Could we have chosen a better Loop Invariant or Method Contract?

Christoph Gladisch

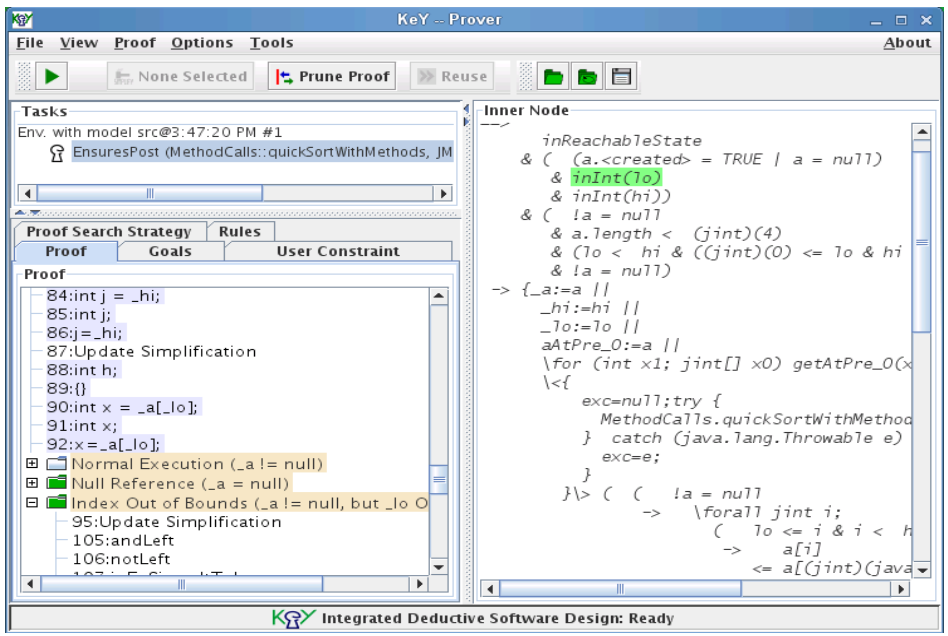
University of Koblenz

Tests and Proofs (TAP'09), July 2nd, 2009, Zürich

The KeY-Project

University of Koblenz (Germany)

KeY-Project



KeY-Tool

.

Introduction

Goal

Combine verification and bug detection in a *very unified* way

Core ideas

- Purely symbolic
- Reuse a maximum of information from a failed proof attempt
- Search bugs on open proof branches

Problem/Challenge

- Falsifiable branches do not imply program error
- Contract rules (more general: abstraction techniques)

Introduction

Goal

Combine verification and bug detection in a *very unified* way

Core ideas

- Purely symbolic
- Reuse a maximum of information from a failed proof attempt
- Search bugs on open proof branches

Problem/Challenge

- Falsifiable branches do not imply program error
- Contract rules (more general: abstraction techniques)

Introduction

Goal

Combine verification and bug detection in a *very unified* way

Core ideas

- Purely symbolic
- Reuse a maximum of information from a failed proof attempt
- Search bugs on open proof branches

Problem/Challenge

- Falsifiable branches do not imply program error
- Contract rules (more general: abstraction techniques)

Dynamic Logic and Sequent Calculus

First-Order Dynamic Logic

$$[\mathrm{p}]\phi \qquad \langle \mathrm{p} \rangle \phi \qquad \{U\}\phi$$

Different Terminology

$$[\mathrm{p}]\phi \qquad wp(p, \phi)$$

(Implication) $\psi \rightarrow [\mathrm{p}]\phi$ $\qquad \{\psi\}p\{\phi\}$ (Hoare triple)

Sequent Calculus Rules

$$\frac{\Gamma_0 \Rightarrow \Delta_0 \ldots \Gamma_n \Rightarrow \Delta_n}{\Gamma \Rightarrow \Delta}$$

Dynamic Logic and Sequent Calculus

First-Order Dynamic Logic

$$[\mathrm{p}]\phi \qquad \langle \mathrm{p} \rangle \phi \qquad \{U\}\phi$$

Different Terminology

$[\mathrm{p}]\phi$ $\qquad$ $wp(p, \phi)$

(Implication) $\psi \rightarrow [\mathrm{p}]\phi$ $\qquad$ $\{\psi\}p\{\phi\}$ (Hoare triple)

Sequent Calculus Rules

$$\frac{\Gamma_0 \Rightarrow \Delta_0 \ldots \Gamma_n \Rightarrow \Delta_n}{\Gamma \Rightarrow \Delta}$$

Dynamic Logic and Sequent Calculus

First-Order Dynamic Logic

$$[\mathrm{p}]\phi \qquad \langle \mathrm{p} \rangle \phi \qquad \{U\}\phi$$

Different Terminology

$$[\mathrm{p}]\phi \qquad wp(p, \phi)$$

(Implication) $\psi \rightarrow [\mathrm{p}]\phi$ $\qquad$ $\{\psi\}p\{\phi\}$ (Hoare triple)

Sequent Calculus Rules

$$\frac{\Gamma_0 \Rightarrow \Delta_0 \ldots \Gamma_n \Rightarrow \Delta_n}{\Gamma \Rightarrow \Delta}$$

Dynamic Logic and Sequent Calculus

First-Order Dynamic Logic

$$[\mathrm{p}]\phi \qquad \langle \mathrm{p} \rangle \phi \qquad \{U\}\phi$$

Different Terminology

$$[\mathrm{p}]\phi \qquad wp(p, \phi)$$

$$\text{(Implication) } \psi \rightarrow [\mathrm{p}]\phi \qquad \{\psi\}p\{\phi\} \text{ (Hoare triple)}$$

Sequent Calculus Rules

$$\frac{\Gamma_0 \Rightarrow \Delta_0 \ldots \Gamma_n \Rightarrow \Delta_n}{\Gamma \Rightarrow \Delta}$$

Dynamic Logic and Sequent Calculus

First-Order Dynamic Logic

$$[\mathrm{p}]\phi \qquad \langle \mathrm{p} \rangle \phi \qquad \{U\}\phi$$

Different Terminology

$$\begin{array}{rll} & [\mathrm{p}]\phi & wp(p,\phi) \\ \text{(Implication) } \psi \rightarrow & [\mathrm{p}]\phi & \{\psi\}p\{\phi\} \text{ (Hoare triple)} \end{array}$$

Sequent Calculus Rules

$$\frac{\Gamma_0 \Rightarrow \Delta_0 \ldots \Gamma_n \Rightarrow \Delta_n}{\Gamma \Rightarrow \Delta}$$

Updates and Modifier Sets

Program transformation to updates

$$[\texttt{o.a = t;}]\phi \quad \rightsquigarrow \quad \{a(o) := t\}\phi$$
$$[\texttt{o.a = t;u.b = s}]\phi \quad \rightsquigarrow \quad \{o.a := t \parallel b(u') := s'\}\phi$$

Modifier Set to Update

- $\{M\}$ (Anonymous Update)
- $a \leq b \wedge c = 1 \wedge \langle\texttt{''prg that modifies c''}\rangle(c = 2 \wedge a \geq c \wedge c \geq b)$
- $a \leq b \wedge c = 1 \wedge \overbrace{\{c := c_{sk}\}}^{\{M\}}(c = 2 \wedge a \geq c \wedge c \geq b)$
- $a \leq b \wedge c = 1 \wedge c_{sk} = 2 \wedge a \geq c_{sk} \wedge c_{sk} \geq b \quad \Rightarrow a = b$

Updates and Modifier Sets

Program transformation to updates

$$[\texttt{o.a = t;}]\phi \quad \rightsquigarrow \quad \{a(o) := t\}\phi$$
$$[\texttt{o.a = t;u.b = s}]\phi \quad \rightsquigarrow \quad \{o.a := t \parallel b(u') := s'\}\phi$$

Modifier Set to Update

- $\{M\}$ (Anonymous Update)
- $a \leq b \wedge c = 1 \wedge \langle\texttt{''prg that modifies c''}\rangle(c = 2 \wedge a \geq c \wedge c \geq b)$
- $a \leq b \wedge c = 1 \wedge \overbrace{\{c := c_{sk}\}}^{\{M\}}(c = 2 \wedge a \geq c \wedge c \geq b)$
- $a \leq b \wedge c = 1 \wedge c_{sk} = 2 \wedge a \geq c_{sk} \wedge c_{sk} \geq b \quad \Rightarrow a = b$

Updates and Modifier Sets

Program transformation to updates

$$\begin{array}{rcl} [\texttt{o.a = t;}]\phi & \rightsquigarrow & \{a(o) := t\}\phi \\ [\texttt{o.a = t;u.b = s}]\phi & \rightsquigarrow & \{o.a := t \,||\, b(u') := s'\}\phi \end{array}$$

Modifier Set to Update

- $\{M\}$ (Anonymous Update)
- $a \leq b \wedge c = 1 \wedge \langle\texttt{''prg that modifies c''}\rangle(c = 2 \wedge a \geq c \wedge c \geq b)$
- $a \leq b \wedge c = 1 \wedge \overbrace{\{c := c_{sk}\}}^{\{M\}}(c = 2 \wedge a \geq c \wedge c \geq b)$
- $a \leq b \wedge c = 1 \wedge c_{sk} = 2 \wedge a \geq c_{sk} \wedge c_{sk} \geq b \quad \Rightarrow a = b$

Updates and Modifier Sets

Program transformation to updates

$$\begin{aligned} [\texttt{o.a = t;}]\phi &\rightsquigarrow \{a(o) := t\}\phi \\ [\texttt{o.a = t;u.b = s}]\phi &\rightsquigarrow \{o.a := t \parallel b(u') := s'\}\phi \end{aligned}$$

Modifier Set to Update

- $\{M\}$ (Anonymous Update)
- $a \leq b \wedge c = 1 \wedge \langle\texttt{''prg that modifies c''}\rangle(c = 2 \wedge a \geq c \wedge c \geq b)$
- $a \leq b \wedge c = 1 \wedge \overbrace{\{c := c_{sk}\}}^{\{M\}}(c = 2 \wedge a \geq c \wedge c \geq b)$
- $a \leq b \wedge c = 1 \wedge c_{sk} = 2 \wedge a \geq c_{sk} \wedge c_{sk} \geq b \quad \Rightarrow a = b$

Updates and Modifier Sets

Program transformation to updates

$$\begin{array}{rcl} [\texttt{o.a = t;}]\phi & \rightsquigarrow & \{a(o) := t\}\phi \\ [\texttt{o.a = t;u.b = s}]\phi & \rightsquigarrow & \{o.a := t \parallel b(u') := s'\}\phi \end{array}$$

Modifier Set to Update

- $\{M\}$ (Anonymous Update)
- $a \leq b \wedge c = 1 \wedge \langle\texttt{''prg that modifies c''}\rangle(c = 2 \wedge a \geq c \wedge c \geq b)$
- $a \leq b \wedge c = 1 \wedge \overbrace{\{c := c_{sk}\}}^{\{M\}}(c = 2 \wedge a \geq c \wedge c \geq b)$
- $a \leq b \wedge c = 1 \wedge c_{sk} = 2 \wedge a \geq c_{sk} \wedge c_{sk} \geq b \quad \Rightarrow a = b$

Contract Rules

Method Contract Rule $(pre_m, post_m, M, \langle[]\rangle)$ $M = mod(p)$

$$\frac{\begin{array}{lll} \Gamma & \Rightarrow & \{U\}pre_m, \Delta \\ \Gamma & \Rightarrow & \{U\}(pre_m \rightarrow \langle[m()]\rangle post_m), \Delta \\ \Gamma & \Rightarrow & \{U\}\{M\}(post_m \rightarrow post), \Delta \end{array}}{\Gamma \Rightarrow \{U\}\langle[m()]\rangle post, \Delta}$$

Loop Invariant Rule $(I, I \wedge \neg c, M, [])$ $M = mod(b)$

$$\frac{\begin{array}{lll} \Gamma & \Rightarrow & \{U\}I, \Delta \\ \Gamma & \Rightarrow & \{U\}\{M\}(I \wedge c \rightarrow [\mathtt{b}]I), \Delta \\ \Gamma & \Rightarrow & \{U\}\{M\}((I \wedge \neg c) \rightarrow post), \Delta \end{array}}{\Gamma \Rightarrow \{U\}[\mathtt{while(c)\{b;\}}]post, \Delta}$$

Example

JAVA + JML

```
/*@ public normal_behavior
 requires x>=0;
 ensures \result*\result<=x && (\result+1)*(\result+1)>x;
 diverges true;
@*/
public int sqrtA(int x){
 int i=0;
 /*@ loop_invariant (i-1)*(i-1)<=x; modifies i;@*/
 while(i*i<=x){i++;}
 return i;
}
```

JAVA + JML

Proof Tree

$$\dfrac{\dfrac{\dfrac{}{\vdots}}{\Gamma \to I} \quad \dfrac{\vdots}{\Gamma \to \{U,M\}(I \wedge c \to [\mathtt{b}]I)} \quad \dfrac{\vdots}{\Gamma \to \{U,M\}(I \wedge \neg c \to [\mathtt{ret..}]\phi)}}{\Gamma \to \{U\}[\mathtt{while(c)\{b\};}][\mathtt{ret..}]\phi}}{\Gamma \to [\mathtt{i=0;}][\mathtt{while(c)\{b\};}][\mathtt{ret..}]\phi}}{\underbrace{\Gamma}_{pre} \to [\mathtt{sqrt();}] \underbrace{\phi}_{post}}$$

1. Try to verify program
2. Improve contract until 1st and 2nd branch of contract rule is proved
3. Check falsifiability of 3rd branch
4. prove **SFP**, "**S***pecial* **F***alsifiability* **P***reservation*"

$$((\{M^1 := M^2\}S_n) \wedge \{U\}\{M^2\}post) \to S_n$$

Proof Tree

$$\dfrac{\dfrac{}{\Gamma \to I} \quad \dfrac{}{\Gamma \to \{U,M\}(I \wedge c \to [\texttt{b}]I)} \quad \dfrac{}{\Gamma \to \{U,M\}(I \wedge \neg c \to [\texttt{ret..}]\phi)}}{\Gamma \to \{U\}[\texttt{while(c)\{b\};}][\texttt{ret..}]\phi}}{\Gamma \to [\texttt{i=0;}][\texttt{while(c)\{b\};}][\texttt{ret..}]\phi}}{\underbrace{\Gamma}_{pre} \to [\texttt{sqrt();}] \underbrace{\phi}_{post}}$$

1. Try to verify program
2. Improve contract until 1st and 2nd branch of contract rule is proved
3. Check falsifiability of 3rd branch
4. prove **SFP**, "**S***pecial* **F***alsifiability* **P***reservation*"

$$((\{M^1 := M^2\}S_n) \wedge \{U\}\{M^2\}post) \to S_n$$

Proof Tree

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdots}}{\Gamma \rightarrow I} \quad \dfrac{\vdots}{\Gamma \rightarrow \{U, M\}(I \wedge c \rightarrow [\mathtt{b}]I)} \quad \dfrac{\vdots}{\Gamma \rightarrow \{U, M\}(I \wedge \neg c \rightarrow [\mathtt{ret..}]\phi)}}{\Gamma \rightarrow \{U\}[\mathtt{while(c)\{b\};}][\mathtt{ret..}]\phi}}{\Gamma \rightarrow [\mathtt{i=0;}][\mathtt{while(c)\{b\};}][\mathtt{ret..}]\phi}}{\underbrace{\Gamma}_{pre} \rightarrow [\mathtt{sqrt();}] \underbrace{\phi}_{post}}$$

1. Try to verify program
2. Improve contract until 1st and 2nd branch of contract rule is proved
3. Check falsifiability of 3rd branch
4. prove **SFP**, "**S***pecial* **F***alsifiability* **P***reservation*"

$$((\{M^1 := M^2\}S_n) \wedge \{U\}\{M^2\}post) \rightarrow S_n$$

Proof Tree

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdots}\quad \Gamma \to I \qquad \Gamma \to \{U, M\}(I \wedge c \to [\mathtt{b}]I) \qquad \Gamma \to \{U, M\}(I \wedge \neg c \to [\mathtt{ret..}]\phi)}{\Gamma \to \{U\}[\mathtt{while(c)\{b\};}][\mathtt{ret..}]\phi}}{\Gamma \to [\mathtt{i=0;}][\mathtt{while(c)\{b\};}][\mathtt{ret..}]\phi}}{\underbrace{\Gamma}_{pre} \to [\mathtt{sqrt();}] \underbrace{\phi}_{post}}$$

1. Try to verify program
2. Improve contract until 1st and 2nd branch of contract rule is proved
3. Check falsifiability of 3rd branch
4. prove **SFP**, "**S***pecial* **F***alsifiability* **P***reservation*"

$$((\{M^1 := M^2\}S_n) \wedge \{U\}\{M^2\}post) \to S_n$$

Proof Tree

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdots}}{\Gamma \to I} \quad \dfrac{\vdots}{\Gamma \to \{U, M\}(I \wedge c \to [\texttt{b}]I)} \quad \dfrac{\vdots}{\Gamma \to \{U, M\}(I \wedge \neg c \to [\texttt{ret..}]\phi)}}{\Gamma \to \{U\}[\texttt{while(c)\{b\};}][\texttt{ret..}]\phi}}{\Gamma \to [\texttt{i=0;}][\texttt{while(c)\{b\};}][\texttt{ret..}]\phi}}{\underbrace{\Gamma}_{pre} \to [\texttt{sqrt();}] \underbrace{\phi}_{post}}$$

1. Try to verify program
2. Improve contract until 1st and 2nd branch of contract rule is proved
3. Check falsifiability of 3rd branch
4. prove **SFP**, "**S***pecial* **F***alsifiability* **P***reservation*"

$$((\{M^1 := M^2\}S_n) \wedge \{U\}\{M^2\}post) \to S_n$$

Proof Tree

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\neg\text{“}(\Gamma \rightarrow I)\text{“}}\,SMT}{\vdots}}{\Gamma \rightarrow I} \quad \dfrac{\vdots}{\Gamma \rightarrow \{U,M\}(I \wedge c \rightarrow [\texttt{b}]I)} \quad \dfrac{\vdots}{\Gamma \rightarrow \{U,M\}(I \wedge \neg c \rightarrow [\texttt{ret}}}{\Gamma \rightarrow \{U\}[\texttt{while(c)\{b\};}][\texttt{ret..}]\phi}}{\Gamma \rightarrow [\texttt{i=0;}][\texttt{while(c)\{b\};}][\texttt{ret..}]\phi}}{\underbrace{\Gamma}_{pre} \rightarrow [\texttt{sqrt();}]\underbrace{\phi}_{post}}$$

1. Try to verify program
2. Improve contract until 1st and 2nd branch of contract rule is proved
3. Check falsifiability of 3rd branch
4. prove **SFP**, “**S***pecial* **F***alsifiability* **P***reservation*”

$$((\{M^1 := M^2\}S_n) \wedge \{U\}\{M^2\}post) \rightarrow S_n$$

Proof Tree

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\neg\text{“}(\Gamma \rightarrow I)\text{“}}\,SMT}{\vdots}}{\Gamma \rightarrow I} \quad \dfrac{\vdots}{\Gamma \rightarrow \{U, M\}(I \wedge c \rightarrow [\texttt{b}]I)} \quad \dfrac{\vdots}{\Gamma \rightarrow \{U, M\}(I \wedge \neg c \rightarrow [\texttt{ret}}}{\Gamma \rightarrow \{U\}[\texttt{while(c)\{b\};}][\texttt{ret..}]\phi}}{\Gamma \rightarrow [\texttt{i=0;}][\texttt{while(c)\{b\};}][\texttt{ret..}]\phi}}{\underbrace{\Gamma}_{pre} \rightarrow [\texttt{sqrt();}] \underbrace{\phi}_{post}}$$

1. Try to verify program
2. Improve contract until 1st and 2nd branch of contract rule is proved
3. Check falsifiability of 3rd branch
4. prove **SFP**, “**S***pecial* **F***alsifiability* **P***reservation*”

$$((\{M^1 := M^2\}S_n) \wedge \{U\}\{M^2\}post) \rightarrow S_n$$

Example: make first branch closeable

JAVA + JML

```
/*@ public normal_behavior
 requires x>=0;
 ensures \result*\result<=x && (\result+1)*(\result+1)>x;
 diverges true;
@*/
public int sqrtA(int x){
 int i=0;
 /*@ loop_invariant (i-1)*(i-1)<=x; modifies i;@*/
 while(i*i<=x){i++;}
 return i;
}
```

JAVA + JML

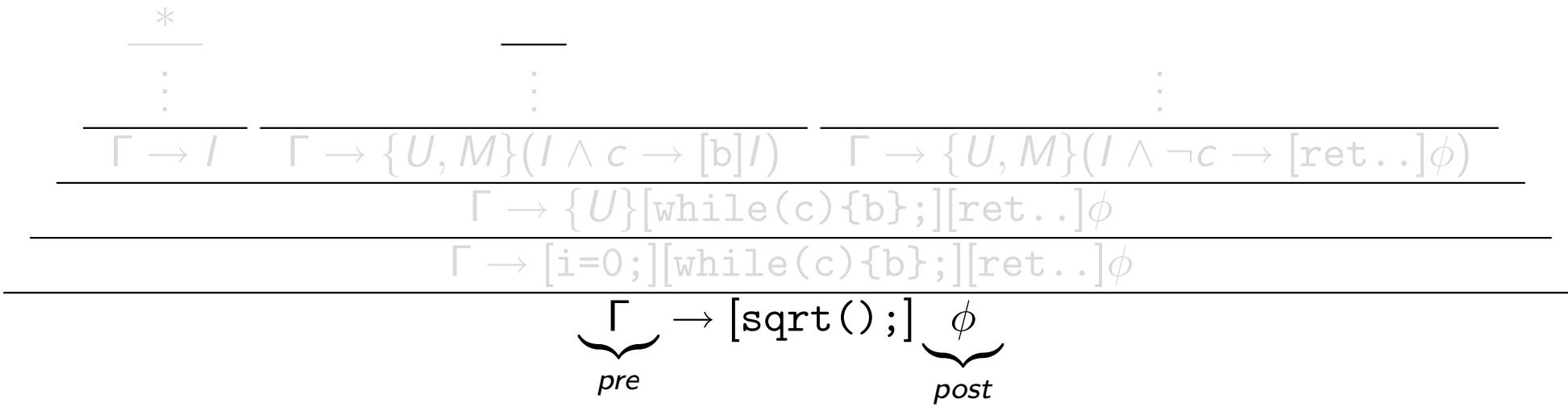
Example: make first branch closeable

JAVA + JML

```
/*@ public normal_behavior
 requires x>=0;
 ensures \result*\result<=x && (\result+1)*(\result+1)>x;
 diverges true;
@*/
public int sqrtA(int x){
 int i=0;
 /*@ loop_invariant i*i<=x+1; modifies i;@*/
 while(i*i<=x){i++;}
 return i;
}
```

JAVA + JML

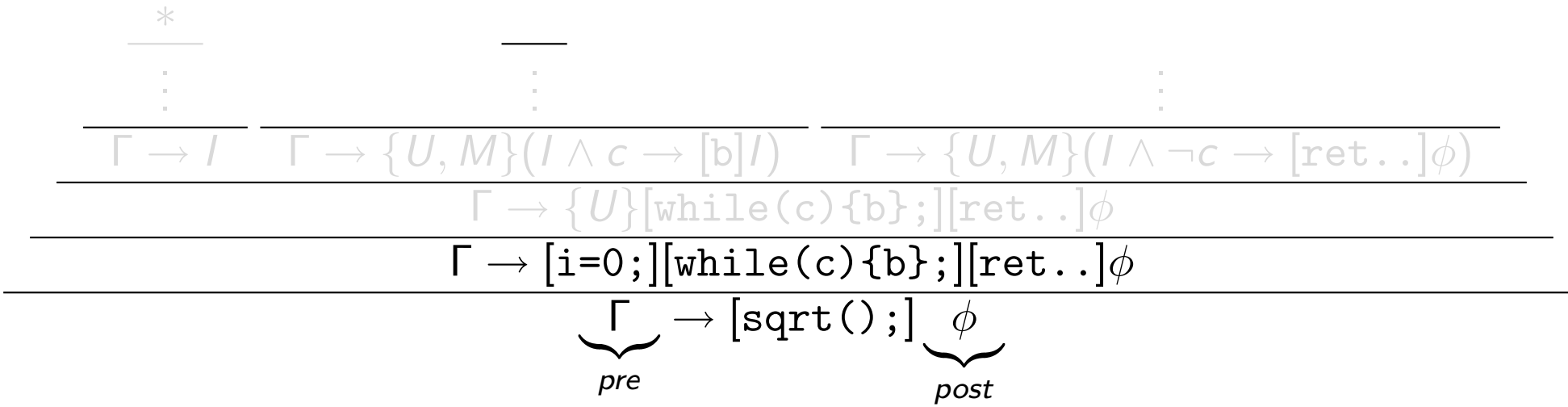
Proof Tree



1. Try to verify program
2. Improve contract until 1st and 2nd branch of contract rule is proved
3. Check falsifiability of 3rd branch
4. prove **SFP**, "**S***pecial* **F***alsifiability* **P***reservation*"

$$((\{M^1 := M^2\}S_n) \wedge \{U\}\{M^2\}post) \rightarrow S_n$$

Proof Tree



1. Try to verify program
2. Improve contract until 1st and 2nd branch of contract rule is proved
3. Check falsifiability of 3rd branch
4. prove **SFP**, "**S***pecial* **F***alsifiability* **P***reservation*"

$$((\{M^1 := M^2\}S_n) \wedge \{U\}\{M^2\}post) \rightarrow S_n$$

Proof Tree

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdots}\,*}{\Gamma \to I} \quad \dfrac{\dfrac{}{\vdots}}{\Gamma \to \{U,M\}(I \wedge c \to [\mathtt{b}]I)} \quad \dfrac{\vdots}{\Gamma \to \{U,M\}(I \wedge \neg c \to [\mathtt{ret..}]\phi)}}{\Gamma \to \{U\}[\mathtt{while(c)\{b\};}][\mathtt{ret..}]\phi}}{\Gamma \to [\mathtt{i=0;}][\mathtt{while(c)\{b\};}][\mathtt{ret..}]\phi}}{\underbrace{\Gamma}_{pre} \to [\mathtt{sqrt();}] \underbrace{\phi}_{post}}$$

1. Try to verify program
2. Improve contract until 1st and 2nd branch of contract rule is proved
3. Check falsifiability of 3rd branch
4. prove **SFP**, "**S***pecial* **F***alsifiability* **P***reservation*"

$$((\{M^1 := M^2\}S_n) \wedge \{U\}\{M^2\}post) \to S_n$$

Proof Tree

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \to I} \quad \dfrac{}{\Gamma \to \{U,M\}(I \wedge c \to [\mathtt{b}]I)} \quad \dfrac{\vdots}{\Gamma \to \{U,M\}(I \wedge \neg c \to [\mathtt{ret..}]\phi)}}{\Gamma \to \{U\}[\mathtt{while(c)\{b\};}][\mathtt{ret..}]\phi}}{\Gamma \to [\mathtt{i=0;}][\mathtt{while(c)\{b\};}][\mathtt{ret..}]\phi}}{\underbrace{\Gamma}_{pre} \to [\mathtt{sqrt();}] \underbrace{\phi}_{post}}$$

1. Try to verify program
2. Improve contract until 1st and 2nd branch of contract rule is proved
3. Check falsifiability of 3rd branch
4. prove **SFP**, "**S***pecial* **F***alsifiability* **P***reservation*"

$$((\{M^1 := M^2\}S_n) \wedge \{U\}\{M^2\}post) \to S_n$$

Proof Tree

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{*}{\vdots}}{\Gamma \to I} \quad \dfrac{\dfrac{}{\vdots}}{\Gamma \to \{U, M\}(I \wedge c \to [\mathtt{b}]I)} \quad \dfrac{\vdots}{\Gamma \to \{U, M\}(I \wedge \neg c \to [\mathtt{ret..}]\phi)}}{\Gamma \to \{U\}[\mathtt{while(c)\{b\};}][\mathtt{ret..}]\phi}}{\Gamma \to [\mathtt{i=0;}][\mathtt{while(c)\{b\};}][\mathtt{ret..}]\phi}}{\underbrace{\Gamma}_{pre} \to [\mathtt{sqrt();}] \underbrace{\phi}_{post}}$$

1. Try to verify program
2. Improve contract until 1st and 2nd branch of contract rule is proved
3. Check falsifiability of 3rd branch
4. prove **SFP**, "**S***pecial* **F***alsifiability* **P***reservation*"

$$((\{M^1 := M^2\}S_n) \wedge \{U\}\{M^2\}post) \to S_n$$

Proof Tree

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{*}{\vdots}}{\Gamma \to I} \quad \dfrac{\dfrac{}{\vdots}}{\Gamma \to \{U, M\}(I \wedge c \to [\mathtt{b}]I)} \quad \dfrac{\vdots}{\Gamma \to \{U, M\}(I \wedge \neg c \to [\mathtt{ret..}]\phi)}}{\Gamma \to \{U\}[\mathtt{while(c)\{b\};}][\mathtt{ret..}]\phi}}{\Gamma \to [\mathtt{i=0;}][\mathtt{while(c)\{b\};}][\mathtt{ret..}]\phi}}{\underbrace{\Gamma}_{pre} \to [\mathtt{sqrt();}] \underbrace{\phi}_{post}}$$

1. Try to verify program
2. Improve contract until 1st and 2nd branch of contract rule is proved
3. Check falsifiability of 3rd branch
4. prove **SFP**, "**S***pecial* **F***alsifiability* **P***reservation*"

$$((\{M^1 := M^2\}S_n) \wedge \{U\}\{M^2\}post) \to S_n$$

Proof Tree

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{*}{\vdots}}{\Gamma \rightarrow I} \quad \dfrac{\neg\text{“}(\Gamma \rightarrow \{U, M\}(I \wedge c \rightarrow [\mathtt{b}]I))\text{“}}{\vdots}\ SMT}{\Gamma \rightarrow \{U, M\}(I \wedge c \rightarrow [\mathtt{b}]I)} \quad \dfrac{\vdots}{\Gamma \rightarrow \{U, M\}(I \wedge \neg c \rightarrow [\mathtt{ret}}}{\Gamma \rightarrow \{U\}[\mathtt{while(c)\{b\};}][\mathtt{ret..}]\phi}}{\Gamma \rightarrow [\mathtt{i=0;}][\mathtt{while(c)\{b\};}][\mathtt{ret..}]\phi}}{\underbrace{\Gamma}_{pre} \rightarrow [\mathtt{sqrt();}] \underbrace{\phi}_{post}}$$

1. Try to verify program
2. Improve contract until 1st and 2nd branch of contract rule is proved
3. Check falsifiability of 3rd branch
4. prove **SFP**, “**S***pecial* **F***alsifiability* **P***reservation*”

$$((\{M^1 := M^2\}S_n) \wedge \{U\}\{M^2\}post) \rightarrow S_n$$

Example: make second branch closeable

JAVA + JML

```
/*@ public normal_behavior
 requires x>=0;
 ensures \result*\result<=x && (\result+1)*(\result+1)>x;
 diverges true;
@*/
public int sqrtA(int x){
 int i=0;
 /*@ loop_invariant i*i<=x+1; modifies i;@*/
 while(i*i<=x){i++;}
 return i;
}
```

JAVA + JML

Example: make second branch closeable

JAVA + JML

```
/*@ public normal_behavior
 requires x>=0;
 ensures \result*\result<=x && (\result+1)*(\result+1)>x;
 diverges true;
@*/
public int sqrtA(int x){
 int i=0;
 /*@ loop_invariant (i-1)*(i-1)<=x || i==0; modifies i;@*/
 while(i*i<=x){i++;}
 return i;
}
```

JAVA + JML

Proof Tree

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdots}\,*}{\Gamma \rightarrow I} \quad \dfrac{\dfrac{}{\vdots}\,*}{\Gamma \rightarrow \{U, M\}(I \wedge c \rightarrow [\mathtt{b}]I)} \quad \dfrac{\dfrac{}{\vdots}}{\Gamma \rightarrow \{U, M\}(I \wedge \neg c \rightarrow [\mathtt{ret..}]\phi)}}{\Gamma \rightarrow \{U\}[\mathtt{while(c)\{b\};}][\mathtt{ret..}]\phi}}{\Gamma \rightarrow [\mathtt{i=0;}][\mathtt{while(c)\{b\};}][\mathtt{ret..}]\phi}}{\underbrace{\Gamma}_{pre} \rightarrow [\mathtt{sqrt();}] \underbrace{\phi}_{post}}$$

1. Try to verify program
2. Improve contract until 1st and 2nd branch of contract rule is proved
3. Check falsifiability of 3rd branch
   (The falsifiability does **not imply** the existence of a bug)
4. prove **SFP**, "**S***pecial* **F***alsifiability* **P***reservation*"

$$((\{M^1 := M^2\}S_n) \wedge \{U\}\{M^2\}\underbrace{\phi}_{post}) \rightarrow S_n$$

Implies the existence of a program bug on execution path S_n!

Proof Tree

$$\dfrac{\dfrac{\dfrac{}{\vdots}}{\Gamma \rightarrow I} \quad \dfrac{\dfrac{}{\vdots}}{\Gamma \rightarrow \{U,M\}(I \wedge c \rightarrow [\mathtt{b}]I)} \quad \dfrac{\neg\text{“}\Gamma \rightarrow \{U,M\}(I \wedge \neg c \rightarrow \phi)\text{“}}{\vdots}\,SMT}{\Gamma \rightarrow \{U,M\}(I \wedge \neg c \rightarrow [\mathtt{ret..}]\phi)}}{\Gamma \rightarrow \{U\}[\mathtt{while(c)\{b\};}][\mathtt{ret..}]\phi}}{\Gamma \rightarrow [\mathtt{i=0;}][\mathtt{while(c)\{b\};}][\mathtt{ret..}]\phi}}{\underbrace{\Gamma}_{pre} \rightarrow [\mathtt{sqrt();}]\underbrace{\phi}_{post}}$$

1. Try to verify program
2. Improve contract until 1st and 2nd branch of contract rule is proved
3. Check falsifiability of 3rd branch
   (The falsifiability does **not imply** the existence of a bug)
4. prove **SFP**, “**S***pecial* **F***alsifiability* **P***reservation*”

$$((\{M^1 := M^2\}S_n) \wedge \{U\}\{M^2\}\underbrace{\phi}_{post}) \rightarrow S_n$$

Implies the existence of a program bug on execution path S_n!

Proof Tree

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \rightarrow I}\,* \quad \dfrac{}{\Gamma \rightarrow \{U,M\}(I \wedge c \rightarrow [\texttt{b}]I)}\,* \quad \dfrac{}{\neg\text{“}\Gamma \rightarrow \{U,M\}(I \wedge \neg c \rightarrow \phi)\text{“}}\,SMT}{\Gamma \rightarrow \{U,M\}(I \wedge \neg c \rightarrow [\texttt{ret..}]\phi)}}{\Gamma \rightarrow \{U\}[\texttt{while(c)\{b\};}][\texttt{ret..}]\phi}}{\Gamma \rightarrow [\texttt{i=0;}][\texttt{while(c)\{b\};}][\texttt{ret..}]\phi}}{\underbrace{\Gamma}_{pre} \rightarrow [\texttt{sqrt();}]\underbrace{\phi}_{post}}$$

1. Try to verify program
2. Improve contract until 1st and 2nd branch of contract rule is proved
3. Check falsifiability of 3rd branch
   (The falsifiability does **not imply** the existence of a bug)
4. prove **SFP**, “**S***pecial* **F***alsifiability* **P***reservation*”

$$((\{M^1 := M^2\}S_n) \wedge \{U\}\{M^2\}\underbrace{\phi}_{post}) \rightarrow S_n$$

Implies the existence of a program bug on execution path S_n!

Proof Tree

$$\dfrac{\dfrac{\dfrac{}{\vdots}\,*}{\Gamma \rightarrow I} \quad \dfrac{\dfrac{}{\vdots}\,*}{\Gamma \rightarrow \{U,M\}(I \wedge c \rightarrow [\mathtt{b}]I)} \quad \dfrac{\dfrac{}{\vdots}\,\mathit{SMT}}{\neg\text{“}\Gamma \rightarrow \{U,M\}(I \wedge \neg c \rightarrow \phi)\text{“}}}{\Gamma \rightarrow \{U\}[\mathtt{while(c)\{b\};}][\mathtt{ret..}]\phi}}{\Gamma \rightarrow [\mathtt{i=0;}][\mathtt{while(c)\{b\};}][\mathtt{ret..}]\phi}}{\underbrace{\Gamma}_{pre} \rightarrow [\mathtt{sqrt();}]\underbrace{\phi}_{post}}$$

1. Try to verify program
2. Improve contract until 1st and 2nd branch of contract rule is proved
3. Check falsifiability of 3rd branch
   (The falsifiability does **not imply** the existence of a bug)
4. prove **SFP**, “**S***pecial* **F***alsifiability* **P***reservation*”

$$((\{M^1 := M^2\}S_n) \wedge \{U\}\{M^2\}\underbrace{\phi}_{post}) \rightarrow S_n$$

Implies the existence of a program bug on execution path S_n!

Proof Tree

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{*}{\vdots}}{\Gamma \rightarrow I} \quad \dfrac{\dfrac{*}{\vdots}}{\Gamma \rightarrow \{U, M\}(I \wedge c \rightarrow [\mathtt{b}]I)} \quad \dfrac{\dfrac{S_n}{\vdots}}{\Gamma \rightarrow \{U, M\}(I \wedge \neg c \rightarrow [\mathtt{ret..}]\phi)}}{\Gamma \rightarrow \{U\}[\mathtt{while(c)\{b\};}][\mathtt{ret..}]\phi}}{\Gamma \rightarrow [\mathtt{i=0;}][\mathtt{while(c)\{b\};}][\mathtt{ret..}]\phi}}{\underbrace{\Gamma}_{pre} \rightarrow [\mathtt{sqrt();}] \underbrace{\phi}_{post}}$$

1. Try to verify program
2. Improve contract until 1st and 2nd branch of contract rule is proved
3. Check falsifiability of 3rd branch
   (The falsifiability does **not imply** the existence of a bug)
4. prove **SFP**, "**S***pecial* **F***alsifiability* **P***reservation*"

$$((\{M^1 := M^2\}S_n) \wedge \{U\}\{M^2\}\underbrace{\phi}_{post}) \rightarrow S_n$$

Implies the existence of a program bug on execution path S_n!

Example: make second branch closeable

JAVA + JML

```
/*@ public normal_behavior
 requires x>=0;
 ensures \result*\result<=x && (\result+1)*(\result+1)>x;
 diverges true;
@*/
public int sqrtA(int x){
 int i=0;
 /*@ loop_invariant (i-1)*(i-1)<=x || i==0; modifies i;@*/
 while(i*i<=x){i++;}
 return i;
}
```

JAVA + JML

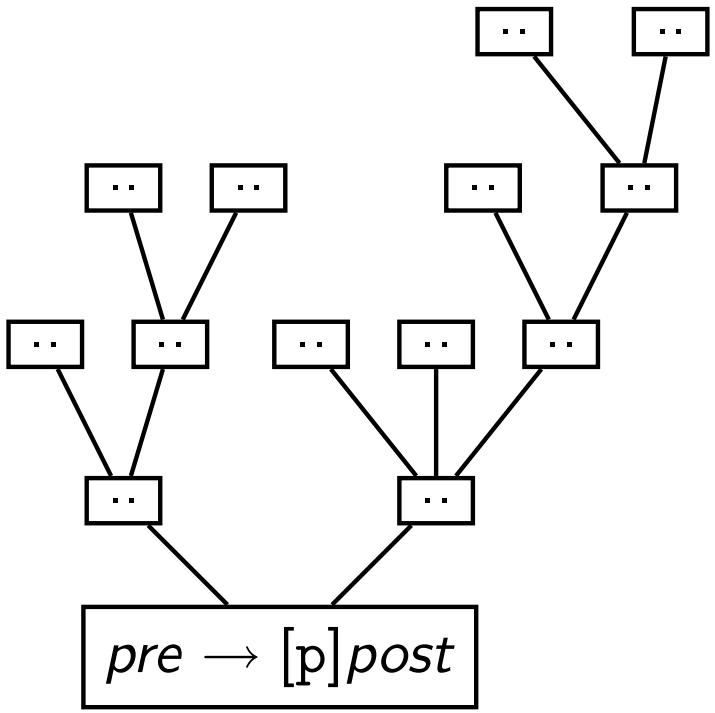
Example: make second branch closeable

JAVA + JML

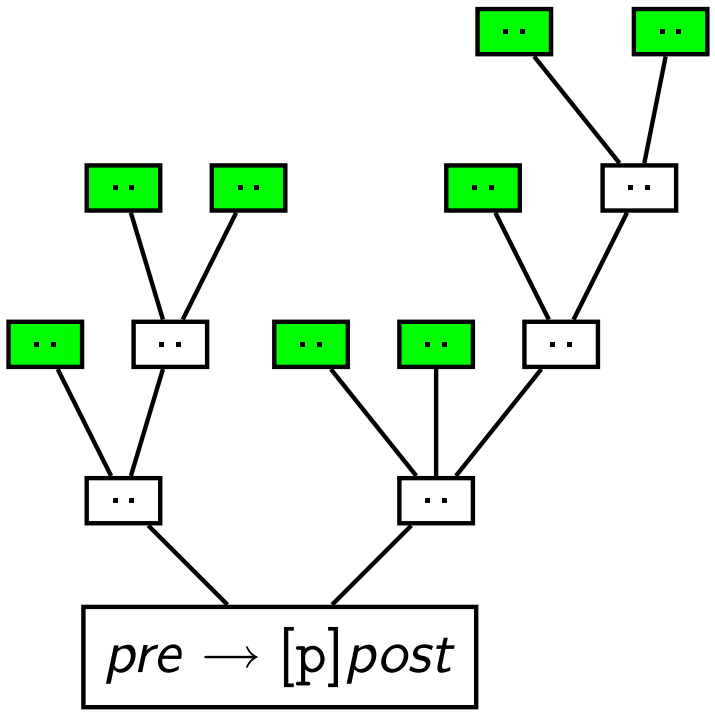
```
/*@ public normal_behavior
 requires x>=0;
 ensures \result*\result<=x && (\result+1)*(\result+1)>x;
 diverges true;
@*/
public int sqrtA(int x){
 int i=0;
 /*@ loop_invariant (i-1)*(i-1)<=x || i==0; modifies i;@*/
 while(i*i<=x){i++;}
 return i-1;
}
```

JAVA + JML

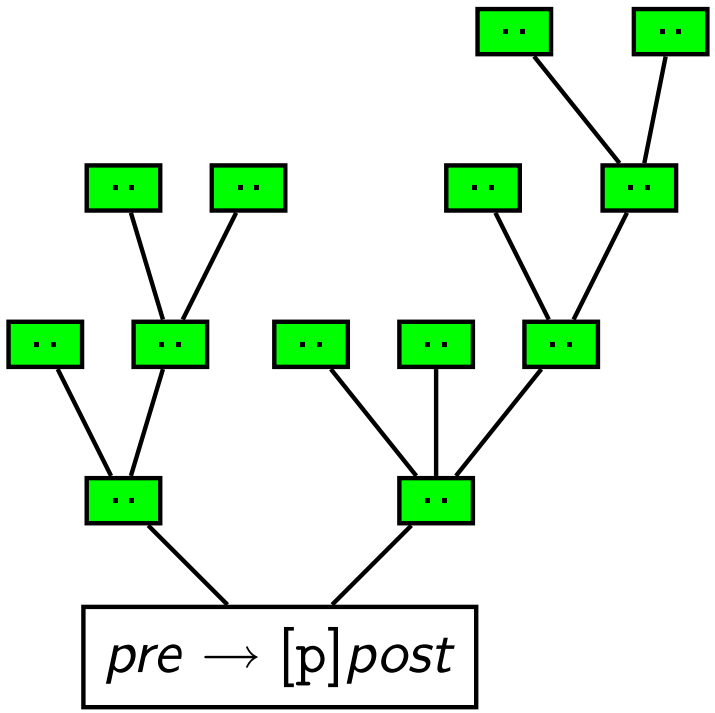
Understanding Falsifiability Preservation Analysis



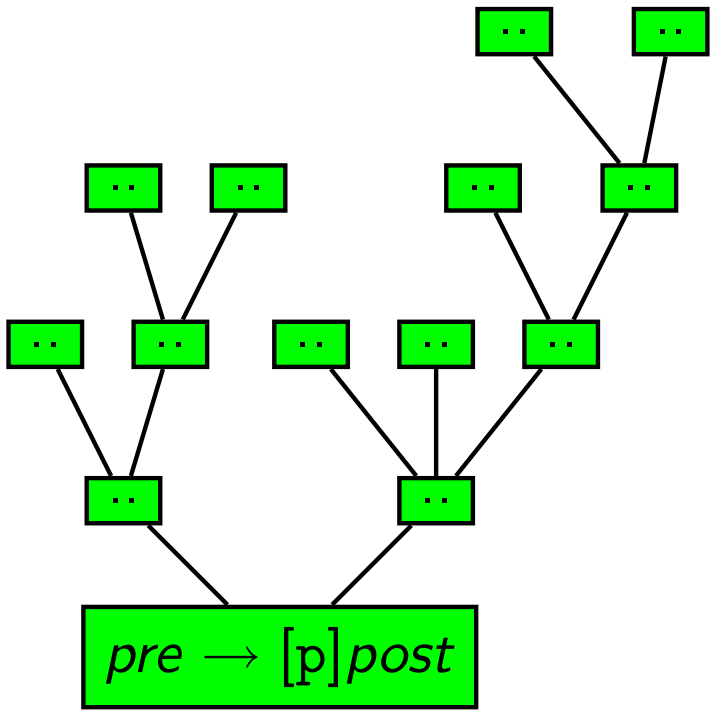
Understanding Falsifiability Preservation Analysis



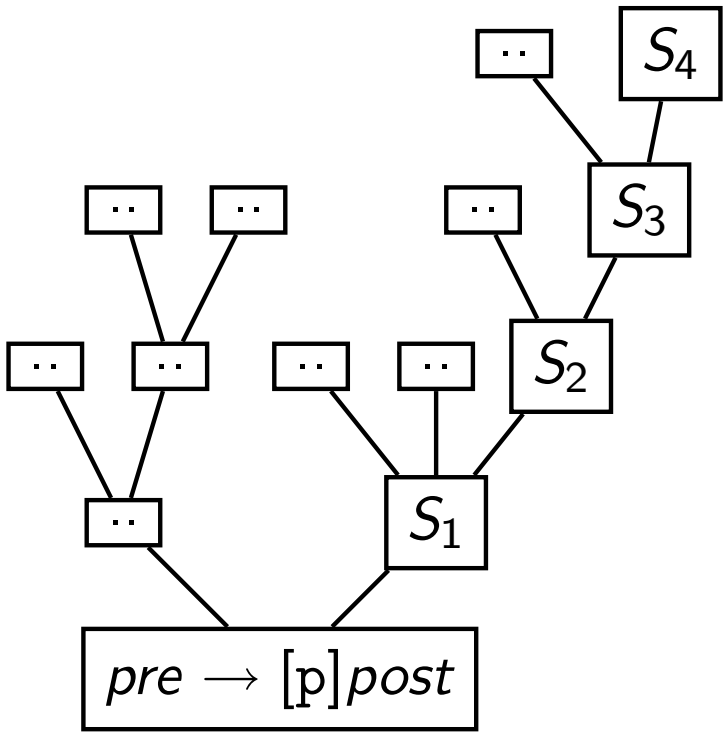
Understanding Falsifiability Preservation Analysis



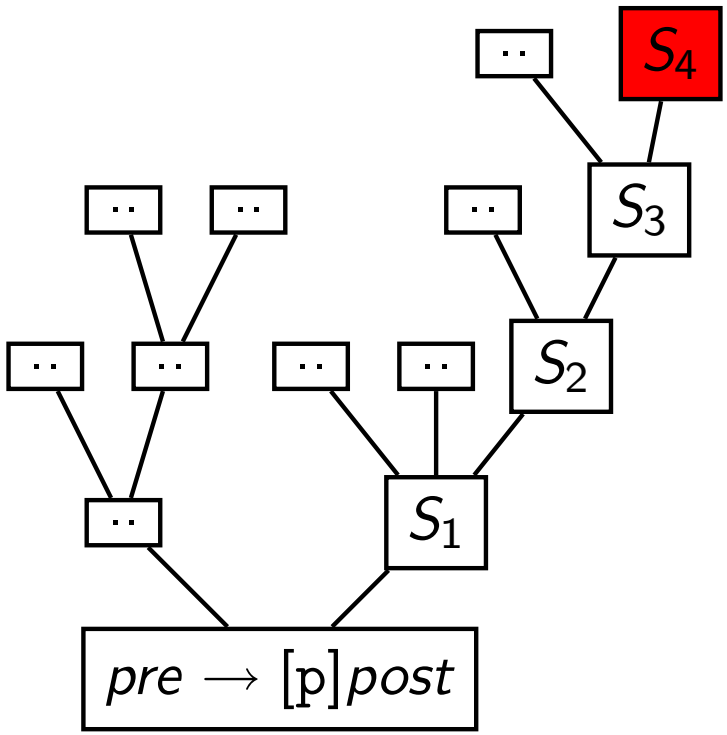
Understanding Falsifiability Preservation Analysis



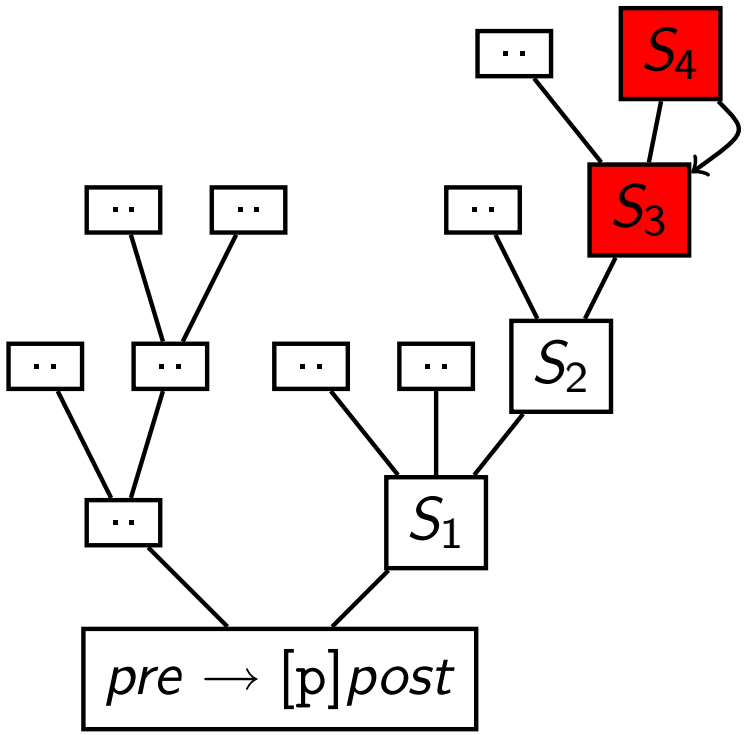
Understanding Falsifiability Preservation Analysis



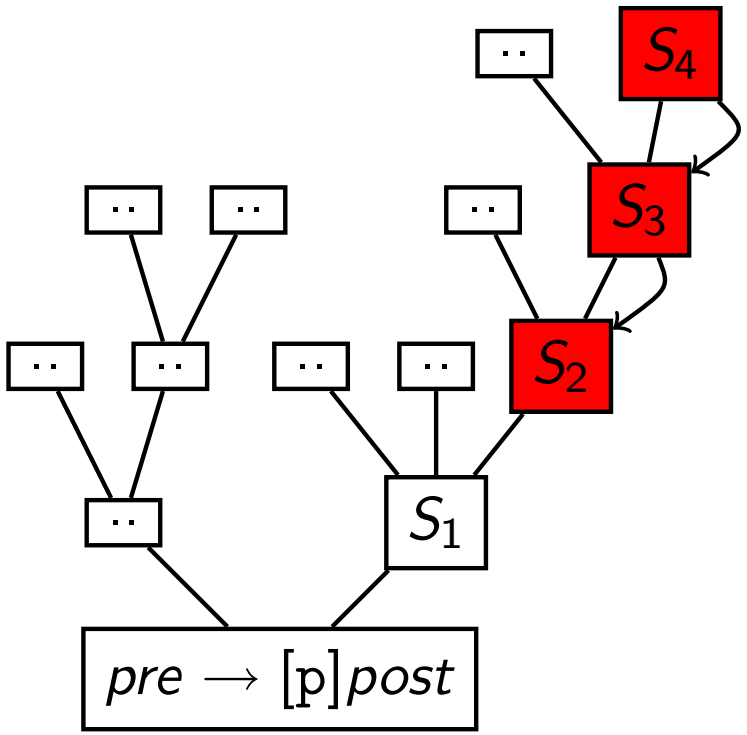
Understanding Falsifiability Preservation Analysis



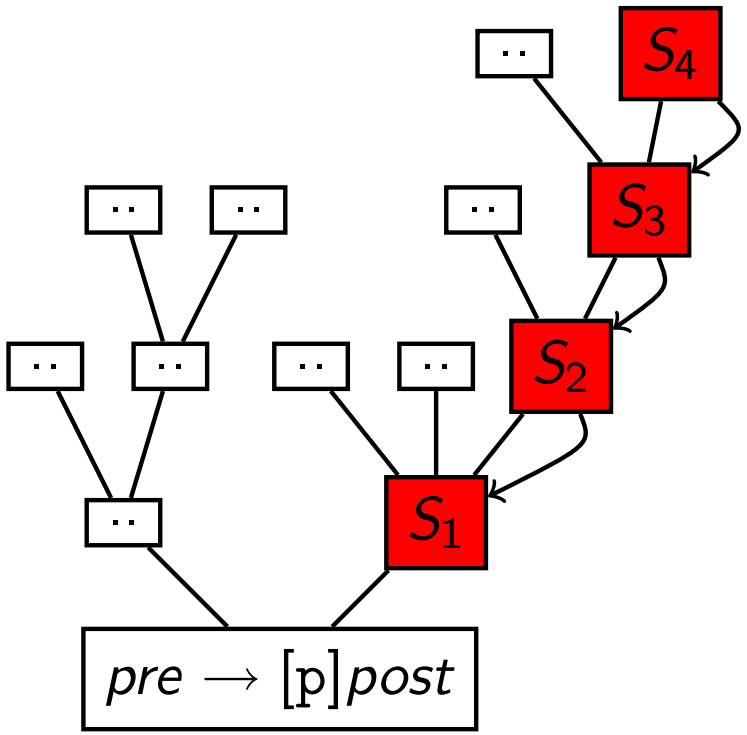
Understanding Falsifiability Preservation Analysis



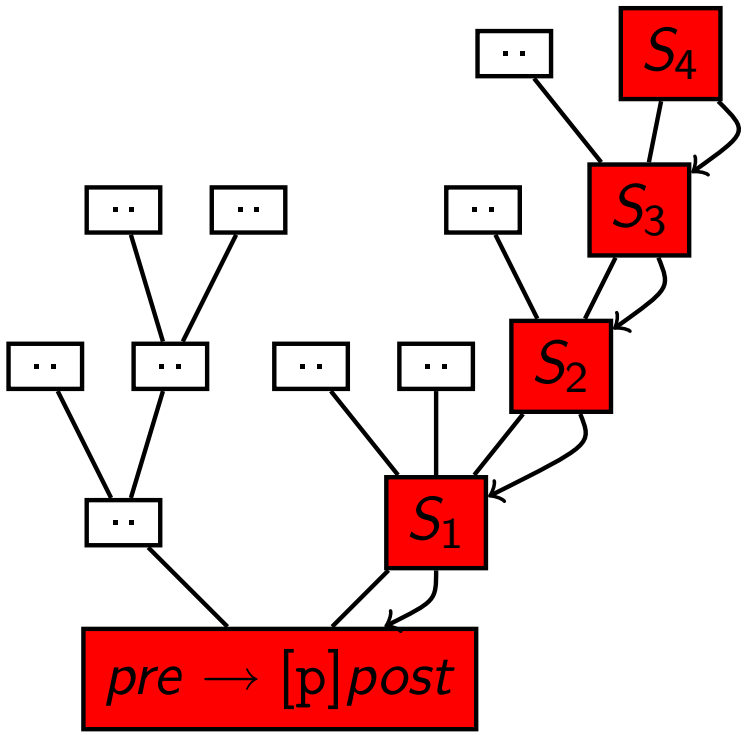
Understanding Falsifiability Preservation Analysis



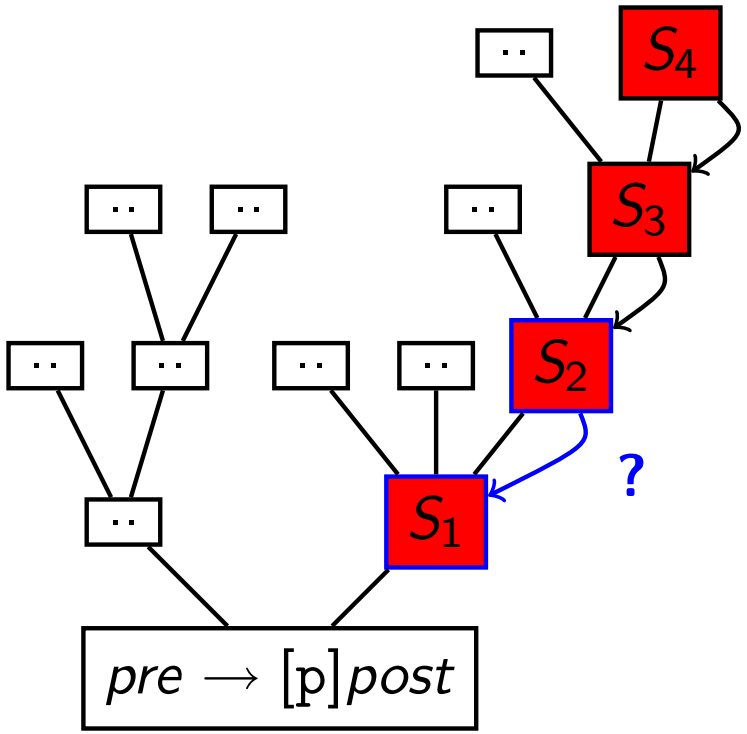
Understanding Falsifiability Preservation Analysis



Understanding Falsifiability Preservation Analysis



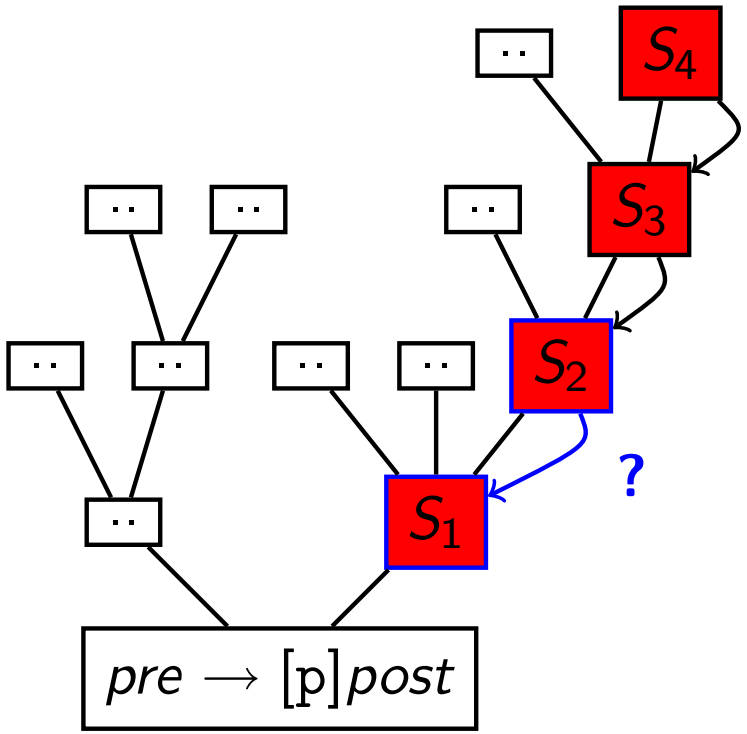
Understanding Falsifiability Preservation Analysis



Contract Rule Application at S_1

$$\frac{\Gamma \rightarrow I \quad \Gamma \rightarrow \{U, M\}(I \wedge c \rightarrow [\mathtt{b}]I) \quad \Gamma \rightarrow \{U, M\}(I \wedge \neg c \rightarrow [\mathtt{ret..}]\phi)}{\Gamma \rightarrow \{U\}[\mathtt{while(c)\{b\};}][\mathtt{ret..}]\phi}$$

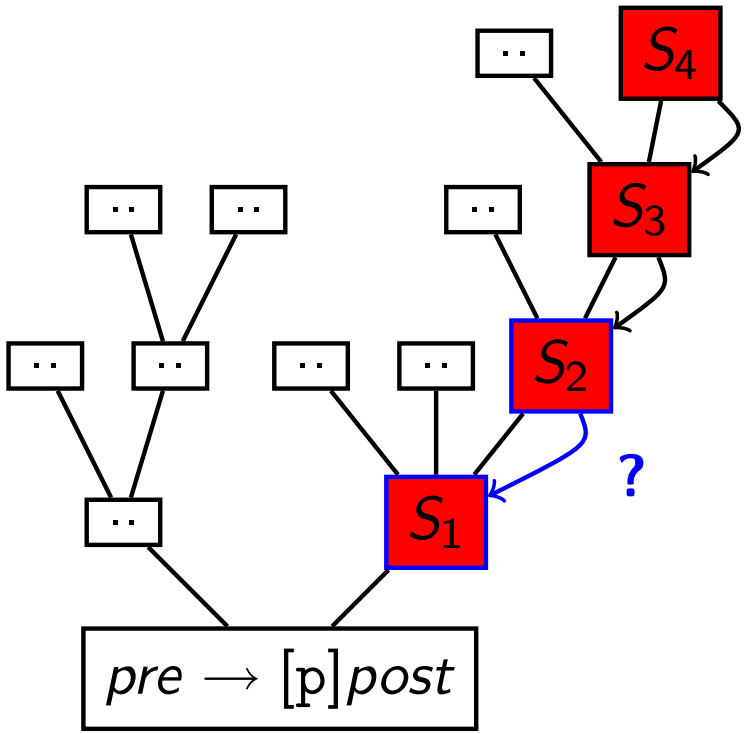
Understanding Falsifiability Preservation Analysis



Falsification Preservation Check (Approach 1)

$$(\exists s'.\{s'\}\neg S_2) \quad \rightarrow \quad (\exists s''.\{s''\}\neg S_1)$$

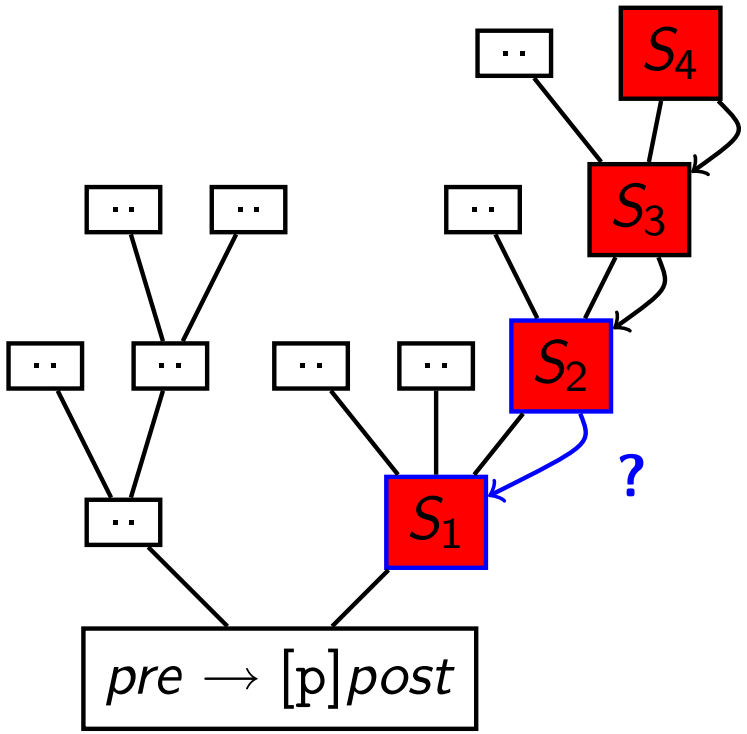
Understanding Falsifiability Preservation Analysis



Falsification Preservation Check (Approach 1)

$$(\exists s'.\{s'\}\neg S_2) \rightarrow (\exists s''.\{s''\}\neg S_1)$$
$$(\{sk'\}\neg S_2) \rightarrow (\exists s''.\{s''\}\neg S_1)$$

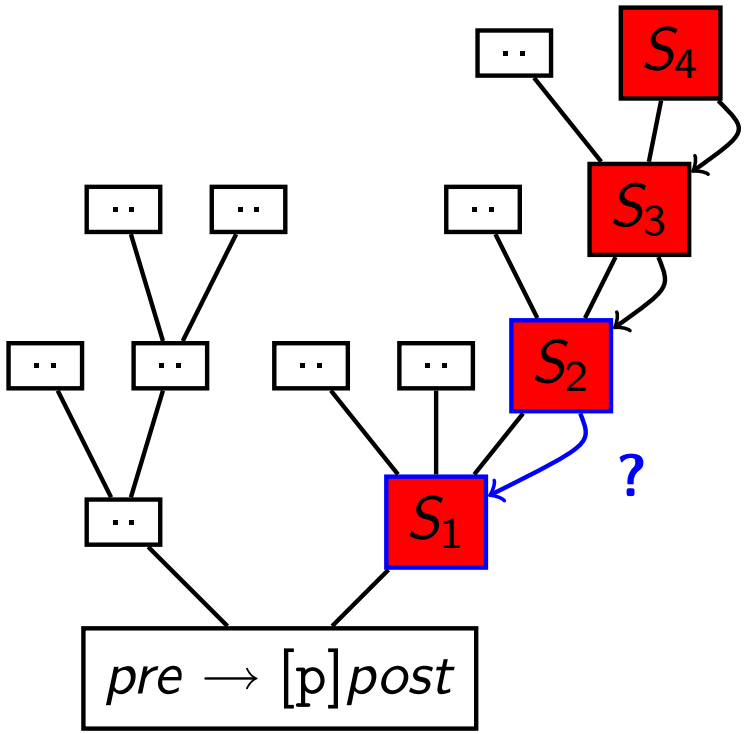
Understanding Falsifiability Preservation Analysis



Falsification Preservation Check (Approach 1)

$$
\begin{aligned}
(\exists s'.\{s'\}\neg S_2) &\rightarrow (\exists s''.\{s''\}\neg S_1) \\
(\{sk'\}\neg S_2) &\rightarrow (\exists s''.\{s''\}\neg S_1) \\
(\{sk'\}\neg S_2) &\rightarrow (\{sk'\}\neg S_1)
\end{aligned}
$$

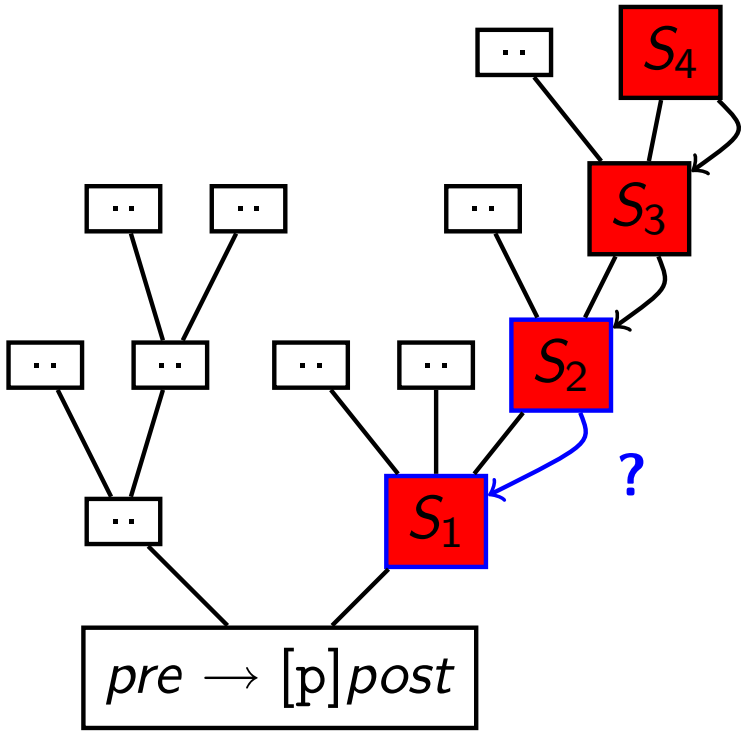
Understanding Falsifiability Preservation Analysis



Falsification Preservation Check (Approach 2)

$$\neg S_2 \quad \longrightarrow \quad \neg S_1$$

Understanding Falsifiability Preservation Analysis

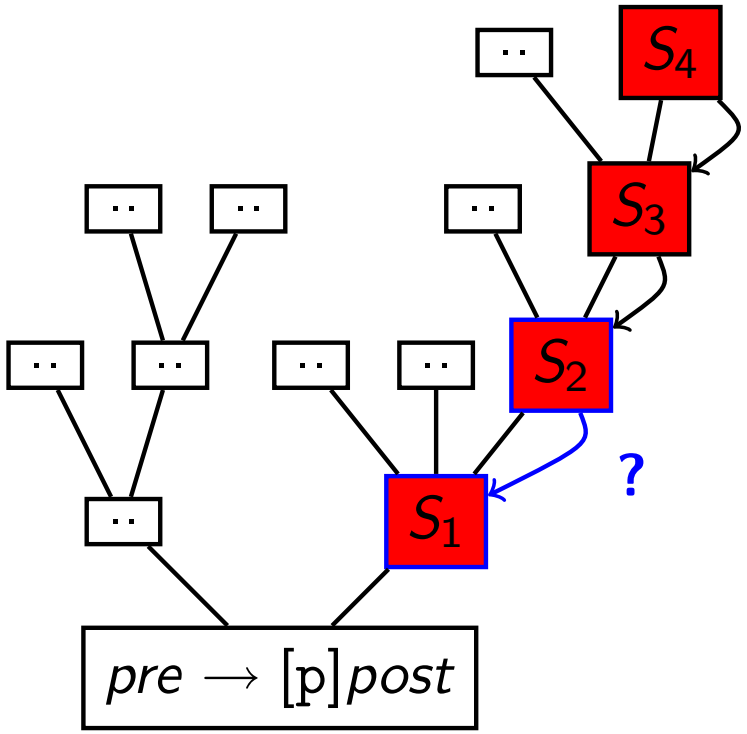


Falsification Preservation Check (Approach 2)

$$\neg S_2 \quad \longrightarrow \quad \neg S_1$$

Similar to checking the **strength of a contract**: *Contr.* $\subset$ *Prog.*

Understanding Falsifiability Preservation Analysis



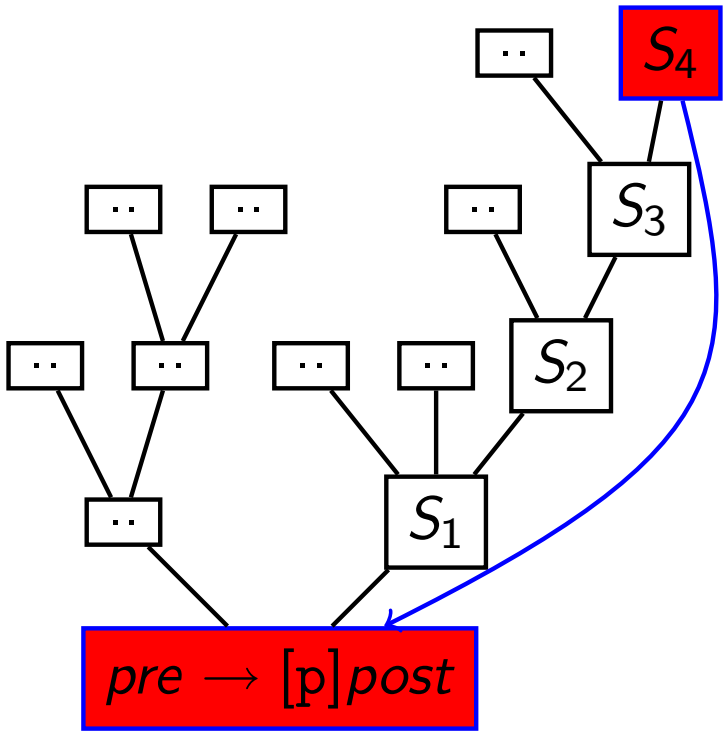
Falsification Preservation Check (Approach 2)

$$\neg S_2 \quad \rightarrow \quad \neg S_1$$

Similar to checking the **strength of a contract**: $Contr. \subset Prog.$

In contrast, a **contract is satisfied** if: $Prog. \subset Contr.$

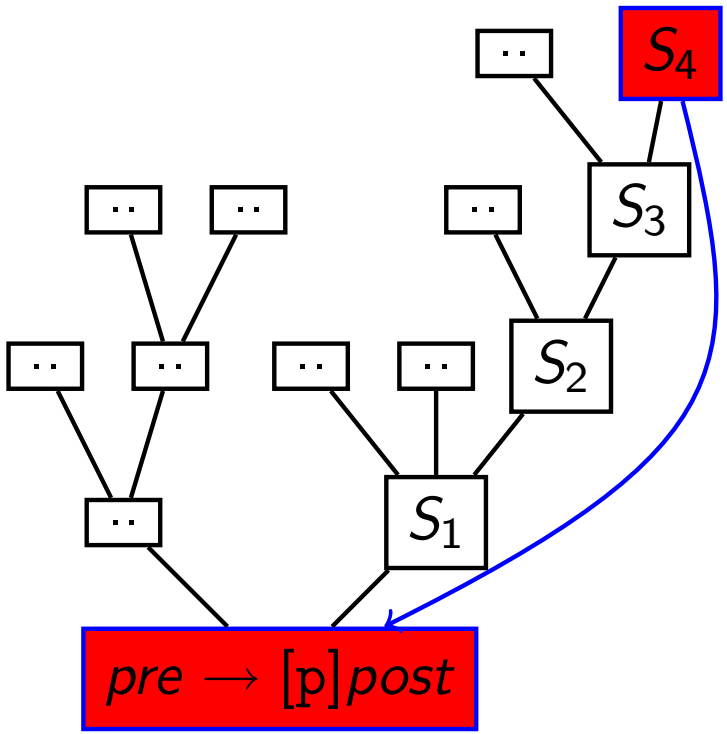
Understanding Falsifiability Preservation Analysis



Falsification Preservation Check (Approach 3)

$$\neg S_4 \quad \longrightarrow \quad \neg S_0$$

Understanding Falsifiability Preservation Analysis

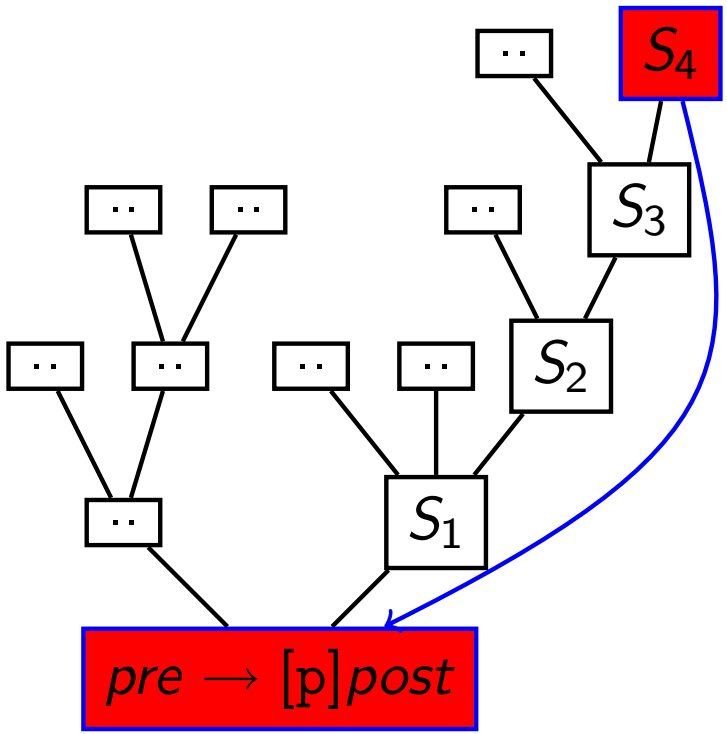


Falsification Preservation Check (Approach 3)

$$\neg S_4 \quad \longrightarrow \quad \neg S_0$$

if $\exists I.I \vDash \neg S_4$, and $\vDash \neg S_4 \longrightarrow \neg S_0$, then $\exists I.I \vDash \neg S_0$

Understanding Falsifiability Preservation Analysis

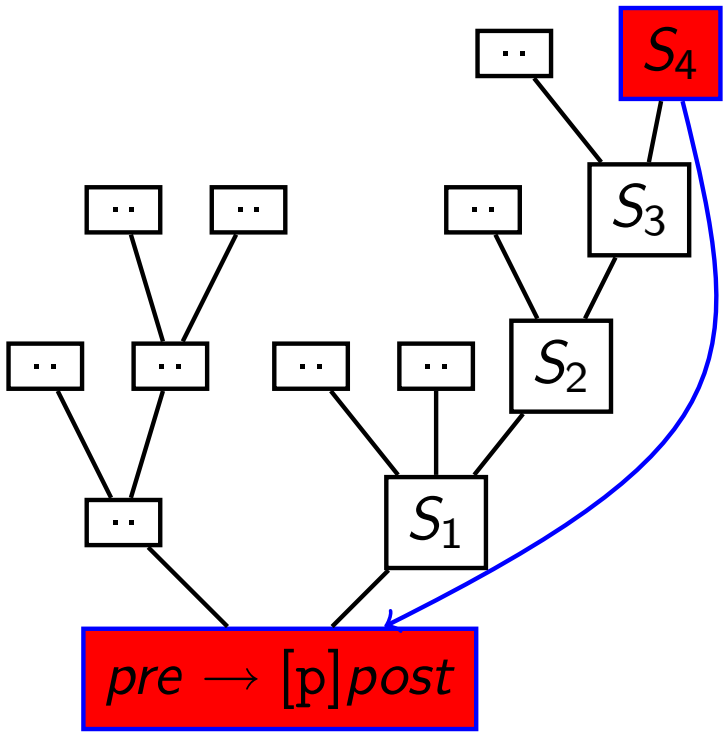


Falsification Preservation Check (Approach 3)

$$\neg S_4 \quad \longrightarrow \quad \neg S_0$$

if $\exists I.I \vDash \neg S_4$, and $\vDash \neg S_4 \longrightarrow \neg S_0$, then $\exists I.I \vDash \neg S_0$

Understanding Falsifiability Preservation Analysis

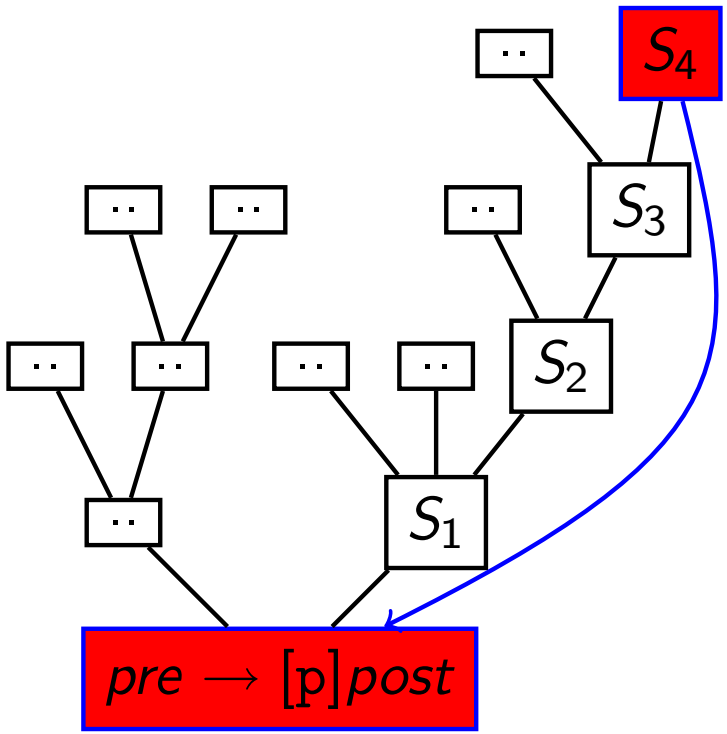


Falsification Preservation Check (Approach 3)

$$\neg S_4 \quad \longrightarrow \quad \neg S_0$$

if $\exists I.I \vDash \neg S_4$, and $\vDash \neg S_4 \longrightarrow \neg S_0$, then $\exists I.I \vDash \neg S_0$

Understanding Falsifiability Preservation Analysis



Falsification Preservation Check (Approach 3)

$$\neg S_4 \quad \longrightarrow \quad \neg S_0$$

if $\exists I.I \vDash \neg S_4$, and $\vDash \neg S_4 \longrightarrow \neg S_0$, then $\exists I.I \vDash \neg S_0$

Falsifiability Preservation Analysis

Falsifiability Preservation of a Branch

- "path- and postcondition-focussed" strength condition
- For a branch $S_0 \ldots S_n$ the $FP_{S_0}^{S_n}$ condition is $\models \neg S_n \rightarrow \neg S_0$
- "falsifiable branch" $\longrightarrow \neg(pre \longrightarrow \langle \mathrm{p} \rangle post)$

Properties of the approach

- Simple to understand and to implement
- Sound for bug detection, wrt. to soundness of constraint/sat solving
- Complete for bug detection (any bug can be found)
- Search space for bugs is pruned drastically: Correct/ infeasible paths
- Starts with what people want: to verify a program
- Can disambiguate verification failure
- **Allows a "performance boost" by specialization**

Falsifiability Preservation Analysis

Falsifiability Preservation of a Branch

- "path- and postcondition-focussed" strength condition
- For a branch $S_0 \ldots S_n$ the $FP_{S_0}^{S_n}$ condition is $\models \neg S_n \rightarrow \neg S_0$
- "falsifiable branch" $\longrightarrow \neg(pre \longrightarrow \langle \mathrm{p} \rangle post)$

Properties of the approach

- Simple to understand and to implement
- Sound for bug detection, wrt. to soundness of constraint/sat solving
- Complete for bug detection (any bug can be found)
- Search space for bugs is pruned drastically: Correct/ infeasible paths
- Starts with what people want: to verify a program
- Can disambiguate verification failure
- **Allows a "performance boost" by specialization**

Falsifiability Preservation Analysis

Falsifiability Preservation of a Branch

- "path- and postcondition-focussed" strength condition
- For a branch $S_0 \ldots S_n$ the $FP_{S_0}^{S_n}$ condition is $\models \neg S_n \rightarrow \neg S_0$
- "falsifiable branch" $\longrightarrow \neg(pre \longrightarrow \langle \mathrm{p} \rangle post)$

Properties of the approach

- Simple to understand and to implement
- Sound for bug detection, wrt. to soundness of constraint/sat solving
- Complete for bug detection (any bug can be found)
- Search space for bugs is pruned drastically: Correct/ infeasible paths
- Starts with what people want: to verify a program
- Can disambiguate verification failure
- **Allows a "performance boost" by specialization**

Falsifiability Preservation Analysis

Falsifiability Preservation of a Branch

- "path- and postcondition-focussed" strength condition
- For a branch $S_0 \ldots S_n$ the $FP_{S_0}^{S_n}$ condition is $\models \neg S_n \rightarrow \neg S_0$
- "falsifiable branch" $\longrightarrow \neg(pre \longrightarrow \langle \mathrm{p} \rangle post)$

Properties of the approach

- Simple to understand and to implement
- Sound for bug detection, wrt. to soundness of constraint/sat solving
- Complete for bug detection (any bug can be found)
- Search space for bugs is pruned drastically: Correct/ infeasible paths
- Starts with what people want: to verify a program
- Can disambiguate verification failure
- **Allows a "performance boost" by specialization**

Falsifiability Preservation Analysis

Falsifiability Preservation of a Branch

- "path- and postcondition-focussed" strength condition
- For a branch $S_0 \ldots S_n$ the $FP_{S_0}^{S_n}$ condition is $\models \neg S_n \rightarrow \neg S_0$
- "falsifiable branch" $\longrightarrow \neg(pre \longrightarrow \langle \mathrm{p} \rangle post)$

Properties of the approach

- Simple to understand and to implement
- Sound for bug detection, wrt. to soundness of constraint/sat solving
- Complete for bug detection (any bug can be found)
- Search space for bugs is pruned drastically: Correct/ infeasible paths
- Starts with what people want: to verify a program
- Can disambiguate verification failure
- **Allows a "performance boost" by specialization**

Falsifiability Preservation Analysis

Falsifiability Preservation of a Branch

- "path- and postcondition-focussed" strength condition
- For a branch $S_0 \ldots S_n$ the $FP_{S_0}^{S_n}$ condition is $\models \neg S_n \rightarrow \neg S_0$
- "falsifiable branch" $\longrightarrow \neg(pre \longrightarrow \langle \mathrm{p} \rangle post)$

Properties of the approach

- Simple to understand and to implement
- Sound for bug detection, wrt. to soundness of constraint/sat solving
- Complete for bug detection (any bug can be found)
- Search space for bugs is pruned drastically: Correct/ infeasible paths
- Starts with what people want: to verify a program
- Can disambiguate verification failure
- Allows a "performance boost" by specialization

Falsifiability Preservation Analysis

Falsifiability Preservation of a Branch

- "path- and postcondition-focussed" strength condition
- For a branch $S_0 \ldots S_n$ the $FP_{S_0}^{S_n}$ condition is $\models \neg S_n \rightarrow \neg S_0$
- "falsifiable branch" $\longrightarrow \neg(pre \longrightarrow \langle \mathrm{p} \rangle post)$

Properties of the approach

- Simple to understand and to implement
- Sound for bug detection, wrt. to soundness of constraint/sat solving
- Complete for bug detection (any bug can be found)
- Search space for bugs is pruned drastically: Correct/ infeasible paths
- Starts with what people want: to verify a program
- Can disambiguate verification failure
- Allows a "performance boost" by specialization

Falsifiability Preservation Analysis

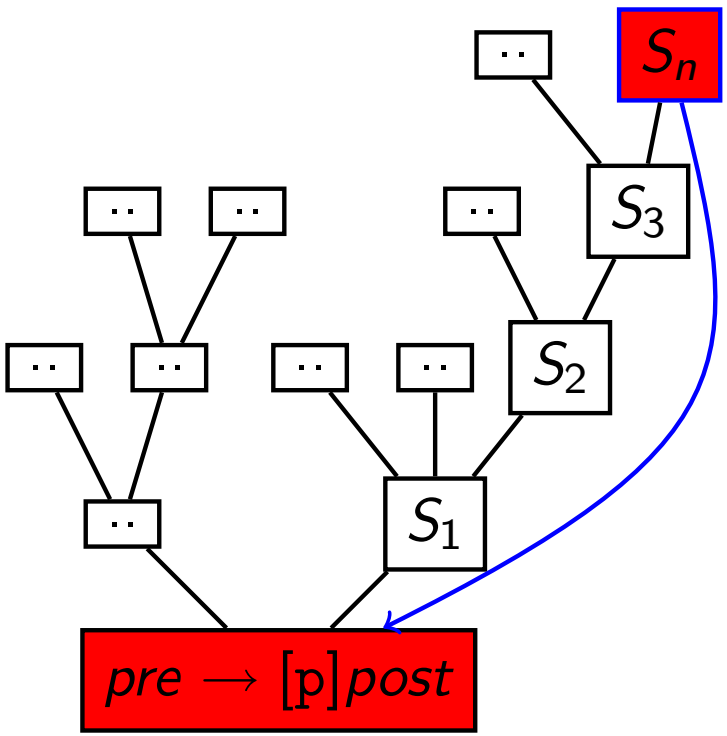
Falsifiability Preservation of a Branch

- "path- and postcondition-focussed" strength condition
- For a branch $S_0 \ldots S_n$ the $FP_{S_0}^{S_n}$ condition is $\models \neg S_n \rightarrow \neg S_0$
- "falsifiable branch" $\longrightarrow \neg(pre \longrightarrow \langle \mathrm{p} \rangle post)$

Properties of the approach

- Simple to understand and to implement
- Sound for bug detection, wrt. to soundness of constraint/sat solving
- Complete for bug detection (any bug can be found)
- Search space for bugs is pruned drastically: Correct/ infeasible paths
- Starts with what people want: to verify a program
- Can disambiguate verification failure
- **Allows a "performance boost" by specialization**

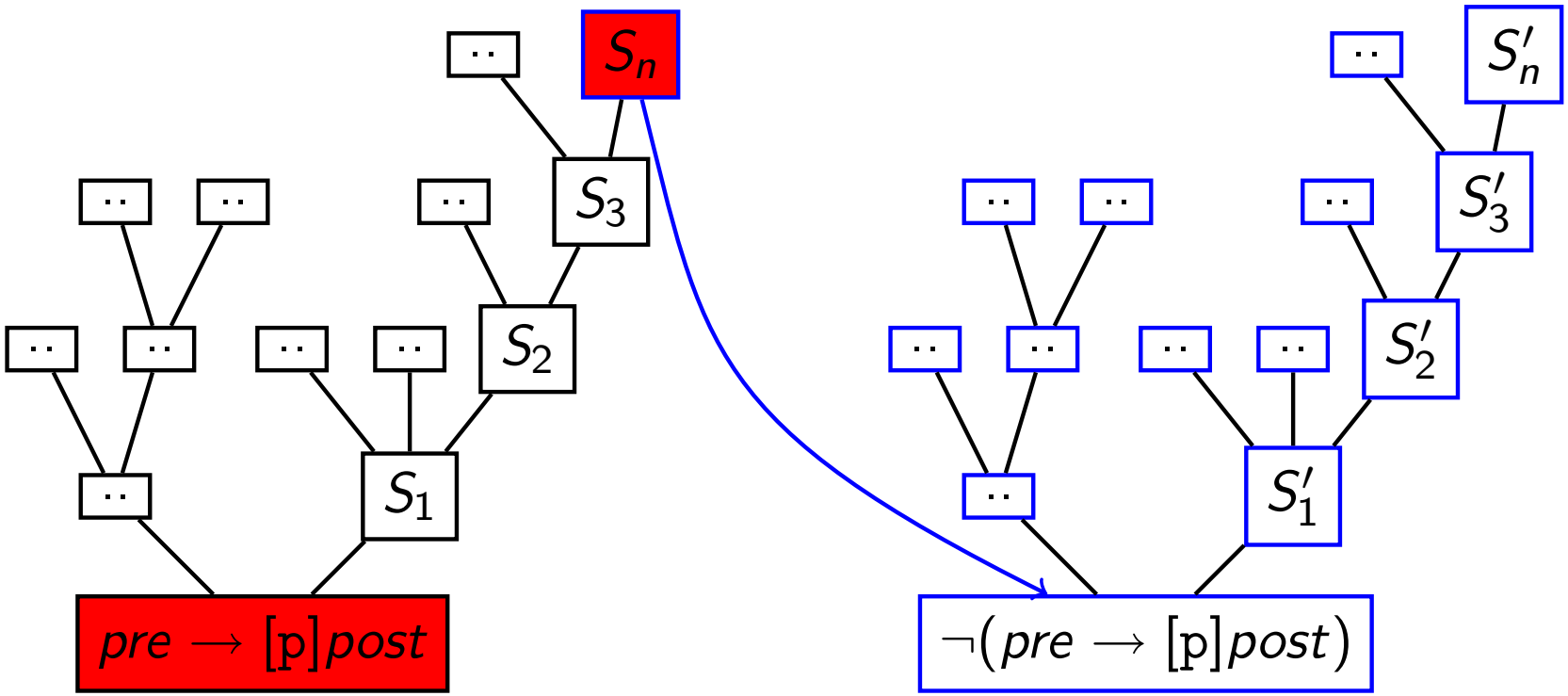
Special Falsifiability Preservation (Perform.-Boost)



Falsification Preservation (Approach 3)

$$\neg S_n \longrightarrow \neg(pre \longrightarrow [\mathrm{p}]post)$$

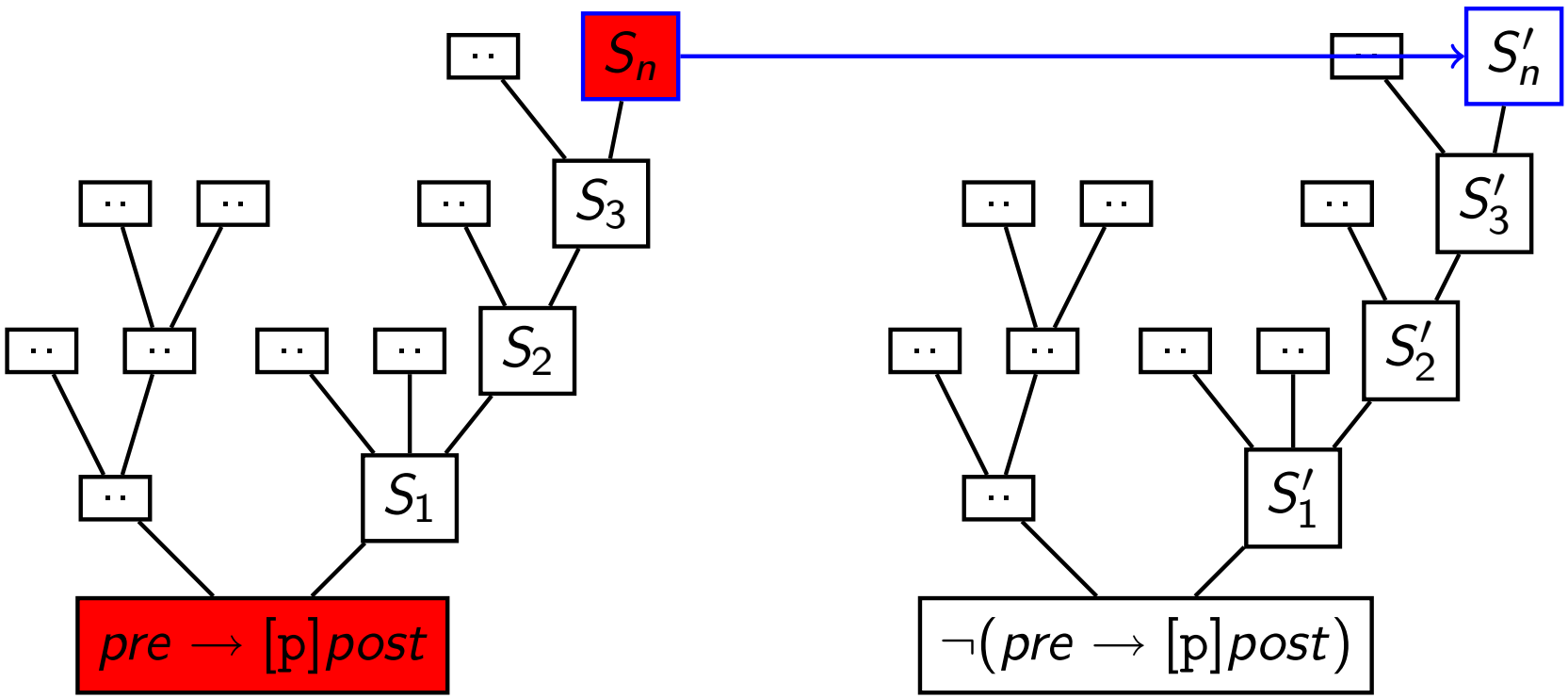
Special Falsifiability Preservation (Perform.-Boost)



Falsification Preservation (Approach 3)

$$\neg S_n \longrightarrow \neg(pre \longrightarrow [\mathrm{p}]post)$$

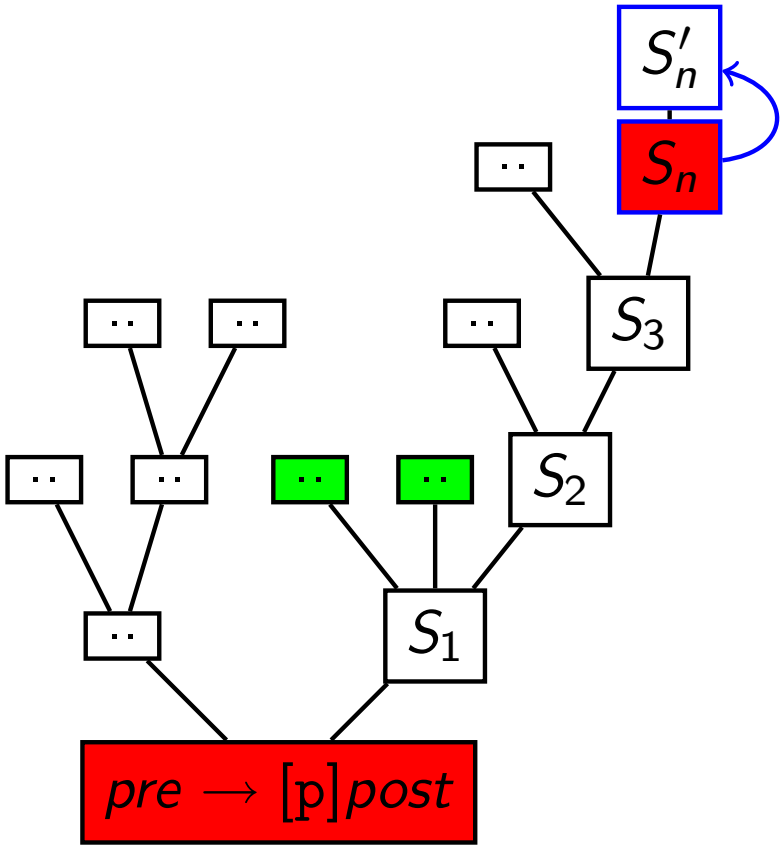
Special Falsifiability Preservation (Perform.-Boost)



Falsification Preservation (Approach $3\frac{1}{2}$)

$$\neg S_n \longrightarrow S'_n$$

Special Falsifiability Preservation (Perform.-Boost)



Special Falsifiability Preservation (Approach 4)

$$((\{M^1 := M^2\}S_n) \wedge \{U\}\{M^2\}post) \rightarrow S_n$$

Assembling the SFP Formula from the Proof Tree

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdots}\,*}{\Gamma \rightarrow I} \quad \dfrac{\dfrac{}{\vdots}\,*}{\Gamma \rightarrow \{U, M\}(I \wedge c \rightarrow [\mathtt{b}]I)} \quad \dfrac{\dfrac{}{\vdots}\,S_n}{\Gamma \rightarrow \{U, M\}(I \wedge \neg c \rightarrow [\mathtt{ret..}]\phi)}}{\Gamma \rightarrow \{U\}[\mathtt{while(c)\{b\};}][\mathtt{ret..}]\phi}}{\Gamma \rightarrow [\mathtt{i=0;}][\mathtt{while(c)\{b\};}][\mathtt{ret..}]\phi}}{\underbrace{\Gamma}_{pre} \rightarrow [\mathtt{sqrt();}]\underbrace{\phi}_{post}}$$

1. Try to verify program
2. Improve contract until 1st and 2nd branch of contract rule is proved
3. Check falsifiability of 3rd branch
4. prove **SFP**, "**S***pecial* **F***alsifiability* **P***reservation*"

$$((\{M^1 := M^2\}S_n) \wedge \{U\}\{M^2\}\underbrace{\phi}_{post}) \rightarrow S_n$$

Implies the existence of a program bug on execution path S_n!

Special Falsifiability Preservation Analysis

SFP: Special Falsification Preservation condition

The special falsifiability preservation condition $SFP_{S_i}^{S_n}$ is the conjunction of

$$((\{M^1 := M^2\}S_n) \wedge \{U\}\{M^2\}post) \rightarrow S_n \quad (1)$$

$$(\neg S_n \wedge \Gamma_i \wedge \neg\Delta_i) \rightarrow \{U\}\Psi \quad (2)$$

$$\neg S_n \rightarrow (\Gamma_i \wedge \neg\Delta_i) \quad (3)$$

where $\Psi = \langle \mathrm{p} \rangle true$ "or" $\Psi = true$, and some side conditions ...

Theorem

Under some side conditions (that are ensured by the iterative algorithm)

$$SFP_{S_i}^{S_n} \rightarrow FP_{S_i}^{S_n}$$

(Optional Remarks)

Proof confluence vs. FOL-confluence

Loop Invariant Rule $(I, I \wedge \neg c, M, []) \; M = mod(b)$

$$\frac{\begin{array}{lll} \mathbf{1:}\ \Gamma \Rightarrow \{U\}I, \Delta \\ \mathbf{2:}\ \Gamma \Rightarrow \{U\}\{M\}(I \wedge c \rightarrow [\mathtt{b}]I), \Delta \\ \mathbf{3:}\ \Gamma \Rightarrow \{U\}\{M\}((I \wedge \neg c) \rightarrow post), \Delta \end{array}}{\Gamma \Rightarrow \{U\}[\mathtt{while(c)\{b;\}}]post, \Delta}$$

(Optional Remarks)

Proof confluence vs. FOL-confluence

Loop Invariant Rule $(I, I \wedge \neg c, M, []) \; M = mod(b)$

$$\frac{\begin{array}{lll} \mathbf{1:}\ \Gamma \Rightarrow & \{U\}I, \Delta, \Phi \\ \mathbf{2:}\ \Gamma \Rightarrow & \{U\}\{M\}(I \wedge c \rightarrow [\mathtt{b}]I), \Delta, \Phi \\ \mathbf{3:}\ \Gamma \Rightarrow & \{U\}\{M\}((I \wedge \neg c) \rightarrow post), \Delta, \Phi \end{array}}{\Gamma \Rightarrow \underbrace{\{U\}[\mathtt{while(c)\{b;\}}]post}_{\Phi}, \Delta}$$

Conclusion

- **Contribution 1**: Falsifiability Preservation Analysis
- **Contribution 2**: Special Falsifiability Preservation Analysis
- It's **simple**, **fast**, **sound**, **complete***, and **symbolic**
- Hopefully pretty complete in practice when automated
- **Extension** for symbolic execution- or wp-based verification techniques. It adds **sound bug detection** and **disambiguation** of the reason for verification failure.

Conclusion

- **Contribution 1**: Falsifiability Preservation Analysis
- **Contribution 2**: Special Falsifiability Preservation Analysis
- It's **simple**, **fast**, **sound**, **complete***, and **symbolic**
- Hopefully pretty complete in practice when automated
- **Extension** for symbolic execution- or wp-based verification techniques. It adds **sound bug detection** and **disambiguation** of the reason for verification failure.

Conclusion

- **Contribution 1**: Falsifiability Preservation Analysis
- **Contribution 2**: Special Falsifiability Preservation Analysis
- It's **simple**, **fast**, **sound**, **complete***, and **symbolic**
- Hopefully pretty complete in practice when automated
- **Extension** for symbolic execution- or wp-based verification techniques. It adds **sound bug detection** and **disambiguation** of the reason for verification failure.

Conclusion

- **Contribution 1**: Falsifiability Preservation Analysis
- **Contribution 2**: Special Falsifiability Preservation Analysis
- It's **simple**, **fast**, **sound**, **complete***, and **symbolic**
- Hopefully pretty complete in practice when automated
- **Extension** for symbolic execution- or wp-based verification techniques. It adds **sound bug detection** and **disambiguation** of the reason for verification failure.

Conclusion

- **Contribution 1**: Falsifiability Preservation Analysis
- **Contribution 2**: Special Falsifiability Preservation Analysis
- It's **simple**, **fast**, **sound**, **complete***, and **symbolic**
- Hopefully pretty complete in practice when automated
- **Extension** for symbolic execution- or wp-based verification techniques. It adds **sound bug detection** and **disambiguation** of the reason for verification failure.