Relative Completeness of Abstraction Refinement for Software Model Checking

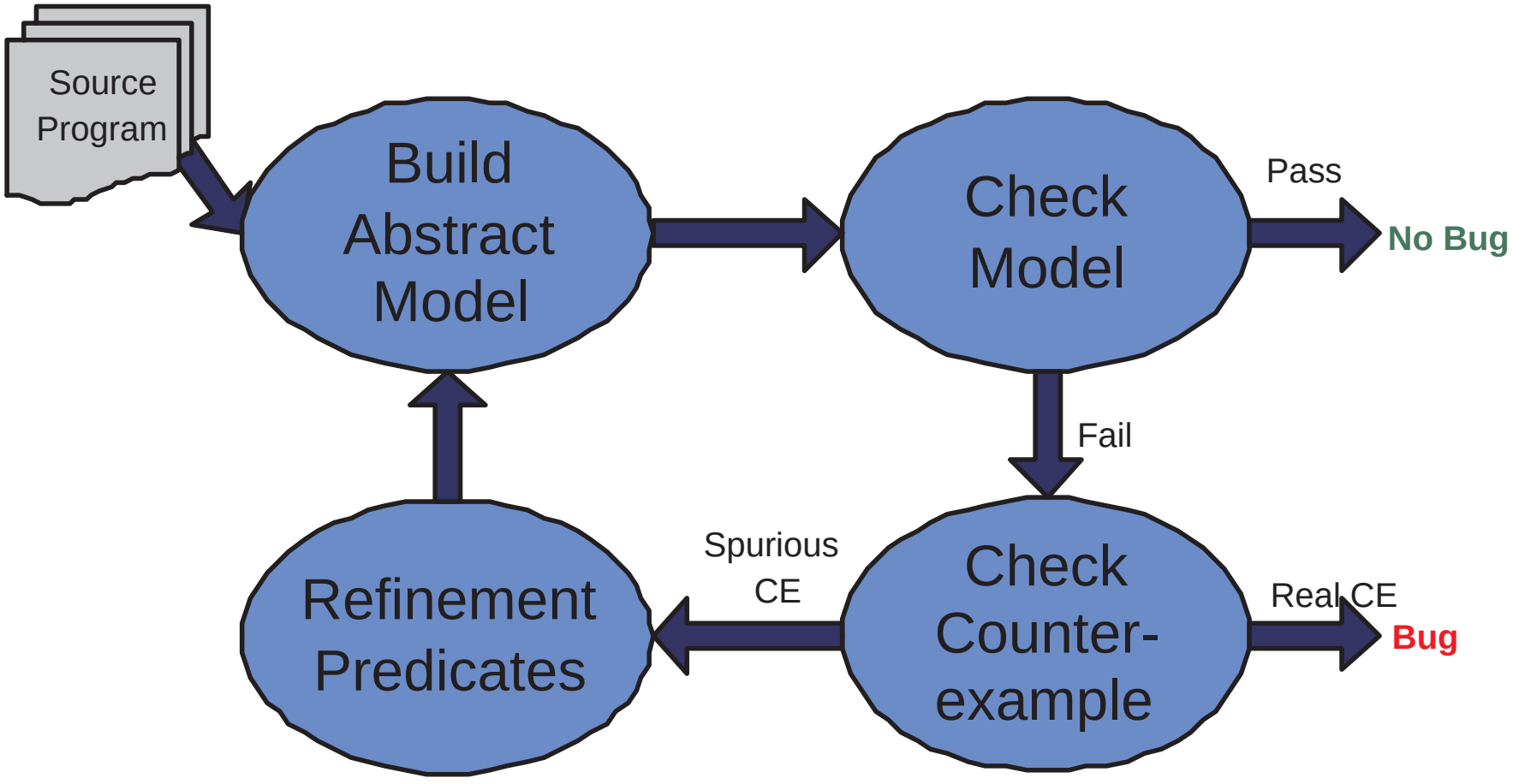
Thomas Ball , Andreas Podelski , Sriram K. Rajamani

Talk by: Rayna Dimitrova

9th June 2005

Outline

Counterexample-Guided Abstraction Refinement



Counterexample-Guided Abstraction Refinement

1. Construct abstraction
   - Finite set of predicates
   - Map concrete states to abstract states
   - Construct abstract transitions
2. Compute the set of reachable abstract states
   - all reachable states satisfy the property $\Rightarrow$ "Success"
   - otherwise check counterexample against the original state space
3. The counterexample is real $\Rightarrow$ "Bug"
4. Spurious counter example $\Rightarrow$
   - add new predicates to the set of predicates
   - go to **1**

Relative Completeness

- Property checking for software is undecidable
- Abstraction refinement is not complete
- Goodness criterion
  - comparison with a method for fixpoint iteration with "oracle-guided" widening
  - widening is used in fixpoint analysis
- Abstraction refinement procedure with backward refinement
  - Satisfies the criterion

Programs

- Program is a set of guarded commands
- Guarded command

$$c \equiv g(X) \wedge x_1' = e_1(X) \wedge \ldots \wedge x_m' = e_m(X)$$

- A program state is a valuation of X
- Transition $s \rightarrow s'$
  - corresponding valuations of primed and unprimed variables satisfy one of the guarded commands
- Each command is deterministic
- The program can be nondeterministic

An Example Program

C Program

```
L1: x = 0;
L2: while(x >= 0){
      x = x + 1;
    }
L3: if(y == 25){
L4:  if (y != 25){
L5:   z = -1;
L6:   while(z!= 0){
        z = z -1;
      }
      error:;
     }
    }
```

Set of guarded commands

variables $X = \{x, y, z, pc\}$

$c_1 : pc = l_1 \rightarrow pc := l_2, x := 0$
$c_2 : pc = l_2 \wedge x \geq 0 \rightarrow x := x + 1$
$c_3 : pc = l_2 \wedge x < 0 \rightarrow pc := l_3$
$c_4 : pc = l_3 \wedge y = 25 \rightarrow pc := l_4$
$c_5 : pc = l_4 \wedge y \neq 25 \rightarrow pc := l_5$
$c_6 : pc = l_5 \rightarrow pc := l_6, z := -1$
$c_7 : pc = l_6 \wedge z \neq 0 \rightarrow z := z - 1$
$c_8 : pc = l_6 \wedge z = 0 \rightarrow pc := error$

Symbolic Representation of Sets Of States

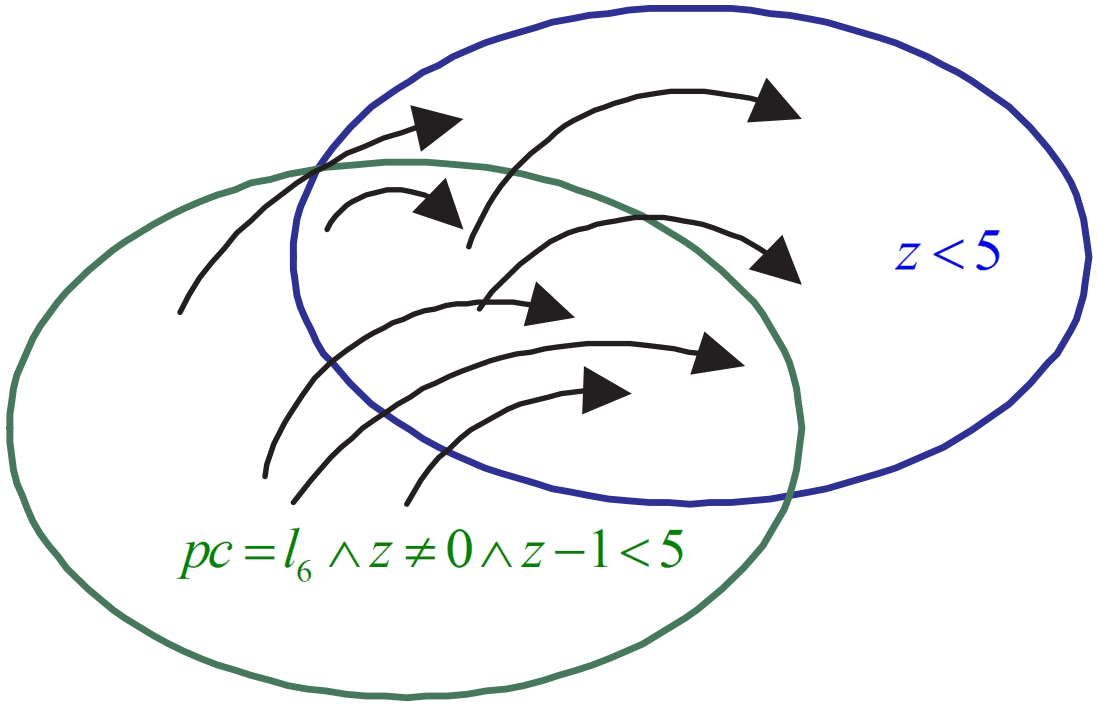
- Symbolic representation of set of states

$$\varphi \equiv \bigvee_{i \in I} \bigwedge_{j \in J_i} \varphi_{ij}$$

- Fixed infinite set of atomic formulas
- Partial order on formulas $\varphi \leq \varphi'$
  $\varphi \Rightarrow \varphi'$ is provable by a given theorem prover
- $\varphi \Rightarrow \varphi'$does not entail $\varphi \leq \varphi'$
- Requirement to the theorem prover
  - $\varphi \wedge \varphi' \Rightarrow \varphi$
  - $\varphi \Rightarrow \varphi \vee \varphi'$

The Pre Operator

- c - guarded command
- φ - formula
- C - program



- $pre_c(\varphi) \equiv g(X) \wedge \varphi[e_1(X), \ldots e_m(X)/x_1, \ldots, x_m]$
- $pre(\varphi) \equiv \bigvee_{c \in C} pre_c(\varphi)$

Example

- $c_7 : pc = l_6 \wedge z \neq 0 \rightarrow z := z - 1$
- $\varphi = z < 5$

The Pre Operator Contd.

Example

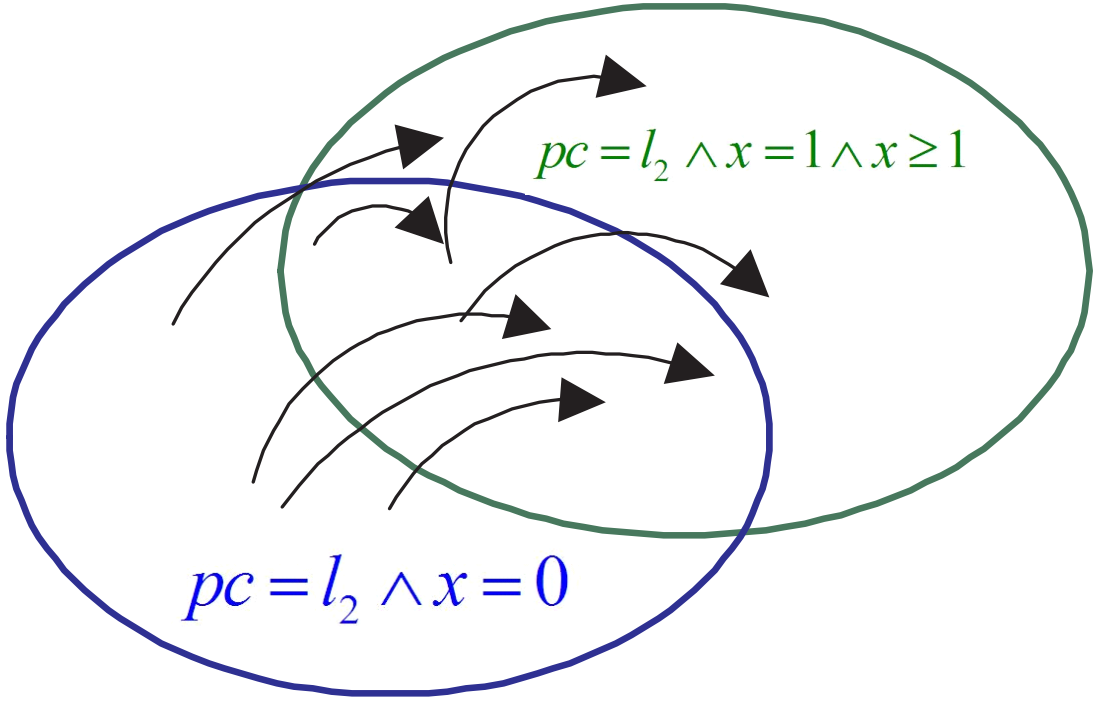
- $\varphi \equiv pc = l_6 \wedge z = 0$
- pre(φ) = $(pc = l_6 \wedge z \neq 0 \wedge z - 1 = 0) \vee (pc = l_5 \wedge -1 = 0)$

Set of guarded commands

$c_1 : pc = l_1 \rightarrow pc := l_2, x := 0$
$c_2 : pc = l_2 \wedge x \geq 0 \rightarrow x := x + 1$
$c_3 : pc = l_2 \wedge x < 0 \rightarrow pc := l_3$
$c_4 : pc = l_3 \wedge y = 25 \rightarrow pc := l_4$
$c_5 : pc = l_4 \wedge y \neq 25 \rightarrow pc := l_5$
$c_6 : pc = l_5 \rightarrow pc := l_6, z := -1$
$c_7 : pc = l_6 \wedge z \neq 0 \rightarrow z := z - 1$
$c_8 : pc = l_6 \wedge z = 0 \rightarrow pc := error$

The Post Operator

- c - guarded command
- φ - formula
- C - program



- $post_c(\varphi) \equiv (\exists X . \varphi \wedge g(X) \wedge x'_1 = e_1(X) \wedge ... \wedge x'_m = e_m(X))[X/X']$
- $post(\varphi) \equiv \bigvee_{c \in C} post_c(\varphi)$

Example

- $c_2 : pc = l_2 \wedge x \geq 0 \rightarrow x := x + 1$
- $\varphi = pc = l_2 \wedge x = 0$

The Post Operator Contd.

Example

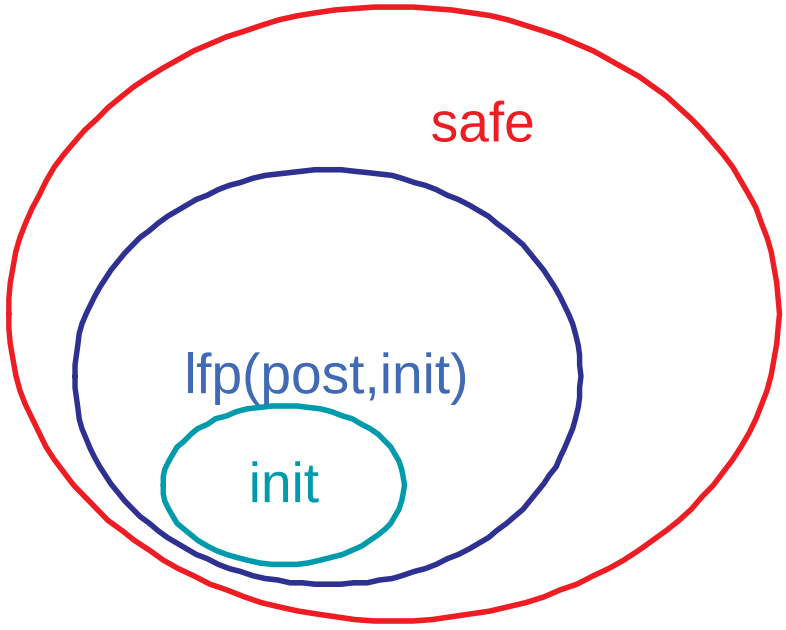
- $\varphi \equiv pc = l_2$
- post(φ) = $(pc = l_2 \wedge x \geq 1) \vee (pc = l_3 \wedge x < 0)$

Set of guarded commands

$c_1 : pc = l_1 \rightarrow pc := l_2, x := 0$
$c_2 : pc = l_2 \wedge x \geq 0 \rightarrow x := x + 1$
$c_3 : pc = l_2 \wedge x < 0 \rightarrow pc := l_3$
$c_4 : pc = l_3 \wedge y = 25 \rightarrow pc := l_4$
$c_5 : pc = l_4 \wedge y \neq 25 \rightarrow pc := l_5$
$c_6 : pc = l_5 \rightarrow pc := l_6, z := -1$
$c_7 : pc = l_6 \wedge z \neq 0 \rightarrow z := z - 1$
$c_8 : pc = l_6 \wedge z = 0 \rightarrow pc := error$

Correctness

- init - initial states
  nonInit
- safe - safe states
  unsafe

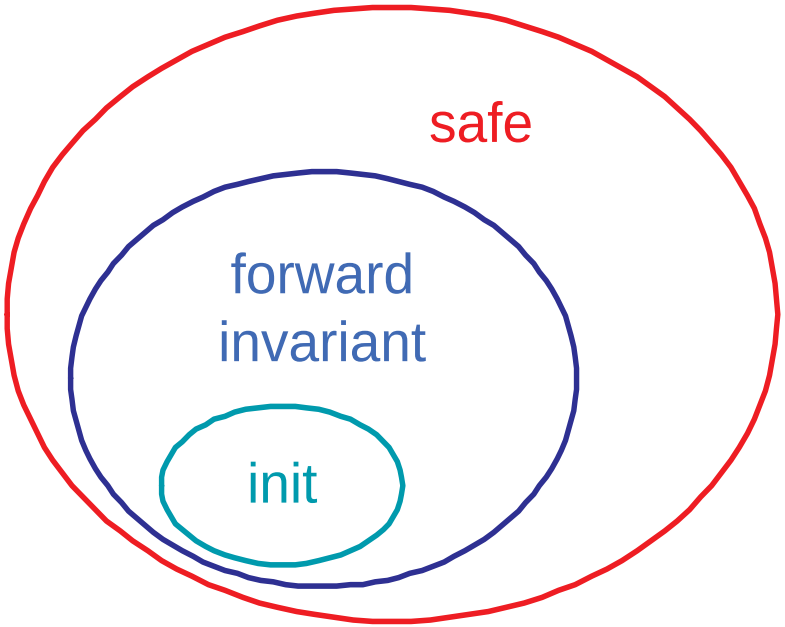


- The program is correct if no unsafe state is reachable from an initial state

Invariants

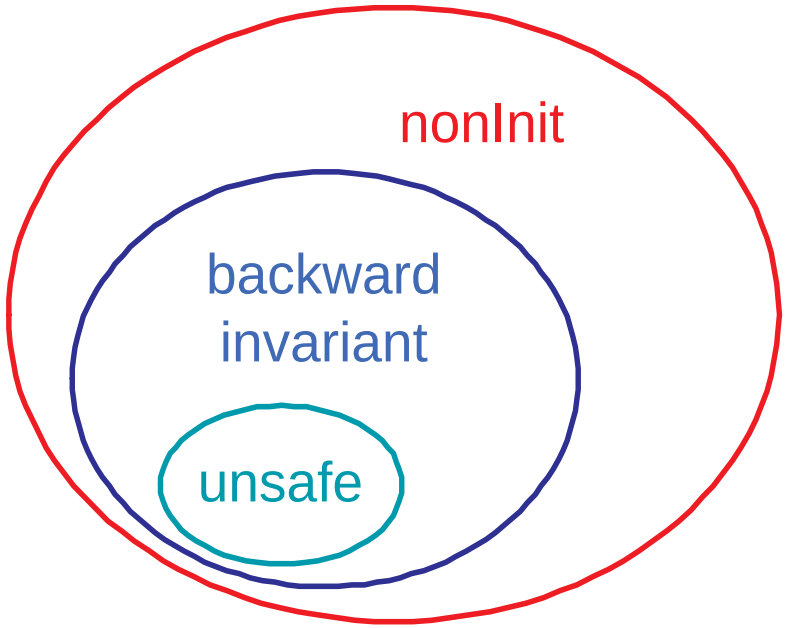
safe(forward) invariant ψ

- $\text{init} \leq \psi$
- $\text{post}(\psi) \leq \psi$
- $\psi \leq \text{safe}$



backward invariant ψ

- $\text{unsafe} \leq \psi$
- $\text{pre}(\psi) \leq \psi$
- $\psi \leq \text{nonInit}$



Proving Correctness

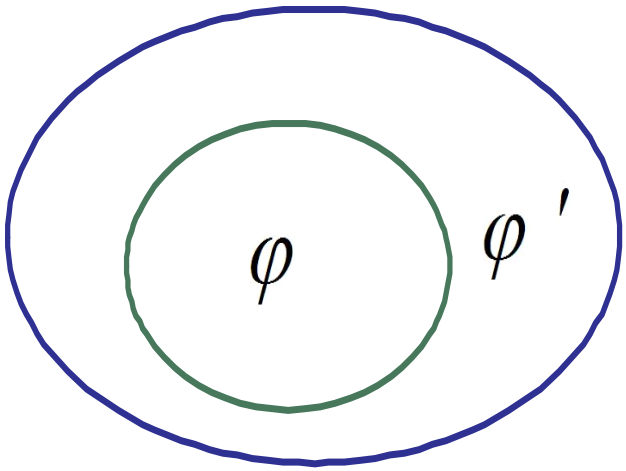
- $\langle$F, start, bound$\rangle$-invariant
  - instantiated to $\langle$post, init, safe$\rangle$ - forward invariant
  - instantiated to $\langle$pre, unsafe, nonInit$\rangle$ - backward invariant
- Domain of formulas
  - Closed under application of F
  - May not contain lfp(F,start), but still contain a formula denoting an $\langle$F, start, bound $\rangle$-invariant
- To prove correctness it suffices
  - compute either a forward invariant
  - or a backward invariant
- Correctness condition: exists ψ
  - start $\leq \psi$
  - $\mathrm{F}(\psi) \leq \psi$
  - $\psi \leq$ bound

Proving Correctness Contd.

- Iteration of F may not terminate
- Upper abstraction F′
  - $F(\varphi) \leq F'(\varphi)$ for all φ
  - lfp(F′,start) can be computed
  - lfp(F′,start)$\leq$ bound
- lfp(F′,start) is a $\langle$F,start,bound$\rangle$-invariant
- Use
  - predicate abstraction
  - widening

Abstraction

- φ is mapped to an element of a finite lattice $\mathcal{L}(\mathcal{P})$
- $\mathcal{L}(\mathcal{P})$
  - generated by a finite set of predicates $\mathcal{P}$
  - partial order $\sqsubseteq$
- Abstraction function:
  $\varphi \mapsto$ the smallest (wrt $\sqsubseteq$) element of the lattice φ' such that $\varphi \leq \varphi'$



The Lattice $\mathcal{L}(\mathcal{P}_n)$

- Finite free distributive lattice generated by the set of predicates $\mathcal{P}_n$
- Bottom element *false*
- Top element *true*
- Operators $\wedge$ and $\vee$
- Partial order $\sqsubseteq$
- Contains start
- Generally not closed under F

The Lattice $\mathcal{L}(\mathcal{P}_n)$ Contd.

- Each element can be written in disjunctive normal form $\bigvee_{i \in I} \bigwedge_{j \in J_i} \varphi_{ij}$
- Predicates are pairwise incomparable
- $\bigvee_{i \in I} \bigwedge_{j \in J_i} \varphi_{ij} \sqsubseteq \bigvee_{k \in K} \bigwedge_{j \in J'_k} \varphi'_{kj}$ if $\forall i \in I \exists k \in K . \{\varphi_{ij} \mid j \in J_i\} \supseteq \{\varphi'_{kj} \mid j \in J'_k\}$

Example

- $((x \leq 0) \wedge (y \geq 5) \wedge (y \neq 10)) \vee ((x \leq y) \wedge (y \neq 0)) \sqsubseteq (x \leq y) \vee ((x \leq 0) \wedge (y \geq 5)) \vee (y = 2x)$
- $(x < 2) \not\sqsubseteq (x < 3)$

- $\varphi \sqsubseteq \varphi'$ implies $\varphi \leq \varphi'$
- $\varphi \wedge \varphi' \sqsubseteq \varphi$
- $\varphi \sqsubseteq \varphi \vee \varphi'$

The Abstract Operator

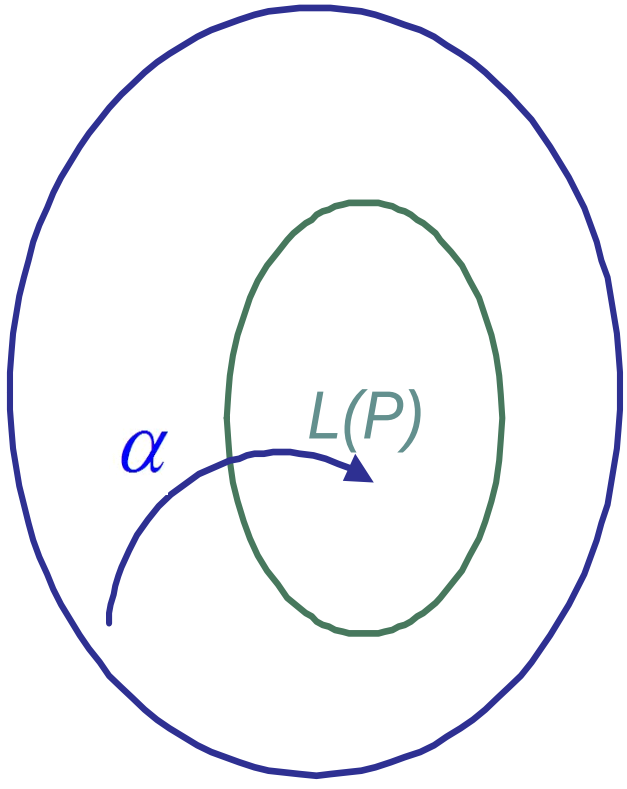
- $F^{\#}$ is the 'best' abstraction of F with respect to $\mathcal{L}(\mathcal{P})$

$$F^{\#} \equiv \alpha \circ F \circ \gamma$$

- The meaning function γ is the identity
- α and γ form a Galois connection ▸ Definition
  - guaranteed by requirement to the theorem prover
  - guarantees correctness of algorithm

Galois Connection

- Two partially ordered sets
  - the infinite set of all formulas with $\leq$
  - $\mathcal{L}(\mathcal{P})$ with $\sqsubseteq$
- Two monotone functions
  - α
  - γ
- α and γ satisfy the properties
  - $\alpha(\gamma(\psi)) \sqsubseteq \psi$
  - $\varphi \leq \gamma(\alpha(\varphi))$
- $\alpha(\varphi) \sqsubseteq \psi$ iff $\varphi \leq \gamma(\psi)$



Method I

Start with n=0 and repeat

1. construct $F_n^{\#}$
2. iterate $F_n^{\#}$ to compute lfp($F_n^{\#}$,start)
   - if lfp($F_n^{\#}$,start) $\leq$ bound then stop with "Success"
3. refine $\mathcal{P}_n$ to obtain $\mathcal{P}_{n+1}$
4. n++

Method I

Start with n=0 and repeat

1. construct $F_n^{\#}$
2. iterate $F_n^{\#}$ to compute lfp($F_n^{\#}$,start)
   - if lfp($F_n^{\#}$,start) $\leq$ bound then stop with "Success"
3. refine $\mathcal{P}_n$ to obtain $\mathcal{P}_{n+1}$
4. n++

lfp($F_n^{\#}$,start) is computed over a finite lattice $\Rightarrow$
computation is guaranteed to terminate

Method I

Start with n=0 and repeat

1. construct $F_n^\#$
2. iterate $F_n^\#$ to compute lfp($F_n^\#$,start)
   - if lfp($F_n^\#$,start) $\leq$ bound then stop with "Success"
3. refine $\mathcal{P}_n$ to obtain $\mathcal{P}_{n+1}$
4. n++

lfp($F_n^\#$,start) is computed over a finite lattice $\Rightarrow$
computation is guaranteed to terminate

Terminates for some n $\Rightarrow$
lfp($F_n^\#$,start) is a $\langle$F,start,bound$\rangle$ - invariant

The Refinement Procedure

- Refinement procedure generates $\mathcal{P}_0 \subset \mathcal{P}_1 \subset \ldots \subset \mathcal{P}_n \subset \mathcal{P}_{n+1} \ldots$
  - $\mathcal{P}_n$ - finite set of predicates over states
  - predicates identified with atomic formulas
- $\mathcal{L}(\mathcal{P}_0) \subset \mathcal{L}(\mathcal{P}_1) \subset \ldots \subset \mathcal{L}(\mathcal{P}_n) \subset \mathcal{L}(\mathcal{P}_{n+1}) \ldots$
- α_n is the abstraction function wrt $\mathcal{L}(\mathcal{P}_n)$
- Increasing precision of α_n for increasing n

The Refinement Procedure Contd.

- The algorithm procedure produces the sequence
  - φ_0 = start
  - $\varphi_{n+1} = \varphi_n \vee \mathsf{F}(\varphi_n)$
- $\mathcal{P}_n$=atoms(φ_n)
- Backward refinement
- Forward refinement

Abstract Fixpoint Iteration with Iterative Abstraction Refinement

φ_0 := start
n: = 0
loop
    $\mathcal{P}_n$:= atoms(φ_n)
    construct abstract operator $F_n^{\#}$ defined by $\mathcal{P}_n$
    ψ := lfp($F_n^{\#}$,start)
    if($\psi \leq$ bound) **then**
        STOP with "Success"
    φ_{n+1} := $\varphi_n \vee F(\varphi_n)$
    n:= n + 1
endloop

Example

- $\langle$ start, F, bound$\rangle$:= $\langle$ unsafe, pre, nonInit $\rangle$
- unsafe =$pc = error$
- nonInit =
  $pc = l_2 \vee pc = l_3 \vee pc = l_4 \vee pc = l_5 \vee pc = l_6 \vee pc = error$

Example

- φ_0 = unsafe = $pc = error$
- $\mathcal{P}_0 = \{pc = error\}$
- $\text{pre}_0^{\#}$(unsafe) = $\alpha_0(pc = l_6 \wedge z = 0)$=true
- lfp($\text{pre}_0^{\#}$,unsafe) = true
- lfp($\text{pre}_0^{\#}$,unsafe) $\not\leq$ nonInit

Example

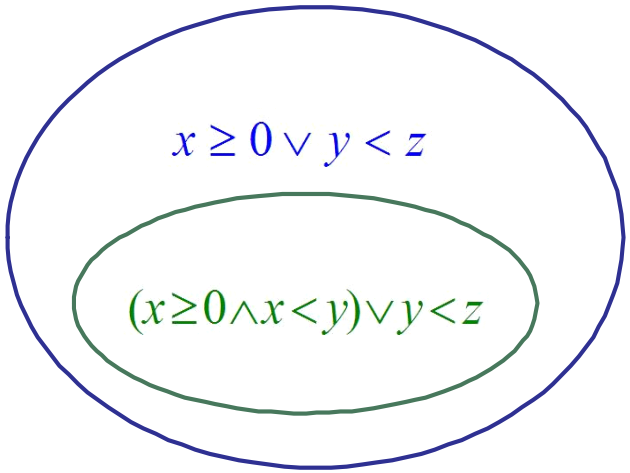
- $\varphi_1 = \varphi_0 \vee \text{pre}(\varphi_0) = (pc = error) \vee (pc = l_6 \wedge z = 0)$
- $\mathcal{P}_1 = \{pc = error, pc = l_6, z = 0\}$
- $\text{pre}_1^{\#}(\text{unsafe}) = \alpha_1(pc = l_6 \wedge z = 0) = pc = l_6 \wedge z = 0$
- $\text{pre}_1^{\#}(\text{unsafe} \vee pc = l_6 \wedge z = 0) =$
  $\alpha_1((pc = l_6 \wedge z = 0) \vee$
  $(pc = l_6 \wedge z \neq 0 \wedge z = 1) \vee$
  $(pc = l_5 \wedge -1 = 0)) =$
  $(pc = l_6 \wedge z = 0) \vee (pc = l_6) \vee \text{true} =$
  true
- lfp($\text{pre}_1^{\#}$,unsafe) = true
- lfp($\text{pre}_1^{\#}$,unsafe) $\not\leq$ *nonInit*

Example

- φ_2=...
- Terminates in four iterations
- lfp($pre^{\#}_4$,unsafe) $\leq$ nonInit
- Disjuncts with unsatisfiable conjuncts not removed

The Widen Operator

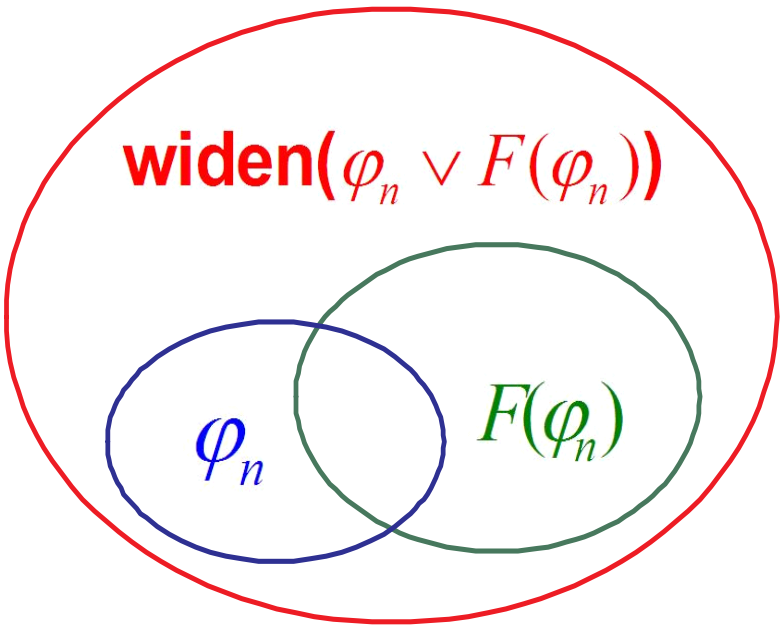
- Applied to a formula in disjunctive normal form yields a formula in disjunctive normal form
- Drops out some conjuncts from some disjuncts
- **widen**(φ) is weaker than φ, i.e. it denotes a larger set of states



- $\varphi \leq \text{widen}(\varphi)$

Overview

- Iteratively applies the concrete operator over formulas
- At each iteration a widening operator is applied to the result



- until $\varphi_{n+1} \leq \varphi_n$
- $\varphi_n \leq$ bound $\Rightarrow$
  φ_n is a $\langle$F, start, bound$\rangle$-invariant

The Oracle

- Enumeration of widening operators: **widen**(0),**widen**(1),. . .
- At each step an oracle gives us a natural number - determines the widening operator to be applied
- Each sequence of natural numbers determines a fixpoint iteration sequence

Method II

φ'_0, old, n := start,false,0
loop
  if($\varphi'_n \leq$ old) **then**
    if($\varphi'_n \leq$ bound) **then**
      STOP with "Success"
    else
      STOP with "Don't know"
  else
  old := φ'_n
  i := guess provided by oracle
  φ'_{n+1} := widen(i,$\varphi'_n \vee$ F(φ'_n))
  n:= n + 1
endloop

Example

- $\langle$ start, F, bound$\rangle$:=$\langle$ unsafe, pre, nonInit$\rangle$
- unsafe =$pc = error$
- nonInit =
  $pc = l_2 \vee pc = l_3 \vee pc = l_4 \vee pc = l_5 \vee pc = l_6 \vee pc = error$

Example

$$
\begin{aligned}
\varphi_0 = \quad & \text{unsafe} = \quad (pc = error) \\
\varphi_1 = \quad & \text{widen}(\varphi_0 \vee pre(\varphi_0)) = \\
& \text{widen}((pc = error) \vee (pc = l_6 \wedge z = 0)) = \\
& (pc = error) \vee (pc = l_6) \\
\varphi_2 = \quad & \text{widen}(\varphi_1 \vee pre(\varphi_1)) = \\
& \text{widen}((pc = error) \vee (pc = l_6) \vee (pc = l_6 \wedge z = 0) \\
& \vee (pc = l_6 \wedge z \neq 0) \vee (pc = l_5))) = \\
& (pc = error) \vee (pc = l_6) \vee (pc = l_5)
\end{aligned}
$$

Example

$$
\begin{aligned}
\varphi_3 = \quad & \text{widen}(\varphi_2 \vee \mathit{pre}(\varphi_2)) = \\
& \text{widen}((pc = \mathit{error}) \vee (pc = l_6) \vee (pc = l_5) \vee \\
& (pc = l_6 \wedge z = 0) \vee (pc = l_6 \wedge z \neq 0) \vee \\
& (pc = l_5) \vee (pc = l_4 \wedge y \neq 25))) = \\
& (pc = \mathit{error}) \vee (pc = l_6) \vee (pc = l_5) \vee \\
& (pc = l_4 \wedge y \neq 25)
\end{aligned}
$$

Example

$$
\begin{aligned}
\varphi_4 = \quad & \text{widen}(\varphi_3 \vee \mathit{pre}(\varphi_3)) = \\
& \text{widen}((\mathit{pc} = \mathit{error}) \vee (\mathit{pc} = l_6) \vee \\
& (\mathit{pc} = l_5) \vee (\mathit{pc} = l_4 \wedge y \neq 25) \vee \\
& (\mathit{pc} = l_6 \wedge z = 0) \vee (\mathit{pc} = l_6 \wedge z \neq 0) \vee \\
& (\mathit{pc} = l_5) \vee (\mathit{pc} = l_4 \wedge y \neq 25) \vee \\
& (\mathit{pc} = l_3 \wedge y = 25 \wedge y \neq 25)) = \\
& (\mathit{pc} = \mathit{error}) \vee (\mathit{pc} = l_6) \vee (\mathit{pc} = l_5) \vee \\
& (\mathit{pc} = l_4 \wedge y \neq 25) \vee \\
& (\mathit{pc} = l_3 \wedge y = 25 \wedge y \neq 25)
\end{aligned}
$$

Example

- $\varphi_4 \leq \varphi_3$
- $\varphi_4 \leq$ nonInit
- φ_4 is an $\langle$pre,unsafe,nonInit$\rangle$ -invariant

Relative Completeness of Abstract Backward Iteration with Backward Refinement

Theorem

If Method II with $\langle F,start,bound\rangle := \langle pre,unsafe,nonInit\rangle$ terminates with success, then
Method I with $\langle F,start,bound\rangle := \langle pre,unsafe,nonInit\rangle$ also terminates with success.

▸ Proof details

- Method I generates

$$(start, pre_n^{\#}(start), \ldots, lfp(pre_n^{\#}, start))_{n=1,2,\ldots}$$

- All possible infinite branches arising from possible choices of the widening operators

$$(start, widen(i_1) \circ pre(start), \ldots)_{(i_1,i_2,\ldots)\in\mathbb{N}^{\mathbb{N}}}$$

Relative Completeness of Abstract Backward Iteration with Backward Refinement

The precision of the abstract operator $F^{\#}$ is related to the expressiveness of the set of predicates $\mathcal{P}$ it is induced by.

Lemma

If the set of predicates $\mathcal{P}$ can express an $\langle F,start,bound\rangle$-invariant ψ, then the least fixpoint of $F^{\#}$, the best abstraction of F over $\mathcal{L}(\mathcal{P})$, is an $\langle F,start,bound\rangle$-invariant as well.

▸ Proof details

We cannot expect a realistic abstraction refinement procedure that generates such a set $\mathcal{P}$ whenever it exists.

Forward Fixpoint Iteration

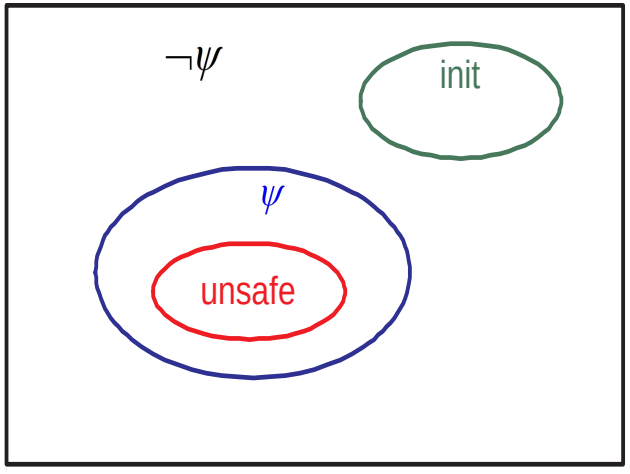
- Relative completeness for forward fixpoint iteration with backward refinement?
- The key point is the backward direction of the refinement
- Weakest precondition operator

$$\widetilde{pre}(\varphi) = \neg pre(\neg\varphi)$$

- s satisfies $\widetilde{pre}(\varphi)$ iff all successors of s satisfy φ
- lfp($\widetilde{pre}, safe$) denotes the set of all states from which only safe states are reachable
- Dual backward refinement
  - φ_0 = safe
  - $\varphi_{n+1} = \varphi_n \vee \widetilde{pre}(\varphi_n)$

Forward Fixpoint Iteration with Backward Abstraction Refinement

- abstract backward fixpoint iteration with backward refinement
  - $\mathcal{P}_0$ = atoms(unsafe)
  - $\mathcal{P}_{n+1} = \mathcal{P}_n \cup$ atoms($pre(\varphi_n)$)
- abstract forward fixpoint iteration with dual backward refinement
  - $\widetilde{\mathcal{P}}_0$ = atoms(safe)
  - $\widetilde{\mathcal{P}}_{n+1} = \widetilde{\mathcal{P}}_n \cup$ atoms($\neg pre(\varphi_n)$)



- ψ - $\langle$pre,unsafe,nonInit$\rangle$ -invariant
- $\neg\psi$ - $\langle$post,init,safe$\rangle$ -invariant

If ψ can be expressed over $\mathcal{P}_n$
then $\neg\psi$ can be expressed over $\{\neg p \mid p \in \mathcal{P}_n\} = \widetilde{\mathcal{P}}_n$

Forward Fixpoint Iteration with Backward Abstraction Refinement(Method III)

φ_0 := safe
n: = 0
loop
    $\widetilde{\mathcal{P}}_n$:= atoms(φ_n)
    construct abstract operator $\text{post}_n^{\#}$ defined by $\widetilde{\mathcal{P}}_n$
    ψ := lfp($\text{post}_n^{\#}$,init)
    if($\psi \leq$ safe) **then**
        STOP with "Success"
    $\varphi_{n+1} := \varphi_n \vee \widetilde{pre}(\varphi_n)$
    n:= n + 1
endloop

Relative Completeness of Abstract Forward Iteration with Backward Refinement

Theorem

If Method II with $\langle F, start, bound \rangle := \langle pre, unsafe, nonInit \rangle$ terminates with success, then
Method III also terminates with success.

Proof.

- ψ - $\langle$pre, unsafe, nonInit$\rangle$ -invariant computed by Method II
- ψ can be expressed over $\mathcal{P}_n$
- $\neg\psi$ - $\langle$post, init, safe$\rangle$ -invariant, can be expressed over $\widetilde{\mathcal{P}}_n$

□

Example Program

The completeness of Method I relative to Method II does not hold for the forward case $\langle$F, start, bound$\rangle$:= $\langle$post, init, safe$\rangle$

C Program

```
L1: x = 0;
L2: while(x >= 0){
      x = x + 1;
    }
L3: if(y == 25){
L4:  if (y != 25){
L5:   z = -1;
L6:   while(z!= 0){
        z = z -1;
      }
      error:;
    }
  }
```

Set of guarded commands

$variablesX = \{x, y, z\}$

$c_1 : pc = l_1 \rightarrow pc := l_2, x := 0$

$c_2 : pc = l_2 \wedge x \geq 0 \rightarrow x := x + 1$

$c_3 : pc = l_2 \wedge x < 0 \rightarrow pc := l_3$

$c_4 : pc = l_3 \wedge y = 25 \rightarrow pc := l_4$

$c_5 : pc = l_4 \wedge y \neq 25 \rightarrow pc := l_5$

$c_6 : pc = l_5 \rightarrow pc := l_6, z := -1$

$c_7 : pc = l_6 \wedge z \neq 0 \rightarrow z := z - 1$

$c_8 : pc = l_6 \wedge z = 0 \rightarrow pc := error$

Method II Forward

- Iterative application of post and oracle-guided widening
- Drops all conjuncts containing x
- Does not get "stuck" in the nonterminating loop
- Terminates with success

Method I with Forward Refinement

- Does not terminate
- Refinement gets "stuck" in the first nonterminating loop
- $x = 0, x = 1, x = 2 \ldots$

Forward vs. Backward Refinement

- Refinement must be based on the concrete execution
- Forward
  - the concrete execution of a guarded command is deterministic
  - an abstract execution is in general nondeterministic
  - the concrete execution follows one branch and may get stuck in a loop
- Backward
  - the concrete execution is nondeterministic
  - the concrete execution reaches as many program points as an abstract one
  - pre must produce also unsatisfiable disjuncts

BDD's and the Free Lattice

- Finite-state model checker
  - implement abstract fixpoint iteration
  - based on BDD's
  - Boolean variable for each predicate
  - fixpoint termination test does not use logical meaning of predicates
- $\sqsubseteq$ strictly stronger than $\leq$
  - $[x < 2] \not\sqsubseteq [x < 3]$
  - The fixpoint test in Method I is strictly weaker than the one of Method II

Boolean Expressions

- No need to add the negated versions of predicates to $\mathcal{L}(\mathcal{P})$
- More efficient construction of the abstract fixpoint operator
- The lattice of Boolean expressions $\mathcal{L}(\mathcal{P} \bigcup \{\neg\varphi \mid \varphi \in \mathcal{P}\})$ is an instance
- In the setting of Boolean expressions Method III iterates **pre** starting with **unsafe** and adds the negation of the predicates

More Powerful Refinement

- Backward refinement procedure also adds predicates that occur in unsatisfiable conjuncts

Example

- $pc = l_5 \wedge z' = -1 \wedge pc' = l_6$
- atoms($pre_c(pc = l_6 \wedge z = 0)$) = atoms($pc = l_5 \wedge -1 = 0$)= $\{pc = l_5, -1 = 0\}$

Widening vs. Predicate Abstraction

- Two abstraction methods for verification
- Widening is not complete relative to the predicate abstraction with backward refinement
  - dropping a conjunct less precise than
  - replacing a conjunct with a formula over already generated predicates
- Their power depends on the given formalism
  - atoms($\{x = 0\}$) $= \{x \leq 0, x \geq 0\}$
  - Method I with forward refinement will succeed on the example

Other Issues

- Incorrect programs
- Finite simulation or bisimulation quotient
- Generating small sets of predicates

Summary

- Different refinement procedures can be evaluated not only practically
- Comparison with oracle-guided fixpoint iteration gives a quality measure
- Predicate abstraction with backward refinement is at least as powerful as oracle-guided fixpoint iteration

Thank you!

Proof.

1. $$atoms(pre(\bigvee_{i \in I} \bigwedge_{j \in J_i} \varphi_{ij})) = \bigcup_{c \in C} \{atoms(pre_c(\varphi_{ij})) \mid i \in I, j \in J_i\}$$

2.
   - φ_n -the formula at the beginning of n-th iteration of Method I
   - φ'_n -the formula at the beginning of n-th iteration of Method II
   - atoms(φ_n) $\supseteq$ atoms(φ'_n)

   ▸ Proof by Induction

3. φ'_n - an $\langle$F,start,bound$\rangle$-invariant that can be expressed in atoms(φ'_n) $\subseteq \mathcal{P}_n$.

□

◂ Return

atoms(φ_n) $\supseteq$atoms(φ'_n)

Proof.

- $n = 0$
  - $\varphi_0 = \mathit{start}$ and $\varphi'_0 = \mathit{start}$
- $n + 1$
  - $\varphi_{n+1} = \varphi_n \vee \mathrm{pre}(\varphi_n)$
  - $\varphi'_{n+1} = \mathrm{widen}(\varphi'_n \vee \mathrm{pre}(\varphi'_n))$
  - IH $\Rightarrow$ atoms(φ_n) $\supseteq$atoms(φ'_n)
  - atoms(pre(φ_n)) $\supseteq$ atoms(pre(φ'_n))
  - widen can only drop atomic formulas

$\square$

Return

Lemma

If the set of predicates $\mathcal{P}$ can express an $\langle$F,start,bound$\rangle$-invariant ψ, then the least fixpoint of $F^{\#}$, the best abstraction of F over $\mathcal{L}(\mathcal{P})$, is an $\langle$F,start,bound$\rangle$-invariant as well.

Proof.

1. For all *k*: $F^{\#^k}(\text{start}) \leq \psi$
   1. $F^{\#^0}(\text{start}) = \text{start} \leq \psi$
   2. $F^{\#}(F^{\#^k}(\text{start})) \leq F^{\#}(\psi) \leq \psi$
      - $F^{\#}(\psi)$-the least element in $\mathcal{L}(\mathcal{P})$ greater or equal to $F(\psi)$
      - $F(\psi) \leq \psi$
      - ψ is an element of $\mathcal{L}(\mathcal{P})$
2. lfp($F^{\#}$,start) $\leq \psi \leq$ bound
3. lfp($F^{\#}$,start) is an $\langle$F,start,bound$\rangle$-invariant

□