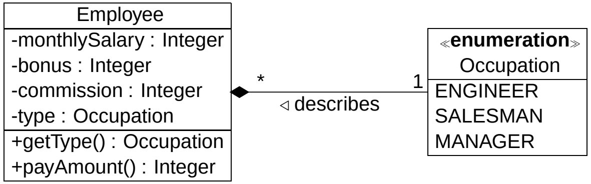
Toward a Formal Evaluation of Refactorings

John Paul, Nadya Kuzmina,
Ruben Gamboa, and James Caldwell

Department of Computer Science
University of Wyoming
Laramie, Wyoming 82071-3315

Overview

Structural Design



Constraints

- Object Invariants

```
context Employee inv:
  this.monthlySalary > 0
```

- Method Invariants

```
context Employee::getType() : Occupation
   pre: this.type = ENGINEER
  post: return = ENGINEER

context Employee::getType() : Occupation
   pre: true
  post: return = this.type
```

Specification

-
  ```
  context Employee::payAmount(): Integer
      pre: type=ENGINEER
     post: return=this.monthlySalary
  ```

-
  ```
  context Employee::payAmount(): Integer
      pre: type=SALESMAN
     post: return=this.monthlySalary + this.commission
  ```

-
  ```
  context Employee::payAmount(): Integer
      pre: type=MANAGER
     post: return=this.monthlySalary + this.bonus
  ```

-
  ```
  context Employee::payAmount(): Integer
      pre: true
     post: orig(this.type) = this.type
  ```
- ....

Domains, Relations and Models

Employee=$\{$E0, E1$\}$
Occupation=$\{$EN, SA, MA$\}$
Integer=$\{0, 1, 2, 3\}$

type=$\{$(E0, EN), (E1, SA)$\}$
monthlySalary=$\{$(E0, 1), (E1, 2)$\}$
commission=$\{$(E0, 1),(E1,1)$\}$

payAmount=$\{$(E0,E0,1), (E1,E1,2)$\}$

Employee={E0, E1}, Occupation={EN, SA, MA},
Integer={0, 1, 2, 3}
type={(E0, EN), (E1, SA)},
monthlySalary={(E0, 1), (E1, 2)},
commission={(E0, 1),(E1,1)}
payAmount={(E0,E0,1), (E1,E1,2)}

-
  ```
  context Employeee::payAmount(): Integer
     pre: type=ENGINEER
    post: return=this.monthlySalary
  ```

-
  ```
  context Employeee::payAmount(): Integer
     pre: type=SALESMAN
    post: return=this.monthlySalary + this.commission
  ```

-
  ```
  context Employee::payAmount(): Integer
     pre: type=MANAGER
    post: return=this.monthlySalary + this.bonus
  ```

-
  ```
  context Employee::payAmount(): Integer
     pre: true
    post: orig(this.type) = this.type
  ```

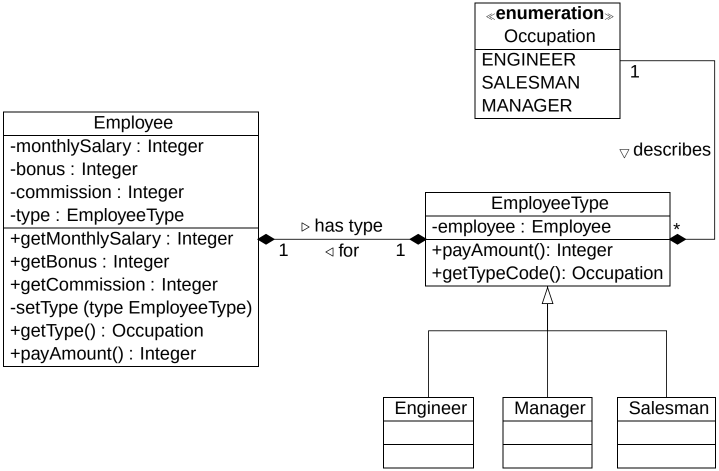
Implementation

```
public class Employee {
  private int type,monthlySalary, commission, bonus;;

  static final int ENGINEER = 0;
  static final int SALESMAN = 1;
  static final int MANAGER = 2;

  public Employee (int type,int monthlySalary,int commission,int bonus
  {....}

  public int payAmount() {
    //pre
    switch (type) {
      case ENGINEER: return monthlySalary;

      case SALESMAN: return monthlySalary + commission;

      case MANAGER: return monthlySalary + bonus;

      default: throw new RuntimeException("Unknown Occupation");
    }
    //post
  }
}
```

Gathering Facts

-
  ```
  context Employee::payAmount(): Integer
      pre: type = SALESMAN
     post: return = monthlySalary + commission
  ```

Does Changing the Design Help?



Composition of Salesman with Employee

-
  ```
  context Salesman::getTypeCode(): Occupation
     pre: true
    post: return = SALESMAN
  ```

-
  ```
  context Salesman::payAmount(): Integer
     pre: true
    post: return=this.employee.getMonthlySalary() +
                     this.employee.getCommission()
  ```

-
  ```
  context Employee::getMonthlySalary(): Integer
     pre: true
    post: return = this.monthlySalary
  ```

-
  ```
  context Employee::getCommission(): Integer
     pre: true
    post: return = this.commission
  ```

Composition of Employee with Salesman

-
  ```
  context Employee::getType(): Occupation
    pre: true
   post: return = this.type.getTypeCode()
  ```

-
  ```
  context Employee::payAmount(): Integer
    pre: true
   post: return = this.type.payAmount()
  ```

-
  ```
  context Employee inv:
   this.commission = this.type.employee.commission
  ```

-
  ```
  context Employee inv:
   this.monthlySalary=this.type.employee.monthlySalary
  ```

Inference

We are close to drawing the desired conclusion

-
  ```
  context Employee::payAmount(): Integer
     pre: this.getType() = SALESMAN
    post: return = monthlySalary + commission
  ```

But we also must know

-
  ```
  context Manager::getTypeCode(): Occupation
    pre: true
   post: return = MANAGER
  ```

-
  ```
  context Engineer::getTypeCode(): Occupation
    pre: true
   post: return = ENGINEER
  ```

Formalizing a design D

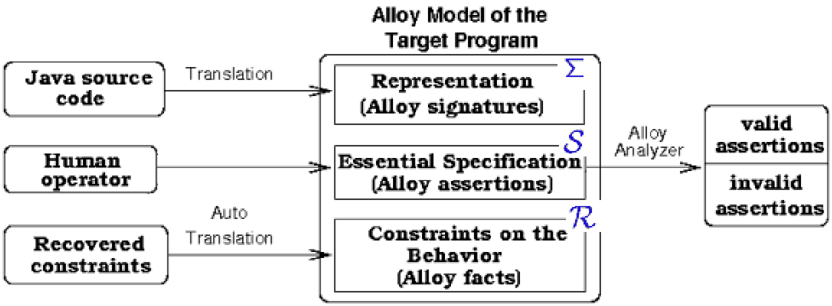
- D is modelled as a first-order theory $\langle \Sigma, \mathcal{R} \rangle$.
  - Σ is a relational signature extracted from D.
  - $\mathcal{R}$ is a set of Σ-sentences describing D's behavior.
- $\mathcal{S}$ expresses what it means for D to be correct
  - A set of Σ-sentences
  - Specification

Comparing D with D'

- Construct $\sigma : \Sigma \to \Sigma'$ by hand.
  - Translate every $\psi \in S$
- D' is better verifiable than D w.r.t. $\psi \in \mathcal{S}$ if

$$\mathcal{R}' \models \sigma(\psi), \text{ but } \mathcal{R} \not\models \psi$$

Our Prototype



Practical Considerations

- Many facts are required to do a proof.
- There are many more irrelevant facts.
- Proofs require a careful selection and combination of facts.
- We want an automated process.

assertion kind	number of assertions	number of checked assertions for D	number of checked assertions for D'
on fields	4	4 (100%)	4 (100%)
on payAmount	7	4 (57%)	7 (100%)
on getType	4	4 (100%)	4 (100%)
total	15	12 (80%)	15 (100%)

Comparative evaluation of designs D and D' of the Employee *example by Daikon.*

Questions?