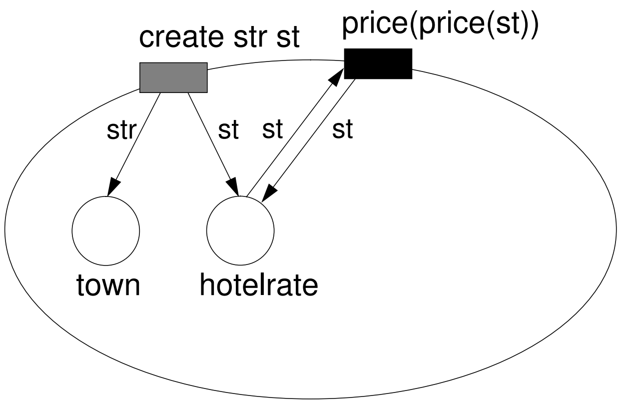
Modèle des objets (Fusion):



Exemple: Agence de voyage (suite)

Spécification CO-OPN/2 de la classe Stop

Class Stop;
Interface
Use Money, String, Stars;
Type stop;
Creation
create _ _ : string, stars;
Method
price _ : money;
Body
Places
town: string;
hotelrate: stars;
Axioms
create str, st : -> town str, hotelrate st;
price(price(st)): hotelrate st -> hotelrate st;
Where st: stars; str: string;
End Stop;



Exemple: Agence de voyage (suite)

Test de la classe Stop

Jeu de tests exhaustif:

$T_{StopExhaust}$ = {<< *stop*.create *str st* > *f*, *result* >
| *stop* ∈ stop, *str* ∈ string, *st* ∈ stars, *f* ∈ *HML*, *result* ∈ {true, false}}

☞ **En supposant que le gestionnaire des objets fonctionne correctement, on peut faire une hypothèse d'uniformité sur la variable *stop*:**

T_{Stop} = {<< s.create *str st* > *f*, *result* >
| *str* ∈ string, *st* ∈ stars, *f* ∈ *HML*, *result* ∈ {true, false}}

☞ **Hypothèse de régularité (de borne 2) sur *f* :**

T_{Stop} = {<< s.create *str st* > < s.price price *st1* > < s.price price *st2* >, *result* >
| *str* ∈ string, *st, st1, st2* ∈ stars, *result* ∈ {true, false}}.

s.create *str st* ⟶ s.price price *st1* ⟶ s.price price *st2* $\boxed{result}$

Exemple: Agence de voyage (suite)

Test de la classe Stop

Etude du domaine de validité des variables

s.create *str st* ⟶ s.price price *st1* ⟶ s.price price *st2* $\boxed{result}$

Domaine de validité de *str* :

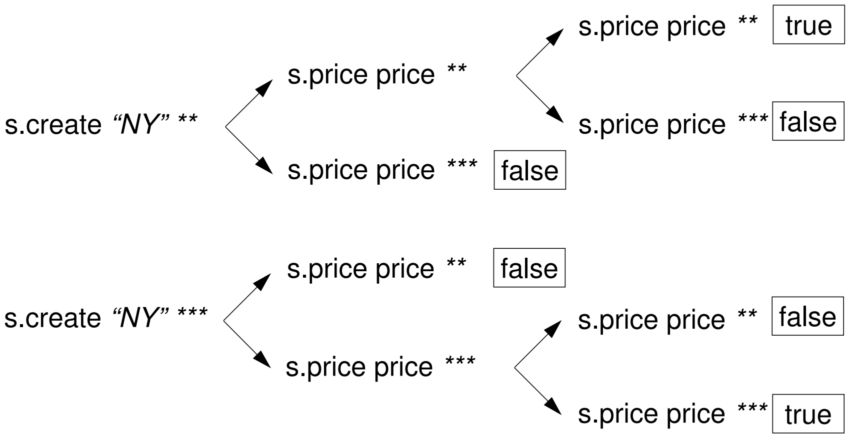
result est indépendant de la valeur de *str*
➥ Uniformité sur *str* : *str* = "NY"

Domaine de validité de *st, st1* and *st2* :

Il existe deux sortes d'étoiles: ** et ***
➥ On conserve la totalité des valeurs

Exemple: Agence de voyage (suite)

Test de la classe Stop

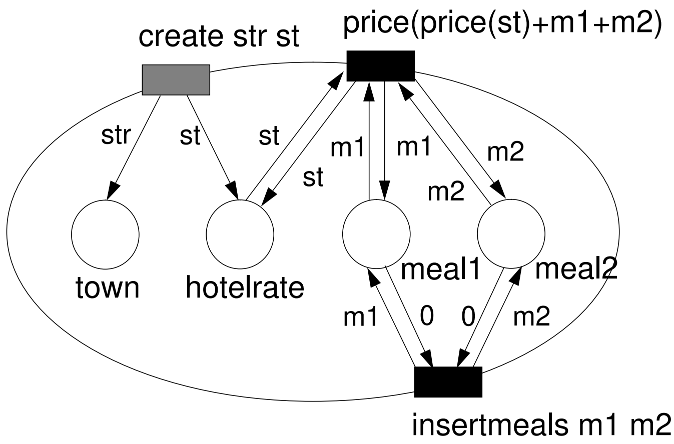


T_{Stop} = { <<*s*.create *“NY” *** > <*s*.price price **** > <*s*.price price **** > T, true>,
<<*s*.create *“NY” *** > <*s*.price price **** > <*s*.price price ***** > T, false>,
<<*s*.create *“NY” *** > <*s*.price price ***** > T, false>,
<<*s*.create *“NY” **** > <*s*.price price **** > T, false>,
<<*s*.create *“NY” **** > <*s*.price price ***** > <*s*.price price **** > T, false>,
<<*s*.create *“NY” **** > <*s*.price price ***** > <*s*.price price ***** > T, true> }

Exemple: Agence de voyage (suite)

Spécification CO-OPN/2 de la classe Luxury_Stop

Class Luxury_Stop;
Inherit Stop;
Rename stop -> luxurystop;
Redefine price _ : money;
Interface
Use Money;
Subtype luxurystop < stop;
Method
insertmeals _ _ : money, money;
Body
Places
meal1: money;
meal2: money;
Initial
meal1 0;
meal2 0;
Axioms
insertmeals m1, m2: meal1 0, meal2 0 -> meal1 m1, meal2 m2;
price(price(st)+m1+m2) : hotelrate st, meal1 m1, meal2 m2 -> hotelrate st, meal1 m1, meal2 m2;
Where m1,m2: money;
End Luxury_Stop;



Exemple: Agence de voyage (suite)

Test de la classe Luxury_Stop

Jeu de tests exhaustif:

$T_{LStopExhaust}$ = {<< *stop*.create *str st* > *f*, *result* >
| *stop* ∈ luxurystop, *str* ∈ string, *st* ∈ stars, *f* ∈ *HML*, *result* ∈ {true, false}}

☞ **En supposant que le gestionnaire des objets fonctionne correctement, on peut faire une hypothèse d'uniformité sur la variable *stop*:**

T_{LStop} = {<< s.create *str st* > *f*, *result* >
| *str* ∈ string, *st* ∈ stars, *f* ∈ *HML*, *result* ∈ {true, false}}

☞ **Hypothèse de régularité (de borne 3) sur *f* :**

f ensemble d'évènements de la forme:

- < s.insertmeals m_i n_i >
- < s.price price $st_i + m_i + n_i$ >

avec st_i ∈ stars et m_i, n_i ∈ money (i =1, 2, 3).

© 1996 Cécile Péraire

Exemple: Agence de voyage (suite)

Test de la classe Stop - Etude du domaine de validité des variables

Domaine de validité de *str* :

result est indépendant de la valeur de *str.*
➥ Uniformité sur *str* : *str* = "NY"

Domaine de validité de st_i :

La méthode **price** a déjà été testée dans la classe **Stop**,
cependant de nouveaux comportements ont été spécifiés et il faut les tester.
➥ Uniformité sur st_i : st_i = **

Domaine de validité de m_i, n_i dans < s.insertmeals m_i n_i >:

Deux sous-domaines: < m_i, n_i > = <0, 0> et < m_i, n_i > > <0, 0>.
➥ Uniformité sur les sous-domaines: < m_i, n_i > = <0, 0> et < m_i, n_i > = <2, 4>.

Domaine de validité de m_i, n_i dans < s.price price $st_i + m_i + n_i$ >:

Trois sous-domaines: < m_i, n_i >=<0,0>, < m_i, n_i >=<2,4> et < m_i, n_i > ≠ {<0,0>,<2,4>}.
➥ Uniformité sur les sous-domaines:
< m_i, n_i > = <0, 0>, < m_i, n_i > = <2, 4> et < m_i, n_i > = <6, 8>.

© 1996 Cécile Péraire

Conclusion

Ce qu'il faut faire...

- **Introduire de nouvelles hypothèses de réduction**

  Ex: Hypothèse d'incrémentalité en cas de sous-typage sémantique.

- **Résoudre le problème de l'oracle**

- **Développer un outil de génération automatique de tests**

Référence

Stéphane Barbey, Didier Buchs and Cécile Péraire,
"A Theory of Specification-Based Testing for Object-Oriented Software",
EDCC-2, Taormina, Italy, October 1996.

© 1996 Cécile Péraire