Contract based Development

Design by Contract, Static Analysis

Anders Kalhauge

Fall 2017

Design by Contract

- First described in 1986 by Bertrand Meyer
- Closely connected to the programming language Eiffel
- Formal specification
- Formal verification
- Hoare logic

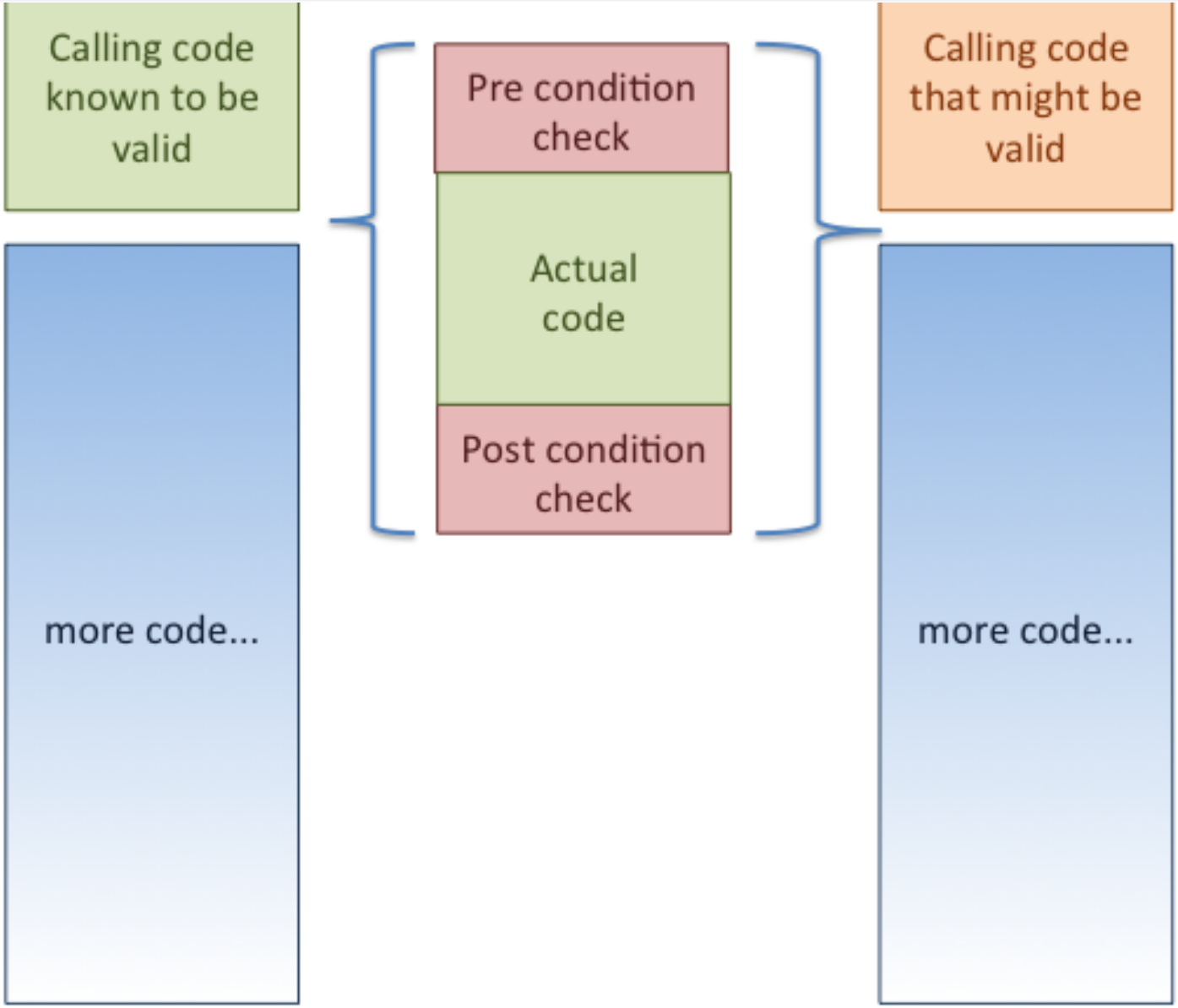
- What does contract expect?
- What does contract guarantee?
- What does contract maintain?

- Acceptable and unacceptable input values or types, and their meanings
- Return values or types, and their meanings
- Error and exception condition values or types that can occur, and their meanings
- Side effects
  - . . . not only collateral damage
- Preconditions
- Postconditions
- Invariants
- (more rarely) Performance guarantees, e.g. for time or space

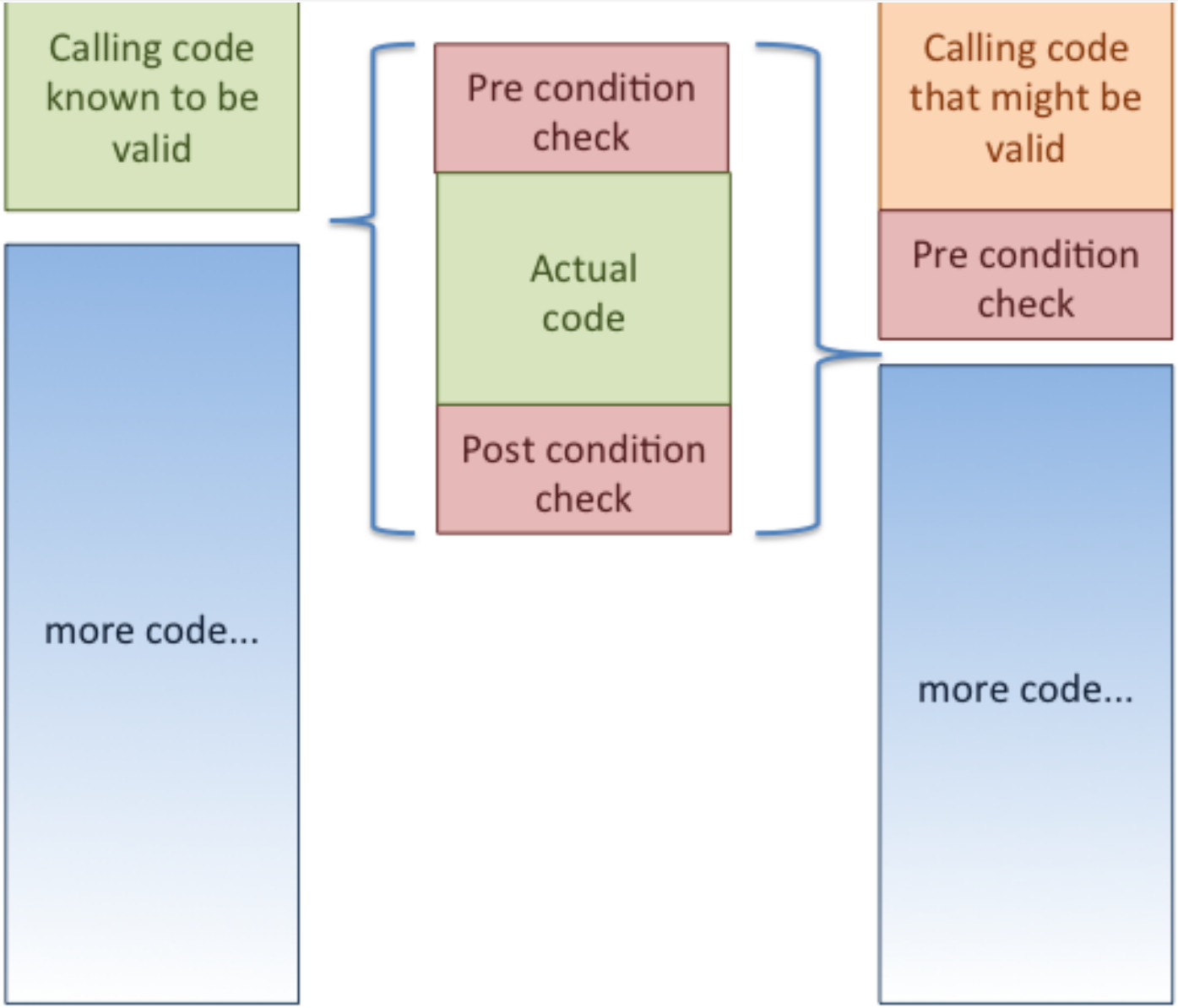
- Runtime contract checking
  - Classical defensive programming
- Static contract checking
  - If possible remove checks at runtime
  - If not possible include runtime checks
  - Might be turned off in production code - **why?**
  - It can be matematically proved that not everythin can be matematically proved ☺

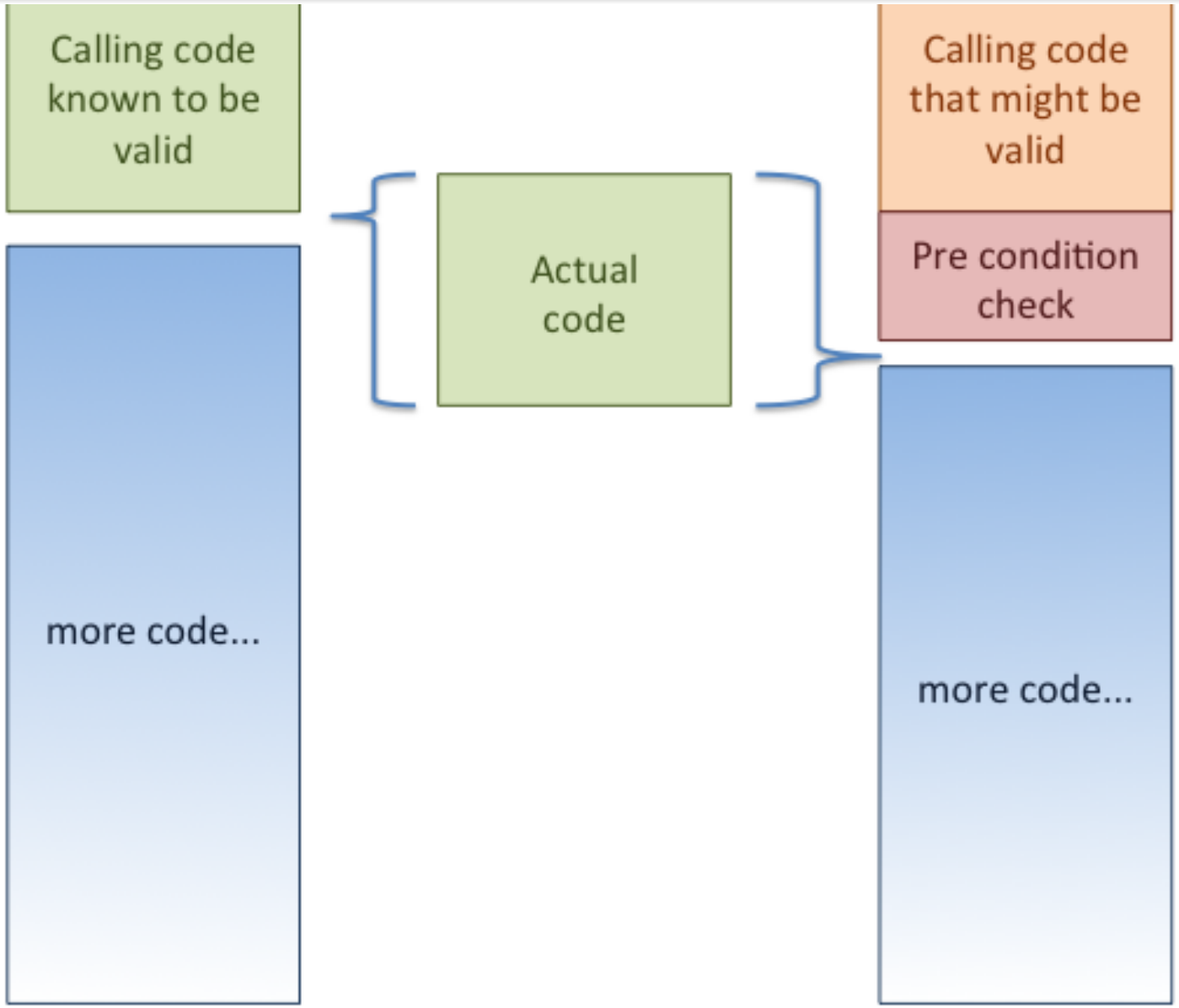
- Eiffel is the language used to define the concept by Bertrand Meyer
- Spec# is the proof of concept project from Microsoft
  - Is very complex in use.
  - Is still in experimental phase.
  - Works only for C#
- Code Contracts
  - Derived from Spec#
  - Works with all .NET languages
  - Implements the most important features.
- Various implementations for Java, primarily implemented with annotations.
- Ada, . . .

Defensive programming
Head under arm approach



Defensive programming
Solid approach




Calling code known to be valid
more code...
Actual code
Calling code that might be valid
Pre condition check
more code...

```
private /*@ spec_public @*/ int x;

/*@    requires a > 0;
  @    requires b != 0;
  @    ensures x == a/b;
  @    ensures \result = a/b;
  @ also
  @    requires a == b;
  @    signals_only StudidCallException;
  @*/
public int divide(int a, int b) {
  if (a == b) throw new StupidCallException();
  x = a/b;
  return x;
  }
```

```
int x;

public int divide(int a, int b) {
  Contract.Requires( a > 0 );
  Contract.Requires<DivideByZeroException>( b != 0 );
  Contract.Ensures(
      Contract.Result<int>()*b == a
      );
  Contract.EnsuresOnThrow<StupidCallException>(
      Contract.OldValue<int>( x ) == x
      );
  if (a == b) throw new StupidCallException();
  x = a/b;
  return x;
  }
```

```
public int add(int[] p, int[] q) {
  if (p == null) throw new ArgumentNullException("p");
  if (q == null) throw new ArgumentNullException("q");
  Contract.Requires( p.Lenght == q.Length );
  Contract.Ensures(
      Contract.ForAll(
          0,
          Contract.Result<int[]>().Length,
          i => Contract.Result<int[]>()[i] ==
              Contract.OldValue<int[]>(p)[i] +
              Contract.OldValue<int[]>(q)[i]
          )
      );
  for (int index = 0; index < p.Length; index++) {
    q[index] = p[index] + q[index];
    }
  return q;
  }
```

```
[ContractInvariantMethod]
private void ObjectInvariant() {
  Contract.Invariant( this.y >= 0 );
  Contract.Invariant( this.x > this.y );
  ...
  }
```

cphbusiness
COPENHAGEN BUSINESS ACADEMY

$$n \geq 3 \Rightarrow \forall a, b, c \in \mathbb{N} : a^n + b^n \neq c^n$$

```
bool Fermat(int n) {
  Contract.Requires(n >= 3);
  Contract.Ensures(Contract.Result<bool>() == true);
  int limit = 1000;
  for (int a = 1; a < limit; a++) {
    for (int b = 1; b < limit; b++) {
      for (int c = 1; c < limit; c++) {
        if (pow(a, n) + pow(b, n) == pow(c, n))
            return false;
        }
      }
    }
  return true;
  }
```

Will this program halt for every $x > 0$?

```
int program(int x) {
  while (x > 1) {
    if (odd(x)) x = 3*x + 1;
    else x = x/2;
    }
  return x;
  }
```

```
Contract.Assert( this.privateField > 0 );
Contract.Assert(
    this.x == 3,
    "Why␣isn't␣the␣value␣of␣x␣3?"
    );
...
Contract.Assume( this.privateField > 0 );
Contract.Assume(
    this.x == 3,
    "Static␣checker␣assumed␣this"
    );
```

cphbusiness
COPENHAGEN BUSINESS ACADEMY

$$\{P\}C\{Q\}$$

- P: precondition
- C: command
- Q: postcondition

cphbusiness
COPENHAGEN BUSINESS ACADEMY

$$\{P\}skip\{P\}$$

cphbusiness
COPENHAGEN BUSINESS ACADEMY

$$\frac{\{P\}S\{Q\}, \{Q\}T\{R\}}{\{P\}S; T\{R\}}$$

$$\frac{\{B \wedge P\}S\{Q\}, \{\neg B \wedge P\}T\{Q\}}{\{P\}\mathit{if}\ \ B\ \ \mathit{then}\ S\ \ \mathit{else}\ \ T\ \ \mathit{endif}\{Q\}}$$

cphbusiness
COPENHAGEN BUSINESS ACADEMY

$$\frac{\{B \wedge P\}S\{P\}}{\{P\}\mathit{while}\ B\ \mathit{do}\ S\ \mathit{done}\{\neg B \wedge P\}}$$

- Define integer variable X

```
DEF X: INTEGER
```

- Define boolean variable P

```
DEF P: BOOLEAN
```

- Instantiate variable X and P

```
LET X = 100
LET P = TRUE
```

- Set variable X to result

```
LET X = Y + 20
```

- Set variable P to result

```
LET P = X > 20
```

VSSL operators

VSSL only supports one operator per statement.

- Integer operators
  `A + B`, `A - B`, `-A`
- Boolean operators
  `AND`, `OR`, `NOT`
- Comparators
  `<`, `<=`, `==`, `>=`, `>`

Statements end with newline.

- Block (Sequence)
  `{ statements }`
- Selection
  `IF (predicate) block`
  or
  `IF (predicate) block`
  `ELSE block`
- Iteration
  `WHILE (predicate) block`

Preconditions: Y is defined and known

$$P = [Y \in \{\dots\}]$$

```
DEF X: Integer
IF (Y < 10) {
  LET X = 100
  }
LET Y = Y + 10
IF (Y >= 20) {
  LET X = 4711
  }
```

Postcondition: X is known

$$Q = [X \in \{\dots\}]$$

VSSL example program

```
DEF X: Integer
```

$$S = [Y \in \{\ldots\}, X \in \{?\}]$$

```
IF (Y < 10) {
  LET X = 100
  }
LET Y = Y + 10
IF (Y >= 20) {
  LET X = 4711
  }
```

VSSL example program

```
DEF X: Integer
IF (Y < 10) {
```

$$S = [Y \in \{\ldots 9\}, X \in \{?\}]$$

```
  LET X = 100
  }
LET Y = Y + 10
IF (Y >= 20) {
  LET X = 4711
  }
```

VSSL example program

```
DEF X: Integer
IF (Y < 10) {
  LET X = 100
```

$$S = [Y \in \{\ldots 9\}, X \in \{100\}]$$

```
  }
LET Y = Y + 10
IF (Y >= 20) {
  LET X = 4711
  }
```

VSSL example program

```
DEF X: Integer
IF (Y < 10) {
  LET X = 100
  }
```

$$S = [Y \in \{\ldots 9\}, X \in \{100\}] \vee [Y \in \{10\ldots\}, X \in \{?\}]$$

```
LET Y = Y + 10
IF (Y >= 20) {
  LET X = 4711
  }
```

VSSL example program

```
DEF X: Integer
IF (Y < 10) {
 LET X = 100
 }
LET Y = Y + 10
```

$$S = [Y \in \{\ldots 19\}, X \in \{100\}] \vee [Y \in \{20\ldots\}, X \in \{?\}]$$

```
IF (Y >= 20) {
 LET X = 4711
 }
```

VSSL example program

```
DEF X: Integer
IF (Y < 10) {
 LET X = 100
 }
LET Y = Y + 10
IF (Y >= 20) {
```

$$S = [Y \in \{20 \ldots\}, X \in \{?\}]$$

```
 LET X = 4711
 }
```

VSSL example program

```
DEF X: Integer
IF (Y < 10) {
 LET X = 100
 }
LET Y = Y + 10
IF (Y >= 20) {
 LET X = 4711
```

$$S = [Y \in \{20 \ldots\}, X \in \{4711\}]$$

```
 }
```

VSSL example program

```
DEF X: Integer
IF (Y < 10) {
 LET X = 100
 }
LET Y = Y + 10
IF (Y >= 20) {
 LET X = 4711
 }
```

$$S = [Y \in \{\ldots 19\}, X \in \{100\}] \vee [Y \in \{20\ldots\}, X \in \{4711\}]$$

Postcondition is:

$$Q = [X \in \{\dots\}] \text{ or } Q = [Y \in \mathbb{U}, X \in \{\dots\}]$$

Where $\mathbb{U}$ is the universal set ($\{?, -\infty \dots \infty\} here$).
Define S' as $[Y \in \{\dots\}, X \in \{100, 4711\}]$, then $S \preceq S'$.
Also $S' \preceq Q$ because:

$$Y_{S'} \subseteq Y_Q \wedge X_{S'} \subseteq X_Q$$

$$\{\dots\} \subseteq \mathbb{U} \wedge \{100, 4711\} \subseteq \{\dots\}$$

Therefore because of the property of transitivity for partly order:

$$S \preceq S' \preceq Q \implies S \preceq Q$$

Conclusion: Q covers all possible states S, analysis succeeded