Abstract Program Slicing: From theory towards an implementation

Isabella Mastroeni   Đurica Nikolić

Dipartimento di Informatica, University of Verona, Italy

November 2010

Agenda

BASIC NOTIONS

- PROGRAM SLICING: A PROGRAM DECOMPOSITION TECHNIQUE THAT EXTRACTS FROM PROGRAMS STATEMENTS WHICH AFFECT PARAMETERS OF INTEREST
- SLICING CRITERION: CONTAINS DIFFERENT PARAMETERS OF INTEREST (E.G., $\mathcal{C} = (V, n)$ [WEISER '79])
- PROGRAM SLICE: AN EXECUTABLE PROGRAM OBTAINED THAT WAY

EXAMPLES OF SLICES

```
1  begin
2    read(x,y);
3    total := 0.0;
4    sum := 0.0;
5    if x <= 1
6    then sum := y;
7    else begin
8              read(z);
9              total := x*y;
10           end;
11   write(total, sum);
12 end.
```

(12, z):

```
begin
  read(x, y);
  if x <= 1
  then
  else
    read(z);
end.
```

(9, x):

```
begin
  read(x, y);
end.
```

(12, total):

```
begin
  read(x, y);
   total := 0.0;
   if x <= 1
  then
  else
   total := x*y;
end.
```

⇒ EVERYTHING DEPENDS ON SLICING CRITERION

ABSTRACT PROGRAM SLICING - IDEA

- SOMETIMES STANDARD CRITERIA ARE TOO STRONG
- SUPPOSE WE WANT A VARIABLE x TO HAVE A PROPERTY ρ AT SOME POINT n
- THE EXACT VALUE OF x CAN BE EXPRESSED AS $\rho = \lambda a.a$
- WE ARE INTERESTED IN THE STATEMENTS THAT AFFECT $\rho(x)$ AT n
- ABSTRACT SLICES SHOULD BE SMALLER

ABSTRACT PROGRAM SLICING - IDEA

$$a := 1;$$
$$b := b + 1;$$
$$c := c + 2;$$
$$d := c + b + a - a + c;$$

ABSTRACT PROGRAM SLICING - IDEA

$$a := 1;$$
$$b := b + 1;$$
$$c := c + 2;$$
$$d := c + b + a - a + c;$$

ABSTRACT CRITERION:
PARITY OF d

ABSTRACT PROGRAM SLICING - IDEA

$a := 1;$

$b := b + 1;$

$c := c + 2;$

$d := c + b + a - a + c;$

ABSTRACT CRITERION:
PARITY OF d

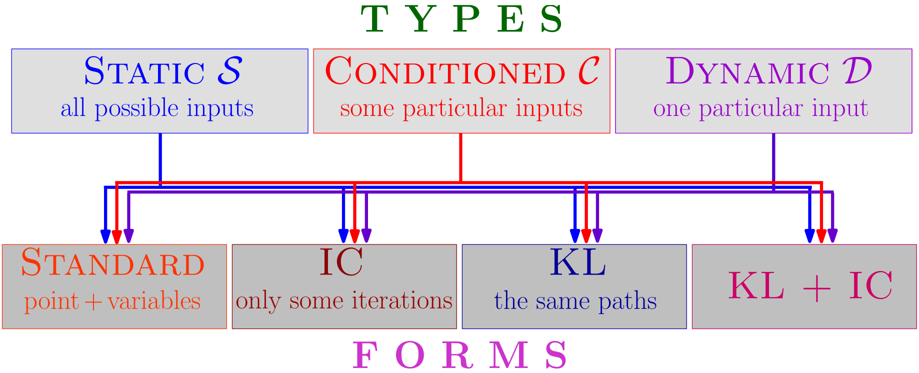
ABSTRACT PROGRAM SLICING - IDEA

$a := 1;$
$b := b + 1;$
$c := c + 2;$
$d := c + b + a - a + c;$

$b := b + 1;$
$d := c + b + a - a + c;$

ABSTRACT CRITERION:
PARITY OF d

Formal Framework [Binkley at al. '06]



Formal Framework [Binkley at al. ’06]

$$\langle \sqsubseteq, \mathcal{E} \rangle$$

Formal Framework [Binkley at al. '06]

SEMANTIC CONSTRAINTS

maps slicing criteria to semantic equivalence relations

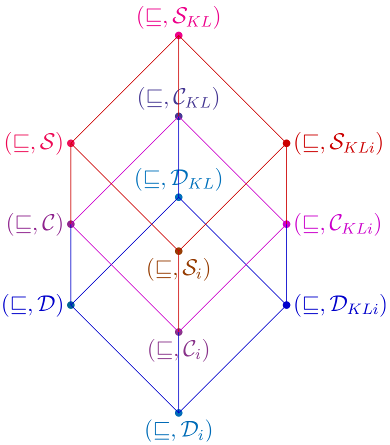
$\langle \sqsubseteq, \mathcal{E} \rangle$

TRADITIONAL SYNTACTIC ORDERING

$$Q \sqsubseteq P \Leftrightarrow \mathcal{F}(Q) \subseteq \mathcal{F}(P)$$

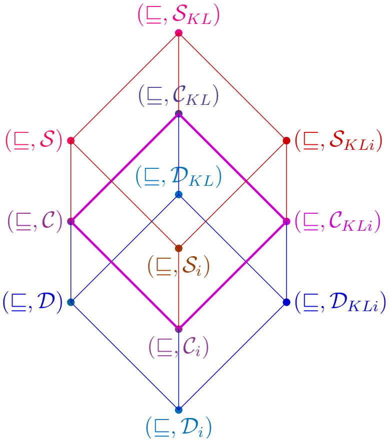
$\mathcal{F}(P)$ maps l to c iff P contains statement c at line l

Formal Framework [Binkley at al. '06]



HIERARCHY OF EXISTING FORMS OF SLICING

Formal Framework [Binkley at al. '06]



HIERARCHY OF EXISTING FORMS OF SLICING

C
R
I
T
E
R
I
O
N

VARIABLES OF INTEREST: V

STATIC	CONDITIONED	DYNAMIC
all possible inputs	some particular inputs	one particular input

C
R
I
T
E
R
I
O
N

VARIABLES OF INTEREST: V

Static	**Conditioned**	**Dynamic**
all possible inputs	some particular inputs	one particular input
$\mathcal{I} = \mathbb{M}$	$\varnothing \subset \mathcal{I} \subseteq \mathbb{M}$	$\lvert\mathcal{I}\rvert = 1$

CRITERION

- **Inputs of Interest:** $\mathcal{I}$
- **Variables of Interest:** V

C R I T E R I O N

INPUTS OF INTEREST: $\mathcal{I}$

VARIABLES OF INTEREST: V

ITERATION COUNT only particular iterations

NON-ITERATION COUNT all iterations

CRITERION

- Inputs of Interest: $\mathcal{I}$
- Variables of Interest: V
- Occurrences of Interest: $\mathcal{O}$

$\mathcal{O} = n \times \wp(\mathbb{N})$

Iteration Count only particular iterations

Non-Iteration Count all iterations

$\mathcal{O} = n \times \mathbb{N}$

C
R
I
T
E
R
I
O
N

Inputs of Interest: $\mathcal{I}$
Variables of Interest: V
Occurrences of Interest: $\mathcal{O}$

KL	Non - KL
the same paths	even different paths

CRITERION

- INPUTS OF INTEREST: $\mathcal{I}$
- VARIABLES OF INTEREST: V
- OCCURRENCES OF INTEREST: $\mathcal{O}$
- PATH REQUIREMENT $\mathcal{L}$

$\mathcal{L} = true$	$\mathcal{L} = false$
KL the same paths	NON - KL even different paths

Static	Conditioned	Dynamic
all possible inputs	some particular inputs	one particular input
$\mathcal{I} = \mathbb{M}$	$\varnothing \subset \mathcal{I} \subseteq \mathbb{M}$	$\lvert\mathcal{I}\rvert = 1$

CRITERION

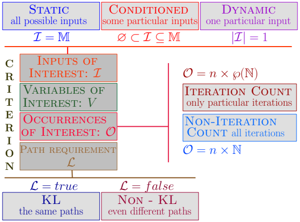
- Inputs of Interest: $\mathcal{I}$
- Variables of Interest: V
- Occurrences of Interest: $\mathcal{O}$
- Path requirement $\mathcal{L}$

Occurrences of Interest:

- $\mathcal{O} = n \times \wp(\mathbb{N})$ — **Iteration Count**: only particular iterations
- **Non-Iteration Count**: all iterations — $\mathcal{O} = n \times \mathbb{N}$

Path requirement:

$\mathcal{L} = true$	$\mathcal{L} = false$
KL	Non - KL
the same paths	even different paths



Generalized Slicing Criterion

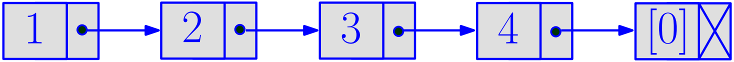
$$\mathcal{C} = \langle \mathcal{I}, \mathsf{V}, \mathcal{O}, \mathcal{L} \rangle,$$

WHERE

- $\mathcal{I} \subseteq \mathbb{M}$ - SET OF INPUTS OF INTEREST,
- V - SET OF VARIABLES OF INTEREST,
- $\mathcal{O} \in n \times \wp(\mathbb{N})$ - SET OF OCCURRENCES OF INTEREST,
- $\mathcal{L} \in \{true, false\}$ - DETERMINES A KL FORM.

ABSTRACT PROGRAM SLICING

REVERSING WELL-FORMED LISTS



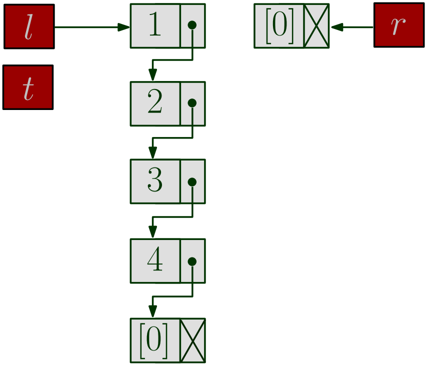
THE LAST ELEMENT OF A WELL-FORMED LIST IS [0]

REVERSING WELL-FORMED LISTS

```
list rev(list l) {
  list *last;
  list *tmp;
  while (l->next != null){
    tmp = l->next;
    l->next = last;
    last = l;
    l = tmp;
  }
  return last;
}
```

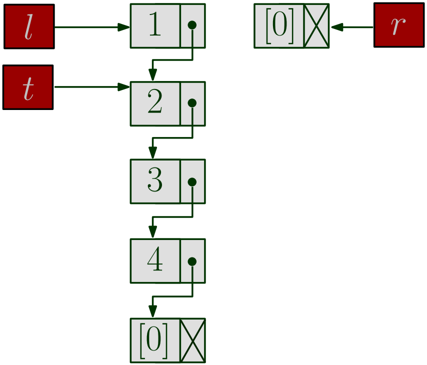
REVERSING WELL-FORMED LISTS

```
list rev(list l) {
  list *r;
  list *t;
  while (l->next != null){
    t = l->next;
    l->next = r;
    r = l;
    l = t;
  }
  return r;
}
```



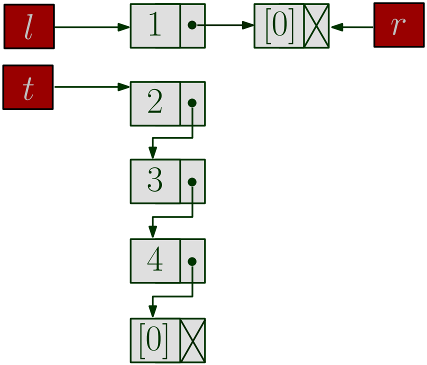
REVERSING WELL-FORMED LISTS

```
list rev(list l) {
  list *r;
  list *t;
  while (l->next != null){
    t = l->next;
    l->next = r;
    r = l;
    l = t;
  }
  return r;
}
```



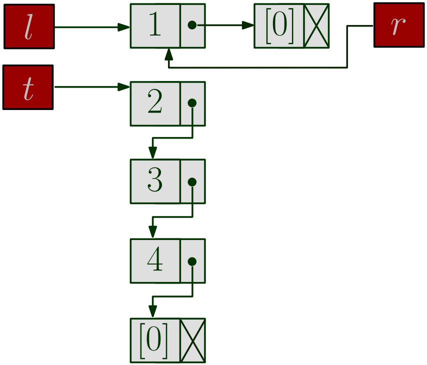
REVERSING WELL-FORMED LISTS

```
list rev(list l) {
  list *r;
  list *t;
  while (l->next != null){
    t = l->next;
    l->next = r;
    r = l;
    l = t;
  }
  return r;
}
```



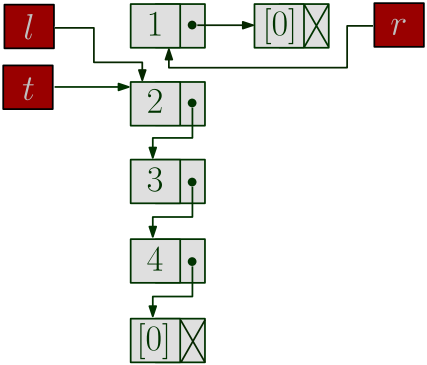
REVERSING WELL-FORMED LISTS

```
list rev(list l) {
  list *r;
  list *t;
  while (l->next != null){
    t = l->next;
    l->next = r;
    r = l;
    l = t;
  }
  return r;
}
```



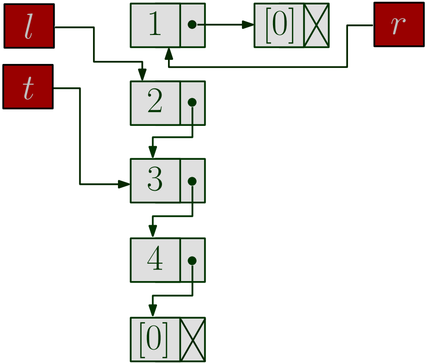
REVERSING WELL-FORMED LISTS

```
list rev(list l) {
  list *r;
  list *t;
  while (l->next != null){
    t = l->next;
    l->next = r;
    r = l;
    l = t;
  }
  return r;
}
```



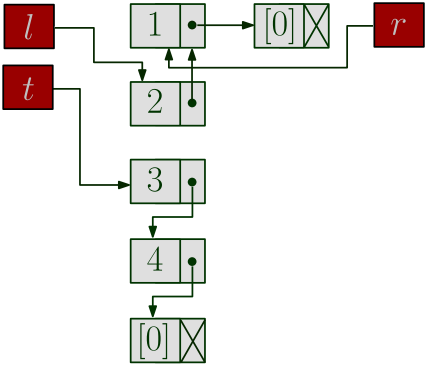
REVERSING WELL-FORMED LISTS

```
list rev(list l) {
  list *r;
  list *t;
  while (l->next != null){
    t = l->next;
    l->next = r;
    r = l;
    l = t;
  }
  return r;
}
```



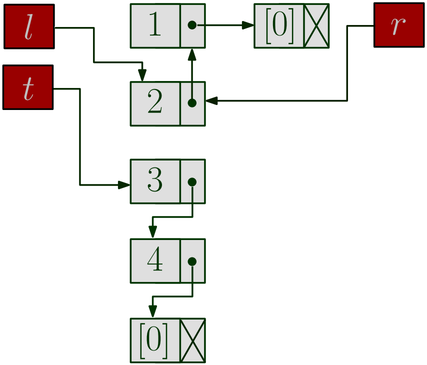
Reversing Well-Formed Lists

```
list rev(list l) {
  list *r;
  list *t;
  while (l->next != null){
    t = l->next;
    l->next = r;
    r = l;
    l = t;
  }
  return r;
}
```



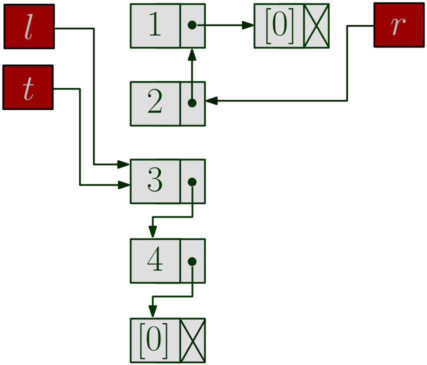
REVERSING WELL-FORMED LISTS

```
list rev(list l) {
  list *r;
  list *t;
  while (l->next != null){
    t = l->next;
    l->next = r;
    r = l;
    l = t;
  }
  return r;
}
```



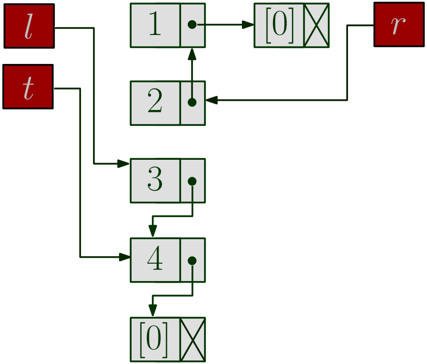
REVERSING WELL-FORMED LISTS

```
list rev(list l) {
  list *r;
  list *t;
  while (l->next != null){
    t = l->next;
    l->next = r;
    r = l;
    l = t;
  }
  return r;
}
```



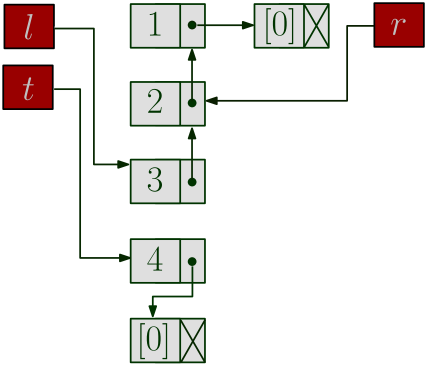
REVERSING WELL-FORMED LISTS

```
list rev(list l) {
  list *r;
  list *t;
  while (l->next != null){
    t = l->next;
    l->next = r;
    r = l;
    l = t;
  }
  return r;
}
```



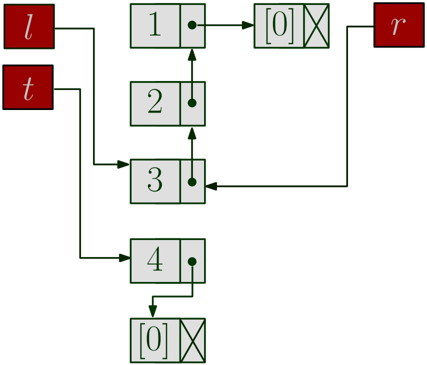
REVERSING WELL-FORMED LISTS

```
list rev(list l) {
  list *r;
  list *t;
  while (l->next != null){
    t = l->next;
    l->next = r;
    r = l;
    l = t;
  }
  return r;
}
```



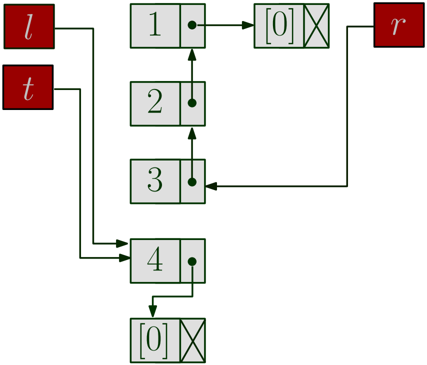
REVERSING WELL-FORMED LISTS

```
list rev(list l) {
  list *r;
  list *t;
  while (l->next != null){
    t = l->next;
    l->next = r;
    r = l;
    l = t;
  }
  return r;
}
```



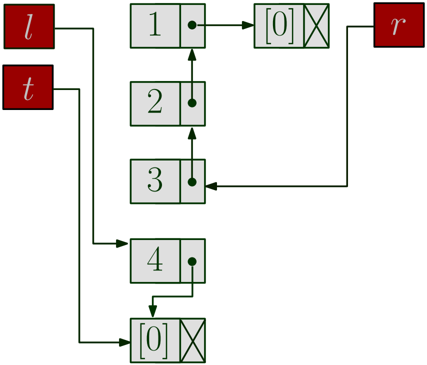
Reversing Well-Formed Lists

```
list rev(list l) {
  list *r;
  list *t;
  while (l->next != null){
    t = l->next;
    l->next = r;
    r = l;
    l = t;
  }
  return r;
}
```



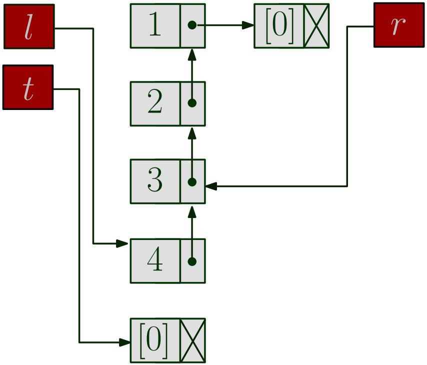
Reversing Well-Formed Lists

```
list rev(list l) {
  list *r;
  list *t;
  while (l->next != null){
    t = l->next;
    l->next = r;
    r = l;
    l = t;
  }
  return r;
}
```



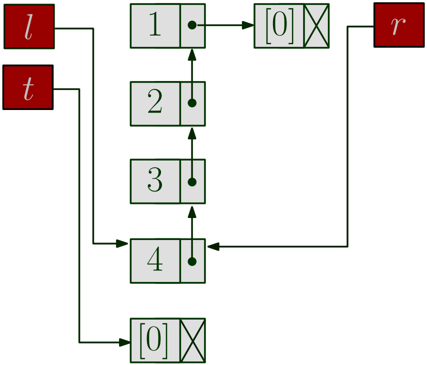
Reversing Well-Formed Lists

```
list rev(list l) {
  list *r;
  list *t;
  while (l->next != null){
    t = l->next;
    l->next = r;
    r = l;
    l = t;
  }
  return r;
}
```



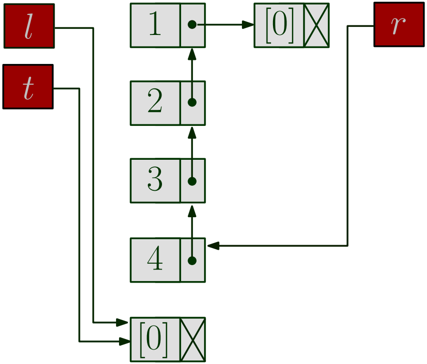
REVERSING WELL-FORMED LISTS

```
list rev(list l) {
  list *r;
  list *t;
  while (l->next != null){
    t = l->next;
    l->next = r;
    r = l;
    l = t;
  }
  return r;
}
```



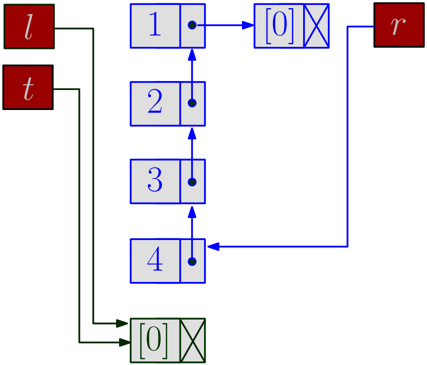
REVERSING WELL-FORMED LISTS

```
list rev(list l) {
  list *r;
  list *t;
  while (l->next != null){
    t = l->next;
    l->next = r;
    r = l;
    l = t;
  }
  return r;
}
```



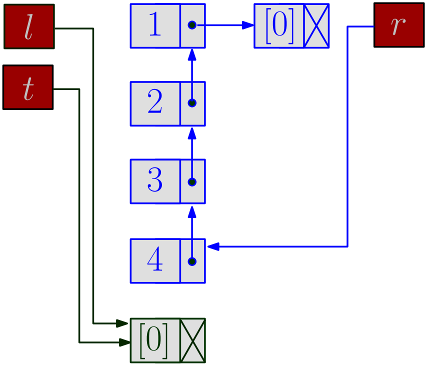
REVERSING WELL-FORMED LISTS

```
list rev(list l) {
  list *r;
  list *t;
  while (l->next != null){
    t = l->next;
    l->next = r;
    r = l;
    l = t;
  }
  return r;
}
```



Reversing Well-Formed Lists

```
list rev(list l) {
  list *r;
  list *t;
  while (l->next != null){
    t = l->next;
    l->next = r;
    r = l;
    l = t;
  }
  return r;
}
```



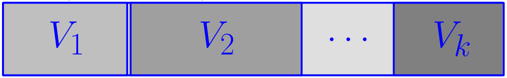
⇒ If r is well-formed before while,
it is well-formed after while as well

GENERALIZED CRITERION:

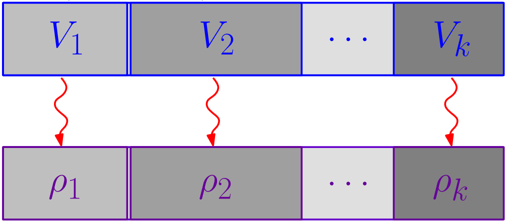
Variables			Paths
$\mathcal{I}$	V	$\mathcal{O}$	$\mathcal{L}$
Inputs		Occurrences	

V - Variables of Interest

$\mathcal{V} = \langle V_1, \ldots, V_k \rangle$ - A PARTITION OF V

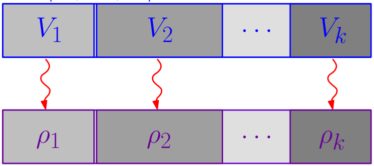


$\mathcal{V} = \langle V_1, \ldots, V_k \rangle$ - A PARTITION OF V



$\mathcal{A} = \langle \rho_1, \ldots, \rho_k \rangle$ - PROPERTIES OF INTEREST

$\mathcal{V} = \langle V_1, \ldots, V_k \rangle$ - A PARTITION OF V



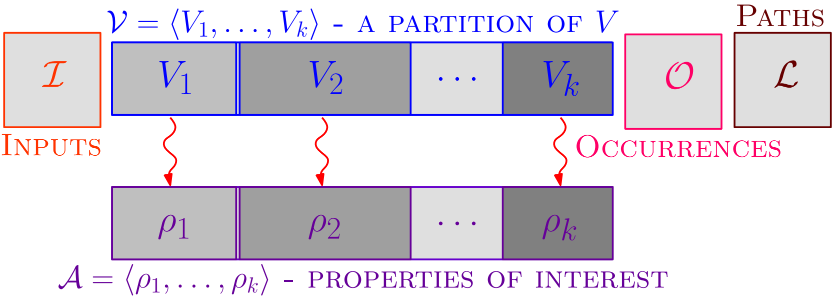
$\mathcal{A} = \langle \rho_1, \ldots, \rho_k \rangle$ - PROPERTIES OF INTEREST

$$\mathsf{Var} = \{x_1, x_2, x_3, x_4\} \qquad V = \{x_1, x_2, x_3\}$$

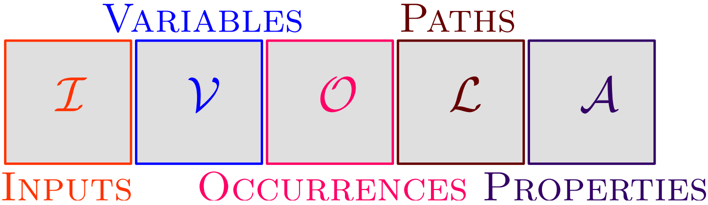
PROPERTIES OF INTEREST: SIGN OF $x_1 \times x_2$ AND PARITY OF x_3

$$\Rightarrow \mathcal{V} = \langle \{x_1, x_2\}, \{x_3\} \rangle \qquad \mathcal{A} = \langle \textsc{Sign}, \textsc{Par} \rangle$$

$$\textsc{Sign}(x, y) = \begin{cases} \text{POS} & \text{if } x * y > 0 \\ 0 & \text{if } x * y = 0 \\ \text{NEG} & \text{otherwise} \end{cases} \qquad \textsc{Par}(x) = \begin{cases} \text{EVEN} & \text{if } x \equiv_2 0 \\ \text{ODD} & \text{otherwise} \end{cases}$$


$\mathcal{V} = \langle V_1, \ldots, V_k \rangle$ - A PARTITION OF V
PATHS
$\mathcal{I}$
V_1
V_2
$\cdots$
V_k
$\mathcal{O}$
$\mathcal{L}$
INPUTS
OCCURRENCES
ρ_1
ρ_2
$\cdots$
ρ_k
$\mathcal{A} = \langle \rho_1, \ldots, \rho_k \rangle$ - PROPERTIES OF INTEREST

GENERALIZED ABSTRACT CRITERION:



Abstract Slicing - Intuitive Definition

- P, Q - executable programs
- $\mathcal{C}_{\mathcal{A}} = \langle \mathcal{I}, \mathcal{V}, \mathcal{O}, \mathcal{L}, \mathcal{A} \rangle$ - abstract criterion
- Q is $\mathcal{T}$ - Abstract Slice of P iff

$\forall \sigma \in \mathcal{I}$ and $\forall \langle n, k \rangle \in \mathcal{O}$:

if P^{σ} reaches $\langle n, k \rangle$ then Q^{σ} reaches $\langle n, k \rangle$

and $\mathcal{A}(\mathcal{V})$ in P is equal to $\mathcal{A}(\mathcal{V})$ in Q

- $\mathcal{T} \in \{$Static, Conditioned, Dynamic$\}$

Abstract Slicing - Intuitive Definition

- P, Q - executable programs
- $\mathcal{C}_{\mathcal{A}} = \langle \mathcal{I}, \mathcal{V}, \mathcal{O}, \mathcal{L}, \mathcal{A} \rangle$ - abstract criterion
- Q is $\mathcal{T}$ - Abstract Slice of P iff

$\forall \sigma \in \mathcal{I}$ and $\forall \langle n, k \rangle \in \mathcal{O}$:

if P^{σ} reaches $\langle n, k \rangle$ then Q^{σ} reaches $\langle n, k \rangle$

and $\mathcal{A}(\mathcal{V})$ in P is equal to $\mathcal{A}(\mathcal{V})$ in Q

- $\mathcal{T} \in \{$Static, Conditioned, Dynamic$\}$

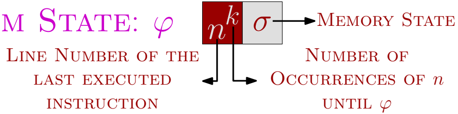
Is it possible to

1. Extend the framework and Formally define these forms?
2. Insert them in the hierarchy?

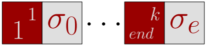
SYSTEM STATES AND TRACES

[Binkley et al. '06]

SYSTEM STATE: φ



TRACE: $T_P^{\sigma_0} = \bigoplus_i \varphi_i$



SYSTEM STATES AND TRACES

```
begin
   read(n);
   i:=1;
   s:=0;
   p:=1;
   while (i<=n) do
   begin
      s:=s+i;
      p:=p*i;
      i:=i+1;
   end;
   write(s);
   write(p);
end;
```

$$\sigma = \{n \leftarrow 2\}$$

System States and Traces

```
1   begin
2     read(n);
3     i:=1;
4     s:=0;
5     p:=1;
6     while (i<=n) do
7     begin
8       s:=s+i;
9       p:=p*i;
10      i:=i+1;
11    end;
12    write(s);
13    write(p);
14  end;
```

$$\sigma = \{n \leftarrow 2\}$$

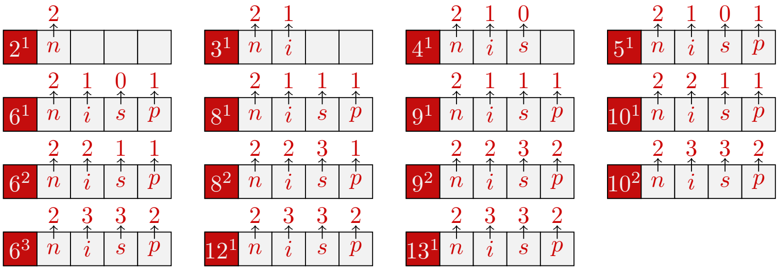
State	n	i	s	p
2^1	2			
3^1	2	1		
4^1	2	1	0	
5^1	2	1	0	1
6^1	2	1	0	1
8^1	2	1	1	1
9^1	2	1	1	1
10^1	2	2	1	1
6^2	2	2	1	1
8^2	2	2	3	1
9^2	2	2	3	2
10^2	2	3	3	2
6^3	2	3	3	2
12^1	2	3	3	2
13^1	2	3	3	2

System States and Traces

L - Additional Points of Interested

$$Proj'^{\alpha}_{(\mathcal{V},\mathcal{O},L,\mathcal{A})}(n,k,\sigma) \stackrel{\text{def}}{=} \begin{cases} (n, \sigma \restriction^{\alpha} \mathcal{V}) & \text{if } (n,k) \in \mathcal{O} \\ (n, \sigma \restriction^{\alpha} \varnothing) & \text{if } (n,k) \notin \mathcal{O} \wedge n \in L \\ \lambda & \text{otherwise} \end{cases}$$

$$\mathcal{V} = \langle\{i\},\{s\}\rangle,\ \mathcal{O} = \{8\} \times \mathbb{N},\ L = \{6\},\ \mathcal{A} = \langle \text{Sign}, \text{Par} \rangle$$

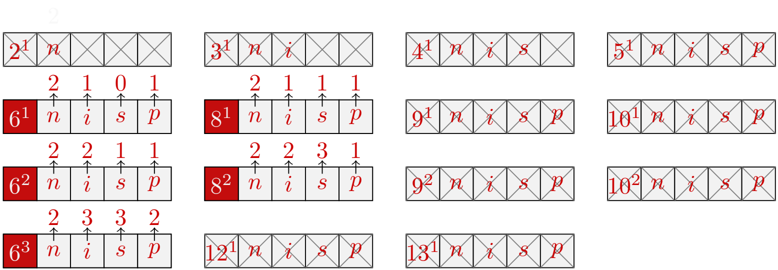


SYSTEM STATES AND TRACES

L - ADDITIONAL POINTS OF INTERESTED

$$Proj'^{\alpha}_{(\mathcal{V},\mathcal{O},L,\mathcal{A})}(n,k,\sigma) \stackrel{\text{def}}{=} \begin{cases} (n, \sigma \restriction^{\alpha} \mathcal{V}) & \text{if } (n,k) \in \mathcal{O} \\ (n, \sigma \restriction^{\alpha} \varnothing) & \text{if } (n,k) \notin \mathcal{O} \wedge n \in L \\ \lambda & \text{otherwise} \end{cases}$$

$$\mathcal{V} = \langle \{i\}, \{s\} \rangle,\ \mathcal{O} = \{8\} \times \mathbb{N},\ L = \{6\},\ \mathcal{A} = \langle \text{SIGN}, \text{PAR} \rangle$$

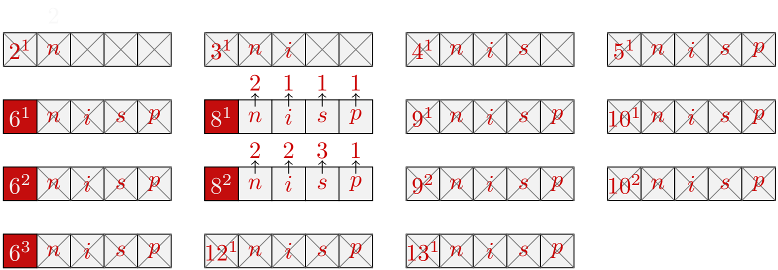


SYSTEM STATES AND TRACES

L - ADDITIONAL POINTS OF INTERESTED

$$Proj'^{\alpha}_{(\mathcal{V},\mathcal{O},L,\mathcal{A})}(n,k,\sigma) \stackrel{\text{def}}{=} \begin{cases} (n, \sigma \restriction^{\alpha} \mathcal{V}) & \text{if } (n,k) \in \mathcal{O} \\ (n, \sigma \restriction^{\alpha} \varnothing) & \text{if } (n,k) \notin \mathcal{O} \wedge n \in L \\ \lambda & \text{otherwise} \end{cases}$$

$$\mathcal{V} = \langle \{i\}, \{s\} \rangle,\ \mathcal{O} = \{8\} \times \mathbb{N},\ L = \{6\},\ \mathcal{A} = \langle \text{SIGN}, \text{PAR} \rangle$$

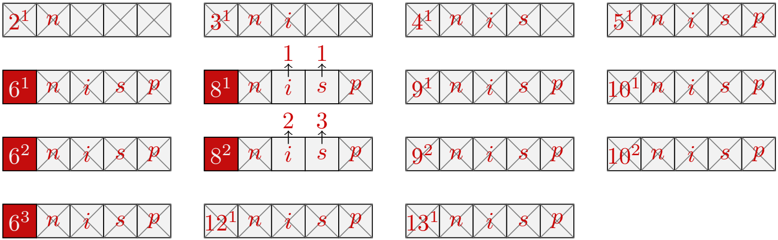


SYSTEM STATES AND TRACES

L - ADDITIONAL POINTS OF INTERESTED

$$Proj'^{\alpha}_{(\mathcal{V},\mathcal{O},L,\mathcal{A})}(n,k,\sigma) \stackrel{\text{def}}{=} \begin{cases} (n, \sigma \restriction^{\alpha} \mathcal{V}) & \text{if } (n,k) \in \mathcal{O} \\ (n, \sigma \restriction^{\alpha} \varnothing) & \text{if } (n,k) \notin \mathcal{O} \wedge n \in L \\ \lambda & \text{otherwise} \end{cases}$$

$$\mathcal{V} = \langle \{i\}, \{s\} \rangle, \ \mathcal{O} = \{8\} \times \mathbb{N}, \ L = \{6\}, \ \mathcal{A} = \langle \text{SIGN}, \text{PAR} \rangle$$

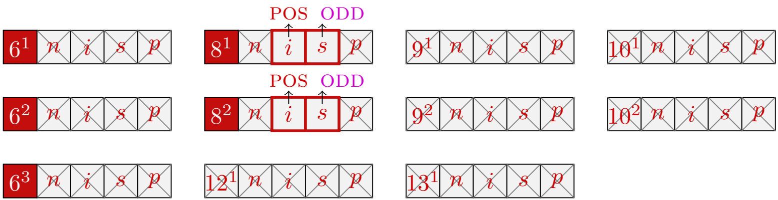


System States and Traces

L - Additional Points of Interested

$$Proj'^{\alpha}_{(\mathcal{V},\mathcal{O},L,\mathcal{A})}(n,k,\sigma) \stackrel{\text{def}}{=} \begin{cases} (n, \sigma \restriction^{\alpha} \mathcal{V}) & \text{if } (n,k) \in \mathcal{O} \\ (n, \sigma \restriction^{\alpha} \varnothing) & \text{if } (n,k) \notin \mathcal{O} \wedge n \in L \\ \lambda & \text{otherwise} \end{cases}$$

$$\mathcal{V} = \langle \{i\}, \{s\} \rangle, \ \mathcal{O} = \{8\} \times \mathbb{N}, \ L = \{6\}, \ \mathcal{A} = \langle \text{SIGN}, \text{PAR} \rangle$$



ABSTRACT UNIFIED EQUIVALENCE

- P, Q - EXECUTABLE PROGRAMS,
- l_P, l_Q - SETS OF LINE NUMBERS OF P AND Q
- $\mathcal{C}_{\mathcal{A}} = \langle \mathcal{I}, \mathcal{V}, \mathcal{O}, \mathcal{L}, \mathcal{A} \rangle$ - ABSTRACT CRITERION
- $L_{\mathcal{L}}(P, Q) = \mathcal{L} \; ? \; l_P \cap l_Q \; : \; \varnothing$
- P IS ABSTRACT EQUIVALENT TO Q ($P \;\; \mathcal{U}^{\mathcal{A}}(\mathcal{I}, \mathcal{V}, \mathcal{O}, L_{\mathcal{L}}, \mathcal{A}) \;\; Q$) IFF

$$\forall \sigma \in \mathcal{I}. Proj^{\alpha}_{(\mathcal{V}, \mathcal{O}, L_{\mathcal{L}}, \mathcal{A})}(T^{\sigma}_{P}) = Proj^{\alpha}_{(\mathcal{V}, \mathcal{O}, L_{\mathcal{L}}, \mathcal{A})}(T^{\sigma}_{Q})$$

Extended Framework

Semantic Constraint

$$\mathcal{E}_{\mathcal{A}} \stackrel{\text{DEF}}{=} \lambda(\mathcal{I}, \mathcal{V}, \mathcal{O}, \mathcal{L}, \mathcal{A}).\, \mathcal{U}^{\mathcal{A}}(\mathcal{I}, \mathcal{V}, \mathcal{O}, L_{\mathcal{L}}, \mathcal{A})$$

$\langle \sqsubseteq, \mathcal{E}_{\mathcal{A}} \rangle$ - Representation of Abstract Forms of Slicing

Extended Framework

SEMANTIC CONSTRAINT

$$\mathcal{E}_{\mathcal{A}} \stackrel{\text{DEF}}{=} \lambda(\mathcal{I}, \mathcal{V}, \mathcal{O}, \mathcal{L}, \mathcal{A}).\,\mathcal{U}^{\mathcal{A}}(\mathcal{I}, \mathcal{V}, \mathcal{O}, L_{\mathcal{L}}, \mathcal{A})$$

$\langle \sqsubseteq, \mathcal{E}_{\mathcal{A}} \rangle$ - REPRESENTATION OF ABSTRACT FORMS OF SLICING

$\Updownarrow$

WE HAVE INSERTED ABSTRACT SLICING IN FORMAL FRAMEWORK

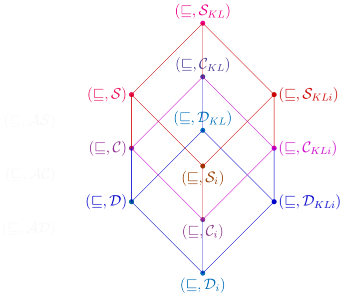
$\Downarrow$ - EXTENDED THEORY

ENRICHED HIERARCHY

Extended Framework

$\langle \sqsubseteq, \mathcal{E}_{\mathcal{A}} \rangle$ - Representation of Abstract Forms of Slicing

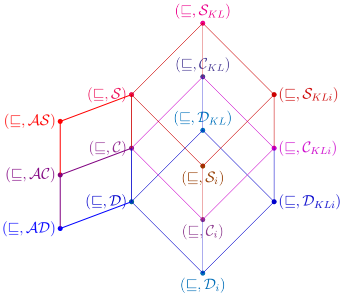
Enriched Hierarchy



Extended Framework

$\langle \sqsubseteq, \mathcal{E}_{\mathcal{A}} \rangle$ - REPRESENTATION OF ABSTRACT FORMS OF SLICING

ENRICHED HIERARCHY

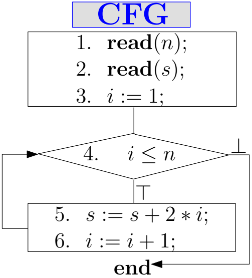


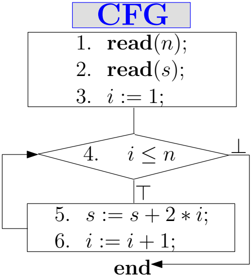
Towards an Implementation

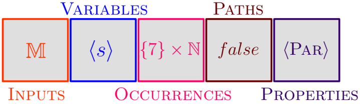
Towards an Implementation - Idea

- START FROM A STATIC SLICE OF A PROGRAM
- DERIVE AN ABSTRACTION ρ FROM $\mathcal{C}_{\mathcal{A}}$ AND CONSTRUCT ABSTRACT STATES USING ρ
- DETERMINE AN ABSTRACT STATE GRAPH ASG
- ABSTRACT SLICE CORRESPONDS TO A PRUNED ASG

```
read(n);
read(s);
i := 1;
while (i<=n) do
   s := s + 2*i;
   i := i+1;
od
```

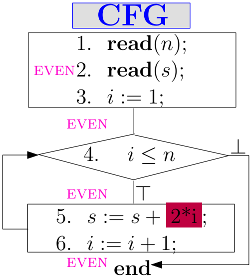

CFG
1. read(n);
2. read(s);
3. i := 1;
4. i ≤ n
⊥
⊤
5. s := s + 2 ∗ i;
6. i := i + 1;
end





$\mathbb{M}$- all possible inputs

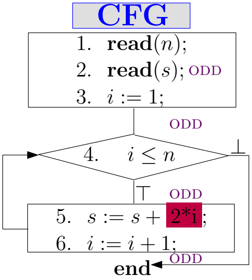
$$\text{PAR}(x) = \begin{cases} \text{EVEN} & \text{if } x \equiv_2 0 \\ \text{ODD} & \text{otherwise} \end{cases}$$



VARIABLES			PATHS	
$\mathbb{M}$	$\langle s \rangle$	$\{7\} \times \mathbb{N}$	*false*	$\langle$PAR$\rangle$
INPUTS		OCCURRENCES		PROPERTIES

$\mathbb{M}$- all possible inputs

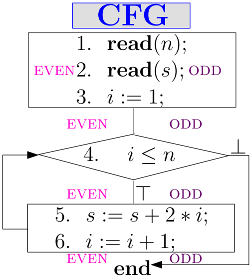
$$\text{PAR}(x) = \begin{cases} \text{EVEN} & \text{if } x \equiv_2 0 \\ \text{ODD} & \text{if } x \equiv_2 0 \end{cases}$$

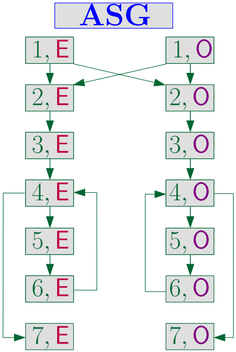


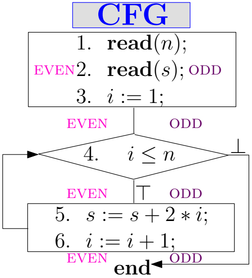
VARIABLES			PATHS	
$\mathbb{M}$	$\langle s \rangle$	$\{7\} \times \mathbb{N}$	*false*	$\langle$PAR$\rangle$
INPUTS		OCCURRENCES		PROPERTIES

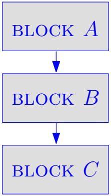
$\mathbb{M}$- all possible inputs

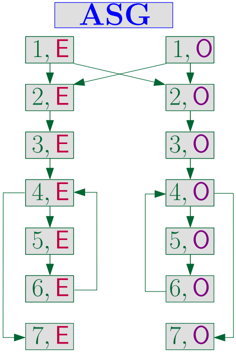
$$\text{PAR}(x) = \begin{cases} \text{EVEN} & \text{if } x \equiv_2 0 \\ \text{ODD} & \text{if } x \equiv_2 0 \end{cases}$$


CFG
1. read(n);
EVEN 2. read(s); ODD
3. i := 1;
EVEN ODD
4. i ≤ n
⊥
EVEN ⊤ ODD
5. s := s + 2 ∗ i;
6. i := i + 1;
EVEN ODD
end

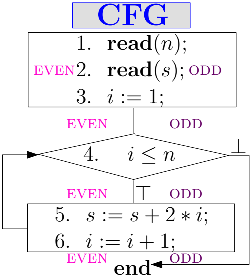

ASG
1, E
1, O
2, E
2, O
3, E
3, O
4, E
4, O
5, E
5, O
6, E
6, O
7, E
7, O

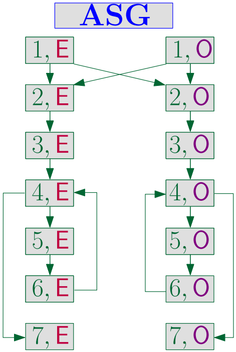


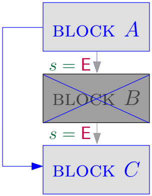




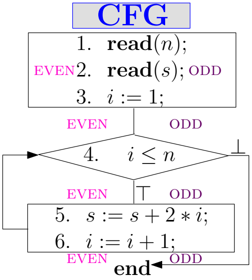
Block - a strongly connected component with 1 incoming and 1 outgoing edge

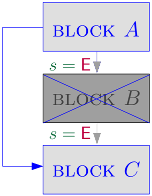


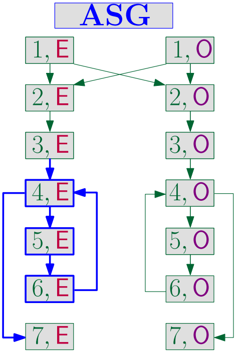




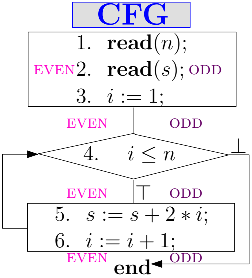
BLOCK - a strongly connected component with 1 incoming and 1 outgoing edge

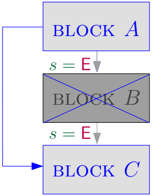


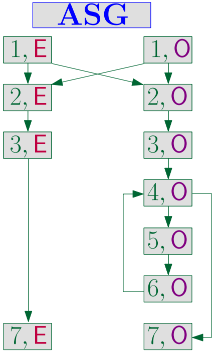


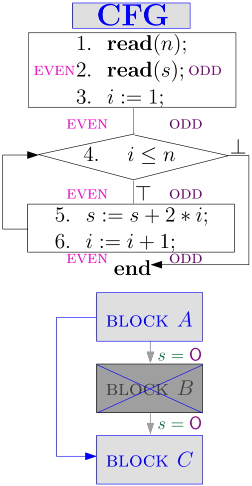


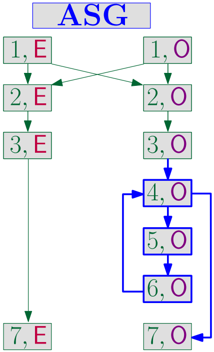
Block $\langle 4, \mathsf{E}\rangle, \langle 5, \mathsf{E}\rangle, \langle 6, \mathsf{E}\rangle$
Input: $s = \mathsf{E}$ Output: $s = \mathsf{E}$
$\Rightarrow$ we can remove it


CFG
1. read(n);
EVEN 2. read(s); ODD
3. i := 1;
EVEN
ODD
4. i ≤ n
⊥
EVEN
⊤
ODD
5. s := s + 2 * i;
6. i := i + 1;
EVEN
ODD
end

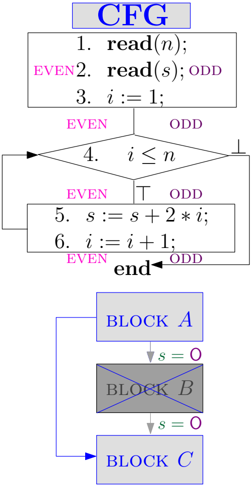

BLOCK A
s = E
BLOCK B
s = E
BLOCK C

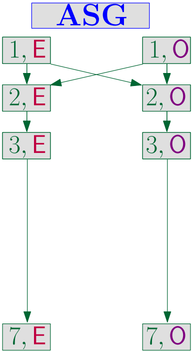

ASG
1, E
1, O
2, E
2, O
3, E
3, O
4, O
5, O
6, O
7, E
7, O

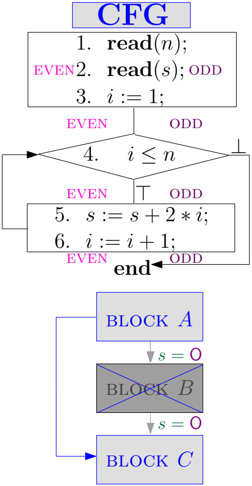


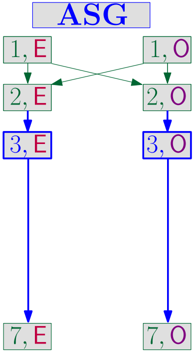


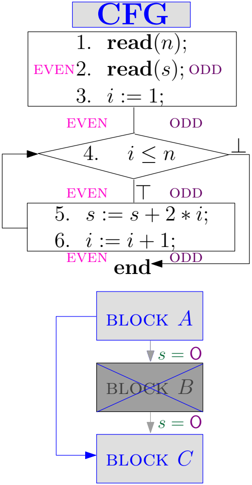
Block $\langle 4, \mathsf{O}\rangle, \langle 5, \mathsf{O}\rangle, \langle 6, \mathsf{O}\rangle$
Input: $s = \mathsf{O}$ Output: $s = \mathsf{O}$
$\Rightarrow$ we can remove it

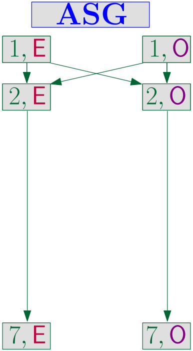

CFG
1. read(n);
EVEN 2. read(s); ODD
3. i := 1;
EVEN
ODD
4. i ≤ n
⊥
EVEN
⊤
ODD
5. s := s + 2 * i;
6. i := i + 1;
EVEN
ODD
end
BLOCK A
s = O
BLOCK B
s = O
BLOCK C


ASG
1, E
1, O
2, E
2, O
3, E
3, O
7, E
7, O

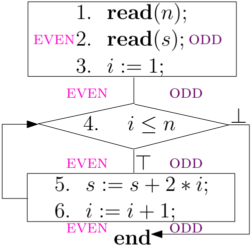

CFG
1. read(n);
EVEN 2. read(s); ODD
3. i := 1;
EVEN ODD
4. i ≤ n
⊥
EVEN ⊤ ODD
5. s := s + 2 * i;
6. i := i + 1;
EVEN ODD
end
BLOCK A
s = O
BLOCK B
s = O
BLOCK C


ASG
1, E
1, O
2, E
2, O
3, E
3, O
7, E
7, O

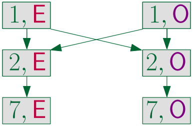

CFG
1. read(n);
EVEN 2. read(s); ODD
3. i := 1;
EVEN
ODD
4. i ≤ n
⊥
EVEN
⊤
ODD
5. s := s + 2 * i;
6. i := i + 1;
EVEN
ODD
end
BLOCK A
s = O
BLOCK B
s = O
BLOCK C


ASG
1, E
1, O
2, E
2, O
7, E
7, O

CFG



ASG



1. **read**(n);
2. **read**(s);
7. **end**

Problems

- If the property used for the construction of ASG is too much abstract, the Simple Approach returns the static slice
- This approach cannot be used for the extraction of dynamic and conditional slices
- Extended Approach is one possible refinement of this algorithm

OUR CONTRIBUTION:

- GENERALIZED SLICING CRITERIA (TRADITIONAL AND ABSTRACT VERSIONS)
- EXTENSION OF UNIFIED FORMAL FRAMEWORK
- FORMAL DEFINITION OF ABSTRACT PROGRAM SLICING
- EXTENSION OF HIERARCHY
- FIRST STEPS TOWARDS AN IMPLEMENTATION

IDEAS FOR THE FUTURE

- IMPROVEMENT AND IMPLEMENTATION OF PROPOSED ALGORITHM(S)
- PREDICATE ABSTRACTION AND MODEL CHECKING VS. ABSTRACT SLICING
- OBFUSCATION AND WATERMARKING VS. ABSTRACT SLICING

Thank You