Synergistic Debug-Repair of Heap Manipulations

Sahil Verma and **Subhajit Roy**

Indian Institute of Technology Kanpur

{*vsahil,subhajit*}*@iitk.ac.in*

September 6th, 2017

Overview

Plan

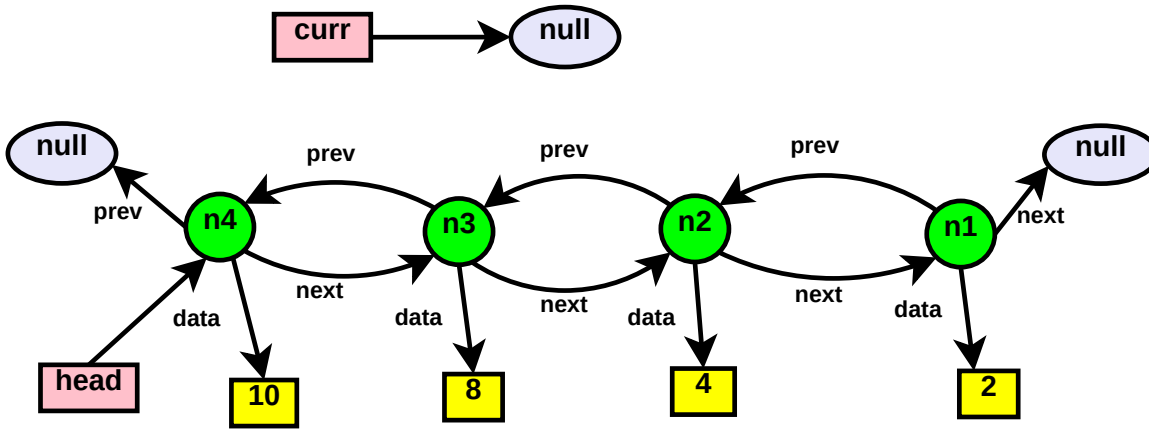
Wolverine in action

```
struct node *head;

void reverse() {
  struct node *temp1 = NULL;
  struct node *temp2 = NULL;
  struct node *current = head;

  // FIX1:  current != NULL
  while (temp1 != NULL) {
    temp1 = current->prev;
    // FIX2:  current->next
    temp2 = current->prev;
    current->prev = temp2;
    current->next = temp1;
    current = current->prev;
  }

  // FIX3:  head = temp1->prev;
  return;
}
  ...
int main(){
  push(2); push(4); push(8);
push(10);
  reverse();
```




```
(Wolverine) start
Starting ...
(Wolverine)
```

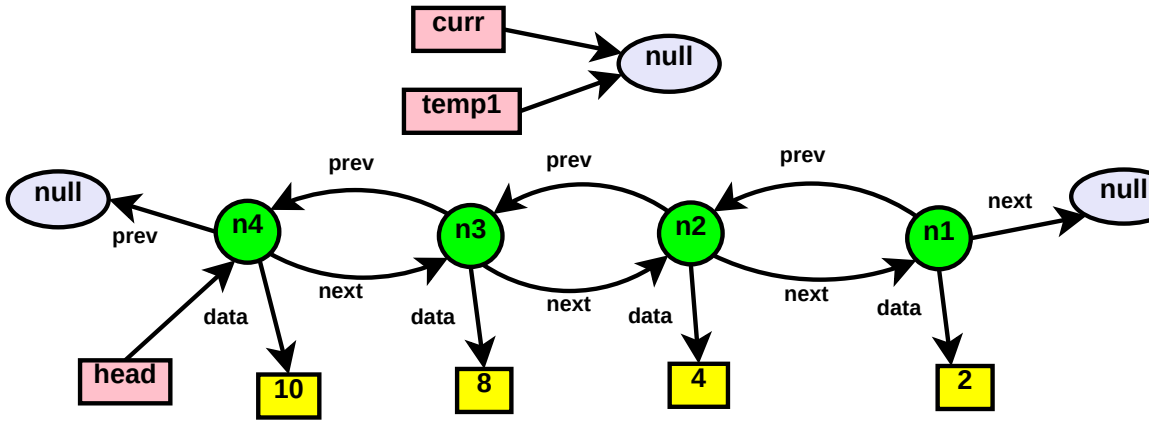
WOLVERINE in action

```
struct node *head;

void reverse() {
  struct node *temp1 = NULL;
  struct node *temp2 = NULL;
  struct node *current = head;

  // FIX1:  current != NULL
  while (temp1 != NULL) {
    temp1 = current->prev;
    // FIX2:  current->next
    temp2 = current->prev;
    current->prev = temp2;
    current->next = temp1;
    current = current->prev;
  }

  // FIX3:  head = temp1->prev;
  return;
}
  ...
int main(){
  push(2); push(4); push(8);
push(10);
  reverse();
```




```
struct node *temp1 = NULL;
(Wolverine) next
```

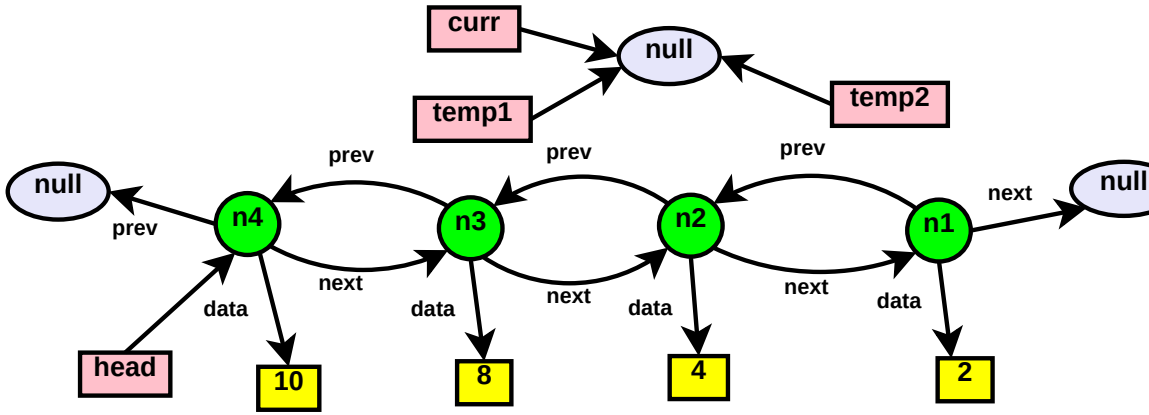
WOLVERINE in action

```
struct node *head;

void reverse() {
  struct node *temp1 = NULL;
  struct node *temp2 = NULL;
  struct node *current = head;

  // FIX1:  current != NULL
  while (temp1 != NULL) {
    temp1 = current->prev;
    // FIX2:  current->next
    temp2 = current->prev;
    current->prev = temp2;
    current->next = temp1;
    current = current->prev;
  }

  // FIX3:  head = temp1->prev;
  return;
}
  ...
int main(){
  push(2); push(4); push(8);
push(10);
  reverse();
```




```
struct node *temp1 = NULL;
(Wolverine) next
struct node *temp2 = NULL;
(Wolverine) next
```

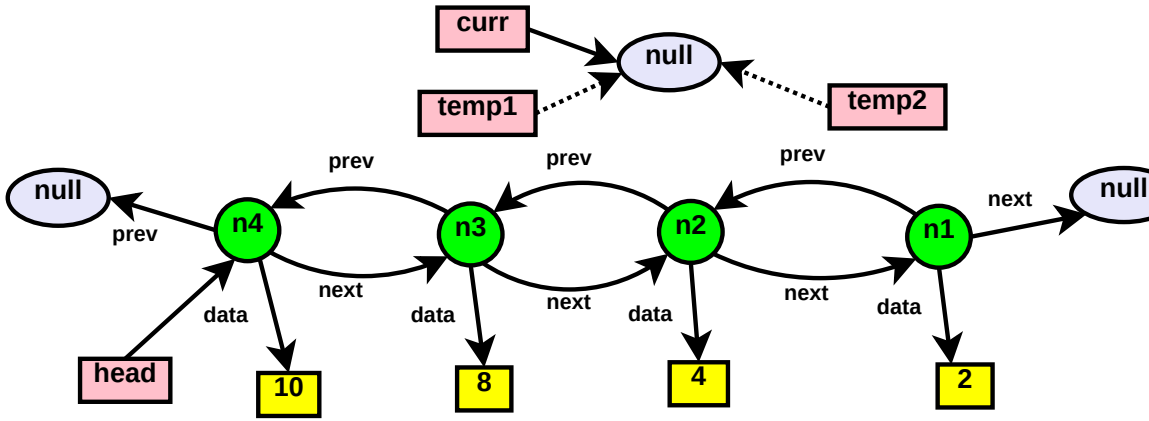
Wolverine in action

```
struct node *head;

void reverse() {
  struct node *temp1 = NULL;
  struct node *temp2 = NULL;
  struct node *current = head;

  // FIX1:  current != NULL
  while (temp1 != NULL) {
    temp1 = current->prev;
    // FIX2:  current->next
    temp2 = current->prev;
    current->prev = temp2;
    current->next = temp1;
    current = current->prev;
  }

  // FIX3:  head = temp1->prev;
  return;
}
  ...
```




```
struct node *current = head;
(Wolverine) track [] [temp1, temp2]
```

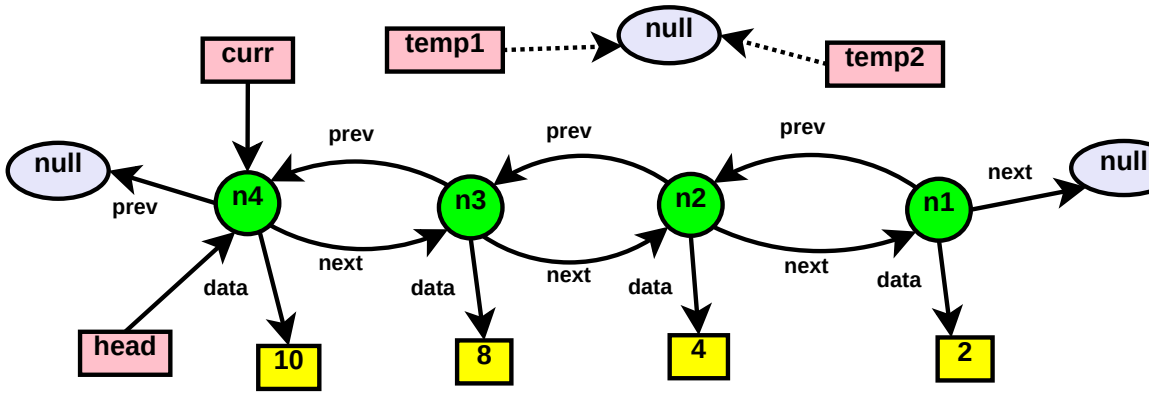
Wolverine in action

```
struct node *head;

void reverse() {
  struct node *temp1 = NULL;
  struct node *temp2 = NULL;
  struct node *current = head;

  // FIX1:  current != NULL
  while (temp1 != NULL) {
    temp1 = current->prev;
    // FIX2:  current->next
    temp2 = current->prev;
    current->prev = temp2;
    current->next = temp1;
    current = current->prev;
  }

  // FIX3:  head = temp1->prev;
  return;
}
  ...
```




```
struct node *current = head;
(Wolverine) next
```

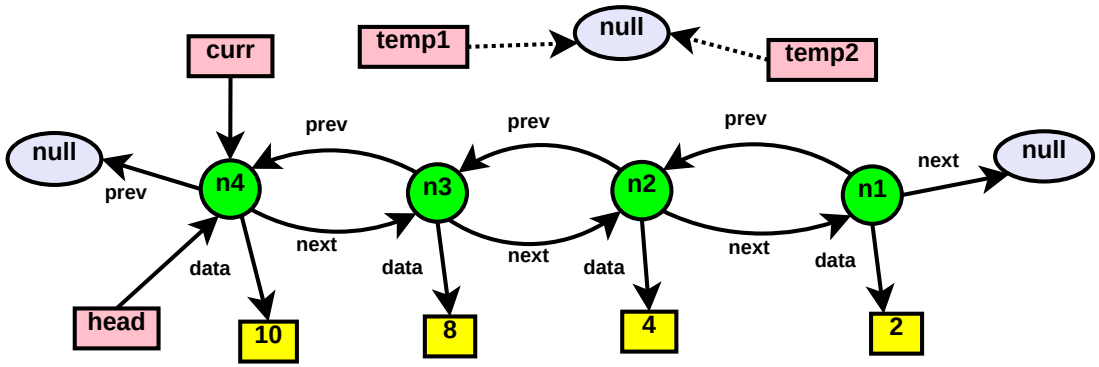
Wolverine in action

```
struct node *head;

void reverse() {
  struct node *temp1 = NULL;
  struct node *temp2 = NULL;
  struct node *current = head;

  // FIX1:  current != NULL
  while (temp1 != NULL) {
    temp1 = current->prev;
    // FIX2:  current->next
    temp2 = current->prev;
    current->prev = temp2;
    current->next = temp1;
    current = current->prev;
  }

  // FIX3:  head = temp1->prev;
  return;
}
  ...
```




```
struct node *current = head;
(Wolverine) next
while(temp1 != NULL)
(Wolverine) enter
```

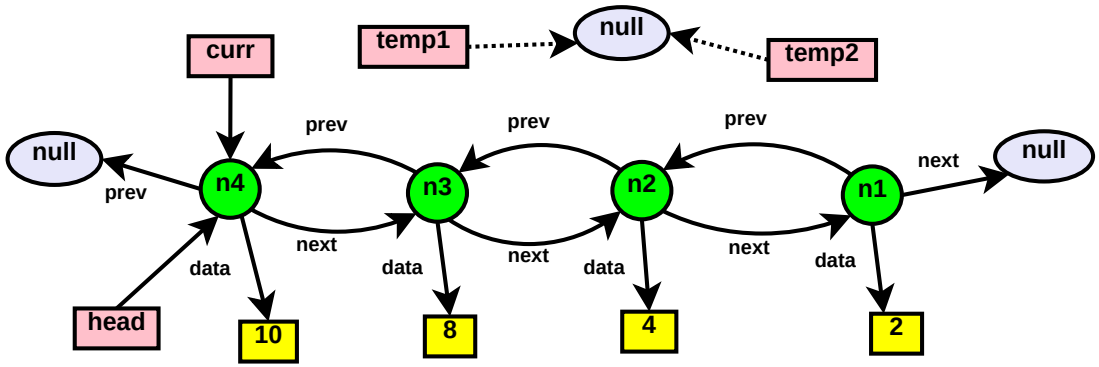
Wolverine in action

```
struct node *head;

void reverse() {
  struct node *temp1 = NULL;
  struct node *temp2 = NULL;
  struct node *current = head;

  // FIX1:  current != NULL
  while (temp1 != NULL) {
    temp1 = current->prev;
    // FIX2:  current->next
    temp2 = current->prev;
    current->prev = temp2;
    current->next = temp1;
    current = current->prev;
  }

  // FIX3:  head = temp1->prev;
  return;
}
  ...
```




```
while(temp1 != NULL)
(Wolverine) enter
 temp1 = current->prev;
(Wolverine) next
```

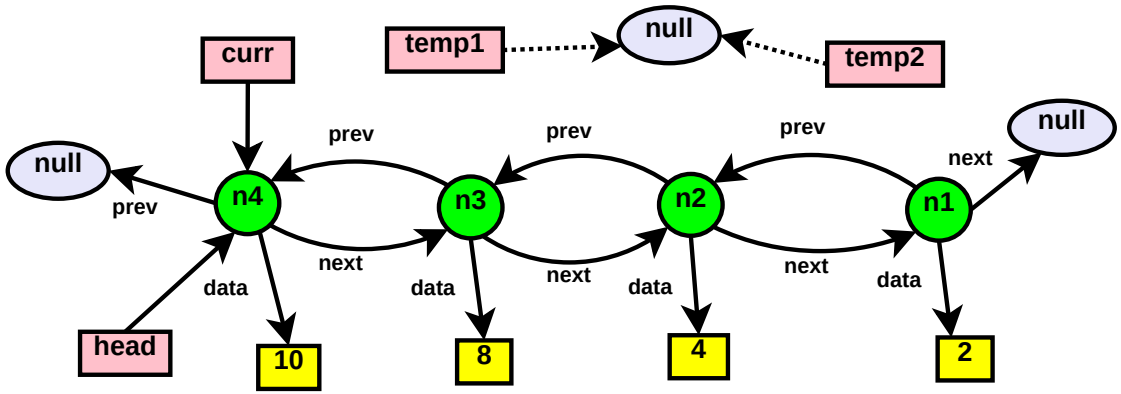
Wolverine in action

```
struct node *head;

void reverse() {
  struct node *temp1 = NULL;
  struct node *temp2 = NULL;
  struct node *current = head;

  // FIX1:  current != NULL
  while (temp1 != NULL) {
    temp1 = current->prev;
    // FIX2:  current->next
    temp2 = current->prev;
    current->prev = temp2;
    current->next = temp1;
    current = current->prev;
  }

  // FIX3:  head = temp1->prev;
  return;
}
  ...
```




```
temp1 = current->prev;
(Wolverine) next
 temp2 = current->prev;
(Wolverine) next
```

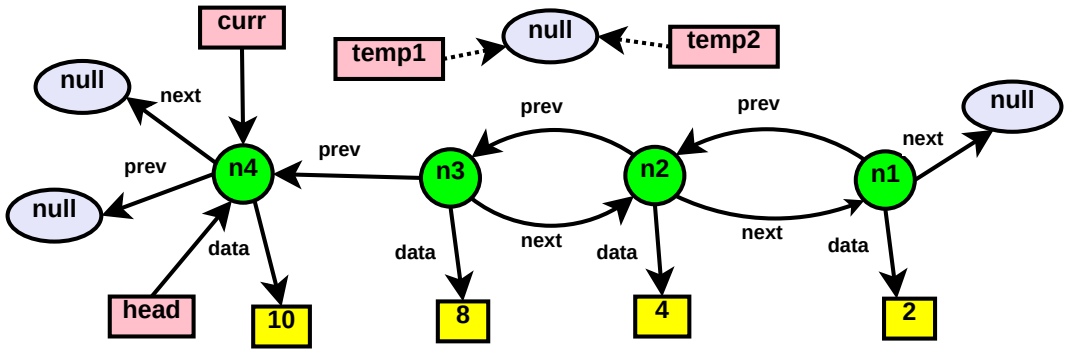
WOLVERINE in action

```
struct node *head;

void reverse() {
  struct node *temp1 = NULL;
  struct node *temp2 = NULL;
  struct node *current = head;

  // FIX1:  current != NULL
  while (temp1 != NULL) {
    temp1 = current->prev;
    // FIX2:  current->next
    temp2 = current->prev;
    current->prev = temp2;
    current->next = temp1;
    current = current->prev;
  }

  // FIX3:  head = temp1->prev;
  return;
}
  ...
```




```
current->prev = temp2;
(Wolverine) next
 current->next = temp1;
(Wolverine) next
```

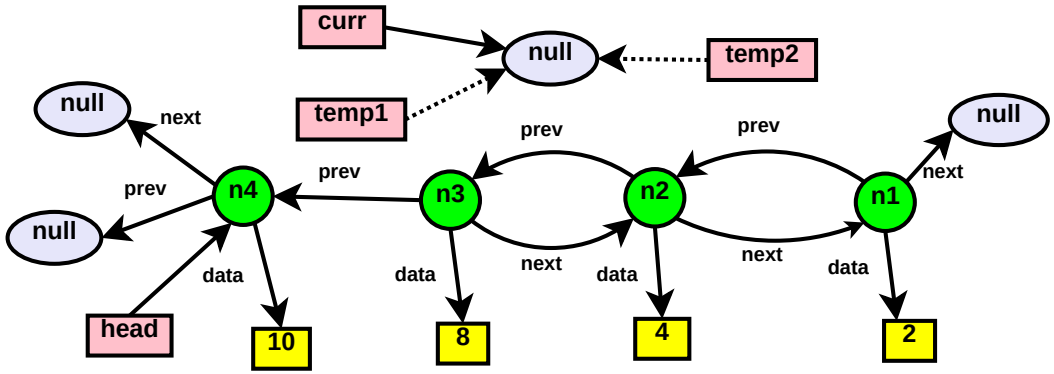
WOLVERINE in action

```
struct node *head;

void reverse() {
  struct node *temp1 = NULL;
  struct node *temp2 = NULL;
  struct node *current = head;

  // FIX1:  current != NULL
  while (temp1 != NULL) {
    temp1 = current->prev;
    // FIX2:  current->next
    temp2 = current->prev;
    current->prev = temp2;
    current->next = temp1;
    current = current->prev;
  }

  // FIX3:  head = temp1->prev;
  return;
}
  ...
```




```
current->next = temp1;
(Wolverine) next
 current = current->prev;
(Wolverine) next
```

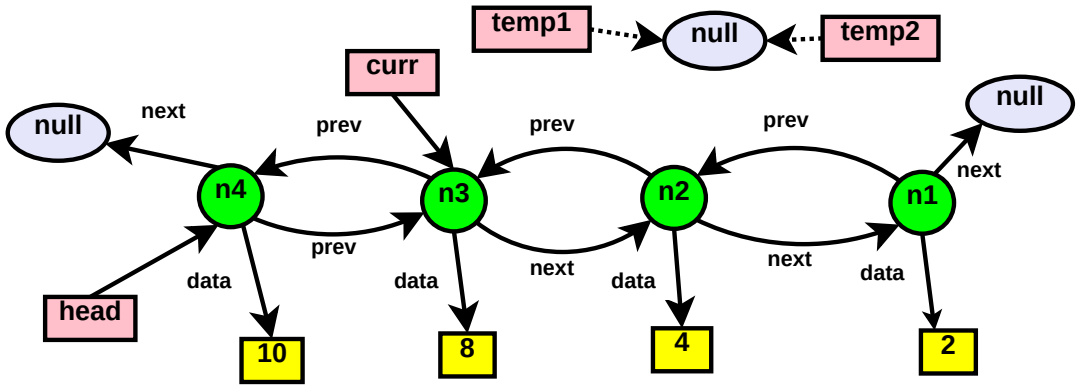
WOLVERINE in action

```
struct node *head;

void reverse() {
  struct node *temp1 = NULL;
  struct node *temp2 = NULL;
  struct node *current = head;

  // FIX1:  current != NULL
  while (temp1 != NULL) {
    temp1 = current->prev;
    // FIX2:  current->next
    temp2 = current->prev;
    current->prev = temp2;
    current->next = temp1;
    current = current->prev;
  }

  // FIX3:  head = temp1->prev;
  return;
}
  ...
```




```
while(temp1 != NULL)
(Wolverine) change curr = n3; n4->prev = n3
 while(temp1 != NULL)
(Wolverine) spec; enter
 Entering loop ...
(Wolverine)
```

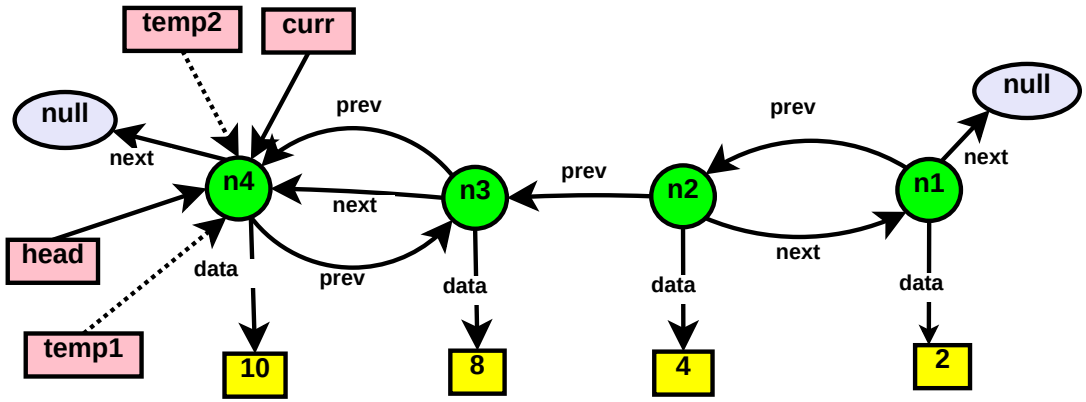
WOLVERINE in action

```
struct node *head;

void reverse() {
  struct node *temp1 = NULL;
  struct node *temp2 = NULL;
  struct node *current = head;

  // FIX1:  current != NULL
  while (temp1 != NULL) {
    temp1 = current->prev;
    // FIX2:  current->next
    temp2 = current->prev;
    current->prev = temp2;
    current->next = temp1;
    current = current->prev;
  }

  // FIX3:  head = temp1->prev;
  return;
}
  ...
```




```
current->next = temp1;
(Wolverine) next
 current = current->prev;
(Wolverine) next
```

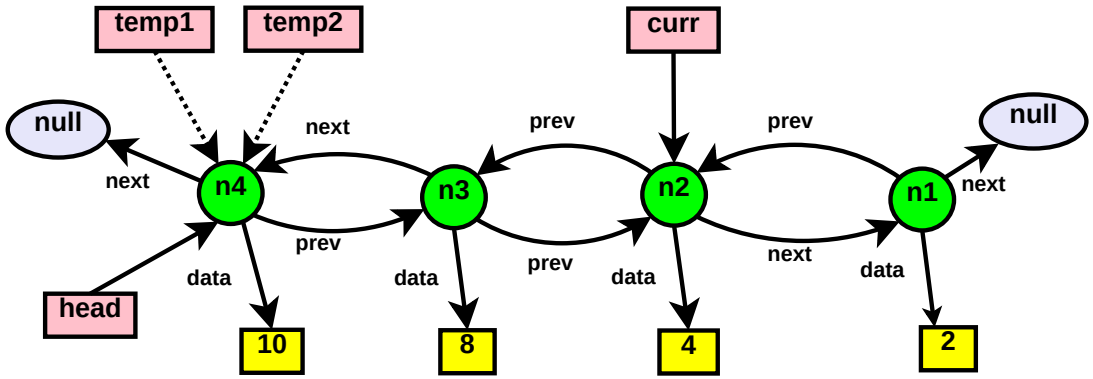
Wolverine in action

```
struct node *head;

void reverse() {
  struct node *temp1 = NULL;
  struct node *temp2 = NULL;
  struct node *current = head;

  // FIX1:  current != NULL
  while (temp1 != NULL) {
    temp1 = current->prev;
    // FIX2:  current->next
    temp2 = current->prev;
    current->prev = temp2;
    current->next = temp1;
    current = current->prev;
  }

  // FIX3:  head = temp1->prev;
  return;
}
  ...
```




```
while(temp1 != NULL)
(Wolverine) change curr = n2; n3->prev = n2
 while(temp1 != NULL)
(Wolverine) spec; repair
 Repair Synthesized ...
(Wolverine)
```

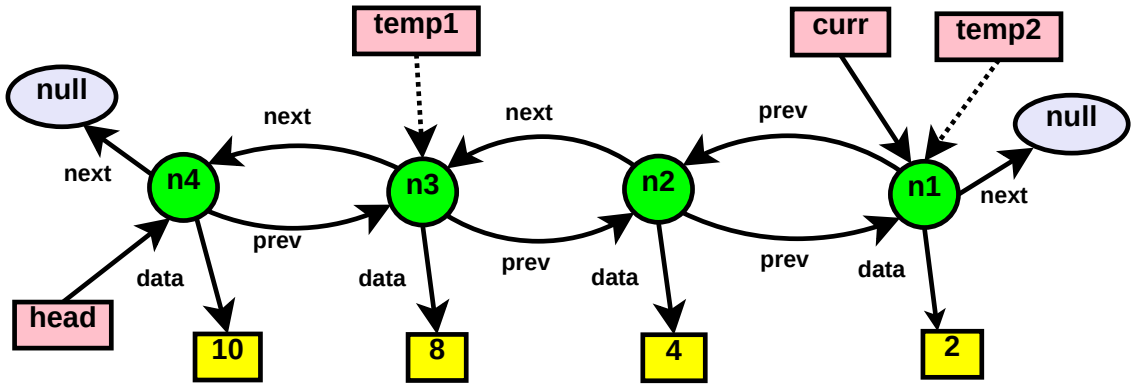
WOLVERINE in action

```
struct node *head;

void reverse() {
  struct node *temp1 = NULL;
  struct node *temp2 = NULL;
  struct node *current = head;

  // FIX1:  current != NULL
  while (temp1 != NULL) {
    temp1 = current->next;
    // FIX2:  current->next
    temp2 = current-> next ;
    current->prev = temp2;
    current->next = temp1;
    current = current->prev;
  }

  // FIX3:  head = temp1->prev;
  return;
}
  ...
```




```
current->next = temp1;
(Wolverine) next
 current = current->prev;
(Wolverine) next while(temp1 != NULL)
(Wolverine) spec; enter
 Entering loop ...
(Wolverine)
```

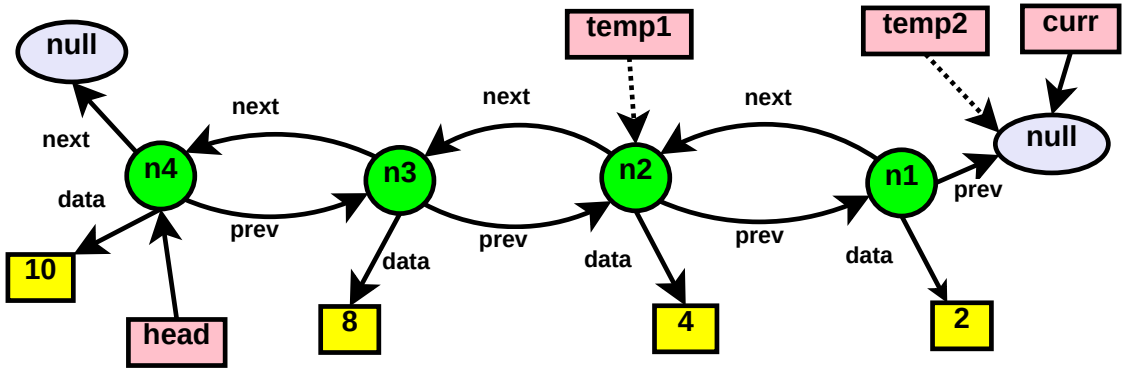
WOLVERINE in action

```
struct node *head;

void reverse() {
  struct node *temp1 = NULL;
  struct node *temp2 = NULL;
  struct node *current = head;

  // FIX1:  current != NULL
  while (temp1 != NULL) {
    temp1 = current->next;
    // FIX2:  current->next
    temp2 = current-> next ;
    current->prev = temp2;
    current->next = temp1;
    current = current->prev;
  }

  // FIX3:  head = temp1->prev;
  return;
}
  ...
```




```
current->next = temp1;
(Wolverine) next
 current = current->prev;
(Wolverine) next
```

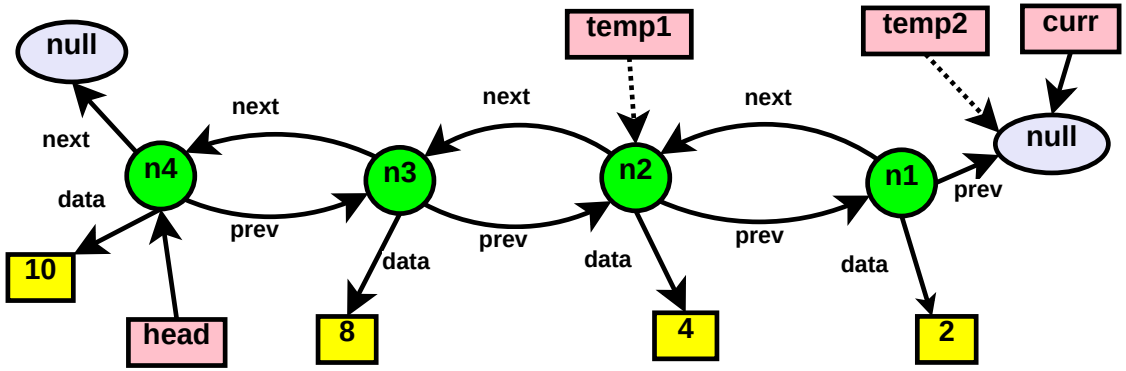
WOLVERINE in action

```
struct node *head;

void reverse() {
  struct node *temp1 = NULL;
  struct node *temp2 = NULL;
  struct node *current = head;

  // FIX1:  current != NULL
  while (temp1 != NULL) {
    temp1 = current->next;
    // FIX2:  current->next
    temp2 = current-> next ;
    current->prev = temp2;
    current->next = temp1;
    current = current->prev;
  }

  // FIX3:  head = temp1->prev;
  return;
}
  ...
```




```
current = current->prev;
(Wolverine) next
 while(temp1 != NULL)
(Wolverine) spec; leave
 Leaving loop ...
(Wolverine)
```

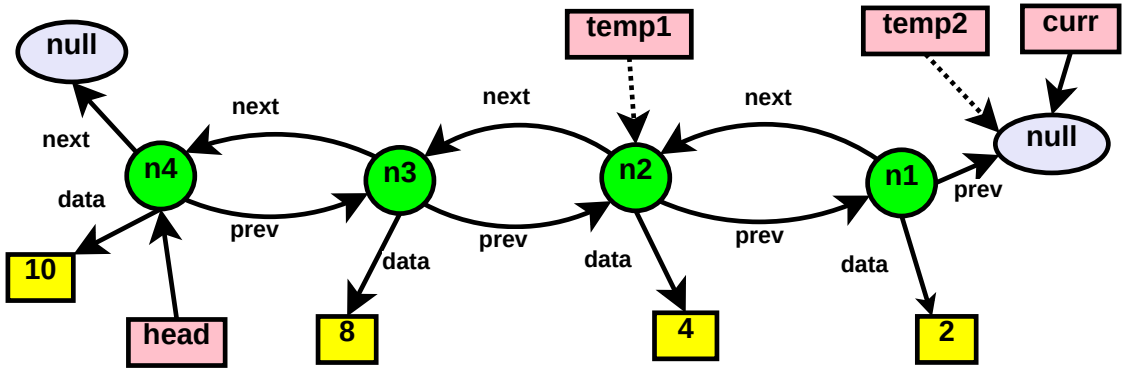
WOLVERINE in action

```
struct node *head;

void reverse() {
  struct node *temp1 = NULL;
  struct node *temp2 = NULL;
  struct node *current = head;

  // FIX1:  current != NULL
  while (temp1 != NULL) {
    temp1 = current->next;
    // FIX2:  current->next
    temp2 = current-> next ;
    current->prev = temp2;
    current->next = temp1;
    current = current->prev;
  }

  // FIX3:  head = temp1->prev;
  return;
}
  ...
```




```
while(temp1 != NULL)
(Wolverine) spec; leave
 Exiting Function ...
(Wolverine) change head = n1
```

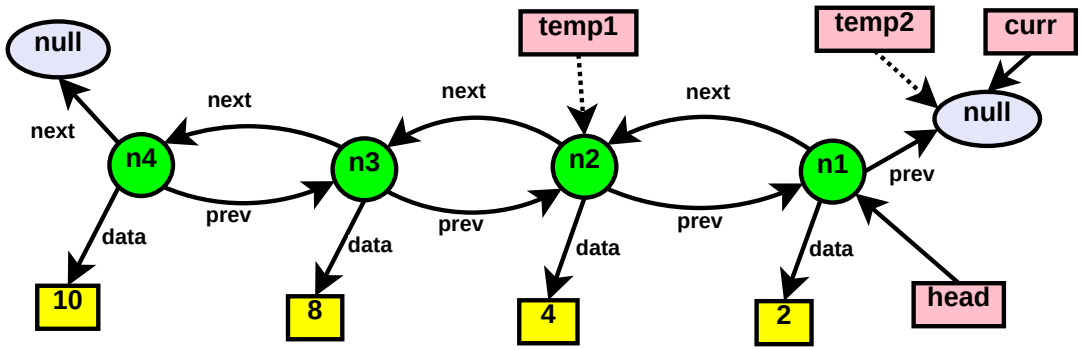
WOLVERINE in action

```
struct node *head;

void reverse() {
  struct node *temp1 = NULL;
  struct node *temp2 = NULL;
  struct node *current = head;

  // FIX1:  current != NULL
  while ( current != NULL) {
    temp1 = current->next;
    // FIX2:  current->next
    temp2 = current-> next ;
    current->prev = temp2;
    current->next = temp1;
    current = current->prev;
  }

  // FIX3:  head = temp1->prev;
  head = temp1->prev;
  return;
}
  ...
```




```
Exiting Function ...
(Wolverine) change head = n1
 Exiting Function ...
(Wolverine) repair
 Repair Synthesized ...
return;
(Wolverine) next
```

Contributions

- We propose that an integrated debug-repair environment can yield significant benefits; we demonstrate it by building a tool, WOLVERINE, to facilitate debug-repair on heap manipulations;

Contributions

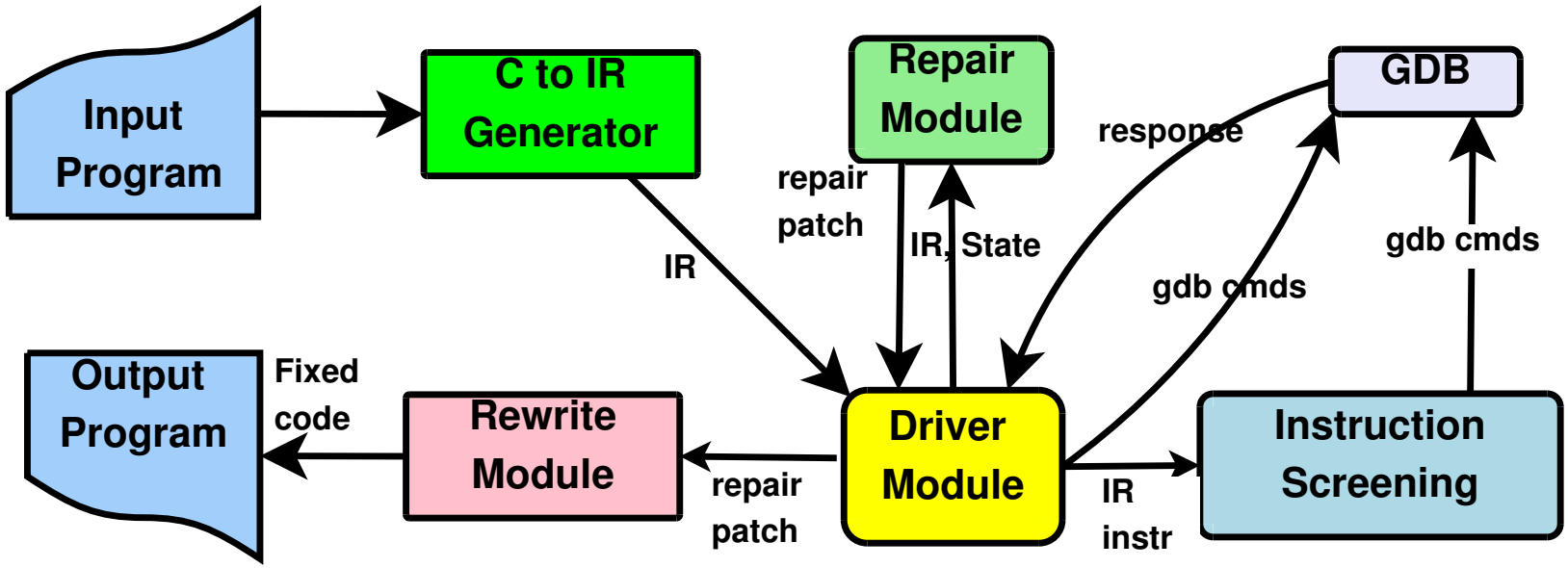
- We propose that an integrated debug-repair environment can yield significant benefits; we demonstrate it by building a tool, WOLVERINE, to facilitate debug-repair on heap manipulations;
- We propose a new proof-directed repair strategy that uses the proof of unsatisfiability to guide the repair along the most promising direction;

Contributions

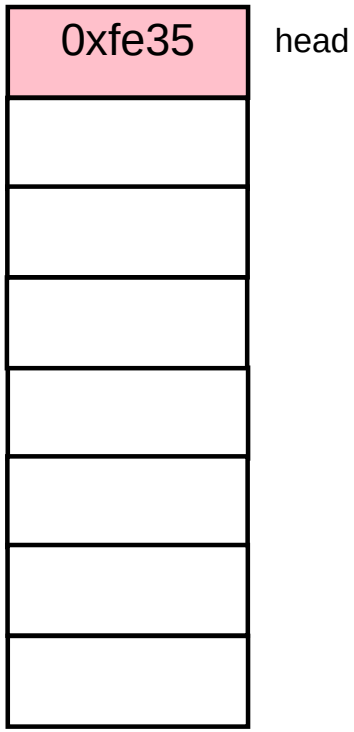
- We propose that an integrated debug-repair environment can yield significant benefits; we demonstrate it by building a tool, WOLVERINE, to facilitate debug-repair on heap manipulations;
- We propose a new proof-directed repair strategy that uses the proof of unsatisfiability to guide the repair along the most promising direction;
- We propose advanced debugging techniques, specification refinement and specification slicing, that are facilitated by this integration of debugging and repair.

Plan

Architecture

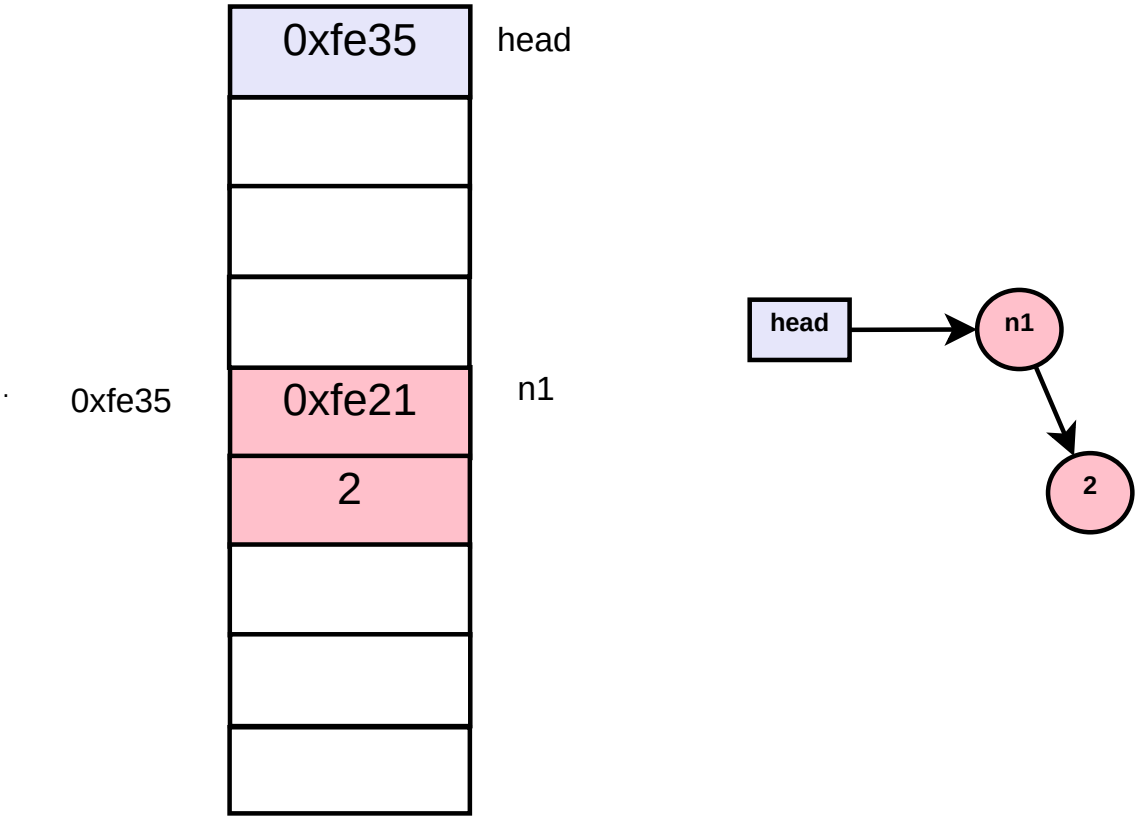


Specification collection: The debug phase

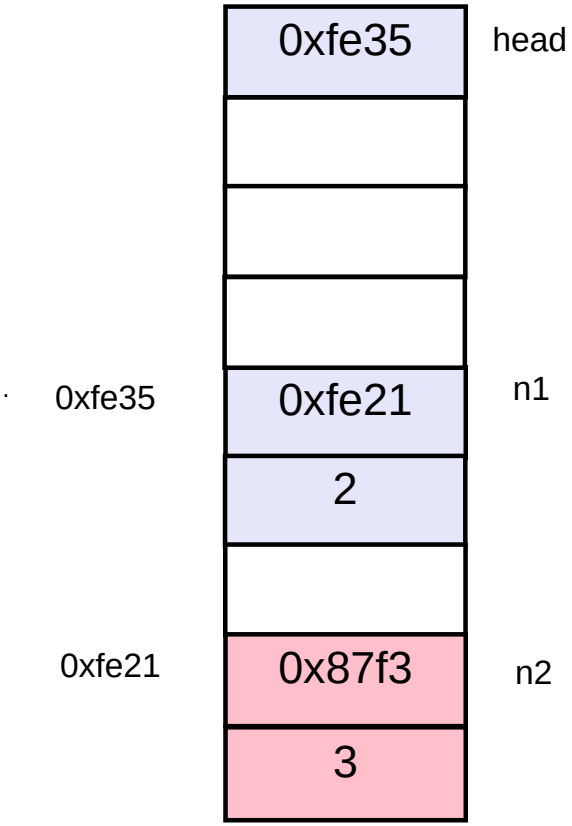


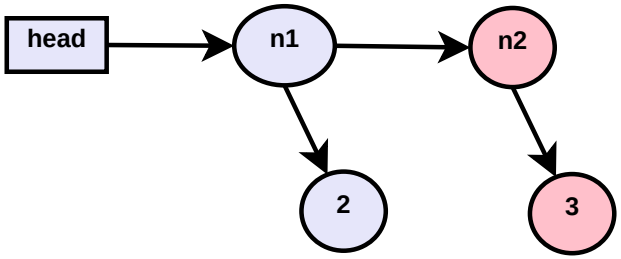
head

Specification collection: The debug phase

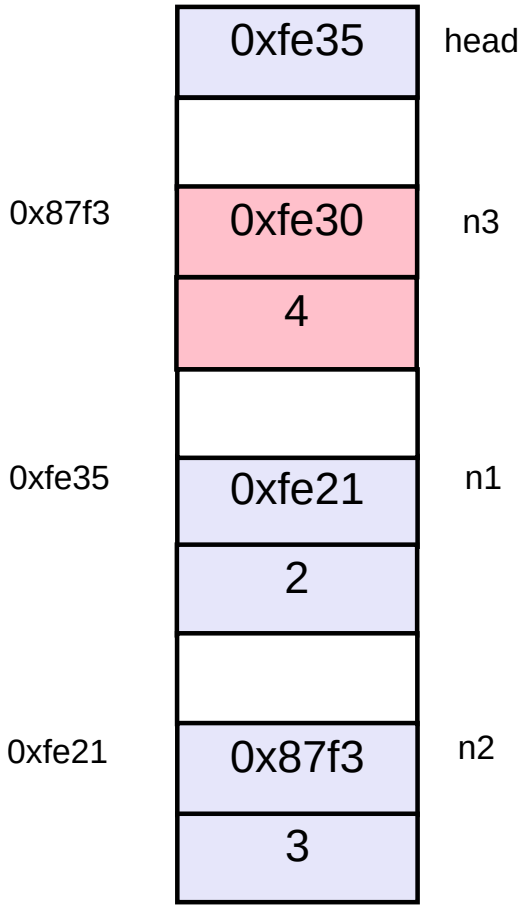


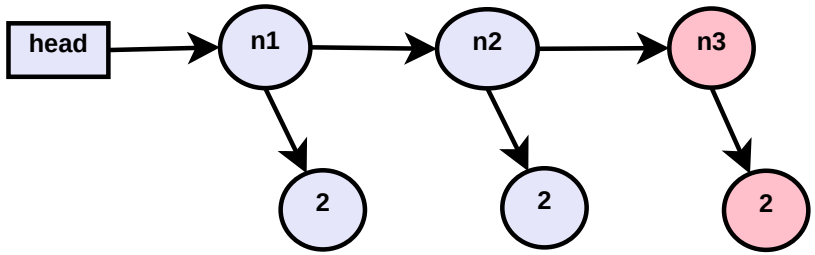
Specification collection: The debug phase





Specification collection: The debug phase





Repair Machinery

- Statement semantics for synthesis of corrections [SAS'13]
- Search for repairs [a new proof-guided methodology]

Correction synthesis: The repair phase

```
struct node *head;

void reverse() {
  (true) ?  current = head;

  (insert)
  while (current != NULL) {
    (true) ?  temp1 = current->prev;
    (true) ?  temp2 = current->next;
    (true) ?  current->prev = temp2;
    (true) ?  current->next = temp1;
    (true) ?  current = current->prev;
    (insert)
  }

  // FIX: head = temp1->prev;
  (insert)
}
```

Correction synthesis: The repair phase

```
struct node *head;

void reverse() {
  (true) ?  current = head;

  (insert)
  while (current != NULL) {
    (true) ?  temp1 = current->prev;
    (true) ?  temp2 = current->next;
    (true) ?  current->prev = temp2;
    (true) ?  current->next = temp1;
    (true) ?  current = current->prev;
    (insert)
  }

  // FIX: head = temp1->prev;
  (insert)
}
```

Correction synthesis: The repair phase

```
struct node *head;

void reverse() {
  (true) ?  current = head;

  (insert)
  while (current != NULL) {
    (true) ?  temp1 = current->prev;
    (true) ?  temp2 = current->next;
    (true) ?  current->prev = temp2;
    (true) ?  current->next = temp1;
    (true) ?  current = current->prev;
    (insert)
  }

  // FIX: head = temp1->prev;
  (insert)
}
```

Correction synthesis: The repair phase

```
struct node *head;

void reverse() {
  (true) ?  current = head;

  (insert)
  while (current != NULL) {
    (true) ?  temp1 = current->prev;
    (true) ?  temp2 = current->next;
    (true) ?  current->prev = temp2;
    (true) ?  current->next = temp1;
    (true) ?  current = current->prev;
    (insert)
  }

  // FIX: head = temp1->prev;
  (insert)
}
```

Correction synthesis: The repair phase

```
struct node *head;

void reverse() {
  (true) ? current = head;

  (insert)
  while (current != NULL) {
    (true) ? temp1 = current->prev;
    (true) ? temp2 = current->next;
    (true) ? current->prev = temp2;
    (true) ? current->next = temp1;
    (true) ? current = current->prev;
    (insert)
  }

  // FIX: head = temp1->prev;
  (insert)
}
```

Correction synthesis: The repair phase

```
struct node *head;

void reverse() {
  (true) ? current = head;

  (insert)
  while (current != NULL) {
    (true) ? temp1 = current->prev;
    (true) ? temp2 = current->next;
    (true) ? current->prev = temp2;
    (true) ? current->next = temp1;
    (true) ? current = current->prev;
    (insert)
  }

  // FIX: head = temp1->prev;
  (insert)
}
```

Correction synthesis: The repair phase

```
struct node *head;

void reverse() {
  (true) ? current = head;

  (insert)
  while (current != NULL) {
    (true) ? temp1 = current->prev;
    (true) ? temp2 = current->next;
    (true) ? current->prev = temp2;
    (true) ? current->next = temp1;
    (true) ? current = current->prev;
    (insert)
  }

  // FIX: head = temp1->prev;
  (insert)
}
```

Correction synthesis: The repair phase

```
struct node *head;

void reverse() {
  (true) ? current = head;

  (insert)
  while (current != NULL) {
    (true) ? temp1 = current->prev;
    (true) ? temp2 = current->next;
    (true) ? current->prev = temp2;
    (true) ? current->next = temp1;
    (true) ? current = current->prev;
    (insert)
  }

  // FIX: head = temp1->prev;
  (insert)
}
```

Correction synthesis: The repair phase

```
struct node *head;

void reverse() {
  (true) ?  current = head;

  (insert)
  while (current != NULL) {
    (true) ?  temp1 = current->prev;
    (true) ?  temp2 = current->next;
    (true) ?  current->prev = temp2;
    (true) ?  current->next = temp1;
    (true) ?  current = current->prev;
    (insert)
  }

  // FIX: head = temp1->prev;
  (insert)
}
```

Correction synthesis: The repair phase

```
struct node *head;

void reverse() {
  (true) ?  current = head;

  (insert)
  while (current != NULL) {
    (true) ?  temp1 = current->prev;
    (true) ?  temp2 = current->next;
    (true) ?  current->prev = temp2;
    (true) ?  current->next = temp1;
    (true) ?  current = current->prev;
    (insert)
  }

  // FIX: head = temp1->prev;
  head = temp1->prev;
}
```

Taming the unsat core

- Unsat core is far from being the minimal unsat core!

Taming the unsat core

- Unsat core is far from being the minimal unsat core!
- A direct consumption of underapproximation widening does not work well!

Taming the unsat core

- Unsat core is far from being the minimal unsat core!
- A direct consumption of underapproximation widening does not work well!

Primary algorithm

Increase the *number of mutations* at every round of underapproximation widening!

Taming the unsat core

- Unsat core is far from being the minimal unsat core!
- A direct consumption of underapproximation widening does not work well!

Primary algorithm

Increase the *number of mutations* at every round of underapproximation widening!

Is able to repair all our benchmarks in reasonable time!

Plan

Evaluation

RQ1 Is our repair algorithm able to fix different types and combination of bugs in a variety of data-structures?

RQ2 Can our repair algorithm fix these bugs in reasonable time?

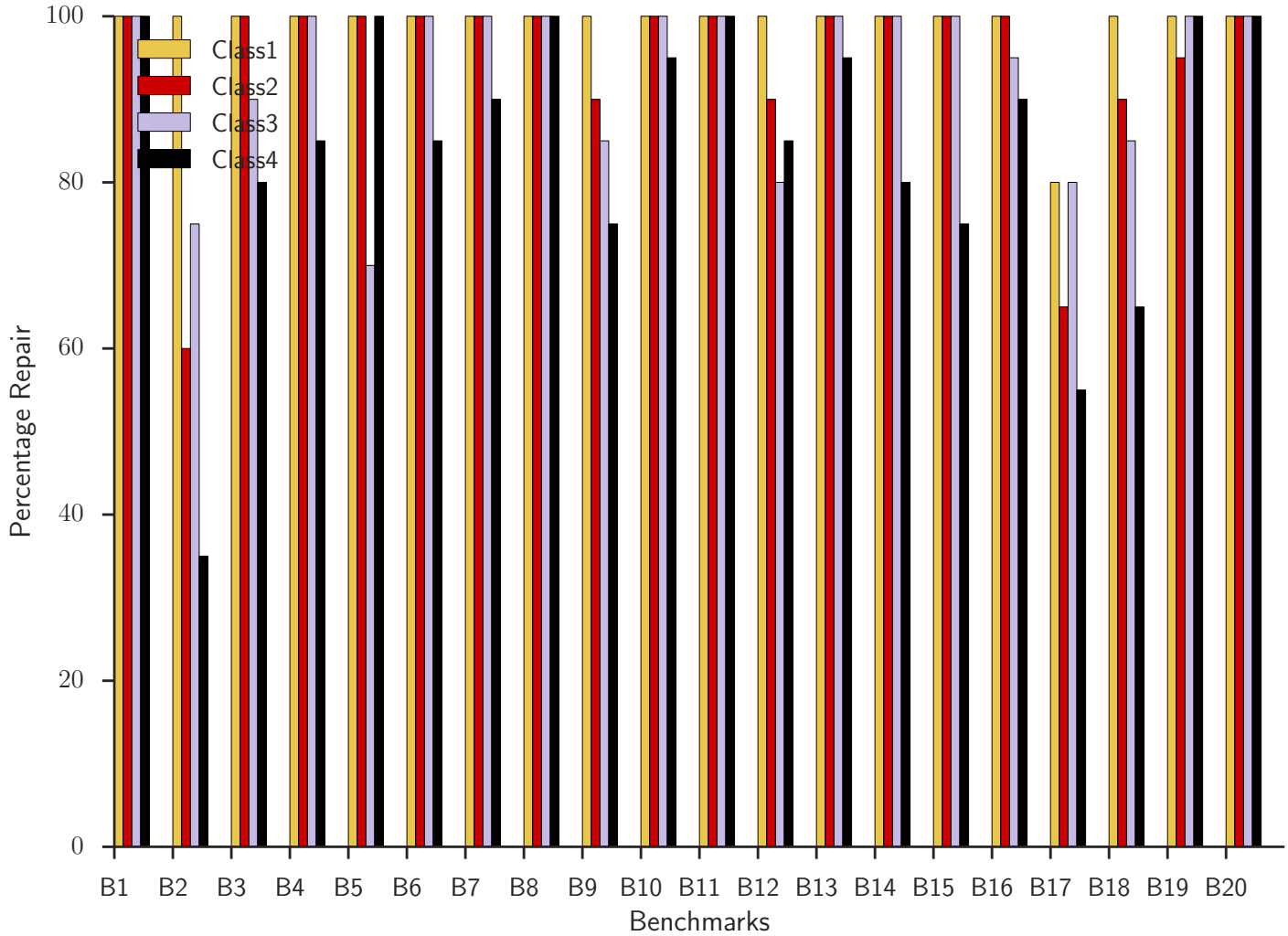
RQ3 How does our repair algorithm scale as the number of bugs are increased?

RQ4 Is WOLVERINE capable of debugging/fixing real bugs?

Evaluation benchmarks

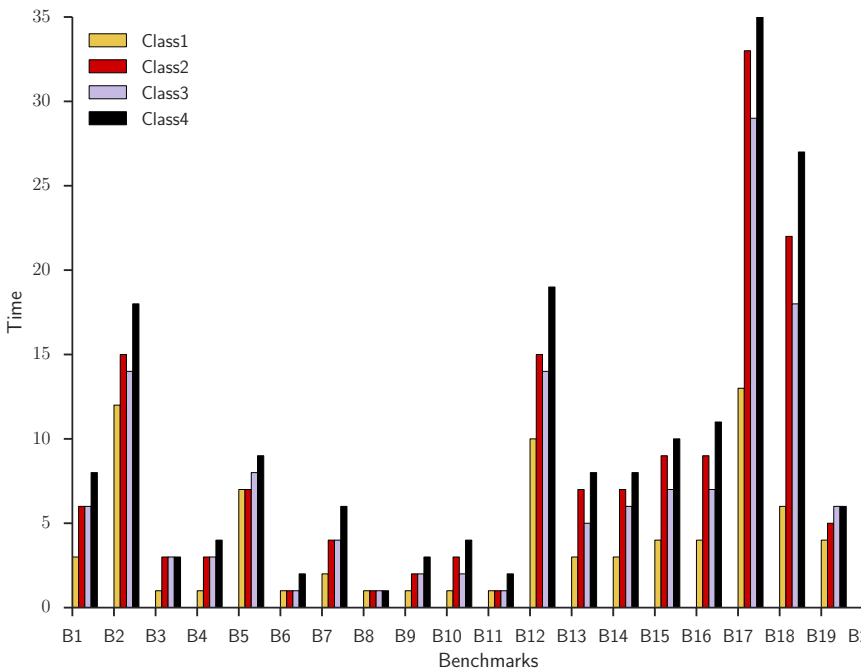
- Set of data-structure benchmarks collected from online sources:
  1. We evaluate each benchmark for four bug classes: Class1 ($\langle 1, 0 \rangle$), Class2($\langle 1, 1 \rangle$), Class3($\langle 2, 0 \rangle$) and Class4($\langle 2, 1 \rangle$);
  2. For each benchmark B_i, at each bug configuration $\langle x, y \rangle$, we run our fault injection engine to create *20 buggy versions* with $\mathtt{x}$ errors that require modification of an IR instruction and $\mathtt{y}$ errors that require insertion of a new statement;
  3. Each of the above buggy program is run twice to amortize the run time variability.
- Set of 247 buggy submissions from students corresponding to 5 programming problems on heap manipulations were collected.

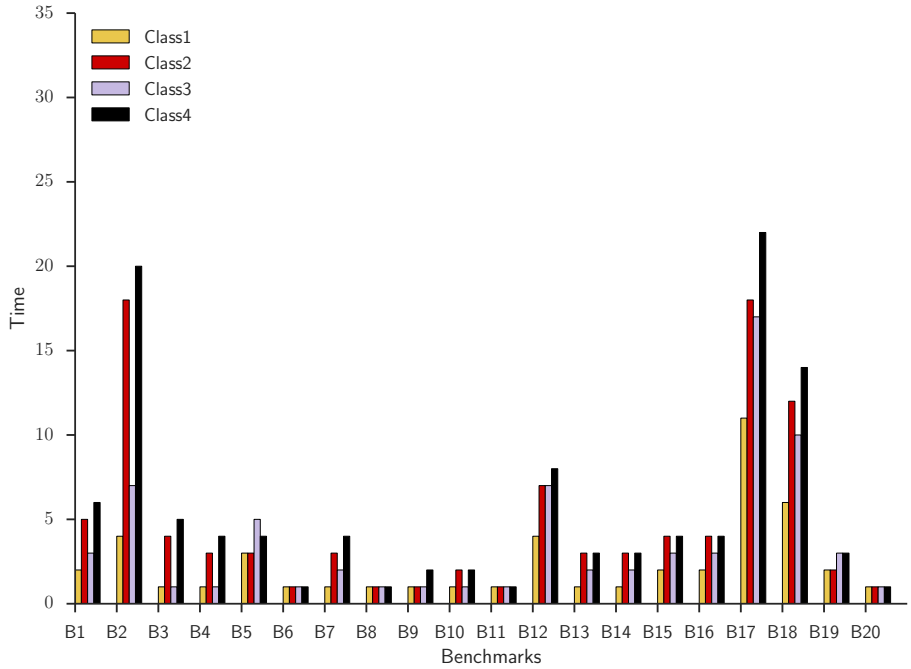
Repair Rate



Our primary algorithm is able to repair all our benchmark programs!

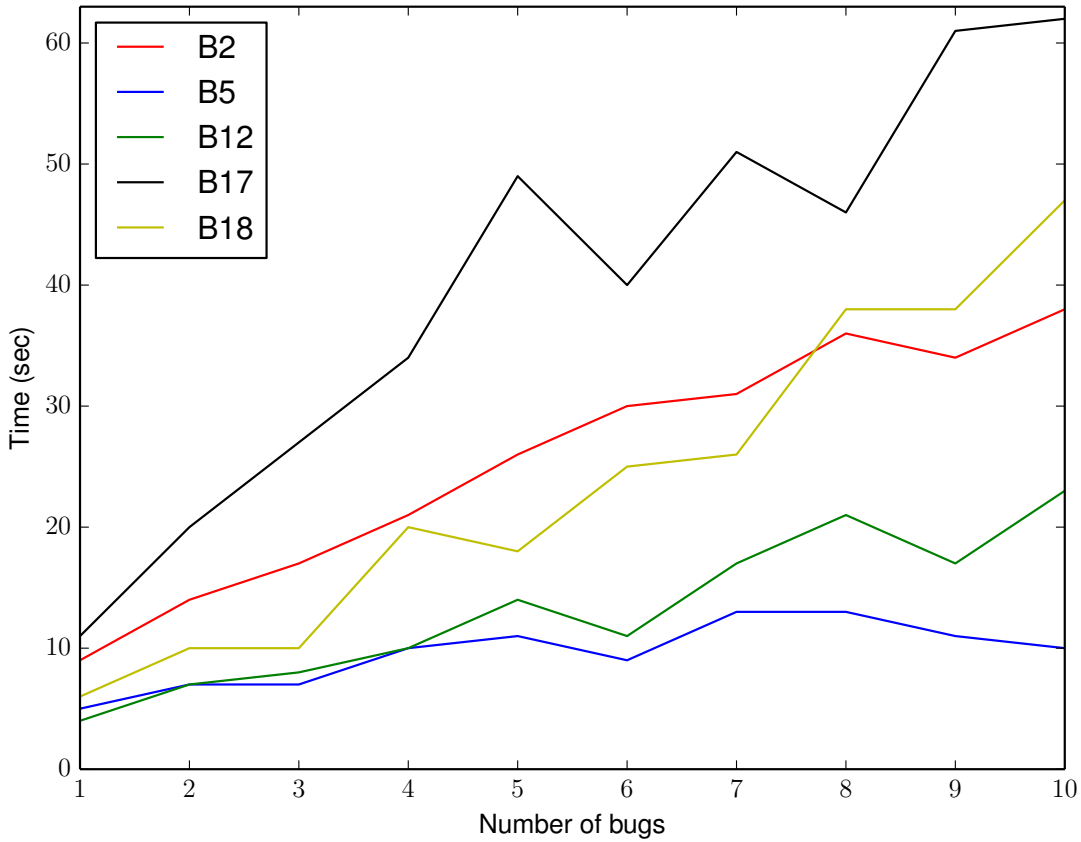
Repair Times





The repair time is mostly under 5s for most of the benchmarks.

Scalability with number of bugs



The time increases about linearly with number of bugs through the search space increases exponentially.

Evaluation on student programs

Id	Total	Fixed	ImLmt	OoScope	Vacuous
S1	47	30	2	8	7
S2	48	29	3	8	8
S3	48	36	0	5	7
S4	61	46	0	6	9
S5	43	25	0	4	14

Overall, we could repair more than 80% of the submissions automatically where the student has made some attempt at the problem (i.e. barring the vacuous cases).

Plan

Advanced debugging: Specification refinement

```
void bar(){
  struct node *current= NULL; int i;
  current = head;
  while (current != NULL){
    i = foo();
    current->data = i;
    current = current->next;
    }
  }
```

Specification refinement

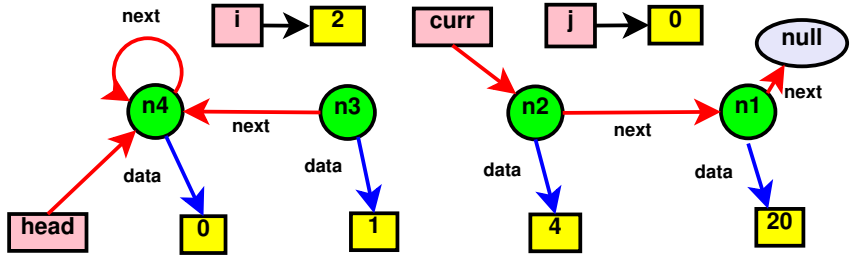
```
void bar(){
  struct node *current= NULL; int i;
  current = head;
  while (current != NULL){
    concrete[i = foo();]
    current->data = i;
    current = current->next;
    }
  }
```

Advanced debugging: Specification slicing

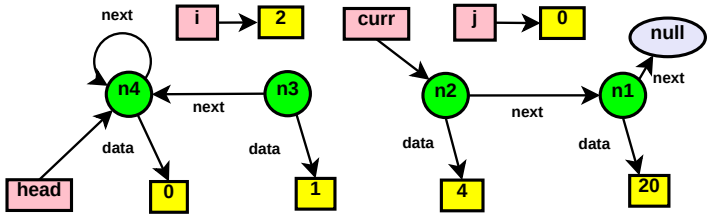
```
void reverse(int i){
  struct node * last, *current;
  struct node *nt = NULL;
  current = head;
  while (current != NULL){
    nt = current->next;
    current->next = prev;
    prev = current;
    // FIX1:  current->data = i
    current->data = j;
    concrete[i = i+1;];
    // FIX2:  current = prev;
    current = nt;
  }
  // FIX3:  head = prev;
}
```

Advanced debugging: Specification slicing

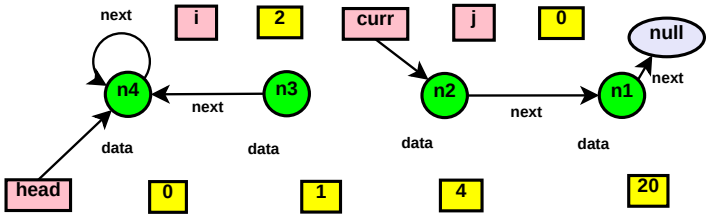
```
void reverse(int i){
  struct node * last, *current;
  struct node *nt = NULL;
  current = head;
  while (current != NULL){
    nt = current->next;
    current->next = prev;
    prev = current;
    // FIX1:  current->data = i
    current->data = j;
    concrete[i = i+1;];
    // FIX2:  current = prev;
    current = nt;
  }
  // FIX3:  head = prev;
}
```



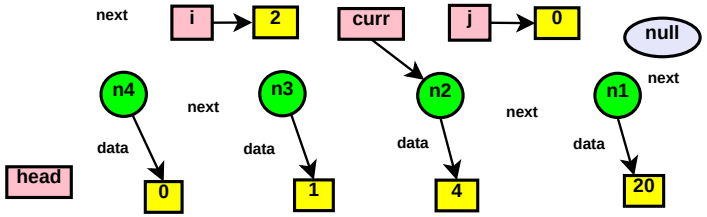
Advanced debugging: Specification slicing



Repair time = 6.5s



Repair time = 3.0s



Repair time = 1.5s

Plan

Conclusions

We believe that tighter integration of dynamic analysis (possibly enabled by a debugger) and static analysis (via symbolic techniques) can open new avenues for debugging tools.

Acknowledgements

We thank Google for supporting the travel of the first author.

Questions

Questions?

Paragraphs of Text

Sed iaculis dapibus gravida. Morbi sed tortor erat, nec interdum arcu. Sed id lorem lectus. Quisque viverra augue id sem ornare non aliquam nibh tristique. Aenean in ligula nisl. Nulla sed tellus ipsum. Donec vestibulum ligula non lorem vulputate fermentum accumsan neque mollis.

Sed diam enim, sagittis nec condimentum sit amet, ullamcorper sit amet libero. Aliquam vel dui orci, a porta odio. Nullam id suscipit ipsum. Aenean lobortis commodo sem, ut commodo leo gravida vitae. Pellentesque vehicula ante iaculis arcu pretium rutrum eget sit amet purus. Integer ornare nulla quis neque ultrices lobortis. Vestibulum ultrices tincidunt libero, quis commodo erat ullamcorper id.

Bullet Points

- Lorem ipsum dolor sit amet, consectetur adipiscing elit
- Aliquam blandit faucibus nisi, sit amet dapibus enim tempus eu
- Nulla commodo, erat quis gravida posuere, elit lacus lobortis est, quis porttitor odio mauris at libero
- Nam cursus est eget velit posuere pellentesque
- Vestibulum faucibus velit a augue condimentum quis convallis nulla gravida

Blocks of Highlighted Text

Block 1

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer lectus nisl, ultricies in feugiat rutrum, porttitor sit amet augue. Aliquam ut tortor mauris. Sed volutpat ante purus, quis accumsan dolor.

Block 2

Pellentesque sed tellus purus. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Vestibulum quis magna at risus dictum tempor eu vitae velit.

Block 3

Suspendisse tincidunt sagittis gravida. Curabitur condimentum, enim sed venenatis rutrum, ipsum neque consectetur orci, sed blandit justo nisi ac lacus.

Multiple Columns

Heading

1. Statement
2. Explanation
3. Example

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer lectus nisl, ultricies in feugiat rutrum, porttitor sit amet augue. Aliquam ut tortor mauris. Sed volutpat ante purus, quis accumsan dolor.

Table

Treatments	Response 1	Response 2
Treatment 1	0.0003262	0.562
Treatment 2	0.0015681	0.910
Treatment 3	0.0009271	0.296

Table: Table caption

Theorem

Theorem (Mass–energy equivalence)

$E = mc^2$

Verbatim

Example (Theorem Slide Code)

```
\begin{frame}
\frametitle{Theorem}
\begin{theorem}[Mass--energy equivalence]
$E = mc^2$
\end{theorem}
\end{frame}
```

Figure

Uncomment the code on this slide to include your own image from the same directory as the template .TeX file.

Citation

An example of the `\cite` command to cite within the presentation:

This statement requires citation [Smith, 2012].

References

John Smith (2012)

Title of the publication

Journal Name 12(3), 45 – 678.

The End