Optimizing JML Feature Compilation using AO Refactorings

Henrique Rebêlo
Ricardo Massa F. Lima
Alexandre Mota
César Oliveira

Federal University of Pernambuco
(hemr, rmfl, acm, calo @cin.ufpe.br)

Gary T. Leavens
University of Central Florida
(leavens@eecs.ucf.edu)

Márcio Cornélio
University of Pernambuco
(mlc@dsc.upe.br)

Java modeling language—JML

- Formal specification language for Java
  - behavioral specification of Java modules
- Adopts design by contract based on Hoare-style with assertions
  - pre-, postconditions and invariants
  - {P} C {Q}
- Main goal → Improve functional software correctness of Java programs

First JML example

```
public class ArrayOps {
  private /*@ spec_public @*/ Object[] a;

  //@ public invariant 0 < a.length;

  /*@ requires 0 < arr.length;
    @ ensures this.a == arr;
    @*/
  public void init(Object[] arr) {
    this.a = arr;
  }
}
```

Field specification

```
public class ArrayOps {
  private /*@ spec_public @*/ Object[] a;

  //@ public inv

  /*@ requires 0
    @ ensures th
    @*/
  public void init(Object[] arr) {
    this.a = arr;
  }
}
```

spec_public makes the field a visible in public specifications

Object invariant

```
public class ArrayOps {
  private /*@ spec_public @*/ Object[] a;

  //@ public invariant 0 < a.length;

  /*@ requir
    @ ensure
    @*/
  public void init(Object[] arr) {
    this.a = arr;
  }
}
```

Defining a constant property

Method specification

```
public class ArrayOps {
  private /*@ spec_public @*/ Object[] a;

  //@ publi

  /*@ requires 0 < arr.length;
    @ ensures this.a == arr;
    @*/
  public void i
    this.a = ar
  }
}
```

Defining a pre-state property

Defining a post-state property

Tool Support

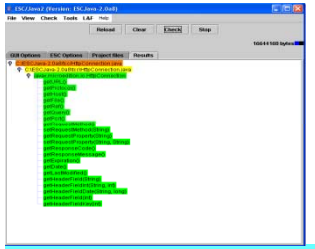
- Many tools, one language

jmldoc

Web pages

JML Annotated Java

```
public class Animal implements Gendered {
  // ...
  protected /*@ spec_public @*/ int age = 0;
  /*@    requires 0 <= a && a <= 150;
     @    ensures age == a;
     @ also
     @    requires a < 0;
     @    ensures age == \old(age);   @*/
  public void setAge(final int a) {
    if (0 <= a) { age = a; }
  }
}
```



ESC/Java2

```
Bag.java:15: Warning: Possible null
dereference (Null)
      if (elements[i] < min) {
          ^
Bag.java:15: Warning: Array index
possibly too large (..
      if (elements[i] < min) {
```

Warnings

jmlunit

Unit tests

jmlc

Class file

jmlrac

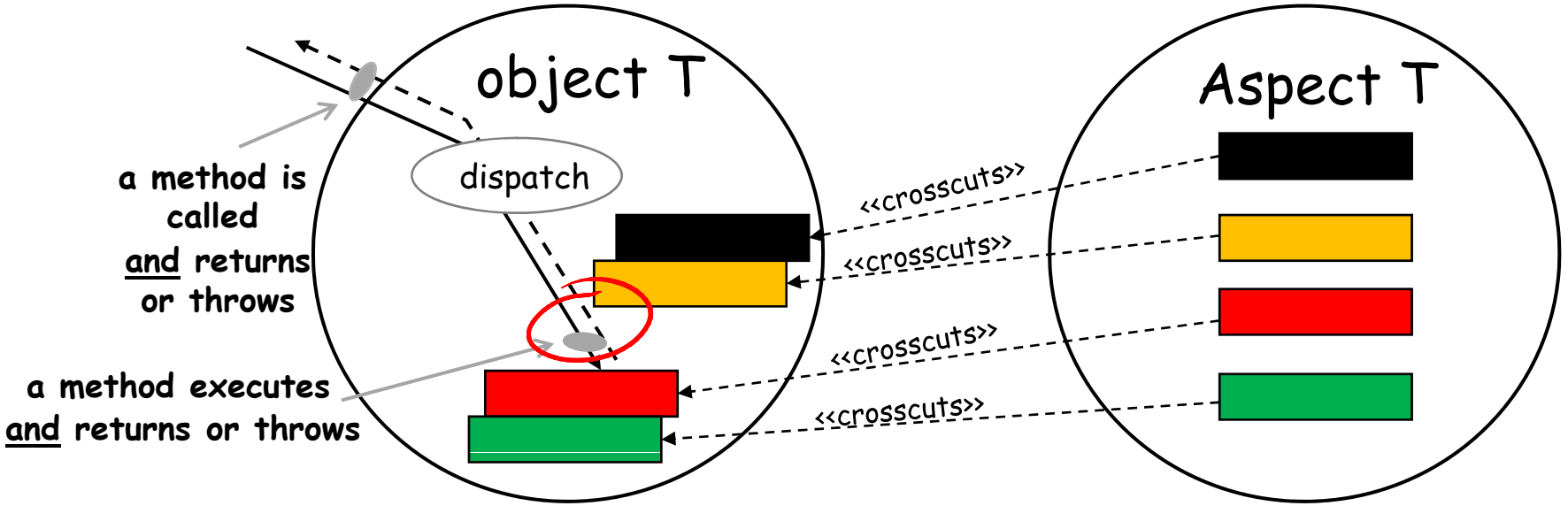
runtime assertion checker

JACK, Krakatoa, LOOP

Correctness proof

In relation to our approach…

ajmlc: an AO JML compiler



Before advices to check invariants

Before advice to check preconditions

After or around advices to check postconditions

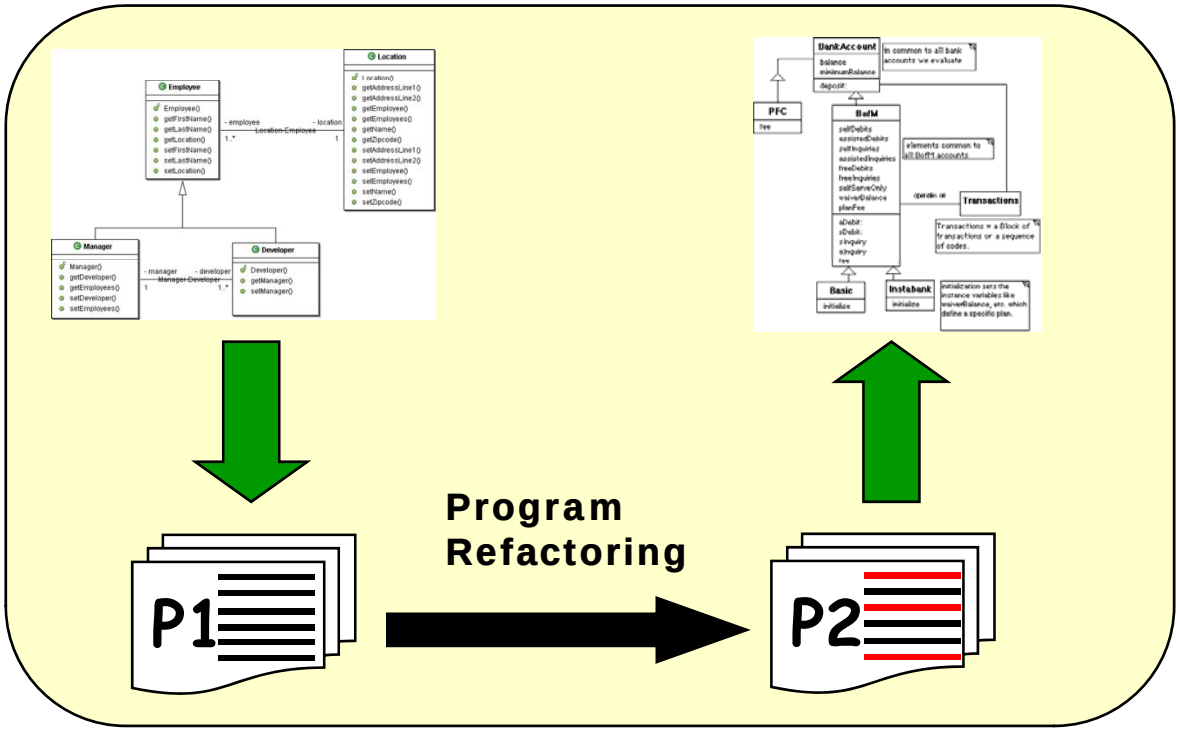
After advices to check invariants

What about refactorings?

First of all...

what are refactorings?

Refactoring is...



- A series of small steps
- A change of code's internal structure
- Improvement of code
- Semantics preserving
- ...

Refactoring and aspects

- Aspect-aware OO refactorings
- Refactorings for turning OO code into AO code
- Refactorings for improving AO code

From Role-based refactoring of crosscutting concerns, Jan Hannemann, Gail Murphy, and Gregor Kiczales AOSD 2005

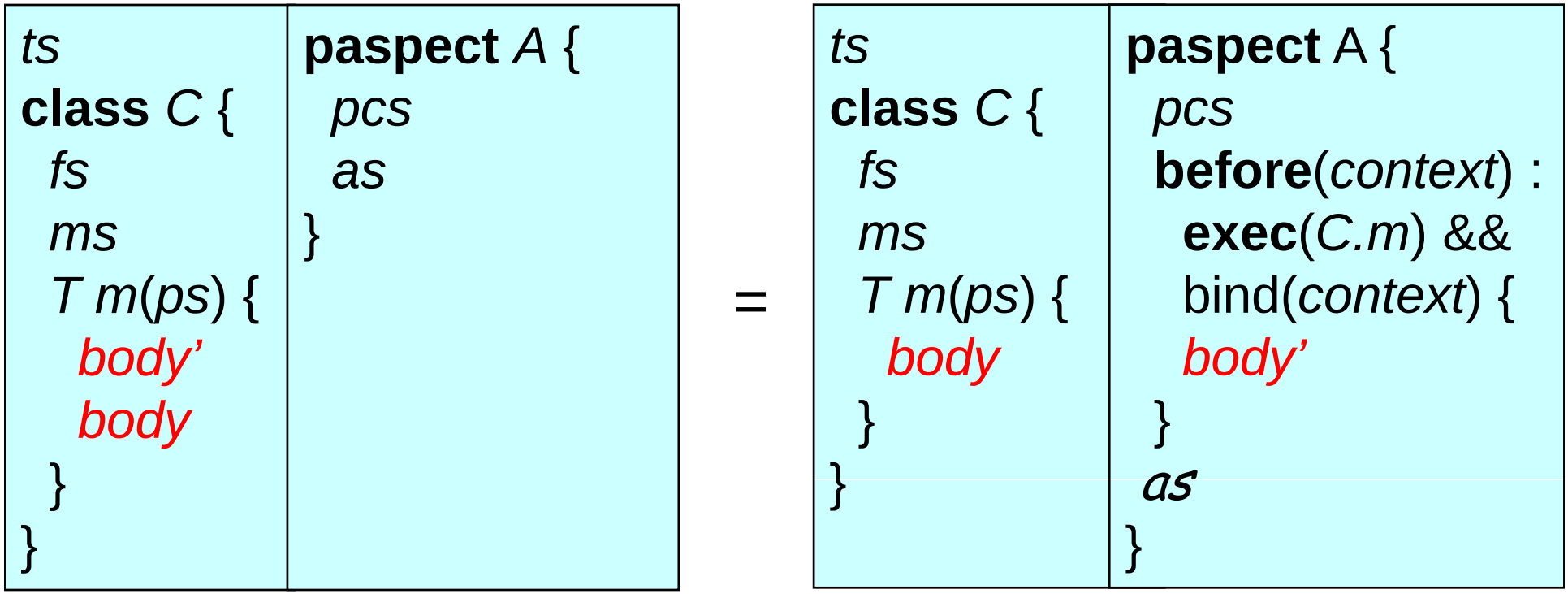
Aspect-aware OO refactorings

```
class C {
  void n(){...body...}
}

aspect A {
  after():
    call(* *m()){
    extra
   }
}
```

```
class C {
  void n(){...m()...}
  void m(){body}
}
aspect A {
  after():
    call(* *m()){
    extra
   }
}
```

Refactorings for turning OO code into AO code



Outline

- Aspect-oriented programming laws
- Aspect-oriented refactorings
- Evaluation using a case study
- Conclusions
- Future work

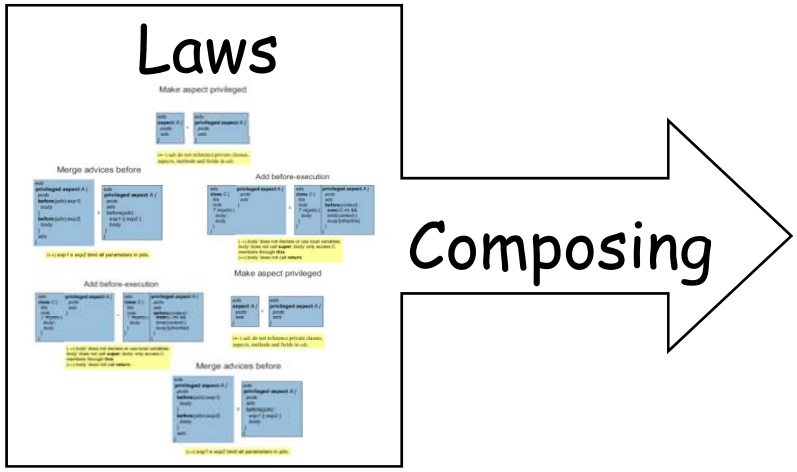
Laws of programming

- ***Laws for imperative programming***
  (Hoare et al. 1987)
- ***Laws for object-oriented programming***
  (Borba et al. 2004)
- ***Laws for aspect-oriented programming***
  (Borba and Cole. 2005)

Our approach

- Primitive laws of programming
  - simple, localized, intuitive and easy to understand
  - one transformation
  - one-way-directional
  - guarded by preconditions

We use laws to...



Law

```
eds                          =   eds
aspect A {                       aspect A {
  pcds                             Pcds
  ads                              ads
  before(pds) : exp {              pointcut p(pds) : exp
    body                           before(pds) :p(αpds) {
  }                                  body
  after(pds) : exp {               }
    body'                          after(pds) :p(αpds){
  }                                  body'
}                                  }
                                 }
```

(→) There is no pointcut named *p* in *pcds*;
(←) *eds*, *ads* and *pcds* do not reference *p*

Refactoring

```
eds                          =   eds
aspect A {                       aspect A {
  pcds                             Pcds
  ads                              ads
  before(pds) : exp {              pointcut p(pds) : exp
    body                           before(pds) :p(αpds) {
  }                                  body
  after(pds) : exp {               }
    body'                          after(pds) :p(αpds){
  }                                  body'
}                                  }
                                 }
```

(→) There is no pointcut named *p* in *pcds*;
(←) *eds*, *ads* and *pcds* do not reference *p*

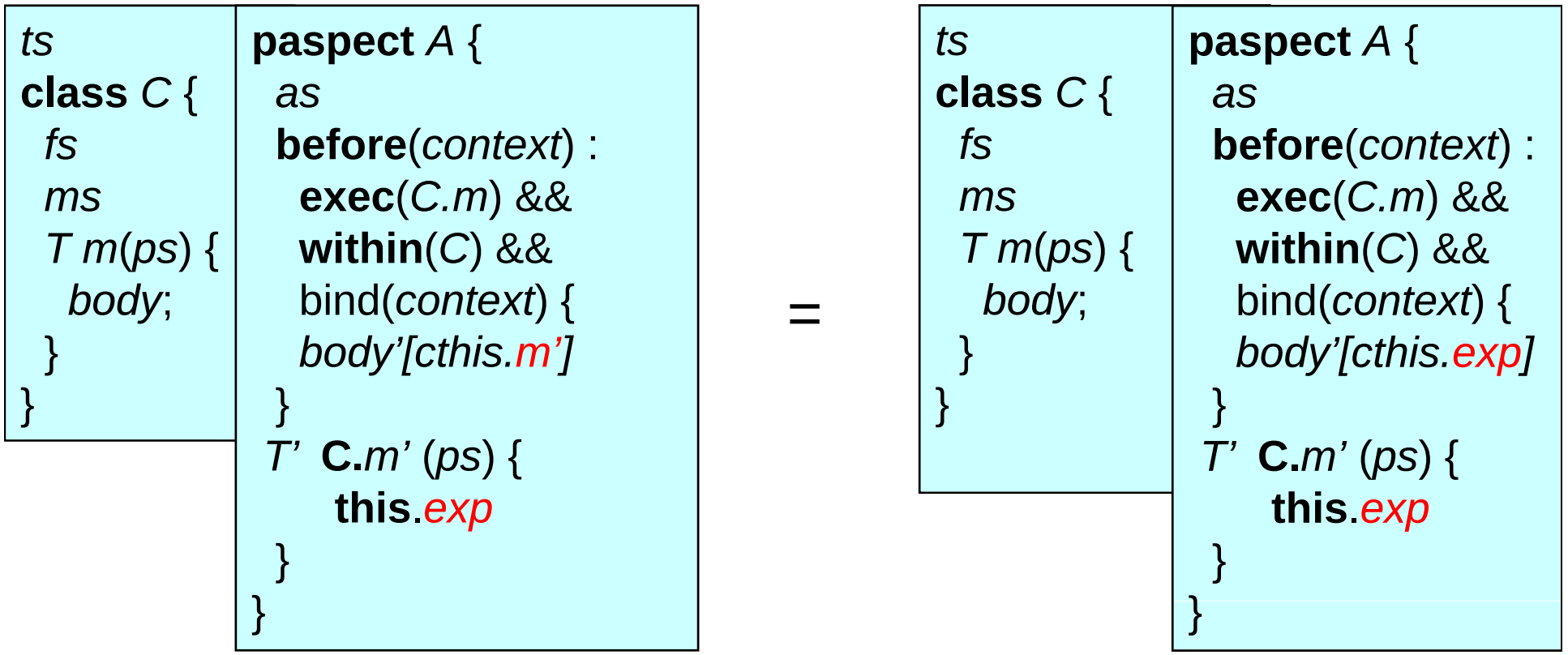
Summary of laws and Refactorings

Law	Name	Refactoring	Name
1	remove empty privileged aspect	1	Inline method ITD within advice
2	Move advice body to other advice	2	Merge distinct advice
3	Replace method ITD reference with methd ITD implementation within advice	3	Split around-execution into after-execution returning and after-execution throwing
4	Remove method ITD related to advice	4	Extract aspect method
5	remove before-execution		
6	remove after-execution-returning		
7	remove before-execution-throwing		
8	remove around-execution		
9	remove this designator		
10	remove within designator		

Law 3

Law	Name	Refactoring	Name
1	remove empty privileged aspect	1	Inline method ITD within advice
2	Move advice body to other advice	2	Merge distinct advice
3	Replace method ITD reference with methd ITD implementation within advice	3	Split around-execution into after-execution returning and after-execution throwing
4	Remove method ITD related to advice	4	Extract aspect method
5	remove before-execution		
6	remove after-execution-returning		
7	remove before-execution-throwing		
8	remove around-execution		
9	remove this designator		
10	remove within designator		

Replace method ITD reference with method ITD implementation within advice

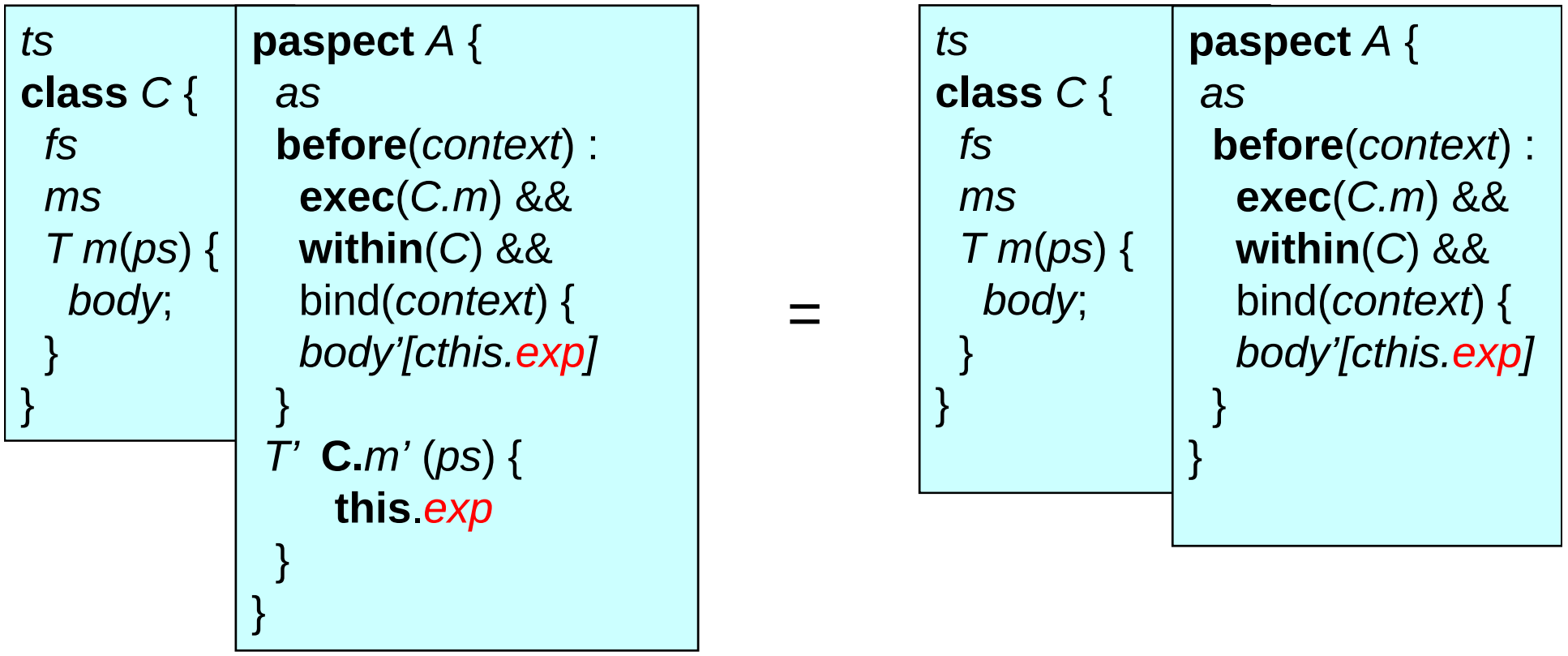


(→) m is referenced from *before* advice.

Law 4

Law	Name	Refactoring	Name
1	remove empty privileged aspect	1	Inline method ITD within advice
2	Move advice body to other advice	2	Merge distinct advice
3	Replace method ITD reference with methd ITD implementation within advice	3	Split around-execution into after-execution returning and after-execution throwing
4	Remove method ITD related to advice	4	Extract aspect method
5	remove before-execution		
6	remove after-execution-returning		
7	remove before-execution-throwing		
8	remove around-execution		
9	remove this designator		
10	remove within designator		

Remove method ITD



(→) `m` is not referenced from *before* advice, `C`, `ts`, or `as`.

Law 5

Law	Name	Refactoring	Name
1	remove empty privileged aspect	1	Inline method ITD within advice
2	Move advice body to other advice	2	Merge distinct advice
3	Replace method ITD reference with methd ITD implementation within advice	3	Split around-execution into after-execution returning and after-execution throwing
4	Remove method ITD related to advice	4	Extract aspect method
5	remove before-execution		
6	remove after-execution-returning		
7	remove before-execution-throwing		
8	remove around-execution		
9	remove this designator		
10	remove within designator		

Remove before-execution

```
ts
class C {
  fs
  ms
  T m(ps) {
    body;
  }
}
```

```
paspect A {
  as
  before(context) :
    exec(C.m) &&
    within(C) &&
    bind(context) {
    body'[cthis.m']
  }
 T'  C.m' (ps) {
       this.exp
  }
```

=

```
ts
class C {
  fs
  ms
  T m(ps) {
    body;
  }
}
```

```
paspect A {
 as
  T'  C.m' (ps) {
       this.exp
  }
}
```

($\rightarrow$) *before* advice does not contribute to execution flow of the affected join point `C.m`, or type `C` is declared `abstract` or it is declared an `interface`.

We derived refactorings from the laws...

Refactoring 1

Law	Name	Refactoring	Name
1	remove empty privileged aspect	1	Inline method ITD within advice
2	Move advice body to other advice	2	Merge distinct advice
3	Replace method ITD reference with methd ITD implementation within advice	3	Split around-execution into after-execution returning and after-execution throwing
4	Remove method ITD related to advice	4	Extract aspect method
5	remove before-execution		
6	remove after-execution-returning		
7	remove before-execution-throwing		
8	remove around-execution		
9	remove this designator		
10	remove within designator		

Inline method ITD within before-execution

```
ts
class C {
  fs
  ms
  T m(ps) {
    body;
  }
}
```

```
paspect A {
  as
  before(context) :
    exec(C.m) &&
    within(C) &&
    bind(context) {
    body'[cthis.m']
  }
 T'  C.m' (ps) {
       this.exp
  }
}
```

=

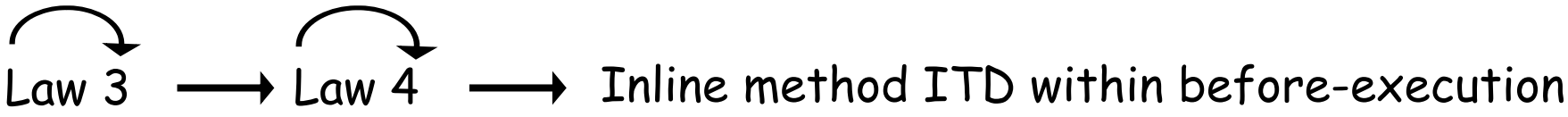
```
ts
class C
  fs
  ms
  T m(ps)
    body;
  }
}
```

```
paspect A {
 as
 before(context) :
    exec(C.m) &&
    within(C) &&
    bind(context) {
    body'[cthis.exp]
  }
}
```

(→) `m` is not referenced from `C`, `ts`, or `as`.

Inline method ITD within before-execution

- Summary
  - Law 3 – replace method ITD reference with method ITD implementation within advice
  - Law 4 – remove method ITD



Summary of laws and Refactorings

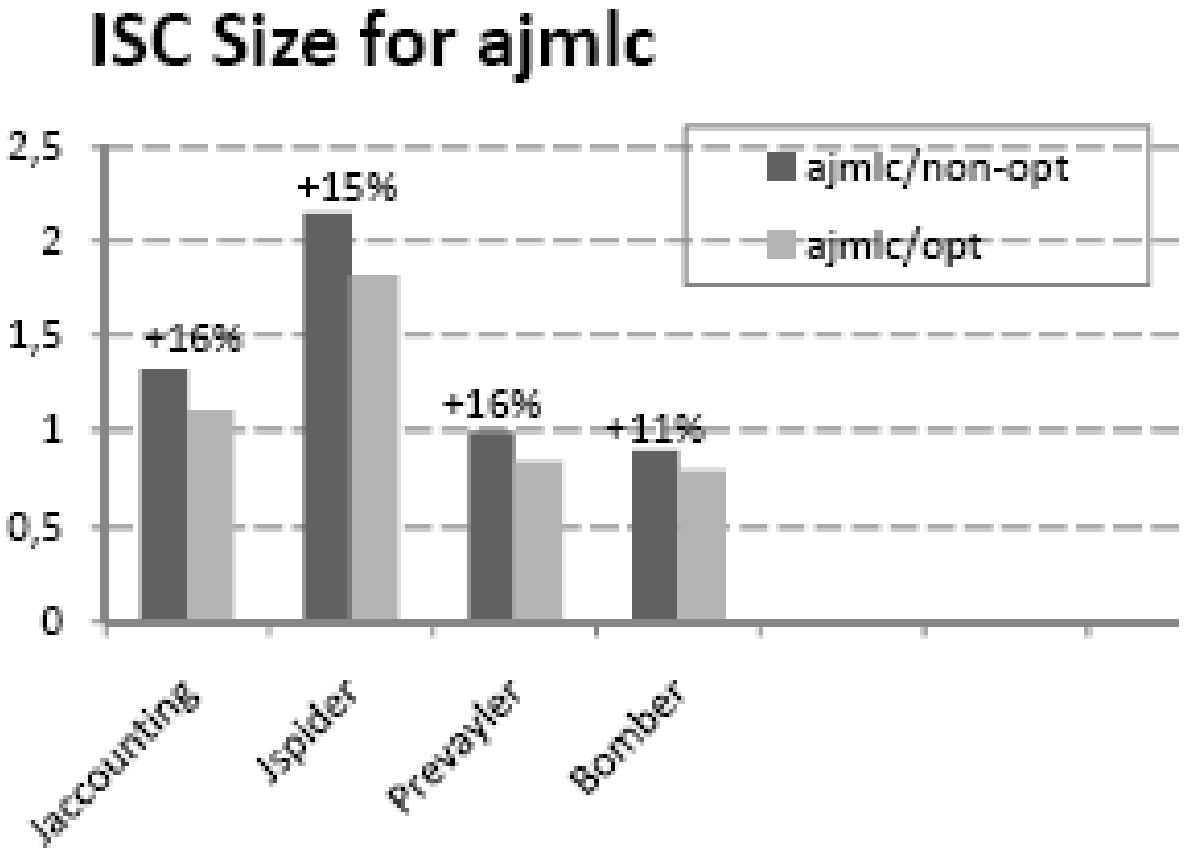
Law	Name	Refactoring	Name
1	remove empty privileged aspect	1	Inline method ITD within advice
2	Move advice body to other advice	2	Merge distinct advice
3	Replace method ITD reference with methd ITD implementation within advice	3	Split around-execution into after-execution returning and after-execution throwing
4	Remove method ITD related to advice	4	Extract aspect method
5	remove before-execution		
6	remove after-execution-returning		
7	remove before-execution-throwing		
8	remove around-execution		
9	remove this designator		
10	remove within designator		

Evaluation with a case study

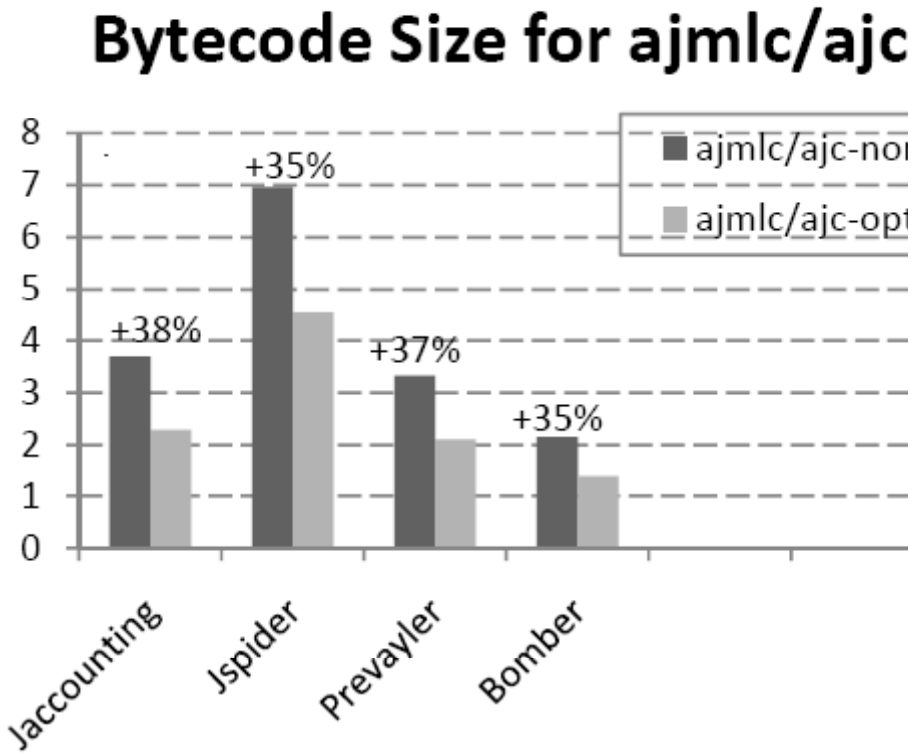
- Four real Java systems specified with JML
  - Jaccounting, Jspider, Prevayler, Bomber

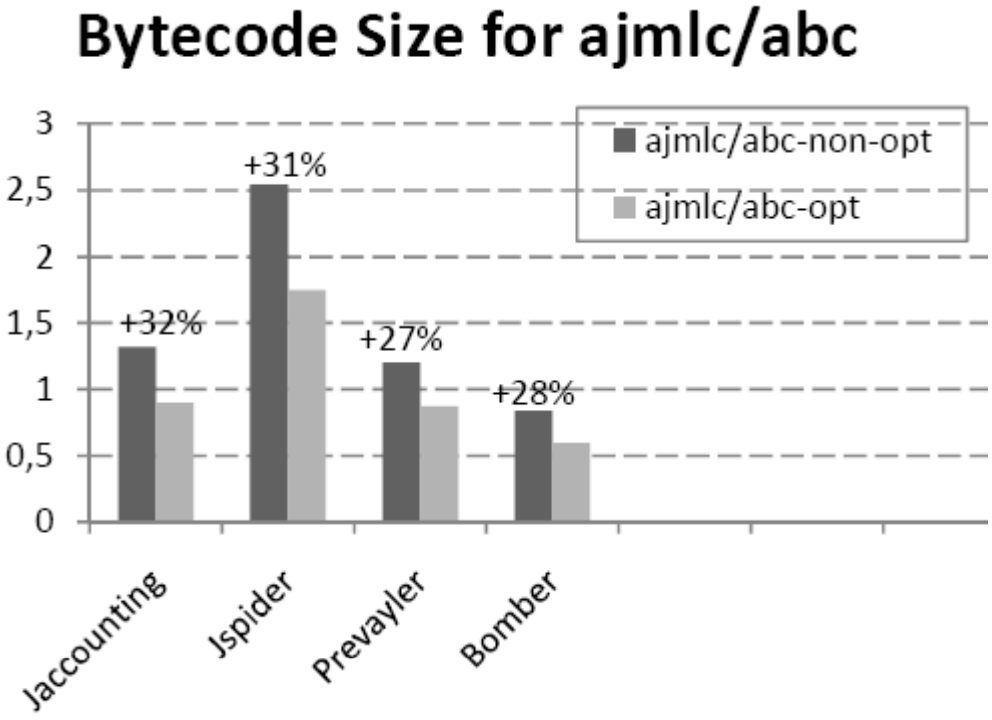
We evaluated the ajmlc compiler with and without the proposed laws/refactorings considering code size and running time

Results—ISC size

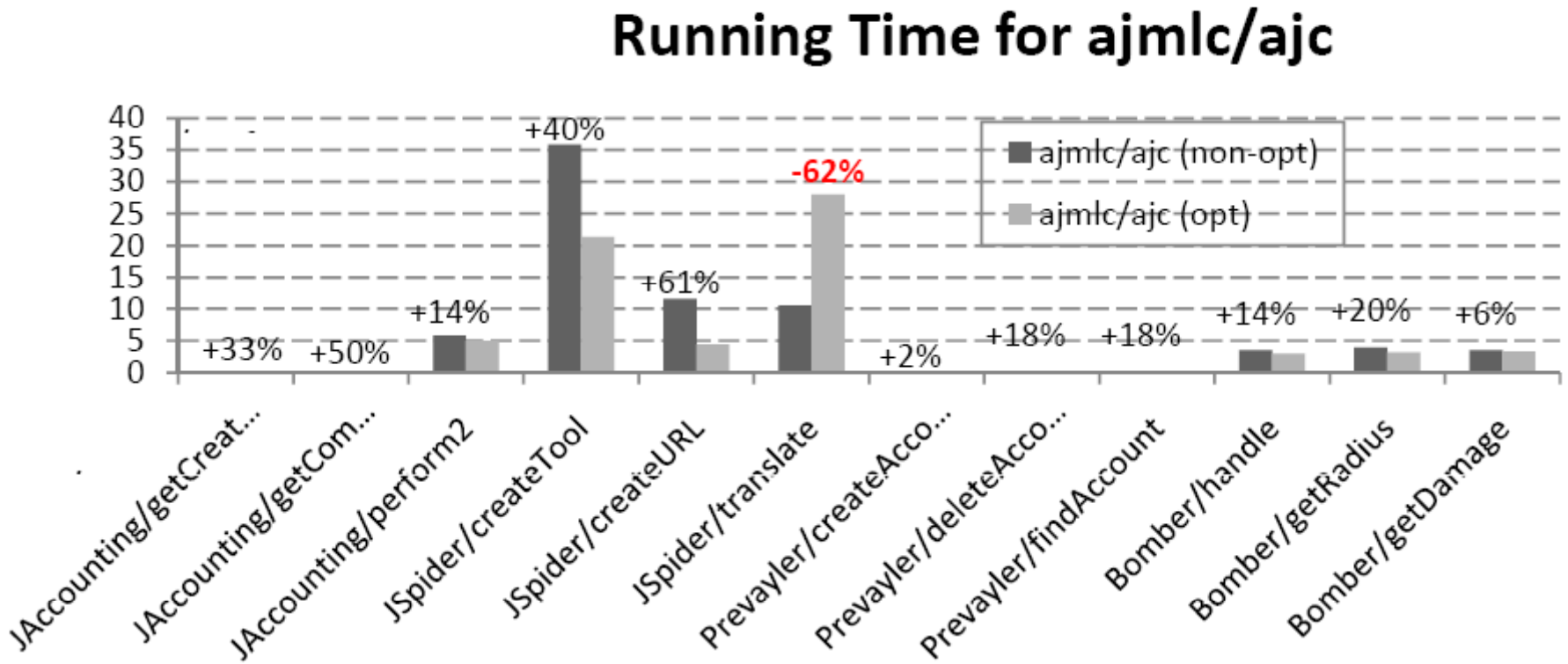


Results—Bytecode size

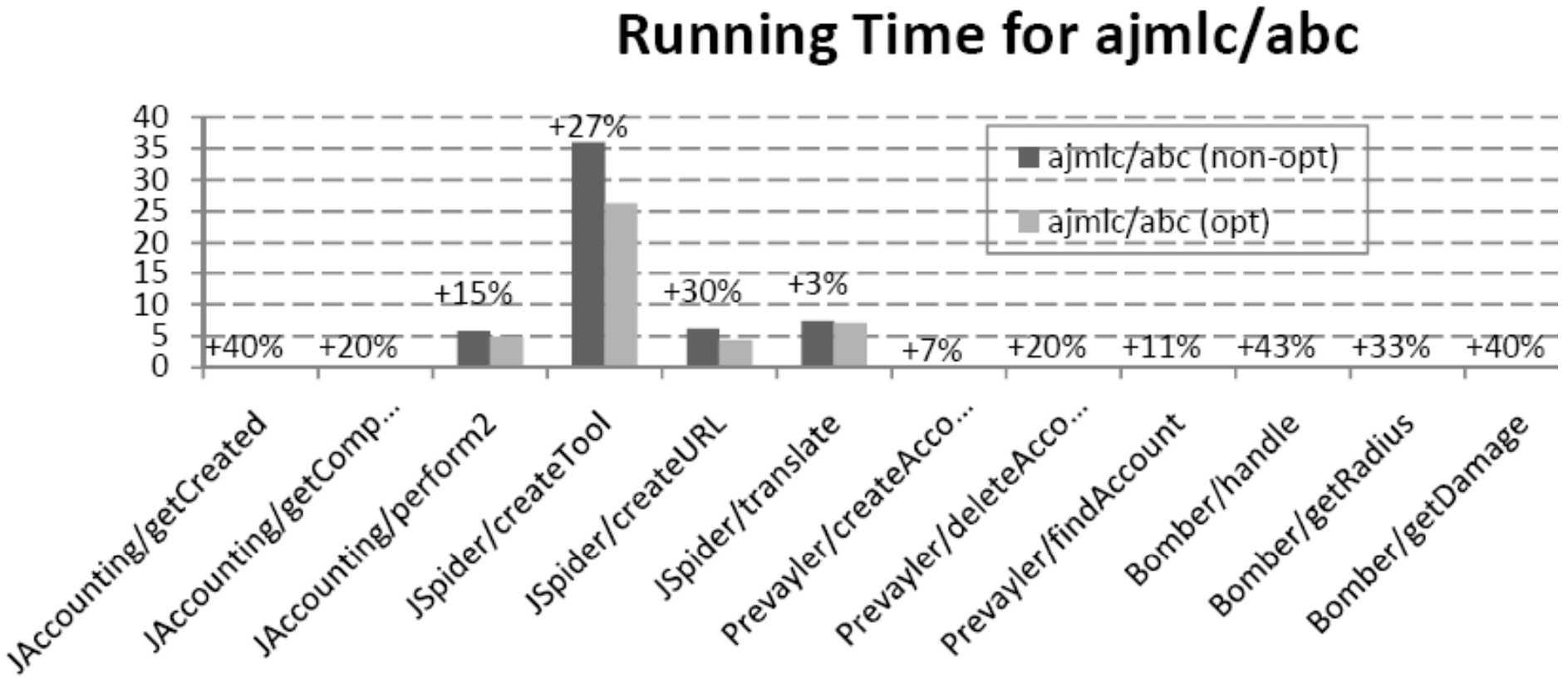




Results—Running time (ajc)



Results—Running time (abc)



Conclusions

- AOP laws useful for deriving aspect-oriented laws/refactorings
- Equivalence notion for aspect-oriented programs
- Informal argumentation about the soundness of some laws

Conclusions

- Main contribuitions:
  - ajmlc compiler optimization (AO code)
  - first work that concerns assertion checking code optimization
- The evaluation with a case study involving four real Java systems
  - Jaccounting, Jspider, Prevayler, and Bomber

Future work

- Current definition relies on the simplicity and intuition of the laws

Formally prove the laws to show that they preserve behaviour

Online material

- The resources used in this work are available from the JML AOP project

http://www.cin.ufpe.br/~hemr/JMLAOP/sblp09

Optimizing JML Feature Compilation using AO Refactorings

Henrique Rebêlo
Ricardo Massa F. Lima
Alexandre Mota
César Oliveira

Federal University of Pernambuco
(hemr, rmfl, acm, calo @cin.ufpe.br)

Gary T. Leavens
University of Central Florida
(leavens@eecs.ucf.edu)

Márcio Cornélio
University of Pernambuco
(mlc@dsc.upe.br)

Law 1

Law	Name	Refactoring	Name
1	remove empty privileged aspect	1	Inline method ITD within advice
2	Move advice body to other advice	2	Merge distinct advice
3	Replace method ITD reference with methd ITD implementation within advice	3	Split around-execution into after-execution returning and after-execution throwing
4	Remove method ITD related to advice	4	Extract aspect method
5	remove before-execution		
6	remove after-execution-returning		
7	remove before-execution-throwing		
8	remove around-execution		
9	remove this designator		
10	remove within designator		

Remove empty privileged aspect

ts
paspect *A* {
}

=

ts

(→) A is not referenced from `ts`.

Remove empty privileged aspect

- Summary
  - Law 2 – Make aspect privileged
  - Law 1 – Add empty aspect

Law 2

Law	Name	Refactoring	Name
1	remove empty privileged aspect	1	Inline method ITD within advice
2	Move advice body to other advice	2	Merge distinct advice
3	Replace method ITD reference with methd ITD implementation within advice	3	Split around-execution into after-execution returning and after-execution throwing
4	Remove method ITD related to advice	4	Extract aspect method
5	remove before-execution		
6	remove after-execution-returning		
7	remove before-execution-throwing		
8	remove around-execution		
9	remove this designator		
10	remove within designator		

Move advice body to other advice

```
ts
class C {
  fs
  ms
  T m(ps) {
    body;
  }
}
```

```
paspect A {
  as
  before(context) :
    exec(C.m) &&
    within(C) &&
    bind(context) {
    body'
  }
  T' around(context) :
    exec(C.m) &&
    within(C) &&
    bind(context) {
    proceed(context)
    body''
  }
}
```

=

```
ts
class C
  fs
  ms
  T m(ps)
    body;
  }
}
```

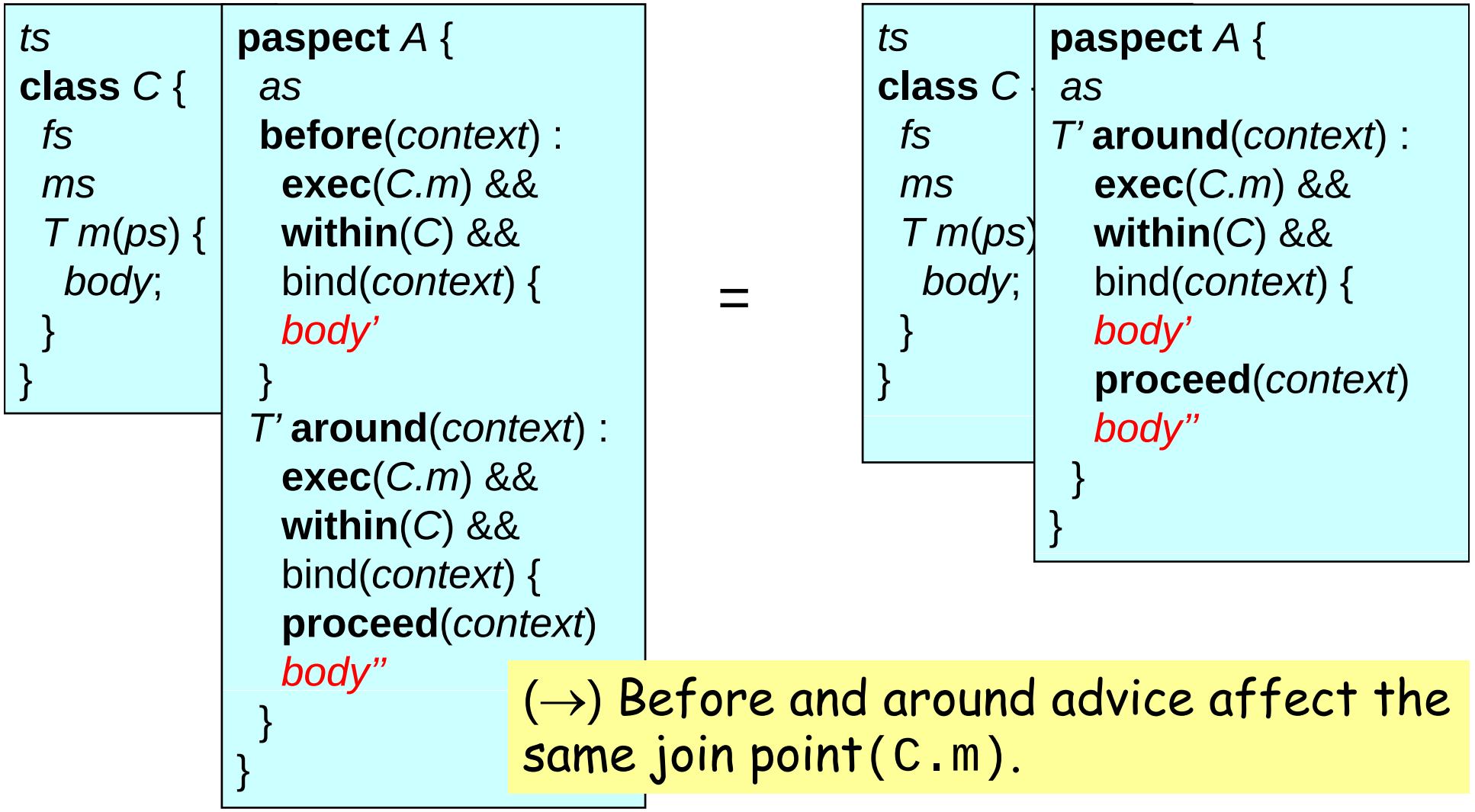
```
paspect A {
 as
 before(context) :
    exec(C.m) &&
    within(C) &&
    bind(context) {
 }
T' around(context) :
    exec(C.m) &&
    within(C) &&
    bind(context) {
    body'
    proceed(context)
    body''
  }
```

(→) Before and around advice affect the same join point(C.m).

Refactoring 2

Law	Name	Refactoring	Name
1	remove empty privileged aspect	1	Inline method ITD within advice
2	Move advice body to other advice	2	Merge distinct advice
3	Replace method ITD reference with methd ITD implementation within advice	3	Split around-execution into after-execution returning and after-execution throwing
4	Remove method ITD related to advice	4	Extract aspect method
5	remove before-execution		
6	remove after-execution-returning		
7	remove before-execution-throwing		
8	remove around-execution		
9	remove this designator		
10	remove within designator		

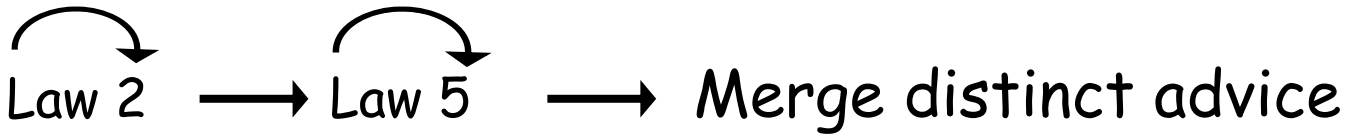
Merge distinct advice



Merge distinct advice

(Cole and Borba 2005)

- Summary
  - Law 2 – move advice body to other advice
  - Law 5 – remove before-execution



Refactoring 3

Law	Name	Refactoring	Name
1	remove empty privileged aspect	1	Inline method ITD within advice
2	Move advice body to other advice	2	Merge distinct advice
3	Replace method ITD reference with methd ITD implementation within advice	3	Split around-execution into after-execution returning and after-execution throwing
4	Remove method ITD related to advice	4	Extract aspect method
5	remove before-execution		
6	remove after-execution-returning		
7	remove before-execution-throwing		
8	remove around-execution		
9	remove this designator		
10	remove within designator		

Split around-execution into after-exec. returning and after-exec. throwing

```
ts
class C {
  fs
  ms
  T m(ps) {
    body;
  }
}
```

```
paspect A {
  as
T' around(context) :
    exec(C.m) &&
    within(C) &&
    bind(context) {
    try {
      proceed(context)
      body'
  } catch(E e) {
      body"
      Throw e
```

=

```
ts
class C
  fs
  ms
  T m(ps)
    body;
  }
}
```

```
paspect A {
 as
after(context)
returning (T' t):
    exec(C.m) &&
    within(C) &&
    bind(context) {
    body'
  }
after(context)
thowing (E e):
    exec(C.m) &&
    within(C) &&
    bind(context) {
    body"
  }
}
```

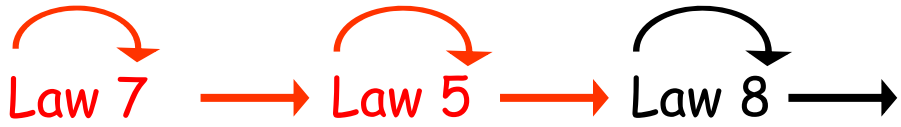
(→) Around advice does not replace the orginal join point (`C.m`) execution;

(→) Around advice does not use pre-state variables.

Split around-execution into after-exec. returning and after-exec. throwing

- Summary
  - Law 7 – add after-execution returning
  - Law 5 – add after-execution throwing
  - Law 8 – remove around-execution

(Cole and Borba 2005)

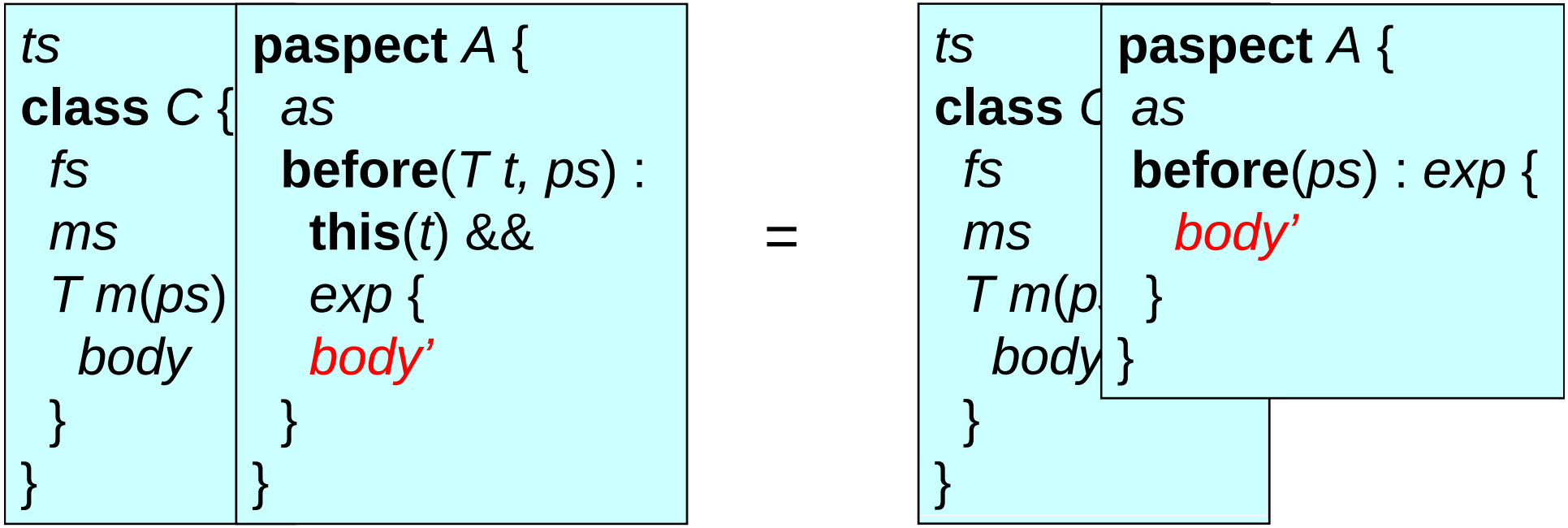


Split around-execution into after-execution returning and after-execution throwing

Law 9

Law	Name	Refactoring	Name
1	remove empty privileged aspect	1	Inline method ITD within advice
2	Move advice body to other advice	2	Merge distinct advice
3	Replace method ITD reference with methd ITD implementation within advice	3	Split around-execution into after-execution returning and after-execution throwing
4	Remove method ITD related to advice	4	Extract aspect method
5	remove before-execution		
6	remove after-execution-returning		
7	remove before-execution-throwing		
8	remove around-execution		
9	remove this designator		
10	remove within designator		

Remove this designator

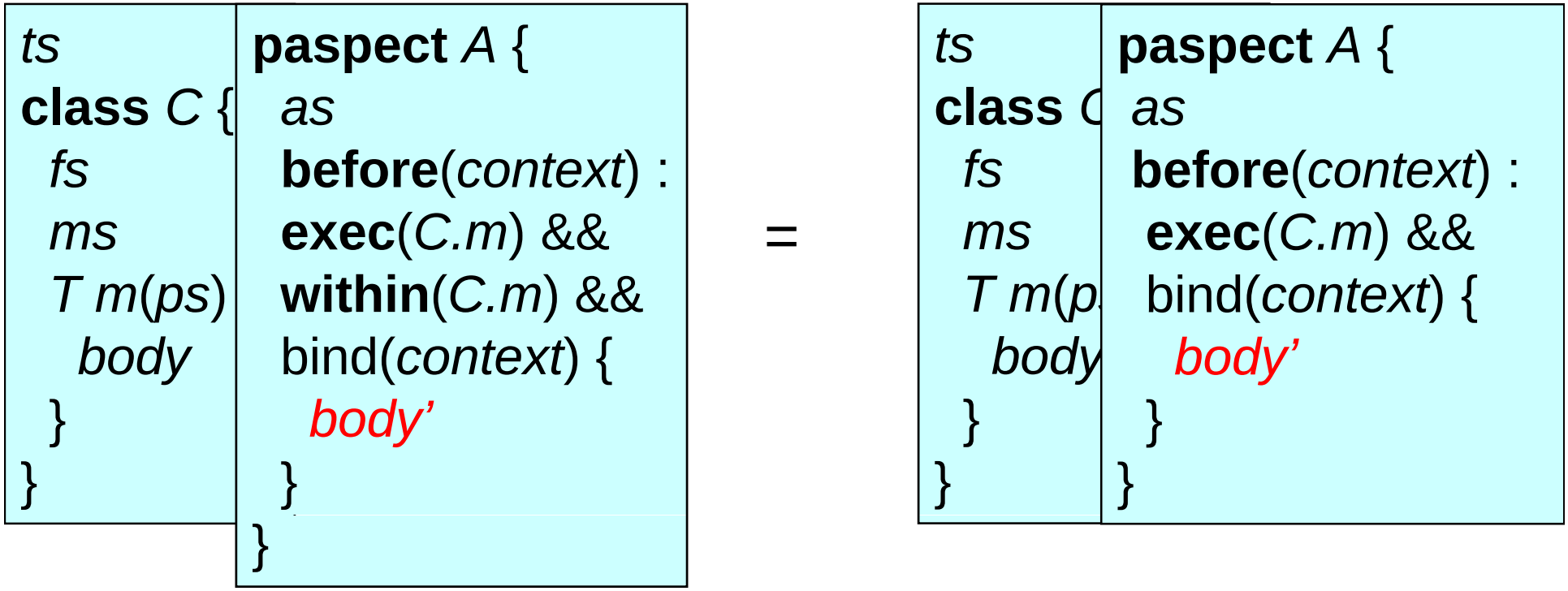


(→) (C.m) is declared static.

Law 10

Law	Name	Refactoring	Name
1	remove empty privileged aspect	1	Inline method ITD within advice
2	Move advice body to other advice	2	Merge distinct advice
3	Replace method ITD reference with methd ITD implementation within advice	3	Split around-execution into after-execution returning and after-execution throwing
4	Remove method ITD related to advice	4	Extract aspect method
5	remove before-execution		
6	remove after-execution-returning		
7	remove before-execution-throwing		
8	remove around-execution		
9	remove this designator		
10	remove within designator		

Remove within designator



(→) (C.m) is declared static.