White-box Testing by Combining Deduction-based Specification Extraction and Black-box Testing

Bernhard Beckert, Christoph Gladisch

www.key-project.org

6th KeY Symposium 2007

Nomborn, Germany
June 14, 2007

Two Kinds of Specifications

Requirement Specificaiton

- Given by the user
- Role: To be tested or verified

Full Specifictaion

- Must comply with the IUT (Impl. Under Test)
- Reflects the structure of the program
- Can be extracted automatically

Two Kinds of Specifications

Requirement Specificaiton

- Given by the user
- Role: To be tested or verified

Full Specifictaion

- Must comply with the IUT (Impl. Under Test)
- Reflects the structure of the program
- Can be extracted automatically

Two Kinds of Specifications

Requirement Specificaiton

- Given by the user
- Role: To be tested or verified

Full Specifictaion

- Must comply with the IUT (Impl. Under Test)
- Reflects the structure of the program
- Can be extracted automatically

Two Kinds of Specifications

Requirement Specificaiton

- Given by the user
- Role: To be tested or verified

Full Specifictaion

- Must comply with the IUT (Impl. Under Test)
- Reflects the structure of the program
- Can be extracted automatically

Two Kinds of Specifications

Requirement Specificaiton

- Given by the user
- Role: To be tested or verified

Full Specifictaion

- Must comply with the IUT (Impl. Under Test)
- Reflects the structure of the program
- Can be extracted automatically

Two Kinds of Specifications

Requirement Specificaiton

- Given by the user
- Role: To be tested or verified

Full Specifictaion

- Must comply with the IUT (Impl. Under Test)
- Reflects the structure of the program
- Can be extracted automatically

Two Kinds of Specifications

Requirement Specificaiton

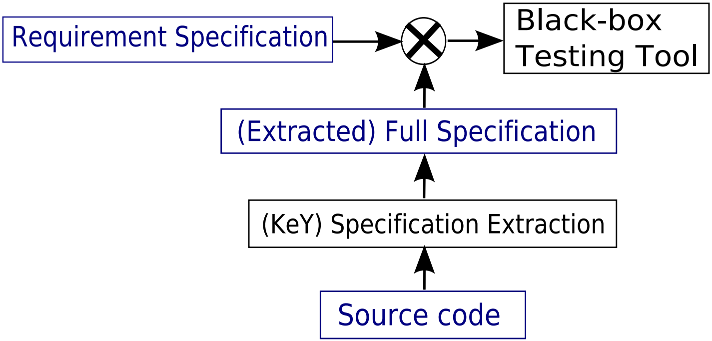
- Given by the user
- Role: To be tested or verified

Full Specifictaion

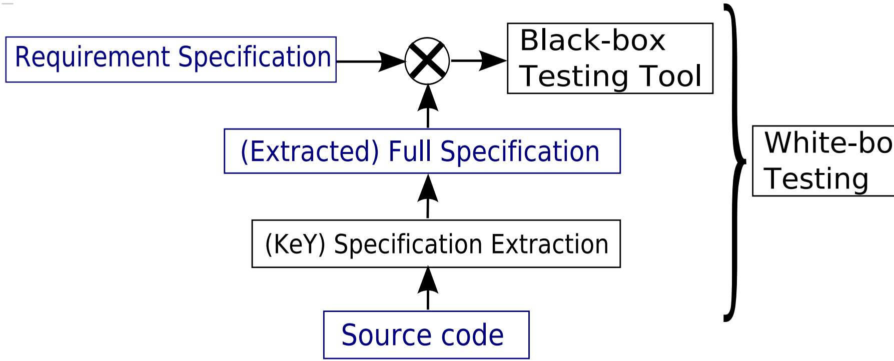
- Must comply with the IUT (Impl. Under Test)
- Reflects the structure of the program
- Can be extracted automatically

Tool Chain

Tool Chain


Requirement Specification
Black-box Testing Tool
(Extracted) Full Specification
(KeY) Specification Extraction
Source code

Tool Chain


Requirement Specification
Black-box
Testing Tool
(Extracted) Full Specification
(KeY) Specification Extraction
Source code
White-bo
Testing

Benefits

- Using of existing Black-box Testing Tools for White-box testing
- Separation of concerns - Modularity
- Combination of Coverage Criteria

Benefits

- Using of existing Black-box Testing Tools for White-box testing
- Separation of concerns - Modularity
- Combination of Coverage Criteria

Benefits

- Using of existing Black-box Testing Tools for White-box testing
- Separation of concerns - Modularity
- Combination of Coverage Criteria

Benefits

- Using of existing Black-box Testing Tools for White-box testing
- Separation of concerns - Modularity
- Combination of Coverage Criteria

The KeYSystem

Program Variable = non-rigid Function Symbol

$$(\textit{prog.var.})\ \texttt{a} \quad = \quad a\ \text{(logic const.)}$$
$$\texttt{o.a} \quad = \quad a(o)$$

Modal Operators

$$[p]\phi \qquad \langle p\rangle\phi \qquad \{a := b\}\phi$$
$$\langle \texttt{o.a = t;u.b = s}\rangle\phi \quad \rightsquigarrow \quad \{a(o) := t \,||\, b(u') := s'\}\phi$$
$$\{\textit{for}\ x; f_x := g_x\}\phi \quad \rightsquigarrow \quad \{f_n := g_n||..||f_0 := g_0\}\phi$$

Sequent Calculus Rules

$$\frac{\Gamma, c = \textit{true} \Longrightarrow \langle \texttt{p}\rangle\phi, \Delta \qquad \Gamma, c = \textit{false} \Longrightarrow \langle \texttt{q}\rangle\phi, \Delta}{\Gamma \Longrightarrow \langle \texttt{if(c)\{p\}else\{q\}..}\rangle\phi, \Delta}$$

The KeYSystem

Program Variable = non-rigid Function Symbol

$$(\mathit{prog.var.})\ \texttt{a} = a \text{ (logic const.)}$$
$$\texttt{o.a} = a(o)$$

Modal Operators

$$[p]\phi \qquad \langle p\rangle\phi \qquad \{a := b\}\phi$$
$$\langle \texttt{o.a = t;u.b = s}\rangle\phi \rightsquigarrow \{a(o) := t \,||\, b(u') := s'\}\phi$$
$$\{\mathit{for}\ x; f_x := g_x\}\phi \rightsquigarrow \{f_n := g_n||..||f_0 := g_0\}\phi$$

Sequent Calculus Rules

$$\frac{\Gamma, c = \mathit{true} \Longrightarrow \langle \texttt{p}\rangle\phi, \Delta \qquad \Gamma, c = \mathit{false} \Longrightarrow \langle \texttt{q}\rangle\phi, \Delta}{\Gamma \Longrightarrow \langle \texttt{if(c)\{p\}else\{q\}..}\rangle\phi, \Delta}$$

The KeYSystem

Program Variable = non-rigid Function Symbol

$$(\textit{prog.var.})\ \texttt{a} \quad = \quad a\ (\text{logic const.})$$
$$\texttt{o.a} \quad = \quad a(o)$$

Modal Operators

$$[p]\phi \qquad \langle p\rangle\phi \qquad \{a := b\}\phi$$
$$\langle \texttt{o.a = t;u.b = s}\rangle\phi \quad \rightsquigarrow \quad \{a(o) := t \,||\, b(u') := s'\}\phi$$
$$\{\textit{for } x;\ f_x := g_x\}\phi \quad \rightsquigarrow \quad \{f_n := g_n||..||f_0 := g_0\}\phi$$

Sequent Calculus Rules

$$\frac{\Gamma, c = \textit{true} \Longrightarrow \langle \texttt{p}\rangle\phi, \Delta \qquad \Gamma, c = \textit{false} \Longrightarrow \langle \texttt{q}\rangle\phi, \Delta}{\Gamma \Longrightarrow \langle \texttt{if(c)\{p\}else\{q\}..}\rangle\phi, \Delta}$$

The KeYSystem

Program Variable = non-rigid Function Symbol

$$(\textit{prog.var.})\ \texttt{a} = a\ (\text{logic const.})$$
$$\texttt{o.a} = a(o)$$

Modal Operators

$$[p]\phi \qquad \langle p\rangle\phi \qquad \{a := b\}\phi$$
$$\langle\texttt{o.a = t;u.b = s}\rangle\phi \rightsquigarrow \{a(o) := t \,||\, b(u') := s'\}\phi$$
$$\{for\ x; f_x := g_x\}\phi \rightsquigarrow \{f_n := g_n||..||f_0 := g_0\}\phi$$

Sequent Calculus Rules

$$\frac{\Gamma, c = true \Longrightarrow \langle\texttt{p}\rangle\phi, \Delta \qquad \Gamma, c = false \Longrightarrow \langle\texttt{q}\rangle\phi, \Delta}{\Gamma \Longrightarrow \langle\texttt{if(c)\{p\}else\{q\}..}\rangle\phi, \Delta}$$

The KeYSystem

Program Variable = non-rigid Function Symbol

$$(prog.var.)\ \texttt{a} = a \text{ (logic const.)}$$
$$\texttt{o.a} = a(o)$$

Modal Operators

$$[p]\phi \qquad \langle p\rangle\phi \qquad \{a := b\}\phi$$
$$\langle \texttt{o.a = t;u.b = s}\rangle\phi \rightsquigarrow \{a(o) := t \,||\, b(u') := s'\}\phi$$
$$\{for\ x; f_x := g_x\}\phi \rightsquigarrow \{f_n := g_n||..||f_0 := g_0\}\phi$$

Sequent Calculus Rules

$$\frac{\Gamma, c = true \Longrightarrow \langle \texttt{p}\rangle\phi, \Delta \qquad \Gamma, c = false \Longrightarrow \langle \texttt{q}\rangle\phi, \Delta}{\Gamma \Longrightarrow \langle \texttt{if(c)\{p\}else\{q\}..}\rangle\phi, \Delta}$$

The KeYSystem

Program Variable = non-rigid Function Symbol

$$(prog.var.)\ \texttt{a} = a \text{ (logic const.)}$$

$$\texttt{o.a} = a(o)$$

Modal Operators

$$[p]\phi \qquad \langle p\rangle\phi \qquad \{a := b\}\phi$$

$$\langle \texttt{o.a = t;u.b = s}\rangle\phi \rightsquigarrow \{a(o) := t \,||\, b(u') := s'\}\phi$$

$$\{for\ x; f_x := g_x\}\phi \rightsquigarrow \{f_n := g_n||..||f_0 := g_0\}\phi$$

Sequent Calculus Rules

$$\frac{\Gamma, c = true \Longrightarrow \langle \texttt{p}\rangle\phi, \Delta \qquad \Gamma, c = false \Longrightarrow \langle \texttt{q}\rangle\phi, \Delta}{\Gamma \Longrightarrow \langle \texttt{if(c)\{p\}else\{q\}..}\rangle\phi, \Delta}$$

The KeYSystem

Program Variable = non-rigid Function Symbol

$$(prog.var.)\ \texttt{a} = a \text{ (logic const.)}$$
$$\texttt{o.a} = a(o)$$

Modal Operators

$$[p]\phi \qquad \langle p\rangle\phi \qquad \{a := b\}\phi$$
$$\langle \texttt{o.a = t;u.b = s}\rangle\phi \rightsquigarrow \{a(o) := t \,||\, b(u') := s'\}\phi$$
$$\{for\ x; f_x := g_x\}\phi \rightsquigarrow \{f_n := g_n||..||f_0 := g_0\}\phi$$

Sequent Calculus Rules

$$\frac{\Gamma, c = true \Longrightarrow \langle \texttt{p}\rangle\phi, \Delta \qquad \Gamma, c = false \Longrightarrow \langle \texttt{q}\rangle\phi, \Delta}{\Gamma \Longrightarrow \langle \texttt{if(c)\{p\}else\{q\}..}\rangle\phi, \Delta}$$

The KeYSystem

Program Variable = non-rigid Function Symbol

$$(\mathit{prog.var.})\ \mathtt{a} \quad = \quad a\ (\text{logic const.})$$
$$\mathtt{o.a} \quad = \quad a(o)$$

Modal Operators

$$[p]\phi \qquad \langle p\rangle\phi \qquad \{a := b\}\phi$$
$$\langle \mathtt{o.a\ =\ t;u.b\ =\ s}\rangle\phi \quad \rightsquigarrow \quad \{a(o) := t \,||\, b(u') := s'\}\phi$$
$$\{\mathit{for}\ x; f_x := g_x\}\phi \quad \rightsquigarrow \quad \{f_n := g_n||..||f_0 := g_0\}\phi$$

Sequent Calculus Rules

$$\frac{\Gamma, c = \mathit{true} \Longrightarrow \langle \mathtt{p}\rangle\phi, \Delta \qquad \Gamma, c = \mathit{false} \Longrightarrow \langle \mathtt{q}\rangle\phi, \Delta}{\Gamma \Longrightarrow \langle \mathtt{if(c)\{p\}else\{q\}..}\rangle\phi, \Delta}$$

Example IUT

```
public class AbsDiff{
    public static int d;
     /*@ public normal_behavior
       @  requires true;
       @  ensures d==x-y || d==y-x;
       @  ensures d>=x-y && d>=y-x;
       @*/
     public static void diff(int x, int y){
         if(x<y) d=y;
          else   d=x;
         if(d<=y)d=d-x;
          else   d=d-y;
     }
}
```

Specification Extraction (Structural properties)

$$\dfrac{\dfrac{\dfrac{\texttt{x} < \texttt{y}, \texttt{x} \leq \texttt{y} \Longrightarrow \{\texttt{d} := \texttt{y-x}\}\Delta \qquad \dfrac{*}{\texttt{x} < \texttt{y}, \texttt{x} > \texttt{y} \Longrightarrow \{\texttt{d} := \texttt{y-x}\}\Delta}}{\ldots}}{\texttt{x} < \texttt{y} \Longrightarrow \{\texttt{d} := \texttt{y}\}[\texttt{if} \ \ldots]\Delta} \qquad \dfrac{(B_3) \qquad (B_4)}{\texttt{x} \geq \texttt{y} \Longrightarrow \{\texttt{d} := \texttt{x}\}[\texttt{if} \ \ldots]\Delta}}{\Longrightarrow [\texttt{if(x<y)d=y;else d=x; if(d<=y)}\ldots]\Delta}$$

Specification Extraction (Structural properties)

$$
\dfrac{
\dfrac{
\dfrac{
\begin{array}{c} \mathtt{x} < \mathtt{y}, \mathtt{x} \leq \mathtt{y} \\ \Longrightarrow \{\mathtt{d} := \mathtt{y\text{-}x}\}\Delta \end{array}
\qquad
\dfrac{*}{\begin{array}{c} \mathtt{x} < \mathtt{y}, \mathtt{x} > \mathtt{y} \\ \Longrightarrow \{\mathtt{d} := \mathtt{y\text{-}x}\}\Delta \end{array}}
}{\ldots}
}{\mathtt{x} < \mathtt{y} \Longrightarrow \{\mathtt{d} := \mathtt{y}\}[\mathtt{if}\ \ldots]\Delta}
\qquad
\dfrac{(B_3) \qquad (B_4)}{\begin{array}{c} \mathtt{x} \geq \mathtt{y} \Longrightarrow \\ \{\mathtt{d} := \mathtt{x}\}[\mathtt{if}\ \ldots]\Delta \end{array}}
}{\Longrightarrow [\mathtt{if(x{<}y)d{=}y;else\ d{=}x;\ if(d{<=}y)}\ldots]\Delta}
$$

Specification Extraction (Structural properties)

$$\dfrac{\dfrac{\dfrac{\mathtt{x} < \mathtt{y}, \mathtt{y} \le \mathtt{y} \Longrightarrow \{\mathtt{d} := \mathtt{y-x}\}\Delta \qquad \dfrac{*}{\mathtt{x} < \mathtt{y}, \mathtt{x} > \mathtt{y} \Longrightarrow \{\mathtt{d} := \mathtt{y-x}\}\Delta}}{\dots}}{\mathtt{x} < \mathtt{y} \Longrightarrow \{\mathtt{d} := \mathtt{y}\}[\mathtt{if}\ \dots]\Delta} \qquad \dfrac{(B_3) \qquad (B_4)}{\mathtt{x} \ge \mathtt{y} \Longrightarrow \{\mathtt{d} := \mathtt{x}\}[\mathtt{if}\ \dots]\Delta}}{\Longrightarrow [\mathtt{if(x<y)d=y;else\ d=x;\ \ if(d<=y)}\dots]\Delta}$$

```
B1: req x<y && y<=y ens d=\old(y-x); also
B3: req x>=y && x<=y ens d=\old(x-y); also
B4: req x>=y && x>y ens d=\old(x-y);
```

Specification Extraction (Structural properties)

$$
\dfrac{
\dfrac{
\dfrac{
\begin{array}{c} \mathtt{x} < \mathtt{y}, \mathtt{y} \leq \mathtt{y} \\ \Longrightarrow \{\mathtt{d} := \mathtt{y\text{-}x}\}\Delta \end{array}
\qquad
\dfrac{*}{\begin{array}{c} \mathtt{x} < \mathtt{y}, \mathtt{x} > \mathtt{y} \\ \Longrightarrow \{\mathtt{d} := \mathtt{y\text{-}x}\}\Delta \end{array}}
}{\ldots}
}{\mathtt{x} < \mathtt{y} \Longrightarrow \{\mathtt{d} := \mathtt{y}\}[\mathtt{if}\ \ldots]\Delta}
\qquad
\dfrac{(B_3) \qquad (B_4)}{\begin{array}{c} \mathtt{x} \geq \mathtt{y} \Longrightarrow \\ \{\mathtt{d} := \mathtt{x}\}[\mathtt{if}\ \ldots]\Delta \end{array}}
}{\Longrightarrow [\mathtt{if(x<y)d=y;else\ d=x;\ if(d<=y)}\ldots]\Delta}
$$

```
B1: req x<y && y<=y ens d=\old(y-x); also
B3: req x>=y && x<=y ens d=\old(x-y); also
B4: req x>=y && x>y ens d=\old(x-y);
```

Example IUT

```
public class AbsDiff{
   public static int d;

    /*@ public normal_behavior
      &  requires true;
      @  ensures d==x-y || d==y-x;
      @  ensures d>=x-y && d>=y-x;
      @*/

     public static void diff(int x, int y){
...
     }
}
```

Example IUT

```
/*@ public normal_behavior
  @  requires true;
  @  ensures d==x-y || d==y-x;
  @  ensures d>=x-y && d>=y-x;
  @ also
  @  requires y < x;
  @  ensures d == \old(x - y);
  @ also
  @  requires y == x;
  @  ensures d == \old(0);
  @ also
  @  requires y > x;
  @  ensures d == \old(y - x);
  @*/
```

Example IUT

```
/*@ public normal_behavior
  @  requires y < x && true;
  @  ensures d == \old(x - y)
  @   && (d==x-y || d==y-x) && d>=x-y && d>=y-x;
  @ also
  @  requires y == x && true;
  @  ensures d == \old(0)
  @   && (d==x-y || d==y-x) && d>=x-y && d>=y-x;
  @ also
  @  requires y > x && true;
  @  ensures d == \old(y - x)
  @   && (d==x-y || d==y-x) && d>=x-y && d>=y-x;
  @*/
```

Using the extracted Post Condition

- Requirement Specification

```
requires true;
ensures  (d==x-y || d==y-x) && d>=x-y && d>=y-x
         && d!=MAX_INT;
```

- With Full Specification

```
requires true && y < x;
ensures  (d==x-y || d==y-x) && d>=x-y && d>=y-x
         && d!=MAX_INT &&  d == \old(x - y);
also
    ...
```

Using the extracted Post Condition

- Requirement Specification

```
requires true;
ensures  (d==x-y || d==y-x) && d>=x-y && d>=y-x
         && d!=MAX_INT;
```

- With Full Specification

```
requires true && y < x;
ensures  (d==x-y || d==y-x) && d>=x-y && d>=y-x
         && d!=MAX_INT &&  d == \old(x - y);
also
    ...
```

Using the extracted Post Condition

- Requirement Specification

```
requires true;
ensures  (d==x-y || d==y-x) && d>=x-y && d>=y-x
         && d!=MAX_INT;
```

- With Full Specification

```
requires true && y < x;
ensures  (d==x-y || d==y-x) && d>=x-y && d>=y-x
         && d!=MAX_INT &&  d == \old(x - y);
also
    ...
```

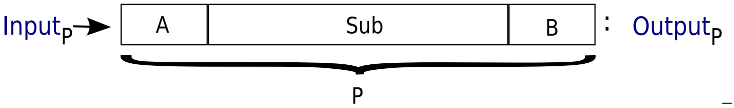
Loops

```
while (k<n) {
  if (j=7) {
    j         = 0;
    line      = new Line(line);
  }
  line.buf[j]=a[k];
  k++; j++;
}
```

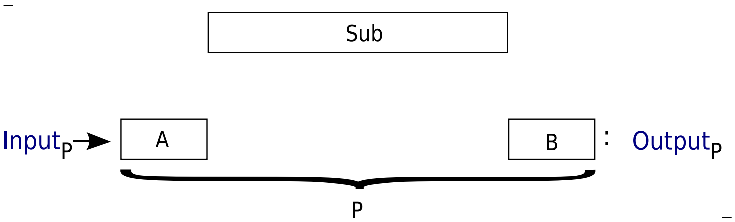
Loops (Unfolding)

```
if (k<n)
  {if(j=7){..};if(j>7||k>n)..;
   line.buf[j]=a[k]; k++; j++;
   if(k<n)
     {if(j=7){..};if(j>7||k>n)..;
      line.buf[j]=a[k]; k++; j++;
      ...
     while(k<n){...}
  }
}
```

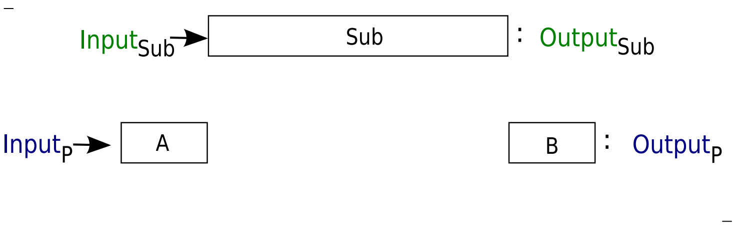
Contracts Program Replacements



Contracts Program Replacements

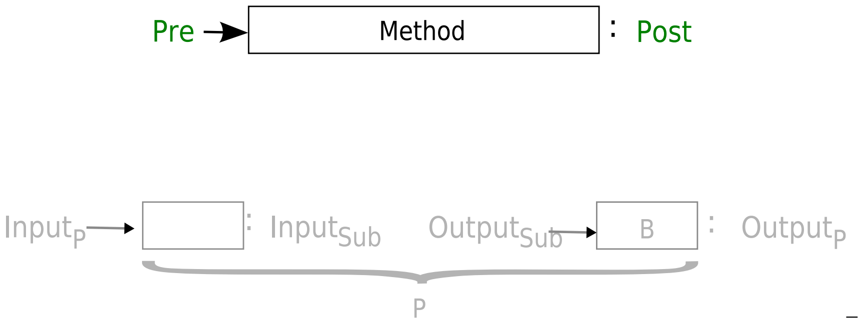


Contracts Program Replacements

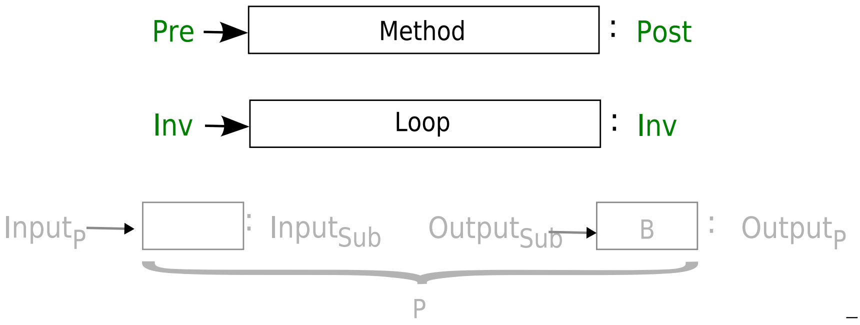


Contracts Program Replacements

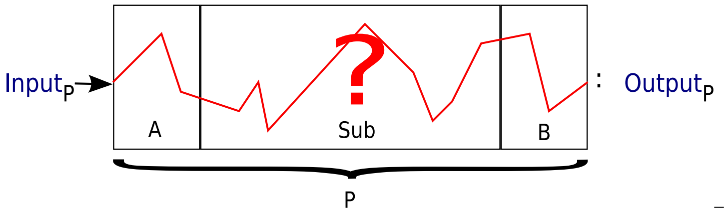
Contracts Program Replacements



Contracts Program Replacements



Contracts Program Replacements



Traditional Contract Rule

$$
\dfrac{\dfrac{Pre \Longrightarrow Pre_C, \langle \mathtt{p} \rangle Post \qquad \dfrac{Post_C \Longrightarrow Post}{\langle \mathtt{p} \rangle Post_C, Pre \Longrightarrow \langle \mathtt{p} \rangle Post}}{Pre_C \rightarrow \langle \mathtt{p} \rangle Post_C, Pre \Longrightarrow \langle \mathtt{p} \rangle Post}}{\underbrace{Pre_C \rightarrow \langle \mathtt{p} \rangle Post_C}_{\textit{Contract}} \Longrightarrow Pre \rightarrow \langle \mathtt{p} \rangle Post}
$$

Traditional Contract Rule

$$\dfrac{Pre \Longrightarrow Pre_C, \langle \mathtt{p} \rangle Post \qquad \dfrac{Post_C \Longrightarrow Post}{\langle \mathtt{p} \rangle Post_C, Pre \Longrightarrow \langle \mathtt{p} \rangle Post}}{\dfrac{Pre_C \to \langle \mathtt{p} \rangle Post_C, Pre \Longrightarrow \langle \mathtt{p} \rangle Post}{\underbrace{Pre_C \to \langle \mathtt{p} \rangle Post_C}_{Contract} \Longrightarrow Pre \to \langle \mathtt{p} \rangle Post}}$$

Traditional Contract Rule

$$\dfrac{\mathit{Pre} \Longrightarrow \mathit{Pre}_C, \langle \mathtt{p} \rangle \mathit{Post} \qquad \dfrac{\mathit{Post}_C \Longrightarrow \mathit{Post}}{\langle \mathtt{p} \rangle \mathit{Post}_C, \mathit{Pre} \Longrightarrow \langle \mathtt{p} \rangle \mathit{Post}}}{\dfrac{\mathit{Pre}_C \rightarrow \langle \mathtt{p} \rangle \mathit{Post}_C, \mathit{Pre} \Longrightarrow \langle \mathtt{p} \rangle \mathit{Post}}{\underbrace{\mathit{Pre}_C \rightarrow \langle \mathtt{p} \rangle \mathit{Post}_C}_{\mathit{Contract}} \Longrightarrow \mathit{Pre} \rightarrow \langle \mathtt{p} \rangle \mathit{Post}}}$$

Traditional Contract Rule

$$\dfrac{Pre \Longrightarrow Pre_C,\ \langle \mathtt{p} \rangle Post \qquad \dfrac{Post_C \Longrightarrow Post}{\langle \mathtt{p} \rangle Post_C,\ Pre \Longrightarrow \langle \mathtt{p} \rangle Post}}{\dfrac{Pre_C \rightarrow \langle \mathtt{p} \rangle Post_C,\ Pre \Longrightarrow \langle \mathtt{p} \rangle Post}{\underbrace{Pre_C \rightarrow \langle \mathtt{p} \rangle Post_C}_{Contract} \Longrightarrow Pre \rightarrow \langle \mathtt{p} \rangle Post}}$$

Traditional Contract Rule

$$\dfrac{\mathit{Pre} \Longrightarrow \mathit{Pre}_C,\ \langle \mathtt{p} \rangle \mathit{Post} \qquad \dfrac{\mathit{Post}_C \Longrightarrow \mathit{Post}}{\langle \mathtt{p} \rangle \mathit{Post}_C,\ \mathit{Pre} \Longrightarrow \langle \mathtt{p} \rangle \mathit{Post}}}{\dfrac{\mathit{Pre}_C \rightarrow \langle \mathtt{p} \rangle \mathit{Post}_C,\ \mathit{Pre} \Longrightarrow \langle \mathtt{p} \rangle \mathit{Post}}{\underbrace{\mathit{Pre}_C \rightarrow \langle \mathtt{p} \rangle \mathit{Post}_C}_{\mathit{Contract}} \Longrightarrow \mathit{Pre} \rightarrow \langle \mathtt{p} \rangle \mathit{Post}}}$$

Traditional Contract Rule

$$\dfrac{\dfrac{\dots \qquad \dfrac{Post_C \Longrightarrow Post}{\langle \mathtt{p} \rangle Post_C, Pre \Longrightarrow \langle \mathtt{p} \rangle Post}}{Pre_C \to \langle \mathtt{p} \rangle Post_C, Pre \Longrightarrow \langle \mathtt{p} \rangle Post}}{\underbrace{Pre_C \to \langle \mathtt{p} \rangle Post_C}_{Contract} \Longrightarrow Pre \to \langle \mathtt{p} \rangle Post}$$

KeY's Contract Rule

$$
\dfrac{\dots \qquad \dfrac{Pre, \{for\ \ x; f(x) := f^{sk}(x)\} Post_C \Longrightarrow \{for\ \ x; f(x) := f^{sk}(x)\} Post}{\langle \mathtt{p} \rangle Post_C, Pre \Longrightarrow \langle \mathtt{p} \rangle Post}}{\dfrac{Pre, Pre_C \rightarrow \langle \mathtt{p} \rangle Post_C \Longrightarrow \langle \mathtt{p} \rangle Post}{\underbrace{Pre_C \rightarrow \langle \mathtt{p} \rangle Post_C}_{Contract} \Longrightarrow Pre \rightarrow \langle \mathtt{p} \rangle Post}}
$$

where $\{forx.f(x) := f^{sk}(x)\}$ abbrev.

$\{for\ \ x_{0,1}.\dots.x_{0,n_0}.f_0(x_{0,1},\dots,x_{0,n_0}) := f_0^{sk}(x_{0,1},\dots,x_{0,n_0})\}$

$\vdots$

$\{for\ \ x_{m,1}.\dots.x_{m,n_m}.f_m(x_{m,1},\dots,x_{m,n_m}) := f_m^{sk}(x_{m,1},\dots,x_{m,n_m})\}$

KeY's Contract Rule

$$
\dfrac{\dots \qquad \dfrac{\begin{array}{c} Pre, \{for\ \ x; f(x) := f^{sk}(x)\} Post_C \\ \Longrightarrow \{for\ \ x; f(x) := f^{sk}(x)\} Post \end{array}}{\langle \mathtt{p} \rangle Post_C, Pre \Longrightarrow \langle \mathtt{p} \rangle Post}}{\dfrac{Pre, Pre_C \rightarrow \langle \mathtt{p} \rangle Post_C \Longrightarrow \langle \mathtt{p} \rangle Post}{\underbrace{Pre_C \rightarrow \langle \mathtt{p} \rangle Post_C}_{Contract} \Longrightarrow Pre \rightarrow \langle \mathtt{p} \rangle Post}}
$$

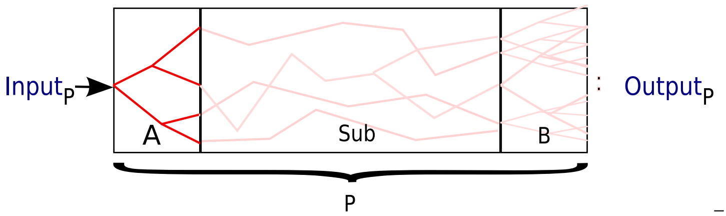
where $\{forx.f(x) := f^{sk}(x)\}$ abbrev.

$\{for\ \ x_{0,1}.\dots x_{0,n_0}.f_0(x_{0,1},\dots,x_{0,n_0}) := f_0^{sk}(x_{0,1},\dots,x_{0,n_0})\}$

$\vdots$

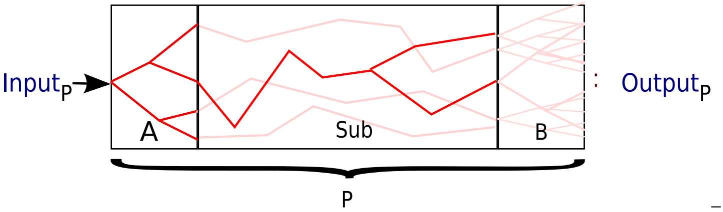
$\{for\ \ x_{m,1}.\dots x_{m,n_m}.f_m(x_{m,1},\dots,x_{m,n_m}) := f_m^{sk}(x_{m,1},\dots,x_{m,n_m})\}$

Explicit Structural Coverage



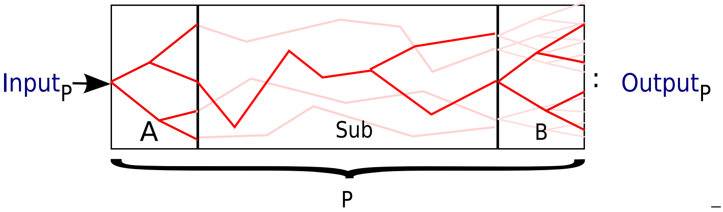
$$Pre, \{for\ x; f(x) := f^{sk}(x)\} Post_C$$
$$\implies \{for\ x; f(x) := f^{sk}(x)\} Post$$

Explicit Structural Coverage



$$Pre, \{for\ \ x; f(x) := f^{sk}(x)\} Post_C$$
$$\implies \{for\ \ x; f(x) := f^{sk}(x)\} Post$$

Explicit Structural Coverage



$$Pre, \{for\ \ x; f(x) := f^{sk}(x)\} Post_C$$
$$\implies \{for\ \ x; f(x) := f^{sk}(x)\} \textcolor{red}{Post}$$

Loops (Invariants)

```
while (k<n) {
  if (j=7) {
    j        = 0;
    line     = new Line(line);
  }
  line.buf[j]=a[k];
  k++; j++;
}
```

Invariant:

$$0 \leq k \leq n \wedge 0 \leq j \leq n \wedge j \leq 7$$

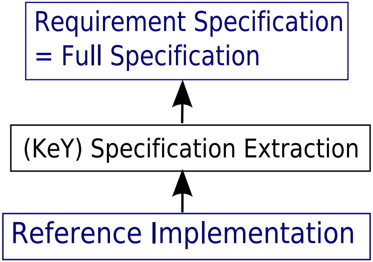
Loops (Invariants)

```
while (k<n) {
  if (j=7) {
    j         = 0;
    line      = new Line(line);
  }
  line.buf[j]=a[k];
  k++; j++;
}
```

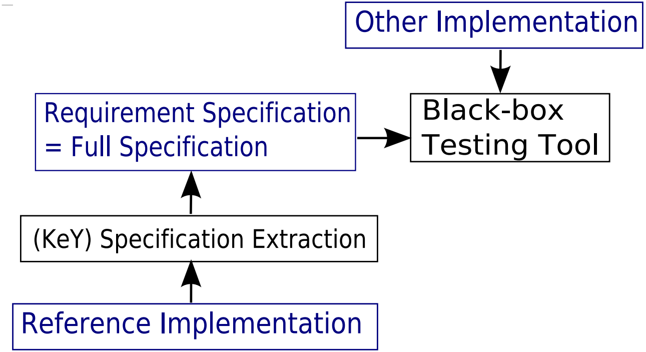
Invariant:

$$0 \leq k \leq n \ \wedge \ 0 \leq j \leq n \ \wedge \ j \leq 7$$

Requirement Specification from a Reference Implementation



Requirement Specification from a Reference Implementation



Conclusion

- Enrich existing Requirement Specification with Program Structure
- Use Black-box Testing tool for White-box testing
- Tools that use Symbolic Execution can be extended
- An Importer and Exporter for a Specification language has to be implemented

Conclusion

- Enrich existing Requirement Specification with Program Structure
- Use Black-box Testing tool for White-box testing
- Tools that use Symbolic Execution can be extended
- An Importer and Exporter for a Specification language has to be implemented

Conclusion

- Enrich existing Requirement Specification with Program Structure
- Use Black-box Testing tool for White-box testing
- Tools that use Symbolic Execution can be extended
- An Importer and Exporter for a Specification language has to be implemented

Conclusion

- Enrich existing Requirement Specification with Program Structure
- Use Black-box Testing tool for White-box testing
- Tools that use Symbolic Execution can be extended
- An Importer and Exporter for a Specification language has to be implemented

Conclusion

- Enrich existing Requirement Specification with Program Structure
- Use Black-box Testing tool for White-box testing
- Tools that use Symbolic Execution can be extended
- An Importer and Exporter for a Specification language has to be implemented

White-box Testing by Combining Deduction-based Specification Extraction and Black-box Testing