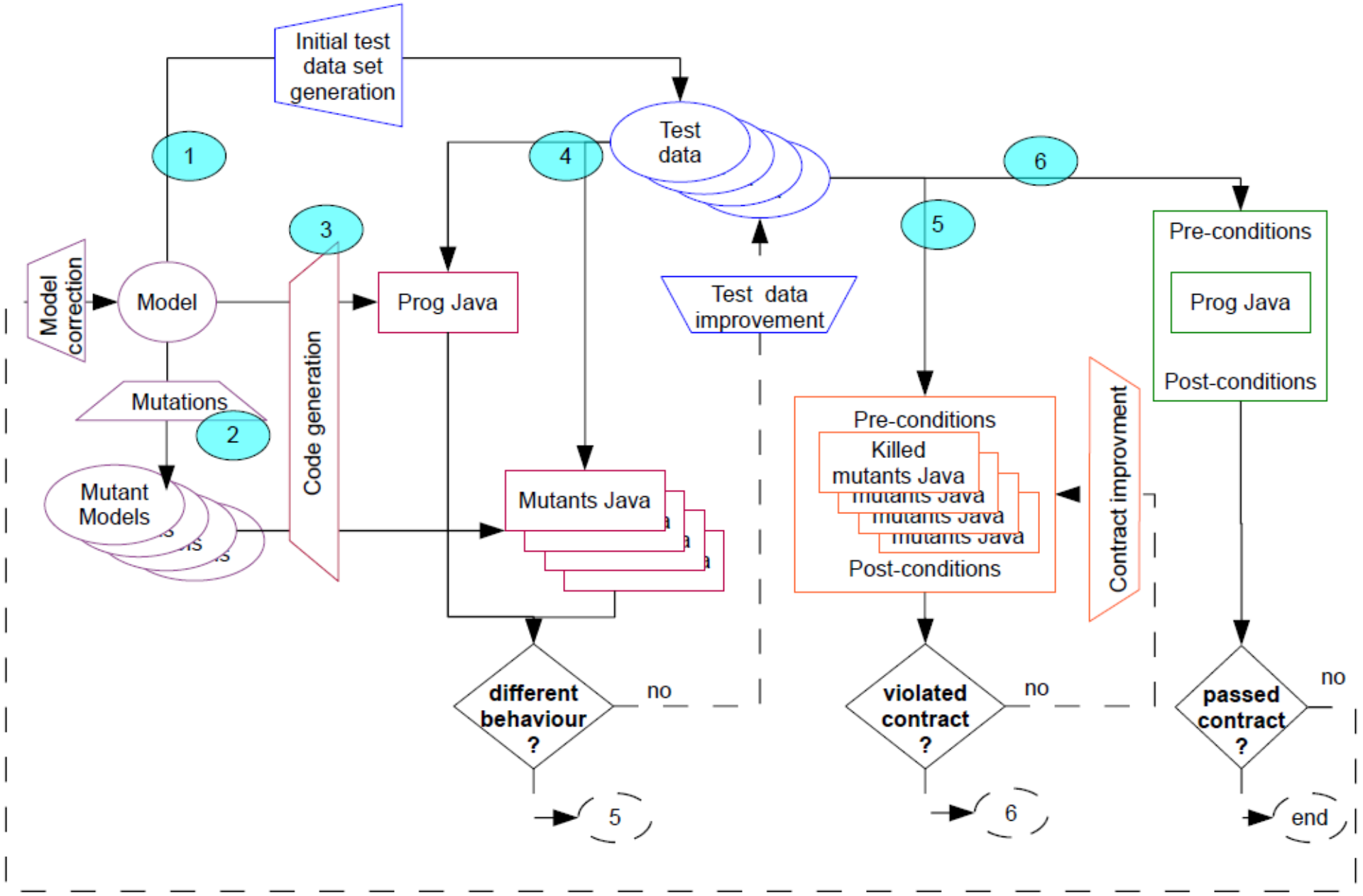
Analyse de mutation

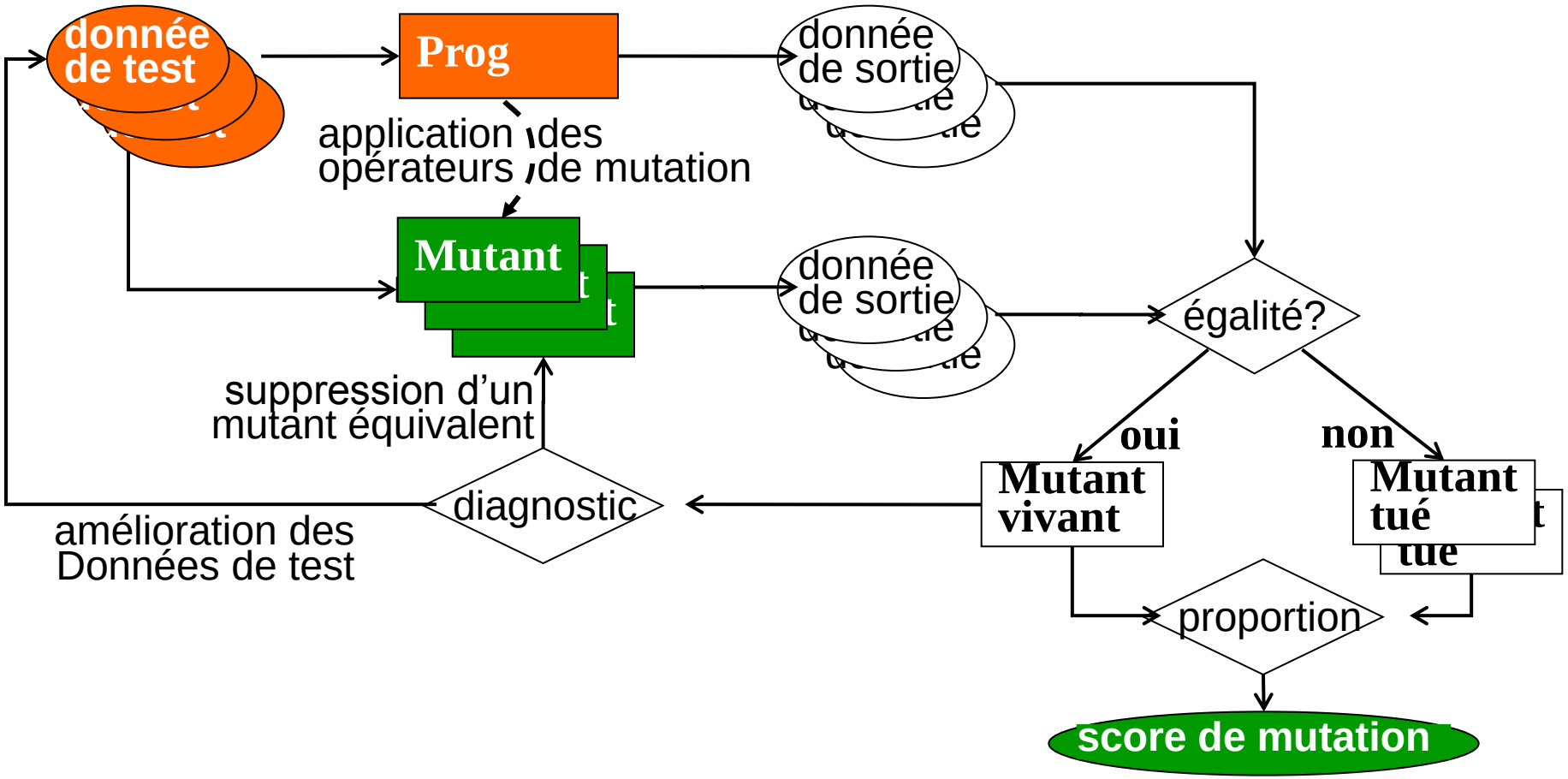
Application au FSM

JM Mottu – Séminaire AELOS

17/02/2011


Initial test data set generation
1
Model correction
Model
3
Code generation
Prog Java
4
Test data
6
5
Test data improvement
Pre-conditions
Prog Java
Post-conditions
Mutations
2
Mutant Models
Mutants Java
Pre-conditions
Killed mutants Java
Post-conditions
Contract improvment
different behaviour ?
no
violated contract ?
no
passed contract ?
no
5
6
end

Processus de l'analyse de mutation



Hypothèses

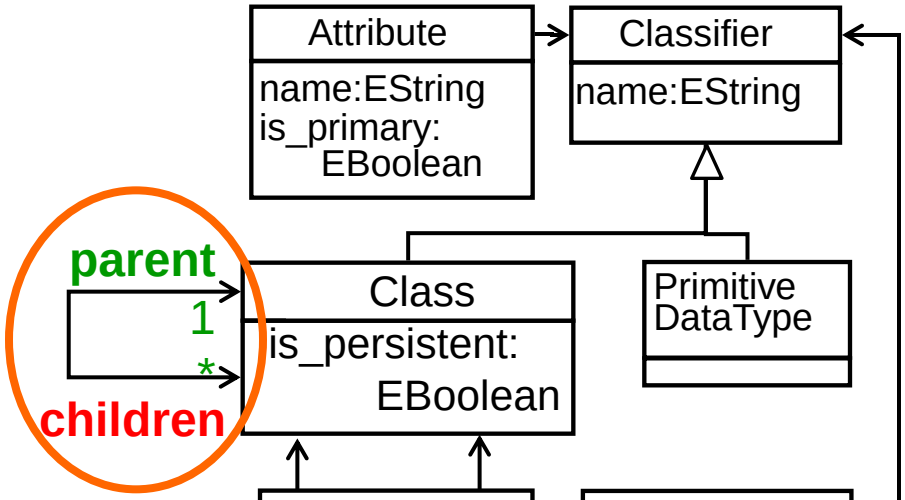
- Le programmeur est compétent
  - En général, les programmeurs sont compétents et écrivent des programmes *presque* corrects.
  - Ces programmes sont seulement un peu différents de la version correcte.

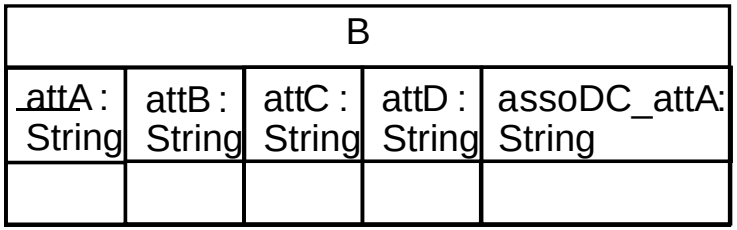
- L'effet de couplage.
  - Une faute complexe introduite par un programmeur compétent est la combinaison de fautes simples
  - Détecter les fautes simples => détecte les fautes complexes

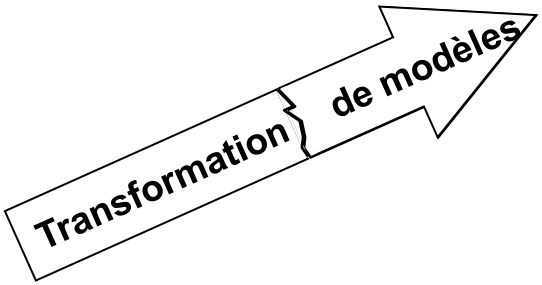
Modèles de fautes

- Quelles fautes doit-on **mettre à l'épreuve** avec des *modèles de test*?
- Quelles fautes doit-on **observer** avec des *oracles*?
- Quelles fautes **apparaissent** dans un *type de programme donné*?

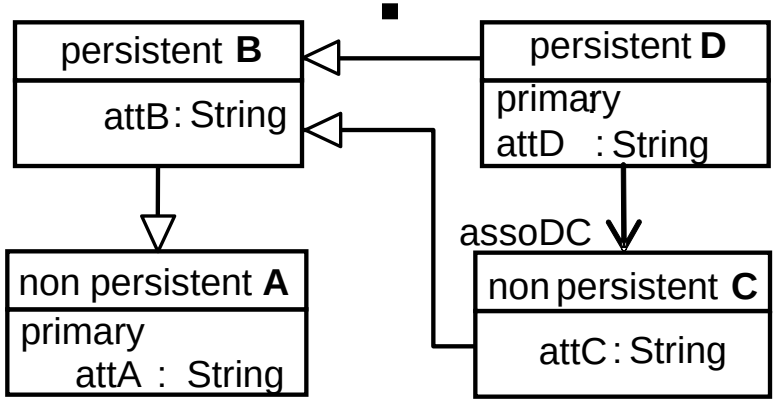
Détecter les erreurs commises des Transfo de Mod.

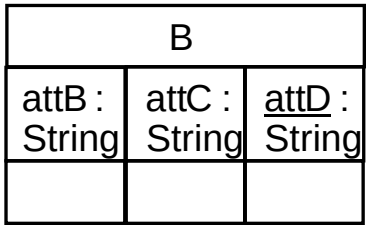






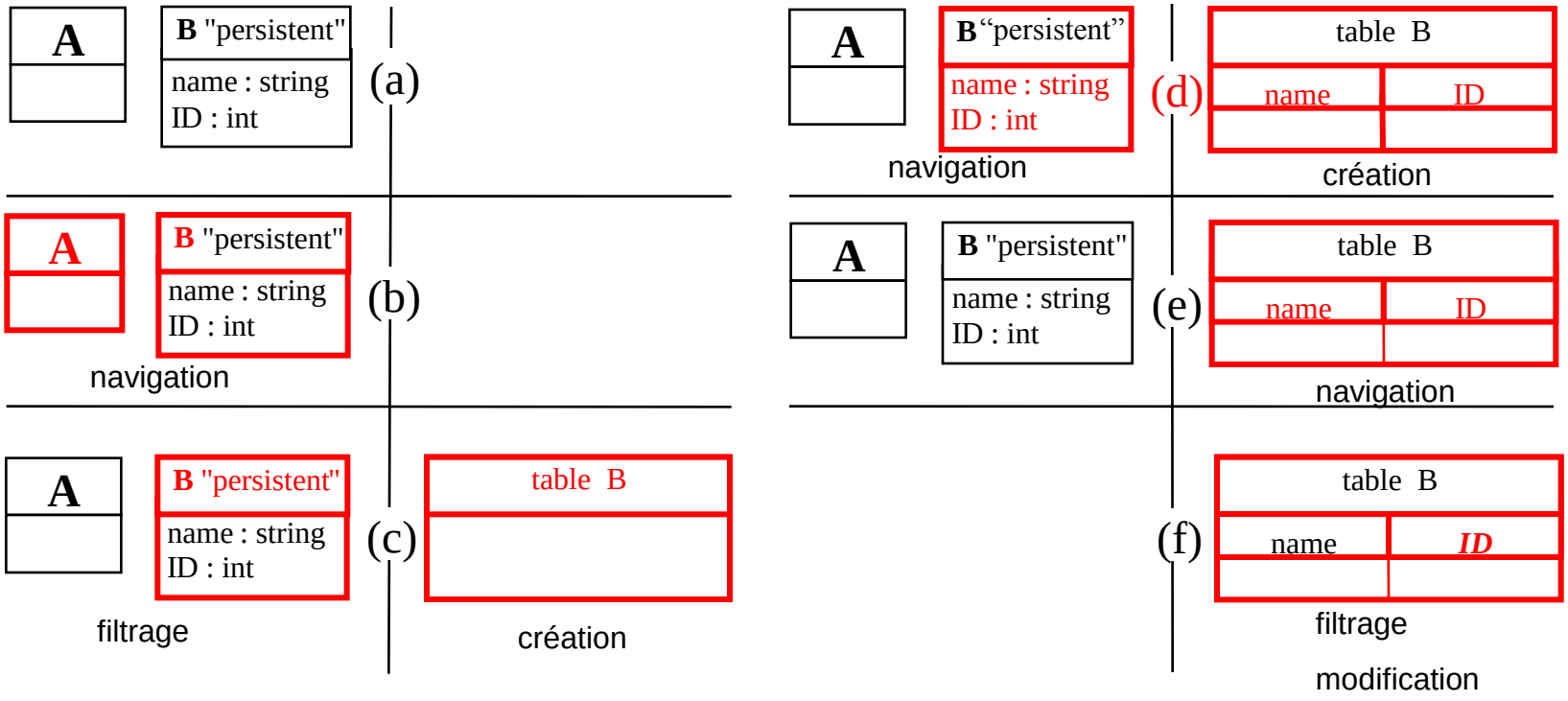
conforme





Opérations d'une transformation

- Opérations abstraites
  - Navigation, filtrage, création, modification
- Par exemple :



Modèles de fautes propres aux transformations de modèles

opération abstraite	modèle de faute	altération réalisée
navigation	RRMC	Remplacement d'une relation vers une même classe
	RRAC	Remplacement d'une relation vers une autre classe
	MSNM	Modification d'une succession de navigation avec manque
	MSNA	Modification d'une succession de navigation avec ajout
filtrage	MFCP	Modification du filtrage d'une collection avec perturbation
	MFCM	Modification du filtrage d'une collection avec manque
	MFCA	Modification du filtrage d'une collection avec ajout
création	RCCC	Remplacement de la création d'une classe par une autre compatible
	MMRR	Modification d'une mise en relation de classes avec retrait
	MRCA	Modification d'une mise en relation de classes avec ajout

Elévation sémantique acceptable

- Ce n’est pas parce qu’une transfo est programmée dans un langage qu’il ne faut se placer qu’au niveau de ce langage
- La sémantique a autant d’intérêt que le syntaxique
  - Et cette sémantique est implantée dans la syntaxe du langage
- Repose sur les deux hypothèses

R. A. DeMillo and A. J. Offutt. Constraint-based automatic test data generation. IEEE Transactions on Soft. Eng.

- Given an original product P and a mutant M of P which contains a fault f inserted by a mutation operator:

  (i) f must be reached and executed
  - reachability

  (ii) the state of M must be infected after the execution off
  - necessity

  (iii) the difference between the states of P and M after the execution of the mutated portion of code must propagate to the end of P and M
  - sufficiency

Adaptation de l'analyse de mutation

- Pertinence des mutants produits
  - Considération des manipulations, de la réflexion effectuées par le développeur
- Indépendance d'un langage d'implantation
  - Les erreurs ne doivent pas forcément être basées sur la syntaxe d'un langage donné

Formel et analyse de mutation

B. K. Aichernig, “Mutation Testing in the Refinement Calculus,” *Formal Aspects of Computing*, 2003

- "The synergy of combining formal methods and testing has become a popular area of research“
- To start the test-case design early in the specification phase and to check which kinds of wrongly implemented requirements, represented by *specification mutations*, can be found by the test-cases
- To find errors in the program caused by wrongly understood, or wrongly refined (implemented), specifications
  - not to find errors in the specification

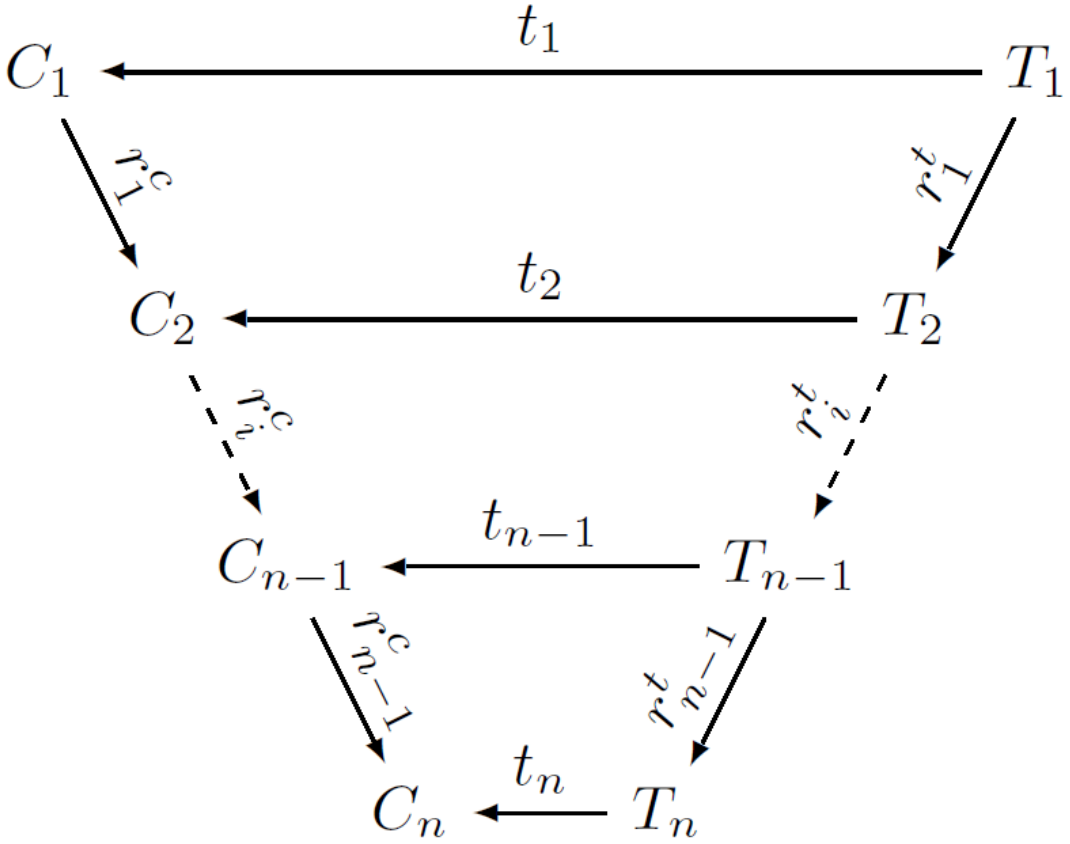
B. K. Aichernig, “Mutation Testing in the Refinement Calculus,” *Formal Aspects of Computing*, 2003

- Mutant contract by introducing small changes to the formal contract definition
- test-case is an abstraction of the original contract

- refinement calculus, formal contract language of Back and von Wright [BvW98], pre- and post-condition style of formal specifications known from VDM, B and Z

B. K. Aichernig,
"Mutation Testing in the Refinement Calculus,"
Formal Aspects of Computing, 2003

- Contrats de spécification sont les raffinements de test cases (réciproquement abstraction)
- Cn est l'implantation de C1.



Paul E. Black, Vadim Okun, and Yaacov Yesha.
Mutation Operators for Specifications.
In *Proceedings of ASE'00,*

- SCR specification (Software Cost Reduction # FSM) is converted to an SMV model (symbolic model checker) and a set of temporal logic constraints
- Mutation applied to both the textual description of the model and the requirement properties

- verify mutant models with regard to the temporal logic constraints

Paul E. Black, Vadim Okun, and Yaacov Yesha. Mutation Operators for Specifications

accelerate & !brake & velocity = stop: slow

- Logical Operator Replacement (LRO) : accelerate | !brake & velocity = stop: slow
- Relational Operator Replacement (RRO) : accelerate & !brake & velocity > stop: slow
- Expression Negation Operator (ENO) :accelerate & !(!brake & velocity = stop): slow
- Simple Expression Negation (SNO) :!accelerate & !brake & velocity = stop: slow
- Operand Replacement Operator (ORO)
  - Variable Replacement Operator (VRO) : accelerate & !accelerate & velocity = stop: slow
  - Constant Replacement Operator (CRO) : accelerate & !brake & velocity = stop: stop
- Missing Condition Operator (MCO) : accelerate & & velocity = stop: slow
- Stuck At Operator (STO) : 1 & !brake & velocity = stop: slow
- Associative Shift Operator (ASO) :
  - (accelerate | !brake) & velocity = stop: slow mutated as accelerate | (!brake & velocity = stop): slow .
- Arithmetic Operator Replacement (ARO)
  - accelerate & !brake & velocity=stop: velocity+2 mutated as accelerate & !brake & velocity=stop: velocity**-2**

Fabbri et al.

- S. P. F. Fabbri, M. E. Delamaro, J. C. Maldonado, and P. Masiero, "**Mutation Analysis Testing for Finite State Machines**,"
  in *Proceedings of SRE'94*.

- S. C. P. F. Fabbri, J. C. Maldonado, P. C. Masiero, and M. E. Delamaro, "**Proteum/FSM: A Tool to Support Finite State Machine Validation Based on Mutation Testing**,"
  in *Proceedings of SCCC'99*

- S. C. P. F. Fabbri, J. C. Maldonado, T. Sugeta, and P. C. Masiero, "**Mutation Testing Applied to Validate Specifications Based on Statecharts**,"
  in *Proceedings of ISSRE'99*.

Fabbri et al. FSM

- 9 mutation operators
  - related to the states, events and outputs of an FSM.
- implemented as an extension of the C mutation tool Proteum -> Proteum/FSM

Fabbri et al. FSM

- 9 mutation operators
  - arc-missing
  - wrong-starting-state (default state)
  - event-missing
  - event-exchanged
  - event-extra
  - state-extra
  - output-exchanged
  - output-missing
  - output-extra

Fabbri et al. FSM Performance

- Compared to test sequences generation by the W and TT methods for FSMs
  - TT (Transition Tour)
    - test sequence that exercises all transitions
    - not efficient to reveal transfer errors (when M' is not equivalent to M, but can be modified to become equivalent to M by changing only the transition function of M)
  - W (Automata Theoretic)
  - Effective and Complementary
- Coupling effect in general is also valid for FSM-based Mutation Analysis
  - Only need 1-mutants

R. M. Hierons and M. G. Merayo, "Mutation Testing from Probabilistic [and Stochastic] Finite State Machines," 2007-2009

- probabilities resolving the non-deterministic choices that a system may undertake
  - Probabilities attached to the transitions
- Stochastic
  - time information is incremented with some kind of probabilistic information
- 4 mutation operators
  - Changing the initial state
  - Altering probabilities
  - Changing the target state of a transition
  - Creating a new transition

extended finite state machine (EFSM)

- Batth et al. , “Specification of Timed EFSM Fault Models in SDL,” in *Proceedings of FORTE’07*,
  - Incorrect Timer Setting Faults

- Bombieri et al. , “A Mutation Model for the SystemC TLM2.0 Communication Interfaces,” in *Proceedings of DATE’08,*
  - enabling function and an update function (register data)
  - Mutations on destination states, on enabling functions, on update functions (modification of the operation, and perturbation of data included in request or response packets)

Statecharts

- hierarchy of states,
- ability to specify parallelism and communication mechanism via broadcasting, and a special notation set augmenting the representational power in relation to FSM
- state history

- Harel : *Statecharts = state diagrams + decomposition + orthogonality + broadcasting*

Statecharts : Fabbri

- FSM operator set

1. wrong-start-state
2. arc-missing
3. event-missing
4. event-extra
5. event-exchanged
6. destination-exchanged
7. output-missing
8. output- exchanged
9. state-missing

- EFSM operator set

1. expression deletion
2. boolean expression negation
3. term associativity shift
4. arithmetic operator by arithmetic operator
5. relational operator by relational operator
6. logical operator by logical operator
7. logical negation
8. variable by variable replacement
9. variable by constant replacement
10. constant by required constant replacement
11. constant by scalar variable replacement

- Statecharts-feature-based operator set

1. transition's history deletion
2. transition with history by transition replacement
3. history-missing
4. h by h* replacement
5. h* by h replacement
6. h-extra
7. h*-extra
8. in(s) condition-missing
9. in(s) condition state replacement
10. not-yet(e) condition-missing
11. not-yet(e) condition event replacement
12. exit(s) event-missing
13. exit(s) event state replacement
14. entered(s) event-missing
15. entered(s) event state replacement
16. broadcasting origin transition replacement
17. broadcasting destination transition replacement