Concrete-Value Analysis with Symbolic Memory Graph Domain Extension and CEGAR in CPAchecker

Daniel Baier

LMU Munich, Germany

Motivation

```
void main() {
  int length = 5;
  int *array = malloc(length * sizeof(int));

  for (int i = 0; i <= length; i++) {
    *array = i;
    array++;
  }

  assert(*array == length);
  free(array);
}
```

Motivation

```
void main() {
  int length = 5;
  int *array = malloc(length * sizeof(int));


  for (int i = 0; i <= length; i++) {
    *array = i;
    array++;
  }

  assert(*array == length);
  free(array);
}
```

Motivation

```
void main() {
  int length = 5;
  int *array = malloc(length * sizeof(int));


  for (int i = 0; i <= length; i++) {
    *array = i;
    array++;
  }

  assert(*array == length);
  free(array);
}
```

Analysis for Heap-Memory

- Fast
- No background solver
- Reach-safety
- Memory-safety

Motivation

```
void main() {
  int length = 5;
  int *array = malloc(length * sizeof(int));


  for (int i = 0; i <= length; i++) {
    *array = i;
    array++;
  }

  assert(*array == length);
  free(array);
}
```

Analysis for Heap-Memory

- Fast
- No background solver
- Reach-safety
- Memory-safety

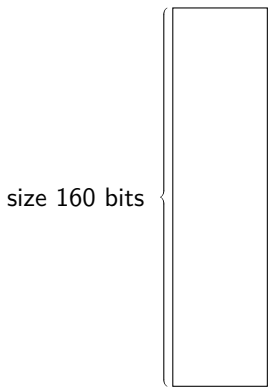
Combine

- Value Analysis
- Symbolic Memory Graphs

Symbolic Memory Graphs (SMG)

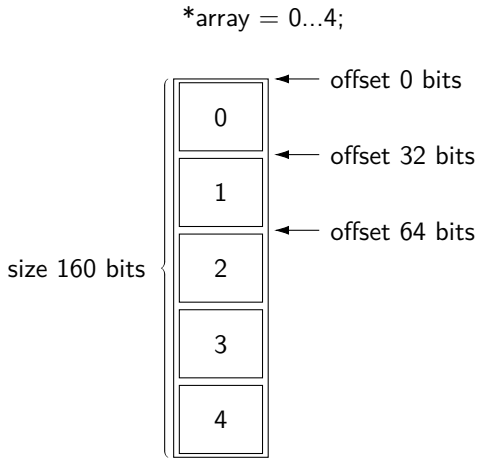
int *array = malloc(5 * sizeof(int));

- Graph based memory model



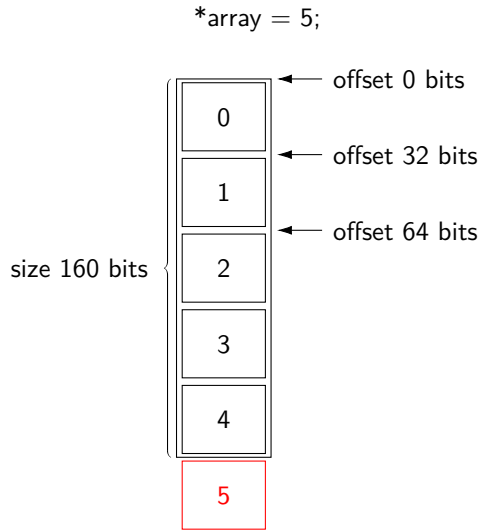
Symbolic Memory Graphs (SMG)

- Graph based memory model



Symbolic Memory Graphs (SMG)

- Graph based memory model
- Detects overwrite/underwrite
- Detects invalid pointer access



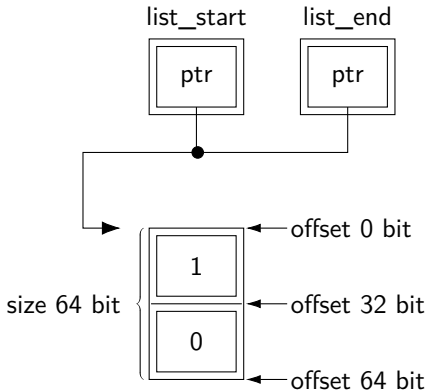
Symbolic Memory Graphs (SMG)

- Graph based memory model
- Detects overwrite/underwrite
- Detects invalid pointer access
- Detects invalid free()

free(array);

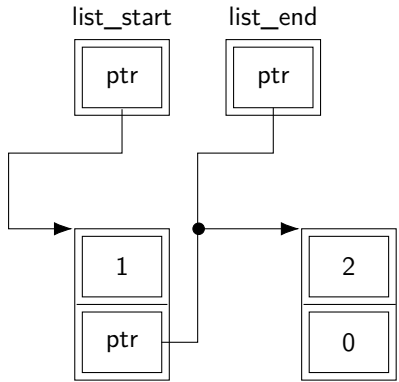
Symbolic Memory Graphs

```
struct node {
  int value;
  struct node * next;
} List;

int main() {
  List * list_start = malloc(...);
  List * list_end = list_start;
  list_end->value = 1;
  list_end->next = 0;




}
```

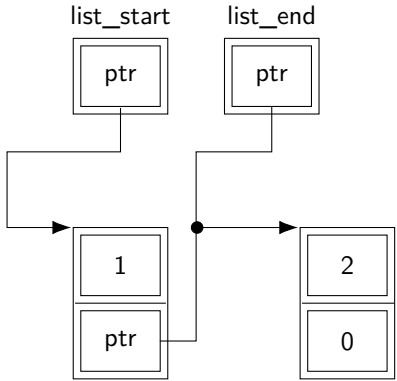


Symbolic Memory Graphs

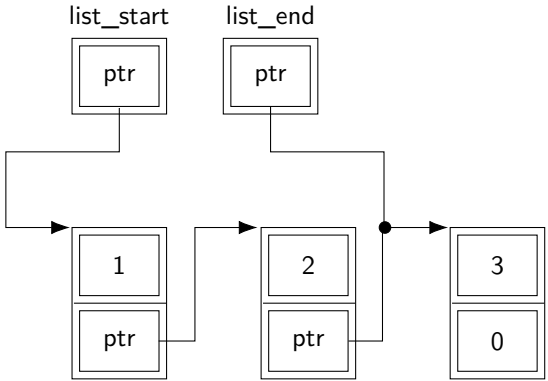
```
struct node {
  int value;
  struct node * next;
} List;

int main() {
  List * list_start = malloc(...);
  List * list_end = list_start;
  list_end->value = 1;
  list_end->next = 0;


  list_end->next = malloc(...);
  list_end = list_end -> next;
  list_end->next = 0;
  list_end->value = 2;

}
```



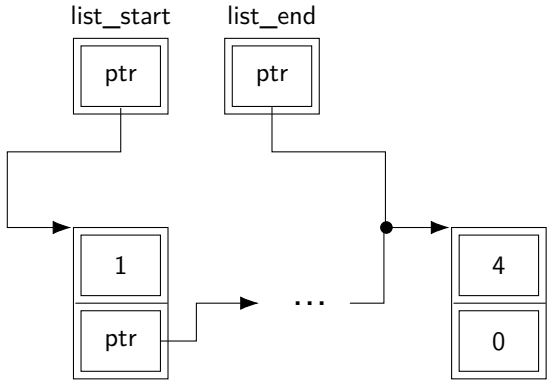
Symbolic Memory Graphs - List Abstraction



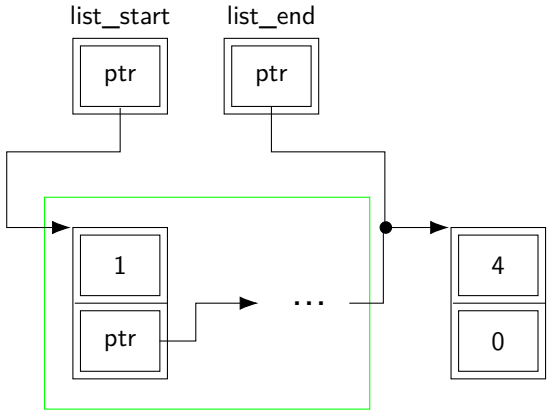
Symbolic Memory Graphs - List Abstraction



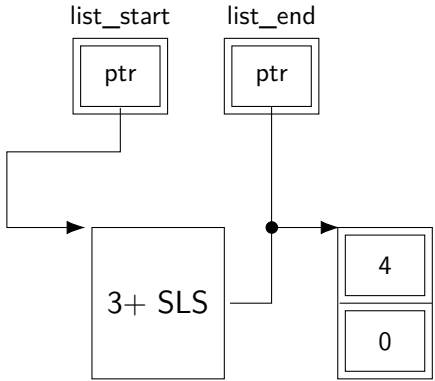
Symbolic Memory Graphs - List Abstraction



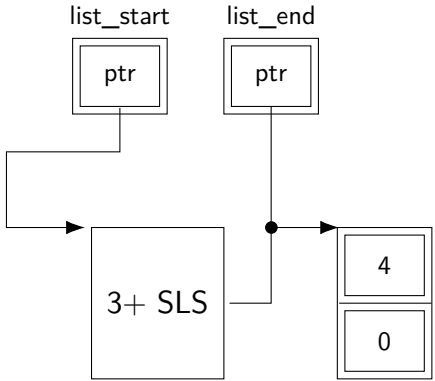
Symbolic Memory Graphs - List Abstraction



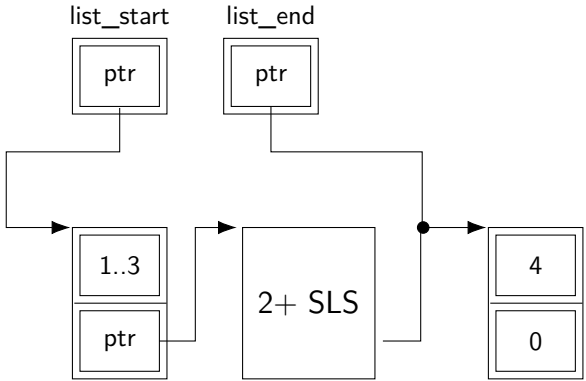
Symbolic Memory Graphs - List Abstraction



Symbolic Memory Graphs - Materialization



Symbolic Memory Graphs - Materialization

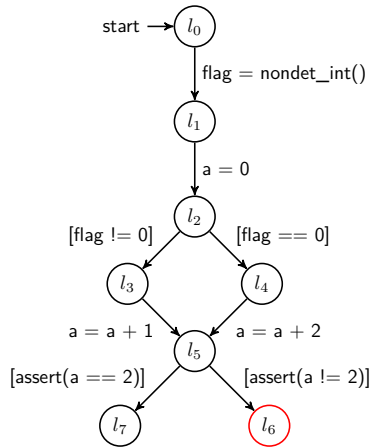


Value Analysis

- Value Analysis as in CPAchecker tracks only local and global variables
- Only concrete values are tracked
- All other values are unknown and are over-approximated
- Unknown values can be assumed in assumption

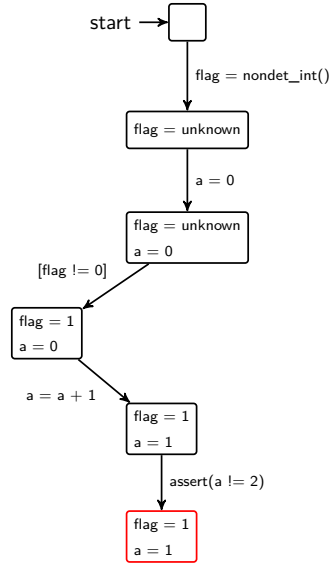
Value Analysis

```
int main() {
  int flag = nondet_int();
  int a = 0;

  if (flag) {
    a = a + 1;
  } else {
    a = a + 2;
  }
  assert(a == 2);
}
```



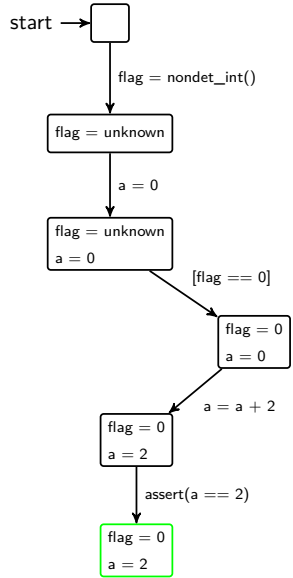
Value Analysis

```
int main() {
  int flag = nondet_int();
  int a = 0;

  if (flag) {
    a = a + 1;
  } else {
    a = a + 2;
  }
  assert(a == 2);
}
```



Value Analysis

```
int main() {
  int flag = nondet_int();
  int a = 0;

  if (flag) {
    a = a + 1;
  } else {
    a = a + 2;
  }
  assert(a == 2);
}
```



Value Analysis - CEGAR

- Idea: track only necessary variables
- CEGAR approach
- Uses interpolation without external solver
- Modified for heap memory

```
int main() {
  int flag = nondet_int();
  int a = 0;
  int b = 0;

  while(b <= 0) {
    b = b - 1;
  }

  if (flag) {
    a = a + 1;
  } else {
    a = a + 2;
  }
  assert(a == 2);
}
```

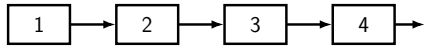
Value Analysis - CEGAR

- Idea: track only necessary variables
- CEGAR approach
- Uses interpolation without external solver
- Modified for heap memory

```
int main() {
  int flag = nondet_int();
  int a = 0;
  int b = 0;

  while(b <= 0) {
    b = b - 1;
  }

  if (flag) {
    a = a + 1;
  } else {
    a = a + 2;
  }
  assert(a == 2);
}
```

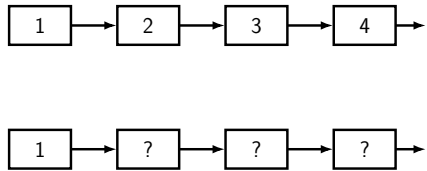
Heap Value CEGAR

- Abstracted lists depend on equal values
- Idea: Use only needed values in the heap
- Goal: Make more lists abstractable



Heap Value CEGAR

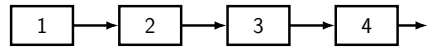
- Abstracted lists depend on equal values
- Idea: Use only needed values in the heap
- Goal: Make more lists abstractable

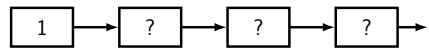


Heap Value CEGAR

- Abstracted lists depend on equal values
- Idea: Use only needed values in the heap
- Goal: Make more lists abstractable

How to find necessary heap values?



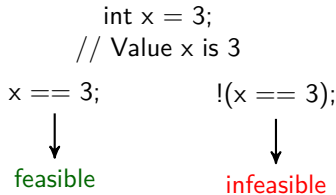


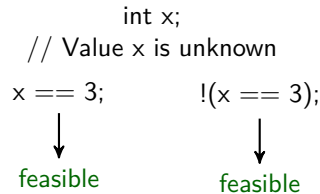
Heap Value CEGAR

- Abstracted lists depend on equal values
- Idea: Use only needed values in the heap
- Goal: Make more lists abstractable

How to find necessary heap values?

- Needed values do not necessarily change the feasibility of the current path
- But they change branching
- Idea: check branching





Value Analysis with SMGs

SMG-ValueAnalysis Advantages

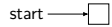
- Value Analysis reasoning
- CEGAR
  - Program variable tracking
  - Heap value tracking
- SMGs as complete memory model
  - List abstraction with shape refinement
- Supports reach-safety
- Supports memory-safety

Implementation

- Analyzes C programs
- CPA in CPAchecker
- Written in Java
- Supports
  - All (concrete) memory allocation
  - All C99 memory operations
  - Most important functions

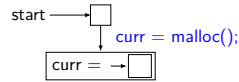
Value Analysis with SMGs - Example

```
typedef struct node {
  int data;
  struct node *next;
} *List;

void main(int flag) {
  List curr = (List) malloc(...);
  List old = curr;

  while (nondet_int()) {
    if (flag) curr->data = 1;
    else      curr->data = 2;
    curr->next = (List) malloc(...);
    curr = curr->next;
  }

  curr->data = 3;
  curr = old;
  if (flag)
    while (curr->data == 1) curr = curr->next;
  else
    while (curr->data == 2) curr = curr->next;
  assert(curr->data == 3);
}
```



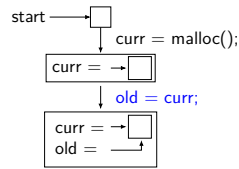
Value Analysis with SMGs - Example

```
typedef struct node {
  int data;
  struct node *next;
} *List;

void main(int flag) {
  List curr = (List) malloc(...);
  List old = curr;

  while (nondet_int()) {
    if (flag) curr->data = 1;
    else      curr->data = 2;
    curr->next = (List) malloc(...);
    curr = curr->next;
  }

  curr->data = 3;
  curr = old;
  if (flag)
    while (curr->data == 1) curr = curr->next;
  else
    while (curr->data == 2) curr = curr->next;
  assert(curr->data == 3);
}
```



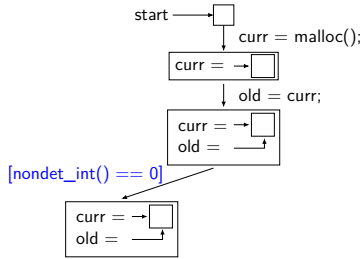
Value Analysis with SMGs - Example

```
typedef struct node {
  int data;
  struct node *next;
} *List;

void main(int flag) {
  List curr = (List) malloc(...);
  List old = curr;

  while (nondet_int()) {
    if (flag) curr->data = 1;
    else      curr->data = 2;
    curr->next = (List) malloc(...);
    curr = curr->next;
  }

  curr->data = 3;
  curr = old;
  if (flag)
    while (curr->data == 1) curr = curr->next;
  else
    while (curr->data == 2) curr = curr->next;
  assert(curr->data == 3);
}
```



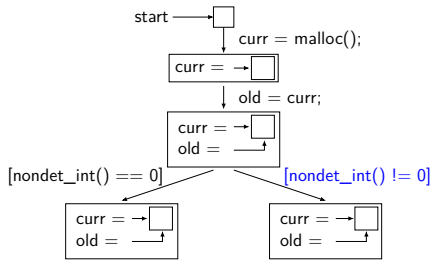
Value Analysis with SMGs - Example

```
typedef struct node {
  int data;
  struct node *next;
} *List;

void main(int flag) {
  List curr = (List) malloc(...);
  List old = curr;

  while (nondet_int()) {
    if (flag) curr->data = 1;
    else      curr->data = 2;
    curr->next = (List) malloc(...);
    curr = curr->next;
  }

  curr->data = 3;
  curr = old;
  if (flag)
    while (curr->data == 1) curr = curr->next;
  else
    while (curr->data == 2) curr = curr->next;
  assert(curr->data == 3);
}
```



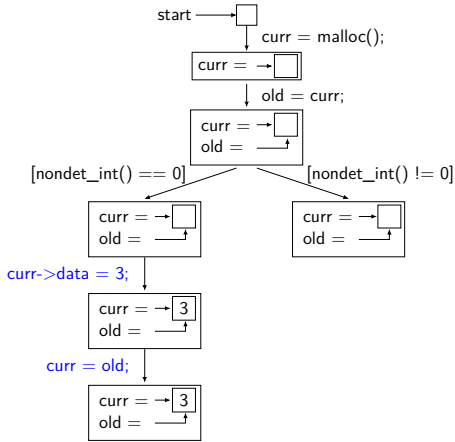
Value Analysis with SMGs - Example

```
typedef struct node {
  int data;
  struct node *next;
} *List;

void main(int flag) {
  List curr = (List) malloc(...);
  List old = curr;

  while (nondet_int()) {
    if (flag) curr->data = 1;
    else      curr->data = 2;
    curr->next = (List) malloc(...);
    curr = curr->next;
  }

  curr->data = 3;
  curr = old;
  if (flag)
    while (curr->data == 1) curr = curr->next;
  else
    while (curr->data == 2) curr = curr->next;
  assert(curr->data == 3);
}
```



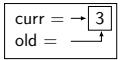
Value Analysis with SMGs - Example

```
typedef struct node {
  int data;
  struct node *next;
} *List;

void main(int flag) {
  List curr = (List) malloc(...);
  List old = curr;

  while (nondet_int()) {
    if (flag) curr->data = 1;
    else      curr->data = 2;
    curr->next = (List) malloc(...);
    curr = curr->next;
  }

  curr->data = 3;
  curr = old;
  if (flag)
    while (curr->data == 1) curr = curr->next;
  else
    while (curr->data == 2) curr = curr->next;
  assert(curr->data == 3);
}
```



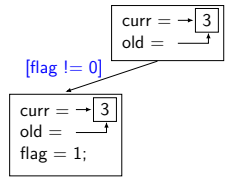
Value Analysis with SMGs - Example

```
typedef struct node {
  int data;
  struct node *next;
} *List;

void main(int flag) {
  List curr = (List) malloc(...);
  List old = curr;

  while (nondet_int()) {
    if (flag) curr->data = 1;
    else      curr->data = 2;
    curr->next = (List) malloc(...);
    curr = curr->next;
  }

  curr->data = 3;
  curr = old;
  if (flag)
    while (curr->data == 1) curr = curr->next;
  else
    while (curr->data == 2) curr = curr->next;
  assert(curr->data == 3);
}
```



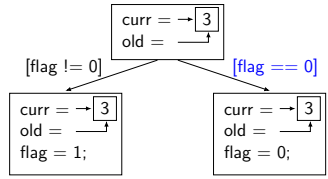
Value Analysis with SMGs - Example

```
typedef struct node {
  int data;
  struct node *next;
} *List;

void main(int flag) {
  List curr = (List) malloc(...);
  List old = curr;

  while (nondet_int()) {
    if (flag) curr->data = 1;
    else      curr->data = 2;
    curr->next = (List) malloc(...);
    curr = curr->next;
  }

  curr->data = 3;
  curr = old;
  if (flag)
    while (curr->data == 1) curr = curr->next;
  else
    while (curr->data == 2) curr = curr->next;
  assert(curr->data == 3);
}
```



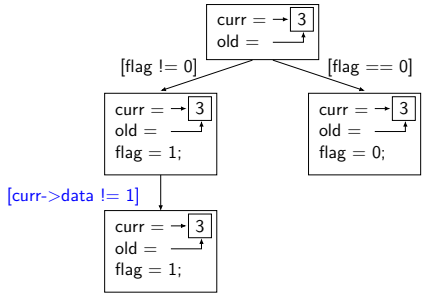
Value Analysis with SMGs - Example

```
typedef struct node {
  int data;
  struct node *next;
} *List;

void main(int flag) {
  List curr = (List) malloc(...);
  List old = curr;

  while (nondet_int()) {
    if (flag) curr->data = 1;
    else      curr->data = 2;
    curr->next = (List) malloc(...);
    curr = curr->next;
  }

  curr->data = 3;
  curr = old;
  if (flag)
    while (curr->data == 1) curr = curr->next;
  else
    while (curr->data == 2) curr = curr->next;
  assert(curr->data == 3);
}
```



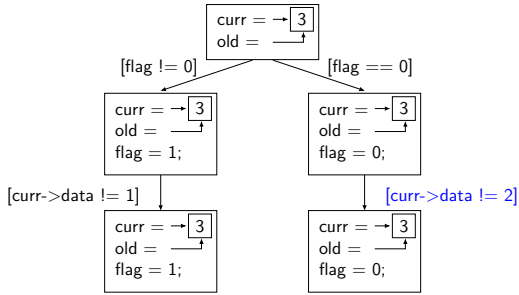
Value Analysis with SMGs - Example

```
typedef struct node {
  int data;
  struct node *next;
} *List;

void main(int flag) {
  List curr = (List) malloc(...);
  List old = curr;

  while (nondet_int()) {
    if (flag) curr->data = 1;
    else      curr->data = 2;
    curr->next = (List) malloc(...);
    curr = curr->next;
  }

  curr->data = 3;
  curr = old;
  if (flag)
    while (curr->data == 1) curr = curr->next;
  else
    while (curr->data == 2) curr = curr->next;
  assert(curr->data == 3);
}
```



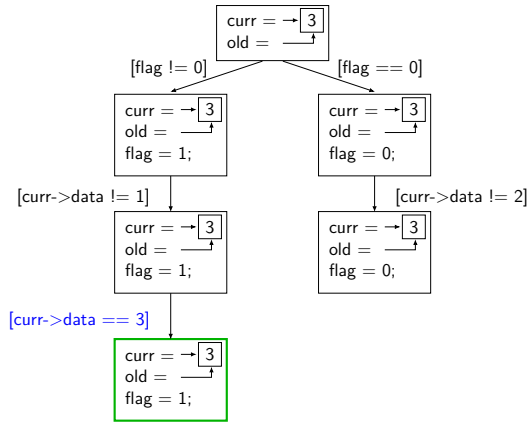
Value Analysis with SMGs - Example

```
typedef struct node {
  int data;
  struct node *next;
} *List;

void main(int flag) {
  List curr = (List) malloc(...);
  List old = curr;

  while (nondet_int()) {
    if (flag) curr->data = 1;
    else      curr->data = 2;
    curr->next = (List) malloc(...);
    curr = curr->next;
  }

  curr->data = 3;
  curr = old;
  if (flag)
    while (curr->data == 1) curr = curr->next;
  else
    while (curr->data == 2) curr = curr->next;
  assert(curr->data == 3);
}
```



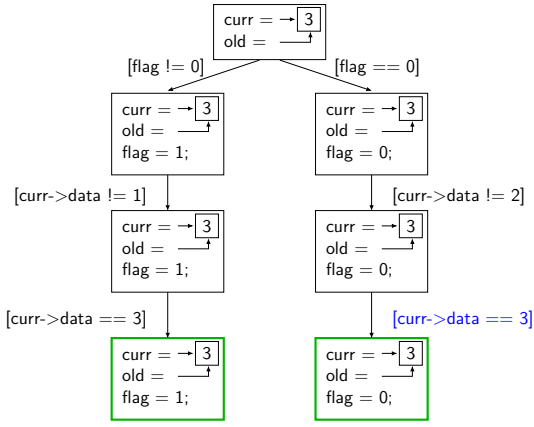
Value Analysis with SMGs - Example

```
typedef struct node {
  int data;
  struct node *next;
} *List;

void main(int flag) {
  List curr = (List) malloc(...);
  List old = curr;

  while (nondet_int()) {
    if (flag) curr->data = 1;
    else      curr->data = 2;
    curr->next = (List) malloc(...);
    curr = curr->next;
  }

  curr->data = 3;
  curr = old;
  if (flag)
    while (curr->data == 1) curr = curr->next;
  else
    while (curr->data == 2) curr = curr->next;
  assert(curr->data == 3);
}
```



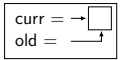
Value Analysis with SMGs - Example

```
typedef struct node {
  int data;
  struct node *next;
} *List;

void main(int flag) {
  List curr = (List) malloc(...);
  List old = curr;

  while (nondet_int()) {
    if (flag) curr->data = 1;
    else      curr->data = 2;
    curr->next = (List) malloc(...);
    curr = curr->next;
  }

  curr->data = 3;
  curr = old;
  if (flag)
    while (curr->data == 1) curr = curr->next;
  else
    while (curr->data == 2) curr = curr->next;
  assert(curr->data == 3);
}
```



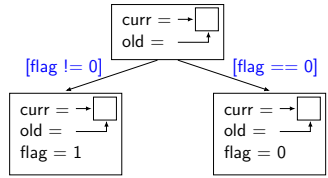
Value Analysis with SMGs - Example

```
typedef struct node {
  int data;
  struct node *next;
} *List;

void main(int flag) {
  List curr = (List) malloc(...);
  List old = curr;

  while (nondet_int()) {
    if (flag) curr->data = 1;
    else      curr->data = 2;
    curr->next = (List) malloc(...);
    curr = curr->next;
  }

  curr->data = 3;
  curr = old;
  if (flag)
    while (curr->data == 1) curr = curr->next;
  else
    while (curr->data == 2) curr = curr->next;
  assert(curr->data == 3);
}
```



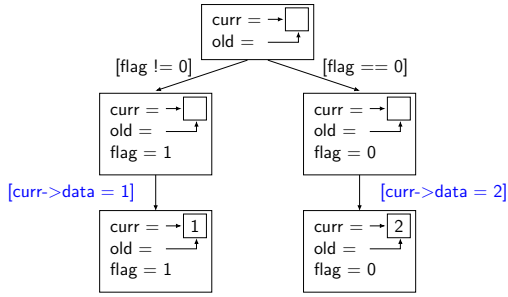
Value Analysis with SMGs - Example

```
typedef struct node {
  int data;
  struct node *next;
} *List;

void main(int flag) {
  List curr = (List) malloc(...);
  List old = curr;

  while (nondet_int()) {
    if (flag) curr->data = 1;
    else      curr->data = 2;
    curr->next = (List) malloc(...);
    curr = curr->next;
  }

  curr->data = 3;
  curr = old;
  if (flag)
    while (curr->data == 1) curr = curr->next;
  else
    while (curr->data == 2) curr = curr->next;
  assert(curr->data == 3);
}
```



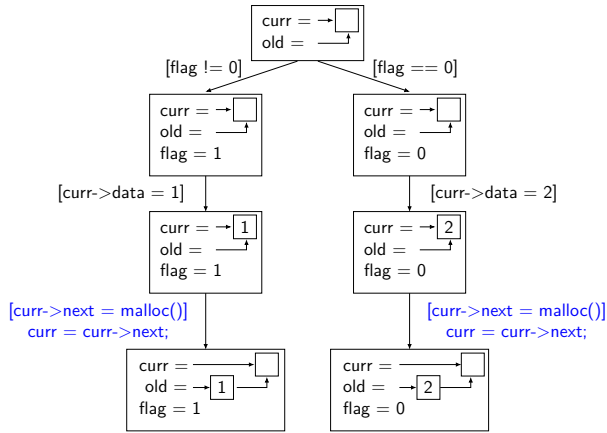
Value Analysis with SMGs - Example

```
typedef struct node {
  int data;
  struct node *next;
} *List;

void main(int flag) {
  List curr = (List) malloc(...);
  List old = curr;

  while (nondet_int()) {
    if (flag) curr->data = 1;
    else      curr->data = 2;
    curr->next = (List) malloc(...);
    curr = curr->next;
  }

  curr->data = 3;
  curr = old;
  if (flag)
    while (curr->data == 1) curr = curr->next;
  else
    while (curr->data == 2) curr = curr->next;
  assert(curr->data == 3);
}
```



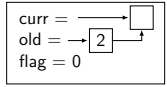
Value Analysis with SMGs - Example

```
typedef struct node {
  int data;
  struct node *next;
} *List;

void main(int flag) {
  List curr = (List) malloc(...);
  List old = curr;

  while (nondet_int()) {
    if (flag) curr->data = 1;
    else      curr->data = 2;
    curr->next = (List) malloc(...);
    curr = curr->next;
  }

  curr->data = 3;
  curr = old;
  if (flag)
    while (curr->data == 1) curr = curr->next;
  else
    while (curr->data == 2) curr = curr->next;
  assert(curr->data == 3);
}
```



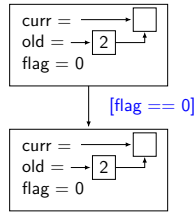
Value Analysis with SMGs - Example

```
typedef struct node {
  int data;
  struct node *next;
} *List;

void main(int flag) {
  List curr = (List) malloc(...);
  List old = curr;

  while (nondet_int()) {
    if (flag) curr->data = 1;
    else      curr->data = 2;
    curr->next = (List) malloc(...);
    curr = curr->next;
  }

  curr->data = 3;
  curr = old;
  if (flag)
    while (curr->data == 1) curr = curr->next;
  else
    while (curr->data == 2) curr = curr->next;
  assert(curr->data == 3);
}
```



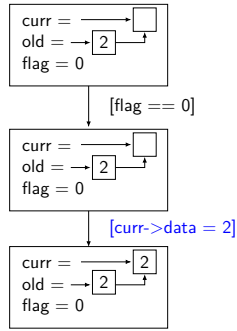
Value Analysis with SMGs - Example

```
typedef struct node {
  int data;
  struct node *next;
} *List;

void main(int flag) {
  List curr = (List) malloc(...);
  List old = curr;

  while (nondet_int()) {
    if (flag) curr->data = 1;
    else      curr->data = 2;
    curr->next = (List) malloc(...);
    curr = curr->next;
  }

  curr->data = 3;
  curr = old;
  if (flag)
    while (curr->data == 1) curr = curr->next;
  else
    while (curr->data == 2) curr = curr->next;
  assert(curr->data == 3);
}
```



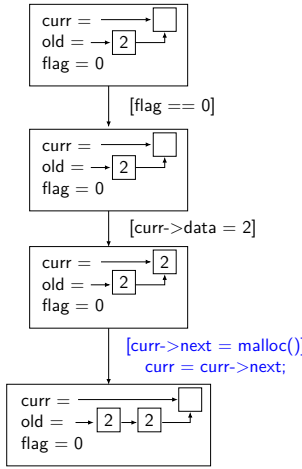
Value Analysis with SMGs - Example

```
typedef struct node {
  int data;
  struct node *next;
} *List;

void main(int flag) {
  List curr = (List) malloc(...);
  List old = curr;

  while (nondet_int()) {
    if (flag) curr->data = 1;
    else      curr->data = 2;
    curr->next = (List) malloc(...);
    curr = curr->next;
  }

  curr->data = 3;
  curr = old;
  if (flag)
    while (curr->data == 1) curr = curr->next;
  else
    while (curr->data == 2) curr = curr->next;
  assert(curr->data == 3);
}
```



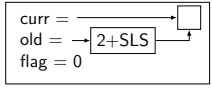
Value Analysis with SMGs - Example

```
typedef struct node {
  int data;
  struct node *next;
} *List;

void main(int flag) {
  List curr = (List) malloc(...);
  List old = curr;

  while (nondet_int()) {
    if (flag) curr->data = 1;
    else      curr->data = 2;
    curr->next = (List) malloc(...);
    curr = curr->next;
  }

  curr->data = 3;
  curr = old;
  if (flag)
    while (curr->data == 1) curr = curr->next;
  else
    while (curr->data == 2) curr = curr->next;
  assert(curr->data == 3);
}
```



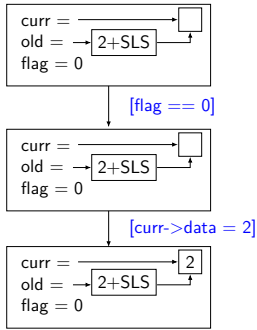
Value Analysis with SMGs - Example

```
typedef struct node {
  int data;
  struct node *next;
} *List;

void main(int flag) {
  List curr = (List) malloc(...);
  List old = curr;

  while (nondet_int()) {
    if (flag) curr->data = 1;
    else      curr->data = 2;
    curr->next = (List) malloc(...);
    curr = curr->next;
  }

  curr->data = 3;
  curr = old;
  if (flag)
    while (curr->data == 1) curr = curr->next;
  else
    while (curr->data == 2) curr = curr->next;
  assert(curr->data == 3);
}
```



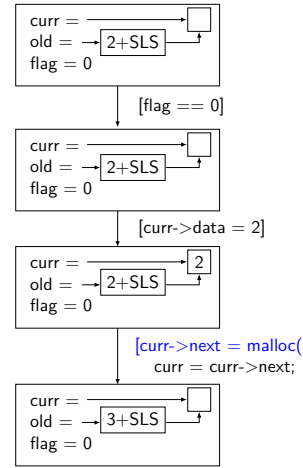
Value Analysis with SMGs - Example

```
typedef struct node {
  int data;
  struct node *next;
} *List;

void main(int flag) {
  List curr = (List) malloc(...);
  List old = curr;

  while (nondet_int()) {
    if (flag) curr->data = 1;
    else      curr->data = 2;
    curr->next = (List) malloc(...);
    curr = curr->next;
  }

  curr->data = 3;
  curr = old;
  if (flag)
    while (curr->data == 1) curr = curr->next;
  else
    while (curr->data == 2) curr = curr->next;
  assert(curr->data == 3);
}
```



Value Analysis with SMGs - Example

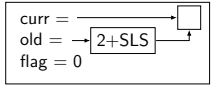
```
typedef struct node {
  int data;
  struct node *next;
} *List;

void main(int flag) {
  List curr = (List) malloc(...);
  List old = curr;

  while (nondet_int()) {
    if (flag) curr->data = 1;
    else      curr->data = 2;
    curr->next = (List) malloc(...);
    curr = curr->next;
  }

  curr->data = 3;
  curr = old;
  if (flag)
    while (curr->data == 1) curr = curr->next;
  else
    while (curr->data == 2) curr = curr->next;
  assert(curr->data == 3);
}
```



3+SLS is subsumed by 2+SLS

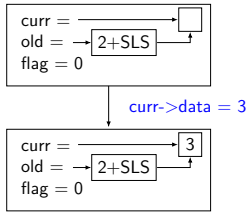
Value Analysis with SMGs - Example

```
typedef struct node {
  int data;
  struct node *next;
} *List;

void main(int flag) {
  List curr = (List) malloc(...);
  List old = curr;

  while (nondet_int()) {
    if (flag) curr->data = 1;
    else      curr->data = 2;
    curr->next = (List) malloc(...);
    curr = curr->next;
  }

  curr->data = 3;
  curr = old;
  if (flag)
    while (curr->data == 1) curr = curr->next;
  else
    while (curr->data == 2) curr = curr->next;
  assert(curr->data == 3);
}
```



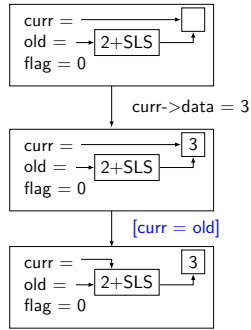
Value Analysis with SMGs - Example

```
typedef struct node {
  int data;
  struct node *next;
} *List;

void main(int flag) {
  List curr = (List) malloc(...);
  List old = curr;

  while (nondet_int()) {
    if (flag) curr->data = 1;
    else      curr->data = 2;
    curr->next = (List) malloc(...);
    curr = curr->next;
  }

  curr->data = 3;
  curr = old;
  if (flag)
    while (curr->data == 1) curr = curr->next;
  else
    while (curr->data == 2) curr = curr->next;
  assert(curr->data == 3);
}
```



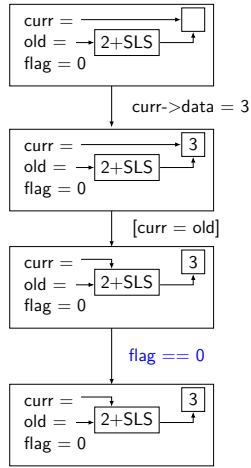
Value Analysis with SMGs - Example

```
typedef struct node {
  int data;
  struct node *next;
} *List;

void main(int flag) {
  List curr = (List) malloc(...);
  List old = curr;

  while (nondet_int()) {
    if (flag) curr->data = 1;
    else      curr->data = 2;
    curr->next = (List) malloc(...);
    curr = curr->next;
  }

  curr->data = 3;
  curr = old;
  if (flag)
    while (curr->data == 1) curr = curr->next;
  else
    while (curr->data == 2) curr = curr->next;
  assert(curr->data == 3);
}
```



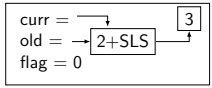
Value Analysis with SMGs - Example

```
typedef struct node {
  int data;
  struct node *next;
} *List;

void main(int flag) {
  List curr = (List) malloc(...);
  List old = curr;

  while (nondet_int()) {
    if (flag) curr->data = 1;
    else      curr->data = 2;
    curr->next = (List) malloc(...);
    curr = curr->next;
  }

  curr->data = 3;
  curr = old;
  if (flag)
    while (curr->data == 1) curr = curr->next;
  else
    while (curr->data == 2) curr = curr->next;
  assert(curr->data == 3);
}
```



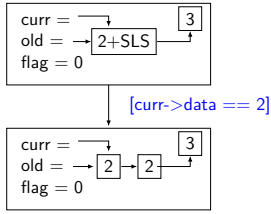
Value Analysis with SMGs - Example

```
typedef struct node {
  int data;
  struct node *next;
} *List;

void main(int flag) {
  List curr = (List) malloc(...);
  List old = curr;

  while (nondet_int()) {
    if (flag) curr->data = 1;
    else      curr->data = 2;
    curr->next = (List) malloc(...);
    curr = curr->next;
  }

  curr->data = 3;
  curr = old;
  if (flag)
    while (curr->data == 1) curr = curr->next;
  else
    while (curr->data == 2) curr = curr->next;
  assert(curr->data == 3);
}
```



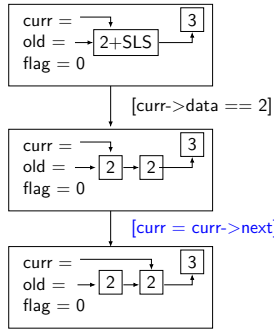
Value Analysis with SMGs - Example

```
typedef struct node {
  int data;
  struct node *next;
} *List;

void main(int flag) {
  List curr = (List) malloc(...);
  List old = curr;

  while (nondet_int()) {
    if (flag) curr->data = 1;
    else      curr->data = 2;
    curr->next = (List) malloc(...);
    curr = curr->next;
  }

  curr->data = 3;
  curr = old;
  if (flag)
    while (curr->data == 1) curr = curr->next;
  else
    while (curr->data == 2) curr = curr->next;
  assert(curr->data == 3);
}
```



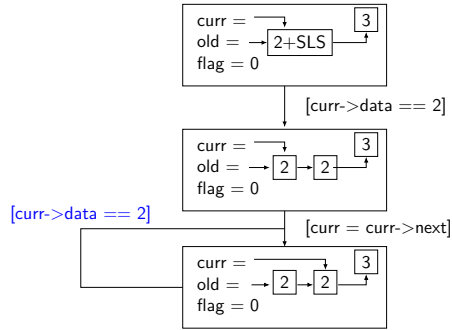
Value Analysis with SMGs - Example

```
typedef struct node {
  int data;
  struct node *next;
} *List;

void main(int flag) {
  List curr = (List) malloc(...);
  List old = curr;

  while (nondet_int()) {
    if (flag) curr->data = 1;
    else      curr->data = 2;
    curr->next = (List) malloc(...);
    curr = curr->next;
  }

  curr->data = 3;
  curr = old;
  if (flag)
    while (curr->data == 1) curr = curr->next;
  else
    while (curr->data == 2) curr = curr->next;
  assert(curr->data == 3);
}
```



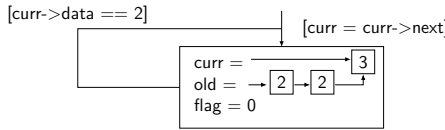
Value Analysis with SMGs - Example

```
typedef struct node {
  int data;
  struct node *next;
} *List;

void main(int flag) {
  List curr = (List) malloc(...);
  List old = curr;

  while (nondet_int()) {
    if (flag) curr->data = 1;
    else      curr->data = 2;
    curr->next = (List) malloc(...);
    curr = curr->next;
  }

  curr->data = 3;
  curr = old;
  if (flag)
    while (curr->data == 1) curr = curr->next;
  else
    while (curr->data == 2) curr = curr->next;
  assert(curr->data == 3);
}
```



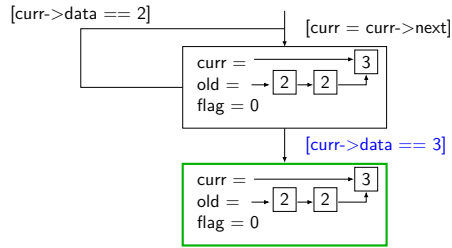
Value Analysis with SMGs - Example

```
typedef struct node {
  int data;
  struct node *next;
} *List;

void main(int flag) {
  List curr = (List) malloc(...);
  List old = curr;

  while (nondet_int()) {
    if (flag) curr->data = 1;
    else      curr->data = 2;
    curr->next = (List) malloc(...);
    curr = curr->next;
  }

  curr->data = 3;
  curr = old;
  if (flag)
    while (curr->data == 1) curr = curr->next;
  else
    while (curr->data == 2) curr = curr->next;
  assert(curr->data == 3);
}
```



Value Analysis with SMGs - Example

```
typedef struct node {
  int data;
  struct node *next;
} *List;

void main(int flag) {
  List curr = (List) malloc(...);
  List old = curr;

  while (nondet_int()) {
    if (flag) curr->data = 1;
    else      curr->data = 2;
    curr->next = (List) malloc(...);
    curr = curr->next;
  }

  curr->data = 3;
  curr = old;
  if (flag)
    while (curr->data == 1) curr = curr->next;
  else
    while (curr->data == 2) curr = curr->next;
  assert(curr->data == 3);
}
```



SAFE

Evaluation Technical Setup

Benchmark Setup - SV-COMP22

- Benchmarked using BenchExec
- Cluster of 167 machines
- Intel Xeon E3-1230 v5 CPUs
- Tasks run using Ubuntu 20.04
- Each individual task has:
  - 15GB of memory
  - 15 min of CPU time
  - 8 processing units

Evaluation Setup ReachSafety

Tasks: ReachSafety-Heap subset

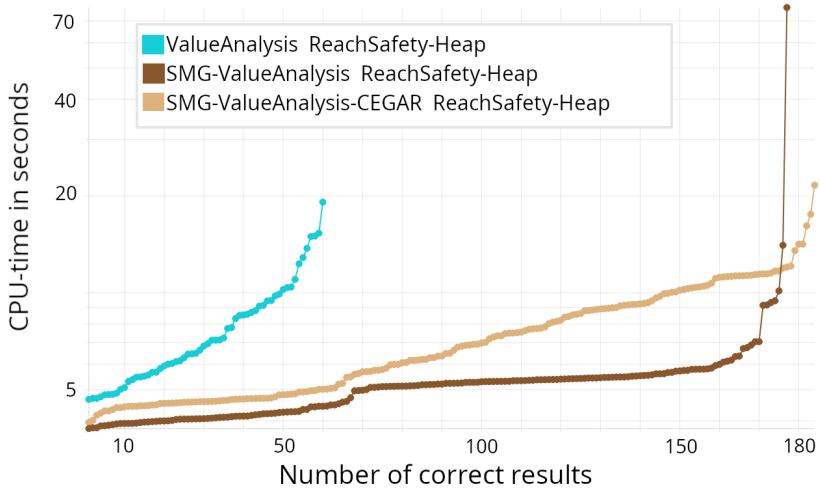
ValueAnalysis as in SV-COMP22

- CEGAR
- Counterexample-Check using Predicate Analysis

SMG-ValueAnalysis with

- Always uses list abstraction
- Two variations tested
  - No CEGAR
  - CEGAR with program variable and heap value tracking

Evaluation ReachSafety-Heap



Identified ReachSafety Issues

Most important issues for the ReachSafety category

- Overapproximation of unknown values
- Memory allocation for unknown values fails
  - CEGAR massively affected
  - Arrays affected

Evaluation Setup MemSafety

SMG-ValueAnalysis with

- List abstraction
- No CEGAR

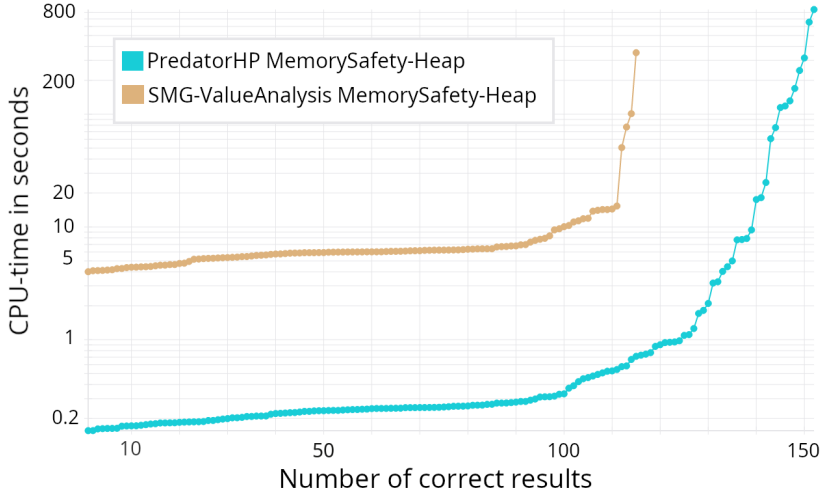
Compare against

- PredatorHP result from SV-COMP22
  - PredatorHP is a comparable memory analysis tool also using SMGs and no background solver

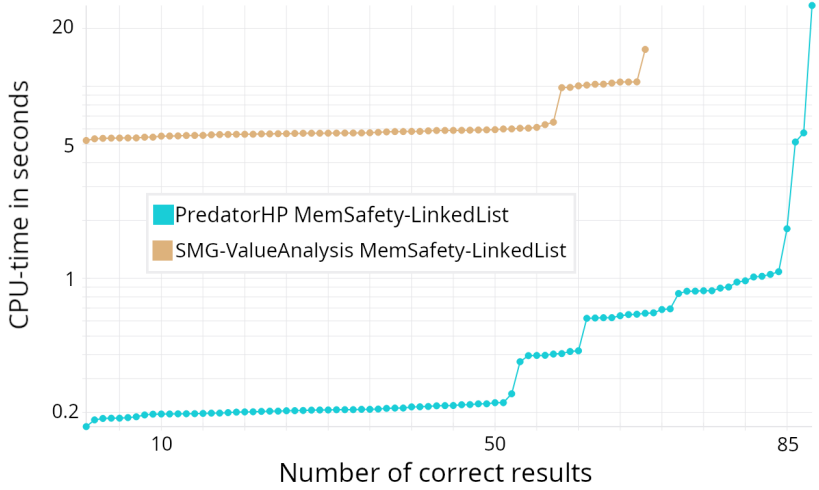
MemSafety benchmarks

- MemSafety-Heap subset
- MemSafety-LinkedList subset
- MemSafety Overall

Evaluation MemSafety-Heap



Evaluation MemSafety-LinkedList



Evaluation MemSafety Overall

	SMG-Value		
	status	cputime (s)	memory (MB)
total	3439	261000	983000
correct results	2900	21800	659000
correct true	1503	12100	347000
correct false	1397	9680	312000
incorrect results	196	1500	42300
incorrect true	121	960	26200
incorrect false	75	535	16100

Identified MemSafety Issues

Most important issues for the MemorySafety category

- Overapproximation for unknown values
- Memory allocation for unknown values fails
  - Especially arrays affected

Evaluation Results

- Improved ReachSafety results compared to Value Analysis for Heap subcategory
- Problems regarding symbolic heap held analysis back
- MemSafety performed worse than PredatorHP in Heap and LinkedList subcategories
- MemSafety performance overall good

Conclusion

Overall Results

- New analysis that combines Value Analysis and SMGs
- CEGAR for program variables and heap values
- ReachSafety and MemSafety capabilities evaluated

Future Work

Possible avenues to explore

- Symbolic memory allocation
- Array abstraction
- Block-Abstraction Memoization (BAM)

Thank you for listening!

Questions?

ReachSafety-Heap

	ValueAnalysis			SMG-Value			SMG-Value-CEGAR		
	status	cputime (s)	memory (MB)	status	cputime (s)	memory (MB)	status	cputime (s)	memory (MB)
total	241	9380	105000	241	30100	78200	241	16100	75900
correct results	60	**472**	**12200**	177	981	31700	**184**	1360	39900
correct true	27	**188**	4700	104	517	18100	**119**	918	26600
correct false	33	**284**	**7530**	**73**	464	13700	65	437	13300
incorrect results	**0**	-	-	28	145	5000	16	89	2950
incorrect true	**0**	-	-	**0**	-	-	**0**	-	-
incorrect false	**0**	-	-	28	145	5000	16	89	2950

MemSafety-Heap

	SMG-Value			PredatorHP		
	status	cputime (s)	memory (MB)	status	cputime (s)	memory (MB)
total	184	27000	78600	184	**18300**	**48700**
correct results	115	1310	31600	**152**	2900	16800
correct true	37	698	12200	**68**	2500	11400
correct false	78	612	19400	**84**	399	5360
incorrect results	28	358	7120	**0**	-	-
incorrect true	2	205	2090	**0**	-	-
incorrect false	26	153	5030	**0**	-	-

MemSafety-LinkedList

	SMG-Value			PredatorHP		
	status	cputime (s)	memory (MB)	status	cputime (s)	memory (MB)
total	103	22300	45800	103	**9090**	**21400**
correct results	68	444	13800	**88**	72	3350
correct true	42	249	7910	**62**	34	2200
correct false	**26**	195	5880	**26**	37	1150
incorrect results	8	80	2000	**0**	-	-
incorrect true	**0**	-	-	**0**	-	-
incorrect false	8	80	2000	**0**	-	-

Identified ReachSafety Issues

Most important issues for the ReachSafety category

- Overapproximation for unknown values
- Memory allocation for unknown values fails
  - CEGAR massively affected
  - Arrays affected

```
int x;
if (x > 0) {
  if (x == 0) {
    assert(x != 0);
  }
}
```

SAFE detected as UNSAFE

Identified ReachSafety Issues

Most important issues for the ReachSafety category

- Overapproximation for unknown values
- Memory allocation for unknown values fails
  - CEGAR massively affected
  - Arrays affected

```
int x;
int *memory = malloc(x);
```

FAILS

```
int x;
int array[x];
```

FAILS

Identified MemSafety Issues

- Overapproximation for unknown values
- Lists not abstractable
- Memory allocation for unknown values fails
  - Especially Arrays affected

```
int x;
int array[10];
if (x >= 0) {
  array[x] = x;
}
```

UNSAFE

Identified MemSafety Issues

- Overapproximation for unknown values
- Lists not abstractable
- Memory allocation for unknown values fails
  - Especially Arrays affected

```
long length = 2147483647;
int array[length];
for (long i = 0; i < length, i++) {
  array[i] = i;
}
```

Even worse with assumptions

Identified MemSafety Issues

- Overapproximation for unknown values
- Lists not abstractable
- Memory allocation for unknown values fails
  - Especially Arrays affected

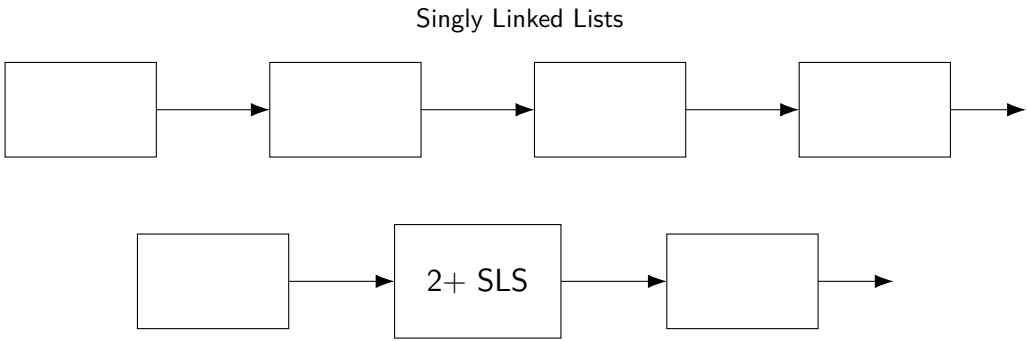
```
int length;
int array[length];
```

ABORTS

```
int length;
int *arrayPointer = malloc(length);
```

ABORTS

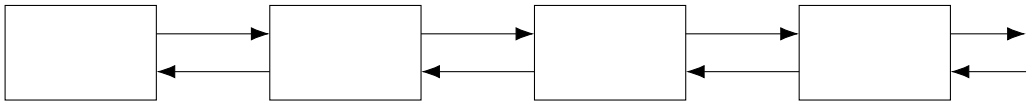
Shape Refinement for Abstracted Lists

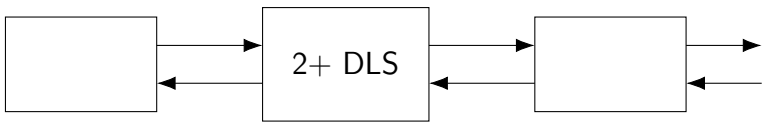


Singly Linked Lists do not have a Backwards Pointer

Shape Refinement for Abstracted Lists

Doubly Linked Lists





Doubly Linked Lists do have a Backwards Pointer