

APPENDIX A

LAWS

Law 1. Add empty aspect

$$\boxed{ts} = \boxed{\begin{array}{l} ts \\ \text{aspect } A \{ \\ \} \end{array}}$$

provided

- (\rightarrow) ts does not declare any class or aspect named A
- (\leftarrow) A is not referenced from ts .

Law 6. Add After-Call

```

ts
class C {
  fs
  ms
  T n(ps') {
    try {
      exp.m(ps)
    } finally {
      body
    }
  }
}
aspect A {
  pcs
  as
}

```

```

ts
class C {
  fs
  ms
  T n(ps') {
    exp.m(ps)
  }
}
=
aspect A {
  pcs
  after(context) :
    withincode( $\sigma(C.n())$ ) &&
    call( $\sigma(O.m())$ ) &&
    bind(context) {
      body[cthis/this]
    }
  as
}

```

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

<pre> <i>ts</i> class C { <i>fs</i> <i>ms</i> T m(<i>ps</i>) { <i>body</i>; <i>body'</i> } } aspect A { <i>pcs</i> <i>as</i> } </pre>	=	<pre> <i>ts</i> class C { <i>fs</i> <i>ms</i> T m(<i>ps</i>) { <i>body</i> } } aspect A { <i>pcs</i> after(<i>context</i>) returning(T t): execution($\sigma(C.m)$) && bind(<i>context</i>) { <i>body'</i>[<i>this</i> → <i>cthis</i>] } <i>as</i> } </pre>
---	---	--

provided

- (\rightarrow) *body'* does not declare or use local variables; *body'* does not call **super**;
- (\leftarrow) *body'* does not call **return**;

Law 8. Add After-Call Returning Successfully

<pre> <i>ts</i> class C { <i>fs</i> <i>ms</i> T n(<i>ps'</i>) { <i>exp.m(ps)</i>; <i>body'</i> } } aspect A { <i>pcs</i> <i>as</i> } </pre>	=	<pre> <i>ts</i> class C { <i>fs</i> <i>ms</i> T n(<i>ps'</i>) { <i>exp.m(ps)</i> } } aspect A { <i>pcs</i> after(<i>context</i>) returning(<i>T' t</i>): withincode($\sigma(C.n())$) && call($\sigma(O.m())$) && bind(<i>context</i>) { <i>body'[this → cthis]</i> } <i>as</i> } </pre>
---	---	---

provided

- (\rightarrow) *body'* does not declare or use local variables; *body'* does not call **super**;
 T' is the return type of method m
- (\leftarrow) *body'* does not call **return**;
- (\leftrightarrow) O is the type of *exp*

Law 9. Add After-Execution Throwing Exception

<pre> <i>ts</i> class C { <i>fs</i> <i>ms</i> T m(<i>ps</i>) throws <i>es</i> { try { <i>body</i> } catch(E e) { <i>body'</i> throw e } } } aspect A { <i>pcs</i> <i>as</i> } </pre>	=	<pre> <i>ts</i> class C { <i>fs</i> <i>ms</i> T m(<i>ps</i>) throws <i>es</i> { <i>body</i> } } aspect A { <i>pcs</i> after(<i>context</i>) throwing(E e): execution($\sigma(C.m)$) && bind(<i>context</i>) { <i>body'</i>[<i>this</i> → <i>cthis</i>] } <i>as</i> } </pre>
--	---	--

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

<pre> <i>ts</i> class C { <i>fs</i> <i>ms</i> T n(<i>ps'</i>) throws <i>es</i> { try { <i>exp.m(ps)</i> } catch(<i>E e</i>) { <i>body'</i> throw <i>e</i> } } } aspect A { <i>pcs</i> <i>as</i> } </pre>	=	<pre> <i>ts</i> class C { <i>fs</i> <i>ms</i> T n(<i>ps'</i>) throws <i>es</i> { <i>exp.m(ps)</i> } } aspect A { <i>pcs</i> after(<i>context</i>) throwing(<i>E e</i>): withincode($\sigma(C.n())$) && call($\sigma(O.m())$) && bind(<i>context</i>) { <i>body'[this → cthis]</i> } <i>as</i> } </pre>
--	---	--

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 11. Add Around-Execution

```

ts
class C {
  fs
  ms
  T m(ps) {
    body'
    if (cond) {
      body
    }
    body''
  }
}
aspect A {
  pcs
  as
}

```

```

ts
class C {
  fs
  ms
  T m(ps) {
    body
  }
}
aspect A {
  pcs
  T around (context) :
    execution( $\sigma(C.m)$ ) &&
    bind(context) {
      body'[this → cthis]
      if (cond) {
        proceed( $\alpha$ context)
      }
      body''[this → cthis]
    }
  as
}

```

provided

- (\rightarrow) *body'*, *body''* and *cond* do not declare or use local variables; and do not call **super**;
- (\leftarrow) *body'* does not call **return**;

Law 12. Add Around-Call

```

ts
class C {
  fs
  ms
  T n(ps') {
    body'
    if (cond) {
      exp.m(ps)
    }
    body''
  }
}
aspect A {
  pcs
  as
}

```

=

```

ts
class C {
  fs
  ms
  T n(ps') {
    exp.m(ps)
  }
}
aspect A {
  pcs
  T' around (context) :
    withincode( $\sigma(C.n())$ ) &&
    call( $\sigma(O.m())$ ) &&
    bind(context) {
      body'[this → cthis]
      if (cond) {
        proceed( $\alpha$ context)
      }
      body''[this → cthis]
    }
  as
}

```

provided

- (\rightarrow) *body'*, *body''* and *cond* do not declare or use local variables; and do not call **super**;
- (\leftarrow) *body'* does not call **return**;
- (\leftrightarrow) *O* is the type of *exp*

Law 14. Remove this Parameter

<pre> <i>ts</i> aspect <i>A</i> { <i>pcs</i> before(<i>T t</i>, <i>ps</i>) : <i>this(t) && exp</i> { <i>body</i> } <i>as</i> } </pre>	=	<pre> <i>ts</i> aspect <i>A</i> { <i>pcs</i> before(<i>ps</i>) : <i>this(T) && exp</i> { <i>body</i> } <i>as</i> } </pre>
---	---	---

provided

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

Law 16. Remove Argument Parameter

<pre> <i>ts</i> aspect A { <i>pcs</i> before(<i>P</i>₁ <i>p</i>₁, ..., <i>P</i>_{<i>i</i>} <i>p</i>_{<i>i</i>}, ..., <i>P</i>_{<i>n</i>} <i>p</i>_{<i>n</i>}, <i>ps</i>) : args(<i>p</i>₁, ..., <i>p</i>_{<i>i</i>}, ..., <i>p</i>_{<i>n</i>}) && <i>exp</i> { <i>body</i> } <i>as</i> } </pre>	=	<pre> <i>ts</i> aspect A { <i>pcs</i> before(<i>P</i>₁ <i>p</i>₁, ..., <i>P</i>_{<i>n</i>} <i>p</i>_{<i>n</i>}, <i>ps</i>) : args(<i>p</i>₁, ..., <i>P</i>_{<i>i</i>}, ..., <i>p</i>_{<i>n</i>}) && <i>exp</i> { <i>body</i> } <i>as</i> } </pre>
--	---	--

provided

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

Law 17. Add Catch for Softened Exception

```

ts
class C {
  fs
  ms
  T m(ps) throws es {
    try {
      body
    } catch(E e) {
      body'
    } cts
  }
}

```

=

```

ts
class C {
  fs
  ms
  T m(ps) throws es {
    try {
      body
      if (false) throw new E()
    } catch(E e) {
      body'
    } catch(SoftException se) {
      if(se.getWT() instanceof E) {
        E e = (E)se.getWT();
        body'
      } else {
        throw se
      }
    } cts
  }
}

```

provided

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

Law 19. Remove Exception from Throws Clause

<pre> <i>ts</i> class C { <i>fs</i> <i>ms</i> T m(<i>ps</i>) throws E, <i>es</i> { <i>body</i> } } </pre>	=	<pre> <i>ts</i> class C { <i>fs</i> <i>ms</i> T m(<i>ps</i>) throws <i>es</i> { <i>body</i> } } </pre>
---	---	--

provided

- (\leftrightarrow) *body* does not throw *E*
- (\leftarrow) 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

<pre> <i>ts</i> class C { <i>fs</i> <i>ms</i> T m(<i>ps</i>) throws <i>es</i> { try { <i>body</i> if (false) throw new E() } catch(E e) { <i>body'</i> } catch(SoftException se) { if(se.getWT() instanceof E) { E e = (E)se.getWT(); <i>body'</i> } else { throw se } } <i>cts</i> } } </pre>	=	<pre> <i>ts</i> class C { <i>fs</i> <i>ms</i> T m(<i>ps</i>) throws <i>es</i> { try { <i>body</i> } catch(SoftException se) { if(se.getWT() instanceof E) { E e = (E)se.getWT(); <i>body'</i> } else { throw se } } <i>cts</i> } } </pre>
--	---	---

provided

- (\leftrightarrow) *body* does not throw *E*

Law 21. Move Exception Handling to Aspect

```

ts
class C {
  fs
  ms
  T m(ps) throws es {
    try {
      body
    } catch(SoftException se) {
      body'
    } cts
  }
}
aspect A {
  pcs
  as
}

```

=

```

ts
class C {
  fs
  ms
  T m(ps) throws es {
    try {
      body
    } cts
  }
}
aspect A {
  pcs
  T around (context) throws es :
    execution( $\sigma(C.m)$ ) &&
    bind(context) {
    try {
      proceed( $\alpha$ context)
    } catch(SoftException se){
      body'
    }
  }
  as
}

```

provided

- (\rightarrow) *body'* does not declare or use local variables; *body'* does not call **super**;
- (\leftarrow) *body'* does not call **return**;

Law 24. Move Implements Declaration to Aspect

<pre> <i>ts</i> class <i>C</i> impl <i>D</i> { <i>fs</i> <i>ms</i> } aspect <i>A</i> { <i>pcs</i> <i>as</i> } </pre>	=	<pre> <i>ts</i> class <i>C</i> { <i>fs</i> <i>ms</i> } aspect <i>A</i> { declare parents : <i>C</i> impl <i>D</i> <i>pcs</i> <i>as</i> } </pre>
--	---	---

Law 25. Move Extends Declaration to Aspect

<pre> <i>ts</i> class <i>C</i> ext <i>D</i> { <i>fs</i> <i>ms</i> } aspect <i>A</i> { <i>pcs</i> <i>as</i> } </pre>	=	<pre> <i>ts</i> class <i>C</i> { <i>fs</i> <i>ms</i> } aspect <i>A</i> { declare parents : <i>C</i> ext <i>D</i> <i>pcs</i> <i>as</i> } </pre>
---	---	--

Law 26. Extract Named Pointcut

<pre> <i>ts</i> aspect <i>A</i> { <i>pcs</i> <i>as</i> advice(<i>ps</i>) : <i>exp</i>(α<i>ps</i>) {...} } </pre>	=	<pre> <i>ts</i> aspect <i>A</i> { <i>pcs</i> pointcut <i>p</i>(<i>ps</i>) : <i>exp</i>(α<i>ps</i>) <i>as</i> advice(<i>ps</i>) : <i>p</i>(α<i>ps</i>) {...} } </pre>
--	---	--

provided

(\rightarrow) There is no pointcut named p in pcs

(\leftarrow) There is no advice in as that references p

Law 27. Use Named Pointcut

<pre> <i>ts</i> aspect A { <i>pcs</i> pointcut <i>p(ps)</i>: <i>exp(αps)</i> <i>as</i> advice(<i>ps</i>): <i>exp(αps)</i> {...} } </pre>	=	<pre> <i>ts</i> aspect A { <i>pcs</i> pointcut <i>p(ps)</i>: <i>exp(αps)</i> <i>as</i> advice(<i>ps</i>): <i>p(αps)</i> {...} } </pre>
--	---	--

Law 28. Move Field Up to Interface

<pre> <i>ts</i> interface D {...} class C impl D {...} aspect A { <i>pcs</i> T C.field <i>as</i> } </pre>	=	<pre> <i>ts</i> interface D {...} class C impl D {...} aspect A { <i>pcs</i> T D.field <i>as</i> } </pre>
---	---	---

provided

- (\rightarrow) *A* does not already introduce an attribute named *field* to interface *D*
- (\leftarrow) *A* does not introduce any method to interface *D* that references *field*

Law 29. Move Method Up to Interface

<pre> <i>ts</i> interface D {...} class C impl D {...} aspect A { <i>pcs</i> T C.m(<i>ps</i>) { <i>body</i> } <i>as</i> } </pre>	=	<pre> <i>ts</i> interface D {...} class C impl D {...} aspect A { <i>pcs</i> T D.m(<i>ps</i>) { <i>body</i> } <i>as</i> } </pre>
--	---	--

provided

- (\rightarrow) Neither *A* or any other aspect in *ts* introduce a method named *m* to interface *D*
- (\leftarrow) Method *m* is not referenced in any implementation of interface *D* other than *C*

Law 30. Remove Method Implementation

```

ts
interface D {
  ms
  T m(ps)
}
class C impl D {
  fs
  ms
  T m(ps) {
    body
  }
}
aspect A {
  pcs
  T D.m(ps) {
    body
  }
}
as
}

```

=

```

ts
interface D {
  ms
  T m(ps)
}
class C impl D {
  fs
  ms
}
aspect A {
  pcs
  T D.m(ps) {
    body
  }
}
as
}

```