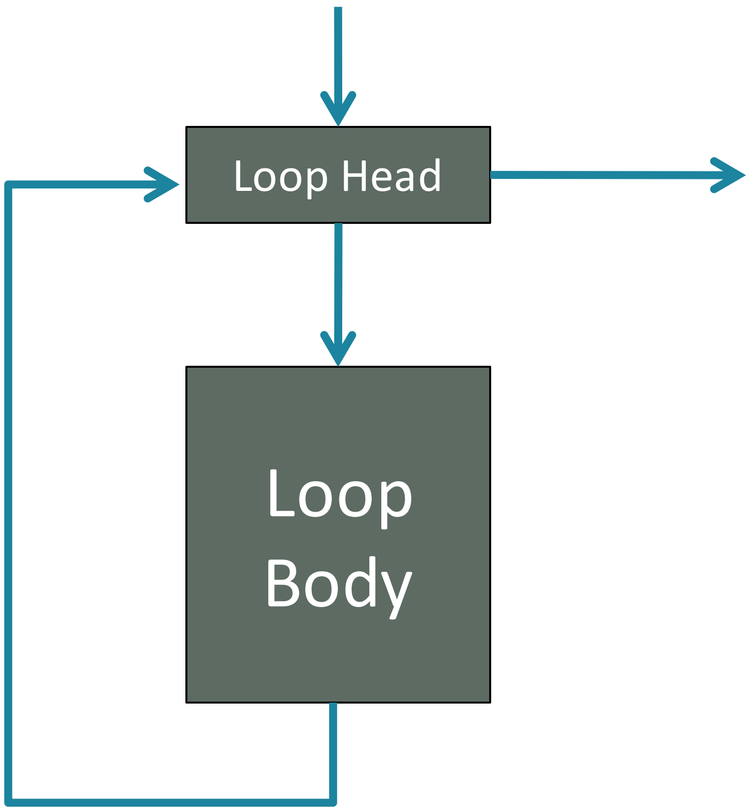
Simplifying Loop Invariant Generation Using Splitter Predicates

Rahul Sharma
Işil Dillig, Thomas Dillig, and Alex Aiken
Stanford University

Loops and Loop Invariants



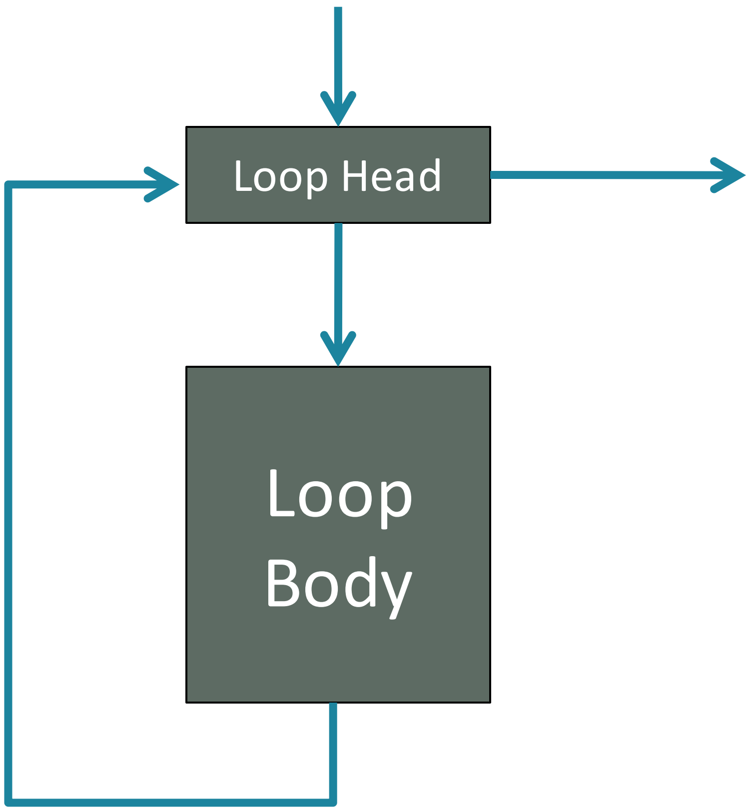

```
x = 0;

while( x <= 49 )
{

   x = x + 1;

}
```

Loops and Loop Invariants



- Key problem
  - Software verification
- Is undecidable

Conjunctive Invariants

```
x = 0; y = 50;
while( x <= 49 )
{
  x = x + 1;
}
```

```
 x <= 50
&& y = 50
&& x >= 0
```

Conjunctive Invariants

- Mature techniques for discovering these

- Cousot and Halbwachs, 1978
  - Abstract interpretation-based technique (Interproc)
  - Keep estimate of reachable states

- StInG and InvGen
  - Assume an invariant template (a.x + b.y <= c)
  - Solve (non-linear) constraints to instantiate template

Disjunctive Invariants

- Some loops require disjunctive invariants

- Some very involved techniques:
  - Probabilistic inference, predicate abstraction, …
  - Difficult to characterize their behavior
  - Most are not fully automatic

Disjunctive Invariants

- Some loops require disjunctive invariants

- Some very involved techniques:
  - Probabilistic inference, predicate abstraction, …
  - Difficult to characterize their behavior
  - Most are not fully automatic

- **Can we reduce disjunctive to conjunctive?**

A Small Study

- About 10% of loops require disjunctive invariants
  - (OpenSSH 9/95)
- Of these about 90% are *multi-phase*
  - (OpenSSH 8/9)

Loops Requiring Disjunctive Invariants

Digikam

```
for(int i = 0; i < num; i++)
{
  if( i == 0 )
  {
     delete w[i];
     continue;
  }
  w[i-1] = w[i];
}
```

Loops Requiring Disjunctive Invariants

OpenSSH

```
for(int i=0; i< size; i++)
{
  if( i == size-1 )
    a[i] = NULL;
  else
    a[i] = malloc(...);
 }
```

Multi-phase Loops

```
x = 0; y = 50;
while( x < 100 )
{
  x = x + 1;
  if( x > 50 )
    y = y + 1;
}
assert( y == 100);
```

```
  (x<=50 && y=50)
||(50<=x<=100 && y=x)
```

Multi-phase Loops

```
x = 0; y = 50;
while( x < 100 )
{
  x = x + 1;
  if( x > 50 )
    y = y + 1;
}
assert( y == 100);
```

  (x<=50 && y=50)
||(50<=x<=100 && y=x)

x = 0, x = 1, …, x = 49

Multi-phase Loops

```
x = 0; y = 50;
while( x < 100 )
{
  x = x + 1;
  if( x > 50 )
    y = y + 1;
}
assert( y == 100);
```

(x<=50 && y=50)
||(50<=x<=100 && y=x)

x = 0, x = 1, …, x = 49

x = 50, x = 51, …, x = 99

Multi-phase Loops Continued…

- Phase transition
  - C : predicate of `if` inside loop
  - Phase: contiguous iterations with C constant
  - Transition: C changes value
- Restrict to 2-phase loops
  - C transitions from false to true

Basic Idea

```
x = 0; y = 50;
while( x < 100 )
{
  x = x + 1;
  if( x > 50 )
    y = y + 1;
}
assert( y == 100);
```

Basic Idea

Disjunctive Invariant

```
x = 0; y = 50;
while( x < 100 )
{
  x = x + 1;
  if( x > 50 )
    y = y + 1;
}
assert( y == 100);
```

```
 (x<=50 && y=50)
||(50<=x<=100 && y=x)
```

Basic Idea

Disjunctive Invariant

```
x = 0; y = 50;
while( x < 100 )
{
  x = x + 1;
  if( x > 50 )
    y = y + 1;
}
assert( y == 100);

 (x<=50 && y=50)
||(50<=x<=100 && y=x)
```

```
x = 0; y = 50;
while( x <= 49 )
{
  x = x + 1;
}
while( x < 100 && x > 49)
{
  x = x + 1;
  y = y + 1;
}
assert( y == 100);
```

Basic Idea

Disjunctive Invariant

```
x = 0; y = 50;
while( x < 100 )
{
  x = x + 1;
  if( x > 50 )
    y = y + 1;
}
assert( y == 100);
```

(x<=50 && y=50)
||(50<=x<=100 && y=x)

Conjunctive Invariants!

```
x = 0; y = 50;
while( x <= 49 )
{                        x<=50
  x = x + 1;           && y=50
}
while( x < 100 && x > 49)
{
  x = x + 1;  50<=x<=100
  y = y + 1;      && y=x
}
assert( y == 100);
```

Our Goal

Reduce the problem of inferring disjunctive invariants for a multi-phase loop to inferring conjunctive invariants for a sequence of loops

Our Approach

- Identify multi-phase loops
- Split the multi-phase loops
- Use standard invariant generation technique
  - Infer conjunctive invariants

Example

Before Splitting

```
x = 0; y = 50;
while( x < 100 )
{
  x = x + 1;
  if( x > 50 )
    y = y + 1;
}
assert( y == 100);
```

After Splitting

```
x = 0; y = 50;
while( x<100 && x <= 49 )
{
  x = x + 1;
  if( false ) y = y + 1;
}
while( x < 100 && x > 49 )
{
  x = x + 1;
  if( true ) y = y + 1;
}
assert( y == 100);
```

Splitter Predicate (Q)

```
while(P) { B [C] }
```

↓

```
while(P && !Q) { B [false] }
while(P && Q) { B [true] }
```

Last Notational Hurdle

```
x = 0; y = 50;
while( x < 100 )
{
   x = x + 1;
   if( x > 50 )
     y = y + 1;
}
assert( y == 100);
```

$\overline{B}$: Code present

- Inside the loop body
- Before control reaches C

Properties of Splitter Predicates

Theorem:

For a loop `while P { B[C] }`, if Q satisfies these three properties then it is a splitter predicate.

1. $\{Q\}\ \overline{B}\ \{C\}$
2. $\{\neg Q\}\ \overline{B}\ \{\neg C\}$
3. $\{P \wedge Q\}\ B[C]\ \{Q \vee \neg P\}$

Splitter Predicate

```
while(P) { B [C] }
```

↓

```
while(P && !Q) { B [C] }
 while(P && Q) { B [C] }
```

Splitter Predicate

```
while(P) { B [C] }
```

```
while(P && !Q) { B [false] }
while(P && Q) { B [C] }
```

Splitter Predicate

```
while(P) { B [C] }
```

```
while(P && !Q) { B [false] }
while(P && Q) { B [true] }
```

Algorithm

- For each condition $\texttt{if}_\rho(\ C\ )$ of loop body
  - Q = *WP* ($\overline{\mathrm{B}}$, C)
  - Check conditions
- If Q is a splitter predicate then
  - Split the multi-phase loop
  - Replace $\texttt{if}_\rho(\ C\ )$ by $\texttt{if}_\rho(\ \text{false}\ )$ and $\texttt{if}_\rho(\ \text{true}\ )$

Example Revisited

Before Splitting

```
x = 0; y = 50;
while( x < 100 )
{
  x = x + 1;
  if( x > 50 )
    y = y + 1;
}
assert( y == 100);
```

$Q = WP\,(x = x + 1, x > 50)$

$Q = x > 49$

Example Revisited

Before Splitting

```
x = 0; y = 50;
while( x < 100 )
{
  x = x + 1;
  if( x > 50 )
    y = y + 1;
}
assert( y == 100);
```

After Splitting

```
x = 0; y = 50;
while( x<100 && x <= 49 )
{
  x = x + 1;
  if( false ) y = y + 1;
}
while( x < 100 && x > 49 )
{
  x = x + 1;
  if( true ) y = y + 1;
}
assert( y == 100);
```

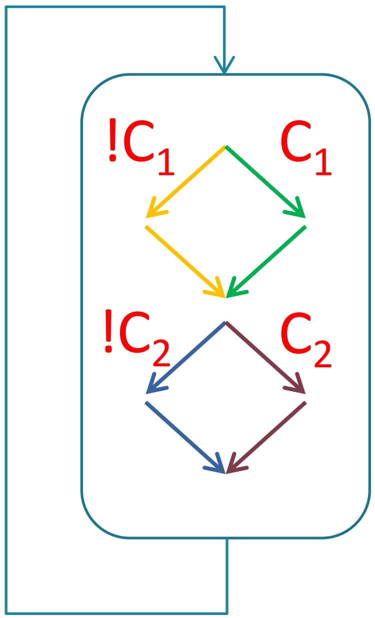
Optimizations

- Simplify precondition computation
  - Process `if` top down
  - For nested loops, process inside out

Optimizations

- Simplify precondition computation
  - Process `if` top down
  - For nested loops, process inside out

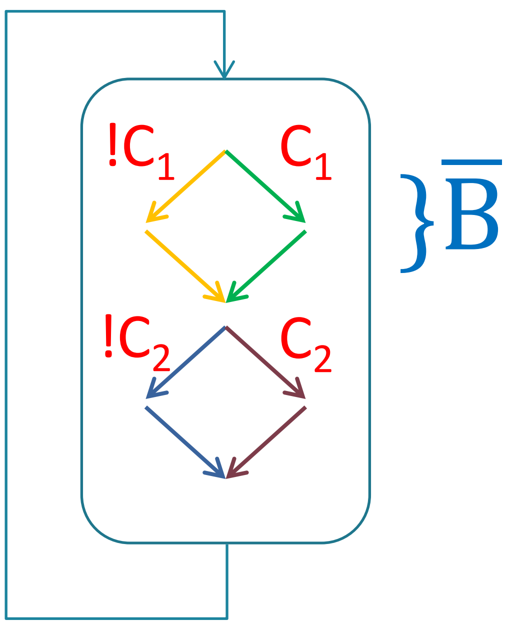
B[C_1, C_2]



Optimizations

- Simplify precondition computation
  - Process `if` top down
  - For nested loops, process inside out

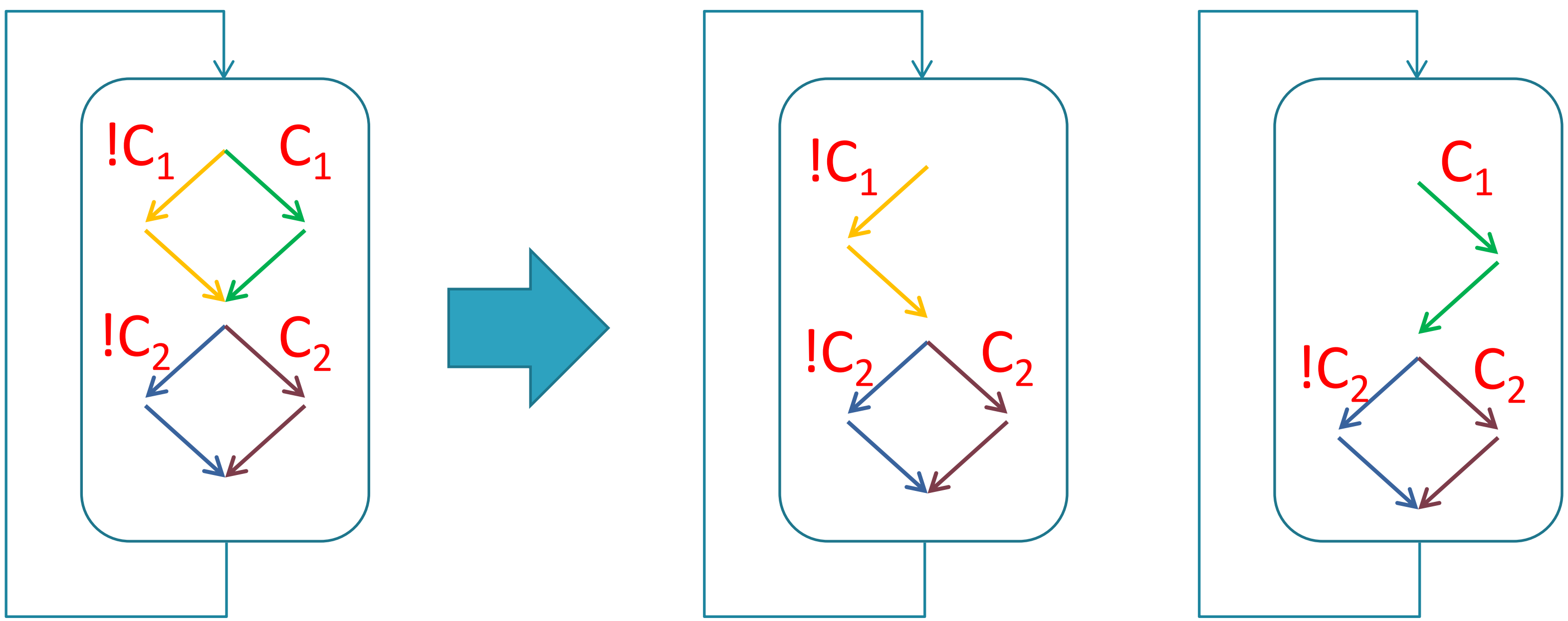
B[C_1, C_2]



Optimizations

- Simplify precondition computation
  - Process `if` top down
  - For nested loops, process inside out

B[C_1, C_2]



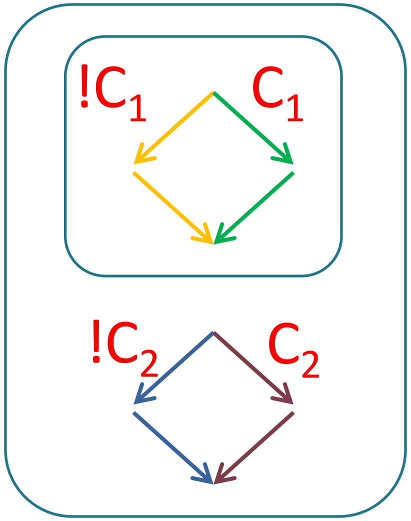
Optimizations

- Simplify precondition computation
  - Process `if` top down
  - For nested loops, process inside out

Optimizations

- Simplify precondition computation
  - Process `if` top down
  - For nested loops, process inside out

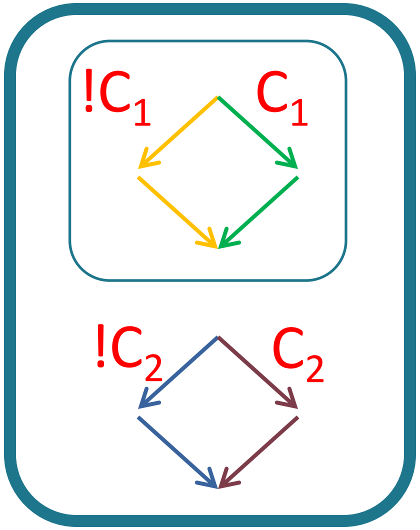
$B_i[C_1]\ B_o[C_2]$



Optimizations

- Simplify precondition computation
  - Process `if` top down
  - For nested loops, process inside out

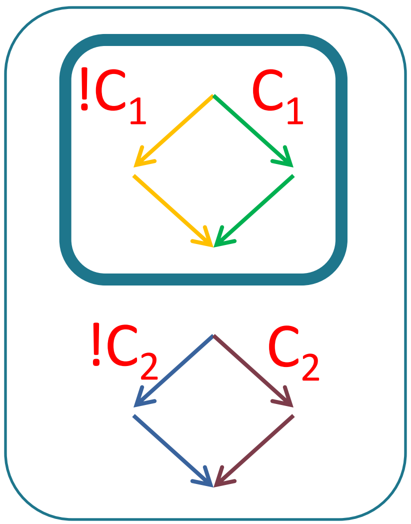
$B_i[C_1]\ B_o[C_2]$



Optimizations

- Simplify precondition computation
  - Process `if` top down
  - For nested loops, process inside out

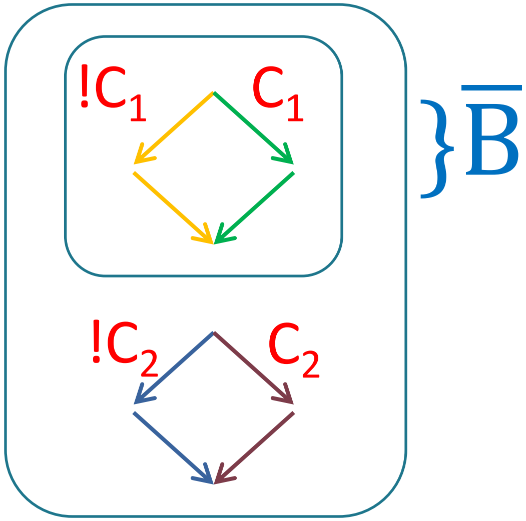
$B_i[C_1]$ $B_o[C_2]$



Optimizations

- Simplify precondition computation
  - Process `if` top down
  - For nested loops, process inside out

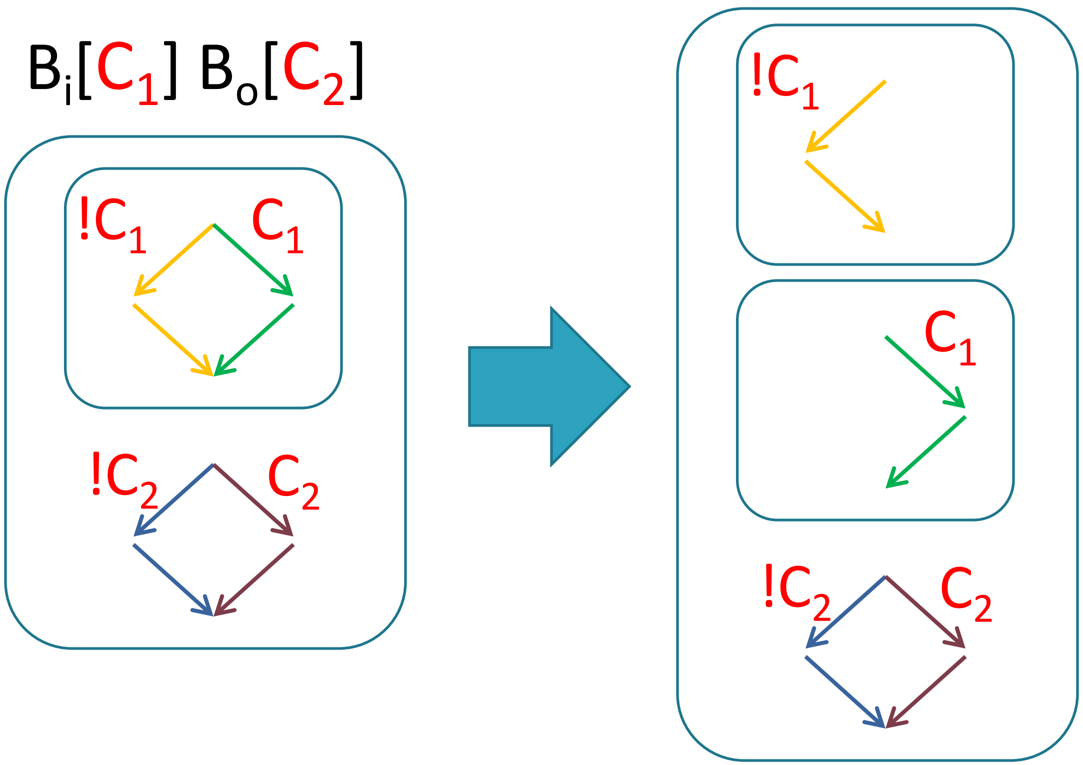
$B_i[C_1]\ B_o[C_2]$



Optimizations

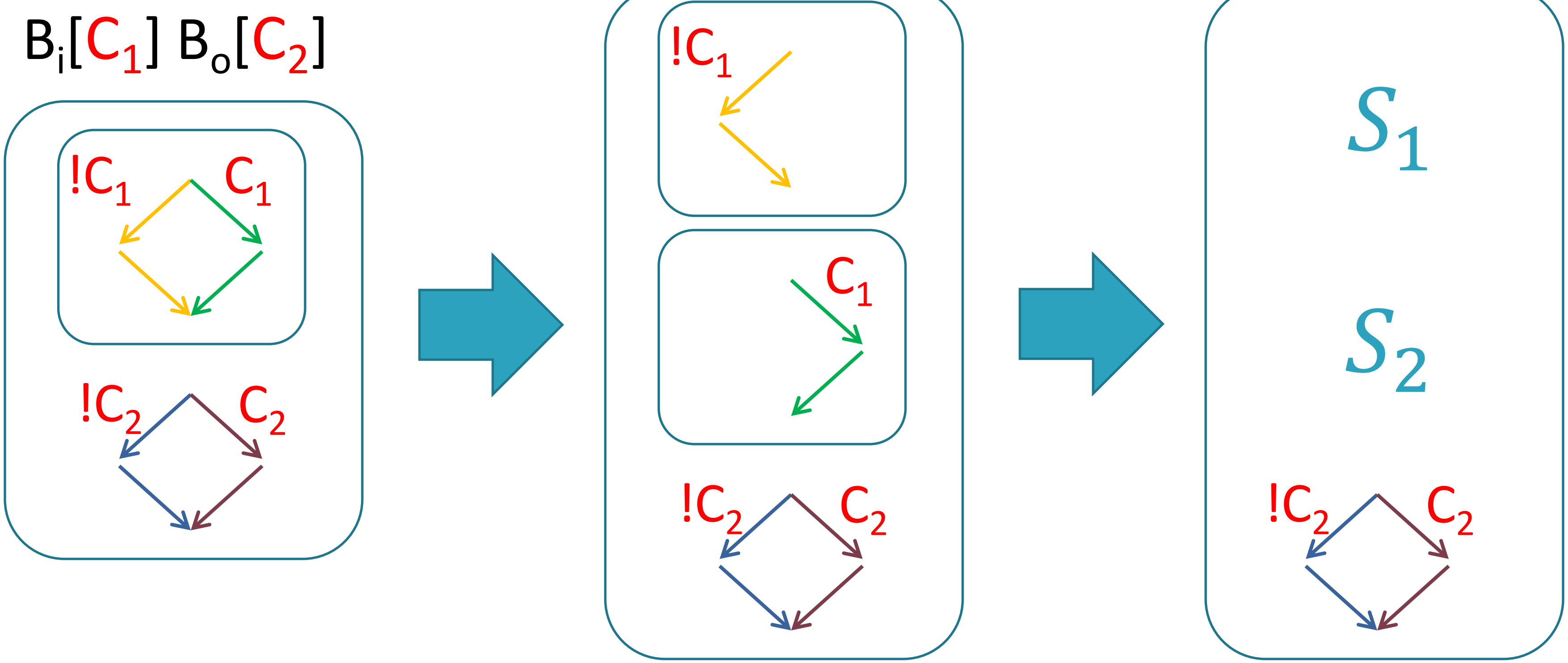
- Simplify precondition computation
  - Process `if` top down
  - For nested loops, process inside out

$B_i[C_1]\ B_o[C_2]$



Optimizations

- Simplify precondition computation
  - Process `if` top down
  - For nested loops, process inside out



Experimental Framework

- Interproc
  - Abstract interpretation-based
- InvGen
  - Constraint solving-based
- Our implementation in C++
  - MISTRAL for precondition and constraint solving

Experiments

File	Interproc			InvGen		
	Before split	After split	Quality	Before split	After split	Quality
popl07	N	Y	$+$	N	Y	$+$
cav06	N	Y	$+$	N	Y	$+$
tacas08	N	Y	$+$	Y	Y	$=$
svd	Y	Y	$+$	Y	Y	$+$
heapsort	Y	Y	$+$	Y	Y	$+$
mergesort	N	N	$\|\|$	Y	Y	$+$
spam	Y	Y	$+$	Y	Y	$+$
ex1	N	Y	$+$	N	Y	$+$
ex2	N	Y	$+$	N	Y	$+$

Y : verification succeeded; N : verification failed

$+$: better invariants; $||$: incomparable; $=$: exactly same

Experiments

File	Interproc			InvGen		
	Before split	After split	Quality	Before split	After split	Quality
popl07	N	Y	$+$	N	Y	$+$
cav06	N	Y	$+$	N	Y	$+$
tacas08	N	Y	$+$	Y	Y	$=$
svd	Y	Y	$+$	Y	Y	$+$
heapsort	Y	Y	$+$	Y	Y	$+$
mergesort	N	N	$\|\|$	Y	Y	$+$
spam	Y	Y	$+$	Y	Y	$+$
ex1	N	Y	$+$	N	Y	$+$
ex2	N	Y	$+$	N	Y	$+$

Y : verification succeeded; N : verification failed

$+$: better invariants; $\|\|$: incomparable; $=$: exactly same

Experiments

File	Interproc			InvGen		
	Before split	After split	Quality	Before split	After split	Quality
popl07	N	Y	$+$	N	Y	$+$
cav06	N	Y	$+$	N	Y	$+$
tacas08	N	Y	$+$	Y	Y	$=$
svd	Y	Y	$+$	Y	Y	$+$
heapsort	Y	Y	$+$	Y	Y	$+$
mergesort	N	N	$\|\|$	Y	Y	$+$
spam	Y	Y	$+$	Y	Y	$+$
ex1	N	Y	$+$	N	Y	$+$
ex2	N	Y	$+$	N	Y	$+$

Y : verification succeeded; N : verification failed

$+$: better invariants; $||$: incomparable; $=$: exactly same

Experiments

File	INTERPROC			INVGEN		
	Before split	After split	Quality	Before split	After split	Quality
popl07	N	Y	$+$	N	Y	$+$
cav06	N	Y	$+$	N	Y	$+$
tacas08	N	Y	$+$	Y	Y	$=$
svd	Y	Y	$+$	Y	Y	$+$
heapsort	Y	Y	$+$	Y	Y	$+$
mergesort	N	N	$\|\|$	Y	Y	$+$
spam	Y	Y	$+$	Y	Y	$+$
ex1	N	Y	$+$	N	Y	$+$
ex2	N	Y	$+$	N	Y	$+$

Y : verification succeeded; N : verification failed

$+$: better invariants; $||$: incomparable; $=$: exactly same

Experiments

File	Interproc			InvGen		
	Before split	After split	Quality	Before split	After split	Quality
popl07	N	Y	$+$	N	Y	$+$
cav06	N	Y	$+$	N	Y	$+$
tacas08	N	Y	$+$	Y	Y	$=$
svd	Y	Y	$+$	Y	Y	$+$
heapsort	Y	Y	$+$	Y	Y	$+$
mergesort	N	N	$\|\|$	Y	Y	$+$
spam	Y	Y	$+$	Y	Y	$+$
ex1	N	Y	$+$	N	Y	$+$
ex2	N	Y	$+$	N	Y	$+$

Y : verification succeeded; N : verification failed

$+$: better invariants; $||$: incomparable; $=$: exactly same

Experiments

File	Interproc			InvGen		
	Before split	After split	Quality	Before split	After split	Quality
popl07	N	Y	$+$	N	Y	$+$
cav06	N	Y	$+$	N	Y	$+$
tacas08	N	Y	$+$	Y	Y	$=$
svd	Y	Y	$+$	Y	Y	$+$
heapsort	Y	Y	$+$	Y	Y	$+$
mergesort	N	N	$\|\|$	Y	Y	$+$
spam	Y	Y	$+$	Y	Y	$+$
ex1	N	Y	$+$	N	Y	$+$
ex2	N	Y	$+$	N	Y	$+$

Y : verification succeeded; N : verification failed

$+$: better invariants; $||$: incomparable; $=$: exactly same

Experiments

File	INTERPROC			INVGEN		
	Before split	After split	Quality	Before split	After split	Quality
popl07	N	Y	$+$	N	Y	$+$
cav06	N	Y	$+$	N	Y	$+$
tacas08	N	Y	$+$	Y	Y	$=$
svd	Y	Y	$+$	Y	Y	$+$
heapsort	Y	Y	$+$	Y	Y	$+$
mergesort	N	N	$\|\|$	Y	Y	$+$
spam	Y	Y	$+$	Y	Y	$+$
ex1	N	Y	$+$	N	Y	$+$
ex2	N	Y	$+$	N	Y	$+$

Y : verification succeeded; N : verification failed

$+$: better invariants; $||$: incomparable; $=$: exactly same

Experiments

File	Interproc			InvGen		
	Before split	After split	Quality	Before split	After split	Quality
popl07	N	Y	$+$	N	Y	$+$
cav06	N	Y	$+$	N	Y	$+$
tacas08	N	Y	$+$	Y	Y	$=$
svd	Y	Y	$+$	Y	Y	$+$
heapsort	Y	Y	$+$	Y	Y	$+$
mergesort	N	N	$\|\|$	Y	Y	$+$
spam	Y	Y	$+$	Y	Y	$+$
ex1	N	Y	$+$	N	Y	$+$
ex2	N	Y	$+$	N	Y	$+$

Y : verification succeeded; N : verification failed

$+$: better invariants; $\|\|$: incomparable; $=$: exactly same

Experiments

File	Interproc			InvGen		
	Before split	After split	Quality	Before split	After split	Quality
popl07	N	Y	$+$	N	Y	$+$
cav06	N	Y	$+$	N	Y	$+$
tacas08	N	Y	$+$	Y	Y	$=$
svd	Y	Y	$+$	Y	Y	$+$
heapsort	Y	Y	$+$	Y	Y	$+$
mergesort	N	N	$\|\|$	Y	Y	$+$
spam	Y	Y	$+$	Y	Y	$+$
ex1	N	Y	$+$	N	Y	$+$
ex2	N	Y	$+$	N	Y	$+$

Y : verification succeeded; N : verification failed

$+$: better invariants; $\|\|$: incomparable; $=$: exactly same

Experiments

File	INTERPROC			INVGEN		
	Before split	After split	Quality	Before split	After split	Quality
popl07	N	Y	$+$	N	Y	$+$
cav06	N	Y	$+$	N	Y	$+$
tacas08	N	Y	$+$	Y	Y	$=$
svd	Y	Y	$+$	Y	Y	$+$
heapsort	Y	Y	$+$	Y	Y	$+$
mergesort	N	N	$\|\|$	Y	Y	$+$
spam	Y	Y	$+$	Y	Y	$+$
ex1	N	Y	$+$	N	Y	$+$
ex2	N	Y	$+$	N	Y	$+$

Y : verification succeeded; N : verification failed

$+$: better invariants; $||$: incomparable; $=$: exactly same

Experiments

File	INTERPROC			INVGEN		
	Before split	After split	Quality	Before split	After split	Quality
popl07	N	Y	$+$	N	Y	$+$
cav06	N	Y	$+$	N	Y	$+$
tacas08	N	Y	$+$	Y	Y	$=$
svd	Y	Y	$+$	Y	Y	$+$
heapsort	Y	Y	$+$	Y	Y	$+$
mergesort	N	N	$\|\|$	Y	Y	$+$
spam	Y	Y	$+$	Y	Y	$+$
ex1	N	Y	$+$	N	Y	$+$
ex2	N	Y	$+$	N	Y	$+$

Y : verification succeeded; N : verification failed

$+$: better invariants; $||$: incomparable; $=$: exactly same

Conclusion

- Static analysis to identify phase transitions
- Decompose multi-phase loops
  - Preserve semantics
- Benefit standard invariant generation tools
  - Better invariants
  - Handle more loops
- Simple, easy to implement, and can be integrated with any invariant generation tool

References

- Balakrishnan, G., Sankaranarayanan, S., Ivancic, F., Gupta, A.:
  - Refining the control structure of loops using static analysis.
  - In: EMSOFT (2009), pp. 49-58.
- Gulwani, S., Jain, S., Koskinen, E.:
  - Control-flow refinement and progress invariants for bound analysis.
  - In: PLDI (2009), pp. 375-385.
- Mauborgne, L., Rival, X.:
  - Trace partitioning in abstract interpretation based static analyzers.
  - In: ESOP (2005), pp. 5-20.
- Gopan, D., Reps, T.:
  - Guided static analysis.
  - In: SAS (2007), pp. 349-365.

References

- Balakrishnan, G., Sankaranarayanan, S., Ivancic, F., Gupta, A.:
  - Refining the control structure of loops using static analysis.
  - In: EMSOFT (2009), pp. 49-58.
- Gulwani, S., Jain, S., Koskinen, E.:
  - Control-flow refinement and progress invariants for bound analysis.
  - In: PLDI (2009), pp. 375-385.
- Mauborgne, L., Rival, X.:
  - Trace partitioning in abstract interpretation based static analyzers.
  - In: ESOP (2005), pp. 5-20.
- Gopan, D., Reps, T.:
  - Guided static analysis.
  - In: SAS (2007), pp. 349-365.