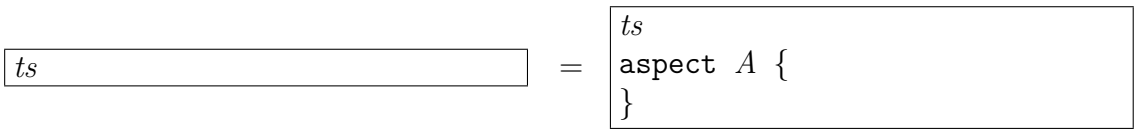
APPENDIX A

LAWS

Law 1. Add empty aspect

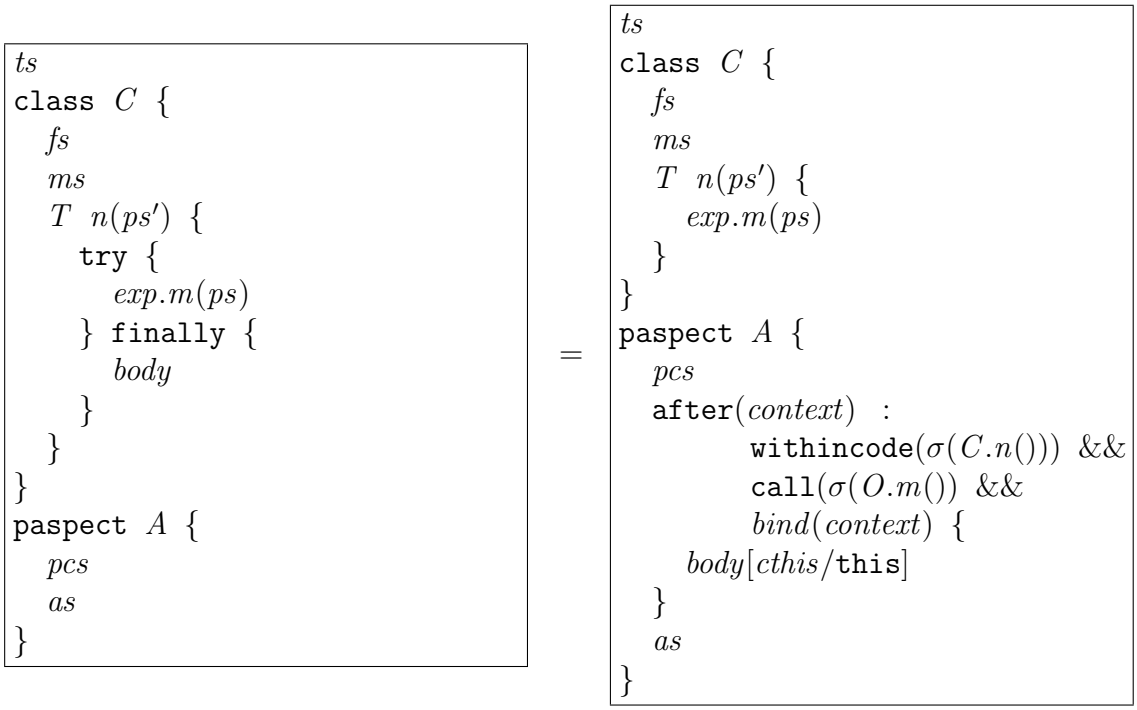


provided

($\rightarrow$) *ts* does not declare any class or aspect named *A*

($\leftarrow$) *A* is not referenced from *ts*.

Law 6. Add After-Call



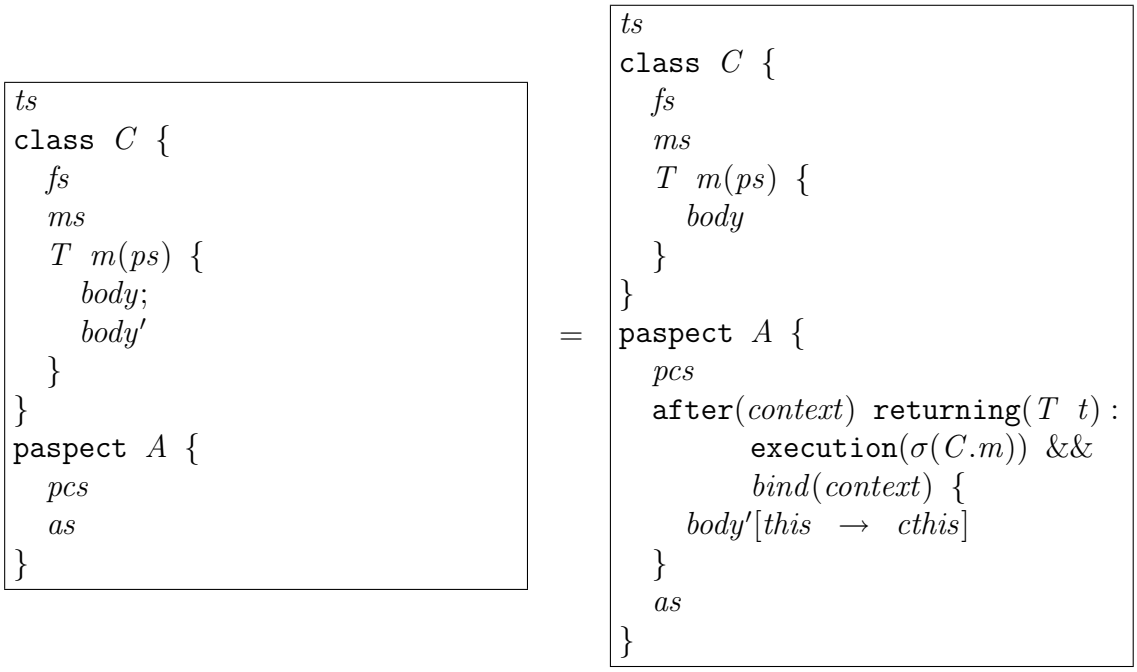
provided

($\rightarrow$) *body′* does not declare or use local variables; *body′* does not call `super`;

($\leftarrow$) *body′* does not call `return`;

($\leftrightarrow$) O is the type of *exp*

Law 7. Add After-Execution Returning Successfully

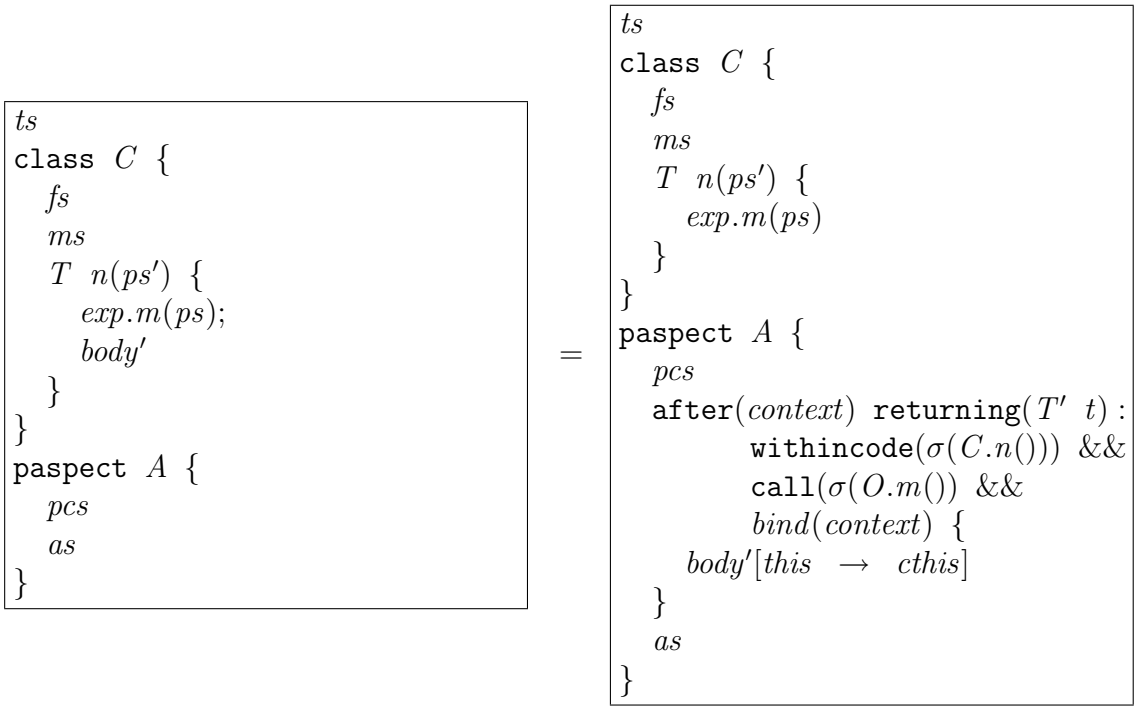


provided

(→) *body′* does not declare or use local variables; *body′* does not call `super`;

(←) *body′* does not call `return`;

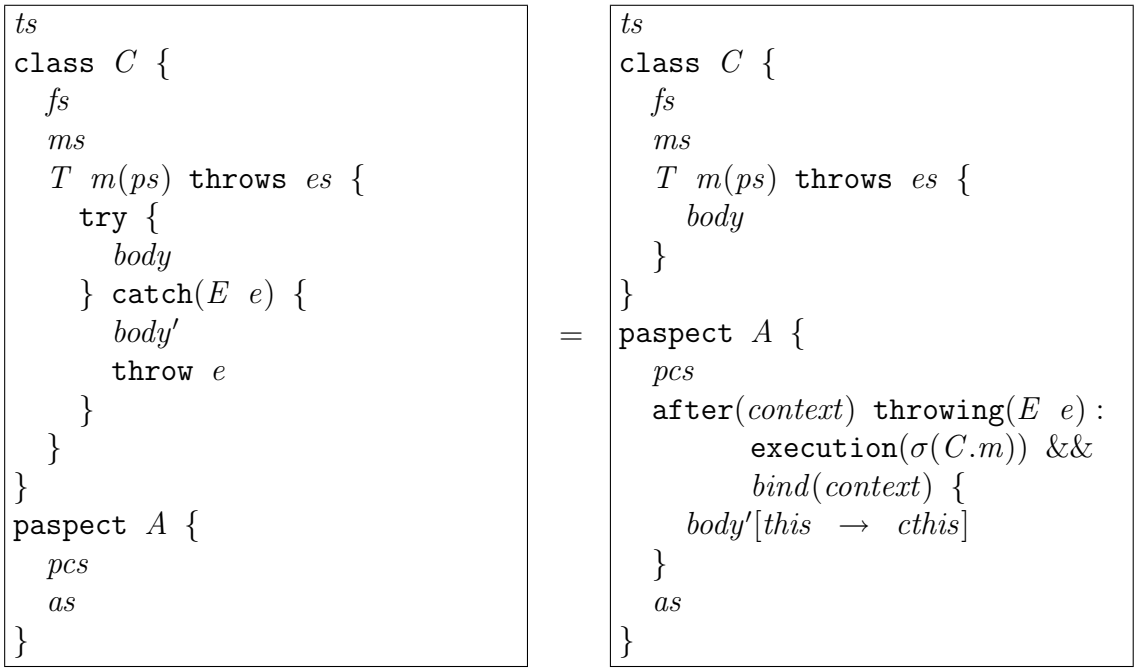
Law 8. Add After-Call Returning Successfully



provided

- (→) *body′* does not declare or use local variables; *body′* does not call `super`; T' is the return type of method m
- (←) *body′* does not call `return`;
- (↔) O is the type of *exp*

Law 9. Add After-Execution Throwing Exception

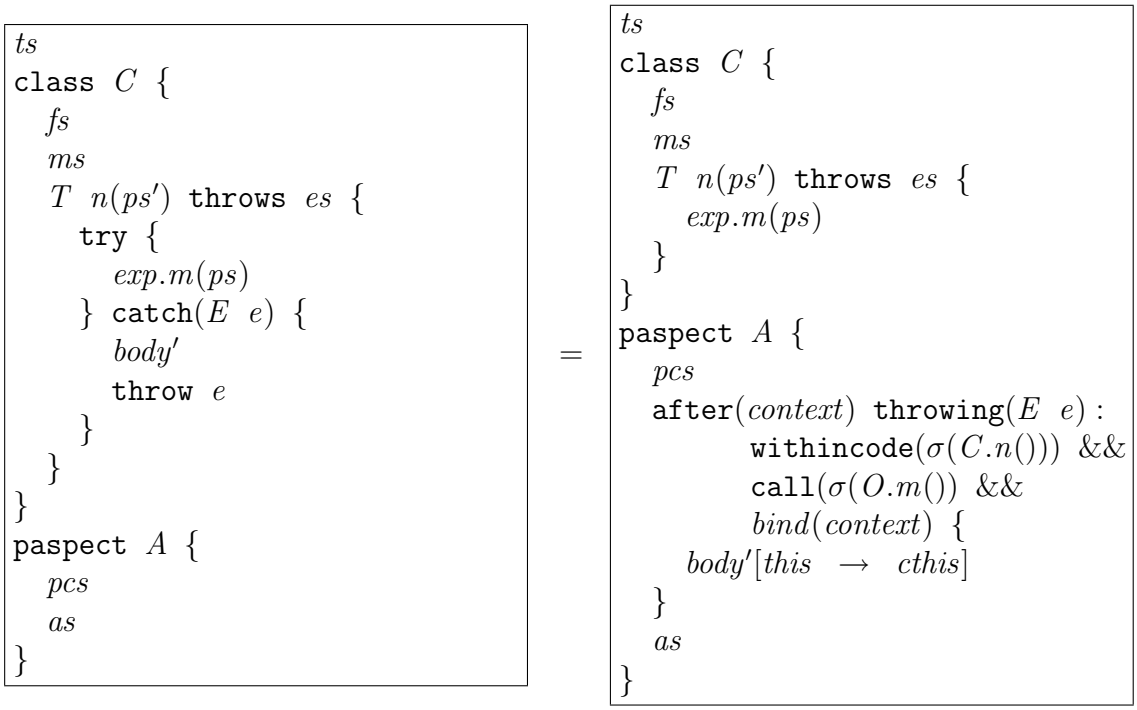


provided

(→) *body′* does not declare or use local variables; *body′* does not call `super`;

(←) *body′* does not call `return`;

Law 10. Add After-Call Throwing Exception



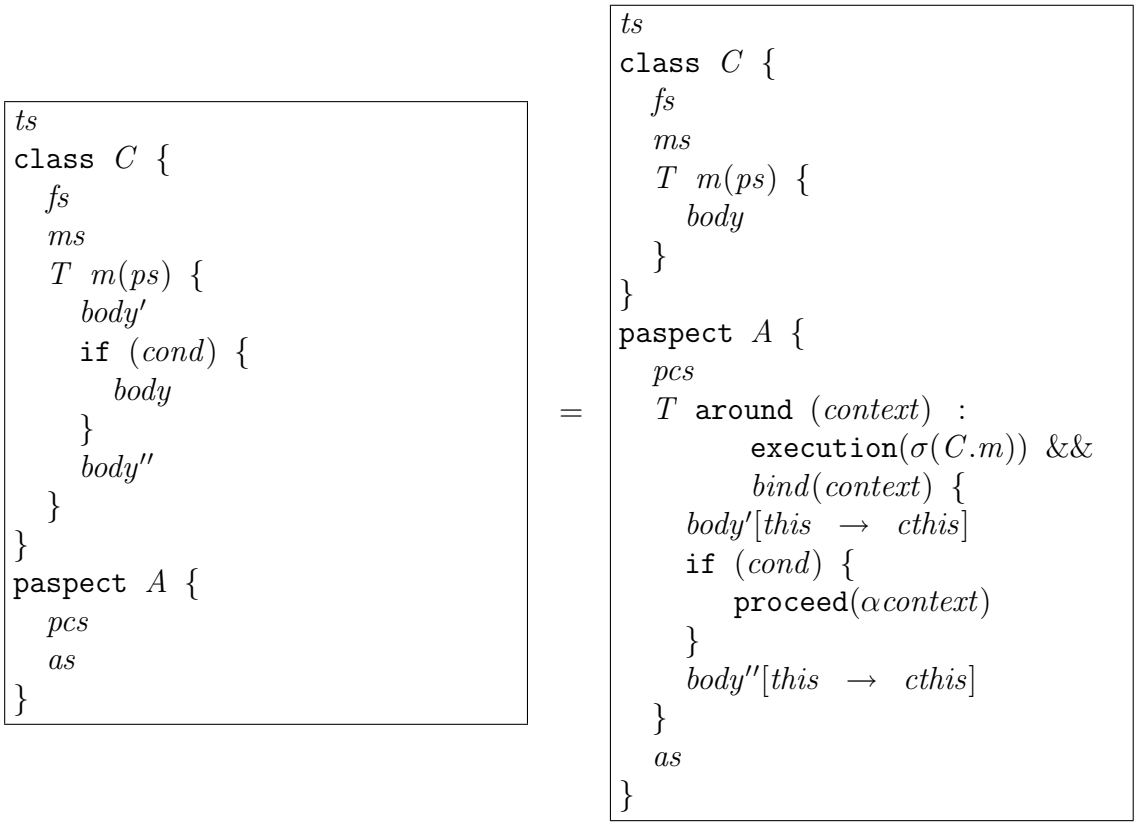
provided

(→) *body′* does not declare or use local variables; *body′* does not call `super`;

(←) *body′* does not call `return`;

(↔) *O* is the type of *exp*

Law 11. Add Around-Execution

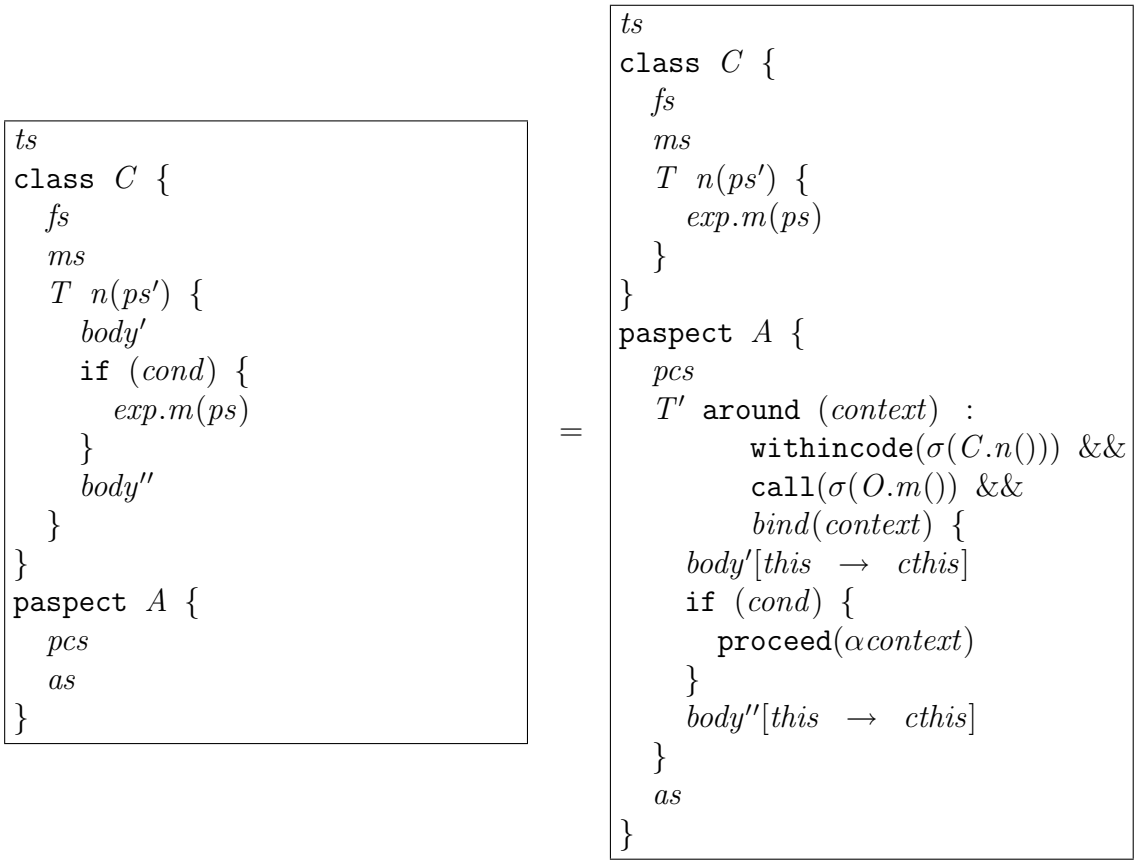


provided

(→) *body′*, *body″* and *cond* do not declare or use local variables; and do not call `super`;

(←) *body′* does not call `return`;

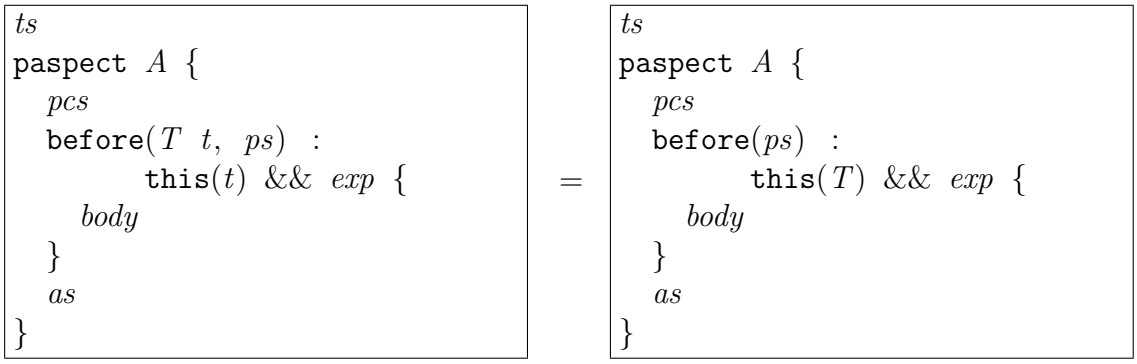
Law 12. Add Around-Call



provided

- (→) *body′*, *body“* and *cond* do not declare or use local variables; and do not call `super`;
- (←) *body′* does not call `return`;
- (↔) *O* is the type of *exp*

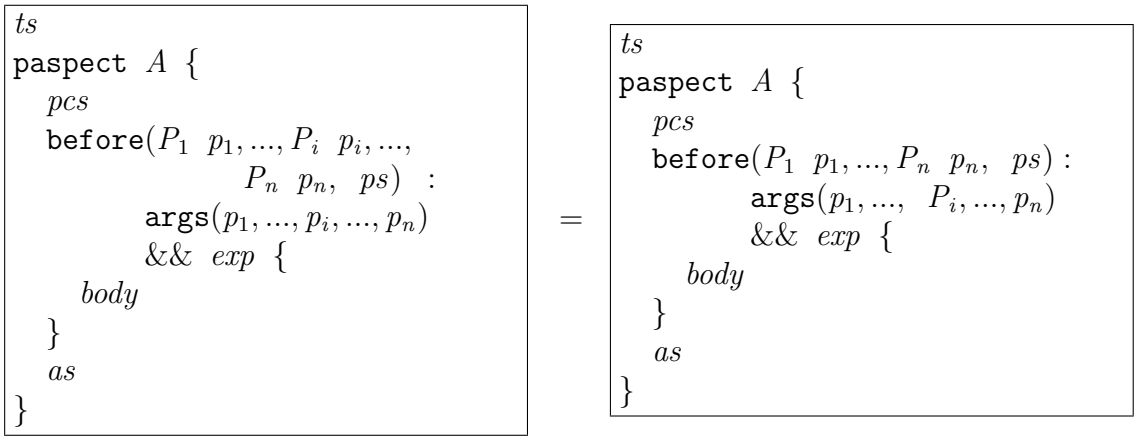
Law 14. Remove this Parameter



provided

($\rightarrow$) t is not referenced from *body*

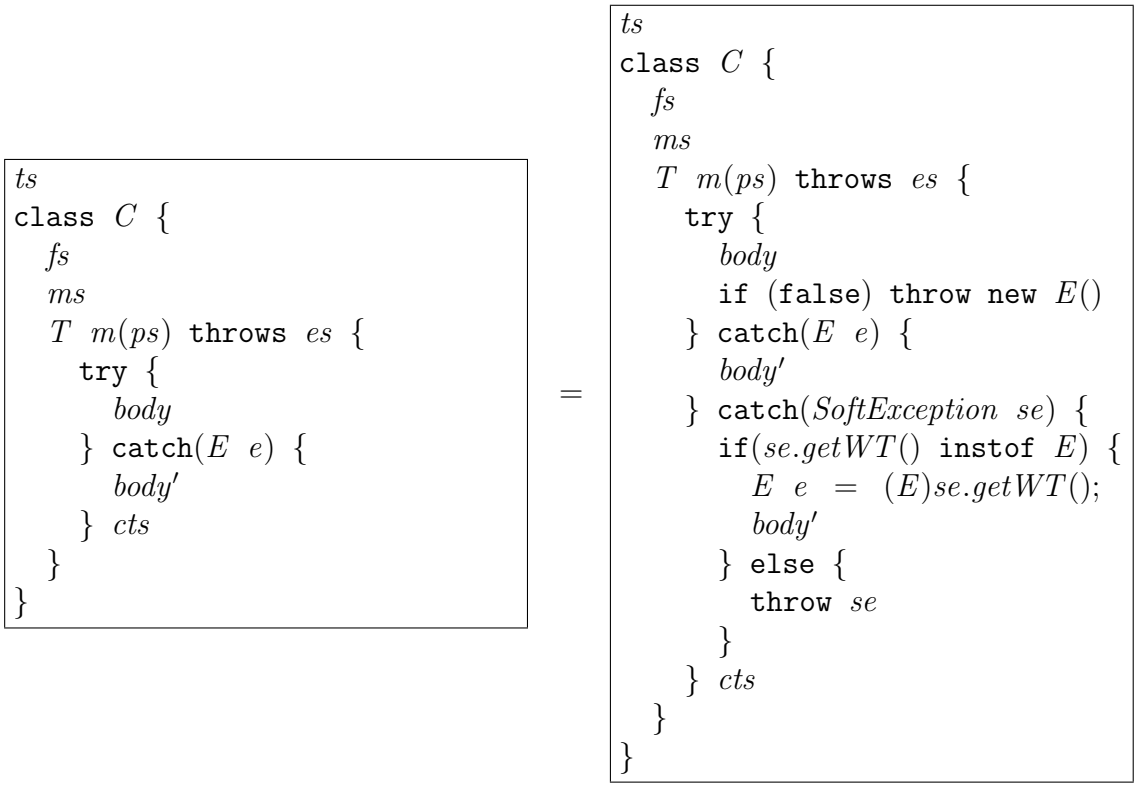
Law 16. Remove Argument Parameter



provided

($\rightarrow$) p_i is not referenced from *body*

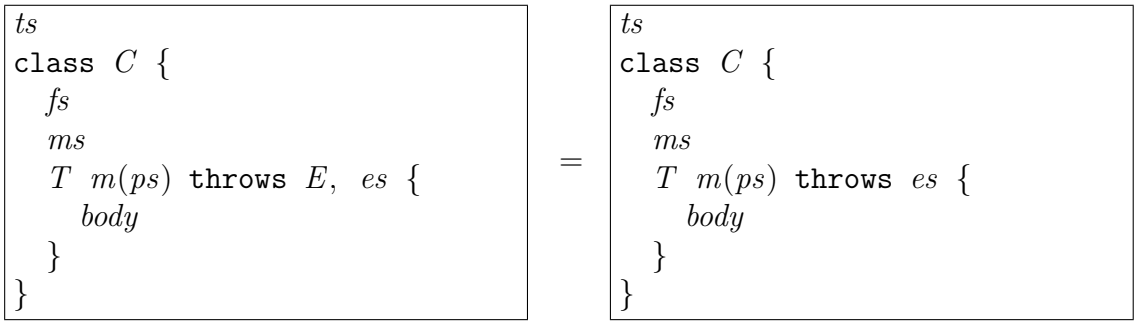
Law 17. Add Catch for Softened Exception



provided

($\leftrightarrow$) E is not declared as soft in any ts joinpoint

Law 19. Remove Exception from Throws Clause

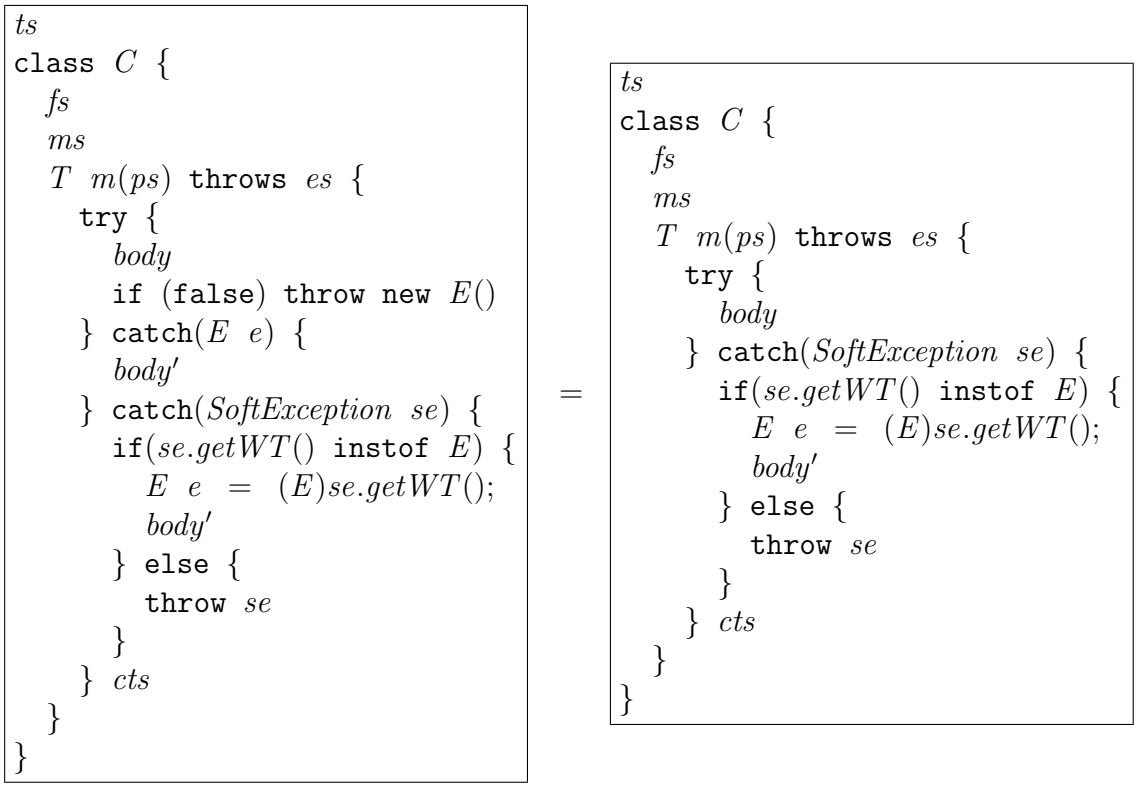


provided

(↔) *body* does not throw E

(←) Every reference to method m of class C catches or throws E, including the super method, if that is the case

Law 20. Remove Exception Handling



provided

(↔) *body* does not throw E

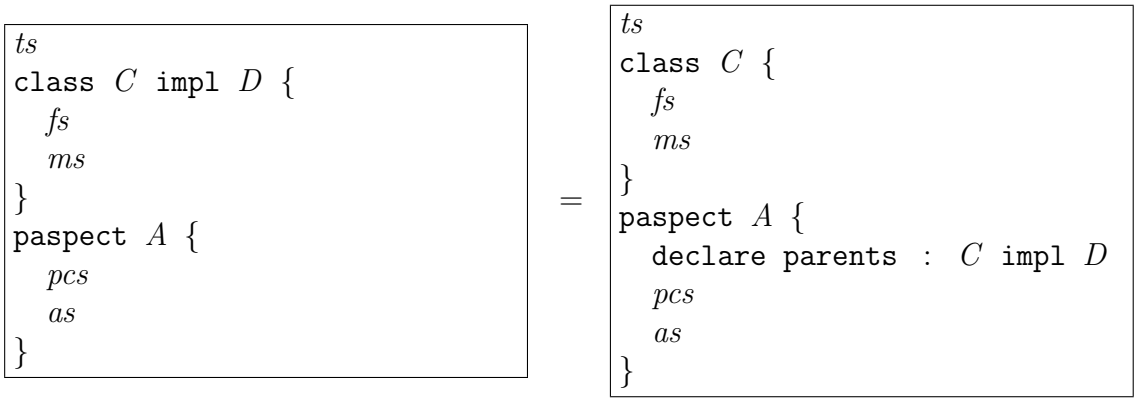
Law 21. Move Exception Handling to Aspect

provided

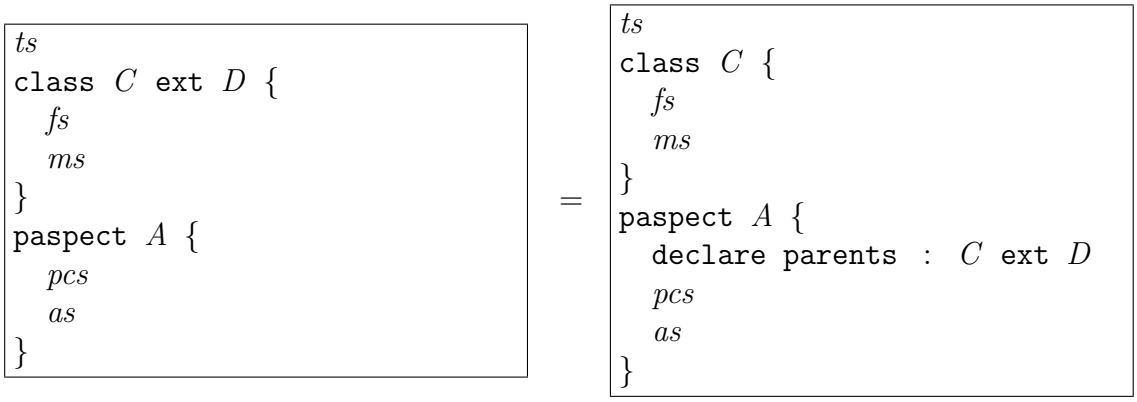
(→) *body′* does not declare or use local variables; *body′* does not call `super`;

(←) *body′* does not call `return`;

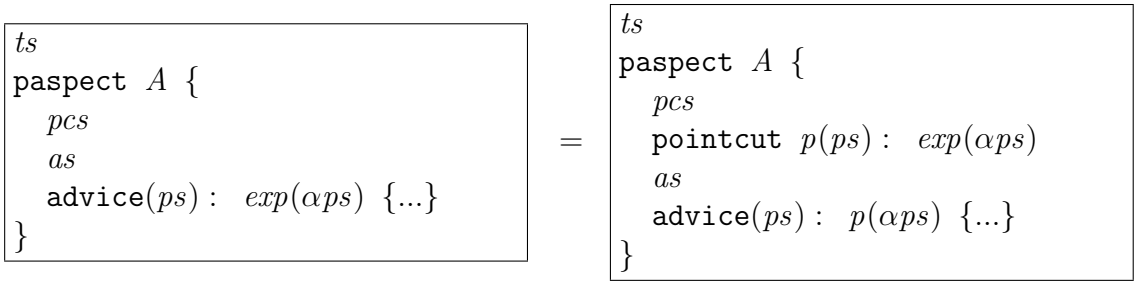
Law 24. Move Implements Declaration to Aspect



Law 25. Move Extends Declaration to Aspect



Law 26. Extract Named Pointcut



provided

(→) There is no pointcut named p in pcs

(←) There is no advice in as that references p

Law 27. Use Named Pointcut

```
ts
paspect A {
  pcs
  pointcut p(ps) : exp(αps)
  as
  advice(ps) : exp(αps) {...}
}
```

=

```
ts
paspect A {
  pcs
  pointcut p(ps) : exp(αps)
  as
  advice(ps) : p(αps) {...}
}
```

Law 28. Move Field Up to Interface

```
ts
interface D {...}
class C impl D {...}
paspect A {
  pcs
  T C.field
  as
}
```

=

```
ts
interface D {...}
class C impl D {...}
paspect A {
  pcs
  T D.field
  as
}
```

provided

(→) A does not already introduce an attribute named *field* to interface D

(←) A does not introduce any method to interface D that references *field*

Law 29. Move Method Up to Interface

```
ts
interface D {...}
class C impl D {...}
paspect A {
  pcs
  T C.m(ps) {
    body
  }
  as
}
```

=

```
ts
interface D {...}
class C impl D {...}
paspect A {
  pcs
  T D.m(ps) {
    body
  }
  as
}
```

provided

(→) Neither A or any other aspect in ts introduce a method named m to interface D

(←) Method m is not referenced in any implementation of interface D other then C

Law 30. Remove Method Implementation