A study of the DASH algorithm for software property checking

Jacob Hougaard

20083206

May 30, 2014

Table of contents

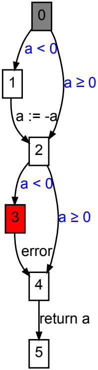
DASH$_{int}$ overview

```
int abs(int a)
{
  if(a < 0)
    a = -a;
  assert a ≥ 0;
  return a;
}
```

DASH$_{int}$ overview

```
int abs(int a)
{
  if(a < 0)
    a = -a;
  assert a ≥ 0;
  return a;
}
```

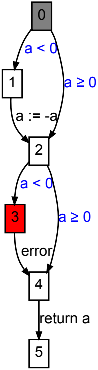
$\longrightarrow$



DASH$_{int}$ overview

```
int abs(int a)
{
  if(a < 0)
    a = -a;
  assert a ≥ 0;
  return a;
}
```

⟶

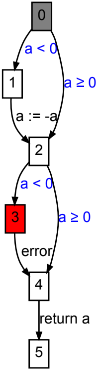


⟶ DASH$_{int}$

DASH$_{int}$ overview

```
int abs(int a)
{
  if(a < 0)
    a = -a;
  assert a >= 0;
  return a;
}
```

$\longrightarrow$



$\longrightarrow$ DASH$_{int}$

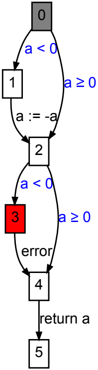
$\nearrow$ fail

$\searrow$ pass

DASH$_{int}$ overview

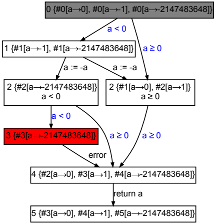
```
int abs(int a)
{
  if(a < 0)
    a = -a;
  assert a ≥ 0;
  return a;
}
```

⟶



⟶ DASH$_{int}$

↗ fail

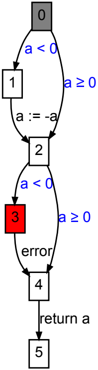


↘ pass

DASH$_{int}$ overview

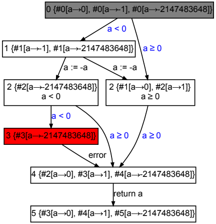
```
int abs(int a)
{
  if(a < 0)
    a = −a;
  assert a ≥ 0;
  return a;
}
```

⟶

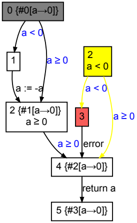


⟶ DASH$_{int}$

↗ fail



↘ pass

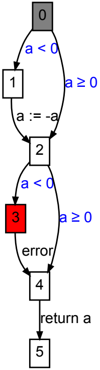


DashLoop - Test

DashLoop(P, $G = \langle \Sigma_{\simeq}, \rightarrow_{\simeq} \rangle$)

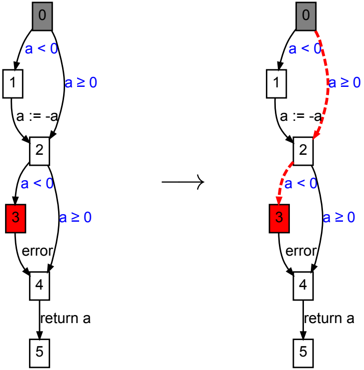
loop
  τ := FindAbstractErrorPath(G)
  if τ = NO-PATH **then**
    return (PASS, G)
  end if

  τ_c := ConvertToRegionTraceWithAbstractFrontier(τ, G)
  $\langle t, \rho \rangle$:= ExtendFrontier(τ_c, P)
  if $t \neq$ UNSAT **then**
    G := RunTest(t, P, G)
    if IsErrorRegionReached(G) **then**
      return (FAIL, t)
    end if
  else
    G := RefineGraph(ρ, τ_c, G)
  end if
end loop



DashLoop - Test

```
DashLoop(P, G = ⟨Σ≃, →≃⟩)
loop
    τ := FindAbstractErrorPath(G)
    if τ = NO-PATH then
        return (PASS, G)
    end if

    τc := ConvertToRegionTraceWithAbstractFrontier(τ, G)
    ⟨t, ρ⟩ := ExtendFrontier(τc, P)
    if t ≠ UNSAT then
        G := RunTest(t, P, G)
        if IsErrorRegionReached(G) then
            return (FAIL, t)
        end if
    else
        G := RefineGraph(ρ, τc, G)
    end if
end loop
```

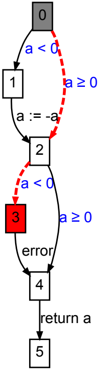


DashLoop - Test

DashLoop(P, $G = \langle \Sigma_{\simeq}, \rightarrow_{\simeq} \rangle$)

loop
  τ := FindAbstractErrorPath(G)
  if τ = NO-PATH **then**
    return (PASS, G)
  end if

  τ_c := ConvertToRegionTraceWithAbstractFrontier(τ, G)
  $\langle t, \rho \rangle$:= ExtendFrontier(τ_c, P)
  if $t \neq$ UNSAT **then**
    G := RunTest(t, P, G)
    if IsErrorRegionReached(G) **then**
      return (FAIL, t)
    end if
  else
    G := RefineGraph(ρ, τ_c, G)
  end if
end loop

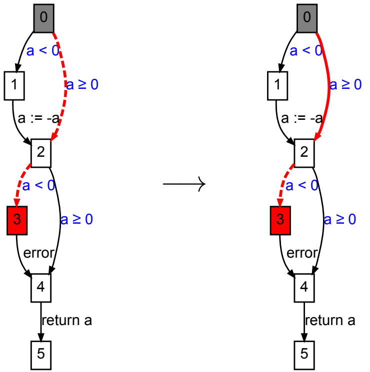


DashLoop - Test

DashLoop(P, $G = \langle \Sigma_{\simeq}, \rightarrow_{\simeq} \rangle$)

loop
    τ := FindAbstractErrorPath(G)
    if τ = NO-PATH **then**
        return (PASS, G)
    end if

    τ_c := ConvertToRegionTraceWithAbstractFrontier(τ, G)
    $\langle t, \rho \rangle$:= ExtendFrontier(τ_c, P)
    if $t \neq$ UNSAT **then**
        G := RunTest(t, P, G)
        if IsErrorRegionReached(G) **then**
            return (FAIL, t)
        end if
    else
        G := RefineGraph(ρ, τ_c, G)
    end if
end loop

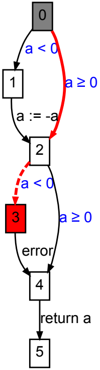


DashLoop - Test

DashLoop(P, $G = \langle \Sigma_{\simeq}, \rightarrow_{\simeq} \rangle$)

loop
    τ := FindAbstractErrorPath(G)
    if τ = NO-PATH **then**
        return (PASS, G)
    end if

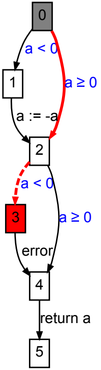
    τ_c := ConvertToRegionTraceWithAbstractFrontier(τ, G)
    $\langle t, \rho \rangle$:= ExtendFrontier(τ_c, P)
    if $t \neq$ UNSAT **then**
        G := RunTest(t, P, G)
        if IsErrorRegionReached(G) **then**
            return (FAIL, t)
        end if
    else
        G := RefineGraph(ρ, τ_c, G)
    end if
end loop



DashLoop - Test

DashLoop(P, $G = \langle \Sigma_{\simeq}, \rightarrow_{\simeq} \rangle$)

```
loop
    τ := FindAbstractErrorPath(G)
    if τ = NO-PATH then
        return (PASS, G)
    end if

    τ_c := ConvertToRegionTraceWithAbstractFrontier(τ, G)
    ⟨t, ρ⟩ := ExtendFrontier(τ_c, P)
    if t ≠ UNSAT then
        G := RunTest(t, P, G)
        if IsErrorRegionReached(G) then
            return (FAIL, t)
        end if
    else
        G := RefineGraph(ρ, τ_c, G)
    end if
end loop
```

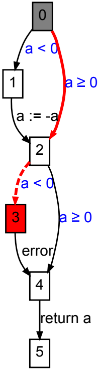


$$\longrightarrow \langle [a_0 \mapsto 0], \mathit{true} \rangle$$

DashLoop - Test

DashLoop(P, $G = \langle \Sigma_{\simeq}, \rightarrow_{\simeq} \rangle$)

```
loop
    τ := FindAbstractErrorPath(G)
    if τ = NO-PATH then
        return (PASS, G)
    end if

    τ_c := ConvertToRegionTraceWithAbstractFrontier(τ, G)
    ⟨t, ρ⟩ := ExtendFrontier(τ_c, P)
    if t ≠ UNSAT then
        G := RunTest(t, P, G)
        if IsErrorRegionReached(G) then
            return (FAIL, t)
        end if
    else
        G := RefineGraph(ρ, τ_c, G)
    end if
end loop
```



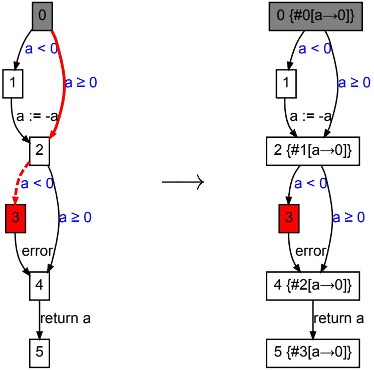
$\longrightarrow \langle [a_0 \mapsto 0], true \rangle$

DashLoop - Test

DashLoop(P, $G = \langle \Sigma_{\simeq}, \rightarrow_{\simeq} \rangle$)

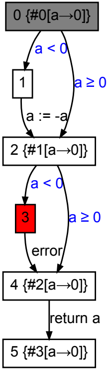
loop
    τ := FindAbstractErrorPath(G)
    if τ = NO-PATH **then**
        return (PASS, G)
    end if

    τ_c := ConvertToRegionTraceWithAbstractFrontier(τ, G)
    $\langle t, \rho \rangle$:= ExtendFrontier(τ_c, P)
    if $t \neq$ UNSAT **then**
        G := RunTest(t, P, G)
        if IsErrorRegionReached(G) **then**
            return (FAIL, t)
        end if
    else
        G := RefineGraph(ρ, τ_c, G)
    end if
end loop



DashLoop - Test

```
DashLoop(P, G = ⟨Σ≃, →≃⟩)
loop
    τ := FindAbstractErrorPath(G)
    if τ = NO-PATH then
        return (PASS, G)
    end if

    τ_c := ConvertToRegionTraceWithAbstractFrontier(τ, G)
    ⟨t, ρ⟩ := ExtendFrontier(τ_c, P)
    if t ≠ UNSAT then
        G := RunTest(t, P, G)
        if IsErrorRegionReached(G) then
            return (FAIL, t)
        end if
    else
        G := RefineGraph(ρ, τ_c, G)
    end if
end loop
```

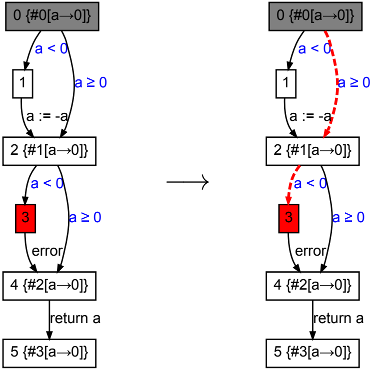


DashLoop - Refine

DashLoop(P, $G = \langle \Sigma_{\simeq}, \rightarrow_{\simeq} \rangle$)

loop
    $\tau :=$ FindAbstractErrorPath(G)
    if $\tau = \text{NO-PATH}$ **then**
        return (PASS, G)
    end if

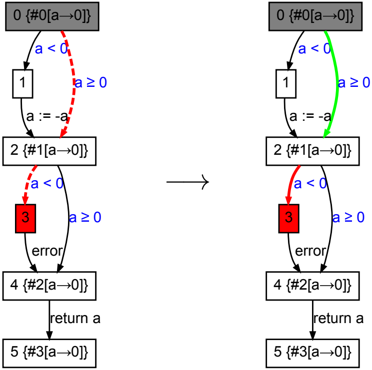
    $\tau_c :=$ ConvertToRegionTraceWithAbstractFrontier(τ, G)
    $\langle t, \rho \rangle :=$ ExtendFrontier(τ_c, P)
    if $t \neq \text{UNSAT}$ **then**
        $G :=$ RunTest(t, P, G)
        if IsErrorRegionReached(G) **then**
            return (FAIL, t)
        end if
    else
        $G :=$ RefineGraph(ρ, τ_c, G)
    end if
end loop



DashLoop - Refine

DashLoop(P, $G = \langle \Sigma_{\simeq}, \rightarrow_{\simeq} \rangle$)
loop
  τ := FindAbstractErrorPath(G)
  if τ = NO-PATH **then**
    return (PASS, G)
  end if

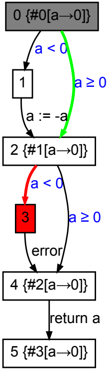
  τ_c := ConvertToRegionTraceWithAbstractFrontier(τ, G)
  $\langle t, \rho \rangle$:= ExtendFrontier(τ_c, P)
  if $t \neq$ UNSAT **then**
    G := RunTest(t, P, G)
    if IsErrorRegionReached(G) **then**
      return (FAIL, t)
    end if
  else
    G := RefineGraph(ρ, τ_c, G)
  end if
end loop



DashLoop - Refine

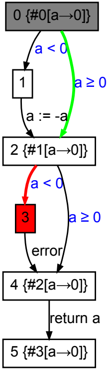
DashLoop(P, $G = \langle \Sigma_{\simeq}, \rightarrow_{\simeq} \rangle$)

```
loop
    τ := FindAbstractErrorPath(G)
    if τ = NO-PATH then
        return (PASS, G)
    end if

    τ_c := ConvertToRegionTraceWithAbstractFrontier(τ, G)
    ⟨t, ρ⟩ := ExtendFrontier(τ_c, P)
    if t ≠ UNSAT then
        G := RunTest(t, P, G)
        if IsErrorRegionReached(G) then
            return (FAIL, t)
        end if
    else
        G := RefineGraph(ρ, τ_c, G)
    end if
end loop
```



DashLoop - Refine

```
DashLoop(P, G = ⟨Σ≃, →≃⟩)
loop
    τ := FindAbstractErrorPath(G)
    if τ = NO-PATH then
        return (PASS, G)
    end if

    τc := ConvertToRegionTraceWithAbstractFrontier(τ, G)
    ⟨t, ρ⟩ := ExtendFrontier(τc, P)
    if t ≠ UNSAT then
        G := RunTest(t, P, G)
        if IsErrorRegionReached(G) then
            return (FAIL, t)
        end if
    else
        G := RefineGraph(ρ, τc, G)
    end if
end loop
```

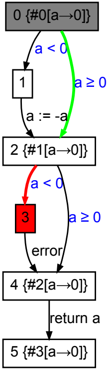


$$\longrightarrow \langle UNSAT, a < 0 \rangle$$

DashLoop - Refine

DashLoop(P, $G = \langle \Sigma_{\simeq}, \rightarrow_{\simeq} \rangle$)

```
loop
    τ := FindAbstractErrorPath(G)
    if τ = NO-PATH then
        return (PASS, G)
    end if

    τ_c := ConvertToRegionTraceWithAbstractFrontier(τ, G)
    ⟨t, ρ⟩ := ExtendFrontier(τ_c, P)
    if t ≠ UNSAT then
        G := RunTest(t, P, G)
        if IsErrorRegionReached(G) then
            return (FAIL, t)
        end if
    else
        G := RefineGraph(ρ, τ_c, G)
    end if
end loop
```



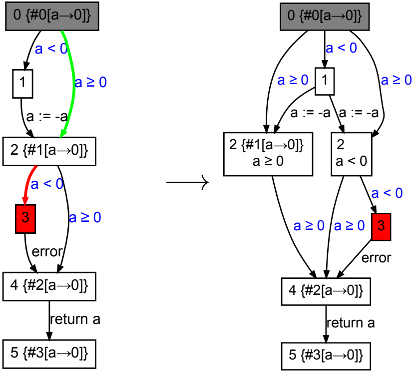
$$\longrightarrow \langle UNSAT, a < 0 \rangle$$

DashLoop - Refine

DashLoop(P, $G = \langle \Sigma_{\simeq}, \to_{\simeq} \rangle$)

loop
    τ := FindAbstractErrorPath(G)
    if τ = NO-PATH **then**
        return (PASS, G)
    end if

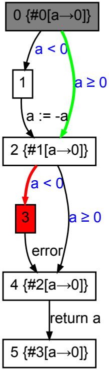
    τ_c := ConvertToRegionTraceWithAbstractFrontier(τ, G)
    $\langle t, \rho \rangle$:= ExtendFrontier(τ_c, P)
    if $t \neq$ UNSAT **then**
        G := RunTest(t, P, G)
        if IsErrorRegionReached(G) **then**
            return (FAIL, t)
        end if
    else
        G := RefineGraph(ρ, τ_c, G)
    end if
end loop



RefineGraph

RefineGraph$(\rho, \tau_c = \langle S_0, \ldots, S_{k-1}, S_k \rangle, G = \langle \Sigma_{\simeq}, \rightarrow_{\simeq} \rangle)$

let $\langle \rho_{k-1}, states \rangle = S_{k-1}$
if $k = 1$ **then**
  return $\langle \Sigma_{\simeq}, \rightarrow_{\simeq} \setminus (S_0, S_1) \rangle$
end if
$\Sigma^*_{\simeq} := \Sigma_{\simeq} \setminus \{S_{k-1}\}$
$\rightarrow^*_{\simeq} := \rightarrow_{\simeq} \setminus \{(S, S_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \setminus \{(S_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\rho^*_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \neg \rho)$
$S^*_{k-1} := \langle \rho^*_{k-1}, states \rangle$
$\Sigma^*_{\simeq} := \Sigma^*_{\simeq} \cup \{S^*_{k-1}\}$
$\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \cup \{(S, S^*_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \cup \{(S^*_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \setminus \{(S^*_{k-1}, S_k)\}$
$\rho^{**}_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \rho)$
$S^{**}_{k-1} := \langle \rho^{**}_{k-1}, \emptyset \rangle$
$\Sigma^*_{\simeq} := \Sigma^*_{\simeq} \cup \{S^{**}_{k-1}\}$
$\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \cup \{(S^{**}_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
if IsSAT$(\rho^{**}_{k-1}) \neq$ UNSAT **then**
  $\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \cup \{(S, S^{**}_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
end if
return $\langle \Sigma^*_{\simeq}, \rightarrow^*_{\simeq} \rangle$

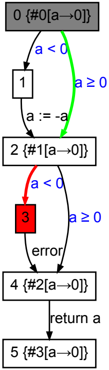


$\longrightarrow$
$\rho \mapsto a < 0$

RefineGraph

RefineGraph($\rho, \tau_c = \langle S_0, \ldots, S_{k-1}, S_k \rangle, G = \langle \Sigma_\simeq, \rightarrow_\simeq \rangle$)

let $\langle \rho_{k-1}, \mathit{states} \rangle = S_{k-1}$
if $k = 1$ **then**
  return $\langle \Sigma_\simeq, \rightarrow_\simeq \setminus (S_0, S_1) \rangle$
end if
$\Sigma^*_\simeq := \Sigma_\simeq \setminus \{S_{k-1}\}$
$\rightarrow^*_\simeq := \rightarrow_\simeq \setminus \{(S, S_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\rightarrow^*_\simeq := \rightarrow^*_\simeq \setminus \{(S_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\rho^*_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \neg\rho)$
$S^*_{k-1} := \langle \rho^*_{k-1}, \mathit{states} \rangle$
$\Sigma^*_\simeq := \Sigma^*_\simeq \cup \{S^*_{k-1}\}$
$\rightarrow^*_\simeq := \rightarrow^*_\simeq \cup \{(S, S^*_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\rightarrow^*_\simeq := \rightarrow^*_\simeq \cup \{(S^*_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\rightarrow^*_\simeq := \rightarrow^*_\simeq \setminus \{(S^*_{k-1}, S_k)\}$
$\rho^{**}_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \rho)$
$S^{**}_{k-1} := \langle \rho^{**}_{k-1}, \emptyset \rangle$
$\Sigma^*_\simeq := \Sigma^*_\simeq \cup \{S^{**}_{k-1}\}$
$\rightarrow^*_\simeq := \rightarrow^*_\simeq \cup \{(S^{**}_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
if IsSAT(ρ^{**}_{k-1}) $\neq$ UNSAT **then**
  $\rightarrow^*_\simeq := \rightarrow^*_\simeq \cup \{(S, S^{**}_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
end if
return $\langle \Sigma^*_\simeq, \rightarrow^*_\simeq \rangle$

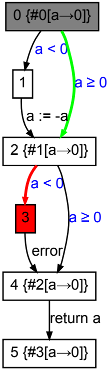


$\longrightarrow$
$\rho \mapsto a < 0$

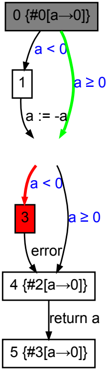
RefineGraph

RefineGraph($\rho, \tau_c = \langle S_0, \ldots, S_{k-1}, S_k \rangle, G = \langle \Sigma_{\simeq}, \rightarrow_{\simeq} \rangle$)

let $\langle \rho_{k-1}, states \rangle = S_{k-1}$
if $k = 1$ **then**
    return $\langle \Sigma_{\simeq}, \rightarrow_{\simeq} \setminus (S_0, S_1) \rangle$
end if
$\Sigma^*_{\simeq} := \Sigma_{\simeq} \setminus \{S_{k-1}\}$
$\rightarrow^*_{\simeq} := \rightarrow_{\simeq} \setminus \{(S, S_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \setminus \{(S_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\rho^*_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \neg\rho)$
$S^*_{k-1} := \langle \rho^*_{k-1}, states \rangle$
$\Sigma^*_{\simeq} := \Sigma^*_{\simeq} \cup \{S^*_{k-1}\}$
$\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \cup \{(S, S^*_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \cup \{(S^*_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \setminus \{(S^*_{k-1}, S_k)\}$
$\rho^{**}_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \rho)$
$S^{**}_{k-1} := \langle \rho^{**}_{k-1}, \emptyset \rangle$
$\Sigma^*_{\simeq} := \Sigma^*_{\simeq} \cup \{S^{**}_{k-1}\}$
$\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \cup \{(S^{**}_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
if IsSAT(ρ^{**}_{k-1}) $\neq$ UNSAT **then**
    $\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \cup \{(S, S^{**}_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
end if
return $\langle \Sigma^*_{\simeq}, \rightarrow^*_{\simeq} \rangle$



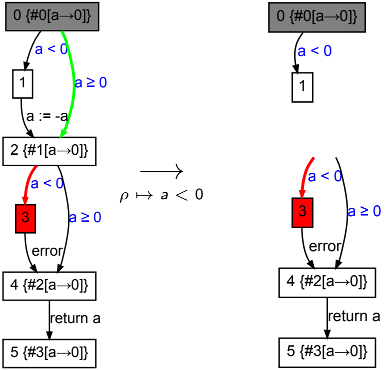
$\longrightarrow$ $\rho \mapsto a < 0$



RefineGraph

RefineGraph$(\rho, \tau_c = \langle S_0, \ldots, S_{k-1}, S_k\rangle, G = \langle \Sigma_\simeq, \to_\simeq\rangle)$

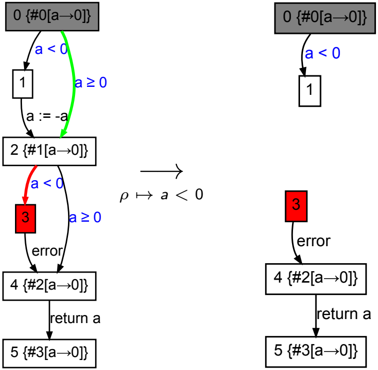
let $\langle \rho_{k-1}, states\rangle = S_{k-1}$
if $k = 1$ **then**
    return $\langle \Sigma_\simeq, \to_\simeq \setminus (S_0, S_1)\rangle$
end if
$\Sigma^*_\simeq := \Sigma_\simeq \setminus \{S_{k-1}\}$
$\to^*_\simeq := \to_\simeq \setminus \{(S, S_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\to^*_\simeq := \to^*_\simeq \setminus \{(S_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\rho^*_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \neg\rho)$
$S^*_{k-1} := \langle \rho^*_{k-1}, states\rangle$
$\Sigma^*_\simeq := \Sigma^*_\simeq \cup \{S^*_{k-1}\}$
$\to^*_\simeq := \to^*_\simeq \cup \{(S, S^*_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\to^*_\simeq := \to^*_\simeq \cup \{(S^*_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\to^*_\simeq := \to^*_\simeq \setminus \{(S^*_{k-1}, S_k)\}$
$\rho^{**}_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \rho)$
$S^{**}_{k-1} := \langle \rho^{**}_{k-1}, \emptyset\rangle$
$\Sigma^*_\simeq := \Sigma^*_\simeq \cup \{S^{**}_{k-1}\}$
$\to^*_\simeq := \to^*_\simeq \cup \{(S^{**}_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
if IsSAT$(\rho^{**}_{k-1}) \neq$ UNSAT **then**
    $\to^*_\simeq := \to^*_\simeq \cup \{(S, S^{**}_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
end if
return $\langle \Sigma^*_\simeq, \to^*_\simeq\rangle$



RefineGraph

RefineGraph$(\rho, \tau_c = \langle S_0, \ldots, S_{k-1}, S_k \rangle, G = \langle \Sigma_{\simeq}, \rightarrow_{\simeq} \rangle)$

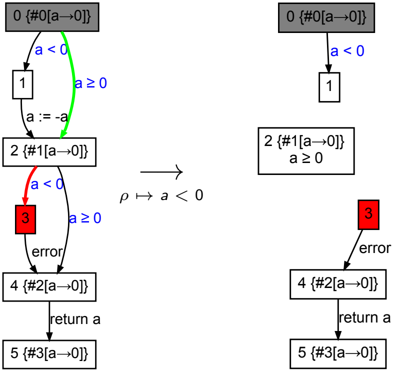
let $\langle \rho_{k-1}, states \rangle = S_{k-1}$
if $k = 1$ **then**
  return $\langle \Sigma_{\simeq}, \rightarrow_{\simeq} \setminus (S_0, S_1) \rangle$
end if
$\Sigma^*_{\simeq} := \Sigma_{\simeq} \setminus \{S_{k-1}\}$
$\rightarrow^*_{\simeq} := \rightarrow_{\simeq} \setminus \{(S, S_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \setminus \{(S_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\rho^*_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \neg\rho)$
$S^*_{k-1} := \langle \rho^*_{k-1}, states \rangle$
$\Sigma^*_{\simeq} := \Sigma^*_{\simeq} \cup \{S^*_{k-1}\}$
$\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \cup \{(S, S^*_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \cup \{(S^*_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \setminus \{(S^*_{k-1}, S_k)\}$
$\rho^{**}_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \rho)$
$S^{**}_{k-1} := \langle \rho^{**}_{k-1}, \emptyset \rangle$
$\Sigma^*_{\simeq} := \Sigma^*_{\simeq} \cup \{S^{**}_{k-1}\}$
$\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \cup \{(S^{**}_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
if IsSAT$(\rho^{**}_{k-1}) \neq$ UNSAT **then**
  $\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \cup \{(S, S^{**}_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
end if
return $\langle \Sigma^*_{\simeq}, \rightarrow^*_{\simeq} \rangle$



RefineGraph

RefineGraph($\rho, \tau_c = \langle S_0, \ldots, S_{k-1}, S_k \rangle, G = \langle \Sigma_{\simeq}, \rightarrow_{\simeq} \rangle$)

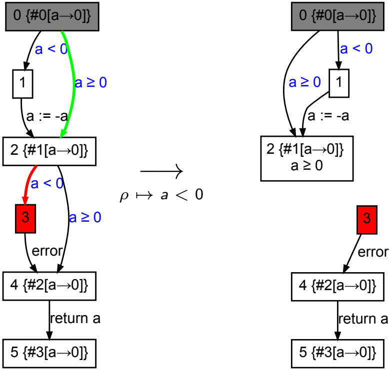
let $\langle \rho_{k-1}, states \rangle = S_{k-1}$
if $k = 1$ **then**
  return $\langle \Sigma_{\simeq}, \rightarrow_{\simeq} \setminus (S_0, S_1) \rangle$
end if
$\Sigma^*_{\simeq} := \Sigma_{\simeq} \setminus \{S_{k-1}\}$
$\rightarrow^*_{\simeq} := \rightarrow_{\simeq} \setminus \{(S, S_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \setminus \{(S_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\rho^*_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \neg\rho)$
$S^*_{k-1} := \langle \rho^*_{k-1}, states \rangle$
$\Sigma^*_{\simeq} := \Sigma^*_{\simeq} \cup \{S^*_{k-1}\}$
$\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \cup \{(S, S^*_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \cup \{(S^*_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \setminus \{(S^*_{k-1}, S_k)\}$
$\rho^{**}_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \rho)$
$S^{**}_{k-1} := \langle \rho^{**}_{k-1}, \emptyset \rangle$
$\Sigma^*_{\simeq} := \Sigma^*_{\simeq} \cup \{S^{**}_{k-1}\}$
$\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \cup \{(S^{**}_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
if IsSAT(ρ^{**}_{k-1}) $\neq$ UNSAT **then**
  $\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \cup \{(S, S^{**}_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
end if
return $\langle \Sigma^*_{\simeq}, \rightarrow^*_{\simeq} \rangle$



RefineGraph

RefineGraph$(\rho, \tau_c = \langle S_0, \ldots, S_{k-1}, S_k \rangle, G = \langle \Sigma_{\simeq}, \to_{\simeq} \rangle)$

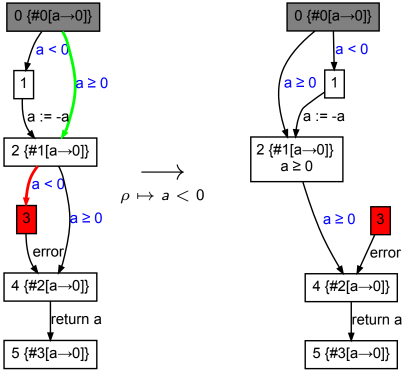
let $\langle \rho_{k-1}, states \rangle = S_{k-1}$
if $k = 1$ **then**
  return $\langle \Sigma_{\simeq}, \to_{\simeq} \setminus (S_0, S_1) \rangle$
end if
$\Sigma^*_{\simeq} := \Sigma_{\simeq} \setminus \{S_{k-1}\}$
$\to^*_{\simeq} := \to_{\simeq} \setminus \{(S, S_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\to^*_{\simeq} := \to^*_{\simeq} \setminus \{(S_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\rho^*_{k-1} := \text{Simplify}(\rho_{k-1} \land \neg \rho)$
$S^*_{k-1} := \langle \rho^*_{k-1}, states \rangle$
$\Sigma^*_{\simeq} := \Sigma^*_{\simeq} \cup \{S^*_{k-1}\}$
$\to^*_{\simeq} := \to^*_{\simeq} \cup \{(S, S^*_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\to^*_{\simeq} := \to^*_{\simeq} \cup \{(S^*_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\to^*_{\simeq} := \to^*_{\simeq} \setminus \{(S^*_{k-1}, S_k)\}$
$\rho^{**}_{k-1} := \text{Simplify}(\rho_{k-1} \land \rho)$
$S^{**}_{k-1} := \langle \rho^{**}_{k-1}, \emptyset \rangle$
$\Sigma^*_{\simeq} := \Sigma^*_{\simeq} \cup \{S^{**}_{k-1}\}$
$\to^*_{\simeq} := \to^*_{\simeq} \cup \{(S^{**}_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
if IsSAT$(\rho^{**}_{k-1}) \neq$ UNSAT **then**
  $\to^*_{\simeq} := \to^*_{\simeq} \cup \{(S, S^{**}_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
end if
return $\langle \Sigma^*_{\simeq}, \to^*_{\simeq} \rangle$



RefineGraph

RefineGraph$(\rho, \tau_c = \langle S_0, \ldots, S_{k-1}, S_k \rangle, G = \langle \Sigma_{\simeq}, \to_{\simeq} \rangle)$

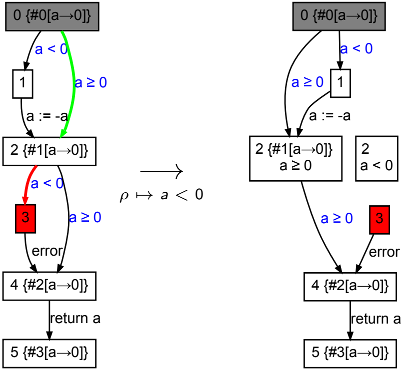
let $\langle \rho_{k-1}, states \rangle = S_{k-1}$
if $k = 1$ **then**
  return $\langle \Sigma_{\simeq}, \to_{\simeq} \setminus (S_0, S_1) \rangle$
end if
$\Sigma^*_{\simeq} := \Sigma_{\simeq} \setminus \{S_{k-1}\}$
$\to^*_{\simeq} := \to_{\simeq} \setminus \{(S, S_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\to^*_{\simeq} := \to^*_{\simeq} \setminus \{(S_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\rho^*_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \neg\rho)$
$S^*_{k-1} := \langle \rho^*_{k-1}, states \rangle$
$\Sigma^*_{\simeq} := \Sigma^*_{\simeq} \cup \{S^*_{k-1}\}$
$\to^*_{\simeq} := \to^*_{\simeq} \cup \{(S, S^*_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\to^*_{\simeq} := \to^*_{\simeq} \cup \{(S^*_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\to^*_{\simeq} := \to^*_{\simeq} \setminus \{(S^*_{k-1}, S_k)\}$
$\rho^{**}_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \rho)$
$S^{**}_{k-1} := \langle \rho^{**}_{k-1}, \emptyset \rangle$
$\Sigma^*_{\simeq} := \Sigma^*_{\simeq} \cup \{S^{**}_{k-1}\}$
$\to^*_{\simeq} := \to^*_{\simeq} \cup \{(S^{**}_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
if IsSAT$(\rho^{**}_{k-1}) \neq$ UNSAT **then**
  $\to^*_{\simeq} := \to^*_{\simeq} \cup \{(S, S^{**}_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
end if
return $\langle \Sigma^*_{\simeq}, \to^*_{\simeq} \rangle$



RefineGraph

RefineGraph$(\rho, \tau_c = \langle S_0, \ldots, S_{k-1}, S_k \rangle, G = \langle \Sigma_{\simeq}, \rightarrow_{\simeq} \rangle)$

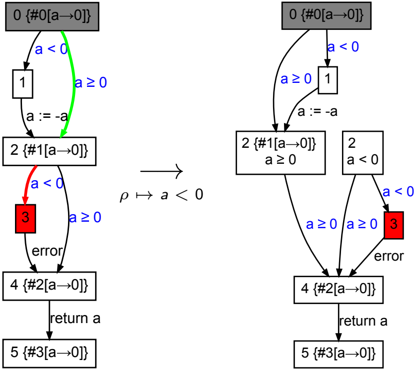
let $\langle \rho_{k-1}, states \rangle = S_{k-1}$
if $k = 1$ **then**
  return $\langle \Sigma_{\simeq}, \rightarrow_{\simeq} \setminus (S_0, S_1) \rangle$
end if
$\Sigma^*_{\simeq} := \Sigma_{\simeq} \setminus \{S_{k-1}\}$
$\rightarrow^*_{\simeq} := \rightarrow_{\simeq} \setminus \{(S, S_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \setminus \{(S_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\rho^*_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \neg\rho)$
$S^*_{k-1} := \langle \rho^*_{k-1}, states \rangle$
$\Sigma^*_{\simeq} := \Sigma^*_{\simeq} \cup \{S^*_{k-1}\}$
$\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \cup \{(S, S^*_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \cup \{(S^*_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \setminus \{(S^*_{k-1}, S_k)\}$
$\rho^{**}_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \rho)$
$S^{**}_{k-1} := \langle \rho^{**}_{k-1}, \emptyset \rangle$
$\Sigma^*_{\simeq} := \Sigma^*_{\simeq} \cup \{S^{**}_{k-1}\}$
$\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \cup \{(S^{**}_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
if IsSAT$(\rho^{**}_{k-1}) \neq$ UNSAT **then**
  $\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \cup \{(S, S^{**}_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
end if
return $\langle \Sigma^*_{\simeq}, \rightarrow^*_{\simeq} \rangle$



RefineGraph

RefineGraph$(\rho, \tau_c = \langle S_0, \ldots, S_{k-1}, S_k \rangle, G = \langle \Sigma_\simeq, \rightarrow_\simeq \rangle)$

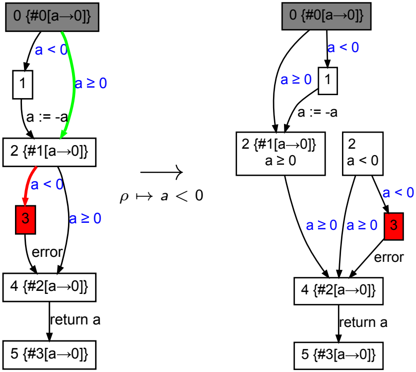
let $\langle \rho_{k-1}, states \rangle = S_{k-1}$
if $k = 1$ **then**
  return $\langle \Sigma_\simeq, \rightarrow_\simeq \setminus (S_0, S_1) \rangle$
end if
$\Sigma^*_\simeq := \Sigma_\simeq \setminus \{S_{k-1}\}$
$\rightarrow^*_\simeq := \rightarrow_\simeq \setminus \{(S, S_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\rightarrow^*_\simeq := \rightarrow^*_\simeq \setminus \{(S_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\rho^*_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \neg\rho)$
$S^*_{k-1} := \langle \rho^*_{k-1}, states \rangle$
$\Sigma^*_\simeq := \Sigma^*_\simeq \cup \{S^*_{k-1}\}$
$\rightarrow^*_\simeq := \rightarrow^*_\simeq \cup \{(S, S^*_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\rightarrow^*_\simeq := \rightarrow^*_\simeq \cup \{(S^*_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\rightarrow^*_\simeq := \rightarrow^*_\simeq \setminus \{(S^*_{k-1}, S_k)\}$
$\rho^{**}_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \rho)$
$S^{**}_{k-1} := \langle \rho^{**}_{k-1}, \emptyset \rangle$
$\Sigma^*_\simeq := \Sigma^*_\simeq \cup \{S^{**}_{k-1}\}$
$\rightarrow^*_\simeq := \rightarrow^*_\simeq \cup \{(S^{**}_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
if IsSAT$(\rho^{**}_{k-1}) \neq$ UNSAT **then**
  $\rightarrow^*_\simeq := \rightarrow^*_\simeq \cup \{(S, S^{**}_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
end if
return $\langle \Sigma^*_\simeq, \rightarrow^*_\simeq \rangle$



RefineGraph

RefineGraph$(\rho, \tau_c = \langle S_0, \ldots, S_{k-1}, S_k \rangle, G = \langle \Sigma_\simeq, \rightarrow_\simeq \rangle)$

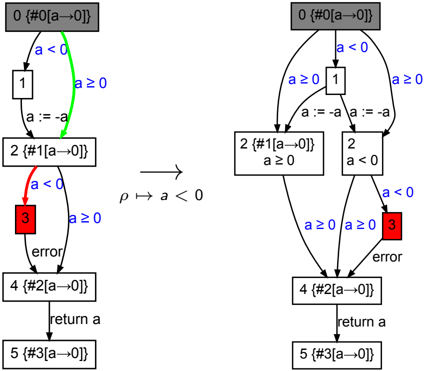
let $\langle \rho_{k-1}, states \rangle = S_{k-1}$
if $k = 1$ **then**
  return $\langle \Sigma_\simeq, \rightarrow_\simeq \setminus (S_0, S_1) \rangle$
end if
$\Sigma_\simeq^* := \Sigma_\simeq \setminus \{S_{k-1}\}$
$\rightarrow_\simeq^* := \rightarrow_\simeq \setminus \{(S, S_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\rightarrow_\simeq^* := \rightarrow_\simeq^* \setminus \{(S_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\rho_{k-1}^* := \text{Simplify}(\rho_{k-1} \wedge \neg\rho)$
$S_{k-1}^* := \langle \rho_{k-1}^*, states \rangle$
$\Sigma_\simeq^* := \Sigma_\simeq^* \cup \{S_{k-1}^*\}$
$\rightarrow_\simeq^* := \rightarrow_\simeq^* \cup \{(S, S_{k-1}^*) \mid S \in \text{Parents}(S_{k-1})\}$
$\rightarrow_\simeq^* := \rightarrow_\simeq^* \cup \{(S_{k-1}^*, S) \mid S \in \text{Children}(S_{k-1})\}$
$\rightarrow_\simeq^* := \rightarrow_\simeq^* \setminus \{(S_{k-1}^*, S_k)\}$
$\rho_{k-1}^{**} := \text{Simplify}(\rho_{k-1} \wedge \rho)$
$S_{k-1}^{**} := \langle \rho_{k-1}^{**}, \emptyset \rangle$
$\Sigma_\simeq^* := \Sigma_\simeq^* \cup \{S_{k-1}^{**}\}$
$\rightarrow_\simeq^* := \rightarrow_\simeq^* \cup \{(S_{k-1}^{**}, S) \mid S \in \text{Children}(S_{k-1})\}$
if $\text{IsSAT}(\rho_{k-1}^{**}) \neq \text{UNSAT}$ **then**
  $\rightarrow_\simeq^* := \rightarrow_\simeq^* \cup \{(S, S_{k-1}^{**}) \mid S \in \text{Parents}(S_{k-1})\}$
end if
return $\langle \Sigma_\simeq^*, \rightarrow_\simeq^* \rangle$



RefineGraph

RefineGraph($\rho, \tau_c = \langle S_0, \ldots, S_{k-1}, S_k \rangle, G = \langle \Sigma_{\simeq}, \to_{\simeq} \rangle$)

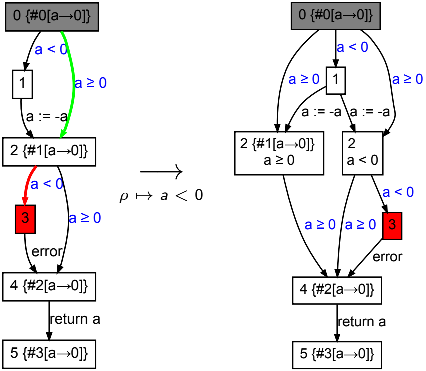
let $\langle \rho_{k-1}, states \rangle = S_{k-1}$
if $k = 1$ **then**
  return $\langle \Sigma_{\simeq}, \to_{\simeq} \setminus (S_0, S_1) \rangle$
end if
$\Sigma^*_{\simeq} := \Sigma_{\simeq} \setminus \{S_{k-1}\}$
$\to^*_{\simeq} := \to_{\simeq} \setminus \{(S, S_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\to^*_{\simeq} := \to^*_{\simeq} \setminus \{(S_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\rho^*_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \neg\rho)$
$S^*_{k-1} := \langle \rho^*_{k-1}, states \rangle$
$\Sigma^*_{\simeq} := \Sigma^*_{\simeq} \cup \{S^*_{k-1}\}$
$\to^*_{\simeq} := \to^*_{\simeq} \cup \{(S, S^*_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\to^*_{\simeq} := \to^*_{\simeq} \cup \{(S^*_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\to^*_{\simeq} := \to^*_{\simeq} \setminus \{(S^*_{k-1}, S_k)\}$
$\rho^{**}_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \rho)$
$S^{**}_{k-1} := \langle \rho^{**}_{k-1}, \emptyset \rangle$
$\Sigma^*_{\simeq} := \Sigma^*_{\simeq} \cup \{S^{**}_{k-1}\}$
$\to^*_{\simeq} := \to^*_{\simeq} \cup \{(S^{**}_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
if IsSAT(ρ^{**}_{k-1}) $\neq$ UNSAT **then**
  $\to^*_{\simeq} := \to^*_{\simeq} \cup \{(S, S^{**}_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
end if
return $\langle \Sigma^*_{\simeq}, \to^*_{\simeq} \rangle$



RefineGraph

RefineGraph($\rho, \tau_c = \langle S_0, \ldots, S_{k-1}, S_k \rangle, G = \langle \Sigma_\simeq, \to_\simeq \rangle$)

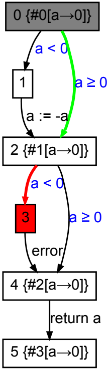
let $\langle \rho_{k-1}, states \rangle = S_{k-1}$
if $k = 1$ **then**
  return $\langle \Sigma_\simeq, \to_\simeq \setminus (S_0, S_1) \rangle$
end if
$\Sigma^*_\simeq := \Sigma_\simeq \setminus \{S_{k-1}\}$
$\to^*_\simeq := \to_\simeq \setminus \{(S, S_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\to^*_\simeq := \to^*_\simeq \setminus \{(S_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\rho^*_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \neg\rho)$
$S^*_{k-1} := \langle \rho^*_{k-1}, states \rangle$
$\Sigma^*_\simeq := \Sigma^*_\simeq \cup \{S^*_{k-1}\}$
$\to^*_\simeq := \to^*_\simeq \cup \{(S, S^*_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\to^*_\simeq := \to^*_\simeq \cup \{(S^*_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\to^*_\simeq := \to^*_\simeq \setminus \{(S^*_{k-1}, S_k)\}$
$\rho^{**}_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \rho)$
$S^{**}_{k-1} := \langle \rho^{**}_{k-1}, \emptyset \rangle$
$\Sigma^*_\simeq := \Sigma^*_\simeq \cup \{S^{**}_{k-1}\}$
$\to^*_\simeq := \to^*_\simeq \cup \{(S^{**}_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
if IsSAT(ρ^{**}_{k-1}) $\neq$ UNSAT **then**
  $\to^*_\simeq := \to^*_\simeq \cup \{(S, S^{**}_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
end if
return $\langle \Sigma^*_\simeq, \to^*_\simeq \rangle$



ExtendFrontier - Refine

ExtendFrontier(τ_c, P)

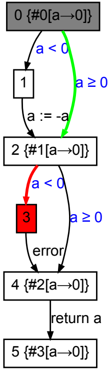
```
φ := ExecuteSymbolic(τc, P)
t := IsSAT(φ, P)
if t = UNSAT then
    ρ := RefinePred(τc)
else
    ρ := true
end if
return ⟨t, ρ⟩
```



ExtendFrontier - Refine

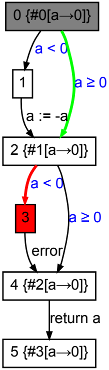
ExtendFrontier(τ_c, P)

ϕ := ExecuteSymbolic(τ_c, P)
t := IsSAT(ϕ, P)
if t = UNSAT **then**
    ρ := RefinePred(τ_c)
else
    ρ := *true*
end if
return $\langle t, \rho \rangle$



ExecuteSymbolic

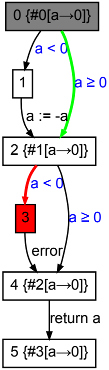
ExecuteSymbolic($\tau_c = \langle S_0, \ldots, S_k \rangle, P$)
let $\langle \rho_0, \_ \rangle = S_0$
$\mathcal{S} := [v \mapsto v_0 \mid v \in \text{params}(P)]$
$\phi := \text{SymbolicEval}(\rho_0, \mathcal{S})$
for $i = 0$ **to** $k - 1$ **do**
    $op := \text{Op}(S_i, S_{i+1})$
    match op
        case($v := e$):
            $\mathcal{S} := \mathcal{S}[v \mapsto \text{SymbolicEval}(e, \mathcal{S})]$
        case(assume c):
            $\phi := \phi \wedge \text{SymbolicEval}(c, \mathcal{S})$
    let $\langle \rho_{i+1}, \_ \rangle = S_{i+1}$
    $\phi := \phi \wedge \text{SymbolicEval}(\rho_{i+1}, \mathcal{S})$
end for
return ϕ



ExecuteSymbolic

ExecuteSymbolic($\tau_c = \langle S_0, \ldots, S_k \rangle, P$)

let $\langle \rho_0, \_ \rangle = S_0$
$\mathcal{S} := [v \mapsto v_0 \mid v \in \text{params}(P)]$
$\phi := \text{SymbolicEval}(\rho_0, \mathcal{S})$
for $i = 0$ **to** $k - 1$ **do**
  $op := \text{Op}(S_i, S_{i+1})$
  match op
    case($v := e$):
      $\mathcal{S} := \mathcal{S}[v \mapsto \text{SymbolicEval}(e, \mathcal{S})]$
    case(assume c):
      $\phi := \phi \wedge \text{SymbolicEval}(c, \mathcal{S})$
  let $\langle \rho_{i+1}, \_ \rangle = S_{i+1}$
  $\phi := \phi \wedge \text{SymbolicEval}(\rho_{i+1}, \mathcal{S})$
end for
return ϕ



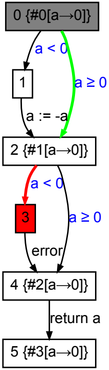
$\longrightarrow$

$$\mathcal{S} \rightarrow [a \mapsto a_0]$$
$$\phi \rightarrow true$$

ExecuteSymbolic

ExecuteSymbolic($\tau_c = \langle S_0, \ldots, S_k \rangle, P$)

let $\langle \rho_0, \_ \rangle = S_0$
$\mathcal{S} := [v \mapsto v_0 \mid v \in \text{params}(P)]$
$\phi := \text{SymbolicEval}(\rho_0, \mathcal{S})$
for $i = 0$ **to** $k - 1$ **do**
    $op := \text{Op}(S_i, S_{i+1})$
    match op
        case($v := e$):
            $\mathcal{S} := \mathcal{S}[v \mapsto \text{SymbolicEval}(e, \mathcal{S})]$
        case(assume c):
            $\phi := \phi \wedge \text{SymbolicEval}(c, \mathcal{S})$
    let $\langle \rho_{i+1}, \_ \rangle = S_{i+1}$
    $\phi := \phi \wedge \text{SymbolicEval}(\rho_{i+1}, \mathcal{S})$
end for
return ϕ



$\overrightarrow{op = a \geq 0}$

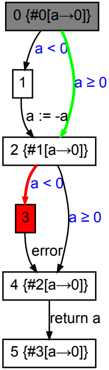
$\mathcal{S} \rightarrow [a \mapsto a_0]$

$\phi \rightarrow true$

ExecuteSymbolic

ExecuteSymbolic($\tau_c = \langle S_0, \ldots, S_k \rangle, P$)

let $\langle \rho_0, \_ \rangle = S_0$
$\mathcal{S} := [v \mapsto v_0 \mid v \in \text{params}(P)]$
$\phi := \text{SymbolicEval}(\rho_0, \mathcal{S})$
for $i = 0$ **to** $k - 1$ **do**
    $op := \text{Op}(S_i, S_{i+1})$
    match op
        case($v := e$):
            $\mathcal{S} := \mathcal{S}[v \mapsto \text{SymbolicEval}(e, \mathcal{S})]$
        case(assume c):
            $\phi := \phi \wedge \text{SymbolicEval}(c, \mathcal{S})$
    let $\langle \rho_{i+1}, \_ \rangle = S_{i+1}$
    $\phi := \phi \wedge \text{SymbolicEval}(\rho_{i+1}, \mathcal{S})$
end for
return ϕ



$\xrightarrow{}$
$op = a \geq 0$

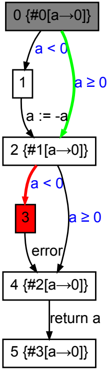
$\mathcal{S} \rightarrow [a \mapsto a_0]$

$\phi \rightarrow a_0 \geq 0$

ExecuteSymbolic

ExecuteSymbolic($\tau_c = \langle S_0, \ldots, S_k \rangle, P$)

let $\langle \rho_0, \_ \rangle = S_0$
$\mathcal{S} := [v \mapsto v_0 \mid v \in \text{params}(P)]$
$\phi := \text{SymbolicEval}(\rho_0, \mathcal{S})$
for $i = 0$ **to** $k - 1$ **do**
  $op := \text{Op}(S_i, S_{i+1})$
  match op
    case($v := e$):
      $\mathcal{S} := \mathcal{S}[v \mapsto \text{SymbolicEval}(e, \mathcal{S})]$
    case(assume c):
      $\phi := \phi \wedge \text{SymbolicEval}(c, \mathcal{S})$
  let $\langle \rho_{i+1}, \_ \rangle = S_{i+1}$
  $\phi := \phi \wedge \text{SymbolicEval}(\rho_{i+1}, \mathcal{S})$
end for
return ϕ



$$\xrightarrow[\rho_{i+1} = true]{}$$

$$\mathcal{S} \rightarrow [a \mapsto a_0]$$
$$\phi \rightarrow a_0 \geq 0$$

ExecuteSymbolic

ExecuteSymbolic($\tau_c = \langle S_0, \ldots, S_k \rangle, P$)

let $\langle \rho_0, \_ \rangle = S_0$

$\mathcal{S} := [v \mapsto v_0 \mid v \in \text{params}(P)]$

$\phi := \text{SymbolicEval}(\rho_0, \mathcal{S})$

for $i = 0$ **to** $k - 1$ **do**

  $op := \text{Op}(S_i, S_{i+1})$

  match op

    case($v := e$):

      $\mathcal{S} := \mathcal{S}[v \mapsto \text{SymbolicEval}(e, \mathcal{S})]$

    case(assume c):

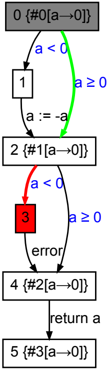
      $\phi := \phi \wedge \text{SymbolicEval}(c, \mathcal{S})$

  let $\langle \rho_{i+1}, \_ \rangle = S_{i+1}$

  $\phi := \phi \wedge \text{SymbolicEval}(\rho_{i+1}, \mathcal{S})$

end for

return ϕ



$\xrightarrow[\rho_{i+1} = true]{}$

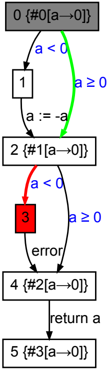
$\mathcal{S} \rightarrow [a \mapsto a_0]$

$\phi \rightarrow a_0 \geq 0$

ExecuteSymbolic

ExecuteSymbolic($\tau_c = \langle S_0, \ldots, S_k \rangle, P$)

let $\langle \rho_0, \_ \rangle = S_0$
$\mathcal{S} := [v \mapsto v_0 \mid v \in \text{params}(P)]$
$\phi := \text{SymbolicEval}(\rho_0, \mathcal{S})$
for $i = 0$ **to** $k - 1$ **do**
  $op := \text{Op}(S_i, S_{i+1})$
  match op
    case($v := e$):
      $\mathcal{S} := \mathcal{S}[v \mapsto \text{SymbolicEval}(e, \mathcal{S})]$
    case(assume c):
      $\phi := \phi \wedge \text{SymbolicEval}(c, \mathcal{S})$
  let $\langle \rho_{i+1}, \_ \rangle = S_{i+1}$
  $\phi := \phi \wedge \text{SymbolicEval}(\rho_{i+1}, \mathcal{S})$
end for
return ϕ



$\xrightarrow[op \,=\, a < 0]{}$

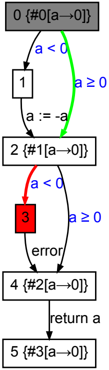
$\mathcal{S} \rightarrow [a \mapsto a_0]$

$\phi \rightarrow a_0 \geq 0$

ExecuteSymbolic

ExecuteSymbolic($\tau_c = \langle S_0, \ldots, S_k \rangle, P$)

let $\langle \rho_0, \_ \rangle = S_0$
$\mathcal{S} := [v \mapsto v_0 \mid v \in \text{params}(P)]$
$\phi := \text{SymbolicEval}(\rho_0, \mathcal{S})$
for $i = 0$ **to** $k - 1$ **do**
  $op := \text{Op}(S_i, S_{i+1})$
  match op
    case($v := e$):
      $\mathcal{S} := \mathcal{S}[v \mapsto \text{SymbolicEval}(e, \mathcal{S})]$
    case(assume c):
      $\phi := \phi \wedge \text{SymbolicEval}(c, \mathcal{S})$
  let $\langle \rho_{i+1}, \_ \rangle = S_{i+1}$
  $\phi := \phi \wedge \text{SymbolicEval}(\rho_{i+1}, \mathcal{S})$
end for
return ϕ



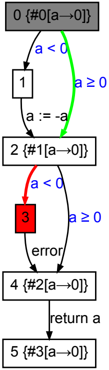
$$\xrightarrow[op \,=\, a < 0]{} \quad \mathcal{S} \rightarrow [a \mapsto a_0]$$

$$\phi \rightarrow a_0 \geq 0 \wedge a_0 < 0$$

ExecuteSymbolic

ExecuteSymbolic($\tau_c = \langle S_0, \ldots, S_k \rangle, P$)

let $\langle \rho_0, \_ \rangle = S_0$
$\mathcal{S} := [v \mapsto v_0 \mid v \in \text{params}(P)]$
$\phi := \text{SymbolicEval}(\rho_0, \mathcal{S})$
for $i = 0$ **to** $k - 1$ **do**
  $op := \text{Op}(S_i, S_{i+1})$
  match op
    case($v := e$):
      $\mathcal{S} := \mathcal{S}[v \mapsto \text{SymbolicEval}(e, \mathcal{S})]$
    case(assume c):
      $\phi := \phi \wedge \text{SymbolicEval}(c, \mathcal{S})$
  let $\langle \rho_{i+1}, \_ \rangle = S_{i+1}$
  $\phi := \phi \wedge \text{SymbolicEval}(\rho_{i+1}, \mathcal{S})$
end for
return ϕ



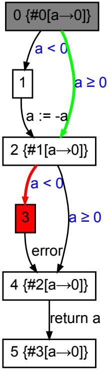
$\xrightarrow[\rho_{i+1} \,=\, true]{}$ $\mathcal{S} \rightarrow [a \mapsto a_0]$

$\phi \rightarrow a_0 \geq 0 \wedge a_0 < 0$

ExecuteSymbolic

ExecuteSymbolic($\tau_c = \langle S_0, \ldots, S_k \rangle, P$)

```
let ⟨ρ0, _⟩ = S0
𝒮 := [v ↦ v0 | v ∈ params(P)]
φ := SymbolicEval(ρ0, 𝒮)
for i = 0 to k − 1 do
    op := Op(Si, Si+1)
    match op
        case(v := e):
            𝒮 := 𝒮[v ↦ SymbolicEval(e, 𝒮)]
        case(assume c):
            φ := φ ∧ SymbolicEval(c, 𝒮)
    let⟨ρi+1, _⟩ = Si+1
    φ := φ ∧ SymbolicEval(ρi+1, 𝒮)
end for
return φ
```



$\xrightarrow{\rho_{i+1} = true}$

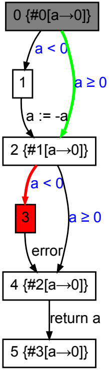
$\mathcal{S} \rightarrow [a \mapsto a_0]$

$\phi \rightarrow a_0 \geq 0 \wedge a_0 < 0$

ExecuteSymbolic

ExecuteSymbolic($\tau_c = \langle S_0, \ldots, S_k \rangle, P$)

let $\langle \rho_0, \_ \rangle = S_0$
$\mathcal{S} := [v \mapsto v_0 \mid v \in \text{params}(P)]$
$\phi := \text{SymbolicEval}(\rho_0, \mathcal{S})$
for $i = 0$ **to** $k - 1$ **do**
    $op := \text{Op}(S_i, S_{i+1})$
    match op
        case($v := e$):
            $\mathcal{S} := \mathcal{S}[v \mapsto \text{SymbolicEval}(e, \mathcal{S})]$
        case(assume c):
            $\phi := \phi \wedge \text{SymbolicEval}(c, \mathcal{S})$
    let $\langle \rho_{i+1}, \_ \rangle = S_{i+1}$
    $\phi := \phi \wedge \text{SymbolicEval}(\rho_{i+1}, \mathcal{S})$
end for
return ϕ



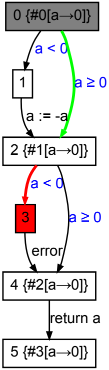
$\longrightarrow$

$$\mathcal{S} \to [a \mapsto a_0]$$
$$\phi \to a_0 \geq 0 \wedge a_0 < 0$$

ExtendFrontier - Refine

ExtendFrontier(τ_c, P)

```
φ := ExecuteSymbolic(τ_c, P)
t := IsSAT(φ, P)
if t = UNSAT then
    ρ := RefinePred(τ_c)
else
    ρ := true
end if
return ⟨t, ρ⟩
```

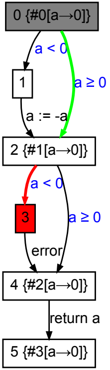


$\longrightarrow \quad \phi \to a_0 \geq 0 \wedge a_0 < 0$

ExtendFrontier - Refine

ExtendFrontier(τ_c, P)

ϕ := ExecuteSymbolic(τ_c, P)
t := IsSAT(ϕ, P)
if $t = \textsc{unsat}$ **then**
    ρ := RefinePred(τ_c)
else
    ρ := *true*
end if
return $\langle t, \rho \rangle$

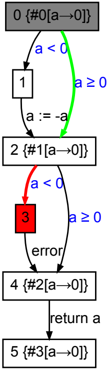


$$\phi \to a_0 \geq 0 \wedge a_0 < 0$$
$$t \to \text{UNSAT}$$

ExtendFrontier - Refine

ExtendFrontier(τ_c, P)

ϕ := ExecuteSymbolic(τ_c, P)
t := IsSAT(ϕ, P)
if $t = \text{UNSAT}$ **then**
    ρ := RefinePred(τ_c)
else
    ρ := *true*
end if
return $\langle t, \rho \rangle$



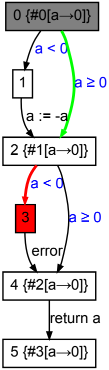
$\longrightarrow$

$$\phi \rightarrow a_0 \geq 0 \wedge a_0 < 0$$
$$t \rightarrow \text{UNSAT}$$

RefinePred

RefinePred($\tau_c = \langle S_0, \ldots, S_{k-1}, S_k \rangle$)
let $\langle \_, states_{k-1} \rangle = S_{k-1}$
let $\langle \rho_k, \_ \rangle = S_k$
op := Op(S_{k-1}, S_k)
if op **matches** assume c **then**
    if $k > 1 \wedge \forall s \in states_{k-1}$: Eval($\neg\rho_k, s$) $= true$ **then**
        return ρ_k
    end if
end if
return WP(op, ρ_k)

WP(op, ρ_k)
match op
    case(v := e):
        return $\rho_k[e/v]$
    case(assume c):
        return $c \wedge \rho_k$

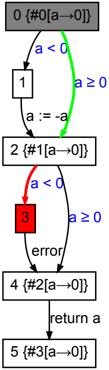


$\longrightarrow$

RefinePred

RefinePred($\tau_c = \langle S_0, \ldots, S_{k-1}, S_k \rangle$)
let $\langle \_, states_{k-1} \rangle = S_{k-1}$
let $\langle \rho_k, \_ \rangle = S_k$
$op := \mathrm{Op}(S_{k-1}, S_k)$
if op **matches** assume c **then**
    if $k > 1 \wedge \forall s \in states_{k-1} : \mathrm{Eval}(\neg \rho_k, s) = true$ **then**
        return ρ_k
    end if
end if
return WP(op, ρ_k)

WP(op, ρ_k)
match op
    case(v := e):
        return $\rho_k[e/v]$
    case(assume c):
        return $c \wedge \rho_k$



$\longrightarrow$
$\rho_k \rightarrow true$

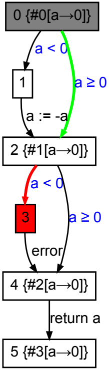
RefinePred

RefinePred($\tau_c = \langle S_0, \ldots, S_{k-1}, S_k \rangle$)

let $\langle \_, states_{k-1} \rangle = S_{k-1}$
let $\langle \rho_k, \_ \rangle = S_k$
$op := \mathrm{Op}(S_{k-1}, S_k)$
if op **matches** assume c **then**
    if $k > 1 \wedge \forall s \in states_{k-1} : \mathrm{Eval}(\neg \rho_k, s) = true$ **then**
        return ρ_k
    end if
end if
return WP(op, ρ_k)

WP(op, ρ_k)

match op
    case(v := e):
        return $\rho_k[e/v]$
    case(assume c):
        return $c \wedge \rho_k$



$\longrightarrow$

$\rho_k \rightarrow true$
$op \rightarrow a < 0$

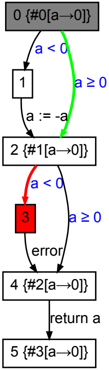
RefinePred

RefinePred($\tau_c = \langle S_0, \ldots, S_{k-1}, S_k \rangle$)

let $\langle \_, states_{k-1} \rangle = S_{k-1}$
let $\langle \rho_k, \_ \rangle = S_k$
$op := \mathrm{Op}(S_{k-1}, S_k)$
if *op* **matches** assume *c* **then**
    if $k > 1 \wedge \forall s \in states_{k-1} : \mathrm{Eval}(\neg \rho_k, s) = true$ **then**
        return ρ_k
    end if
end if
return $\mathrm{WP}(op, \rho_k)$

WP(op, ρ_k)

match *op*
    case(v := e):
        return $\rho_k[e/v]$
    case(assume c):
        return $c \wedge \rho_k$



$\longrightarrow$

$\rho_k \rightarrow true$
$op \rightarrow a < 0$

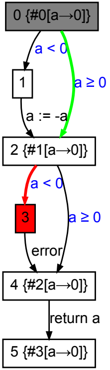
RefinePred

RefinePred($\tau_c = \langle S_0, \ldots, S_{k-1}, S_k \rangle$)

let $\langle \_, states_{k-1} \rangle = S_{k-1}$
let $\langle \rho_k, \_ \rangle = S_k$
op := Op(S_{k-1}, S_k)
if op **matches** assume c **then**
  if $k > 1 \wedge \forall s \in states_{k-1}$: Eval($\neg \rho_k, s$) $= true$ **then**
    return ρ_k
  end if
end if
return WP(op, ρ_k)

WP(op, ρ_k)

match op
  case(v := e):
    return $\rho_k[e/v]$
  case(assume c):
    return $c \wedge \rho_k$



$\longrightarrow$

$\rho_k \rightarrow true$
$op \rightarrow a < 0$

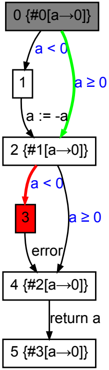
RefinePred

RefinePred($\tau_c = \langle S_0, \ldots, S_{k-1}, S_k \rangle$)

let $\langle \_, states_{k-1} \rangle = S_{k-1}$
let $\langle \rho_k, \_ \rangle = S_k$
$op :=$ Op(S_{k-1}, S_k)
if op **matches** assume c **then**
  if $k > 1 \wedge \forall s \in states_{k-1}$: Eval$(\neg\rho_k, s) = true$ **then**
    return ρ_k
  end if
end if
return WP(op, ρ_k)

WP(op, ρ_k)

match op
  case(v := e):
    return $\rho_k[e/v]$
  case(assume c):
    return $c \wedge \rho_k$



$\longrightarrow$

$\rho_k \rightarrow true$
$op \rightarrow a < 0$

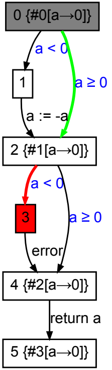
RefinePred

RefinePred($\tau_c = \langle S_0, \ldots, S_{k-1}, S_k \rangle$)

let $\langle \_, states_{k-1} \rangle = S_{k-1}$
let $\langle \rho_k, \_ \rangle = S_k$
$op := \text{Op}(S_{k-1}, S_k)$
if op **matches** assume c **then**
  if $k > 1 \wedge \forall s \in states_{k-1} : \text{Eval}(\neg \rho_k, s) = true$ **then**
    return ρ_k
  end if
end if
return WP(op, ρ_k)

WP(op, ρ_k)

match op
  case(v := e):
    return $\rho_k[e/v]$
  case(assume c):
    return $c \wedge \rho_k$



$\longrightarrow$

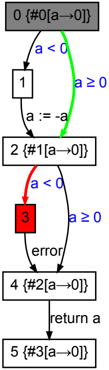
$\rho_k \to true$
$op \to a < 0$

$a < 0$

ExtendFrontier - Refine

ExtendFrontier(τ_c, P)

ϕ := ExecuteSymbolic(τ_c, P)
t := IsSAT(ϕ, P)
if $t = \text{UNSAT}$ **then**
    ρ := RefinePred(τ_c)
else
    ρ := *true*
end if
return $\langle t, \rho \rangle$

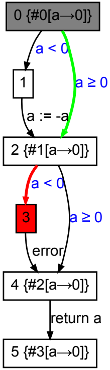


$\longrightarrow$

$$\phi \rightarrow a \geq 0 \wedge a < 0$$
$$t \rightarrow \text{UNSAT}$$
$$\rho \rightarrow a < 0$$

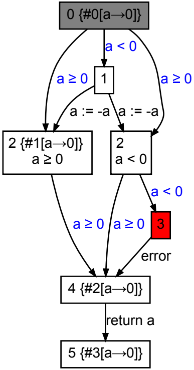
ExtendFrontier - Refine

ExtendFrontier(τ_c, P)
1: ϕ := ExecuteSymbolic(τ_c, P)
2: t := IsSAT(ϕ, P)
3: **if** t = UNSAT **then**
4: ρ := RefinePred(τ_c)
5: **else**
6: ρ := *true*
7: **end if**
8: **return** $\langle t, \rho \rangle$

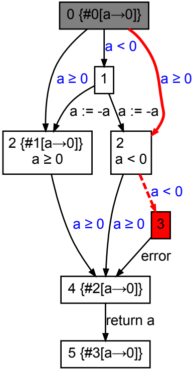


$\longrightarrow \quad \langle UNSAT, a < 0 \rangle$

Last steps of example



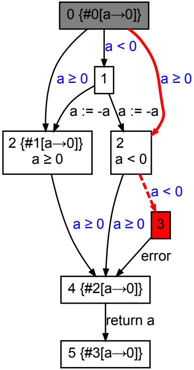
Last steps of example



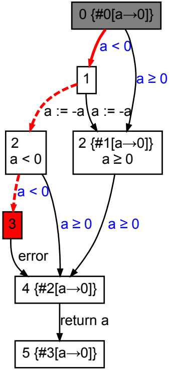
$$\longrightarrow$$

$$\phi \rightarrow a_0 \geq 0 \wedge a_0 < 0$$

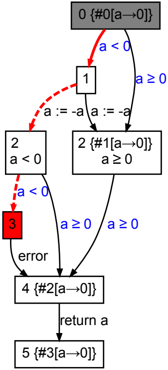
Last steps of example



$$\xrightarrow{\qquad}$$
$$\phi \to a_0 \geq 0 \wedge a_0 < 0$$

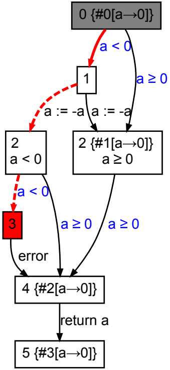


Last steps of example

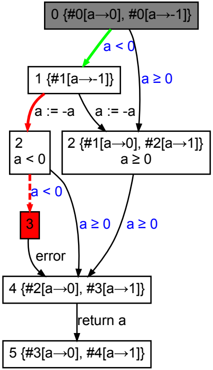


$$\xrightarrow{\quad}$$
$$\phi \rightarrow a_0 < 0$$

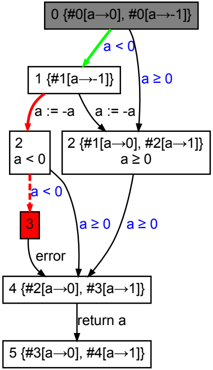
Last steps of example



$$\xrightarrow[\phi \rightarrow a_0 < 0]{}$$

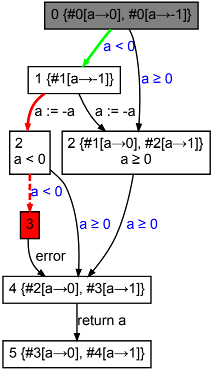


Last steps of example

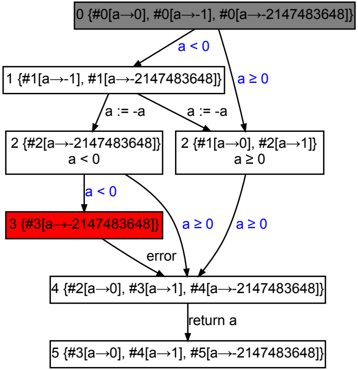


$$\xrightarrow[\phi \rightarrow a_0 < 0 \land -a_0 < 0]{}$$

Last steps of example



$$\xrightarrow{}$$
$$\phi \rightarrow a_0 < 0 \wedge -a_0 < 0$$



Modifications and Challenges

Modifications to DashLoop

loop
    τ := FindAbstractErrorPath(G)
    if τ = NO-PATH **then**
        return (PASS, G)
    end if

    τ_c := ConvertToRegionTraceWithAbstractFrontier(τ, G)
    $\langle t, \rho \rangle$:= ExtendFrontier(τ_c, P)
    if $t \neq$ UNSAT **then**
        G := RunTest(t, P, G)
        if IsErrorRegionReached(G) **then**
            return (FAIL, t)
        end if
    else
        G := RefineGraph(ρ, τ_c, G)
    end if
end loop

$\Sigma_{\simeq} := \bigcup_{l \in L} \{\{(pc, v) \in \Sigma \mid pc = l\}\}$
$\sigma^{I}{}_{\simeq} := \{S \in \Sigma_{\simeq} \mid \text{pc}(S) \textit{ is the the initial pc}\}$
$\rightarrow_{\simeq} := \{(S, S') \in \Sigma_{\simeq} \times \Sigma_{\simeq} \mid \text{Edge}(S, S') \in E\}$
$P_{\simeq} := \langle \Sigma_{\simeq}, \sigma^{I}_{\sigma}, \rightarrow_{\simeq} \rangle$
F:= Test(P)
loop
    if $\varphi \cap F \neq \emptyset$ **then**
        choose $s \in \varphi \cap F$
        t := TestForWitness(s)
        return ("fail", t)
    end if
    τ := GetAbstractTrace($P_{\simeq}$, φ)
    if $\tau = \epsilon$ **then**
        return ("pass", $\Sigma_{\simeq}$)
    else
        τ_0 := GetOrderedAbstractTrace(τ, F)
        $\langle t, \rho \rangle$:= ExtendFrontier(τ_0, F, P)
        if ρ = *true* **then**
            F := AddTestToForest(t, F)
        else
            Refinement of graph
        end if
    end if
end loop

Modifications to DashLoop

loop
    $\tau :=$ FindAbstractErrorPath(G)
    if $\tau =$ NO-PATH **then**
        return (PASS, G)
    end if

    $\tau_c :=$ ConvertToRegionTraceWithAbstractFrontier(τ, G)
    $\langle t, \rho \rangle :=$ ExtendFrontier(τ_c, P)
    if $t \neq$ UNSAT **then**
        $G :=$ RunTest(t, P, G)
        if IsErrorRegionReached(G) **then**
            return (FAIL, t)
        end if
    else
        $G :=$ RefineGraph(ρ, τ_c, G)
    end if
end loop

- Remove creation of region graph

$\Sigma_{\simeq} := \bigcup_{l \in L} \{\{(pc, v) \in \Sigma \mid pc = l\}\}$
$\sigma^{I}{}_{\simeq} := \{S \in \Sigma_{\simeq} \mid \mathrm{pc}(S) \textit{ is the the initial } pc\}$
$\rightarrow_{\simeq} := \{(S, S') \in \Sigma_{\simeq} \times \Sigma_{\simeq} \mid \mathrm{Edge}(S, S') \in E\}$
$P_{\simeq} := \langle \Sigma_{\simeq}, \sigma^{I}_{\sigma}, \rightarrow_{\simeq} \rangle$
$F :=$ Test(P)
loop
    if $\varphi \cap F \neq \emptyset$ **then**
        choose $s \in \varphi \cap F$
        $t :=$ TestForWitness(s)
        return ("fail", t)
    end if
    $\tau :=$ GetAbstractTrace($P_{\simeq}$, φ)
    if $\tau = \epsilon$ **then**
        return ("pass", $\Sigma_{\simeq}$)
    else
        $\tau_0 :=$ GetOrderedAbstractTrace(τ, F)
        $\langle t, \rho \rangle :=$ ExtendFrontier(τ_0, F, P)
        if $\rho =$ *true* **then**
            $F :=$ AddTestToForest(t, F)
        else
            Refinement of graph
        end if
    end if
end loop

Modifications to DashLoop

loop
    $\tau :=$ FindAbstractErrorPath(G)
    if $\tau =$ NO-PATH **then**
        return (PASS, G)
    end if

    $\tau_c :=$ ConvertToRegionTraceWithAbstractFrontier(τ, G)
    $\langle t, \rho \rangle :=$ ExtendFrontier(τ_c, P)
    if $t \neq$ UNSAT **then**
        $G :=$ RunTest(t, P, G)
        if IsErrorRegionReached(G) **then**
            return (FAIL, t)
        end if
    else
        $G :=$ RefineGraph(ρ, τ_c, G)
    end if
end loop

- Remove creation of region graph
- Do no load test input

$\Sigma_{\simeq} := \bigcup_{l \in L} \{\{(pc, v) \in \Sigma \mid pc = l\}\}$
$\sigma^{I}{}_{\simeq} := \{S \in \Sigma_{\simeq} \mid \mathrm{pc}(S)$ *is the the initial pc*$\}$
$\rightarrow_{\simeq} := \{(S, S') \in \Sigma_{\simeq} \times \Sigma_{\simeq} \mid \mathrm{Edge}(S, S') \in E\}$
$P_{\simeq} := \langle \Sigma_{\simeq}, \sigma^{I}_{\sigma}, \rightarrow_{\simeq} \rangle$
$F :=$ Test(P)
loop
    if $\varphi \cap F \neq \emptyset$ **then**
        choose $s \in \varphi \cap F$
        $t :=$ TestForWitness(s)
        return ("fail", t)
    end if
    $\tau :=$ GetAbstractTrace$(P_{\simeq}, \varphi)$
    if $\tau = \epsilon$ **then**
        return ("pass", $\Sigma_{\simeq}$)
    else
        $\tau_0 :=$ GetOrderedAbstractTrace(τ, F)
        $\langle t, \rho \rangle :=$ ExtendFrontier(τ_0, F, P)
        if $\rho =$ *true* **then**
            $F :=$ AddTestToForest(t, F)
        else
            Refinement of graph
        end if
    end if
end loop

Modifications to DashLoop

loop
  $\tau :=$ FindAbstractErrorPath(G)
  if $\tau =$ NO-PATH **then**
    return (PASS, G)
  end if

  $\tau_c :=$ ConvertToRegionTraceWithAbstractFrontier(τ, G)
  $\langle t, \rho \rangle :=$ ExtendFrontier(τ_c, P)
  if $t \neq$ UNSAT **then**
    $G :=$ RunTest(t, P, G)
    if IsErrorRegionReached(G) **then**
      return (FAIL, t)
    end if
  else
    $G :=$ RefineGraph(ρ, τ_c, G)
  end if
end loop

- Remove creation of region graph
- Do no load test input
- No forest

$\Sigma_{\simeq} := \bigcup_{l \in L} \{\{(pc, v) \in \Sigma \mid pc = l\}\}$
$\sigma^{I}{}_{\simeq} := \{S \in \Sigma_{\simeq} \mid \text{pc}(S)$ *is the the initial pc*$\}$
$\rightarrow_{\simeq} := \{(S, S') \in \Sigma_{\simeq} \times \Sigma_{\simeq} \mid \text{Edge}(S, S') \in E\}$
$P_{\simeq} := \langle \Sigma_{\simeq}, \sigma^{I}_{\sigma}, \rightarrow_{\simeq} \rangle$
$F :=$ Test(P)
loop
  if $\varphi \cap F \neq \emptyset$ **then**
    choose $s \in \varphi \cap F$
    $t :=$ TestForWitness(s)
    return ("fail", t)
  end if
  $\tau :=$ GetAbstractTrace$(P_{\simeq}, \varphi)$
  if $\tau = \epsilon$ **then**
    return ("pass", $\Sigma_{\simeq}$)
  else
    $\tau_0 :=$ GetOrderedAbstractTrace(τ, F)
    $\langle t, \rho \rangle :=$ ExtendFrontier(τ_0, F, P)
    if $\rho = $ *true* **then**
      $F :=$ AddTestToForest(t, F)
    else
      Refinement of graph
    end if
  end if
end loop

Modifications to DashLoop

```
loop
    τ := FindAbstractErrorPath(G)
    if τ = NO-PATH then
        return (PASS, G)
    end if

    τ_c := ConvertToRegionTraceWithAbstractFrontier(τ, G)
    ⟨t, ρ⟩ := ExtendFrontier(τ_c, P)
    if t ≠ UNSAT then
        G := RunTest(t, P, G)
        if IsErrorRegionReached(G) then
            return (FAIL, t)
        end if
    else
        G := RefineGraph(ρ, τ_c, G)
    end if
end loop
```

- Remove creation of region graph
- Do no load test input
- No forest
- Move test for error region reached

```
Σ_≃ := ⋃_{l∈L} {{(pc, v) ∈ Σ | pc = l}}
σ^I_≃ := {S ∈ Σ_≃ | pc(S) is the initial pc}
→_≃ := {(S, S′) ∈ Σ_≃ × Σ_≃ | Edge(S, S′) ∈ E}
P_≃ := ⟨Σ_≃, σ^I_σ, →_≃⟩
F := Test(P)
loop
    if φ ∩ F ≠ ∅ then
        choose s ∈ φ ∩ F
        t := TestForWitness(s)
        return ("fail", t)
    end if
    τ := GetAbstractTrace(P_≃, φ)
    if τ = ϵ then
        return ("pass", Σ_≃)
    else
        τ_0 := GetOrderedAbstractTrace(τ, F)
        ⟨t, ρ⟩ := ExtendFrontier(τ_0, F, P)
        if ρ = true then
            F := AddTestToForest(t, F)
        else
            Refinement of graph
        end if
    end if
end loop
```

Modifications to DashLoop

```
loop
    τ := FindAbstractErrorPath(G)
    if τ = NO-PATH then
        return (PASS, G)
    end if

    τ_c := ConvertToRegionTraceWithAbstractFrontier(τ, G)
    ⟨t, ρ⟩ := ExtendFrontier(τ_c, P)
    if t ≠ UNSAT then
        G := RunTest(t, P, G)
        if IsErrorRegionReached(G) then
            return (FAIL, t)
        end if
    else
        G := RefineGraph(ρ, τ_c, G)
    end if
end loop
```

- Remove creation of region graph
- Do no load test input
- No forest
- Move test for error region reached
- The rest is nearly the same

```
Σ_≃ := ⋃_{l∈L} {{(pc, v) ∈ Σ | pc = l}}
σ^I_≃ := {S ∈ Σ_≃ | pc(S) is the initial pc}
→_≃ := {(S, S′) ∈ Σ_≃ × Σ_≃ | Edge(S, S′) ∈ E}
P_≃ := ⟨Σ_≃, σ^I_σ, →_≃⟩
F := Test(P)
loop
    if φ ∩ F ≠ ∅ then
        choose s ∈ φ ∩ F
        t := TestForWitness(s)
        return ("fail", t)
    end if
    τ := GetAbstractTrace(P_≃, φ)
    if τ = ϵ then
        return ("pass", Σ_≃)
    else
        τ_0 := GetOrderedAbstractTrace(τ, F)
        ⟨t, ρ⟩ := ExtendFrontier(τ_0, F, P)
        if ρ = true then
            F := AddTestToForest(t, F)
        else
            Refinement of graph
        end if
    end if
end loop
```

Modifications to DashLoop

```
loop
    τ := FindAbstractErrorPath(G)
    if τ = NO-PATH then
        return (PASS, G)
    end if

    τ_c := ConvertToRegionTraceWithAbstractFrontier(τ, G)
    ⟨t, ρ⟩ := ExtendFrontier(τ_c, P)
    if t ≠ UNSAT then
        G := RunTest(t, P, G)
        if IsErrorRegionReached(G) then
            return (FAIL, t)
        end if
    else
        G := RefineGraph(ρ, τ_c, G)
    end if
end loop
```

- Remove creation of region graph
- Do no load test input
- No forest
- Move test for error region reached
- The rest is nearly the same

```
Σ_≃ := ⋃_{l∈L} {{(pc, v) ∈ Σ | pc = l}}
σ^I_≃ := {S ∈ Σ_≃ | pc(S) is the initial pc}
→_≃ := {(S, S′) ∈ Σ_≃ × Σ_≃ | Edge(S, S′) ∈ E}
P_≃ := ⟨Σ_≃, σ^I_σ, →_≃⟩
F := Test(P)
loop
    if φ ∩ F ≠ ∅ then
        choose s ∈ φ ∩ F
        t := TestForWitness(s)
        return ("fail", t)
    end if
    τ := GetAbstractTrace(P_≃, φ)
    if τ = ϵ then
        return ("pass", Σ_≃)
    else
        τ_0 := GetOrderedAbstractTrace(τ, F)
        ⟨t, ρ⟩ := ExtendFrontier(τ_0, F, P)
        if ρ = true then
            F := AddTestToForest(t, F)
        else
            Refinement of graph
        end if
    end if
end loop
```

Modifications to DashLoop

```
loop
    τ := FindAbstractErrorPath(G)
    if τ = NO-PATH then
        return (PASS, G)
    end if

    τ_c := ConvertToRegionTraceWithAbstractFrontier(τ, G)
    ⟨t, ρ⟩ := ExtendFrontier(τ_c, P)
    if t ≠ UNSAT then
        G := RunTest(t, P, G)
        if IsErrorRegionReached(G) then
            return (FAIL, t)
        end if
    else
        G := RefineGraph(ρ, τ_c, G)
    end if
end loop
```

- Remove creation of region graph
- Do no load test input
- No forest
- Move test for error region reached
- The rest is nearly the same

```
Σ_≃ := ⋃_{l∈L} {{(pc, v) ∈ Σ | pc = l}}
σ^I_≃ := {S ∈ Σ_≃ | pc(S) is the initial pc}
→_≃ := {(S, S′) ∈ Σ_≃ × Σ_≃ | Edge(S, S′) ∈ E}
P_≃ := ⟨Σ_≃, σ^I_σ, →_≃⟩
F := Test(P)
loop
    if φ ∩ F ≠ ∅ then
        choose s ∈ φ ∩ F
        t := TestForWitness(s)
        return ("fail", t)
    end if
    τ := GetAbstractTrace(P_≃, φ)
    if τ = ϵ then
        return ("pass", Σ_≃)
    else
        τ_0 := GetOrderedAbstractTrace(τ, F)
        ⟨t, ρ⟩ := ExtendFrontier(τ_0, F, P)
        if ρ = true then
            F := AddTestToForest(t, F)
        else
            Refinement of graph
        end if
    end if
end loop
```

Modifications to DashLoop

loop
  τ := FindAbstractErrorPath(G)
  if τ = NO-PATH **then**
    return (PASS, G)
  end if

  τ_c := ConvertToRegionTraceWithAbstractFrontier(τ, G)
  $\langle t, \rho \rangle$:= ExtendFrontier(τ_c, P)
  if $t \neq$ UNSAT **then**
    G := RunTest(t, P, G)
    if IsErrorRegionReached(G) **then**
      return (FAIL, t)
    end if
  else
    G := RefineGraph(ρ, τ_c, G)
  end if
end loop

- Remove creation of region graph
- Do no load test input
- No forest
- Move test for error region reached
- The rest is nearly the same

$\Sigma_{\simeq} := \bigcup_{l \in L} \{\{(pc, v) \in \Sigma \mid pc = l\}\}$
$\sigma^{I}{}_{\simeq} := \{S \in \Sigma_{\simeq} \mid \mathrm{pc}(S)$ *is the the initial pc*$\}$
$\rightarrow_{\simeq} := \{(S, S') \in \Sigma_{\simeq} \times \Sigma_{\simeq} \mid \mathrm{Edge}(S, S') \in E\}$
$P_{\simeq} := \langle \Sigma_{\simeq}, \sigma^{I}_{\sigma}, \rightarrow_{\simeq} \rangle$
F:= Test(P)
loop
  if $\varphi \cap F \neq \emptyset$ **then**
    choose $s \in \varphi \cap F$
    t := TestForWitness(s)
    return ("fail", t)
  end if
  τ := GetAbstractTrace($P_{\simeq}$, φ)
  if $\tau = \epsilon$ **then**
    return ("pass", $\Sigma_{\simeq}$)
  else
    τ_0 := GetOrderedAbstractTrace(τ, F)
    $\langle t, \rho \rangle$:= ExtendFrontier(τ_0, F, P)
    if ρ = *true* **then**
      F := AddTestToForest(t, F)
    else
      Refinement of graph
    end if
  end if
end loop

Modifications to DashLoop

loop
  $\tau := \text{FindAbstractErrorPath}(G)$
  if $\tau = \text{NO-PATH}$ **then**
    return (PASS, G)
  end if

  $\tau_c := \text{ConvertToRegionTraceWithAbstractFrontier}(\tau, G)$
  $\langle t, \rho \rangle := \text{ExtendFrontier}(\tau_c, P)$
  if $t \neq \text{UNSAT}$ **then**
    $G := \text{RunTest}(t, P, G)$
    if $\text{IsErrorRegionReached}(G)$ **then**
      return (FAIL, t)
    end if
  else
    $G := \text{RefineGraph}(\rho, \tau_c, G)$
  end if
end loop

- Remove creation of region graph
- Do no load test input
- No forest
- Move test for error region reached
- The rest is nearly the same
- Refinement moved to later slide

$\Sigma_{\simeq} := \bigcup_{l \in L} \{\{(pc, v) \in \Sigma \mid pc = l\}\}$
$\sigma^{I}{}_{\simeq} := \{S \in \Sigma_{\simeq} \mid \text{pc}(S)$ *is the initial pc*$\}$
$\rightarrow_{\simeq} := \{(S, S') \in \Sigma_{\simeq} \times \Sigma_{\simeq} \mid \text{Edge}(S, S') \in E\}$
$P_{\simeq} := \langle \Sigma_{\simeq}, \sigma^{I}_{\sigma}, \rightarrow_{\simeq} \rangle$
$F := \text{Test}(P)$
loop
  if $\varphi \cap F \neq \emptyset$ **then**
    choose $s \in \varphi \cap F$
    $t := \text{TestForWitness}(s)$
    return ("fail", t)
  end if
  $\tau := \text{GetAbstractTrace}(P_{\simeq}, \varphi)$
  if $\tau = \epsilon$ **then**
    return ("pass", $\Sigma_{\simeq}$)
  else
    $\tau_0 := \text{GetOrderedAbstractTrace}(\tau, F)$
    $\langle t, \rho \rangle := \text{ExtendFrontier}(\tau_0, F, P)$
    if $\rho = \mathit{true}$ **then**
      $F := \text{AddTestToForest}(t, F)$
    else
      Refinement of graph
    end if
  end if
end loop

How to create a trace from a path

- What makes a trace ordered?
  - GetOrderedAbstractTrace and τ_o

How to create a trace from a path

- What makes a trace ordered?
  - GetOrderedAbstractTrace and τ_o
- Why include the tail of the path in the trace?
  - $\tau_c = S_0, \ldots, S_{k-1}, S_k$
  - $\tau_o = S_0, \ldots, S_{k-1}, S_k, \ldots S_n$

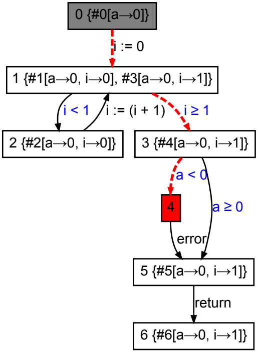
How to create a trace from a path

- What makes a trace ordered?
  - GetOrderedAbstractTrace and τ_o
- Why include the tail of the path in the trace?
  - $\tau_c = S_0, \ldots, S_{k-1}, S_k$
  - $\tau_o = S_0, \ldots, S_{k-1}, S_k, \ldots S_n$
- Should the trace τ_c follow the path τ?

```
void foo(int a)
{
  int i = 0;
  while(i < 1)
    i = i + 1;
  assert(a < 0);
}
```

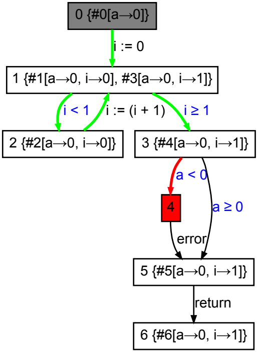
How to create a trace from a path

- What makes a trace ordered?
  - GetOrderedAbstractTrace and τ_o
- Why include the tail of the path in the trace?
  - $\tau_c = S_0, \ldots, S_{k-1}, S_k$
  - $\tau_o = S_0, \ldots, S_{k-1}, S_k, \ldots S_n$
- Should the trace τ_c follow the path τ?



How to create a trace from a path

- What makes a trace ordered?
  - GetOrderedAbstractTrace and τ_o
- Why include the tail of the path in the trace?
  - $\tau_c = S_0, \ldots, S_{k-1}, S_k$
  - $\tau_o = S_0, \ldots, S_{k-1}, S_k, \ldots S_n$
- Should the trace τ_c follow the path τ?
  - No.



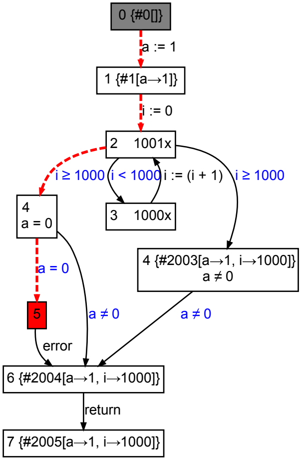
How to create a trace from a path

- What makes a trace ordered?
  - GetOrderedAbstractTrace and τ_o
- Why include the tail of the path in the trace?
  - $\tau_c = S_0, \ldots, S_{k-1}, S_k$
  - $\tau_o = S_0, \ldots, S_{k-1}, S_k, \ldots S_n$
- Should the trace τ_c follow the path τ?
  - No.
- Which state to pick when generating a trace?

```
void foo()
{
  int a = 1;
  int i = 0;
  while(i < 1000)
    i = i + 1;
  if(a == 0)
    error;
}
```

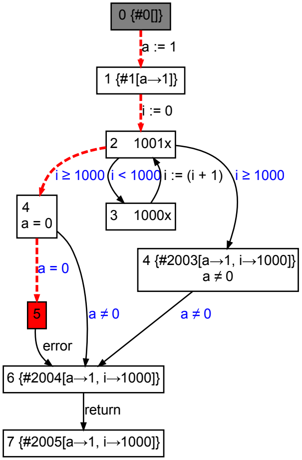
How to create a trace from a path

- What makes a trace ordered?
  - GetOrderedAbstractTrace and τ_o
- Why include the tail of the path in the trace?
  - $\tau_c = S_0, \ldots, S_{k-1}, S_k$
  - $\tau_o = S_0, \ldots, S_{k-1}, S_k, \ldots S_n$
- Should the trace τ_c follow the path τ?
  - No.
- Which state to pick when generating a trace?



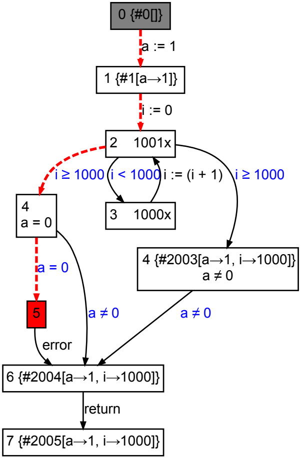
How to create a trace from a path

- What makes a trace ordered?
  - GetOrderedAbstractTrace and τ_o
- Why include the tail of the path in the trace?
  - $\tau_c = S_0, \ldots, S_{k-1}, S_k$
  - $\tau_o = S_0, \ldots, S_{k-1}, S_k, \ldots S_n$
- Should the trace τ_c follow the path τ?
  - No.
- Which state to pick when generating a trace?
  - Pick random



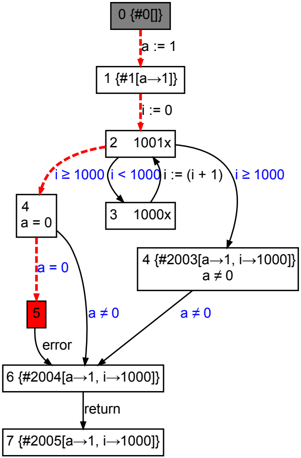
How to create a trace from a path

- What makes a trace ordered?
  - GetOrderedAbstractTrace and τ_o
- Why include the tail of the path in the trace?
  - $\tau_c = S_0, \dots, S_{k-1}, S_k$
  - $\tau_o = S_0, \dots, S_{k-1}, S_k, \dots S_n$
- Should the trace τ_c follow the path τ?
  - No.
- Which state to pick when generating a trace?
  - Pick random
  - Pick the last state



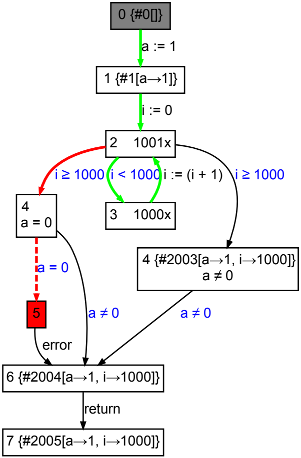
How to create a trace from a path

- What makes a trace ordered?
  - GetOrderedAbstractTrace and τ_o
- Why include the tail of the path in the trace?
  - $\tau_c = S_0, \ldots, S_{k-1}, S_k$
  - $\tau_o = S_0, \ldots, S_{k-1}, S_k, \ldots S_n$
- Should the trace τ_c follow the path τ?
  - No.
- Which state to pick when generating a trace?
  - Pick random
  - Pick the last state
  - Pick a state the enters the sought region



How to create a trace from a path

- What makes a trace ordered?
  - GetOrderedAbstractTrace and τ_o
- Why include the tail of the path in the trace?
  - $\tau_c = S_0, \ldots, S_{k-1}, S_k$
  - $\tau_o = S_0, \ldots, S_{k-1}, S_k, \ldots S_n$
- Should the trace τ_c follow the path τ?
  - No.
- Which state to pick when generating a trace?
  - Pick random
  - Pick the last state
  - Pick a state the enters the sought region



Modifications to refinement

RefineGraph$(\rho, \tau_c = \langle S_0, \ldots, S_{k-1}, S_k \rangle, G = \langle \Sigma_\simeq, \to_\simeq \rangle)$

let $\langle \rho_{k-1}, states \rangle = S_{k-1}$
if $k = 1$ **then**
  return $\langle \Sigma_\simeq, \to_\simeq \backslash (S_0, S_1) \rangle$
end if
$\Sigma^*_\simeq := \Sigma_\simeq \setminus \{S_{k-1}\}$
$\to^*_\simeq := \to_\simeq \backslash \{(S, S_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\to^*_\simeq := \to^*_\simeq \backslash \{(S_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\rho^*_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \neg\rho)$
$S^*_{k-1} := \langle \rho^*_{k-1}, states \rangle$
$\Sigma^*_\simeq := \Sigma^*_\simeq \cup \{S^*_{k-1}\}$
$\to^*_\simeq := \to^*_\simeq \cup \{(S, S^*_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\to^*_\simeq := \to^*_\simeq \cup \{(S^*_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\to^*_\simeq := \to^*_\simeq \backslash \{(S^*_{k-1}, S_k)\}$
$\rho^{**}_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \rho)$
$S^{**}_{k-1} := \langle \rho^{**}_{k-1}, \emptyset \rangle$
$\Sigma^*_\simeq := \Sigma^*_\simeq \cup \{S^{**}_{k-1}\}$
$\to^*_\simeq := \to^*_\simeq \cup \{(S^{**}_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
if IsSAT$(\rho^{**}_{k-1}) \neq$ UNSAT **then**
  $\to^*_\simeq := \to^*_\simeq \cup \{(S, S^{**}_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
end if
return $\langle \Sigma^*_\simeq, \to^*_\simeq \rangle$

let $S_0, S_1, \ldots, S_n = \tau_0$ and
$(k-1, k) = \text{Frontier}(\tau_0)$ in
$\Sigma_\simeq := (\Sigma_\simeq \backslash \{S_{k-1}\}) \cup \{S_{k-1} \wedge \rho, S_{k-1} \wedge \neg\rho\}$
$\to_\simeq := (\to_\simeq \backslash \{(S, S_{k-1}) \mid S \in \text{Parents}(S_{k-1})\})$
$\backslash \{(S_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\to_\simeq := \to_\simeq \cup \{(S, S_{k-1} \wedge \rho) \mid S \in \text{Parents}(S_{k-1})\}$
$\cup \{(S, S_{k-1} \wedge \neg\rho) \mid S \in \text{Parents}(S_{k-1})\}$
$\cup \{(S_{k-1} \wedge \rho, S) \mid S \in \text{Children}(S_{k-1})\}$
$\cup \{(S_{k-1} \wedge \neg\rho, S) \mid S \in (\text{Children}(S_{k-1}) \backslash \{S_k\})\}$

Modifications to refinement

RefineGraph($\rho, \tau_c = \langle S_0, \ldots, S_{k-1}, S_k \rangle, G = \langle \Sigma_\simeq, \to_\simeq \rangle$)

let $\langle \rho_{k-1}, states \rangle = S_{k-1}$
if $k = 1$ **then**
  return $\langle \Sigma_\simeq, \to_\simeq \setminus (S_0, S_1) \rangle$
end if
$\Sigma^*_\simeq := \Sigma_\simeq \setminus \{S_{k-1}\}$
$\to^*_\simeq := \to_\simeq \setminus \{(S, S_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\to^*_\simeq := \to^*_\simeq \setminus \{(S_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\rho^*_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \neg\rho)$
$S^*_{k-1} := \langle \rho^*_{k-1}, states \rangle$
$\Sigma^*_\simeq := \Sigma^*_\simeq \cup \{S^*_{k-1}\}$
$\to^*_\simeq := \to^*_\simeq \cup \{(S, S^*_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\to^*_\simeq := \to^*_\simeq \cup \{(S^*_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\to^*_\simeq := \to^*_\simeq \setminus \{(S^*_{k-1}, S_k)\}$
$\rho^{**}_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \rho)$
$S^{**}_{k-1} := \langle \rho^{**}_{k-1}, \emptyset \rangle$
$\Sigma^*_\simeq := \Sigma^*_\simeq \cup \{S^{**}_{k-1}\}$
$\to^*_\simeq := \to^*_\simeq \cup \{(S^{**}_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
if IsSAT(ρ^{**}_{k-1}) $\neq$ UNSAT **then**
  $\to^*_\simeq := \to^*_\simeq \cup \{(S, S^{**}_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
end if
return $\langle \Sigma^*_\simeq, \to^*_\simeq \rangle$

let $S_0, S_1, \ldots, S_n = \tau_0$ and
$(k-1, k) = \text{Frontier}(\tau_0)$ in
$\Sigma_\simeq := (\Sigma_\simeq \setminus \{S_{k-1}\}) \cup \{S_{k-1} \wedge \rho, S_{k-1} \wedge \neg\rho\}$
$\to_\simeq := (\to_\simeq \setminus \{(S, S_{k-1}) \mid S \in \text{Parents}(S_{k-1})\})$
$\setminus \{(S_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\to_\simeq := \to_\simeq \cup \{(S, S_{k-1} \wedge \rho) \mid S \in \text{Parents}(S_{k-1})\}$
$\cup \{(S, S_{k-1} \wedge \neg\rho) \mid S \in \text{Parents}(S_{k-1})\}$
$\cup \{(S_{k-1} \wedge \rho, S) \mid S \in \text{Children}(S_{k-1})\}$
$\cup \{(S_{k-1} \wedge \neg\rho, S) \mid S \in (\text{Children}(S_{k-1}) \setminus \{S_k\})\}$

- Disambiguate regions and region predicates

Modifications to refinement

RefineGraph($\rho, \tau_c = \langle S_0, \ldots, S_{k-1}, S_k \rangle, G = \langle \Sigma_\simeq, \to_\simeq \rangle$)

let $\langle \rho_{k-1}, states \rangle = S_{k-1}$
if $k = 1$ **then**
  return $\langle \Sigma_\simeq, \to_\simeq \setminus (S_0, S_1) \rangle$
end if
$\Sigma^*_\simeq := \Sigma_\simeq \setminus \{S_{k-1}\}$
$\to^*_\simeq := \to_\simeq \setminus \{(S, S_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\to^*_\simeq := \to^*_\simeq \setminus \{(S_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\rho^*_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \neg\rho)$
$S^*_{k-1} := \langle \rho^*_{k-1}, states \rangle$
$\Sigma^*_\simeq := \Sigma^*_\simeq \cup \{S^*_{k-1}\}$
$\to^*_\simeq := \to^*_\simeq \cup \{(S, S^*_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\to^*_\simeq := \to^*_\simeq \cup \{(S^*_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\to^*_\simeq := \to^*_\simeq \setminus \{(S^*_{k-1}, S_k)\}$
$\rho^{**}_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \rho)$
$S^{**}_{k-1} := \langle \rho^{**}_{k-1}, \emptyset \rangle$
$\Sigma^*_\simeq := \Sigma^*_\simeq \cup \{S^{**}_{k-1}\}$
$\to^*_\simeq := \to^*_\simeq \cup \{(S^{**}_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
if IsSAT(ρ^{**}_{k-1}) $\neq$ UNSAT **then**
  $\to^*_\simeq := \to^*_\simeq \cup \{(S, S^{**}_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
end if
return $\langle \Sigma^*_\simeq, \to^*_\simeq \rangle$

let $S_0, S_1, \ldots, S_n = \tau_0$ and
$(k-1, k) = \text{Frontier}(\tau_0)$ in
$\Sigma_\simeq := (\Sigma_\simeq \setminus \{S_{k-1}\}) \cup \{S_{k-1} \wedge \rho, S_{k-1} \wedge \neg\rho\}$
$\to_\simeq := (\to_\simeq \setminus \{(S, S_{k-1}) \mid S \in \text{Parents}(S_{k-1})\})$
$\setminus \{(S_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\to_\simeq := \to_\simeq \cup \{(S, S_{k-1} \wedge \rho) \mid S \in \text{Parents}(S_{k-1})\}$
$\cup \{(S, S_{k-1} \wedge \neg\rho) \mid S \in \text{Parents}(S_{k-1})\}$
$\cup \{(S_{k-1} \wedge \rho, S) \mid S \in \text{Children}(S_{k-1})\}$
$\cup \{(S_{k-1} \wedge \neg\rho, S) \mid S \in (\text{Children}(S_{k-1}) \setminus \{S_k\})\}$

- Disambiguate regions and region predicates
- Spell out what happens

Modifications to refinement

RefineGraph(ρ, $\tau_c = \langle S_0, \ldots, S_{k-1}, S_k \rangle$, $G = \langle \Sigma_\simeq, \rightarrow_\simeq \rangle$)

```
let⟨ρ_{k−1}, states⟩ = S_{k−1}
if k = 1 then
    return ⟨Σ_≃, →_≃ \(S_0, S_1)⟩
end if
Σ*_≃ := Σ_≃ \ {S_{k−1}}
→*_≃ := →_≃ \{(S, S_{k−1}) | S ∈ Parents(S_{k−1})}
→*_≃ := →*_≃ \{(S_{k−1}, S) | S ∈ Children(S_{k−1})}
ρ*_{k−1} := Simplify(ρ_{k−1} ∧ ¬ρ)
S*_{k−1} := ⟨ρ*_{k−1}, states⟩
Σ*_≃ := Σ*_≃ ∪ {S*_{k−1}}
→*_≃ := →*_≃ ∪{(S, S*_{k−1}) | S ∈ Parents(S_{k−1})}
→*_≃ := →*_≃ ∪{(S*_{k−1}, S) | S ∈ Children(S_{k−1})}
→*_≃ := →*_≃ \{(S*_{k−1}, S_k)}
ρ**_{k−1} := Simplify(ρ_{k−1} ∧ ρ)
S**_{k−1} := ⟨ρ**_{k−1}, ∅⟩
Σ*_≃ := Σ*_≃ ∪ {S**_{k−1}}
→*_≃ := →*_≃ ∪{(S**_{k−1}, S) | S ∈ Children(S_{k−1})}
if IsSAT(ρ**_{k−1}) ≠ UNSAT then
    →*_≃ := →*_≃ ∪{(S, S**_{k−1}) | S ∈ Parents(S_{k−1})}
end if
return ⟨Σ*_≃, →*_≃⟩
```

```
let S_0, S_1, ..., S_n = τ_0 and
(k − 1, k) = Frontier(τ_0) in
Σ_≃ := (Σ_≃\{S_{k−1}}) ∪ {S_{k−1} ∧ ρ, S_{k−1} ∧ ¬ρ}
→_≃ := (→_≃ \{(S, S_{k−1}) | S ∈Parents(S_{k−1})})
        \{(S_{k−1}, S) | S ∈ Children(S_{k−1})}
→_≃ := →_≃ ∪{(S, S_{k−1} ∧ ρ) | S ∈ Parents(S_{k−1})}
        ∪{(S, S_{k−1} ∧ ¬ρ) | S ∈ Parents(S_{k−1})}
        ∪{(S_{k−1} ∧ ρ, S) | S ∈ Children(S_{k−1})}
        ∪{(S_{k−1} ∧ ¬ρ, S) | S ∈ (Children(S_{k−1})\{S_k})}
```

- Disambiguate regions and region predicates
- Spell out what happens

Modifications to refinement

RefineGraph$(\rho, \tau_c = \langle S_0, \ldots, S_{k-1}, S_k \rangle, G = \langle \Sigma_{\simeq}, \to_{\simeq} \rangle)$

let $\langle \rho_{k-1}, \mathit{states} \rangle = S_{k-1}$
if $k = 1$ **then**
  return $\langle \Sigma_{\simeq}, \to_{\simeq} \setminus (S_0, S_1) \rangle$
end if
$\Sigma^*_{\simeq} := \Sigma_{\simeq} \setminus \{S_{k-1}\}$
$\to^*_{\simeq} := \to_{\simeq} \setminus \{(S, S_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\to^*_{\simeq} := \to^*_{\simeq} \setminus \{(S_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\rho^*_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \neg\rho)$
$S^*_{k-1} := \langle \rho^*_{k-1}, \mathit{states} \rangle$
$\Sigma^*_{\simeq} := \Sigma^*_{\simeq} \cup \{S^*_{k-1}\}$
$\to^*_{\simeq} := \to^*_{\simeq} \cup \{(S, S^*_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\to^*_{\simeq} := \to^*_{\simeq} \cup \{(S^*_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\to^*_{\simeq} := \to^*_{\simeq} \setminus \{(S^*_{k-1}, S_k)\}$
$\rho^{**}_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \rho)$
$S^{**}_{k-1} := \langle \rho^{**}_{k-1}, \emptyset \rangle$
$\Sigma^*_{\simeq} := \Sigma^*_{\simeq} \cup \{S^{**}_{k-1}\}$
$\to^*_{\simeq} := \to^*_{\simeq} \cup \{(S^{**}_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
if IsSAT$(\rho^{**}_{k-1}) \neq$ UNSAT **then**
  $\to^*_{\simeq} := \to^*_{\simeq} \cup \{(S, S^{**}_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
end if
return $\langle \Sigma^*_{\simeq}, \to^*_{\simeq} \rangle$

let $S_0, S_1, \ldots, S_n = \tau_0$ and
$(k-1, k) = \text{Frontier}(\tau_0)$ in
$\Sigma_{\simeq} := (\Sigma_{\simeq} \setminus \{S_{k-1}\}) \cup \{S_{k-1} \wedge \rho, S_{k-1} \wedge \neg\rho\}$
$\to_{\simeq} := (\to_{\simeq} \setminus \{(S, S_{k-1}) \mid S \in \text{Parents}(S_{k-1})\})$
$\quad \setminus \{(S_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\to_{\simeq} := \to_{\simeq} \cup \{(S, S_{k-1} \wedge \rho) \mid S \in \text{Parents}(S_{k-1})\}$
$\quad \cup \{(S, S_{k-1} \wedge \neg\rho) \mid S \in \text{Parents}(S_{k-1})\}$
$\quad \cup \{(S_{k-1} \wedge \rho, S) \mid S \in \text{Children}(S_{k-1})\}$
$\quad \cup \{(S_{k-1} \wedge \neg\rho, S) \mid S \in (\text{Children}(S_{k-1}) \setminus \{S_k\})\}$

- Disambiguate regions and region predicates
- Spell out what happens

Modifications to refinement

RefineGraph($\rho, \tau_c = \langle S_0, \ldots, S_{k-1}, S_k \rangle, G = \langle \Sigma_{\simeq}, \to_{\simeq} \rangle$)

let $\langle \rho_{k-1}, states \rangle = S_{k-1}$
if $k = 1$ **then**
  return $\langle \Sigma_{\simeq}, \to_{\simeq} \setminus (S_0, S_1) \rangle$
end if
$\Sigma_{\simeq}^* := \Sigma_{\simeq} \setminus \{S_{k-1}\}$
$\to_{\simeq}^* := \to_{\simeq} \setminus \{(S, S_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\to_{\simeq}^* := \to_{\simeq}^* \setminus \{(S_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\rho_{k-1}^* := \text{Simplify}(\rho_{k-1} \wedge \neg\rho)$
$S_{k-1}^* := \langle \rho_{k-1}^*, states \rangle$
$\Sigma_{\simeq}^* := \Sigma_{\simeq}^* \cup \{S_{k-1}^*\}$
$\to_{\simeq}^* := \to_{\simeq}^* \cup \{(S, S_{k-1}^*) \mid S \in \text{Parents}(S_{k-1})\}$
$\to_{\simeq}^* := \to_{\simeq}^* \cup \{(S_{k-1}^*, S) \mid S \in \text{Children}(S_{k-1})\}$
$\to_{\simeq}^* := \to_{\simeq}^* \setminus \{(S_{k-1}^*, S_k)\}$
$\rho_{k-1}^{**} := \text{Simplify}(\rho_{k-1} \wedge \rho)$
$S_{k-1}^{**} := \langle \rho_{k-1}^{**}, \emptyset \rangle$
$\Sigma_{\simeq}^* := \Sigma_{\simeq}^* \cup \{S_{k-1}^{**}\}$
$\to_{\simeq}^* := \to_{\simeq}^* \cup \{(S_{k-1}^{**}, S) \mid S \in \text{Children}(S_{k-1})\}$
if IsSAT(ρ_{k-1}^{**}) $\neq$ UNSAT **then**
  $\to_{\simeq}^* := \to_{\simeq}^* \cup \{(S, S_{k-1}^{**}) \mid S \in \text{Parents}(S_{k-1})\}$
end if
return $\langle \Sigma_{\simeq}^*, \to_{\simeq}^* \rangle$

let $S_0, S_1, \ldots, S_n = \tau_0$ and
$(k-1, k) = \text{Frontier}(\tau_0)$ in
$\Sigma_{\simeq} := (\Sigma_{\simeq} \setminus \{S_{k-1}\}) \cup \{S_{k-1} \wedge \rho, S_{k-1} \wedge \neg\rho\}$
$\to_{\simeq} := (\to_{\simeq} \setminus \{(S, S_{k-1}) \mid S \in \text{Parents}(S_{k-1})\})$
$\setminus \{(S_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\to_{\simeq} := \to_{\simeq} \cup \{(S, S_{k-1} \wedge \rho) \mid S \in \text{Parents}(S_{k-1})\}$
$\cup \{(S, S_{k-1} \wedge \neg\rho) \mid S \in \text{Parents}(S_{k-1})\}$
$\cup \{(S_{k-1} \wedge \rho, S) \mid S \in \text{Children}(S_{k-1})\}$
$\cup \{(S_{k-1} \wedge \neg\rho, S) \mid S \in (\text{Children}(S_{k-1}) \setminus \{S_k\})\}$

- Disambiguate regions and region predicates
- Spell out what happens

Modifications to refinement

RefineGraph($\rho, \tau_c = \langle S_0, \ldots, S_{k-1}, S_k \rangle, G = \langle \Sigma_{\simeq}, \rightarrow_{\simeq} \rangle$)

let $\langle \rho_{k-1}, states \rangle = S_{k-1}$
if $k = 1$ **then**
    return $\langle \Sigma_{\simeq}, \rightarrow_{\simeq} \setminus (S_0, S_1) \rangle$
end if
$\Sigma^*_{\simeq} := \Sigma_{\simeq} \setminus \{S_{k-1}\}$
$\rightarrow^*_{\simeq} := \rightarrow_{\simeq} \setminus \{(S, S_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \setminus \{(S_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\rho^*_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \neg\rho)$
$S^*_{k-1} := \langle \rho^*_{k-1}, states \rangle$
$\Sigma^*_{\simeq} := \Sigma^*_{\simeq} \cup \{S^*_{k-1}\}$
$\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \cup \{(S, S^*_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \cup \{(S^*_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \setminus \{(S^*_{k-1}, S_k)\}$
$\rho^{**}_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \rho)$
$S^{**}_{k-1} := \langle \rho^{**}_{k-1}, \emptyset \rangle$
$\Sigma^*_{\simeq} := \Sigma^*_{\simeq} \cup \{S^{**}_{k-1}\}$
$\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \cup \{(S^{**}_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
if IsSAT(ρ^{**}_{k-1}) $\neq$ UNSAT **then**
    $\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \cup \{(S, S^{**}_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
end if
return $\langle \Sigma^*_{\simeq}, \rightarrow^*_{\simeq} \rangle$

1: let $S_0, S_1, \ldots, S_n = \tau_0$ and
2: $(k-1, k) = \text{Frontier}(\tau_0)$ in
3: $\Sigma_{\simeq} := (\Sigma_{\simeq} \setminus \{S_{k-1}\}) \cup \{S_{k-1} \wedge \rho, S_{k-1} \wedge \neg\rho\}$
4: $\rightarrow_{\simeq} := (\rightarrow_{\simeq} \setminus \{(S, S_{k-1}) \mid S \in \text{Parents}(S_{k-1})\})$
$\setminus \{(S_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
5: $\rightarrow_{\simeq} := \rightarrow_{\simeq} \cup \{(S, S_{k-1} \wedge \rho) \mid S \in \text{Parents}(S_{k-1})\}$
$\cup \{(S, S_{k-1} \wedge \neg\rho) \mid S \in \text{Parents}(S_{k-1})\}$
$\cup \{(S_{k-1} \wedge \rho, S) \mid S \in \text{Children}(S_{k-1})\}$
$\cup \{(S_{k-1} \wedge \neg\rho, S) \mid S \in (\text{Children}(S_{k-1}) \setminus \{S_k\})\}$

- Disambiguate regions and region predicates
- Spell out what happens

Modifications to refinement

RefineGraph($\rho, \tau_c = \langle S_0, \ldots, S_{k-1}, S_k \rangle, G = \langle \Sigma_{\simeq}, \to_{\simeq} \rangle$)

let $\langle \rho_{k-1}, \mathit{states} \rangle = S_{k-1}$
if $k = 1$ **then**
  return $\langle \Sigma_{\simeq}, \to_{\simeq} \setminus (S_0, S_1) \rangle$
end if
$\Sigma_{\simeq}^{*} := \Sigma_{\simeq} \setminus \{S_{k-1}\}$
$\to_{\simeq}^{*} := \to_{\simeq} \setminus \{(S, S_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\to_{\simeq}^{*} := \to_{\simeq}^{*} \setminus \{(S_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\rho_{k-1}^{*} := \text{Simplify}(\rho_{k-1} \wedge \neg\rho)$
$S_{k-1}^{*} := \langle \rho_{k-1}^{*}, \mathit{states} \rangle$
$\Sigma_{\simeq}^{*} := \Sigma_{\simeq}^{*} \cup \{S_{k-1}^{*}\}$
$\to_{\simeq}^{*} := \to_{\simeq}^{*} \cup \{(S, S_{k-1}^{*}) \mid S \in \text{Parents}(S_{k-1})\}$
$\to_{\simeq}^{*} := \to_{\simeq}^{*} \cup \{(S_{k-1}^{*}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\to_{\simeq}^{*} := \to_{\simeq}^{*} \setminus \{(S_{k-1}^{*}, S_k)\}$
$\rho_{k-1}^{**} := \text{Simplify}(\rho_{k-1} \wedge \rho)$
$S_{k-1}^{**} := \langle \rho_{k-1}^{**}, \emptyset \rangle$
$\Sigma_{\simeq}^{*} := \Sigma_{\simeq}^{*} \cup \{S_{k-1}^{**}\}$
$\to_{\simeq}^{*} := \to_{\simeq}^{*} \cup \{(S_{k-1}^{**}, S) \mid S \in \text{Children}(S_{k-1})\}$
if $\text{IsSAT}(\rho_{k-1}^{**}) \neq \textsc{unsat}$ **then**
  $\to_{\simeq}^{*} := \to_{\simeq}^{*} \cup \{(S, S_{k-1}^{**}) \mid S \in \text{Parents}(S_{k-1})\}$
end if
return $\langle \Sigma_{\simeq}^{*}, \to_{\simeq}^{*} \rangle$

let $S_0, S_1, \ldots, S_n = \tau_0$ and
$(k-1, k) = \text{Frontier}(\tau_0)$ in
$\Sigma_{\simeq} := (\Sigma_{\simeq} \setminus \{S_{k-1}\}) \cup \{S_{k-1} \wedge \rho, S_{k-1} \wedge \neg\rho\}$
$\to_{\simeq} := (\to_{\simeq} \setminus \{(S, S_{k-1}) \mid S \in \text{Parents}(S_{k-1})\})$
$\setminus \{(S_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\to_{\simeq} := \to_{\simeq} \cup \{(S, S_{k-1} \wedge \rho) \mid S \in \text{Parents}(S_{k-1})\}$
$\cup \{(S, S_{k-1} \wedge \neg\rho) \mid S \in \text{Parents}(S_{k-1})\}$
$\cup \{(S_{k-1} \wedge \rho, S) \mid S \in \text{Children}(S_{k-1})\}$
$\cup \{(S_{k-1} \wedge \neg\rho, S) \mid S \in (\text{Children}(S_{k-1}) \setminus \{S_k\})\}$

- Disambiguate regions and region predicates
- Spell out what happens

Modifications to refinement

RefineGraph($\rho, \tau_c = \langle S_0, \ldots, S_{k-1}, S_k \rangle, G = \langle \Sigma_{\simeq}, \to_{\simeq} \rangle$)

let $\langle \rho_{k-1}, \mathit{states} \rangle = S_{k-1}$
if $k = 1$ **then**
  return $\langle \Sigma_{\simeq}, \to_{\simeq} \setminus (S_0, S_1) \rangle$
end if
$\Sigma^*_{\simeq} := \Sigma_{\simeq} \setminus \{S_{k-1}\}$
$\to^*_{\simeq} := \to_{\simeq} \setminus \{(S, S_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\to^*_{\simeq} := \to^*_{\simeq} \setminus \{(S_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\rho^*_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \neg\rho)$
$S^*_{k-1} := \langle \rho^*_{k-1}, \mathit{states} \rangle$
$\Sigma^*_{\simeq} := \Sigma^*_{\simeq} \cup \{S^*_{k-1}\}$
$\to^*_{\simeq} := \to^*_{\simeq} \cup \{(S, S^*_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\to^*_{\simeq} := \to^*_{\simeq} \cup \{(S^*_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\to^*_{\simeq} := \to^*_{\simeq} \setminus \{(S^*_{k-1}, S_k)\}$
$\rho^{**}_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \rho)$
$S^{**}_{k-1} := \langle \rho^{**}_{k-1}, \emptyset \rangle$
$\Sigma^*_{\simeq} := \Sigma^*_{\simeq} \cup \{S^{**}_{k-1}\}$
$\to^*_{\simeq} := \to^*_{\simeq} \cup \{(S^{**}_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
if IsSAT(ρ^{**}_{k-1}) $\neq$ UNSAT **then**
  $\to^*_{\simeq} := \to^*_{\simeq} \cup \{(S, S^{**}_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
end if
return $\langle \Sigma^*_{\simeq}, \to^*_{\simeq} \rangle$

let $S_0, S_1, \ldots, S_n = \tau_0$ and
$(k-1, k) = \text{Frontier}(\tau_0)$ in
$\Sigma_{\simeq} := (\Sigma_{\simeq} \setminus \{S_{k-1}\}) \cup \{S_{k-1} \wedge \rho, S_{k-1} \wedge \neg\rho\}$
$\to_{\simeq} := (\to_{\simeq} \setminus \{(S, S_{k-1}) \mid S \in \text{Parents}(S_{k-1})\})$
$\setminus \{(S_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\to_{\simeq} := \to_{\simeq} \cup \{(S, S_{k-1} \wedge \rho) \mid S \in \text{Parents}(S_{k-1})\}$
$\cup \{(S, S_{k-1} \wedge \neg\rho) \mid S \in \text{Parents}(S_{k-1})\}$
$\cup \{(S_{k-1} \wedge \rho, S) \mid S \in \text{Children}(S_{k-1})\}$
$\cup \{(S_{k-1} \wedge \neg\rho, S) \mid S \in (\text{Children}(S_{k-1}) \setminus \{S_k\})\}$

- Disambiguate regions and region predicates
- Spell out what happens

Modifications to refinement

RefineGraph$(\rho, \tau_c = \langle S_0, \ldots, S_{k-1}, S_k \rangle, G = \langle \Sigma_{\simeq}, \to_{\simeq} \rangle)$

let $\langle \rho_{k-1}, states \rangle = S_{k-1}$
if $k = 1$ **then**
  return $\langle \Sigma_{\simeq}, \to_{\simeq} \setminus (S_0, S_1) \rangle$
end if
$\Sigma^*_{\simeq} := \Sigma_{\simeq} \setminus \{S_{k-1}\}$
$\to^*_{\simeq} := \to_{\simeq} \setminus \{(S, S_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\to^*_{\simeq} := \to^*_{\simeq} \setminus \{(S_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\rho^*_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \neg\rho)$
$S^*_{k-1} := \langle \rho^*_{k-1}, states \rangle$
$\Sigma^*_{\simeq} := \Sigma^*_{\simeq} \cup \{S^*_{k-1}\}$
$\to^*_{\simeq} := \to^*_{\simeq} \cup \{(S, S^*_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\to^*_{\simeq} := \to^*_{\simeq} \cup \{(S^*_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\to^*_{\simeq} := \to^*_{\simeq} \setminus \{(S^*_{k-1}, S_k)\}$
$\rho^{**}_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \rho)$
$S^{**}_{k-1} := \langle \rho^{**}_{k-1}, \emptyset \rangle$
$\Sigma^*_{\simeq} := \Sigma^*_{\simeq} \cup \{S^{**}_{k-1}\}$
$\to^*_{\simeq} := \to^*_{\simeq} \cup \{(S^{**}_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
if IsSAT$(\rho^{**}_{k-1}) \neq$ UNSAT **then**
  $\to^*_{\simeq} := \to^*_{\simeq} \cup \{(S, S^{**}_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
end if
return $\langle \Sigma^*_{\simeq}, \to^*_{\simeq} \rangle$

let $S_0, S_1, \ldots, S_n = \tau_0$ and
$(k-1, k) = \text{Frontier}(\tau_0)$ in
$\Sigma_{\simeq} := (\Sigma_{\simeq} \setminus \{S_{k-1}\}) \cup \{S_{k-1} \wedge \rho, S_{k-1} \wedge \neg\rho\}$
$\to_{\simeq} := (\to_{\simeq} \setminus \{(S, S_{k-1}) \mid S \in \text{Parents}(S_{k-1})\})$
$\setminus \{(S_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\to_{\simeq} := \to_{\simeq} \cup \{(S, S_{k-1} \wedge \rho) \mid S \in \text{Parents}(S_{k-1})\}$
$\cup \{(S, S_{k-1} \wedge \neg\rho) \mid S \in \text{Parents}(S_{k-1})\}$
$\cup \{(S_{k-1} \wedge \rho, S) \mid S \in \text{Children}(S_{k-1})\}$
$\cup \{(S_{k-1} \wedge \neg\rho, S) \mid S \in (\text{Children}(S_{k-1}) \setminus \{S_k\})\}$

- Disambiguate regions and region predicates
- Spell out what happens

Modifications to refinement

RefineGraph($\rho, \tau_c = \langle S_0, \ldots, S_{k-1}, S_k \rangle, G = \langle \Sigma_{\simeq}, \rightarrow_{\simeq} \rangle$)

let $\langle \rho_{k-1}, states \rangle = S_{k-1}$
if $k = 1$ **then**
  return $\langle \Sigma_{\simeq}, \rightarrow_{\simeq} \setminus (S_0, S_1) \rangle$
end if
$\Sigma^*_{\simeq} := \Sigma_{\simeq} \setminus \{S_{k-1}\}$
$\rightarrow^*_{\simeq} := \rightarrow_{\simeq} \setminus \{(S, S_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \setminus \{(S_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\rho^*_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \neg\rho)$
$S^*_{k-1} := \langle \rho^*_{k-1}, states \rangle$
$\Sigma^*_{\simeq} := \Sigma^*_{\simeq} \cup \{S^*_{k-1}\}$
$\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \cup \{(S, S^*_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \cup \{(S^*_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \setminus \{(S^*_{k-1}, S_k)\}$
$\rho^{**}_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \rho)$
$S^{**}_{k-1} := \langle \rho^{**}_{k-1}, \emptyset \rangle$
$\Sigma^*_{\simeq} := \Sigma^*_{\simeq} \cup \{S^{**}_{k-1}\}$
$\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \cup \{(S^{**}_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
if IsSAT(ρ^{**}_{k-1}) $\neq$ UNSAT **then**
  $\rightarrow^*_{\simeq} := \rightarrow^*_{\simeq} \cup \{(S, S^{**}_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
end if
return $\langle \Sigma^*_{\simeq}, \rightarrow^*_{\simeq} \rangle$

let $S_0, S_1, \ldots, S_n = \tau_0$ and
$(k-1, k) = \text{Frontier}(\tau_0)$ in
$\Sigma_{\simeq} := (\Sigma_{\simeq} \setminus \{S_{k-1}\}) \cup \{S_{k-1} \wedge \rho, S_{k-1} \wedge \neg\rho\}$
$\rightarrow_{\simeq} := (\rightarrow_{\simeq} \setminus \{(S, S_{k-1}) \mid S \in \text{Parents}(S_{k-1})\})$
$\setminus \{(S_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\rightarrow_{\simeq} := \rightarrow_{\simeq} \cup \{(S, S_{k-1} \wedge \rho) \mid S \in \text{Parents}(S_{k-1})\}$
$\cup \{(S, S_{k-1} \wedge \neg\rho) \mid S \in \text{Parents}(S_{k-1})\}$
$\cup \{(S_{k-1} \wedge \rho, S) \mid S \in \text{Children}(S_{k-1})\}$
$\cup \{(S_{k-1} \wedge \neg\rho, S) \mid S \in (\text{Children}(S_{k-1}) \setminus \{S_k\})\}$

- Disambiguate regions and region predicates
- Spell out what happens
- Remove incoming edges (remove infeasible regions)

Modifications to refinement

RefineGraph($\rho, \tau_c = \langle S_0, \ldots, S_{k-1}, S_k \rangle, G = \langle \Sigma_{\simeq}, \to_{\simeq} \rangle$)

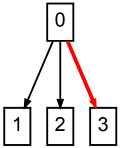
let $\langle \rho_{k-1}, \mathit{states} \rangle = S_{k-1}$
if $k = 1$ **then**
  return $\langle \Sigma_{\simeq}, \to_{\simeq} \setminus (S_0, S_1) \rangle$
end if
$\Sigma^*_{\simeq} := \Sigma_{\simeq} \setminus \{S_{k-1}\}$
$\to^*_{\simeq} := \to_{\simeq} \setminus \{(S, S_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\to^*_{\simeq} := \to^*_{\simeq} \setminus \{(S_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\rho^*_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \neg\rho)$
$S^*_{k-1} := \langle \rho^*_{k-1}, \mathit{states} \rangle$
$\Sigma^*_{\simeq} := \Sigma^*_{\simeq} \cup \{S^*_{k-1}\}$
$\to^*_{\simeq} := \to^*_{\simeq} \cup \{(S, S^*_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
$\to^*_{\simeq} := \to^*_{\simeq} \cup \{(S^*_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\to^*_{\simeq} := \to^*_{\simeq} \setminus \{(S^*_{k-1}, S_k)\}$
$\rho^{**}_{k-1} := \text{Simplify}(\rho_{k-1} \wedge \rho)$
$S^{**}_{k-1} := \langle \rho^{**}_{k-1}, \emptyset \rangle$
$\Sigma^*_{\simeq} := \Sigma^*_{\simeq} \cup \{S^{**}_{k-1}\}$
$\to^*_{\simeq} := \to^*_{\simeq} \cup \{(S^{**}_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
if IsSAT(ρ^{**}_{k-1}) $\neq$ UNSAT **then**
  $\to^*_{\simeq} := \to^*_{\simeq} \cup \{(S, S^{**}_{k-1}) \mid S \in \text{Parents}(S_{k-1})\}$
end if
return $\langle \Sigma^*_{\simeq}, \to^*_{\simeq} \rangle$

let $S_0, S_1, \ldots, S_n = \tau_0$ and
$(k-1, k) = \text{Frontier}(\tau_0)$ in
$\Sigma_{\simeq} := (\Sigma_{\simeq} \setminus \{S_{k-1}\}) \cup \{S_{k-1} \wedge \rho, S_{k-1} \wedge \neg\rho\}$
$\to_{\simeq} := (\to_{\simeq} \setminus \{(S, S_{k-1}) \mid S \in \text{Parents}(S_{k-1})\})$
$\setminus \{(S_{k-1}, S) \mid S \in \text{Children}(S_{k-1})\}$
$\to_{\simeq} := \to_{\simeq} \cup \{(S, S_{k-1} \wedge \rho) \mid S \in \text{Parents}(S_{k-1})\}$
$\cup \{(S, S_{k-1} \wedge \neg\rho) \mid S \in \text{Parents}(S_{k-1})\}$
$\cup \{(S_{k-1} \wedge \rho, S) \mid S \in \text{Children}(S_{k-1})\}$
$\cup \{(S_{k-1} \wedge \neg\rho, S) \mid S \in (\text{Children}(S_{k-1}) \setminus \{S_k\})\}$

- Disambiguate regions and region predicates
- Spell out what happens
- Remove incoming edges (remove infeasible regions)
- Initial region refinement

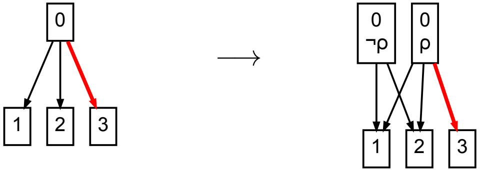
Initial region refinement

- How should the initial region be refined?



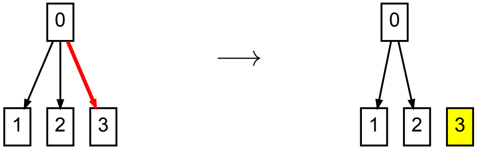
Initial region refinement

- How should the initial region be refined?
  - Like any other region (Multiple initial regions)



Initial region refinement

- How should the initial region be refined?
  - Like any other region (Multiple initial regions)
  - Remove infeasible edge



Initial region refinement

- How should the initial region be refined?
  - Like any other region (Multiple initial regions)
  - Remove infeasible edge

RefineGraph($\rho, \tau_c = \langle S_0, \ldots, S_{k-1}, S_k\rangle, G = \langle \Sigma_{\simeq}, \to_{\simeq}\rangle$)

let $\langle \rho_{k-1}, \mathit{states}\rangle = S_{k-1}$
if $k = 1$ **then**
    return $\langle \Sigma_{\simeq}, \to_{\simeq} \setminus (S_0, S_1)\rangle$
end if
. . .
return $\langle \Sigma^*_{\simeq}, \to^*_{\simeq}\rangle$

Modifications to ExtendFrontier

ExtendFrontier(τ_c, P)

ϕ:= ExecuteSymbolic(τ_c, P)
t := IsSAT(ϕ, P)
if t = UNSAT **then**
  ρ := RefinePred(τ_c)
else
  ρ := *true*
end if
return $\langle t, \rho \rangle$

ExtendFrontier(τ, F, P)

$(k-1, k)$:= Frontier(τ)
$\langle \phi_1, S, \phi_2 \rangle$:= ExecuteSymbolic(τ, F, P)
t := IsSAT(ϕ_1, S, ϕ_2, P)
if $t = \epsilon$ **then**
  ρ := RefinePred(S,τ)
else
  ρ := *true*
end if
return $\langle t, \rho \rangle$

Modifications to ExtendFrontier

ExtendFrontier(τ_c, P)

```
ϕ := ExecuteSymbolic(τ_c, P)
t := IsSAT(ϕ, P)
if t = UNSAT then
    ρ := RefinePred(τ_c)
else
    ρ := true
end if
return ⟨t, ρ⟩
```

ExtendFrontier(τ, F, P)

```
(k − 1, k) := Frontier(τ)
⟨ϕ_1, S, ϕ_2⟩ := ExecuteSymbolic(τ, F, P)
t := IsSAT(ϕ_1, S, ϕ_2, P)
if t = ε then
    ρ := RefinePred(S, τ)
else
    ρ := true
end if
return ⟨t, ρ⟩
```

- Consistent naming of τ and τ_o/τ_c

Modifications to ExtendFrontier

ExtendFrontier(τ_c, P)

ϕ:= ExecuteSymbolic(τ_c, P)
t := IsSAT(ϕ, P)
if t = UNSAT **then**
    ρ := RefinePred(τ_c)
else
    ρ := *true*
end if
return $\langle t, \rho \rangle$

ExtendFrontier(τ, F, P)

$(k-1, k)$:= Frontier(τ)
$\langle \phi_1, S, \phi_2 \rangle$:= ExecuteSymbolic(τ, F, P)
t := IsSAT(ϕ_1, S, ϕ_2, P)
if $t = \epsilon$ **then**
    ρ := RefinePred(S,τ)
else
    ρ := *true*
end if
return $\langle t, \rho \rangle$

- Consistent naming of τ and τ_o/τ_c
- Removal of Frontier since $k-1$ and k is unused

Modifications to ExtendFrontier

ExtendFrontier(τ_c, P)

```
ϕ := ExecuteSymbolic(τ_c, P)
t := IsSAT(ϕ, P)
if t = UNSAT then
    ρ := RefinePred(τ_c)
else
    ρ := true
end if
return ⟨t, ρ⟩
```

ExtendFrontier(τ, F, P)

```
(k − 1, k) := Frontier(τ)
⟨ϕ1, S, ϕ2⟩ := ExecuteSymbolic(τ, F, P)
t := IsSAT(ϕ1, S, ϕ2, P)
if t = ϵ then
    ρ := RefinePred(S,τ)
else
    ρ := true
end if
return ⟨t, ρ⟩
```

- Consistent naming of τ and τ_o/τ_c
- Removal of Frontier since $k-1$ and k is unused
- Simpler path constraint $(\phi_1 \wedge S \wedge \phi_2)$

Modifications to ExtendFrontier

ExtendFrontier(τ_c, P)

ϕ:= ExecuteSymbolic(τ_c, P)
t := IsSAT(ϕ, P)
if $t = \textsc{unsat}$ **then**
  ρ := RefinePred(τ_c)
else
  ρ := *true*
end if
return $\langle t, \rho \rangle$

ExtendFrontier(τ, F, P)

$(k-1, k)$:= Frontier(τ)
$\langle \phi_1, S, \phi_2 \rangle$:= ExecuteSymbolic(τ, F, P)
t := IsSAT(ϕ_1, S, ϕ_2, P)
if $t = \epsilon$ **then**
  ρ := RefinePred(S,τ)
else
  ρ := *true*
end if
return $\langle t, \rho \rangle$

- Consistent naming of τ and τ_o/τ_c
- Removal of Frontier since $k-1$ and k is unused
- Simpler path constraint $(\phi_1 \wedge S \wedge \phi_2)$
- Use $\textsc{unsat}$ to indicate unsatisfiability, instead of ϵ

Modifications to ExecuteSymbolic

ExecuteSymbolic($\tau_c = \langle S_0, \ldots, S_k \rangle, P$)

let $\langle \rho_0, \_ \rangle = S_0$
$\mathcal{S} := [v \mapsto v_0 \mid v \in \text{params}(P)]$
$\phi := \text{SymbolicEval}(\rho_0, \mathcal{S})$
for $i = 0$ **to** $k - 1$ **do**
  $op := \text{Op}(S_i, S_{i+1})$
  match *op*
    case($v := e$):
      $\mathcal{S} := \mathcal{S}[v \mapsto \text{SymbolicEval}(e, \mathcal{S})]$
    case(*assumec*):
      $\phi := \phi \wedge \text{SymbolicEval}(c, \mathcal{S})$
  let $\langle \rho_{i+1}, \_ \rangle = S_{i+1}$
  $\phi := \phi \wedge \text{SymbolicEval}(\rho_{i+1}, \mathcal{S})$
end for
return ϕ

ExecuteSymbolic(τ_0, F, P)

$(k-1, k) := \text{Frontier}(\tau_0 = \langle S_0, S_1, \ldots, S_n \rangle)$
$S := [v \mapsto v_0 \mid *v \in \mathit{inputs}(P)]$
$\phi_1 := \text{SymbolicEval}(S_0, S)$
$\phi_2 := \mathit{true}$
$i := 0$
while $i \neq k - 1$ **do**
  $op := \lambda(\text{Edge}(S_i, S_{i+1}))$
  match *op*
    case($*$m=e):
      $S := S + [\text{SymbolicEval}(m, S) \mapsto \text{SymbolicEval}(e, S)]$
    case(if e goto l):
      $\phi_1 := \phi_1 \wedge \text{SymbolicEval}(e, S)$
  $i := i + 1$
  $\phi_1 := \phi_1 \wedge \text{SymbolicEval}(S_i, S)$
end while
$op := \lambda(\text{Edge}(S_{k-1}, S_k))$
match *op*
  case($*$m=e):
    $\phi_2 := \phi_2 \wedge *(\text{SymbolicEval}(m, S)) = \text{SymbolicEval}(e, S)$
    $S' := S + [\text{SymbolicEval}(m, S) \mapsto \text{SymbolicEval}(e, S)]$
  case(if e goto l):
    $\phi_2 := \phi_2 \wedge \text{SymbolicEval}(e, S)$
    $S' = S$
$\phi_2 := \phi_2 \wedge \text{SymbolicEval}(S_k, S')$
return $\langle \phi_1, S, \phi_2 \rangle$

Modifications to ExecuteSymbolic

ExecuteSymbolic($\tau_c = \langle S_0, \ldots, S_k \rangle, P$)

let $\langle \rho_0, \_ \rangle = S_0$
$\mathcal{S} := [v \mapsto v_0 \mid v \in \text{params}(P)]$
$\phi := \text{SymbolicEval}(\rho_0, \mathcal{S})$
for $i = 0$ **to** $k - 1$ **do**
    $op := \text{Op}(S_i, S_{i+1})$
    match op
        case($v := e$):
            $\mathcal{S} := \mathcal{S}[v \mapsto \text{SymbolicEval}(e, \mathcal{S})]$
        case(*assumec*):
            $\phi := \phi \wedge \text{SymbolicEval}(c, \mathcal{S})$
    let $\langle \rho_{i+1}, \_ \rangle = S_{i+1}$
    $\phi := \phi \wedge \text{SymbolicEval}(\rho_{i+1}, \mathcal{S})$
end for
return ϕ

- $\phi = \phi_1 \wedge \phi 2$

ExecuteSymbolic(τ_0, F, P)

$(k-1, k) := \text{Frontier}(\tau_0 = \langle S_0, S_1, \ldots, S_n \rangle)$
$S := [v \mapsto v_0 \mid *v \in \mathit{inputs}(P)]$
$\phi_1 := \text{SymbolicEval}(S_0, S)$
$\phi_2 := \mathit{true}$
$i := 0$
while $i \neq k - 1$ **do**
    $op := \lambda(\text{Edge}(S_i, S_{i+1}))$
    match op
        case($*$m=e):
            $S := S + [\text{SymbolicEval}(m, S) \mapsto \text{SymbolicEval}(e, S)]$
        case(if e goto l):
            $\phi_1 := \phi_1 \wedge \text{SymbolicEval}(e, S)$
    $i := i + 1$
    $\phi_1 := \phi_1 \wedge \text{SymbolicEval}(S_i, S)$
end while
$op := \lambda(\text{Edge}(S_{k-1}, S_k))$
match op
    case($*$m=e):
        $\phi_2 := \phi_2 \wedge *(\text{SymbolicEval}(m, S)) = \text{SymbolicEval}(e, S)$
        $S' := S + [\text{SymbolicEval}(m, S) \mapsto \text{SymbolicEval}(e, S)]$
    case(if e goto l):
        $\phi_2 := \phi_2 \wedge \text{SymbolicEval}(e, S)$
        $S' = S$
$\phi_2 := \phi_2 \wedge \text{SymbolicEval}(S_k, S')$
return $\langle \phi_1, S, \phi_2 \rangle$

Modifications to ExecuteSymbolic

ExecuteSymbolic($\tau_c = \langle S_0, \ldots, S_k \rangle, P$)

let $\langle \rho_0, \_ \rangle = S_0$
$\mathcal{S} := [v \mapsto v_0 \mid v \in \text{params}(P)]$
$\phi := \text{SymbolicEval}(\rho_0, \mathcal{S})$
for $i = 0$ **to** $k - 1$ **do**
    $op := \text{Op}(S_i, S_{i+1})$
    match op
        case($v := e$):
            $\mathcal{S} := \mathcal{S}[v \mapsto \text{SymbolicEval}(e, \mathcal{S})]$
        case(*assumec*):
            $\phi := \phi \wedge \text{SymbolicEval}(c, \mathcal{S})$
    let $\langle \rho_{i+1}, \_ \rangle = S_{i+1}$
    $\phi := \phi \wedge \text{SymbolicEval}(\rho_{i+1}, \mathcal{S})$
end for
return ϕ

- $\phi = \phi_1 \wedge \phi 2$
- Remove case for assigning to dereferenced pointer

ExecuteSymbolic(τ_0, F, P)

$(k-1, k) := \text{Frontier}(\tau_0 = \langle S_0, S_1, \ldots, S_n \rangle)$
$S := [v \mapsto v_0 \mid *v \in \textit{inputs}(P)]$
$\phi_1 := \text{SymbolicEval}(S_0, S)$
$\phi_2 := \textit{true}$
$i := 0$
while $i \neq k - 1$ **do**
    $op := \lambda(\text{Edge}(S_i, S_{i+1}))$
    match op
        case($*\text{m}=\text{e}$):
            $S := S + [\text{SymbolicEval}(m, S) \mapsto \text{SymbolicEval}(e, S)]$
        case(if e goto l):
            $\phi_1 := \phi_1 \wedge \text{SymbolicEval}(e, S)$
    $i := i + 1$
    $\phi_1 := \phi_1 \wedge \text{SymbolicEval}(S_i, S)$
end while
$op := \lambda(\text{Edge}(S_{k-1}, S_k))$
match op
    case($*\text{m}=\text{e}$):
        $\phi_2 := \phi_2 \wedge *(\text{SymbolicEval}(m, S)) = \text{SymbolicEval}(e, S)$
        $S' := S + [\text{SymbolicEval}(m, S) \mapsto \text{SymbolicEval}(e, S)]$
    case(if e goto l):
        $\phi_2 := \phi_2 \wedge \text{SymbolicEval}(e, S)$
        $S' = S$
$\phi_2 := \phi_2 \wedge \text{SymbolicEval}(S_k, S')$
return $\langle \phi_1, S, \phi_2 \rangle$

Modifications to ExecuteSymbolic

ExecuteSymbolic($\tau_c = \langle S_0, \ldots, S_k \rangle, P$)

let $\langle \rho_0, \_ \rangle = S_0$
$\mathcal{S} := [v \mapsto v_0 \mid v \in \text{params}(P)]$
$\phi := \text{SymbolicEval}(\rho_0, \mathcal{S})$
for $i = 0$ **to** $k - 1$ **do**
  $op := \text{Op}(S_i, S_{i+1})$
  match op
    case($v := e$):
      $\mathcal{S} := \mathcal{S}[v \mapsto \text{SymbolicEval}(e, \mathcal{S})]$
    case(*assumec*):
      $\phi := \phi \wedge \text{SymbolicEval}(c, \mathcal{S})$
  let $\langle \rho_{i+1}, \_ \rangle = S_{i+1}$
  $\phi := \phi \wedge \text{SymbolicEval}(\rho_{i+1}, \mathcal{S})$
end for
return ϕ

- $\phi = \phi_1 \wedge \phi 2$
- Remove case for assigning to dereferenced pointer
  - No need to return the symbolic map

ExecuteSymbolic(τ_0, F, P)

$(k-1, k) := \text{Frontier}(\tau_0 = \langle S_0, S_1, \ldots, S_n \rangle)$
$S := [v \mapsto v_0 \mid *v \in \textit{inputs}(P)]$
$\phi_1 := \text{SymbolicEval}(S_0, S)$
$\phi_2 := \textit{true}$
$i := 0$
while $i \neq k - 1$ **do**
  $op := \lambda(\text{Edge}(S_i, S_{i+1}))$
  match op
    case($*$m=e):
      $S := S + [\text{SymbolicEval}(m, S) \mapsto \text{SymbolicEval}(e, S)]$
    case(if e goto l):
      $\phi_1 := \phi_1 \wedge \text{SymbolicEval}(e, S)$
  $i := i + 1$
  $\phi_1 := \phi_1 \wedge \text{SymbolicEval}(S_i, S)$
end while
$op := \lambda(\text{Edge}(S_{k-1}, S_k))$
match op
  case($*$m=e):
    $\phi_2 := \phi_2 \wedge *(\text{SymbolicEval}(m, S)) = \text{SymbolicEval}(e, S)$
    $S' := S + [\text{SymbolicEval}(m, S) \mapsto \text{SymbolicEval}(e, S)]$
  case(if e goto l):
    $\phi_2 := \phi_2 \wedge \text{SymbolicEval}(e, S)$
    $S' = S$
$\phi_2 := \phi_2 \wedge \text{SymbolicEval}(S_k, S')$
return $\langle \phi_1, S, \phi_2 \rangle$

Modifications to ExecuteSymbolic

ExecuteSymbolic($\tau_c = \langle S_0, \ldots, S_k \rangle, P$)

let $\langle \rho_0, \_ \rangle = S_0$
$\mathcal{S} := [v \mapsto v_0 \mid v \in \text{params}(P)]$
$\phi := \text{SymbolicEval}(\rho_0, \mathcal{S})$
for $i = 0$ **to** $k - 1$ **do**
  $op := \text{Op}(S_i, S_{i+1})$
  match op
    case($v := e$):
      $\mathcal{S} := \mathcal{S}[v \mapsto \text{SymbolicEval}(e, \mathcal{S})]$
    case(*assumec*):
      $\phi := \phi \wedge \text{SymbolicEval}(c, \mathcal{S})$
  let $\langle \rho_{i+1}, \_ \rangle = S_{i+1}$
  $\phi := \phi \wedge \text{SymbolicEval}(\rho_{i+1}, \mathcal{S})$
end for
return ϕ

- $\phi = \phi_1 \wedge \phi 2$
- Remove case for assigning to dereferenced pointer
  - No need to return the symbolic map
  - No special case for the last iteration

ExecuteSymbolic(τ_0, F, P)

$(k-1, k) := \text{Frontier}(\tau_0 = \langle S_0, S_1, \ldots, S_n \rangle)$
$S := [v \mapsto v_0 \mid *v \in \mathit{inputs}(P)]$
$\phi_1 := \text{SymbolicEval}(S_0, S)$
$\phi_2 := \mathit{true}$
$i := 0$
while $i \neq k - 1$ **do**
  $op := \lambda(\text{Edge}(S_i, S_{i+1}))$
  match op
    case($*$m=e):
      $S := S + [\text{SymbolicEval}(m, S) \mapsto \text{SymbolicEval}(e, S)]$
    case(if e goto l):
      $\phi_1 := \phi_1 \wedge \text{SymbolicEval}(e, S)$
  $i := i + 1$
  $\phi_1 := \phi_1 \wedge \text{SymbolicEval}(S_i, S)$
end while
$op := \lambda(\text{Edge}(S_{k-1}, S_k))$
match op
  case($*$m=e):
    $\phi_2 := \phi_2 \wedge *(\text{SymbolicEval}(m, S)) = \text{SymbolicEval}(e, S)$
    $S' := S + [\text{SymbolicEval}(m, S) \mapsto \text{SymbolicEval}(e, S)]$
  case(if e goto l):
    $\phi_2 := \phi_2 \wedge \text{SymbolicEval}(e, S)$
    $S' = S$
$\phi_2 := \phi_2 \wedge \text{SymbolicEval}(S_k, S')$
return $\langle \phi_1, S, \phi_2 \rangle$

Modifications to ExecuteSymbolic

ExecuteSymbolic($\tau_c = \langle S_0, \ldots, S_k \rangle, P$)

- **let** $\langle \rho_0, \_ \rangle = S_0$
- $\mathcal{S} := [v \mapsto v_0 \mid v \in \text{params}(P)]$
- $\phi := \text{SymbolicEval}(\rho_0, \mathcal{S})$
- **for** $i = 0$ **to** $k - 1$ **do**
  - $op := \text{Op}(S_i, S_{i+1})$
  - **match** op
    - **case**($v := e$):
      - $\mathcal{S} := \mathcal{S}[v \mapsto \text{SymbolicEval}(e, \mathcal{S})]$
    - **case**($assume c$):
      - $\phi := \phi \wedge \text{SymbolicEval}(c, \mathcal{S})$
  - **let** $\langle \rho_{i+1}, \_ \rangle = S_{i+1}$
  - $\phi := \phi \wedge \text{SymbolicEval}(\rho_{i+1}, \mathcal{S})$
- **end for**
- **return** ϕ

- $\phi = \phi_1 \wedge \phi 2$
- Remove case for assigning to dereferenced pointer
  - No need to return the symbolic map
  - No special case for the last iteration
- Add case for regular assignment

ExecuteSymbolic(τ_0, F, P)

- $(k-1, k) := \text{Frontier}(\tau_0 = \langle S_0, S_1, \ldots, S_n \rangle)$
- $S := [v \mapsto v_0 \mid *v \in inputs(P)]$
- $\phi_1 := \text{SymbolicEval}(S_0, S)$
- $\phi_2 := true$
- $i := 0$
- **while** $i \neq k - 1$ **do**
  - $op := \lambda(\text{Edge}(S_i, S_{i+1}))$
  - **match** op
    - **case**($*$m=e):
      - $S := S + [\text{SymbolicEval}(m, S) \mapsto \text{SymbolicEval}(e, S)]$
    - **case**(if e goto l):
      - $\phi_1 := \phi_1 \wedge \text{SymbolicEval}(e, S)$
  - $i := i + 1$
  - $\phi_1 := \phi_1 \wedge \text{SymbolicEval}(S_i, S)$
- **end while**
- $op := \lambda(\text{Edge}(S_{k-1}, S_k))$
- **match** op
  - **case**($*$m=e):
    - $\phi_2 := \phi_2 \wedge *(\text{SymbolicEval}(m, S)) = \text{SymbolicEval}(e, S)$
    - $S' := S + [\text{SymbolicEval}(m, S) \mapsto \text{SymbolicEval}(e, S)]$
  - **case**(if e goto l):
    - $\phi_2 := \phi_2 \wedge \text{SymbolicEval}(e, S)$
    - $S' = S$
- $\phi_2 := \phi_2 \wedge \text{SymbolicEval}(S_k, S')$
- **return** $\langle \phi_1, S, \phi_2 \rangle$

Modifications to ExecuteSymbolic

ExecuteSymbolic($\tau_c = \langle S_0, \ldots, S_k \rangle$, P)

let $\langle \rho_0, \_ \rangle = S_0$

$\mathcal{S} := [v \mapsto v_0 \mid v \in \text{params}(P)]$

$\phi := \text{SymbolicEval}(\rho_0, \mathcal{S})$

for $i = 0$ **to** $k - 1$ **do**

  $op := \text{Op}(S_i, S_{i+1})$

  match op

    case($v := e$):

      $\mathcal{S} := \mathcal{S}[v \mapsto \text{SymbolicEval}(e, \mathcal{S})]$

    case(*assumec*):

      $\phi := \phi \wedge \text{SymbolicEval}(c, \mathcal{S})$

  let $\langle \rho_{i+1}, \_ \rangle = S_{i+1}$

  $\phi := \phi \wedge \text{SymbolicEval}(\rho_{i+1}, \mathcal{S})$

end for

return ϕ

- $\phi = \phi_1 \wedge \phi 2$
- Remove case for assigning to dereferenced pointer
  - No need to return the symbolic map
  - No special case for the last iteration
- Add case for regular assignment
- Simpler handling of trace

ExecuteSymbolic(τ_0, F, P)

$(k-1, k) := \text{Frontier}(\tau_0 = \langle S_0, S_1, \ldots, S_n \rangle)$

$S := [v \mapsto v_0 \mid *v \in \mathit{inputs}(P)]$

$\phi_1 := \text{SymbolicEval}(S_0, S)$

$\phi_2 := \mathit{true}$

$i := 0$

while $i \neq k - 1$ **do**

  $op := \lambda(\text{Edge}(S_i, S_{i+1}))$

  match op

    case($*$m=e):

      $S := S + [\text{SymbolicEval}(m, S) \mapsto \text{SymbolicEval}(e, S)]$

    case(if e goto l):

      $\phi_1 := \phi_1 \wedge \text{SymbolicEval}(e, S)$

  $i := i + 1$

  $\phi_1 := \phi_1 \wedge \text{SymbolicEval}(S_i, S)$

end while

$op := \lambda(\text{Edge}(S_{k-1}, S_k))$

match op

  case($*$m=e):

    $\phi_2 := \phi_2 \wedge *(\text{SymbolicEval}(m, S)) = \text{SymbolicEval}(e, S)$

    $S' := S + [\text{SymbolicEval}(m, S) \mapsto \text{SymbolicEval}(e, S)]$

  case(if e goto l):

    $\phi_2 := \phi_2 \wedge \text{SymbolicEval}(e, S)$

    $S' = S$

$\phi_2 := \phi_2 \wedge \text{SymbolicEval}(S_k, S')$

return $\langle \phi_1, S, \phi_2 \rangle$

Modifications to RefinePred

RefinePred($\tau_c = \langle S_0, \ldots, S_{k-1}, S_k \rangle$)

let $\langle \_, states_{k-1} \rangle = S_{k-1}$
let $\langle \rho_k, \_ \rangle = S_k$
$op := \mathrm{Op}(S_{k-1}, S_k)$
if *op* **matches** assume c **then**
    if $k > 1 \wedge \forall s \in states_{k-1} : \mathrm{Eval}(\neg \rho_k, s) = \mathrm{true}$ **then**
        return ρ_k
    end if
end if
return $\mathrm{WP}(op, \rho_k)$

WP(op, p)

match *op*
    case(v := e):
        return $p[\mathrm{e}/\mathrm{V}]$
    case(assume c):
        return $\mathbf{C} \wedge p$

RefinePred(S,τ_o)

$(k-1, k) := \mathrm{Frontier}(\tau_o = \langle S_0, S_1, \ldots, S_m \rangle)$
$op := \lambda(\mathrm{Edge}(S_{k-1}, S_k))$
$\alpha := \mathrm{Aliases}(S, op, S_k)$
return $\mathrm{WP}_\alpha(op, S_k)$

$$WP_\alpha(op, \phi) = \neg(\alpha \wedge \neg(\alpha \wedge WP(op, \phi)))$$

Modifications to RefinePred

RefinePred($\tau_c = \langle S_0, \ldots, S_{k-1}, S_k \rangle$)

let $\langle \_, states_{k-1} \rangle = S_{k-1}$
let $\langle \rho_k, \_ \rangle = S_k$
$op := \mathrm{Op}(S_{k-1}, S_k)$
if op **matches** assume c **then**
    if $k > 1 \wedge \forall s \in states_{k-1} : \mathrm{Eval}(\neg \rho_k, s) = \mathrm{true}$ **then**
        return ρ_k
    end if
end if
return WP(op, ρ_k)

WP(op, p)

match op
    case(v := e):
        return $p[\mathrm{e}/\mathrm{V}]$
    case(assume c):
        return $\mathrm{C} \wedge p$

RefinePred(S,τ_o)

$(k-1, k) := \mathrm{Frontier}(\tau_o = \langle S_0, S_1, \ldots, S_m \rangle)$
$op := \lambda(\mathrm{Edge}(S_{k-1}, S_k))$
$\alpha := \mathrm{Aliases}(S, op, S_k)$
return $\mathrm{WP}_\alpha(op, S_k)$

$$WP_\alpha(op, \phi) = \neg(\alpha \wedge \neg(\alpha \wedge WP(op, \phi)))$$

- Provide implementation for WP

Modifications to RefinePred

RefinePred($\tau_c = \langle S_0, \dots, S_{k-1}, S_k \rangle$)

let $\langle \_, states_{k-1} \rangle = S_{k-1}$
let $\langle \rho_k, \_ \rangle = S_k$
$op :=$ Op(S_{k-1}, S_k)
if op **matches** assume c **then**
  if $k > 1 \wedge \forall s \in states_{k-1}$: Eval($\neg\rho_k, s$) $=$ true **then**
    return ρ_k
  end if
end if
return WP(op, ρ_k)

WP(op, p)

match op
  case(v := e):
    return $p[\mathrm{e}/\mathrm{v}]$
  case(assume c):
    return $\mathbf{C} \wedge p$

RefinePred(S,τ_o)

$(k-1, k) :=$ Frontier($\tau_o = \langle S_0, S_1, \dots, S_m \rangle$)
$op := \lambda$(Edge(S_{k-1}, S_k))
$\alpha :=$Aliases(S, op, S_k)
return WP$_\alpha$(op, S_k)

$$WP_\alpha(op, \phi) = \neg(\alpha \wedge \neg(\alpha \wedge WP(op, \phi)))$$

- Provide implementation for WP
- Remove α

Modifications to RefinePred

RefinePred($\tau_c = \langle S_0, \ldots, S_{k-1}, S_k \rangle$)

let $\langle \_, states_{k-1} \rangle = S_{k-1}$
let $\langle \rho_k, \_ \rangle = S_k$
$op :=$ Op(S_{k-1}, S_k)
if op **matches** assume c **then**
    if $k > 1 \land \forall s \in states_{k-1} : \text{Eval}(\neg \rho_k, s) = \text{true}$ **then**
        return ρ_k
    end if
end if
return WP(op, ρ_k)

WP(op, p)

match op
    case(v := e):
        return $p[\mathrm{e}/\mathrm{v}]$
    case(assume c):
        return $\mathbf{C} \land p$

RefinePred(S, τ_o)

$(k-1, k) :=$ Frontier($\tau_o = \langle S_0, S_1, \ldots, S_m \rangle$)
$op := \lambda$(Edge(S_{k-1}, S_k))
$\alpha :=$Aliases(S, op, S_k)
return WP$_\alpha$(op, S_k)

$$WP_\alpha(op, \phi) = \neg(\alpha \land \neg(\alpha \land WP(op, \phi)))$$

- Provide implementation for WP
- Remove α
- Simpler trace

Modifications to RefinePred

RefinePred($\tau_c = \langle S_0, \ldots, S_{k-1}, S_k \rangle$)

let $\langle \_, states_{k-1} \rangle = S_{k-1}$
let $\langle \rho_k, \_ \rangle = S_k$
$op := \mathrm{Op}(S_{k-1}, S_k)$
if op **matches** assume c **then**
  if $k > 1 \wedge \forall s \in states_{k-1} : \mathrm{Eval}(\neg \rho_k, s) = \mathrm{true}$ **then**
    return ρ_k
  end if
end if
return WP(op, ρ_k)

WP(op, p)

match op
  case(v := e):
    return $p[\mathrm{e}/\mathrm{v}]$
  case(assume c):
    return $\mathbf{C} \wedge p$

RefinePred(S,τ_o)

$(k-1, k) := \mathrm{Frontier}(\tau_o = \langle S_0, S_1, \ldots, S_m \rangle)$
$op := \lambda(\mathrm{Edge}(S_{k-1}, S_k))$
$\alpha :=\mathrm{Aliases}(S, op, S_k)$
return $\mathrm{WP}_\alpha(op, S_k)$

$WP_\alpha(op, \phi) = \neg(\alpha \wedge \neg(\alpha \wedge WP(op, \phi)))$

- Provide implementation for WP
- Remove α
- Simpler trace
- Add the loop optimization

Loop optimization

1: **let** $\langle \_, states_{k-1} \rangle = S_{k-1}$
2: **let** $\langle \rho_k, \_ \rangle = S_k$
3: $op := \mathrm{Op}(S_{k-1}, S_k)$
4: **if** op **matches** assume c **then**
5: **if** $\forall s \in states_{k-1} : \mathrm{Eval}(\neg\rho_k, s) = \mathrm{true}$ **then**
6: **return** ρ_k
7: **end if**
8: **end if**
9: **return** $\mathrm{WP}(op, \rho_k)$

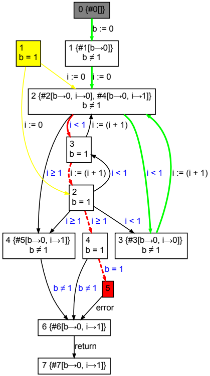
```
void test()
{
  int b = 0;
  int i = 0;
  while (i < 1)
    i++;
  if (b == 1)
    error;
}
```

- Loop optimization ignores assume on edge

Loop optimization

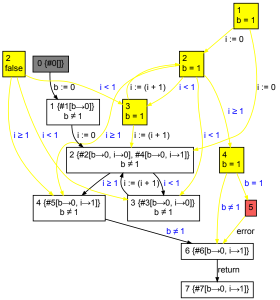
let $\langle \_, states_{k-1} \rangle = S_{k-1}$
let $\langle \rho_k, \_ \rangle = S_k$
$op := \mathrm{Op}(S_{k-1}, S_k)$
if op **matches** assume c **then**
  if $\forall s \in states_{k-1} : \mathrm{Eval}(\neg\rho_k, s) = \mathrm{true}$ **then**
    return ρ_k
  end if
end if
return WP(op, ρ_k)

- Loop optimization ignores assume on edge



Loop optimization

let $\langle \_, states_{k-1} \rangle = S_{k-1}$
let $\langle \rho_k, \_ \rangle = S_k$
$op := \mathrm{Op}(S_{k-1}, S_k)$
if op **matches** assume c **then**
    if $\forall s \in states_{k-1} : \mathrm{Eval}(\neg \rho_k, s) = \mathrm{true}$ **then**
        return ρ_k
    end if
end if
return $\mathrm{WP}(op, \rho_k)$



- Loop optimization ignores assume on edge

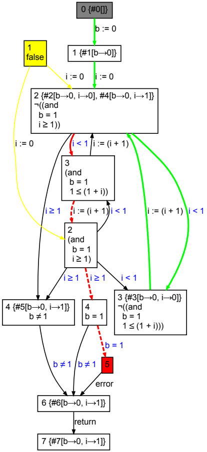
Loop optimization

```
let ⟨_, states_{k-1}⟩ = S_{k-1}
let ⟨ρ_k, _⟩ = S_k
op := Op(S_{k-1}, S_k)
if op matches assume c then
    if k > 1 ∧ ∀s ∈ states_{k-1} : Eval(¬ρ_k, s) = true then
        return ρ_k
    end if
end if
return WP(op, ρ_k)
```

- Loop optimization ignores assume on edge
- Disable loop optimization for initial region

Loop optimization

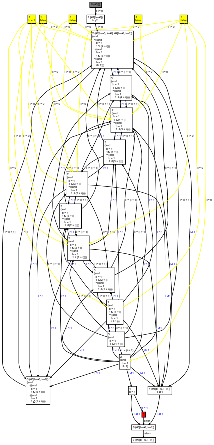
let $\langle \_, states_{k-1} \rangle = S_{k-1}$
let $\langle \rho_k, \_ \rangle = S_k$
$op := \mathrm{Op}(S_{k-1}, S_k)$
if op **matches** assume c **then**
  if $k > 1 \wedge \forall s \in states_{k-1} : \mathrm{Eval}(\neg \rho_k, s) = \mathrm{true}$ **then**
    return ρ_k
  end if
end if
return $\mathrm{WP}(op, \rho_k)$



- Loop optimization ignores assume on edge
- Disable loop optimization for initial region
- Problem: Infinite refines, due to irrelevant predicates

Loop optimization

let $\langle \_, states_{k-1} \rangle = S_{k-1}$
let $\langle \rho_k, \_ \rangle = S_k$
$op := \mathrm{Op}(S_{k-1}, S_k)$
if *op* **matches** assume c **then**
    if $k > 1 \wedge \forall s \in states_{k-1} : \mathrm{Eval}(\neg \rho_k, s) = \text{true}$ **then**
        return ρ_k
    end if
end if
return $\mathrm{WP}(op, \rho_k)$



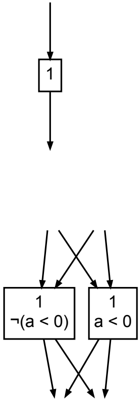
- Loop optimization ignores assume on edge
- Disable loop optimization for initial region
- Problem: Infinite refines, due to irrelevant predicates

Optimizations overview

- Split regions initially
- SP heuristic (Loop optimization)
- CD heuristic
- Interprocedural
- Other

Split regions initially

Split the regions in the original region graph, based on conditionals that are needed to reach the error statement.



CD heuristic

- Preprocess program and set all assumes that are not relevant, based on conditionals that are needed to reach the error statement, to true.

CD heuristic

- Preprocess program and set all assumes that are not relevant, based on conditionals that are needed to reach the error statement, to true.
  - If DASH can prove it correct, then we are done.

CD heuristic

- Preprocess program and set all assumes that are not relevant, based on conditionals that are needed to reach the error statement, to true.
  - If DASH can prove it correct, then we are done.
  - If not, run a test to see if it is a real error.

CD heuristic

- Preprocess program and set all assumes that are not relevant, based on conditionals that are needed to reach the error statement, to true.
  - If DASH can prove it correct, then we are done.
  - If not, run a test to see if it is a real error.
  - Else add back assumes for the found error trace and then try again.

Interprocedural optimizations

1. Compute overapproximation (modification analysis) of all procedures, for the values that can be modified, check then if there is any chance that S_k can be satisfied, e.i. the correct variables are modified.

Interprocedural optimizations

1. Compute overapproximation (modification analysis) of all procedures, for the values that can be modified, check then if there is any chance that S_k can be satisfied, e.i. the correct variables are modified.
2. Cache summaries for procedure analysis

Interprocedural optimizations

1. Compute overapproximation (modification analysis) of all procedures, for the values that can be modified, check then if there is any chance that S_k can be satisfied, e.i. the correct variables are modified.
2. Cache summaries for procedure analysis
3. Regular analysis

Other

- Test optimering

Other

- Test optimering
  - Limit number of steps after frontier

Other

- Test optimering
  - Limit number of steps after frontier
  - Store delta states, to save space on storing states - very important for large programs

Other

- Test optimering
  - Limit number of steps after frontier
  - Store delta states, to save space on storing states - very important for large programs
- Model "external" code

Other

- Test optimering
  - Limit number of steps after frontier
  - Store delta states, to save space on storing states - very important for large programs
- Model "external" code
  - DASH needs to inspect the code

Other

- Test optimering
  - Limit number of steps after frontier
  - Store delta states, to save space on storing states - very important for large programs
- Model "external" code
  - DASH needs to inspect the code
  - Creating stub is difficult, must behave exactly like the modeled code.

Other

- Test optimering
  - Limit number of steps after frontier
  - Store delta states, to save space on storing states - very important for large programs
- Model "external" code
  - DASH needs to inspect the code
  - Creating stub is difficult, must behave exactly like the modeled code.
  - For Java this is native methods