Sound and Complete Invariant-Based Heap Encodings

Zafer Esen[1] Philipp Rümmer[1,2] Tjark Weber[1]

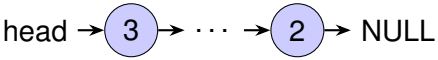
[1]Uppsala University
[2]University of Regensburg

17 September 2025
Orsay, France

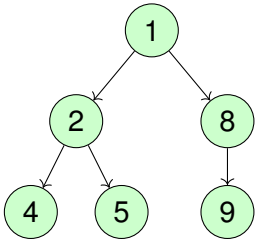
Heap-Allocated Data Structures

Programs often use dynamic data on the *heap*:

- Linked lists, trees, graphs, ...
- Accessed via pointers or references.
- *Flexible*: their structure and size can change during runtime.



A linked list



A tree

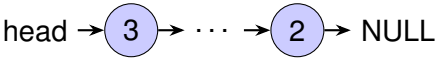
Heap-Allocated Data Structures

Programs often use dynamic data on the *heap*:

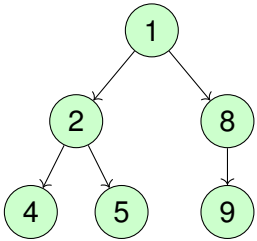
- Linked lists, trees, graphs, ...
- Accessed via pointers or references.
- *Flexible*: their structure and size can change during runtime.

Challenging for verification:

- Prone to *memory-safety errors* in languages that are not memory safe (e.g., C, C++).
- Potentially *unbounded* sizes.
- Properties to verify are often *quantified*.



A linked list



A tree

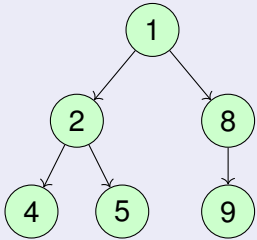
Properties of Interest

Memory Safety

- No invalid pointer accesses
- No memory leaks
- No double-freeing of memory

Functional Correctness

- *The list is sorted.*
- *The maximum value in the tree is 3.*



Relation to WG3 Objectives

- *Automatic verification* of programs with heaps
  (which requires the *automatic inference of program invariants*).

- Implement the approach in a verification tool and compare it to other approaches used in SV-COMP.

Program Transformations

Program transformation is a hot topic these days; many advocate using an interactive system to transform a problem description into an efficient program through a series of such transformations.

Program Transformations

Program transformation is a hot topic these days; many advocate using an interactive system to transform a problem description into an efficient program through a series of such transformations.

— David Gries, **The Science of Programming**, 1981

Program Transformations in This Work

Goal: Transform a program (+ specification) p into another program p' that is *easier for automatic verification*.

Program Transformations in This Work

Goal: Transform a program (+ specification) p into another program p' that is *easier for automatic verification*.

Soundness of a program transformation

p' is safe $\implies$ p is safe.
(or: p is unsafe $\implies$ p' is unsafe.)

Program Transformations in This Work

Goal: Transform a program (+ specification) p into another program p' that is *easier for automatic verification*.

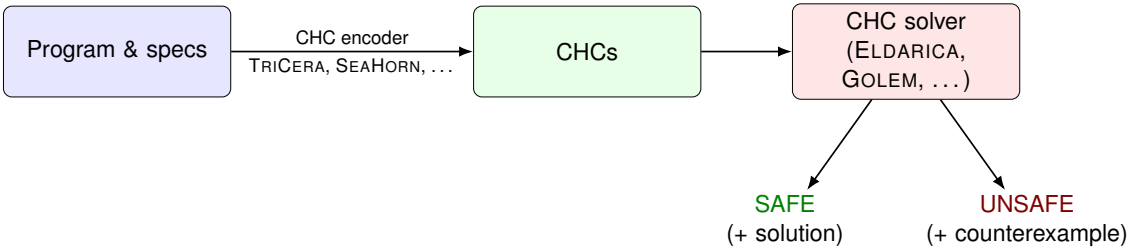
Soundness of a program transformation

p' is safe $\implies$ p is safe.
(or: p is unsafe $\implies$ p' is unsafe.)

Completeness of a program transformation

p is safe $\implies$ p' is safe.

Verification Using Constrained Horn Clauses (CHCs)



Encoding of programs using CHCs

```
1  int x = *;
2  while(x > 0){
3    x--;
4  }
5  assert(x == 0);
```

Encoding of programs using CHCs

```
1  int x = *; l1
2  while(x > 0){ l2
3    x--;
4  } l3
5  assert(x == 0);
```

Encoding of programs using CHCs

```
1  int x = *; I1
2  while(x > 0){ I2
3    x--;
4  } I3
5  assert(x == 0);
```

$$
\begin{array}{ll}
I_1(x) & \leftarrow \mathit{true} \\
I_2(x) & \leftarrow I_1(x) \wedge x > 0 \\
I_1(x-1) & \leftarrow I_2(x) \\
I_3(x) & \leftarrow I_1(x) \wedge x \not> 0 \\
\mathit{false} & \leftarrow I_3(x) \wedge x \neq 0.
\end{array}
$$

I_1, I_2, I_3 are *uninterpreted* predicates (i.e., program *invariants*).

Encoding of programs using CHCs

```
1  int x = 
2  while(x > 0
3    x--;
4
5  assert(x == 0);
```

$l_1 : true$

$l_2 : x \geq 0$

$l_3 : x = 0$

$$
\begin{array}{ll}
l_1(x) & \leftarrow true \\
l_2(x) & \leftarrow l_1(x) \wedge x > 0 \\
l_1(x-1) & \leftarrow l_2(x) \\
l_3(x) & \leftarrow l_1(x) \wedge x \not> 0 \\
false & \leftarrow l_3(x) \wedge x \neq 0.
\end{array}
$$

l_1, l_2, l_3 are *uninterpreted* predicates (i.e., program *invariants*).

A CHC solver (ELDARICA, GOLEM, SPACER, ...) tries to compute a solution...

Encoding of programs using CHCs

```
1  int x = I1 : true
2  while(x > 0)   I2 : x ≥ 0
3    x--;
4  I3 : x = 0
5  assert(x == 0);
```

$$
\begin{array}{ll}
I_1(x) & \leftarrow \mathit{true} \\
I_2(x) & \leftarrow I_1(x) \wedge x > 0 \\
I_1(x-1) & \leftarrow I_2(x) \\
I_3(x) & \leftarrow I_1(x) \wedge x \not> 0 \\
\mathit{false} & \leftarrow I_3(x) \wedge x \neq 0.
\end{array}
$$

I_1, I_2, I_3 are *uninterpreted* predicates (i.e., program *invariants*).

A CHC solver (ELDARICA, GOLEM, SPACER, ...) tries to compute a solution...

... or fails and provides a *counterexample trace* to *false*: e.g., any trace starting with $x < 0$ at I_1.

Counterexample: $\mathit{true} \rightarrow I_1(-1) \xrightarrow{x \not> 0} I_3(-1) \xrightarrow{x \neq 0} \mathit{false}$

Constrained Horn Clauses

A constrained Horn clause (CHC) in predicate logic is the formula:

$$\overbrace{H}^{\text{Head}} \leftarrow \overbrace{C \wedge B_1 \wedge ... \wedge B_n}^{\text{Body}}$$

Constrained Horn Clauses

A constrained Horn clause (CHC) in predicate logic is the formula:

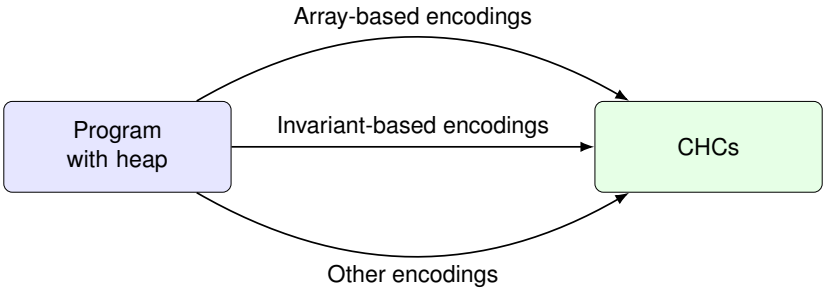
$$\overbrace{H}^{\text{Head}} \leftarrow \overbrace{C \wedge B_1 \wedge ... \wedge B_n}^{\text{Body}}$$

$\forall x.(I_2(x) \leftarrow I_1(x) \wedge x > 0)$

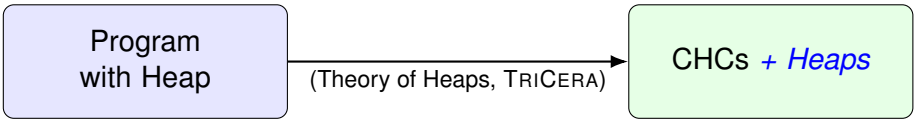
Often written in Prolog:

```
I2(x) :- I1(x), x > 0.
```

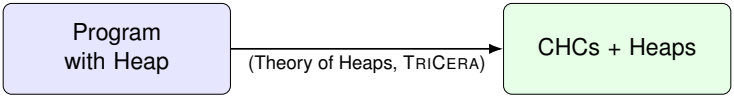
Handling Programs with Heaps



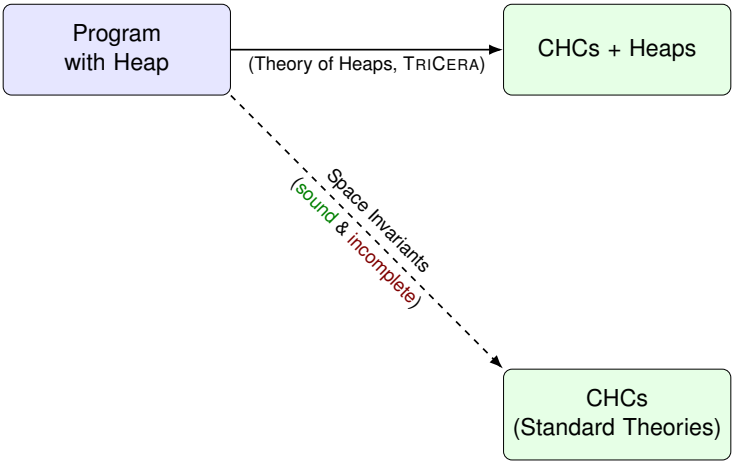
Handling Programs with Heaps Using the Theory of Heaps



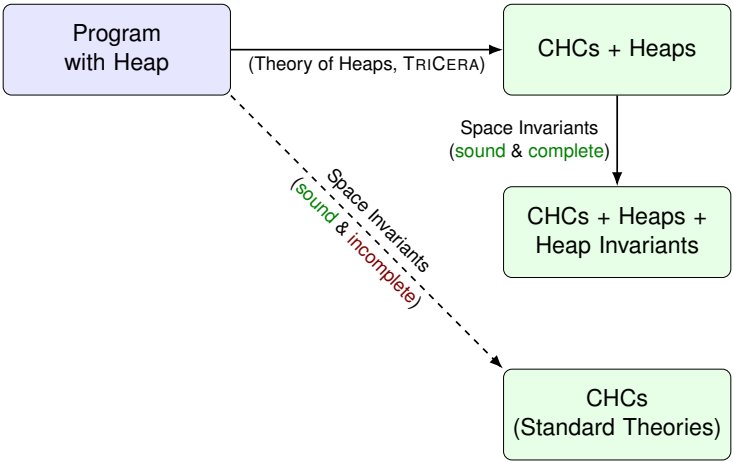
Transformations for Heap Reasoning



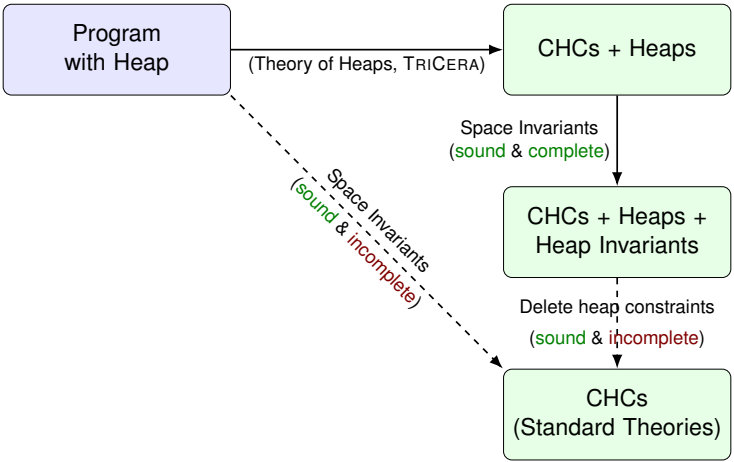
Transformations for Heap Reasoning



Transformations for Heap Reasoning



Transformations for Heap Reasoning



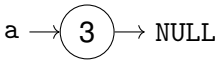
Space / Heap Invariants – An Example

```
1  typedef struct Node {
2    int data;
3    struct Node* next;
4  } Node;
5
6  void main() {
7    Node* a = NULL;
8    while (*) {
9      Node* t = new Node(3, a);
10
11     a = t;
12   }
13   while (a) {
14     Node n = *a;
15
16     assert(n.data == 3);
17     a = n.next;
18   }
19 }
```

$a \longrightarrow \mathtt{NULL}$

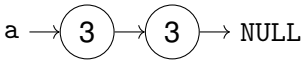
Space / Heap Invariants – An Example

```
1  typedef struct Node {
2    int data;
3    struct Node* next;
4  } Node;
5
6  void main() {
7    Node* a = NULL;
8    while (*) {
9      Node* t = new Node(3, a);
10
11     a = t;
12   }
13   while (a) {
14     Node n = *a;
15
16     assert(n.data == 3);
17     a = n.next;
18   }
19 }
```



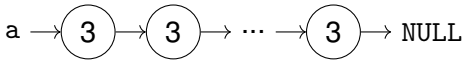
Space / Heap Invariants – An Example

```
1  typedef struct Node {
2    int data;
3    struct Node* next;
4  } Node;
5
6  void main() {
7    Node* a = NULL;
8    while (*) {
9      Node* t = new Node(3, a);
10
11     a = t;
12   }
13   while (a) {
14     Node n = *a;
15
16     assert(n.data == 3);
17     a = n.next;
18   }
19 }
```



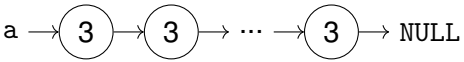
Space / Heap Invariants – An Example

```
1  typedef struct Node {
2    int data;
3    struct Node* next;
4  } Node;
5
6  void main() {
7    Node* a = NULL;
8    while (*) {
9      Node* t = new Node(3, a);
10
11     a = t;
12   }
13   while (a) {
14     Node n = *a;
15
16     assert(n.data == 3);
17     a = n.next;
18   }
19 }
```



Space / Heap Invariants – An Example

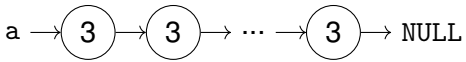
```
1  typedef struct Node {
2    int data;
3    struct Node* next;
4  } Node;
5
6  void main() {
7    Node* a = NULL;
8    while (*) {
9      Node* t = new Node(3, a);
10
11     a = t;
12   }
13   while (a) {
14     Node n = *a;
15
16     assert(n.data == 3);
17     a = n.next;
18   }
19 }
```



Invariant:
Every `Node` reachable from `a` contains 3.

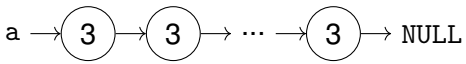
Space / Heap Invariants – An Example

```
1  typedef struct Node {
2    int data;
3    struct Node* next;
4  } Node;
5
6  void main() {
7    Node* a = NULL;
8    while (*) {
9      Node* t = new Node(3, a);
10     assert(I(t, Node(3, a)));
11     a = t;
12   }
13   while (a) {
14     Node n = *a;
15     assume(I(a, n));
16     assert(n.data == 3);
17     a = n.next;
18   }
19 }
```



Space / Heap Invariants – An Example

```
1  typedef struct Node {
2    int data;
3    struct Node* next;
4  } Node;
5
6  void main() {
7    Node* a = NULL;
8    while (*) {
9      Node* t = new Node(3, a);
10     assert(t->data == 3);
11     a = t;
12   }
13   while (a) {
14     Node n = *a;
15     assume(n.data == 3);
16     assert(n.data == 3);
17     a = n.next;
18   }
19 }
```

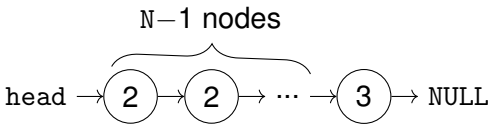


Invariant $I(a, n) \equiv data(n) = 3$

A more challenging example

Based on the SV-COMP benchmark `simple_and_skiplist_2lvl-1.c`

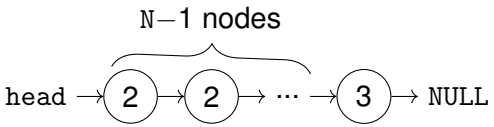
```
1  void main(int N) {
2    Node *head = malloc(sizeof(Node));
3    Node *cur = head;
4
5    for (int i = 0; i < N; i++) {
6      cur->next = malloc(sizeof(Node));
7      cur->data = 2;
8      cur = cur->next;
9    }
10   cur->data = 3; cur->next = NULL;
11
12   cur = head;
13   while (cur != NULL) {
14     if(cur->next != NULL)
15       assert(cur->data == 2);
16     else
17       assert(cur->data == 3);
18     cur = cur->next;
19   }
20 }
```



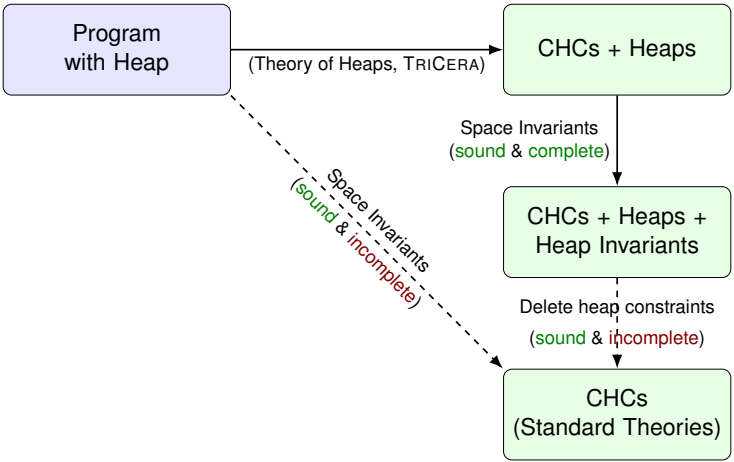
A more challenging example

Based on the SV-COMP benchmark `simple_and_skiplist_2lvl-1.c`

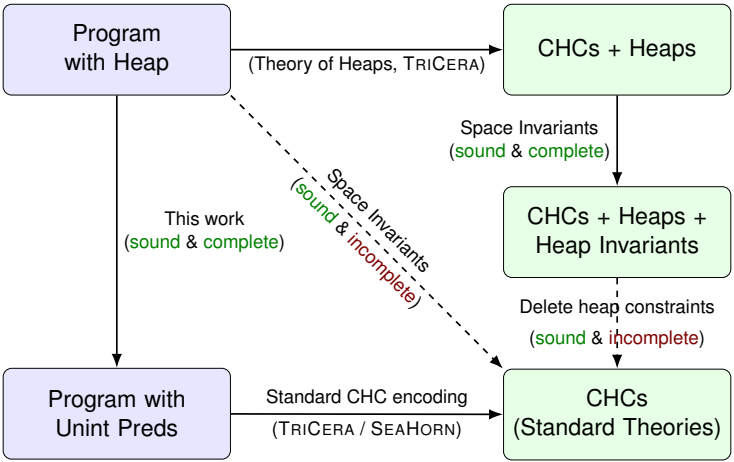
```
1  void main(int N) {
2    Node *head = malloc(sizeof(Node));
3    Node *cur = head;
4
5    for (int i = 0; i < N; i++) {
6      cur->next = malloc(sizeof(Node));
7      cur->data = 2;
8      cur = cur->next;
9    }
10   cur->data = 3; cur->next = NULL;
11
12   cur = head;
13   while (cur != NULL) {
14     if(cur->next != NULL)
15       assert(cur->data == 2);
16     else
17       assert(cur->data == 3);
18     cur = cur->next;
19   }
20 }
```



Sound and Complete Invariant-Based Heap Encodings



Sound and Complete Invariant-Based Heap Encodings



Example: A Heap Trace

```
1  void main(int N) {
2    Node *head = malloc(sizeof(Node));
3    Node *cur = head;
4
5    for (int i = 0; i < N; i++) {
6      cur->next = malloc(sizeof(Node));
7      cur->data = 2;
8      cur = cur->next;
9    }
10   cur->data = 3;
11   cur->next = NULL;
12
13   cur = head;
14   while (cur != NULL) {
15     Node n = *cur;
16     if(n.next != NULL)
17       assert(n.data == 2);
18     else
19       assert(n.data == 3);
20     cur = n.next;
21   }
22 }
```

Name	Value
head	*
cur	*
i	*
N	3

.

Example: A Heap Trace

```
1  void main(int N) {
2    Node *head = malloc(sizeof(Node));
3    Node *cur = head;
4
5    for (int i = 0; i < N; i++) {
6      cur->next = malloc(sizeof(Node));
7      cur->data = 2;
8      cur = cur->next;
9    }
10   cur->data = 3;
11   cur->next = NULL;
12
13   cur = head;
14   while (cur != NULL) {
15     Node n = *cur;
16     if(n.next != NULL)
17       assert(n.data == 2);
18     else
19       assert(n.data == 3);
20     cur = n.next;
21   }
22 }
```

Name	Value
head	1
cur	*
i	*
N	3

Example: A Heap Trace

```
1  void main(int N) {
2    Node *head = malloc(sizeof(Node));
3    Node *cur = head;
4
5    for (int i = 0; i < N; i++) {
6      cur->next = malloc(sizeof(Node));
7      cur->data = 2;
8      cur = cur->next;
9    }
10   cur->data = 3;
11   cur->next = NULL;
12
13   cur = head;
14   while (cur != NULL) {
15     Node n = *cur;
16     if(n.next != NULL)
17       assert(n.data == 2);
18     else
19       assert(n.data == 3);
20     cur = n.next;
21   }
22 }
```

Name	Value
head	1
cur	1
i	*
N	3

Example: A Heap Trace

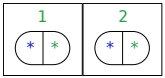
```
1  void main(int N) {
2    Node *head = malloc(sizeof(Node));
3    Node *cur = head;
4
5    for (int i = 0; i < N; i++) {
6      cur->next = malloc(sizeof(Node));
7      cur->data = 2;
8      cur = cur->next;
9    }
10   cur->data = 3;
11   cur->next = NULL;
12
13   cur = head;
14   while (cur != NULL) {
15     Node n = *cur;
16     if(n.next != NULL)
17       assert(n.data == 2);
18     else
19       assert(n.data == 3);
20     cur = n.next;
21   }
22 }
```

Name	Value
head	1
cur	1
i	0
N	3

Example: A Heap Trace

```
1  void main(int N) {
2    Node *head = malloc(sizeof(Node));
3    Node *cur = head;
4
5    for (int i = 0; i < N; i++) {
6      cur->next = malloc(sizeof(Node));
7      cur->data = 2;
8      cur = cur->next;
9    }
10   cur->data = 3;
11   cur->next = NULL;
12
13   cur = head;
14   while (cur != NULL) {
15     Node n = *cur;
16     if(n.next != NULL)
17       assert(n.data == 2);
18     else
19       assert(n.data == 3);
20     cur = n.next;
21   }
22 }
```

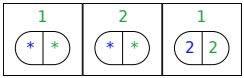
Name	Value
head	1
cur	1
i	0
N	3



Example: A Heap Trace

```
1  void main(int N) {
2    Node *head = malloc(sizeof(Node));
3    Node *cur = head;
4
5    for (int i = 0; i < N; i++) {
6      cur->next = malloc(sizeof(Node));
7      cur->data = 2;
8      cur = cur->next;
9    }
10   cur->data = 3;
11   cur->next = NULL;
12
13   cur = head;
14   while (cur != NULL) {
15     Node n = *cur;
16     if(n.next != NULL)
17       assert(n.data == 2);
18     else
19       assert(n.data == 3);
20     cur = n.next;
21   }
22 }
```

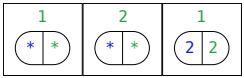
Name	Value
head	1
cur	1
i	0
N	3



Example: A Heap Trace

```
1  void main(int N) {
2    Node *head = malloc(sizeof(Node));
3    Node *cur = head;
4
5    for (int i = 0; i < N; i++) {
6      cur->next = malloc(sizeof(Node));
7      cur->data = 2;
8      cur = cur->next;
9    }
10   cur->data = 3;
11   cur->next = NULL;
12
13   cur = head;
14   while (cur != NULL) {
15     Node n = *cur;
16     if(n.next != NULL)
17       assert(n.data == 2);
18     else
19       assert(n.data == 3);
20     cur = n.next;
21   }
22 }
```

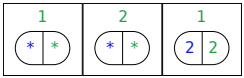
Name	Value
head	1
cur	2
i	0
N	3



Example: A Heap Trace

```
1  void main(int N) {
2    Node *head = malloc(sizeof(Node));
3    Node *cur = head;
4
5    for (int i = 0; i < N; i++) {
6      cur->next = malloc(sizeof(Node));
7      cur->data = 2;
8      cur = cur->next;
9    }
10   cur->data = 3;
11   cur->next = NULL;
12
13   cur = head;
14   while (cur != NULL) {
15     Node n = *cur;
16     if(n.next != NULL)
17       assert(n.data == 2);
18     else
19       assert(n.data == 3);
20     cur = n.next;
21   }
22 }
```

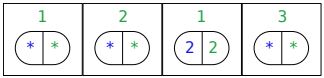
Name	Value
head	1
cur	2
i	1
N	3



Example: A Heap Trace

```
1  void main(int N) {
2    Node *head = malloc(sizeof(Node));
3    Node *cur = head;
4
5    for (int i = 0; i < N; i++) {
6      cur->next = malloc(sizeof(Node));
7      cur->data = 2;
8      cur = cur->next;
9    }
10   cur->data = 3;
11   cur->next = NULL;
12
13   cur = head;
14   while (cur != NULL) {
15     Node n = *cur;
16     if(n.next != NULL)
17       assert(n.data == 2);
18     else
19       assert(n.data == 3);
20     cur = n.next;
21   }
22 }
```

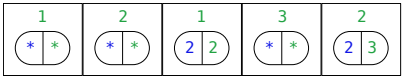
Name	Value
head	1
cur	2
i	1
N	3



Example: A Heap Trace

```
1  void main(int N) {
2    Node *head = malloc(sizeof(Node));
3    Node *cur = head;
4
5    for (int i = 0; i < N; i++) {
6      cur->next = malloc(sizeof(Node));
7      cur->data = 2;
8      cur = cur->next;
9    }
10   cur->data = 3;
11   cur->next = NULL;
12
13   cur = head;
14   while (cur != NULL) {
15     Node n = *cur;
16     if(n.next != NULL)
17       assert(n.data == 2);
18     else
19       assert(n.data == 3);
20     cur = n.next;
21   }
22 }
```

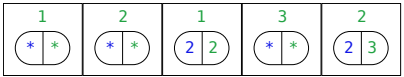
Name	Value
head	1
cur	2
i	1
N	3



Example: A Heap Trace

```
1  void main(int N) {
2    Node *head = malloc(sizeof(Node));
3    Node *cur = head;
4
5    for (int i = 0; i < N; i++) {
6      cur->next = malloc(sizeof(Node));
7      cur->data = 2;
8      cur = cur->next;
9    }
10   cur->data = 3;
11   cur->next = NULL;
12
13   cur = head;
14   while (cur != NULL) {
15     Node n = *cur;
16     if(n.next != NULL)
17       assert(n.data == 2);
18     else
19       assert(n.data == 3);
20     cur = n.next;
21   }
22 }
```

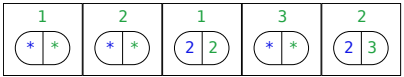
Name	Value
head	1
cur	3
i	1
N	3



Example: A Heap Trace

```
1  void main(int N) {
2    Node *head = malloc(sizeof(Node));
3    Node *cur = head;
4
5    for (int i = 0; i < N; i++) {
6      cur->next = malloc(sizeof(Node));
7      cur->data = 2;
8      cur = cur->next;
9    }
10   cur->data = 3;
11   cur->next = NULL;
12
13   cur = head;
14   while (cur != NULL) {
15     Node n = *cur;
16     if(n.next != NULL)
17       assert(n.data == 2);
18     else
19       assert(n.data == 3);
20     cur = n.next;
21   }
22 }
```

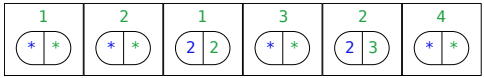
Name	Value
head	1
cur	3
i	2
N	3



Example: A Heap Trace

```
1  void main(int N) {
2    Node *head = malloc(sizeof(Node));
3    Node *cur = head;
4
5    for (int i = 0; i < N; i++) {
6      cur->next = malloc(sizeof(Node));
7      cur->data = 2;
8      cur = cur->next;
9    }
10   cur->data = 3;
11   cur->next = NULL;
12
13   cur = head;
14   while (cur != NULL) {
15     Node n = *cur;
16     if(n.next != NULL)
17       assert(n.data == 2);
18     else
19       assert(n.data == 3);
20     cur = n.next;
21   }
22 }
```

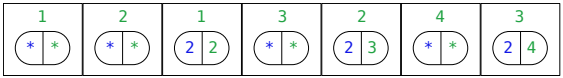
Name	Value
head	1
cur	3
i	2
N	3



Example: A Heap Trace

```
1  void main(int N) {
2    Node *head = malloc(sizeof(Node));
3    Node *cur = head;
4
5    for (int i = 0; i < N; i++) {
6      cur->next = malloc(sizeof(Node));
7      cur->data = 2;
8      cur = cur->next;
9    }
10   cur->data = 3;
11   cur->next = NULL;
12
13   cur = head;
14   while (cur != NULL) {
15     Node n = *cur;
16     if(n.next != NULL)
17       assert(n.data == 2);
18     else
19       assert(n.data == 3);
20     cur = n.next;
21   }
22 }
```

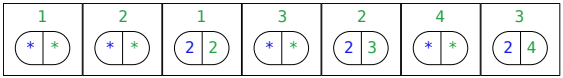
Name	Value
head	1
cur	3
i	2
N	3



Example: A Heap Trace

```
1  void main(int N) {
2    Node *head = malloc(sizeof(Node));
3    Node *cur = head;
4
5    for (int i = 0; i < N; i++) {
6      cur->next = malloc(sizeof(Node));
7      cur->data = 2;
8      cur = cur->next;
9    }
10   cur->data = 3;
11   cur->next = NULL;
12
13   cur = head;
14   while (cur != NULL) {
15     Node n = *cur;
16     if(n.next != NULL)
17       assert(n.data == 2);
18     else
19       assert(n.data == 3);
20     cur = n.next;
21   }
22 }
```

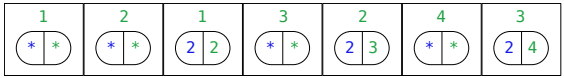
Name	Value
head	1
cur	4
i	2
N	3



Example: A Heap Trace

```
1  void main(int N) {
2    Node *head = malloc(sizeof(Node));
3    Node *cur = head;
4
5    for (int i = 0; i < N; i++) {
6      cur->next = malloc(sizeof(Node));
7      cur->data = 2;
8      cur = cur->next;
9    }
10   cur->data = 3;
11   cur->next = NULL;
12
13   cur = head;
14   while (cur != NULL) {
15     Node n = *cur;
16     if(n.next != NULL)
17       assert(n.data == 2);
18     else
19       assert(n.data == 3);
20     cur = n.next;
21   }
22 }
```

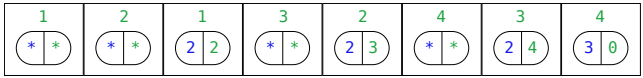
Name	Value
head	1
cur	4
i	3
N	3



Example: A Heap Trace

```
1  void main(int N) {
2    Node *head = malloc(sizeof(Node));
3    Node *cur = head;
4
5    for (int i = 0; i < N; i++) {
6      cur->next = malloc(sizeof(Node));
7      cur->data = 2;
8      cur = cur->next;
9    }
10   cur->data = 3;
11   cur->next = NULL;
12
13   cur = head;
14   while (cur != NULL) {
15     Node n = *cur;
16     if(n.next != NULL)
17       assert(n.data == 2);
18     else
19       assert(n.data == 3);
20     cur = n.next;
21   }
22 }
```

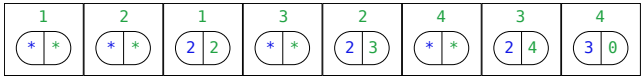
Name	Value
head	1
cur	4
i	3
N	3



Example: A Heap Trace

```
1  void main(int N) {
2    Node *head = malloc(sizeof(Node));
3    Node *cur = head;
4
5    for (int i = 0; i < N; i++) {
6      cur->next = malloc(sizeof(Node));
7      cur->data = 2;
8      cur = cur->next;
9    }
10   cur->data = 3;
11   cur->next = NULL;
12
13   cur = head;
14   while (cur != NULL) {
15     Node n = *cur;
16     if(n.next != NULL)
17       assert(n.data == 2);
18     else
19       assert(n.data == 3);
20     cur = n.next;
21   }
22 }
```

Name	Value
head	1
cur	1
i	3
N	3



First step: Normalization

```
1  void main(int in) {
2    Node* a = NULL;
3    while (in --> 0) {
4      Node* t = new Node(3, a);
5
6      a = t;
7    }
8    while (a) {
9      Node n = *a;
10     assert(n.data == 3);
11     a = n.next;
12   }
13 }
```

$\Rightarrow$

```
1  void main(int in) {
2    int a = 0;
3    while (in --> 0) {
4      int t = alloc();
5      write(t, Node(3, a));
6      a = t;
7    }
8    while (a) {
9      Node n = read(a);
10     assert(n.data == 3);
11     a = n.next;
12   }
13 }
```

Second step: Inroduce auxiliary variables

```
1
2
3
4
5
6
7  void main(int in) {
8
9    int a = 0;
10   while (in --> 0) {
11     int t = alloc();
12     write(t, Node(3, a));
13     a = t;
14   }
15   while (a) {
16     Node n = read(a);
17     assert(n.data == 3);
18     a = n.next;
19   }
20 }
```

⇒

```
1  unsigned int cnt = 0;
2  unsigned int cnt_alloc = 0;
3  Node last = *;
4  int last_addr = *;
5  int inG;
6
7  void main(int in) {
8    inG = in;
9    int a = 0;
10   while (in --> 0) {
11     int t = alloc();
12     write(t, Node(3, a));
13     a = t;
14   }
15   while (a) {
16     Node n = read(a);
17     assert(n.data == 3);
18     a = n.next;
19   }
20 }
```

Last step: Define `read` and `write` (*R*-Encoding)

```
1  unsigned int cnt = 0;
2  unsigned int cnt_alloc = 0;
3  Node last = *;
4  int last_addr = *;
5  int inG;
6
7  void main(int in) {
8    inG = in;
9    int a = 0;
10   while (in —> 0) {
11     int t = alloc();
12     write(t, Node(3, a));
13     a = t;
14   }
15   while (a) {
16     Node n = read(a);
17     assert(n.data == 3);
18     a = n.next;
19   }
20 }
```

```
20 R(int in, int cnt_r, Node n);
21
22 Node read(int p) {
23   Node result;
24   ++cnt;
25   if (last_addr == p) {
26     assert(R(inG, cnt, last));
27     result = last;
28   } else {
29     result = *;
30     assume(R(inG, cnt, result));
31   }
32   return result;
33 }
34
35 void write(int p, Node v) {
36   if (last_addr == p &&
37       0 < p <= cnt_alloc)
38     last = v;
39 }
```

Example: *R*-Encoding Execution (last_addr = 1)

```
1  void main(int N) {
2    int head = alloc();
3    int cur = head;
4
5    for (int i = 0; i < N; i++) {
6      writeNode(cur, 2, alloc());
7      Node n = read(cur);
8      cur = n.next;
9    }
10   writeNode(cur, 3, 0);
11
12   cur = head;
13   while (cur != 0) {
14     Node n = read(cur);
15     if(n.next != 0)
16       assert(n.data == 2);
17     else
18       assert(n.data == 3);
19     cur = n.next;
20   }
21 }
```

Name	Value
head	*
cur	*
i	*
N	3
last	*
last_addr	1
R	{}

.

Example: *R*-Encoding Execution (last_addr = 1)

```
1  void main(int N) {
2    int head = alloc();
3    int cur = head;
4
5    for (int i = 0; i < N; i++) {
6      writeNode(cur, 2, alloc());
7      Node n = read(cur);
8      cur = n.next;
9    }
10   writeNode(cur, 3, 0);
11
12   cur = head;
13   while (cur != 0) {
14     Node n = read(cur);
15     if(n.next != 0)
16       assert(n.data == 2);
17     else
18       assert(n.data == 3);
19     cur = n.next;
20   }
21 }
```

Name	Value
head	1
cur	*
i	*
N	3
last	Node(*,*)
last_addr	1
R	{}

Example: *R*-Encoding Execution (last_addr = 1)

```
1  void main(int N) {
2    int head = alloc();
3    int cur = head;
4
5    for (int i = 0; i < N; i++) {
6      writeNode(cur, 2, alloc());
7      Node n = read(cur);
8      cur = n.next;
9    }
10   writeNode(cur, 3, 0);
11
12   cur = head;
13   while (cur != 0) {
14     Node n = read(cur);
15     if(n.next != 0)
16       assert(n.data == 2);
17     else
18       assert(n.data == 3);
19     cur = n.next;
20   }
21 }
```

Name	Value
head	1
cur	1
i	*
N	3
last	Node(*,*)
last_addr	1
R	{}

Example: *R*-Encoding Execution (last_addr = 1)

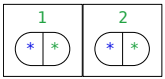
```
1  void main(int N) {
2    int head = alloc();
3    int cur = head;
4
5    for (int i = 0; i < N; i++) {
6      writeNode(cur, 2, alloc());
7      Node n = read(cur);
8      cur = n.next;
9    }
10   writeNode(cur, 3, 0);
11
12   cur = head;
13   while (cur != 0) {
14     Node n = read(cur);
15     if(n.next != 0)
16       assert(n.data == 2);
17     else
18       assert(n.data == 3);
19     cur = n.next;
20   }
21 }
```

Name	Value
head	1
cur	1
i	0
N	3
last	Node(*,*)
last_addr	1
R	{}

Example: *R*-Encoding Execution (last_addr = 1)

```
1  void main(int N) {
2    int head = alloc();
3    int cur = head;
4
5    for (int i = 0; i < N; i++) {
6      writeNode(cur, 2, alloc());
7      Node n = read(cur);
8      cur = n.next;
9    }
10   writeNode(cur, 3, 0);
11
12   cur = head;
13   while (cur != 0) {
14     Node n = read(cur);
15     if(n.next != 0)
16       assert(n.data == 2);
17     else
18       assert(n.data == 3);
19     cur = n.next;
20   }
21 }
```

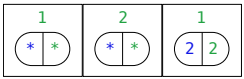
Name	Value
head	1
cur	1
i	0
N	3
last	Node(*,*)
last_addr	1
R	{}



Example: *R*-Encoding Execution (last_addr = 1)

```
1  void main(int N) {
2    int head = alloc();
3    int cur = head;
4
5    for (int i = 0; i < N; i++) {
6      writeNode(cur, 2, alloc());
7      Node n = read(cur);
8      cur = n.next;
9    }
10   writeNode(cur, 3, 0);
11
12   cur = head;
13   while (cur != 0) {
14     Node n = read(cur);
15     if(n.next != 0)
16       assert(n.data == 2);
17     else
18       assert(n.data == 3);
19     cur = n.next;
20   }
21 }
```

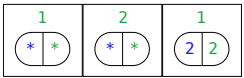
Name	Value
head	1
cur	1
i	0
N	3
last	Node(2,2)
last_addr	1
R	{}



Example: *R*-Encoding Execution (last_addr = 1)

```
1  void main(int N) {
2    int head = alloc();
3    int cur = head;
4
5    for (int i = 0; i < N; i++) {
6      writeNode(cur, 2, alloc());
7      Node n = read(cur);
8      cur = n.next;
9    }
10   writeNode(cur, 3, 0);
11
12   cur = head;
13   while (cur != 0) {
14     Node n = read(cur);
15     if(n.next != 0)
16       assert(n.data == 2);
17     else
18       assert(n.data == 3);
19     cur = n.next;
20   }
21 }
```

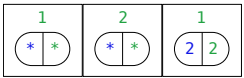
Name	Value
head	1
cur	1
i	0
N	3
last	Node(2,2)
last_addr	1
R	{(1, Node(2,2))}



Example: *R*-Encoding Execution (last_addr = 1)

```
1  void main(int N) {
2    int head = alloc();
3    int cur = head;
4
5    for (int i = 0; i < N; i++) {
6      writeNode(cur, 2, alloc());
7      Node n = read(cur);
8      cur = n.next;
9    }
10   writeNode(cur, 3, 0);
11
12   cur = head;
13   while (cur != 0) {
14     Node n = read(cur);
15     if(n.next != 0)
16       assert(n.data == 2);
17     else
18       assert(n.data == 3);
19     cur = n.next;
20   }
21 }
```

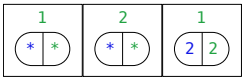
Name	Value
head	1
cur	2
i	0
N	3
last	Node(2,2)
last_addr	1
R	{(1, Node(2,2))}



Example: *R*-Encoding Execution (last_addr = 1)

```
1  void main(int N) {
2    int head = alloc();
3    int cur = head;
4
5    for (int i = 0; i < N; i++) {
6      writeNode(cur, 2, alloc());
7      Node n = read(cur);
8      cur = n.next;
9    }
10   writeNode(cur, 3, 0);
11
12   cur = head;
13   while (cur != 0) {
14     Node n = read(cur);
15     if(n.next != 0)
16       assert(n.data == 2);
17     else
18       assert(n.data == 3);
19     cur = n.next;
20   }
21 }
```

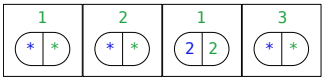
Name	Value
head	1
cur	2
i	1
N	3
last	Node(2,2)
last_addr	1
R	{(1, Node(2,2))}



Example: *R*-Encoding Execution (last_addr = 1)

```
1  void main(int N) {
2    int head = alloc();
3    int cur = head;
4
5    for (int i = 0; i < N; i++) {
6      writeNode(cur, 2, alloc());
7      Node n = read(cur);
8      cur = n.next;
9    }
10   writeNode(cur, 3, 0);
11
12   cur = head;
13   while (cur != 0) {
14     Node n = read(cur);
15     if(n.next != 0)
16       assert(n.data == 2);
17     else
18       assert(n.data == 3);
19     cur = n.next;
20   }
21 }
```

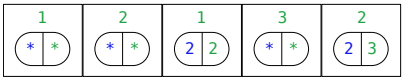
Name	Value
head	1
cur	2
i	1
N	3
last	Node(2,2)
last_addr	1
R	{(1, Node(2,2))}



Example: *R*-Encoding Execution (last_addr = 1)

```
1  void main(int N) {
2    int head = alloc();
3    int cur = head;
4
5    for (int i = 0; i < N; i++) {
6      writeNode(cur, 2, alloc());
7      Node n = read(cur);
8      cur = n.next;
9    }
10   writeNode(cur, 3, 0);
11
12   cur = head;
13   while (cur != 0) {
14     Node n = read(cur);
15     if(n.next != 0)
16       assert(n.data == 2);
17     else
18       assert(n.data == 3);
19     cur = n.next;
20   }
21 }
```

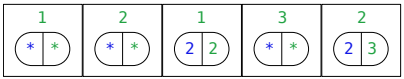
Name	Value
head	1
cur	2
i	1
N	3
last	Node(2,2)
last_addr	1
R	{(1, Node(2,2))}



Example: *R*-Encoding Execution (last_addr = 1)

```
1  void main(int N) {
2    int head = alloc();
3    int cur = head;
4
5    for (int i = 0; i < N; i++) {
6      writeNode(cur, 2, alloc());
7      Node n = read(cur);
8      cur = n.next;
9    }
10   writeNode(cur, 3, 0);
11
12   cur = head;
13   while (cur != 0) {
14     Node n = read(cur);
15     if(n.next != 0)
16       assert(n.data == 2);
17     else
18       assert(n.data == 3);
19     cur = n.next;
20   }
21 }
```

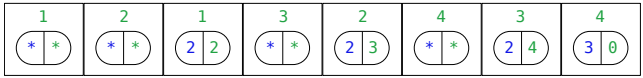
Name	Value
head	1
cur	2
i	1
N	3
last	Node(2,2)
last_addr	1
R	{(1, Node(2,2))}



Example: *R*-Encoding Execution (last_addr = 1)

```
1  void main(int N) {
2    int head = alloc();
3    int cur = head;
4
5    for (int i = 0; i < N; i++) {
6      writeNode(cur, 2, alloc());
7      Node n = read(cur);
8      cur = n.next;
9    }
10   writeNode(cur, 3, 0);
11
12   cur = head;
13   while (cur != 0) {
14     Node n = read(cur);
15     if(n.next != 0)
16       assert(n.data == 2);
17     else
18       assert(n.data == 3);
19     cur = n.next;
20   }
21 }
```

Name	Value
head	1
cur	1
i	3
N	3
last	Node(2,2)
last_addr	1
R	{(1, Node(2,2))}



Meanwhile, in another execution. . . (last_addr = 2)

```
1  void main(int N) {
2    int head = alloc();
3    int cur = head;
4
5    for (int i = 0; i < N; i++) {
6      writeNode(cur, 2, alloc());
7      Node n = read(cur);
8      cur = n.next;
9    }
10   writeNode(cur, 3, 0);
11
12   cur = head;
13   while (cur != 0) {
14     Node n = read(cur);
15     if(n.next != 0)
16       assert(n.data == 2);
17     else
18       assert(n.data == 3);
19     cur = n.next;
20   }
21 }
```

Name	Value
head	*
cur	*
i	*
N	3
last	*
last_addr	2
R	{(1, Node(2,2))}

.

Meanwhile, in another execution... (last_addr = 2)

```
1  void main(int N) {
2    int head = alloc();
3    int cur = head;
4
5    for (int i = 0; i < N; i++) {
6      writeNode(cur, 2, alloc());
7      Node n = read(cur);
8      cur = n.next;
9    }
10   writeNode(cur, 3, 0);
11
12   cur = head;
13   while (cur != 0) {
14     Node n = read(cur);
15     if(n.next != 0)
16       assert(n.data == 2);
17     else
18       assert(n.data == 3);
19     cur = n.next;
20   }
21 }
```

Name	Value
head	1
cur	*
i	*
N	3
last	*
last_addr	2
R	{(1, Node(2,2))}

Meanwhile, in another execution. . . (last_addr = 2)

```
1  void main(int N) {
2    int head = alloc();
3    int cur = head;
4
5    for (int i = 0; i < N; i++) {
6      writeNode(cur, 2, alloc());
7      Node n = read(cur);
8      cur = n.next;
9    }
10   writeNode(cur, 3, 0);
11
12   cur = head;
13   while (cur != 0) {
14     Node n = read(cur);
15     if(n.next != 0)
16       assert(n.data == 2);
17     else
18       assert(n.data == 3);
19     cur = n.next;
20   }
21 }
```

Name	Value
head	1
cur	1
i	*
N	3
last	*
last_addr	2
R	{(1, Node(2,2))}

Meanwhile, in another execution. . . (last_addr = 2)

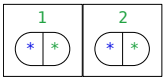
```
1  void main(int N) {
2    int head = alloc();
3    int cur = head;
4
5    for (int i = 0; i < N; i++) {
6      writeNode(cur, 2, alloc());
7      Node n = read(cur);
8      cur = n.next;
9    }
10   writeNode(cur, 3, 0);
11
12   cur = head;
13   while (cur != 0) {
14     Node n = read(cur);
15     if(n.next != 0)
16       assert(n.data == 2);
17     else
18       assert(n.data == 3);
19     cur = n.next;
20   }
21 }
```

Name	Value
head	1
cur	1
i	0
N	3
last	*
last_addr	2
R	{(1, Node(2,2))}

Meanwhile, in another execution. . . (last_addr = 2)

```
1  void main(int N) {
2    int head = alloc();
3    int cur = head;
4
5    for (int i = 0; i < N; i++) {
6      writeNode(cur, 2, alloc());
7      Node n = read(cur);
8      cur = n.next;
9    }
10   writeNode(cur, 3, 0);
11
12   cur = head;
13   while (cur != 0) {
14     Node n = read(cur);
15     if(n.next != 0)
16       assert(n.data == 2);
17     else
18       assert(n.data == 3);
19     cur = n.next;
20   }
21 }
```

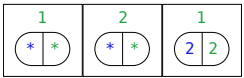
Name	Value
head	1
cur	1
i	0
N	3
last	Node(*,*)
last_addr	2
R	{(1, Node(2,2))}



Meanwhile, in another execution. . . (last_addr = 2)

```
1  void main(int N) {
2    int head = alloc();
3    int cur = head;
4
5    for (int i = 0; i < N; i++) {
6      writeNode(cur, 2, alloc());
7      Node n = read(cur);
8      cur = n.next;
9    }
10   writeNode(cur, 3, 0);
11
12   cur = head;
13   while (cur != 0) {
14     Node n = read(cur);
15     if(n.next != 0)
16       assert(n.data == 2);
17     else
18       assert(n.data == 3);
19     cur = n.next;
20   }
21 }
```

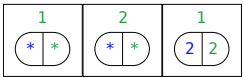
Name	Value
head	1
cur	1
i	0
N	3
last	Node(*,*)
last_addr	2
R	{(1, Node(2,2))}



Meanwhile, in another execution. . . (last_addr = 2)

```
1  void main(int N) {
2    int head = alloc();
3    int cur = head;
4
5    for (int i = 0; i < N; i++) {
6      writeNode(cur, 2, alloc());
7      Node n = read(cur);
8      cur = n.next;
9    }
10   writeNode(cur, 3, 0);
11
12   cur = head;
13   while (cur != 0) {
14     Node n = read(cur);
15     if(n.next != 0)
16       assert(n.data == 2);
17     else
18       assert(n.data == 3);
19     cur = n.next;
20   }
21 }
```

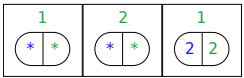
Name	Value
head	1
cur	1
i	0
N	3
last	Node(*,*)
last_addr	2
R	{(1, Node(2,2))}



Meanwhile, in another execution... (last_addr = 2)

```
1  void main(int N) {
2    int head = alloc();
3    int cur = head;
4
5    for (int i = 0; i < N; i++) {
6      writeNode(cur, 2, alloc());
7      Node n = read(cur);
8      cur = n.next;
9    }
10   writeNode(cur, 3, 0);
11
12   cur = head;
13   while (cur != 0) {
14     Node n = read(cur);
15     if(n.next != 0)
16       assert(n.data == 2);
17     else
18       assert(n.data == 3);
19     cur = n.next;
20   }
21 }
```

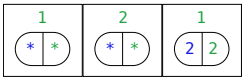
Name	Value
head	1
cur	2
i	0
N	3
last	Node(*,*)
last_addr	2
R	{(1, Node(2,2))}



Meanwhile, in another execution. . . (last_addr = 2)

```
1  void main(int N) {
2    int head = alloc();
3    int cur = head;
4
5    for (int i = 0; i < N; i++) {
6      writeNode(cur, 2, alloc());
7      Node n = read(cur);
8      cur = n.next;
9    }
10   writeNode(cur, 3, 0);
11
12   cur = head;
13   while (cur != 0) {
14     Node n = read(cur);
15     if(n.next != 0)
16       assert(n.data == 2);
17     else
18       assert(n.data == 3);
19     cur = n.next;
20   }
21 }
```

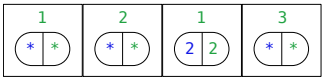
Name	Value
head	1
cur	2
i	1
N	3
last	Node(*,*)
last_addr	2
R	{(1, Node(2,2))}



Meanwhile, in another execution. . . (last_addr = 2)

```
1  void main(int N) {
2    int head = alloc();
3    int cur = head;
4
5    for (int i = 0; i < N; i++) {
6      writeNode(cur, 2, alloc());
7      Node n = read(cur);
8      cur = n.next;
9    }
10   writeNode(cur, 3, 0);
11
12   cur = head;
13   while (cur != 0) {
14     Node n = read(cur);
15     if(n.next != 0)
16       assert(n.data == 2);
17     else
18       assert(n.data == 3);
19     cur = n.next;
20   }
21 }
```

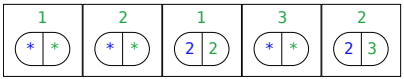
Name	Value
head	1
cur	2
i	1
N	3
last	Node(*,*)
last_addr	2
R	{(1, Node(2,2))}



Meanwhile, in another execution... (last_addr = 2)

```
1  void main(int N) {
2    int head = alloc();
3    int cur = head;
4
5    for (int i = 0; i < N; i++) {
6      writeNode(cur, 2, alloc());
7      Node n = read(cur);
8      cur = n.next;
9    }
10   writeNode(cur, 3, 0);
11
12   cur = head;
13   while (cur != 0) {
14     Node n = read(cur);
15     if(n.next != 0)
16       assert(n.data == 2);
17     else
18       assert(n.data == 3);
19     cur = n.next;
20   }
21 }
```

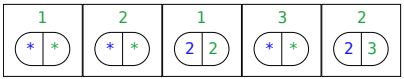
Name	Value
head	1
cur	2
i	1
N	3
last	Node(2,3)
last_addr	2
R	{(1, Node(2,2))}



Meanwhile, in another execution... (last_addr = 2)

```
1  void main(int N) {
2    int head = alloc();
3    int cur = head;
4
5    for (int i = 0; i < N; i++) {
6      writeNode(cur, 2, alloc());
7      Node n = read(cur);
8      cur = n.next;
9    }
10   writeNode(cur, 3, 0);
11
12   cur = head;
13   while (cur != 0) {
14     Node n = read(cur);
15     if(n.next != 0)
16       assert(n.data == 2);
17     else
18       assert(n.data == 3);
19     cur = n.next;
20   }
21 }
```

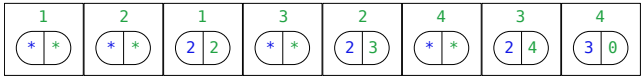
Name	Value
head	1
cur	2
i	1
N	3
last	Node(2,3)
last_addr	2
R	{(1, Node(2,2)), (2, Node(2,3))}



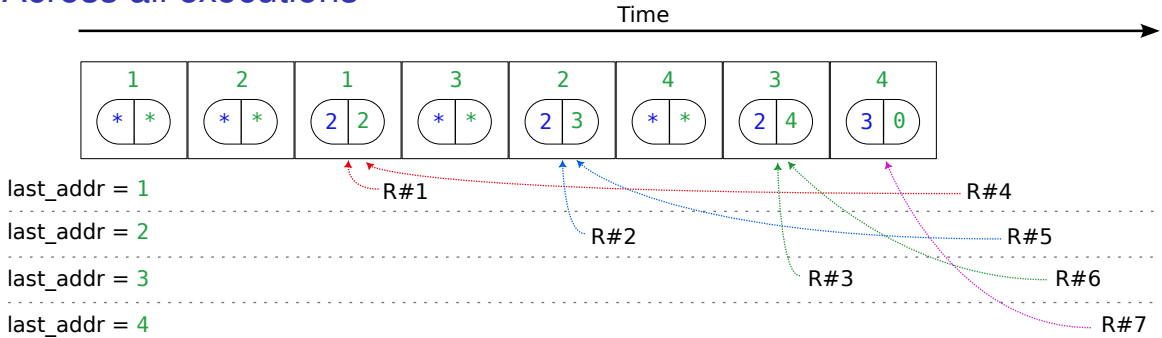
Meanwhile, in another execution... (last_addr = 2)

```
1  void main(int N) {
2    int head = alloc();
3    int cur = head;
4
5    for (int i = 0; i < N; i++) {
6      writeNode(cur, 2, alloc());
7      Node n = read(cur);
8      cur = n.next;
9    }
10   writeNode(cur, 3, 0);
11
12   cur = head;
13   while (cur != 0) {
14     Node n = read(cur);
15     if(n.next != 0)
16       assert(n.data == 2);
17     else
18       assert(n.data == 3);
19     cur = n.next;
20   }
21 }
```

Name	Value
head	1
cur	1
i	3
N	3
last	Node(2,3)
last_addr	2
R	{(1, Node(2,2)), (2, Node(2,3))}

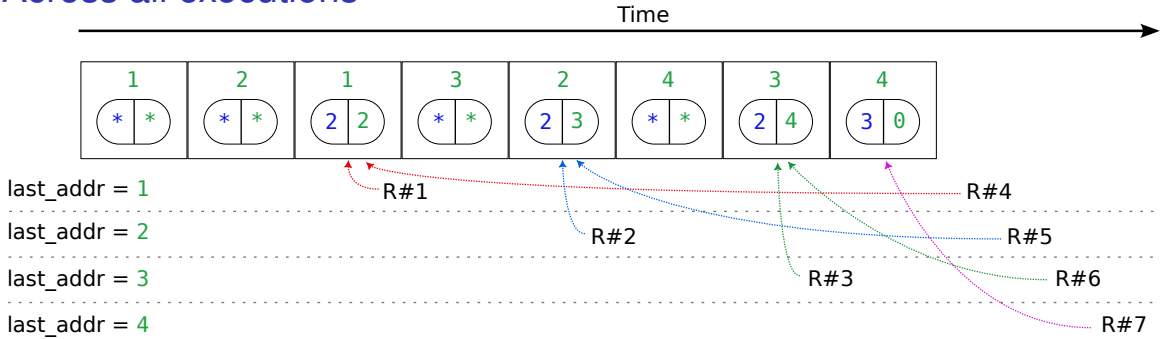


Across all executions



R = {(1, [2,2]), (2, [2,3]), (3, [2,4]), (4, [2,2]), (5, [2,3]),(6, [2,4]), (7, [3,0])}

Across all executions



R = {(1, [2,2]), (2, [2,3]), (3, [2,4]), (4, [2,2]), (5, [2,3]),(6, [2,4]), (7, [3,0])}

$$
\begin{aligned}
R(N, cnt, n) \equiv\ & (N < 0 \wedge cnt = 1 \wedge n.next = 0 \wedge n.data = 3) \\
\vee\ & (N \geq 0 \wedge ((cnt \leq N \wedge n.next = cnt + 1) \\
& \vee\ (N < cnt \leq 2N \wedge n.next = cnt - N + 1 \wedge n.data = 2) \\
& \vee\ (cnt = 2N + 1 \wedge n.next = 0 \wedge n.data = 3))).
\end{aligned}
$$

Other Encodings

p statement	$Enc_{RWfun}(p)$ statement	$Enc_{RWmem}(p)$ statement
$x := \textbf{read}(p)$	$cnt := cnt + 1;$ $\textbf{if } last_{Addr} = p \textbf{ then } \{$ $\quad \textbf{assert}(R(in, cnt, cnt_{last}));$ $\quad t := cnt_{last}$ $\} \textbf{ else } \{$ $\quad havoc(t);$ $\quad \textbf{assume}(R(in, cnt, t))$ $\};$ $havoc(x);$ $\textbf{assume}(W(in, t, x));$	$cnt := cnt + 1;$ $\textbf{if } \neg(0 < p \leq cnt_{alloc}) \{\textbf{assert}(0)\};$ $\textbf{if } last_{Addr} = p \textbf{ then } \{$ $\quad \textbf{assert}(R(in, cnt, cnt_{last}));$ $\quad t := cnt_{last}$ $\} \textbf{ else } \{$ $\quad havoc(t);$ $\quad \textbf{assume}(R(in, cnt, t))$ $\};$ $havoc(x);$ $\textbf{assume}(W(in, t, x));$

Experimental Results

Tool	Encoding	Safe (Correct)	Unsafe (Correct)	Unknown	Total
PredatorHP	None	7 (7)	8 (8)	5	20
TriCera	None	4 (4)	8 (8)	8	20
CPAchecker	None	4 (4)	12 (8)	4	20
SeaHorn	None	3 (3)	17 (8)	0	20
SeaHorn	*RW-fun*	11 (11)	8 (8)	1	20
TriCera	*RW-fun*	11 (11)	8 (8)	1	20
TriCera	*R*	6 (6)	8 (8)	6	20
TriCera	*RW*	6 (6)	8 (8)	6	20
SeaHorn	*R*	7 (7)	12 (8)	1	20
SeaHorn	*RW*	2 (2)	18 (8)	0	20

Making Input Programs Deterministic

The encoding only works for *deterministic* programs.

```
1
2
3
4
5
6
7
8  void main() {
9    [...]
10   while (nondet_int()) {
11     Node node = nondet_Node();
12     [...]
13   }
14   [...]
15 }
```

```
1  /*
2    Inputs:
3      - Node arr_Node[]
4      - int arr_int[]
5  */
6  int i_int = 0;
7  int i_Node = 0
8  void main() {
9    [...]
10   while (arr_int[i_int++]) {
11     Node node = arr_Node[i_Node++];
12     [...]
13   }
14   [...]
15 }
```

Implemented in the Clang-based preprocessor of TRICERA.

Conclusion & Future Work

Preliminary experiments: input programs were normalized by hand.

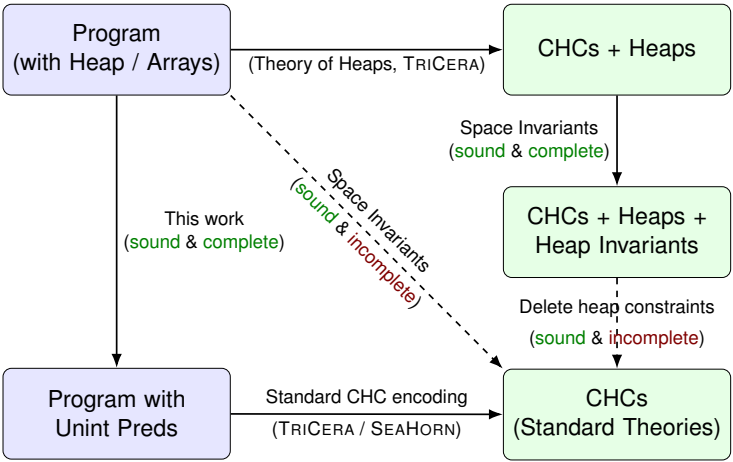
Ongoing work to implement the fully automated approach in TriCera[1][2].

“Sound and Complete Invariant-Based Heap Encodings (Technical Report)”, Z. Esen, P. Rümmer, T. Weber. `https://arxiv.org/abs/2504.15844` (under submission).

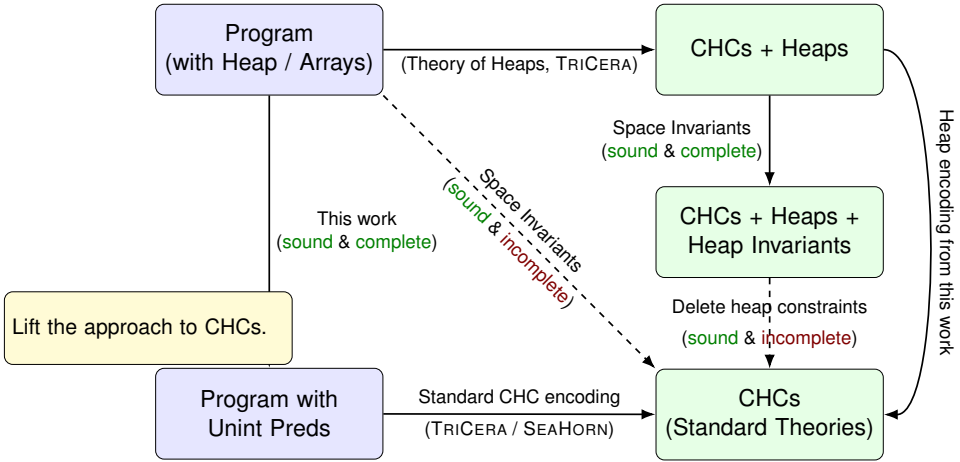
[1] `https://github.com/uuverifiers/tricera`

[2] `https://github.com/EuroProofNet/ProgramVerification/wiki/TriCera`

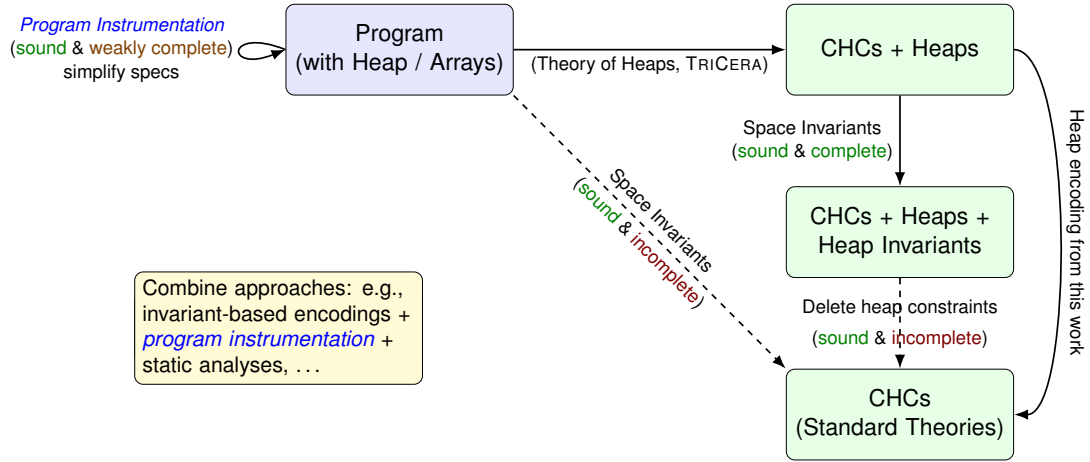
Conclusion & Future Work



Conclusion & Future Work



Conclusion & Future Work



Thank you!

Example Transformation (Space / Heap Invariants)

$$r_1(h, a) \leftarrow h = \text{emptyHeap} \wedge a = \text{nullAddr} \tag{1}$$

$$r_1(h', a') \leftarrow r_1(h, a) \wedge n = \mathit{Node}(3, a) \wedge (h', a') = \text{allocate}(h, n) \tag{2}$$

$$\underline{\mathtt{I}(a', n) \leftarrow r_1(h, a) \wedge n = \mathit{Node}(3, a) \wedge (h', a') = \text{allocate}(h, n)} \tag{3}$$

$$r_2(h, a) \leftarrow r_1(h, a) \tag{4}$$

$$r_2(h, a') \leftarrow r_2(h, a) \wedge n = \text{read}(h, a) \wedge a \neq \text{nullAddr} \wedge a' = \mathit{next}(n) \wedge \underline{\text{valid}(h, a) \wedge \mathtt{I}(a, n)} \tag{5}$$

$$\underline{r_2(h, a') \leftarrow r_2(h, a) \wedge n = \text{read}(h, a) \wedge a \neq \text{nullAddr} \wedge a' = \mathit{next}(n) \wedge n = \mathit{defObj} \wedge \neg\text{valid}(h, a)} \tag{6}$$

$$\bot \leftarrow r_2(h, a) \wedge n = \text{read}(h, a) \wedge a \neq \text{nullAddr} \wedge \mathit{data}(n) \neq 3 \wedge \underline{\text{valid}(h, a) \wedge \mathtt{I}(a, n)} \tag{7}$$

$$\underline{\bot \leftarrow r_2(h, a) \wedge n = \text{read}(h, a) \wedge a \neq \text{nullAddr} \wedge \mathit{data}(n) \neq 3 \wedge n = \mathit{defObj} \wedge \neg\text{valid}(h, a)} \tag{8}$$

Correctness of Heap Invariants Transformation

Safety of a program with uninterpreted predicates

A program is safe if there exists an interpretation under which no execution leads to an assertion failure.

p: original program, p': transformed program

- **Soundness:** p' is safe $\implies$ p is safe. (or equivalently: p is unsafe $\implies$ p' is unsafe.)
- **Completeness:** p is safe $\implies$ p' is safe.

Formal proofs via transfinite induction over the fixed-point semantics of CHCs.

Extensions: Flow Sensitivity

```
1  Node* a = NULL;
2  while (nondetInt()) {
3    Node* t = new Node(3, a); // update site 1
4    assert(I(t, Node(3, a), 1));
5    *t.data = 4;              // update site 2
6    assert(I(t, Node(4, a), 2));
7    a = t;
8  }
9  while (a) {
10   Node n = *a;
11   int lastWrite = *;        // nondet
12   assume(I(a, n, lastWrite));
13   assume(lastWrite == 2);   // only updates from 2 reach here
14   assert(n.data == 4);
15   a = n.next;
16 }
```

Related Work (Heap Reasoning)

- Global heap reasoning
  - Shape analysis (abstraction of entire heap)
  - Directly solving universally quantified Horn clauses (Bjørner et al., 2013)
  - Cell morphing – reducing CHCs + arrays to CHCs over ints (Monniaux et al., 2016)

Related Work (Heap Reasoning)

- Global heap reasoning
  - Shape analysis (abstraction of entire heap)
  - Directly solving universally quantified Horn clauses (Bjørner et al., 2013)
  - Cell morphing – reducing CHCs + arrays to CHCs over ints (Monniaux et al., 2016)

- Local heap reasoning
  - Separation logic and the flow framework
  - Refinement/Liquid types (embed invariants into types)
  - *Invariant-based heap encodings*
    - Space invariants (Kahsai, Kersten, Rümmer and Schäf, LPAR, 2017)
    - Heap invariants (Esen, Rümmer and Weber, FSEN, 2025)
    - Sound and complete invariant-based heap encodings

Functions & Predicates of the Theory

$$
\begin{array}{rcll}
\text{nullAddr} & : & () & \to \mathit{Address} \\
\text{emptyHeap} & : & () & \to \mathit{Heap} \\
\text{allocate} & : & \mathit{Heap} \times \mathit{Object} & \to \mathit{Heap} \times \mathit{Address} \\
\text{read} & : & \mathit{Heap} \times \mathit{Address} & \to \mathit{Object} \\
\text{write} & : & \mathit{Heap} \times \mathit{Address} \times \mathit{Object} & \to \mathit{Heap} \\
\text{valid} & : & \mathit{Heap} \times \mathit{Address} & \to \mathit{Bool}
\end{array}
$$

Example Encoding Using the Theory

```
1 Node* list = new Node(0, NULL);
2 list->next = new Node(list->data + 1, NULL);
```

```
1 I1(emptyHeap).
```

Example Encoding Using the Theory

```
1 Node* list = new Node(0, NULL);
2 list->next = new Node(list->data + 1, NULL);
```

```
1 I1(emptyHeap).
2 I2(ar._1, ar._2) :- I1(h),
3                     ar = alloc(h, Node(0, nullAddress)).
```

Example Encoding Using the Theory

```
1 Node* list = new Node(0, NULL);
2 list->next = new Node(list->data + 1, NULL);
```

```
1 I1(emptyHeap).
2 I2(ar._1, ar._2) :- I1(h),
3                     ar = alloc(h, Node(0, nullAddress)).
4
5 I3(h, list, n)   :- I2(h, list), n = read(h, list).
```

Example Encoding Using the Theory

```
1 Node* list = new Node(0, NULL);
2 list->next = new Node(list->data + 1, NULL);
```

```
1 I1(emptyHeap).
2 I2(ar._1, ar._2) :- I1(h),
3                     ar = alloc(h, Node(0, nullAddress)).
4
5 I3(h, list, n)   :- I2(h, list), n = read(h, list).
6 false            :- I2(h, list), !valid(h, list).
7 ...
```

Axioms of the Theory

read-over-write

$\text{valid}(h, p) \implies \text{read}(\text{write}(h, p, o), p) = o$

$p_1 \neq p_2 \implies \text{read}(\text{write}(h, p1, o), p2) = \text{read}(h, p2)$

Other axioms and details in *Paper I*.

Experiments / SV-COMP 2022 Results

Heap Benchmarks

Tool	Solved
VeriAbs	304
Esbmc-kind	171
Symbiotic	164
Lart	120
Cbmc	116
PeSCo	112
CPAchecker	104
Crux	103
TriCera (portfolio)	97
TriCera (Eldarica-array)	85
VeriFuzz	71
UKojak	67
UAutomizer	67
UTaipan	65
TriCera (Z3/Spacer)	60
2ls	56
TriCera (Eldarica-heap)	48
Graves-CPA	48
Goblint	27
Theta	17

Non-Heap Benchmarks

Tool	Solved
VeriAbs	1246
CPAchecker	1136
TriCera (portfolio)	1109
Graves-CPA	1078
TriCera (Eldarica)	1058
PeSCo	1042
UAutomizer	914
UTaipan	896
Symbiotic	881
Esbmc-kind	864
Lart	738
Crux	720
TriCera (Z3/Spacer)	713
Cbmc	707
2ls	693
VeriFuzz	515
UKojak	499
Theta	390
Goblint	180

Heap Invariants Transformation Rules (A Subset)

$$\frac{\mathit{head} \leftarrow \mathit{body} \wedge h_1 = \text{write}(h_0, a, o)}{\begin{array}{l}\mathit{head} \leftarrow \mathit{body} \wedge h_1 = \text{write}(h_0, a, o) \\ \texttt{I}(a, o) \leftarrow \mathit{body} \wedge \text{valid}(h_0, a)\end{array}} \text{ (write)}$$

$$\frac{\mathit{head} \leftarrow \mathit{body} \wedge o = \text{read}(h, a)}{\begin{array}{l}\mathit{head} \leftarrow \mathit{body} \wedge o = \text{read}(h, a) \wedge \text{valid}(h, a) \wedge \texttt{I}(a, o) \\ \mathit{head} \leftarrow \mathit{body} \wedge o = \text{read}(h, a) \wedge o = \mathit{defObj} \wedge \neg\text{valid}(h, a)\end{array}} \text{ (read)}$$