Block-wise abstract interpretation by combining abstract domains with SMT

Jiahong Jiang, Liqian Chen, Xueguang Wu, Ji Wang

National University of Defense Technology, China

01/16/2017 VMCAI 2017

Overview

- Motivation
- Block-wise Abstract Interpretation (BWAI) Framework
- Practical Concerns for BWAI
- Implementation and Experiments
- Conclusion

Statement-wise Abstract Interpretation (SWAI)

- SWAI
  - each statement as an individual transfer function
- Advantage
  - scalable

Statement-wise Abstract Interpretation (SWAI)

- SWAI
  - each statement as an individual transfer function
- Advantage
  - scalable
- Disadvantage
  - may cause precision loss

```
// x ∈ [-2, 2], y ∈ [-3, 3]
x = y + 1;  // x ∈ [-2, 4], y ∈ [-3, 3]
y = x – y;  // x ∈ [-2, 4], y ∈ [-5, 7]
y = 1 / (y - 2);  // y ∈ [-5, 7]
```

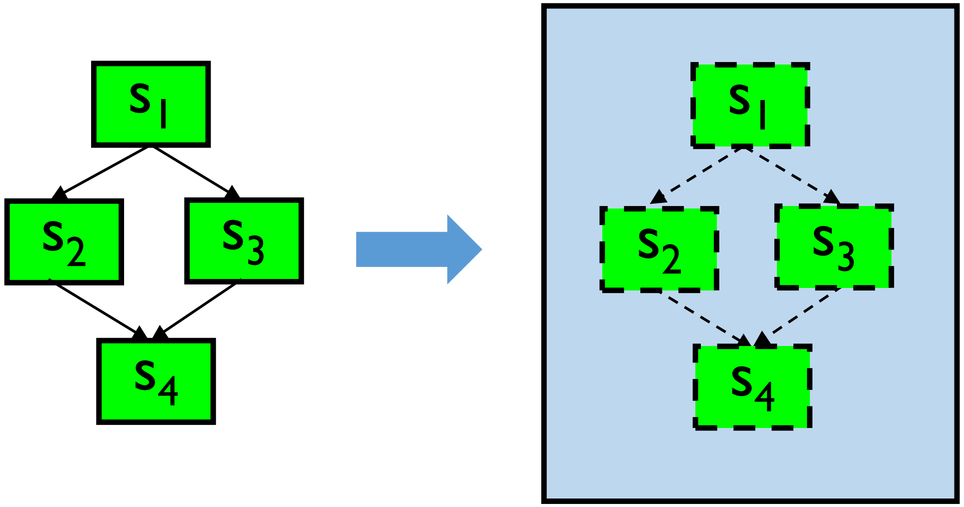
Ex. 1

```
if (brandom())
    y = 1;
else
    y = -1;
x = 1 / y;   // y ∈ [-1, 1]
```

Ex. 2

Main Idea

- Block-wise abstract interpretation (BWAI)
  - partition the program into several blocks
  - analyze the program block by block under AI



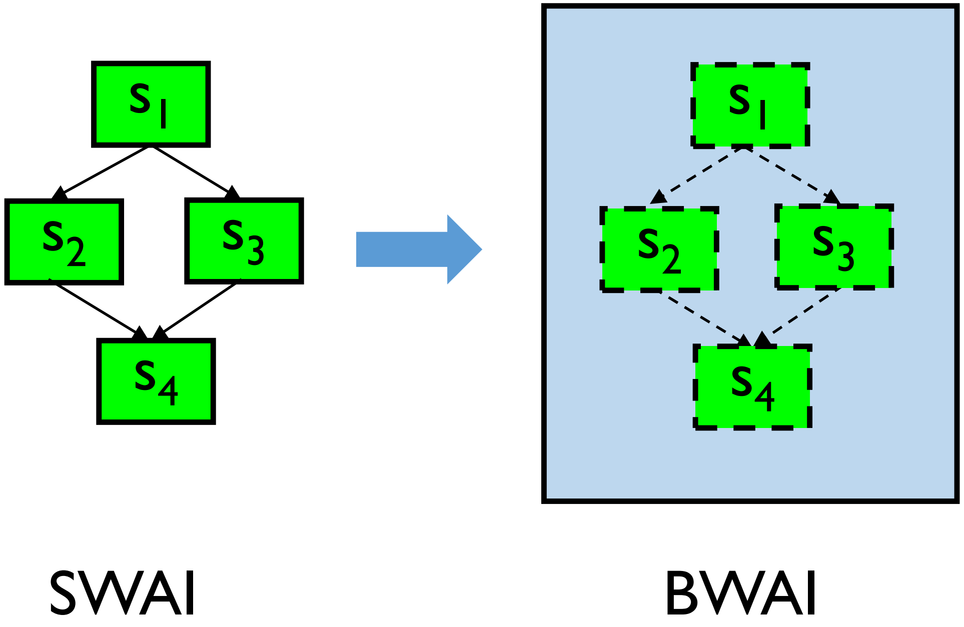
multiple statements
as a block

SWAI

BWAI

Main Idea

- Block-wise abstract interpretation (BWAI)
  - partition the program into several blocks
  - analyze the program block by block under AI



multiple statements as a block

BWAI could see more information than SWAI at one step

Overview

- Motivation
- Block-wise Abstract Interpretation (BWAI) Framework
- Practical Concerns for BWAI
- Implementation and Experiments
- Conclusion

Questions

- How to partition the program into blocks
- How to encode semanics of a block
- How to transmit information between blocks

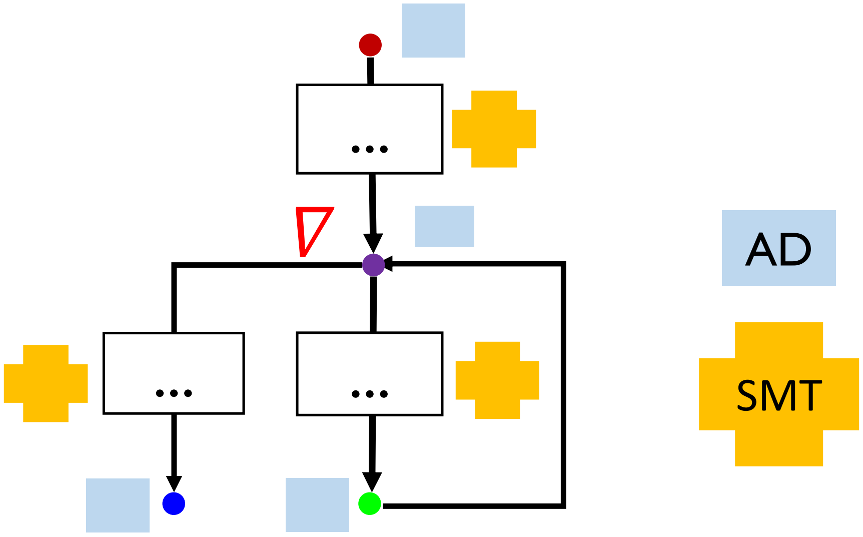
Choices for Expressing Transfer Semantics of a Block

- Abstract domains
  - pros: efficient
  - cons: most domains have limitations in expressing disjunctions
- SMT
  - pros: expressive for disjunctions
    - E.g., (cond == true ∧ x1== 2) ∨ (cond == false ∧ x1 == -2)
  - cons: loops are challenging to cope with when using SMT

Workflow of BWAI

- BWAI by combining abstract domains (AD) with SMT
  - partition the program into several blocks
  - encode transfer semantics of a block via SMT
  - use abstract domains between blocks
  - use widening of abstract domains at loop heads

```
● …
  …
● while(brandom()){
      …
● }
  …
● …
```



Block Partitioning

- Partitioning based on cutpoints[Beyer et al., FMCAD'09]
  - a set of cutpoints : a subset of program points
    - entry/exit points, loop heads, ...
  - two extreme partitioning strategies
    - minimize the size of a block
      - each statement as a block (SWAI)
    - maximize the size of a block
      - only at necessary points (loop heads, etc.)

Block Encoding

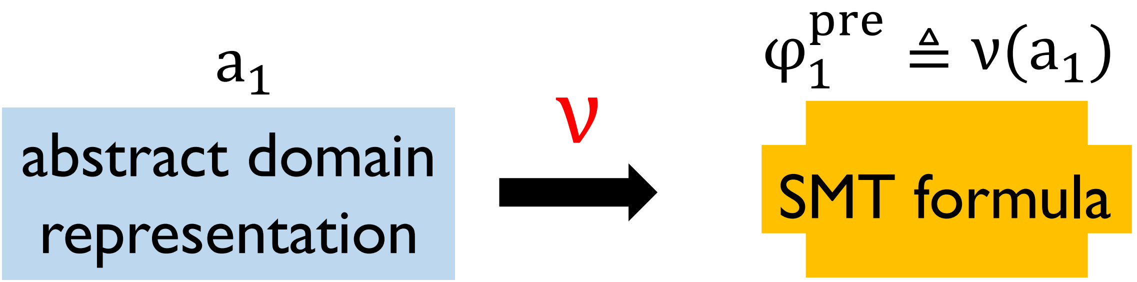
- Encoding of the transfer semantics of a block
  - via SMT formula in $\mathcal{T}$-theroy (e.g, Linear Real Arithmetic)

```
  while(brandom()){
●     if(phase == 1){
          x = x – 1;
           y = y +2;
      }else{
          x = x + 2;
          y = y – 1;
      }
       phase = 1 – phase;
● }
```

$$\varphi_2^{\mathrm{trans}} \triangleq \textcolor{red}{\mathrm{ite}}(phase0 == 1,\ (x1 = x0 - 1) \wedge (y1 = y0 + 2),\ (x1 = x0 + 2) \wedge (y1 = y0 - 1)) \wedge (phase1 = 1 - phase0)$$

Representation Conversion

- Conversion between abstract domain representation and SMT



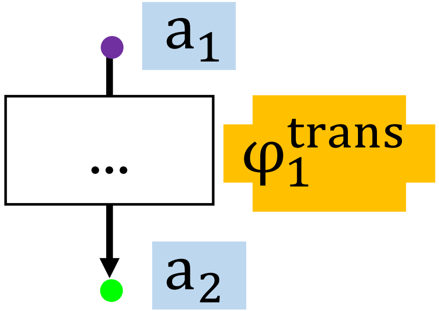
a_1

abstract domain representation

ν

$\varphi_1^{\mathrm{pre}} \triangleq \nu(a_1)$

SMT formula

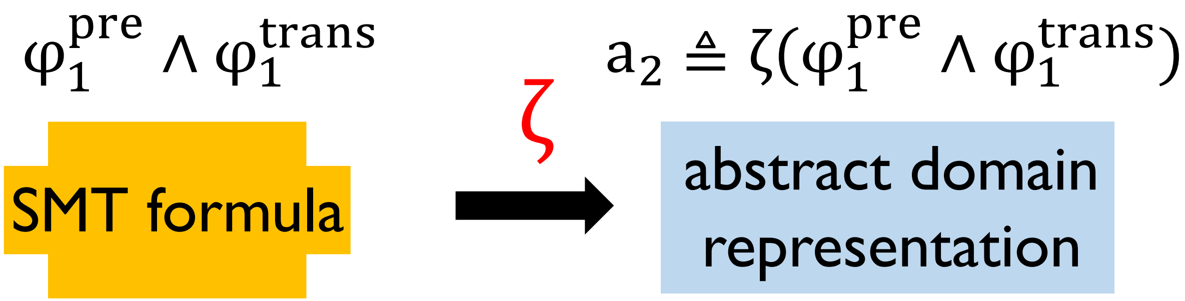


a_1

…

$\varphi_1^{\mathrm{trans}}$

a_2



$\varphi_1^{\mathrm{pre}} \wedge \varphi_1^{\mathrm{trans}}$

SMT formula

ζ

$a_2 \triangleq \zeta(\varphi_1^{\mathrm{pre}} \wedge \varphi_1^{\mathrm{trans}})$

abstract domain representation

Symbolic Abstraction : SMT to Abstract Domain Representation

- Symbolic abstraction [Thakur et al., SAS'12]
  - the consequence "a" of an SMT formula φ in the abstract domain
  - sound symbolic abstraction "a"
    - Sol(φ) $\subseteq$ Sol (a)

Symbolic Abstraction : SMT to Abstract Domain Representation

- Using optimization techniques based on SMT (SMT-opt)

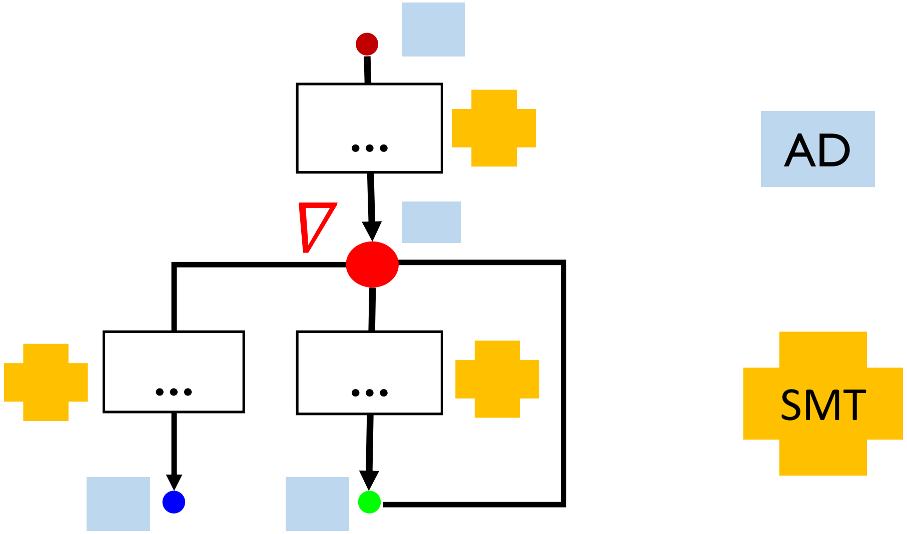
[Li et al., POPL'14]

  - SMT-opt problem: "**max** e **s.t.** φ"
  - fit for abstract domains based on templates
    - e.g., boxes, octagons, TCMs

"**max**(x + y) **s.t.** (2x+y >10 ∨ 3x-2y < -5)" for Octagon domain

Block-wise Iteration Strategy

- “iteration + widening” on abstract domains
  - iterating on CFG with blocks
  - use widening at loop heads



Overview

- Motivation
- Block-wise Abstract Interpretation (BWAI) Framework
- Practical Concerns for BWAI
  - precision
  - efficiency
- Implementation and Experiments
- Conclusion

Precision Loss Problem in BWAI

- SMT is often more expressive than abstract domain

```
●  phase = [0, 1];
   x = y = 0;
●  while(brandom()){
       if(phase == 1){
           x = x – 1;
           y = y +2;
       }else{
           x = x +2;
           y = y – 1;
       }
        phase = 1 – phase;
●  }
   if(x – y > 3) { /* error() */ };
●  …
```

$\boldsymbol{\varphi}_2^{\mathbf{pre}} \wedge \boldsymbol{\varphi}_2^{\mathbf{trans}} \triangleq$

$(0 \le phase0 \le 1) \wedge (x0 == 1) \wedge (y0 == 1)$

$\wedge(ite(phase0 == 1),$

$(x1 = x0 - 1) \wedge (y1 = y0 + 2),$

$(x1 = x0 + 2) \wedge (y1 = y0 - 1))$

$\wedge(phase1 = 1 - phase0)$

↓ SMT-opt for Octagon

$((-3 \le x - y \le 3) \wedge (0 \le phase \le 1)$
$\wedge(-1 \le x \le 2) \wedge (-1 \le y \le 2) \wedge \ldots)$

↓

…

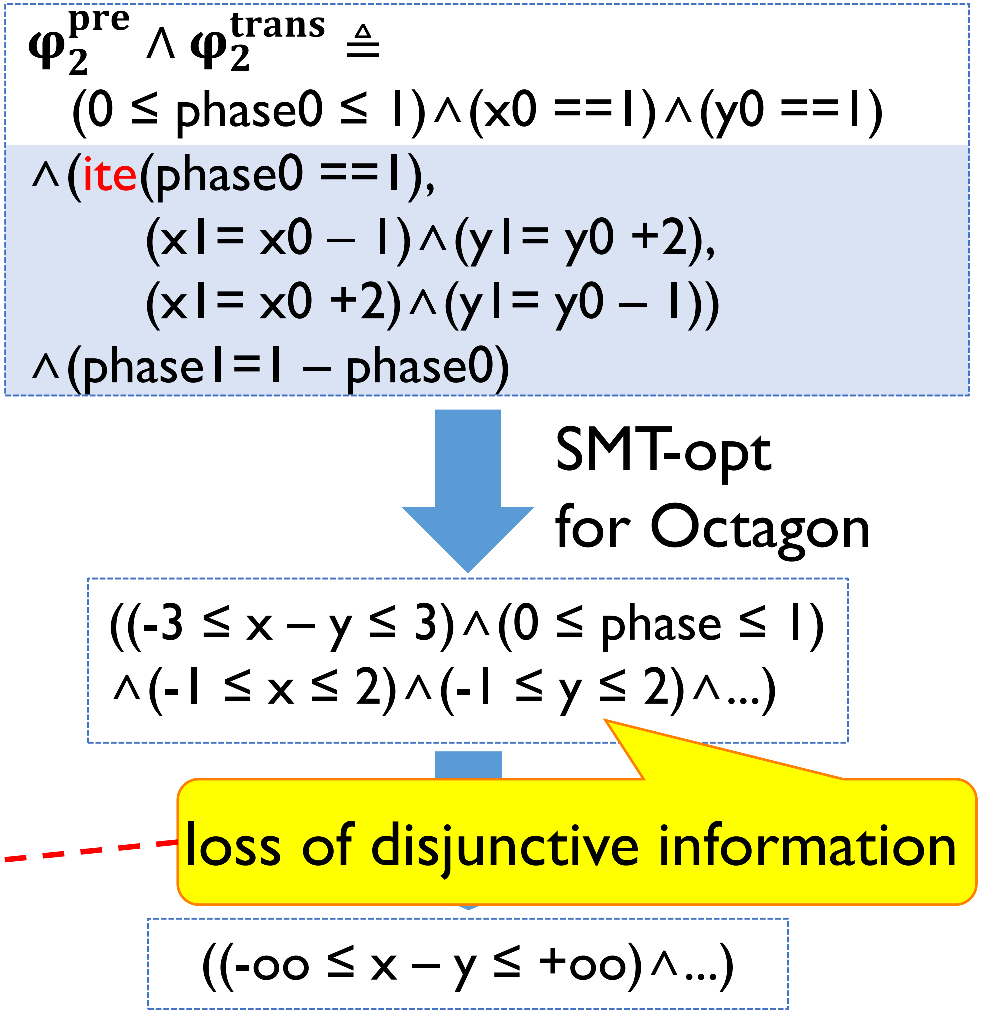
↓

$((-oo \le x - y \le +oo) \wedge \ldots)$

Precision Loss Problem in BWAI

- SMT is often more expressive than abstract domain

```
●  phase = [0, 1];
   x = y = 0;
●  while(brandom()){
       if(phase == 1){
           x = x – 1;
           y = y +2;
       }else{
           x = x +2;
           y = y – 1;
       }
        phase = 1 – phase;
●  }
   if(x – y > 3) { /* error() */ };
●  ...
```



$\varphi_2^{\mathrm{pre}} \wedge \varphi_2^{\mathrm{trans}} \triangleq$

$(0 \le phase0 \le 1) \wedge (x0 == 1) \wedge (y0 == 1)$

$\wedge(ite(phase0 == 1),$

$\quad (x1 = x0 - 1) \wedge (y1 = y0 + 2),$

$\quad (x1 = x0 + 2) \wedge (y1 = y0 - 1))$

$\wedge(phase1 = 1 - phase0)$

SMT-opt for Octagon

$((-3 \le x - y \le 3) \wedge (0 \le phase \le 1)$

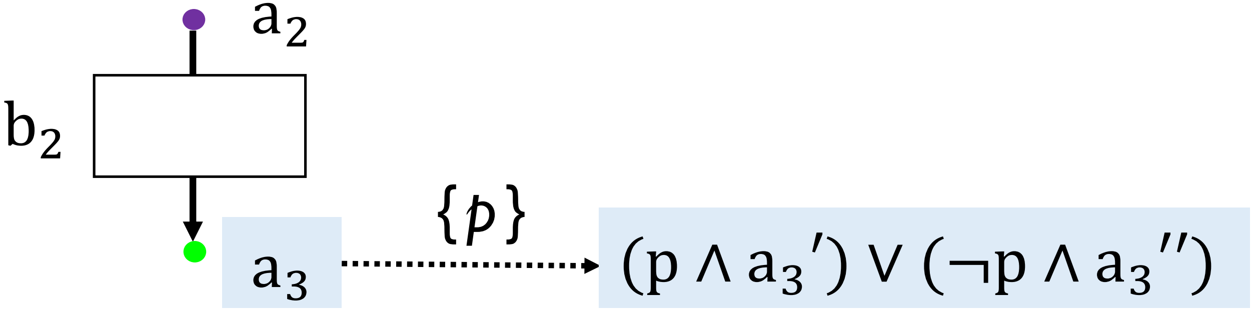
$\wedge(-1 \le x \le 2) \wedge (-1 \le y \le 2) \wedge \ldots)$

loss of disjunctive information

$((-oo \le x - y \le +oo) \wedge \ldots)$

Our Solution

- Abstract domain lifting functor for BWAI
  - **goal**: pass necessary disjunctive information between blocks
  - **idea**:
    - choose a set of predicates for each block
      - branch conditions in direct syntactic successor blocks
    - partition the post-state according to predicate values

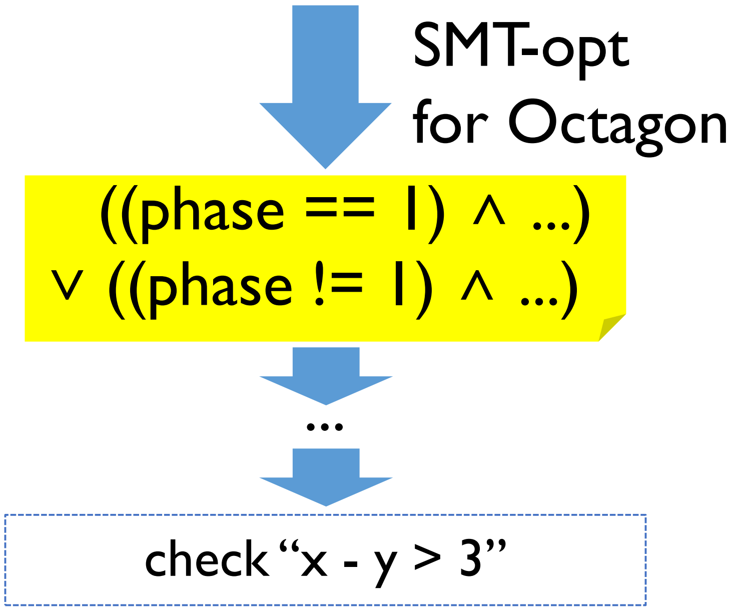


Our Solution

- SMT is often more expressive than abstract domain

```
● phase = [0, 1];
  x = y = 0;
● while(brandom()){
      if(phase == 1){
          x = x – 1;
          y = y +2;
      }else{
          x = x +2;
          y = y – 1;
      }
       phase = 1 – phase;
● }
  if(x – y > 3) { /* error() */ };
● ...
```

$$\varphi_2^{\text{pre}} \wedge \varphi_2^{\text{trans}} \triangleq$$
$$(0 \leq phase0 \leq 1) \wedge (x0 == 1) \wedge (y0 == 1)$$
$$\wedge(\text{ite}(phase0 == 1),$$
$$(x1 = x0 - 1) \wedge (y1 = y0 + 2),$$
$$(x1 = x0 + 2) \wedge (y1 = y0 - 1))$$
$$\wedge(phase1 = 1 - phase0)$$



SMT-opt for Octagon

((phase == 1) ∧ ...)
∨ ((phase != 1) ∧ ...)

...

check "x - y > 3"

Overview

- Motivation
- Block-wise Abstract Interpretation (BWAI) framework
- Practical Concerns for BWAI
    - precision
    - efficiency
- Implementation and Experiments
- Conclusion

Scalability Problem due to Large Blocks

- Big-size formula for a large block
- Large predicate set
  - when many braches in a large block

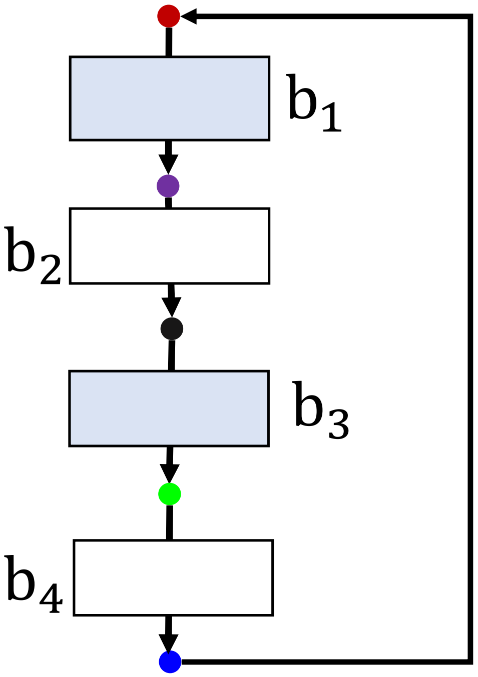
```
while(brandom()){
    if(p1 != 0)
        lk1 = 1;
    if(p2 != 0)
        lk2 = 1;
    if(p1 != 0 && lk1 != 0)
        // ...
    if(p2 != 0 && lk2 != 0)
        // ...
}
```

at least 4 predicates for this large block

Our Solution

- Dividing a large block into small blocks
  - exploiting variable clustering based on data dependency

```
  while(brandom()){
●     if(p1 != 0)
          lk1 = 1;
●     if(p2 != 0)
          lk2 = 1;
●     if(p1 != 0 && lk1 != 0)
          // ...
●     if(p2 != 0 && lk2 != 0)
          // ...
● }
```



variable clusters :
{p1, lk1} for b1 and b3
{p2, lk2} for b2 and b4

Our Solution

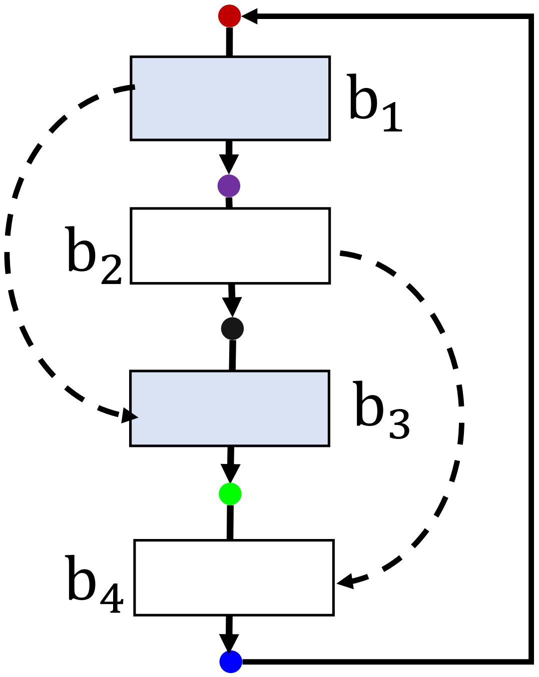
- Considering direct semantic successive blocks
  - the closest successive blocks that share the same variable cluster with the current block

- Benefits of using direct semantic successive blocks
  - more effective information transfer
  - more useful predicates

Our Solution

- BWAI by considering direct semantic successive blocks

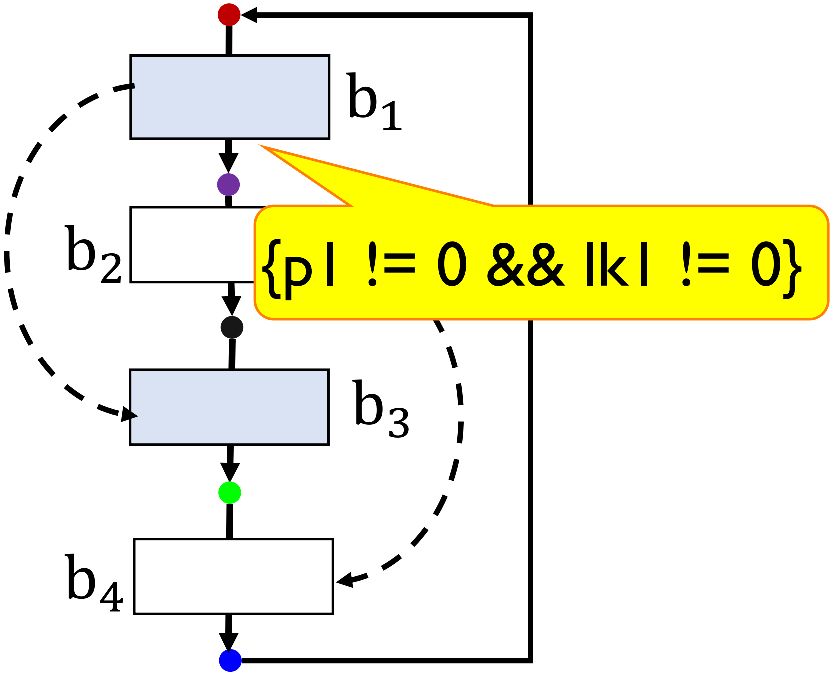
```
     while(brandom()){
●        if(p1 != 0)
             lk1 = 1;
●        if(p2 != 0)
             lk2 = 1;
●        if(p1 != 0 && lk1 != 0)
             // ...
●        if(p2 != 0 && lk2 != 0)
             // ...
●    }
```



Our Solution

- BWAI by considering direct semantic successive blocks

```
    while(brandom()){
●       if(p1 != 0)
            lk1 = 1;
●       if(p2 != 0)
            lk2 = 1;
●       if(p1 != 0 && lk1 != 0)
            // ...
●       if(p2 != 0 && lk2 != 0)
            // ...
●   }
```

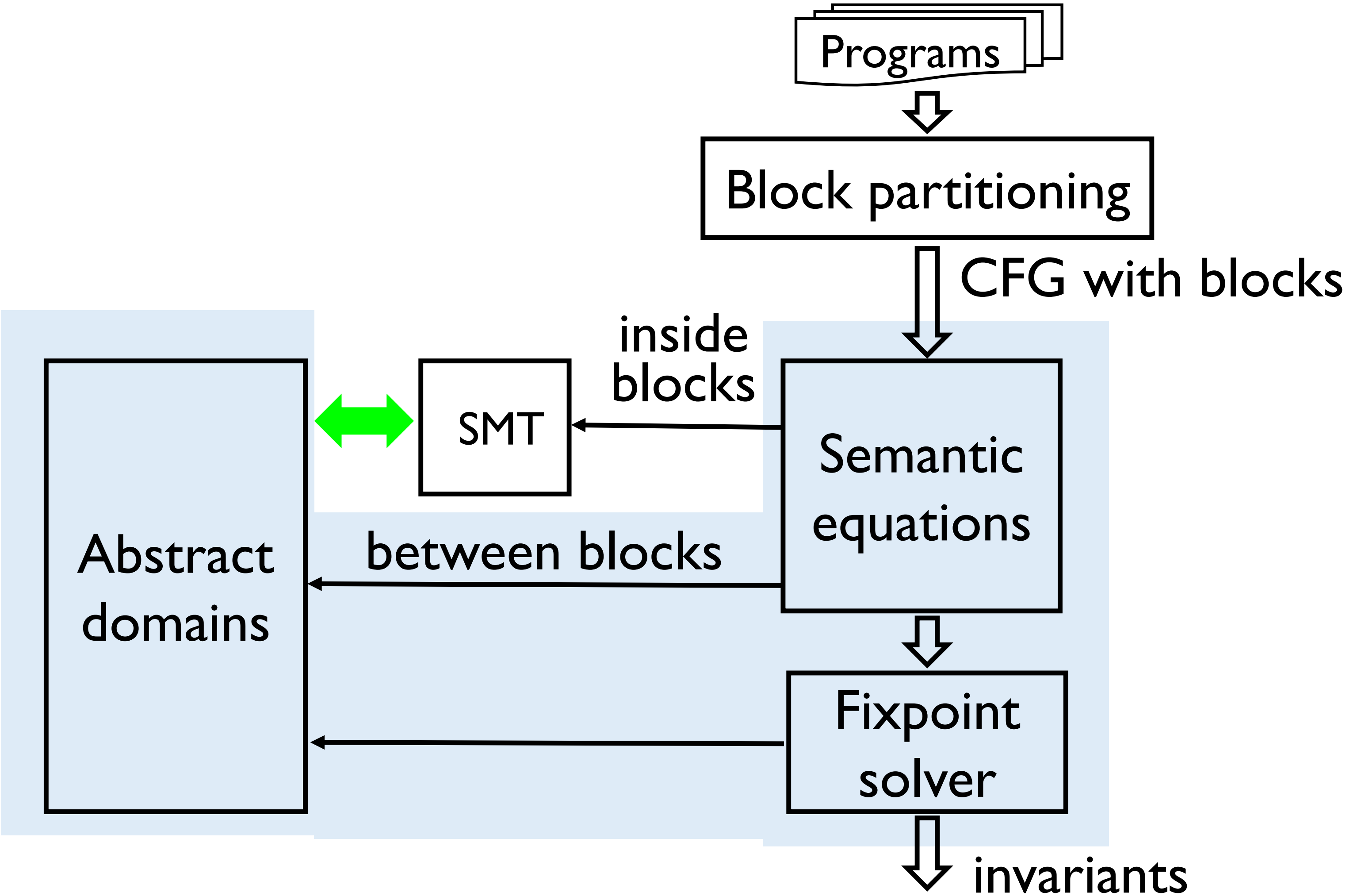


Overview

- Motivation
- Block-wise Abstract Interpretation (BWAI) Framework
- Practical Concerns for BWAI
- Implementation and Experiments
- Conclusion

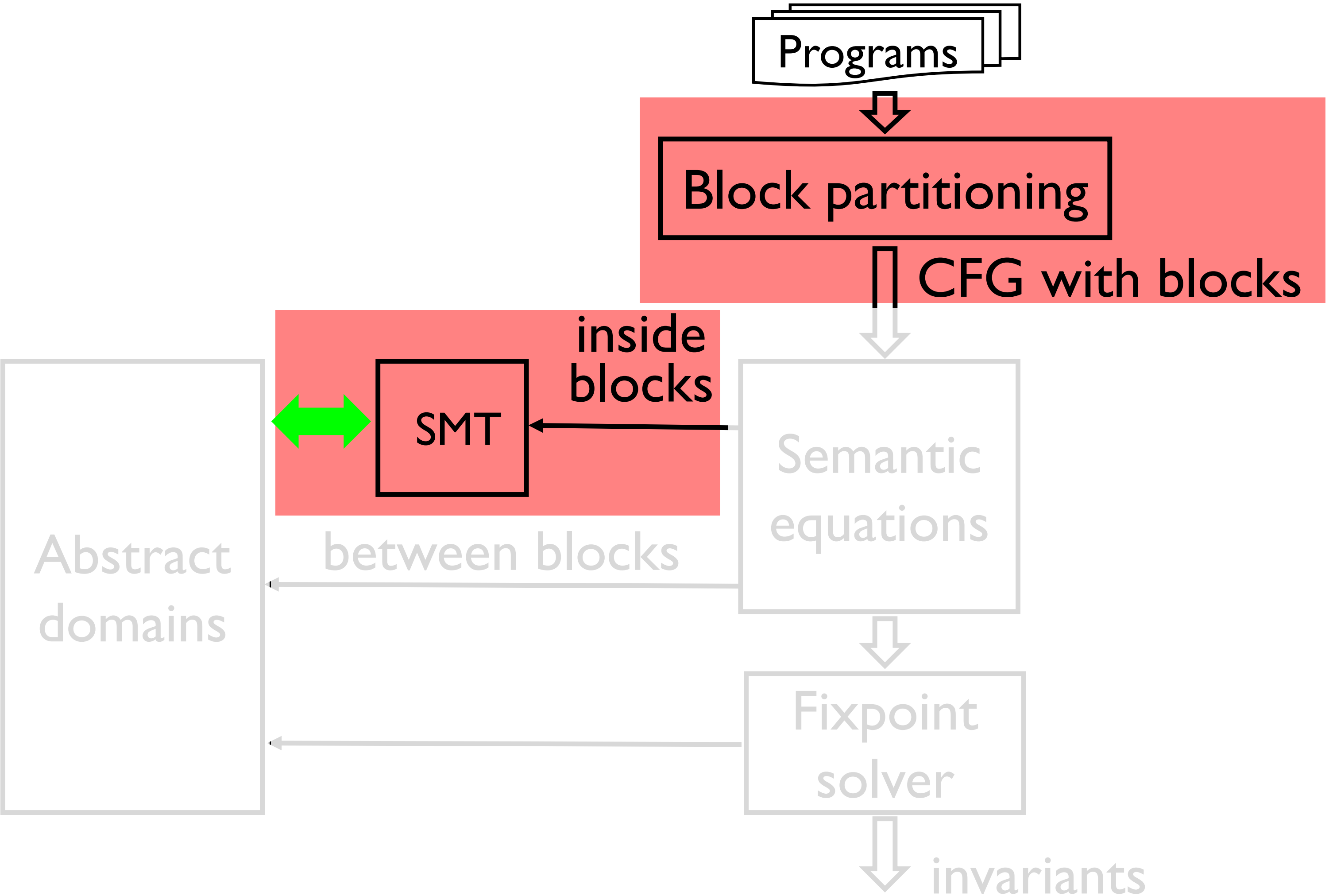
Implementation

- BWCAI: a prototype under BWAI framework



Implementation

- BWCAI: a prototype under BWAI framework



Experiments

- BWAI vs. SWAI

SV-COMP Directories (Numbers of files)	SWAI				BWAI			
	Box		Oct		Box		Oct	
	#Y	t(s)	#Y	t(s)	#Y	t(s)	#Y	t(s)
locks(11)	0	0.28	0	6.40	11	9.13	11	435.14
loop-lit(14)	1	0.09	2	0.12	3	0.95	7	6.77
systemc(20)	0	24.77	0	89.74	1	846.35	5	4733.16
termination-crafted(16)	13	0.08	13	0.09	14	0.35	16	5.22
termination-crafted-lit(12)	10	0.08	10	0.09	10	0.44	10	2.13
termination-restricted-15(12)	6	0.09	8	0.09	10	3.05	16	16.75

BWAI could check around 66% properties (65 out of 98 ones), around one times more than SWAI (33 out of 98 ones)

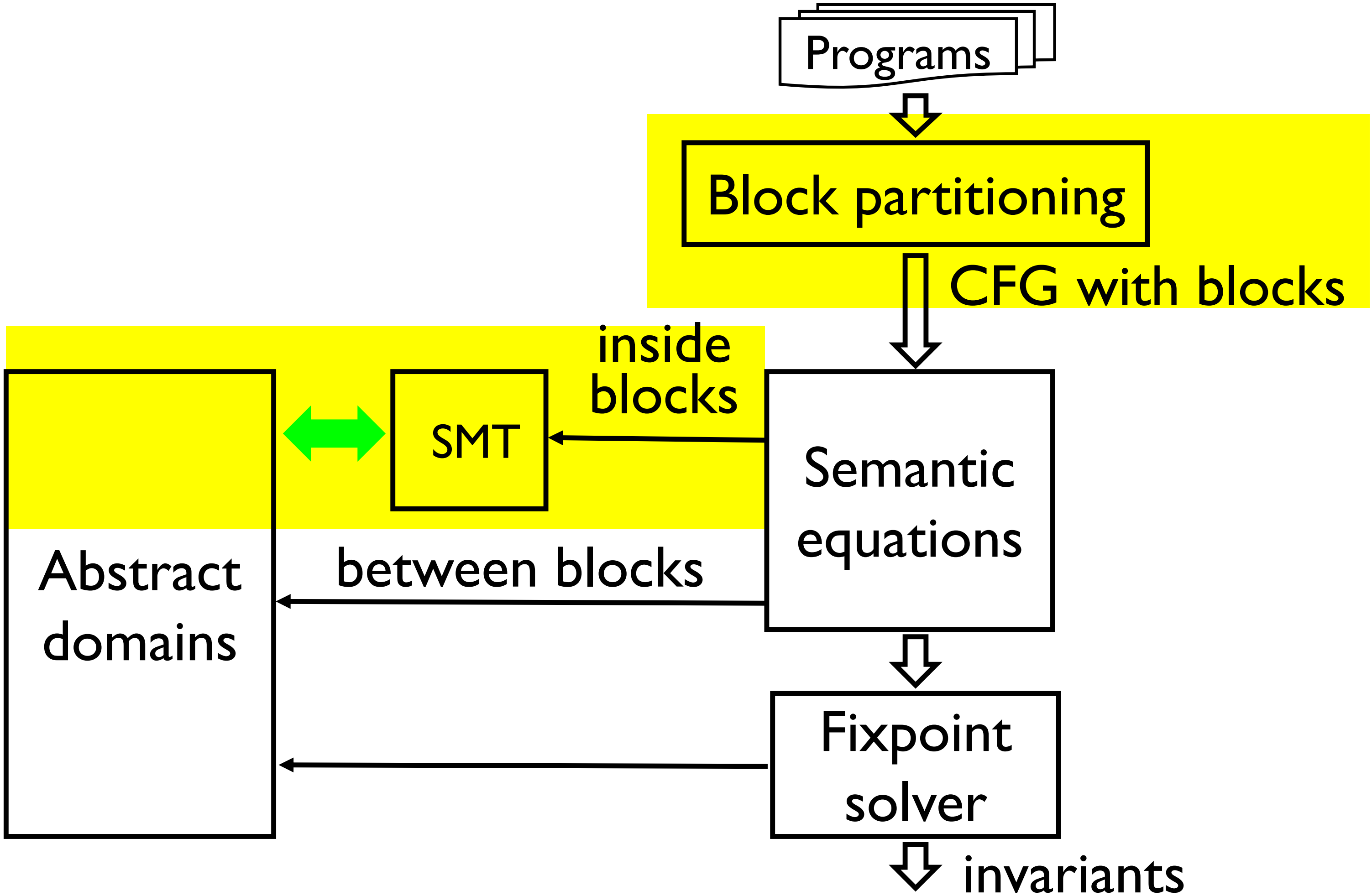
SV-COMP Directories (Numbers of files)	SWAI				BWAI			
	Box		Oct		Box		Oct	
	#Y	t(s)	#Y	t(s)	#Y	t(s)	#Y	t(s)
locks(11)	0	0.28	0	6.40	11	9.13	11	435.14
loop-lit(14)	1	0.09	2	0.12	3	0.95	7	6.77
systemc(20)	0	24.77	0	89.74	1	846.35	5	4733.16
termination-crafted(16)	13	0.08	13	0.09	14	0.35	16	5.22
termination-crafted-lit(12)	10	0.08	10	0.09	10	0.44	10	2.13
termination-restricted-15(12)	6	0.09	8	0.09	10	3.05	16	16.75

Overview

- Motivation
- Block-wise Abstract Interpretation (BWAI) Framework
- Practical Concerns under BWAI
- Implementation and Experiments
- Conclusion

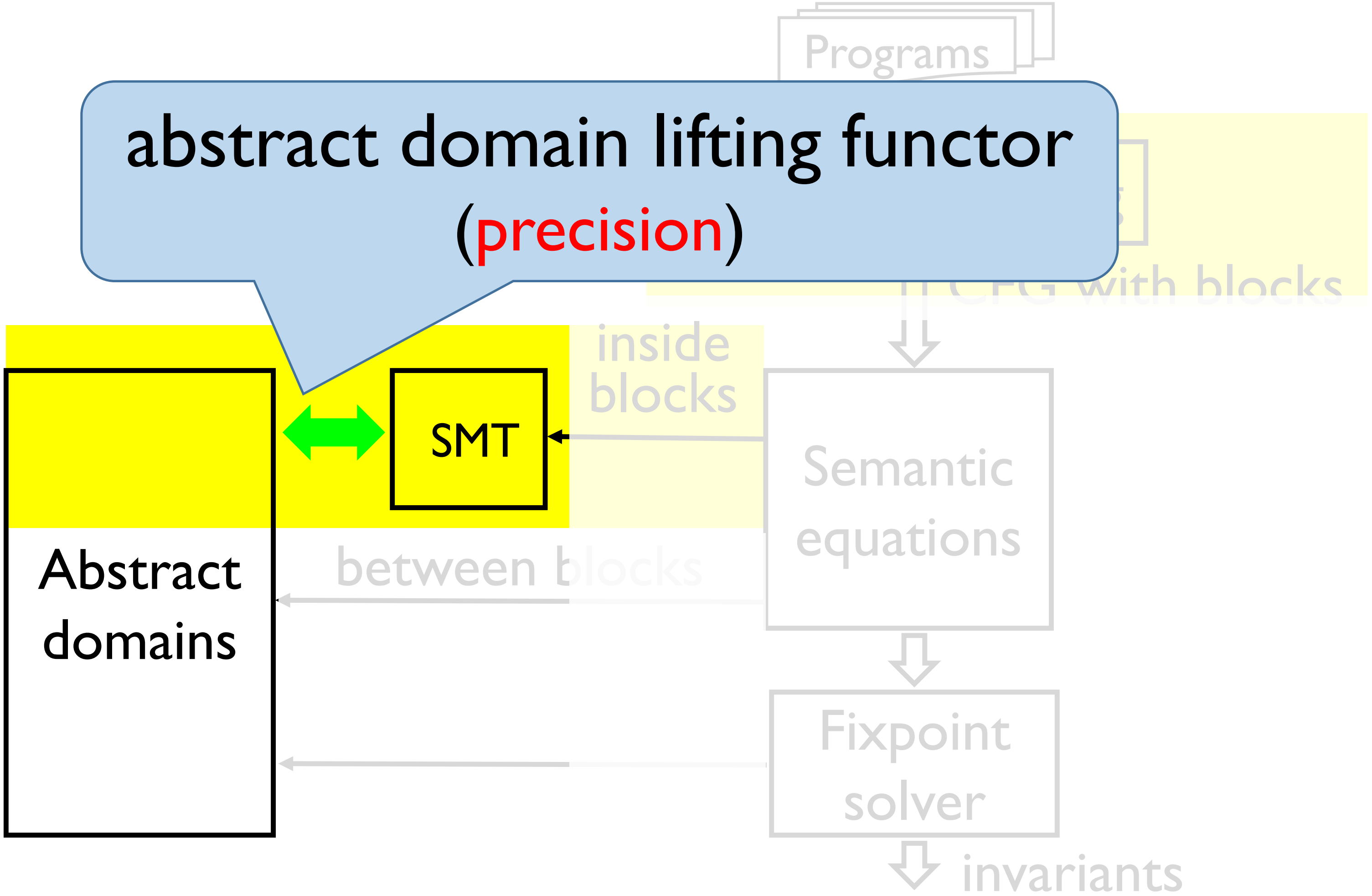
Conclusion

- Block-wise AI instead of statement-wise AI
    - by combining abstract domains with SMT



Conclusion

- A block-wise AI instead of statement-wise AI
  - by combining abstract domains with SMT

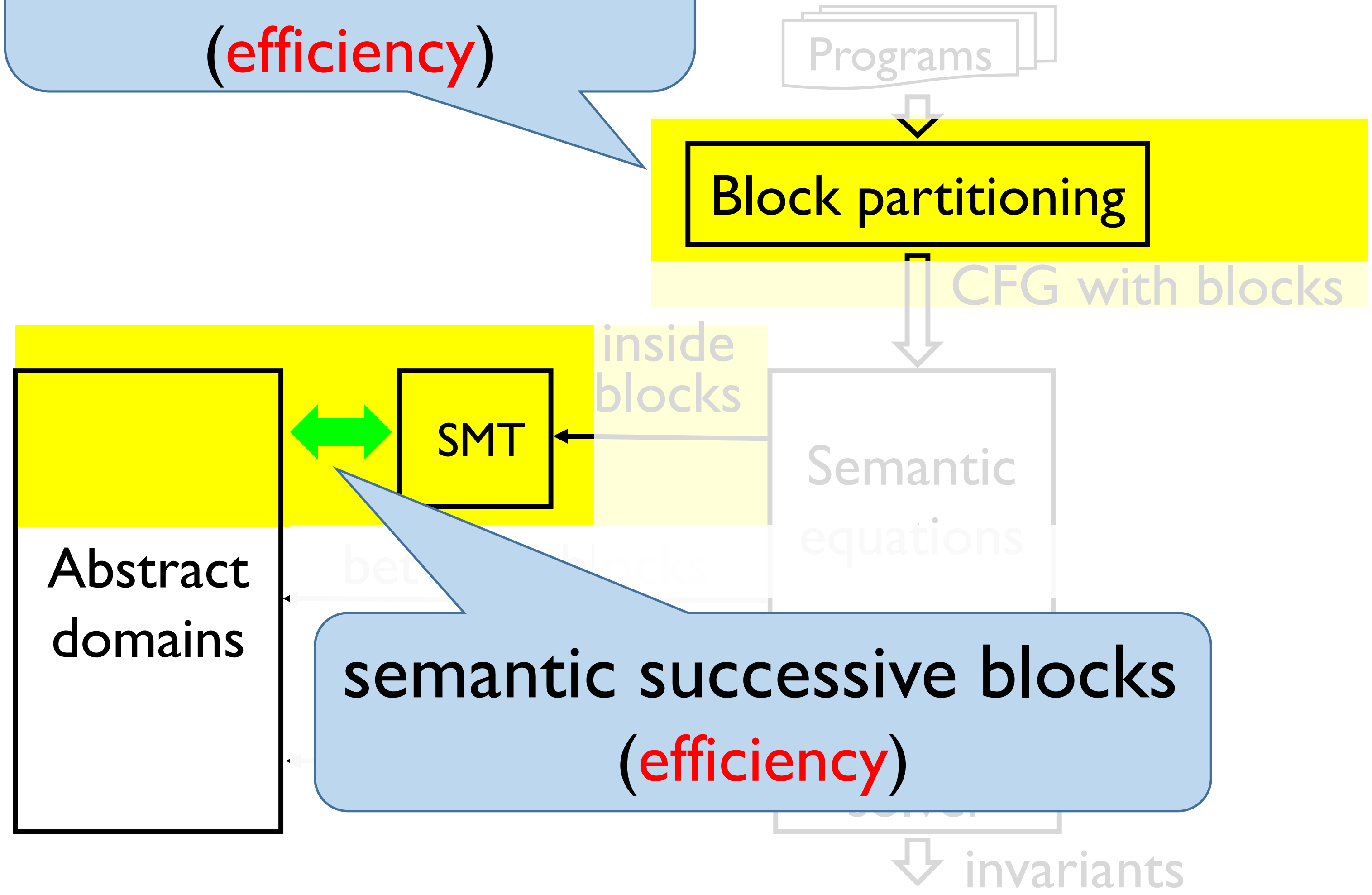


abstract domain lifting functor (precision)

Conclusion

- [illegible]d of statement-wise AI

  [illegible] domains with SMT



divide a large block into small blocks (efficiency)

semantic successive blocks (efficiency)

Future Work

- More flexible block partitioning strategies
  - trade off between precision and efficiency

- Support more SMT theories
  - e.g., floating point, array, ...