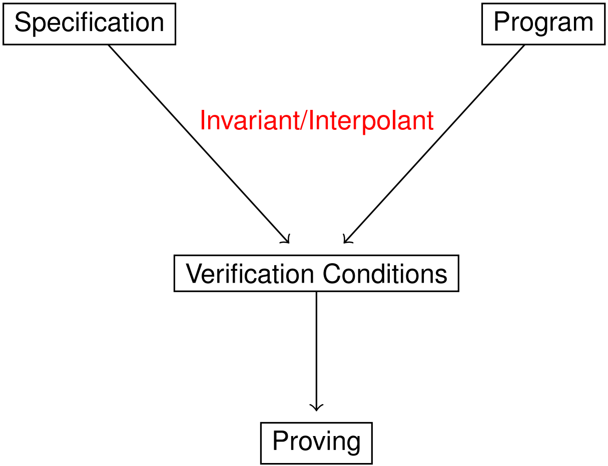
New Features and Applications of Vampire

Symbol Elimination and Interpolation for Software Verification

Kryštof Hoder, Laura Kovács, Andrei Voronkov


Specification
Program
Invariant/Interpolant
Verification Conditions
Proving

Outline

Part I: Quantified Invariants and Symbol Elimination

Part II: Symbol Elimination and Interpolation

Summary: Invariant Generation, Interpolation, Symbol Elimination

Outline

Part I: Quantified Invariants and Symbol Elimination

Part II: Symbol Elimination and Interpolation

Summary: Invariant Generation, Interpolation, Symbol Elimination

Example: Array Partition partition.c

$a := 0$; $b := 0$; $c := 0$;
while $(a \leq k)$ **do**
  if $A[a] \geq 0$
    then $B[b] := A[a]$;$b := b + 1$;
    else $C[c] := A[a]$;$c := c + 1$;
  $a := a + 1$;
end do

A :

1	3	-1	-5	8	0	-2

$a = 0$

B :

*	*	*	*	*	*	*

$b = 0$

C :

*	*	*	*	*	*	*

$c = 0$

Example: Array Partition partition.c

while $(a \leq k)$ **do**
$a := 0;\ b := 0;\ c := 0;$

  if $A[a] \geq 0$
    then $B[b] := A[a]; b := b + 1;$
    else $C[c] := A[a]; c := c + 1;$
  $a := a + 1;$
end do

A :

1	3	-1	-5	8	0	-2

$a = 7$

B :

1	3	8	0	*	*	*

$b = 4$

C :

-1	-5	-2	*	*	*	*

$c = 3$

Example: Array Partition partition.c

$a := 0$; $b := 0$; $c := 0$;
while $(a \leq k)$ **do**
  if $A[a] \geq 0$
    then $B[b] := A[a]; b := b + 1$;
    else $C[c] := A[a]; c := c + 1$;
  $a := a + 1$;
end do

A:	1	3	-1	-5	8	0	-2

$a = 7$

B:	1	3	8	0	*	*	*

$b = 4$

C:	-1	-5	-2	*	*	*	*

$c = 3$

Invariants with $\forall$ $\exists$

- Each of $B[0], \ldots, B[b-1]$ is non-negative and equal to one of $A[0], \ldots, A[a-1]$.

$$(\forall p)(0 \leq p < b \rightarrow B[p] \geq 0 \wedge (\exists i)(0 \leq i < a \ \wedge \ A[i] = B[p]))$$

Example: Array Partition partition.c

$a := 0; b := 0; c := 0;$
while $(a \leq k)$ **do**
  if $A[a] \geq 0$
    then $B[b] := A[a]; b := b + 1;$
    else $C[c] := A[a]; c := c + 1;$
  $a := a + 1;$
end do

A :

1	3	-1	-5	8	0	-2

$a = 7$

B :

1	3	8	0	*	*	*

$b = 4$

C :

-1	-5	-2	*	*	*	*

$c = 3$

Invariants with $\forall$ $\exists$

- Each of $B[0], \ldots, B[b-1]$ is non-negative and equal to one of $A[0], \ldots, A[a-1]$.

$$(\forall p)(0 \leq p < b \rightarrow B[p] \geq 0 \wedge (\exists i)(0 \leq i < a \ \wedge \ A[i] = B[p]))$$

Example: Array Partition partition.c

$a := 0;\ b := 0;\ c := 0;$
while $(a \leq k)$ **do**
  if $A[a] \geq 0$
    then $B[b] := A[a]; b := b + 1;$
    else $C[c] := A[a]; c := c + 1;$
  $a := a + 1;$
end do

A :

1	3	-1	-5	8	0	-2

$a = 7$

B :

1	3	8	0	*	*	*

$b = 4$

C :

-1	-5	-2	*	*	*	*

$c = 3$

Invariants with $\forall$ $\exists$

- Each of $B[0], \ldots, B[b-1]$ is non-negative and equal to one of $A[0], \ldots, A[a-1]$.
- Each of $C[0], \ldots, C[c-1]$ is negative and equal to one of $A[0], \ldots, A[a-1]$.

Invariants with $\forall$

- For every $p \geq b$, the value of $B[p]$ is equal to its initial value.
- For every $p \geq c$, the value of $C[p]$ is equal to its initial value.

Example: Array Partition - Vampire Experiments

$a := 0$; $b := 0$; $c := 0$;
while $(a \leq k)$ **do**
  if $A[a] \geq 0$
    then $B[b] := A[a]; b := b + 1$;
    else $C[c] := A[a]; c := c + 1$;
  $a := a + 1$;
end do

1. B doesn't change at positions after final value of b ($1s$):

$$\forall p(p \geq b \rightarrow B[p] = B_0[p])$$

2. Each $B[0], \ldots, B[b-1]$ is a positive value in $\{A[0], \ldots, A[a-1]\}$ ($1s$):

$$\forall p(b > p \wedge p \geq 0 \rightarrow B[p] \geq 0 \wedge \exists k(a > k \wedge k \geq 0 \wedge A[k] = B[p])$$

Invariant Generation in Vampire: Overview

- Given loop $\mathcal{L}$;
- Extend $\mathcal{L}$ to $\mathcal{L}'$;
- Extract a set P of loop properties in $\mathcal{L}'$;
- Generate loop property p in $\mathcal{L}$ s.t. $P \to p$.

Invariant Generation in Vampire: Overview

- Given loop $\mathcal{L}$;
- Extend $\mathcal{L}$ to $\mathcal{L}'$;
- Extract a set P of loop properties in $\mathcal{L}'$;
- Generate loop property p in $\mathcal{L}$ s.t. $P \to p$.

Invariant Generation in Vampire: Overview

- Given loop $\mathcal{L}$;
- Extend $\mathcal{L}$ to $\mathcal{L}'$;
- Extract a set P of loop properties in $\mathcal{L}'$;
- Generate loop property p in $\mathcal{L}$ s.t. $P \rightarrow p$.

  $\leftarrow$ Symbol elimination!

Invariant Generation - The Method

```
a := 0; b := 0; c := 0;
while (a ≤ k) do
  if A[a] ≥ 0
    then B[b] := A[a];b := b + 1;
    else C[c] := A[a];c := c + 1;
  a := a + 1;
end do
```

1. Extend the language $\mathcal{L}$ to $\mathcal{L}'$:
   - variables as functions of loop cnt n: $v^{(i)}$ with $0 \leq i < n$
   - predicates as loop properties: $iter, upd_V(i,p), upd_V(i,p,x)$

- $upd_V(i,p)$: at iteration i, V is updated at position p;
- $upd_V(i,p,x)$: at iteration i, V is updated at position p by value x.

$$(\forall i)(i \in iter \Leftrightarrow 0 \leq i \wedge i < n)$$
$$upd_B(i,p) \Leftrightarrow i \in iter \wedge p = b^{(i)} \wedge A[a^{(i)}] \geq 0$$
$$upd_B(i,p,x) \Leftrightarrow upd_B(i,p) \wedge x = A[a^{(i)}]$$

$$a = b + c,\ a \geq 0,\ b \geq 0,\ c \geq 0$$
$$(\forall i \in iter)(a^{(i+1)} > a^{(i)})$$
$$(\forall i \in iter)(b^{(i+1)} = b^{(i)} \vee b^{(i+1)} = b^{(i)} + 1)$$
$$(\forall i \in iter)(a^{(i)} = a^{(0)} + i)$$
$$(\forall j,k \in iter)(k \geq j \to b^{(k)} \geq b^{(j)})$$
$$(\forall j,k \in iter)(k \geq j \to b^{(j)} + k \geq b^{(k)} + j)$$
$$(\forall p)(b^{(0)} \leq p < b^{(n)} \to (\exists i \in iter)(b^{(i)} = p \wedge A[a^{(i)}] \geq 0))$$
$$(\forall i) \neg upd_B(i,p) \to B^{(n)}[p] = B^{(0)}[p]$$
$$upd_B(i,p,x) \wedge (\forall j > i) \neg upd_B(j,p) \to B^{(n)}[p] = x$$

3. Eliminate symbols $\to$ Invariants

$$(\forall i \in iter)(A[a^{(i)}] \geq 0 \to B^{(i+1)}[b^{(i)}] = A[a^{(i)}] \wedge b^{(i+1)} = b^{(i)} + 1 \wedge c^{(i+1)} = c^{(i)})$$

Invariant Generation - The Method

```
a := 0; b := 0; c := 0;
while (a ≤ k) do
  if A[a] ≥ 0
    then B[b] := A[a];b := b + 1;
    else C[c] := A[a];c := c + 1;
  a := a + 1;
end do
```

1. Extend the language $\mathcal{L}$ to $\mathcal{L}'$:
   - variables as functions of loop cnt n: $v^{(i)}$ with $0 \leq i < n$
   - predicates as loop properties: $iter, upd_V(i,p), upd_V(i,p,x)$

$$(\forall i)(i \in iter \Leftrightarrow 0 \leq i \wedge i < n)$$
$$upd_B(i,p) \Leftrightarrow i \in iter \wedge p = b^{(i)} \wedge A[a^{(i)}] \geq 0$$
$$upd_B(i,p,x) \Leftrightarrow upd_B(i,p) \wedge x = A[a^{(i)}]$$

2. Collect loop properties in $\mathcal{L}'$:

- Polynomial scalar properties
- Monotonicity properties of scalars
- Update predicates of arrays
- Translation of guarded assignments

$$a = b + c,\ a \geq 0,\ b \geq 0,\ c \geq 0$$
$$(\forall i \in iter)(a^{(i+1)} > a^{(i)})$$
$$(\forall i \in iter)(b^{(i+1)} = b^{(i)} \vee b^{(i+1)} = b^{(i)} + 1)$$
$$(\forall i \in iter)(a^{(i)} = a^{(0)} + i)$$
$$(\forall j,k \in iter)(k \geq j \rightarrow b^{(k)} \geq b^{(j)})$$
$$(\forall j,k \in iter)(k \geq j \rightarrow b^{(j)} + k \geq b^{(k)} + j)$$
$$(\forall p)(b^{(0)} \leq p < b^{(n)} \rightarrow (\exists i \in iter)(b^{(i)} = p \wedge A[a^{(i)}] \geq 0))$$
$$(\forall i)\neg upd_B(i,p) \rightarrow B^{(n)}[p] = B^{(0)}[p]$$
$$upd_B(i,p,x) \wedge (\forall j > i)\neg upd_B(j,p) \rightarrow B^{(n)}[p] = x$$
$$(\forall i \in iter)(A[a^{(i)}] \geq 0 \rightarrow B^{(i+1)}[b^{(i)}] = A[a^{(i)}] \wedge b^{(i+1)} = b^{(i)} + 1 \wedge c^{(i+1)} = c^{(i)})$$

3. Eliminate symbols $\rightarrow$ Invariants

Invariant Generation - The Method

```
a := 0; b := 0; c := 0;
while (a ≤ k) do
  if A[a] ≥ 0
    then B[b] := A[a]; b := b + 1;
    else C[c] := A[a]; c := c + 1;
  a := a + 1;
end do
```

1. Extend the language $\mathcal{L}$ to $\mathcal{L}'$:
   - variables as functions of loop cnt n: $v^{(i)}$ with $0 \leq i < n$
   - predicates as loop properties: $iter, upd_V(i,p), upd_V(i,p,x)$

$$(\forall i)(i \in iter \Leftrightarrow 0 \leq i \wedge i < n)$$
$$upd_B(i,p) \Leftrightarrow i \in iter \wedge p = b^{(i)} \wedge A[a^{(i)}] \geq 0$$
$$upd_B(i,p,x) \Leftrightarrow upd_B(i,p) \wedge x = A[a^{(i)}]$$

2. Collect loop properties in $\mathcal{L}'$:

- Polynomial scalar properties
- Monotonicity properties of scalars
- Update predicates of arrays
- Translation of guarded assignments

$$a = b + c,\ a \geq 0,\ b \geq 0,\ c \geq 0$$
$$(\forall i \in iter)(a^{(i+1)} > a^{(i)})$$
$$(\forall i \in iter)(b^{(i+1)} = b^{(i)} \vee b^{(i+1)} = b^{(i)} + 1)$$
$$(\forall i \in iter)(a^{(i)} = a^{(0)} + i)$$
$$(\forall j,k \in iter)(k \geq j \rightarrow b^{(k)} \geq b^{(j)})$$
$$(\forall j,k \in iter)(k \geq j \rightarrow b^{(j)} + k \geq b^{(k)} + j)$$
$$(\forall p)(b^{(0)} \leq p < b^{(n)} \rightarrow (\exists i \in iter)(b^{(i)} = p \wedge A[a^{(i)}] \geq 0))$$
$$(\forall i)\neg upd_B(i,p) \rightarrow B^{(n)}[p] = B^{(0)}[p]$$
$$upd_B(i,p,x) \wedge (\forall j > i)\neg upd_B(j,p) \rightarrow B^{(n)}[p] = x$$

3. Eliminate symbols $\rightarrow$ Invariants

$$(\forall i \in iter)(A[a^{(i)}] \geq 0 \rightarrow B^{(i+1)}[b^{(i)}] = A[a^{(i)}] \wedge b^{(i+1)} = b^{(i)} + 1 \wedge c^{(i+1)} = c^{(i)})$$

Invariant Generation by Symbol Elimination

$$(\forall i)(i \in iter \Leftrightarrow 0 \leq i \wedge i < n)$$
$$upd_B(i,p) \Leftrightarrow i \in iter \wedge p = b^{(i)} \wedge A[a^{(i)}] \geq 0$$
$$upd_B(i,p,x) \Leftrightarrow upd_B(i,p) \wedge x = A[a^{(i)}]$$
$$a = b + c,\ a \geq 0,\ b \geq 0,\ c \geq 0$$
$$(\forall i \in iter)(a^{(i+1)} > a^{(i)})$$
$$(\forall i \in iter)(b^{(i+1)} = b^{(i)} \vee b^{(i+1)} = b^{(i)} + 1)$$
$$(\forall i \in iter)(a^{(i)} = a^{(0)} + i)$$
$$(\forall j,k \in iter)(k \geq j \rightarrow b^{(k)} \geq b^{(j)})$$
$$(\forall j,k \in iter)(k \geq j \rightarrow b^{(j)} + k \geq b^{(k)} + j)$$
$$(\forall p)(b^{(0)} \leq p < b^{(n)} \rightarrow (\exists i \in iter)(b^{(i)} = p \wedge A[a^{(i)}] \geq 0))$$
$$(\forall i) \neg upd_B(i,p) \rightarrow B^{(n)}[p] = B^{(0)}[p]$$
$$upd_B(i,p,x) \wedge (\forall j > i) \neg upd_B(j,p) \rightarrow B^{(n)}[p] = x$$
$$(\forall i \in iter)(A[a^{(i)}] \geq 0 \rightarrow B^{(i+1)}[b^{(i)}] = A[a^{(i)}] \wedge b^{(i+1)} = b^{(i)} + 1 \wedge c^{(i+1)} = c^{(i)})$$

$$\xrightarrow[\text{Theorem Proving}]{\text{Saturation}} I_1, I_2, I_3, I_4, I_5, \ldots$$

Symbol Elimination in Vampire

1. Reasoning in first-order theories

$$
\begin{array}{l}
x \geq y \iff x > y \vee x = y \\
x > y \rightarrow x \neq y \\
x \geq y \wedge y \geq z \rightarrow x \geq z \\
s(x) > x \\
x \geq s(y) \iff x > y
\end{array}
$$

2. Procedures for eliminating symbols $\rightarrow$ USEFUL clauses: Invariants

Symbol Elimination in Vampire

1. Reasoning in first-order theories

$$
\begin{aligned}
& x \geq y \iff x > y \lor x = y \\
& x > y \rightarrow x \neq y \\
& x \geq y \land y \geq z \rightarrow x \geq z \\
& s(x) > x \\
& x \geq s(y) \iff x > y
\end{aligned}
$$

2. Procedures for eliminating symbols $\rightarrow$ USEFUL clauses: Invariants

- For every loop variable v $\rightarrow$ TARGET SYMBOLS v_0 and v:

$$v^{(0)} = v_0 \qquad \text{and} \qquad v^{(n)} = v$$

- USABLE symbols (variables are not symbols):
  - target or interpreted symbols;
  - skolem functions introduced by Vampire;
- USEFUL clauses:
  - contains only usable symbols;
  - contains at least a target symbol or a skolem function;
- Reduction ordering $\succ$: useless symbols $\succ$ usable symbols

Symbol Elimination in Vampire

1. Reasoning in first-order theories

$$
\begin{array}{l}
x \geq y \iff x > y \vee x = y \\
x > y \rightarrow x \neq y \\
x \geq y \wedge y \geq z \rightarrow x \geq z \\
s(x) > x \\
x \geq s(y) \iff x > y
\end{array}
$$

2. Procedures for eliminating symbols $\rightarrow$ USEFUL clauses: Invariants

- For every loop variable v $\rightarrow$ TARGET SYMBOLS v_0 and v:

$$v^{(0)} = v_0 \qquad \text{and} \qquad v^{(n)} = v$$

- USABLE symbols (variables are not symbols):
  - target or interpreted symbols;
  - skolem functions introduced by Vampire;
- USEFUL clauses: $x + y = y + x$ is not useful
  - contains only usable symbols;
  - contains at least a target symbol or a skolem function;
- Reduction ordering $\succ$: useless symbols $\succ$ usable symbols

Symbol Elimination in Vampire

1. Reasoning in first-order theories

$$
\begin{array}{l}
x \geq y \iff x > y \vee x = y \\
x > y \rightarrow x \neq y \\
x \geq y \wedge y \geq z \rightarrow x \geq z \\
s(x) > x \\
x \geq s(y) \iff x > y
\end{array}
$$

2. Procedures for eliminating symbols $\rightarrow$ USEFUL clauses: Invariants

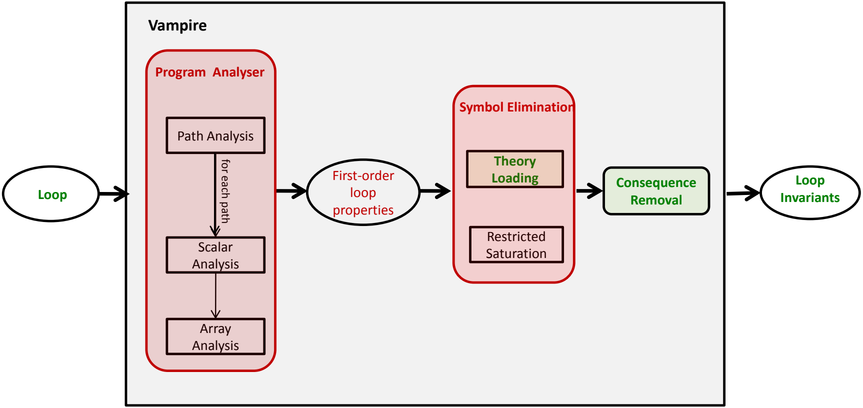
- For every loop variable $v \rightarrow$ TARGET SYMBOLS v_0 and v:

$$v^{(0)} = v_0 \qquad \text{and} \qquad v^{(n)} = v$$

- USABLE symbols (variables are not symbols):
  - target or interpreted symbols;
  - skolem functions introduced by Vampire;
- USEFUL clauses:
  - contains only usable symbols;
  - contains at least a target symbol or a skolem function;
- Reduction ordering $\succ$: useless symbols $\succ$ usable symbols.

Invariant Generation in Vampire

1. Program analysis (new Vampire mode);
2. Theory loading (new Vampire opion);
3. Elimination of "colored" symbols (new Vampire option);
4. Generation of "minimal" set of invariants (new Vampire mode).



1. Sample Output for Program Analysis in Vampire

vampire --mode program_analysis partition.c

```
Loops found: 1
Analyzing loop...
---------------------
while (a < m)
{
if (A[a] >= 0)
{
B[b] = A[a]; b = b + 1;
}
else
{
C[c] = A[a]; c = c + 1;
}
a = a + 1;
}
---------------------
Analyzing variables...
---------------------
Variable: A: constant
Variable: C: (updated)
Variable: m: constant
Variable: b: (updated)
Variable: B: (updated)
Variable: c: (updated)
Variable: a: (updated)
Counter: b
Counter: c
Counter: a
```

```
Collecting paths...
---------------------
Path:
false: A[a] >= 0
C[c] = A[a];
c = c + 1;
a = a + 1;
Path:
true: A[a] >= 0
B[b] = A[a];
b = b + 1;
a = a + 1;
Counter a: 1 min, 1 max, 1 gcd
Counter b: 0 min, 1 max, 1 gcd
Counter c: 0 min, 1 max, 1 gcd
...
Collected first-order loop properties...
---------------------
37. iter(X0) <=> (0<= X0 & X0<n) [program analysis]
...
7. ![X1,X0,X3]:(X1>X0 & c(X1)>X3 & X3>c(X0)) =>
   ?[X2]:(c(X2)=X3 & X2>X0 & X1>X2) [program analysis]
6. ![X0]:c(X0)>=c0 (0:4) [program analysis]
5. ![X0]:c(X0)<=c0+X0 (0:6) [program analysis]
4. ![X1,X0,X3]:(X1>X0 & b(X1)>X3 & X3>b(X0))
   => ?[X2]:(b(X2)=X3 & X2>X0 & X1>X2) [program
analysis]
3. ![X0]:b(X0)>=b0 (0:4) [program analysis]
2. ![X0]:b(X0)<=b0+X0 (0:6) [program analysis]
1. ![X0]:a(X0)=a0+X0 (0:6) [program analysis]
```

Figure: Partial output of Vampire's program analyser on the Partition program.

2. Theory Loading in Vampire

We use incomplete but sound theory axiomatisation (Session 2).

Example: Integers in Vampire

- 0, 1, 2, etc;
- Integer predicates/funcions:
  - addition;
  - subtraction;
  - multiplication;
  - successor;
  - division;
  - inequality relations;

3. Elimination of Colored Symbols

vampire(option,show_symbol_elimination,on).

```
vampire(option,time_limit,1).

...
tff(b_type,type,a:$int).
tff(b_fcttype,type,a:$int>$int).
tff(bb_type,type,bb:$int>$int).
tff(bb_fct2type,type,bb:($int*$int)>$int).
tff(iter_fcttype,type,iter:$int>$o).
tff(upd2_type,type,updbb:($int*$int)>$o).
tff(upd3_type,type,updbb:($int*$int*$int)>$o).
...

vampire(symbol,function,n,0,left).
vampire(symbol,function,a,1,left).
vampire(symbol,function,b,1,left).
vampire(symbol,function,c,1,left).
vampire(symbol,function,bb,2,left).
vampire(symbol,function,cc,2,left).
vampire(symbol,predicate,updB,2,left).
vampire(symbol,predicate,updB,3,left).
vampire(symbol,predicate,updC,2,left).
vampire(symbol,predicate,updC,3,left).
vampire(symbol,predicate,iter,1,left).

vampire(symbol,function,a,0,skip).
vampire(symbol,function,b,0,skip).
vampire(symbol,function,c,0,skip).
vampire(symbol,function,m,0,left).
vampire(symbol,function,aa,1,skip).
vampire(symbol,function,bb0,2,skip).
vampire(symbol,function,bb0,1,skip).
vampire(symbol,function,cc0,2,skip).
vampire(symbol,function,cc0,1,skip).
```

Figure: Partial input for symbol elimination in Vampire.

```
./vampire array_partition.tptp
```

4. Generation of Minimal Set of Invariants

Set of invariants: S

Minimal set S' of invariants with $S' \subset S$:

Remove $C \in S$ iff $S \setminus \{C\} \Rightarrow C$

Compute $S' \subset S$

Run Vampire on S within, e.g., 20s time limit

Experiments:

- consequence elimination ran in conjunction with 4 combination of strategies
- eliminated $\sim$ 80% invariants

4. Generation of Minimal Set of Invariants

```
vampire --mode consequence_elimination
```

Set of invariants: S

Minimal set S' of invariants with $S' \subset S$:

Remove $C \in S$ iff $S \setminus \{C\} \Rightarrow C$

Compute $S' \subset S$

Run Vampire on S within, e.g., 20s time limit

Experiments:

- consequence elimination ran in conjunction with 4 combination of strategies
- eliminated $\sim$ 80% invariants

4. Generation of Minimal Set of Invariants

```
vampire --mode consequence_elimination
```

Set of invariants: S

Minimal set S' of invariants with $S' \subset S$:

Remove $C \in S$ iff $S \setminus \{C\} \Rightarrow C$

Compute $S' \subset S$

Run Vampire on S within, e.g., 20s time limit

Experiments:

- consequence elimination ran in conjunction with 4 combination of strategies
- eliminated $\sim$ 80% invariants

Summary: Invariant Generation and Symbol Elimination

Given a loop:

1. Express loop properties in a language containing extra symbols (loop counter, predicates expressing array updates, etc.);

2. Every logical consequence of these properties is a valid loop property, but not an invariant;

3. Run a theorem prover for eliminating extra symbols;

4. Every derived formula in the language of the loop is a loop invariant;

5. Invariants are **consequences of symbol-eliminating inferences (SEI)**.

SEI: premise contains extra symbols, conclusion is in the loop language.

Symbol Elimination and Interpolation

{Pre} Loop

P_1
P_2
$\vdots$
P_n

A

$\Longrightarrow$ Invariant/Interpolant $\Longrightarrow$

{Post}

B

Symbol Elimination and Interpolation

{Pre} Loop

A

P_1
P_2
$\vdots$
P_n

$\Longrightarrow$ Invariant/Interpolant $\Longrightarrow$

{Post}

B

Symbol Elimination and Interpolation

{Pre} Loop

A

P_1
P_2
$\vdots$
P_n

$\Longrightarrow$ Invariant/Interpolant $\Longrightarrow$

{Post}

B

Q_1
Q_2
$\vdots$
Q_m

Outline

Part I: Quantified Invariants and Symbol Elimination

Part II: Symbol Elimination and Interpolation

Summary: Invariant Generation, Interpolation, Symbol Elimination

Invariants, Symbol Elimination, and Interpolation

$\{c = d = 0 \ \wedge N > 0 \ \wedge \ (\forall k) \ (0 \leq k < N \rightarrow D[k] = 0)\}$ precondition $A(c, d)$

while $(c < N)$ **do**

$C[c] \ := \ D[d];$

$c \ := \ c + 1;$
$d \ := \ d + 1$

end do

$\{(\forall k)(0 \leq k < N \rightarrow C[k] = 0)\}$ postcondition $B(c, d)$

Invariants, Symbol Elimination, and Interpolation

Reachability of B in ONE iteration: $A(c,d) \wedge T(c,d,c',d') \rightarrow B(c',d')$

$\{c = d = 0 \ \wedge N > 0 \ \wedge \ (\forall k)\ (0 \leq k < N \rightarrow D[k] = 0)\}$ precondition $A(c,d)$

while $(c < N)$ **do**

$C[c] \ := \ D[d];$ $\underbrace{c < N \wedge C[c] = D[d] \wedge c' = c+1 \wedge d' = d+1 \wedge c' \geq N}_{T(c,d,c',d')}$

$c \ := \ c+1;$
$d \ := \ d+1$
end do

$\{(\forall k)(0 \leq k < N \rightarrow C[k] = 0)\}$ postcondition $B(c',d')$

Invariants, Symbol Elimination, and Interpolation

Reachability of B in ONE iteration: $A(c,d) \wedge T(c,d,c',d') \rightarrow B(c',d')$

$\{c = d = 0 \ \wedge N > 0 \ \wedge \ (\forall k)\ (0 \leq k < N \rightarrow D[k] = 0)\}$ precondition $A(c,d)$

while $(c < N)$ **do**

$C[c] \ := \ D[d];$ $\underbrace{c < N \wedge C[c] = D[d] \wedge c' = c + 1 \wedge d' = d + 1 \wedge c' \geq N}_{T(c,d,c',d')}$

$c \ := \ c + 1;$
$d \ := \ d + 1$

end do

$\{(\forall k)(0 \leq k < N \rightarrow C[k] = 0)\}$ postcondition $B(c',d')$

Refutation: $A(c,d) \wedge T(c,d,c',d') \wedge \neg B(c',d')$

- The formula is of 2 states (c, d, c', d').
- Need a state formula $I(c',d')$ such that: (Jhala and McMillan)

$A(c,d) \wedge T(c,d,c',d') \rightarrow I(c',d')$ and $I(c',d') \wedge \neg B(c',d') \rightarrow \perp$

Invariants, Symbol Elimination, and Interpolation

Reachability of B in ONE iteration: $A(c,d) \wedge T(c,d,c',d') \rightarrow B(c',d')$

$\{c = d = 0 \ \wedge N > 0 \ \wedge \ (\forall k)\ (0 \leq k < N \rightarrow D[k] = 0)\}$ precondition $A(c,d)$

while $(c < N)$ **do**

$C[c] \ := \ D[d];$ $\underbrace{c < N \wedge C[c] = D[d] \wedge c' = c+1 \wedge d' = d+1 \wedge c' \geq N}_{T(c,d,c',d')}$

$c \ := \ c+1;$
$d \ := \ d+1$

end do

$\{(\forall k)(0 \leq k < N \rightarrow C[k] = 0)\}$ postcondition $B(c',d')$

Refutation: $A(c,d) \wedge T(c,d,c',d') \wedge \neg B(c',d')$

- The formula is of 2 states (c, d, c', d').
- Need a state formula $I(c',d')$ such that: (Jhala and McMillan)

$$A(c,d) \wedge T(c,d,c',d') \rightarrow I(c',d') \quad \text{and} \quad I(c',d') \wedge \neg B(c',d') \rightarrow \perp$$

Invariants, Symbol Elimination, and Interpolation

Reachability of B in ONE iteration: $A(c,d) \wedge T(c,d,c',d') \rightarrow B(c',d')$

$\{c = d = 0 \ \wedge N > 0 \ \wedge \ (\forall k)\ (0 \leq k < N \rightarrow D[k] = 0)\}$ precondition $A(c,d)$

while $(c < N)$ **do**

$C[c] := D[d];$ $\quad \underbrace{c < N \wedge C[c] = D[d] \wedge c' = c + 1 \wedge d' = d + 1 \wedge c' \geq N}_{T(c,d,c',d')}$

$c := c + 1;$
$d := d + 1$

end do

$\{(\forall k)(0 \leq k < N \rightarrow C[k] = 0)\}$ postcondition $B(c',d')$

Refutation: $A(c,d) \wedge T(c,d,c',d') \wedge \neg B(c',d')$
• The formula is of 2 states (c, d, c', d').
• Need a state formula $I(c',d')$ such that: (Jhala and McMillan)
$A(c,d) \wedge T(c,d,c',d') \rightarrow I(c',d')$ and $I(c',d') \wedge \neg B(c',d') \rightarrow \perp$

Taks: Compute interpolant $I(c',d')$ by eliminating symbols c, d.

Invariants, Symbol Elimination, and Interpolation

Reachability of B in ONE iteration: $A(c,d) \wedge T(c,d,c',d') \rightarrow B(c',d')$

$\{c = d = 0 \ \wedge N > 0 \ \wedge \ (\forall k)\ (0 \leq k < N \rightarrow D[k] = 0)\}$ precondition $A(c,d)$

while $(c < N)$ **do**

$C[c] \ := \ D[d];$ $\quad \underbrace{c < N \wedge C[c] = D[d] \wedge c' = c + 1 \wedge d' = d + 1 \wedge c' \geq N}_{T(c,d,c',d')}$

$c \ := \ c + 1;$

$d \ := \ d + 1$

end do

$\{(\forall k)(0 \leq k < N \rightarrow C[k] = 0)\}$ postcondition $B(c',d')$

$$I(c',d') \quad \equiv \quad 0 < c' = 1 \wedge C[0] = D[0]$$

$$I(c'',d'') \quad \equiv \quad 0 < c'' = 2 \wedge C[0] = D[0] \wedge C[1] = D[1]$$

Taks: Compute interpolant $I(c',d')$ by eliminating symbols c, d.

Invariants, Symbol Elimination, and Interpolation

Reachability of B in TWO iterations: $A(c,d) \wedge T(c,d,c',d') \wedge T(c',d',c'',d'') \rightarrow B(c'',d'')$

$\{c = d = 0 \ \wedge N > 0 \ \wedge \ (\forall k)\ (0 \leq k < N \rightarrow D[k] = 0)\}$ precondition $A(c,d)$

while $(c < N)$ **do**

$C[c] \ := \ D[d];$

$c \ := \ c + 1;$
$d \ := \ d + 1$

end do

$\{(\forall k)(0 \leq k < N \rightarrow C[k] = 0)\}$ postcondition $B(c', d')$

$$I(c',d') \equiv 0 < c' = 1 \wedge C[0] = D[0]$$
$$I(c'',d'') \equiv 0 < c'' = 2 \wedge C[0] = D[0] \wedge C[1] = D[1]$$

Taks: Compute interpolant $I(c'',d'')$ by eliminating symbols c, d, c', d'.

Invariants, Symbol Elimination, and Interpolation

Reachability of B in TWO iterations: $A(c,d) \wedge T(c,d,c',d') \wedge T(c',d',c'',d'') \rightarrow B(c'',d'')$

$\{c = d = 0 \ \wedge N > 0 \ \wedge \ (\forall k)\ (0 \leq k < N \rightarrow D[k] = 0)\}$ precondition $A(c,d)$

while $(c < N)$ **do**

$C[c] \ := \ D[d];$

$c \ := \ c + 1;$
$d \ := \ d + 1$

end do

$\{(\forall k)(0 \leq k < N \rightarrow C[k] = 0)\}$ postcondition $B(c',d')$

$$I(c',d') \quad \equiv \quad (\forall k) 0 \leq k < c' \rightarrow C[k] = D[k]$$
$$I(c'',d'') \quad \equiv \quad (\forall k) 0 \leq k < c'' \rightarrow C[k] = D[k]$$

Taks: Compute interpolant $I(c'',d'')$ implying invariant in any state.

Interpolation in Vampire: Outline

What is an Interpolant?

Computing Interpolants

- Local Derivations and Symbol Elimination
- Interpolation in Vampire

Notation

- First-order predicate logic with equality.
- $\top$: always true,
  $\bot$: always false.
- $\forall A$: universal closure of A.
- Symbols:
  - predicate symbols;
  - function symbols;
  - constants.

  Equality is part of the language $\rightarrow$ equality is not a symbol.
- $\mathcal{L}_A$: the language of A: the set of all formulas built from the symbols occurring in A.

What is an Interpolant?

Let A, B be closed formulas such that $A \rightarrow B$.

Theorem (Craig's Interpolation Theorem)

There exists a closed formula $I \in \mathcal{L}_A \cap \mathcal{L}_B$ such that

$$A \rightarrow I \quad \textit{and} \quad I \rightarrow B.$$

I is an interpolant of A and B.

Note: if A and B are ground, they also have a ground interpolant.

What is an Interpolant?

Let A, B be closed formulas such that $A \to B$.

Theorem (Craig's Interpolation Theorem)

There exists a closed formula $I \in \mathcal{L}_A \cap \mathcal{L}_B$ such that

$$A \to I \quad \textit{and} \quad I \to B.$$

I is an interpolant of A and B.

$\boxed{\text{Reverse interpolant of } A \text{ and } B\text{:}}$ any formula I such that

$$A \to I \quad \text{and} \quad I, \neg B \to \bot.$$

Interpolation with Theories

- Theory T: any set of closed formulas.
- $C_1, \ldots, C_n \rightarrow_T C$ means that the formula $C_1 \wedge \ldots \wedge C_1 \rightarrow C$ holds in all models of T.
- Interpreted symbols: symbols occurring in T.
- Uninterpreted symbols: all other symbols.

Theorem

Let A, B be formulas and let $A \rightarrow_T B$.

Then there exists a formula I such that

1. $A \rightarrow_T I$ *and* $I \rightarrow B$;
2. *every uninterpreted symbol of I occurs both in A and B;*
3. *every interpreted symbol of I occurs in B.*

Likewise, there exists a formula I such that

1. $A \rightarrow I$ *and* $I \rightarrow_T B$;
2. *every uninterpreted symbol of I occurs both in A and B;*
3. *every interpreted symbol of I occurs in A.*

Interpolation with Theories

- Theory T: any set of closed formulas.
- $C_1, \ldots, C_n \rightarrow_T C$ means that the formula $C_1 \wedge \ldots \wedge C_1 \rightarrow C$ holds in all models of T.
- Interpreted symbols: symbols occurring in T.
- Uninterpreted symbols: all other symbols.

Theorem

Let A, B be formulas and let $A \rightarrow_T B$.

Then there exists a formula I such that

1. $A \rightarrow_T I$ *and* $I \rightarrow B$;
2. *every uninterpreted symbol of I occurs both in A and B;*
3. *every interpreted symbol of I occurs in B.*

Likewise, there exists a formula I such that

1. $A \rightarrow I$ *and* $I \rightarrow_T B$;
2. *every uninterpreted symbol of I occurs both in A and B;*
3. *every interpreted symbol of I occurs in A.*

Computing Interpolants using Inference Systems

- Inference Rule:

$$\frac{A_1 \quad \dots \quad A_n}{A}$$

- Inference system: a set of inference rules.
- Axiom: an inference rule with 0 premises.
- Derivation of A: tree with the root A built from inferences.

Interpolants and Local AB-Derivations

AB-derivation

Let $\mathcal{L} = \mathcal{L}_A \cap \mathcal{L}_B$.

A derivation Π is an AB-derivation if

(AB1) For every leaf C of Π one of following conditions holds:

1. $A \rightarrow_T \forall C$ and $C \in \mathcal{L}_A$ or
2. $B \rightarrow_T \forall C$ and $C \in \mathcal{L}_B$.

(AB2) For every inference

$$\frac{C_1 \quad \ldots \quad C_n}{C}$$

of Π we have $\forall C_1, \ldots, \forall C_n \rightarrow_T \forall C$.

We will refer to property (AB2) as soundness.

Interpolants and Local AB-Derivations

$$\frac{C_1 \quad \dots \quad C_n}{C}$$

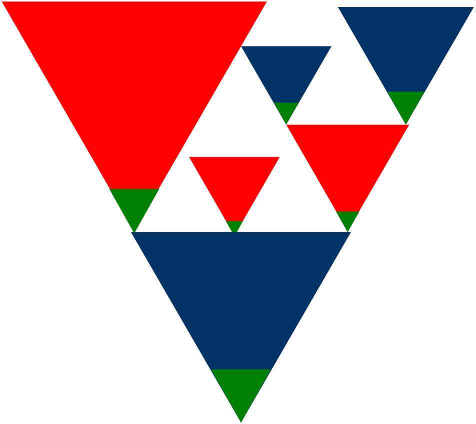
This inference is local if the following two conditions hold:

(L1) Either $\{C_1, \dots, C_n, C\} \subseteq \mathcal{L}_A$ or $\{C_1, \dots, C_n, C\} \subseteq \mathcal{L}_B$.

(L2) If all of the formulas $C_1, \dots, C_n$ are colorless, then C is colorless, too.

A derivation is called local if so is every inference of this derivation.

Shape of local derivations for $A \to B$



Local Derivations: Example $A \rightarrow B$

interpolation1.tptp

- A := $\forall x(x = c)$
- B := $a = b$
- Universal interpolant I: $\forall x \forall y(x = y)$

A local refutation in the superposition calculus:

$$\dfrac{\dfrac{\dfrac{x = c \quad y = c}{x = y} \qquad a \neq b}{y \neq b}}{\bot}$$

Local Derivations: Example $A \to B$

interpolation1.tptp

- A := $\forall x(x = c)$
- B := $a = b$
- Universal interpolant I: $\forall x \forall y(x = y)$

A local refutation in the superposition calculus:

$$\cfrac{\cfrac{x = c \quad y = c}{x = y} \qquad a \neq b}{\cfrac{y \neq b}{\bot}}$$

Interpolants and Symbol Eliminating Inference

- At least one of the premises colored.
- The conclusion is not colored.

$$\dfrac{\boxed{\dfrac{x = c \quad y = c}{x = y}} \qquad a \neq b}{\boxed{\dfrac{y \neq b}{\bot}}}$$

Interpolant $\forall x \forall y (x = y)$: conclusion of a symbol-eliminating inference.

Interpolants and Symbol Eliminating Inference

- At least one of the premises colored.
- The conclusion is not colored.

$$\dfrac{\boxed{\dfrac{x = c \quad y = c}{x = y}} \qquad a \neq b}{\boxed{\dfrac{y \neq b}{\perp}}}$$

Interpolant $\forall x \forall y (x = y)$: conclusion of a symbol-eliminating inference.

Interpolation in Vampire

$A, B \rightarrow \perp$ where:

$A := Q(f(a)) \wedge \neg(Q(f(b)))$

$B := \forall x(f(x) = c)$

1. Generation of interpolants (new Vampire option).
2. Color specification of left-formula A and right-formula B;
3. Color-annotated formulas A and B;
4. Theory loading;

   interpolation3.tptp
5. Restricted saturation: local proofs and symbol elimination.

Interpolation in Vampire

$A, B \to \perp$ where:

$A := Q(f(a)) \wedge \neg(Q(f(b)))$

$B := \forall x(f(x) = c)$

1. Generation of interpolants `vampire(option, show_interpolant, on).` (new Vampire option).
2. Color specification of left-formula A and right-formula B;
3. Color-annotated formulas A and B;
4. Theory loading;

   interpolation3.tptp
5. Restricted saturation: local proofs and symbol elimination.

Interpolation in Vampire

$A, B \rightarrow \bot$ where:

$A := Q(f(a)) \wedge \neg(Q(f(b)))$

$B := \forall x(f(x) = c)$

1. Generation of interpolants (new Vampire option).

```
vampire(option, show_interpolant, on).
```

2. Color specification of left-formula A and right-formula B;

```
vampire(symbol, predicate, q, 1, left).
vampire(symbol, function, a, 0, left).
vampire(symbol, predicate, b, 0, left).
vampire(symbol, predicate, c, 1, right).
```

3. Color-annotated formulas A and B;

4. Theory loading;

   interpolation3.tptp

5. Restricted saturation: local proofs and symbol elimination.

Interpolation in Vampire

$A, B \to \perp$ where:

$A := Q(f(a)) \wedge \neg(Q(f(b)))$

$B := \forall x(f(x) = c)$

1. Generation of interpolants (new Vampire option).
2. Color specification of left-formula A and right-formula B;
3. Color-annotated formulas A and B;
4. Theory loading;

   interpolation3.tptp
5. Restricted saturation: local proofs and symbol elimination.

```
vampire(option, show_interpolant, on).

vampire(symbol, predicate, q, 1, left).
vampire(symbol, function, a, 0, left).
vampire(symbol, predicate, b, 0, left).
vampire(symbol, predicate, c, 1, right).

vampire(left_formula).
  fof(fA,hypothesis,( q(f(a))& q(f(b)) )).
vampire(end_formula).

vampire(right_formula).
  fof(fB,hypothesis,( f(X)=c )).
vampire(end_formula).
```

interpolation2.tptp

Interpolation in Vampire

$A, B \rightarrow \bot$ where:

$A := Q(f(a)) \wedge \neg(Q(f(b)))$

$B := \forall x(f(x) = c)$

1. Generation of interpolants (new Vampire option).

```
vampire(option, show_interpolant, on).
```

2. Color specification of left-formula A and right-formula B;

```
vampire(symbol, predicate, q, 1, left).
vampire(symbol, function, a, 0, left).
vampire(symbol, predicate, b, 0, left).
vampire(symbol, predicate, c, 1, right).
```

3. Color-annotated formulas A and B;

```
vampire(left_formula).
  fof(fA,hypothesis,( q(f(a))& q(f(b)) )).
vampire(end_formula).

vampire(right_formula).
  fof(fB,hypothesis,( f(X)=c )).
vampire(end_formula).
```

4. Theory loading;

   interpolation3.tptp

5. Restricted saturation: local proofs and symbol elimination.

interpolation2.tptp

Interpolation in Vampire

$A, B \to \bot$ where:

$A := Q(f(a)) \land \neg(Q(f(b)))$

$B := \forall x(f(x) = c)$

$\boxed{I := \exists x_0, x_1(f(x_0) \neq f(x_1))}$

1. Generation of interpolants (new Vampire option).

```
vampire(option, show_interpolant, on).
```

2. Color specification of left-formula A and right-formula B;

```
vampire(symbol, predicate, q, 1, left).
vampire(symbol, function, a, 0, left).
vampire(symbol, predicate, b, 0, left).
vampire(symbol, predicate, c, 1, right).
```

3. Color-annotated formulas A and B;

```
vampire(left_formula).
  fof(fA,hypothesis,( q(f(a))& q(f(b)) )).
vampire(end_formula).

vampire(right_formula).
  fof(fB,hypothesis,( f(X)=c )).
vampire(end_formula).
```

4. Theory loading;

interpolation3.tptp

5. Restricted saturation: local proofs and symbol elimination.

`interpolation2.tptp`

Interpolation in Vampire: Outline

Part I: Quantified Invariants and Symbol Elimination

Part II: Symbol Elimination and Interpolation

Summary: Invariant Generation, Interpolation, Symbol Elimination

Summary: Invariant Generation, Interpolation, Symbol Elimination

Given the proof obligation $A \to B$:

1. Run a theorem prover and eliminate extra symbols;
2. Generate a (reverse) interpolant from a refutation;
3. Interpolant is a boolean combination of consequences of **symbol-eliminating inferences**.

Given a loop:

1. Express loop properties in a language containing extra symbols;
2. Every logical consequence of these properties is a valid loop property, but not an invariant;
3. Run a theorem prover for eliminating extra symbols;
4. Every derived formula in the language of the loop is a loop invariant;
5. Invariants are consequences of **symbol-eliminating inferences**.

Summary: Invariant Generation, Interpolation, Symbol Elimination

Given the proof obligation $A \rightarrow B$:

1. Run a theorem prover and eliminate extra symbols;
2. Generate a (reverse) interpolant from a refutation;
3. Interpolant is a boolean combination of consequences of **symbol-eliminating inferences**.

Given a loop:

1. Express loop properties in a language containing extra symbols;
2. Every logical consequence of these properties is a valid loop property, but not an invariant;
3. Run a theorem prover for eliminating extra symbols;
4. Every derived formula in the language of the loop is a loop invariant;
5. Invariants are consequences of **symbol-eliminating inferences**.