Testing for Buffer Overflows by Length-Driven Symbolic Execution

Zhang Qirun,
Department of Computer Science and Engineering,
The Chinese University of Hong Kong

November 13, 2009

Outline

Outline

The Essentials

- The Definition

The Essentials

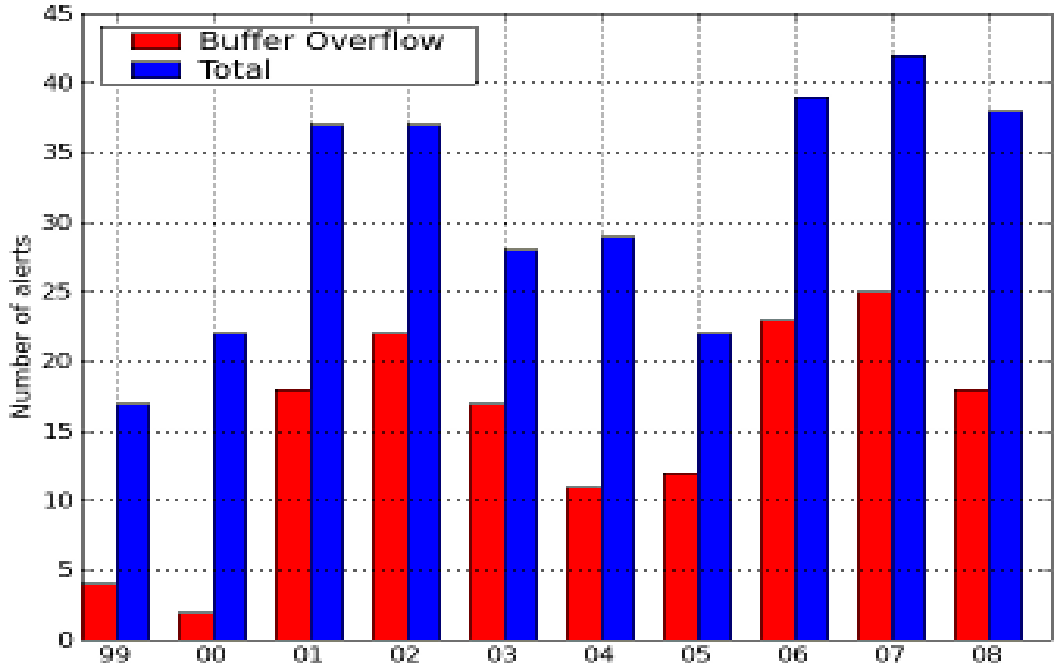
- The Definition
- Problem with C/C++

The Essentials

- The Definition
- Problem with C/C++
- US-CERT (United States Computer Emergency Readiness Team)

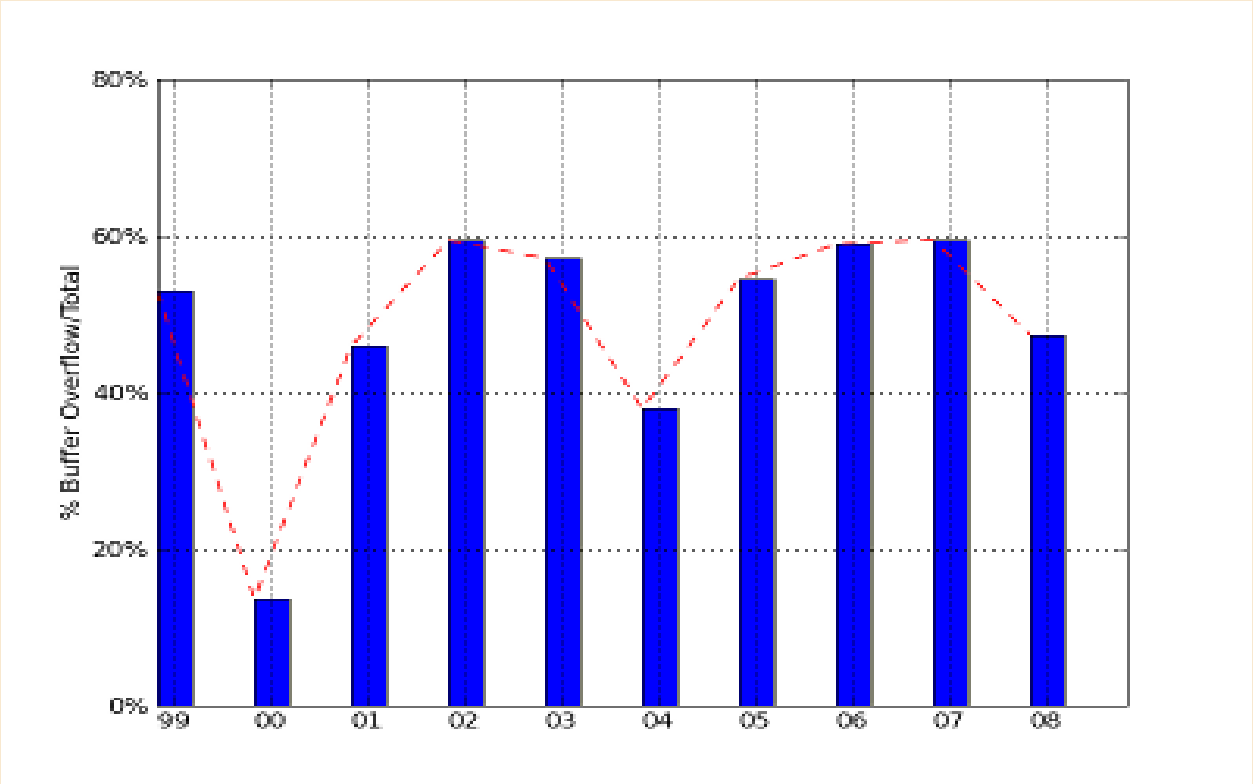
Why So Important?

- Number of Reported CERT Security Advisories



Why So Important?

- Precentage of reported Buffer Overflow



Two Kinds of Buffer Overflow

- Stack Overflow

Two Kinds of Buffer Overflow

- Stack Overflow
- Heap Corruption

The power of Buffer Overflow

- An example...

Fighting with Buffer Overflow

- Choice of programming language

Fighting with Buffer Overflow

- Choice of programming language
- Use of safe libraries

Fighting with Buffer Overflow

- Choice of programming language
- Use of safe libraries
- Buffer overflow protection(Dynamic and Static) example...

Fighting with Buffer Overflow

- Choice of programming language
- Use of safe libraries
- Buffer overflow protection(Dynamic and Static) example...
- Pointer protection

Fighting with Buffer Overflow

- Choice of programming language
- Use of safe libraries
- Buffer overflow protection(Dynamic and Static) example...
- Pointer protection
- Address space layout randomization

What We want to do...

- We focus on...

What's Symbolic Execution

- What's Symbolic Execution

What's Symbolic Execution

- What's Symbolic Execution
- Advantages

What's Symbolic Execution

- What's Symbolic Execution
- Advantages
- Disadvantages

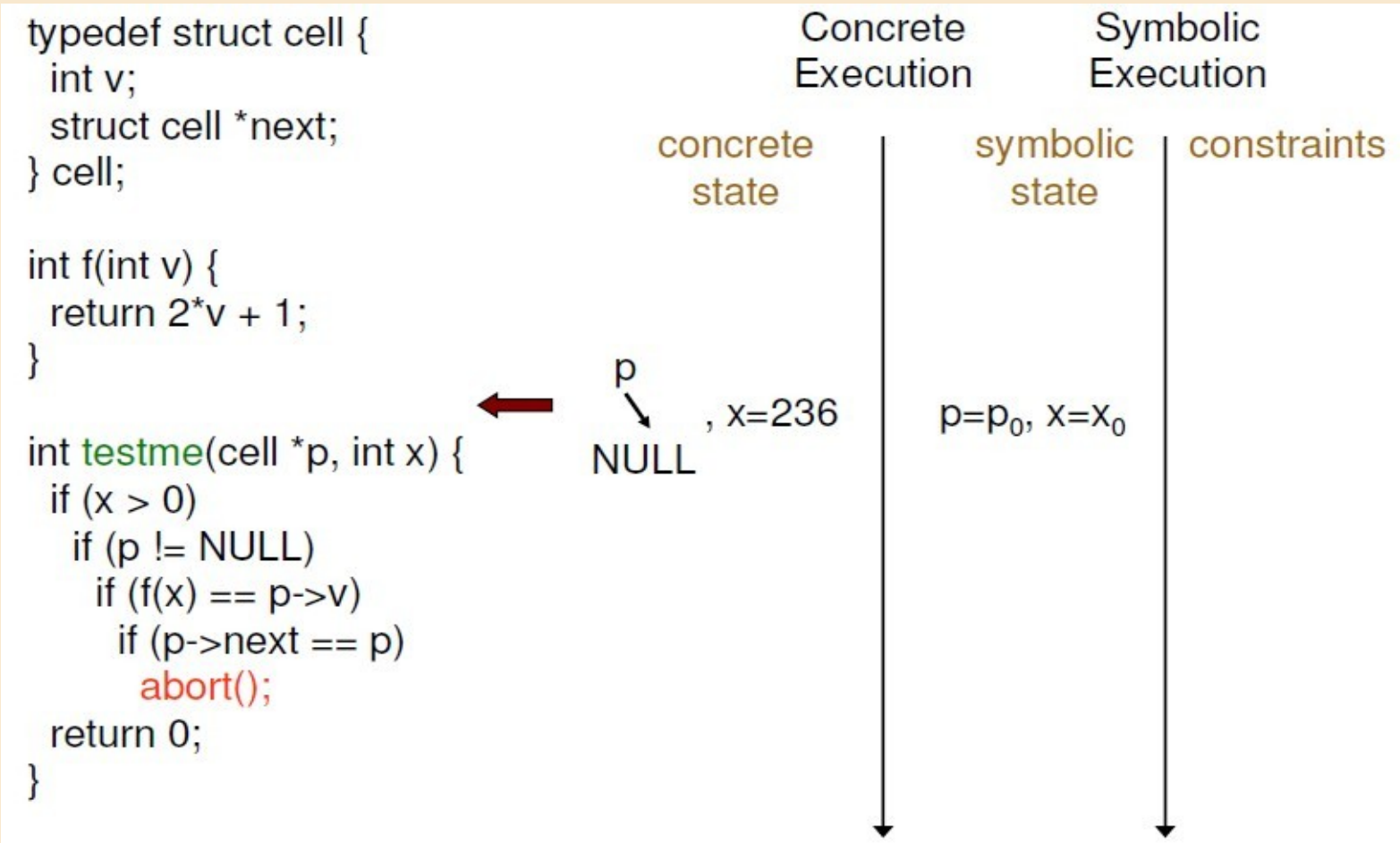
An example

```
typedef struct cell {
  int v;
  struct cell *next;
} cell;

int f(int v) {
  return 2*v + 1;
}

int testme(cell *p, int x) {
  if (x > 0)
    if (p != NULL)
      if (f(x) == p->v)
        if (p->next == p)
          abort();
  return 0;
}
```

- Random Test Driver:
  - random memory graph reachable from p
  - random value for x
- Probability of reaching abort( ) is extremely low

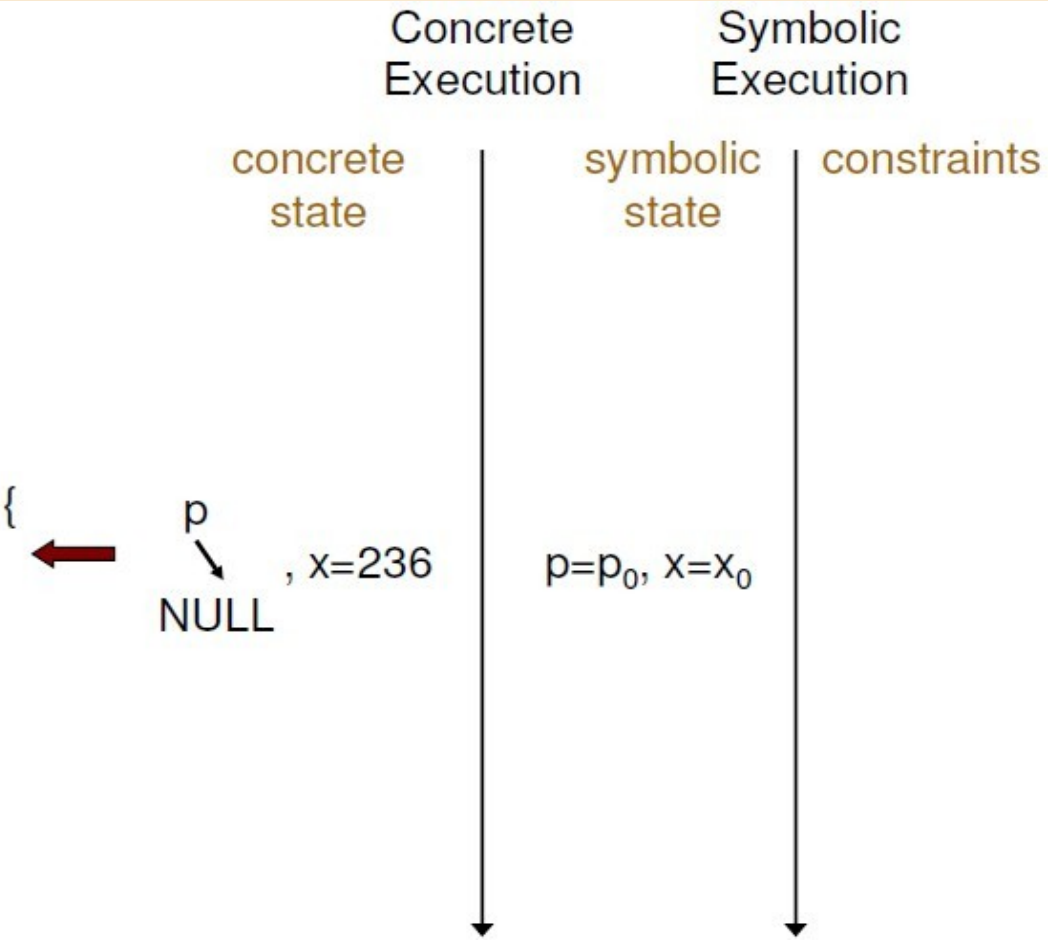
An example

```
typedef struct cell {
  int v;
  struct cell *next;
} cell;

int f(int v) {
  return 2*v + 1;
}

int testme(cell *p, int x) {
  if (x > 0)
    if (p != NULL)
      if (f(x) == p->v)
        if (p->next == p)
          abort();
  return 0;
}
```

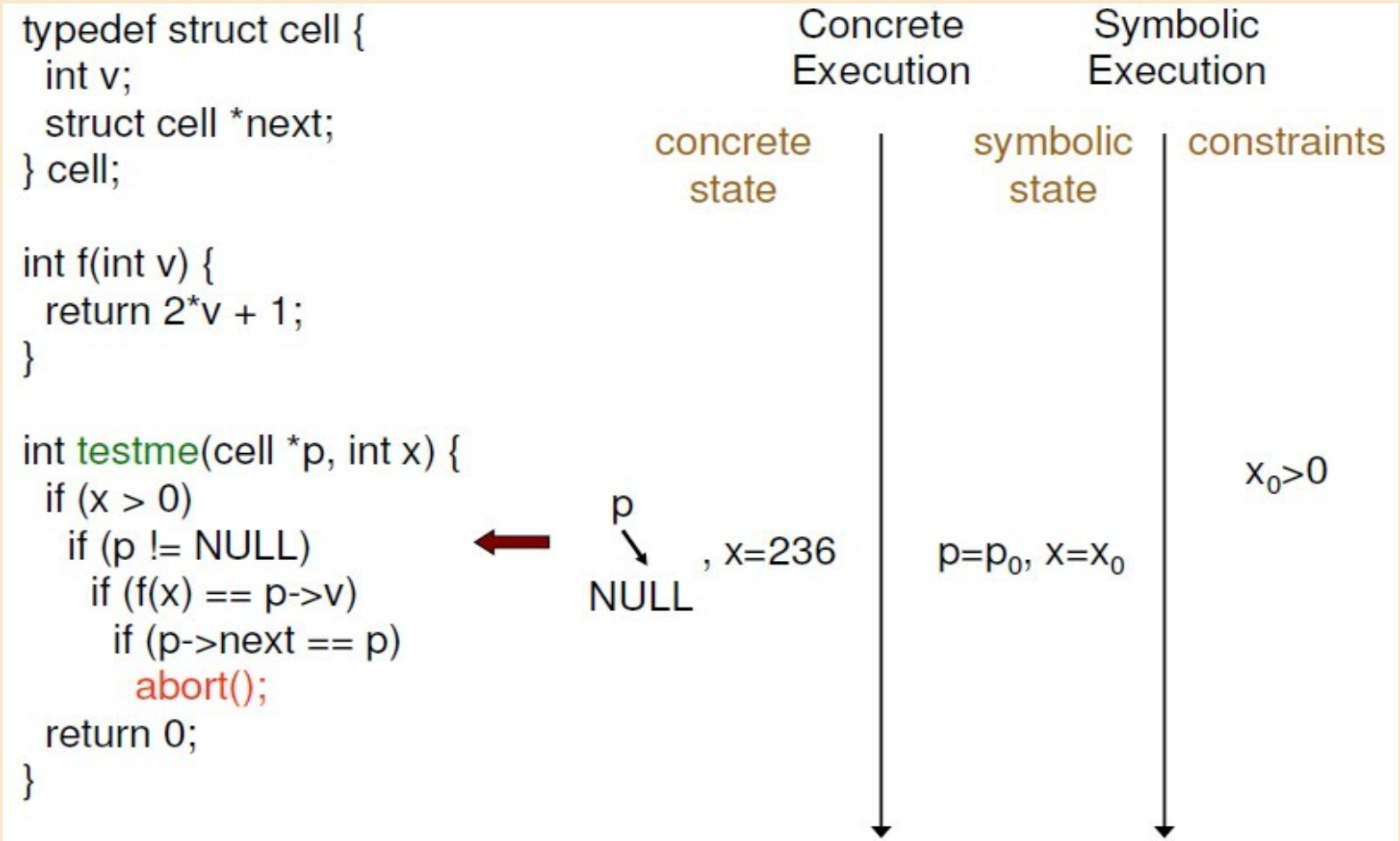


An example

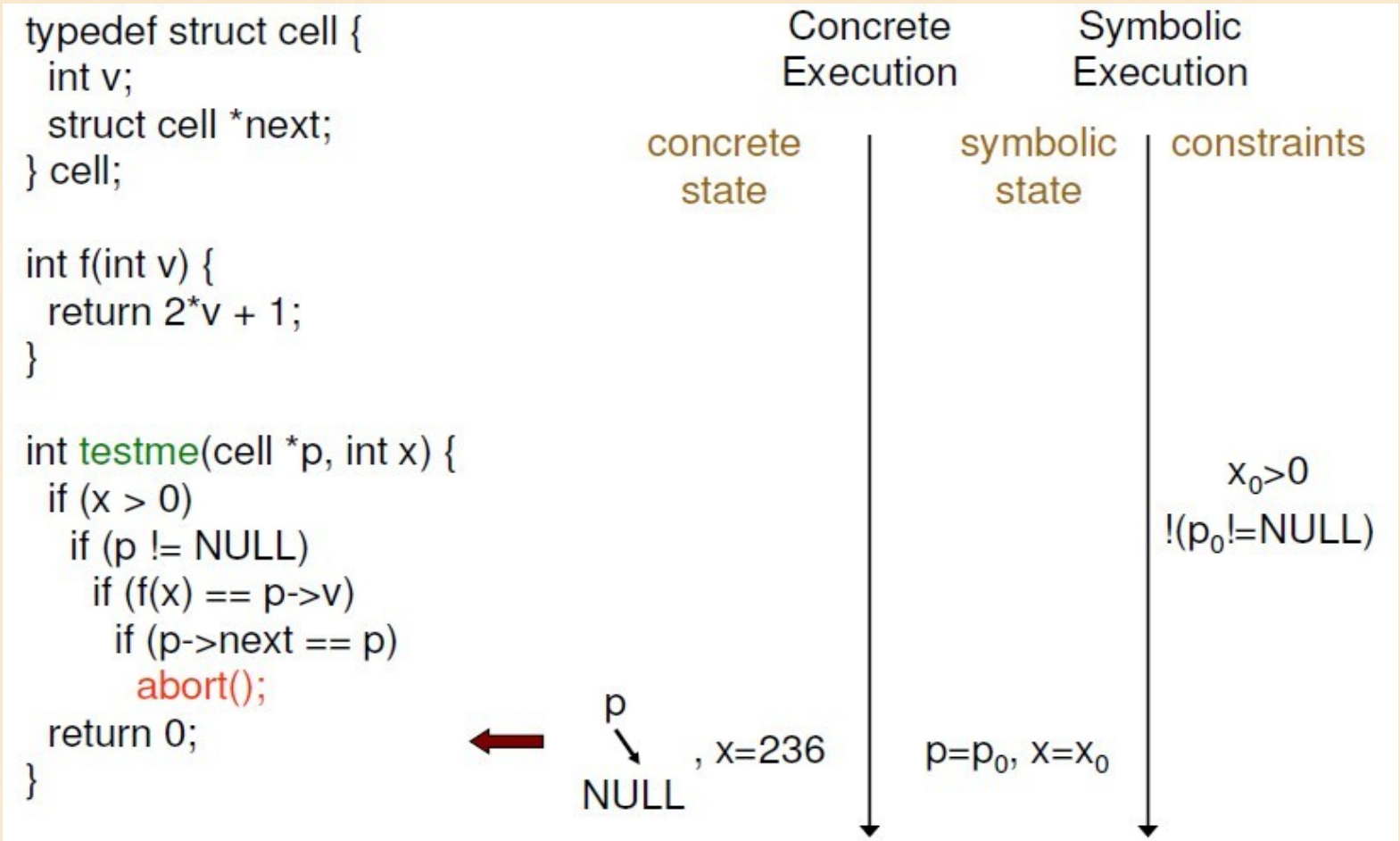
```
typedef struct cell {
  int v;
  struct cell *next;
} cell;

int f(int v) {
  return 2*v + 1;
}

int testme(cell *p, int x) {
  if (x > 0)
    if (p != NULL)
      if (f(x) == p->v)
        if (p->next == p)
          abort();
  return 0;
}
```



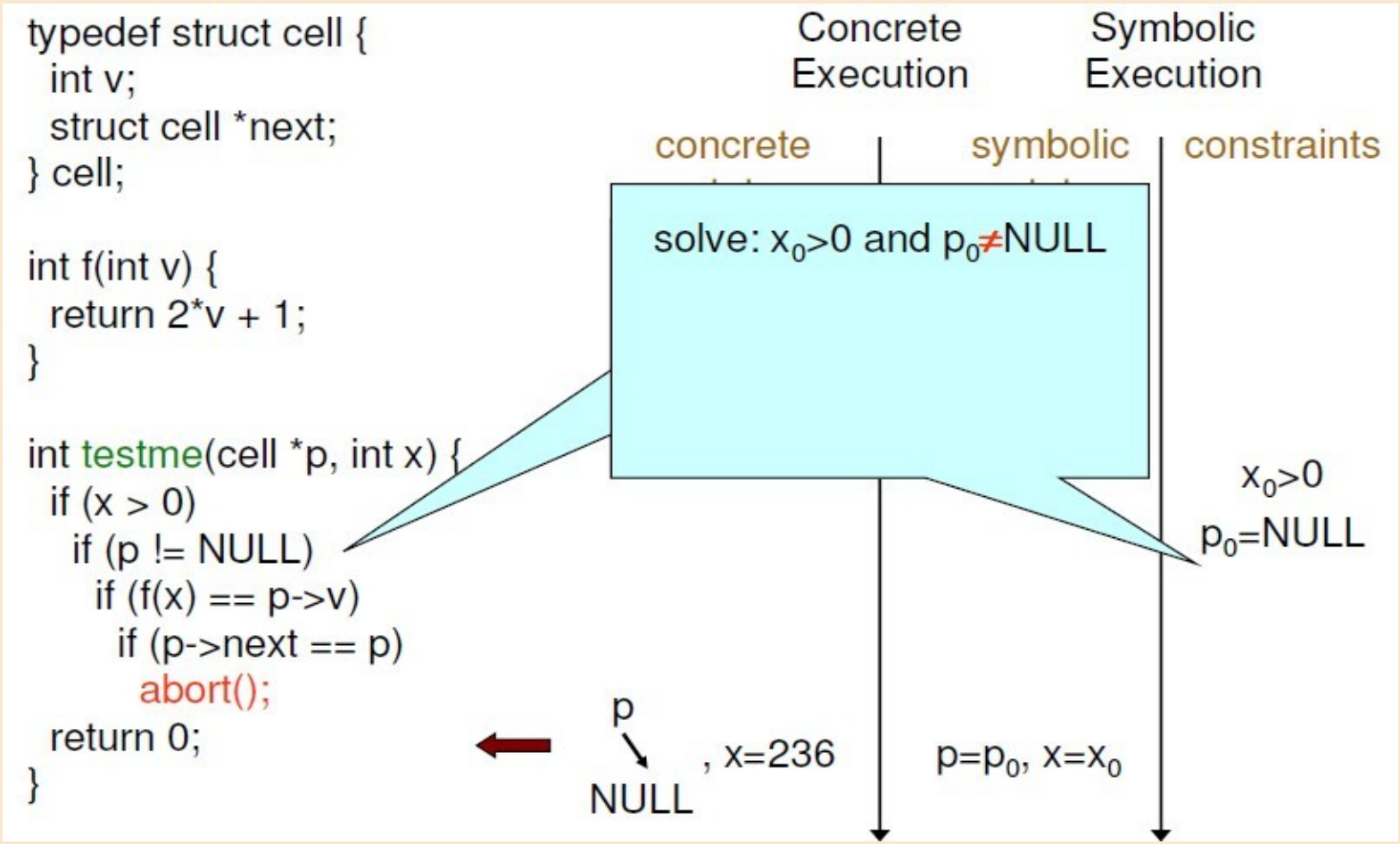
An example


typedef struct cell {
int v;
struct cell *next;
} cell;
int f(int v) {
return 2*v + 1;
}
int testme(cell *p, int x) {
if (x > 0)
if (p != NULL)
if (f(x) == p->v)
if (p->next == p)
abort();
return 0;
}
Concrete Execution
Symbolic Execution
concrete state
symbolic state
constraints
p
NULL
, x=236
p=p0, x=x0
x0>0

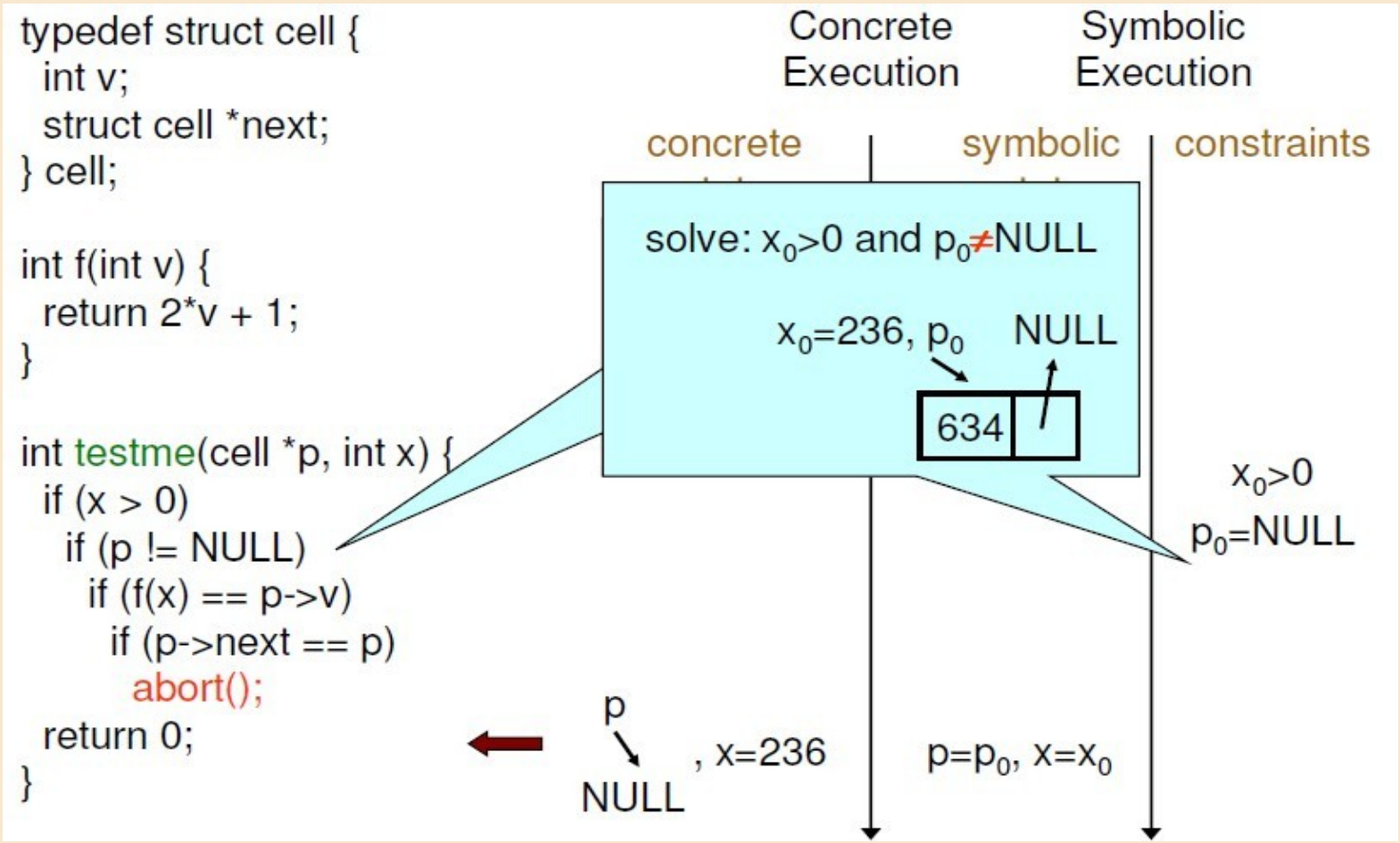
An example


typedef struct cell {
int v;
struct cell *next;
} cell;
int f(int v) {
return 2*v + 1;
}
int testme(cell *p, int x) {
if (x > 0)
if (p != NULL)
if (f(x) == p->v)
if (p->next == p)
abort();
return 0;
}
Concrete Execution
Symbolic Execution
concrete state
symbolic state
constraints
$x_0>0$
$!(p_0!=NULL)$
p
NULL
, x=236
$p=p_0$, $x=x_0$

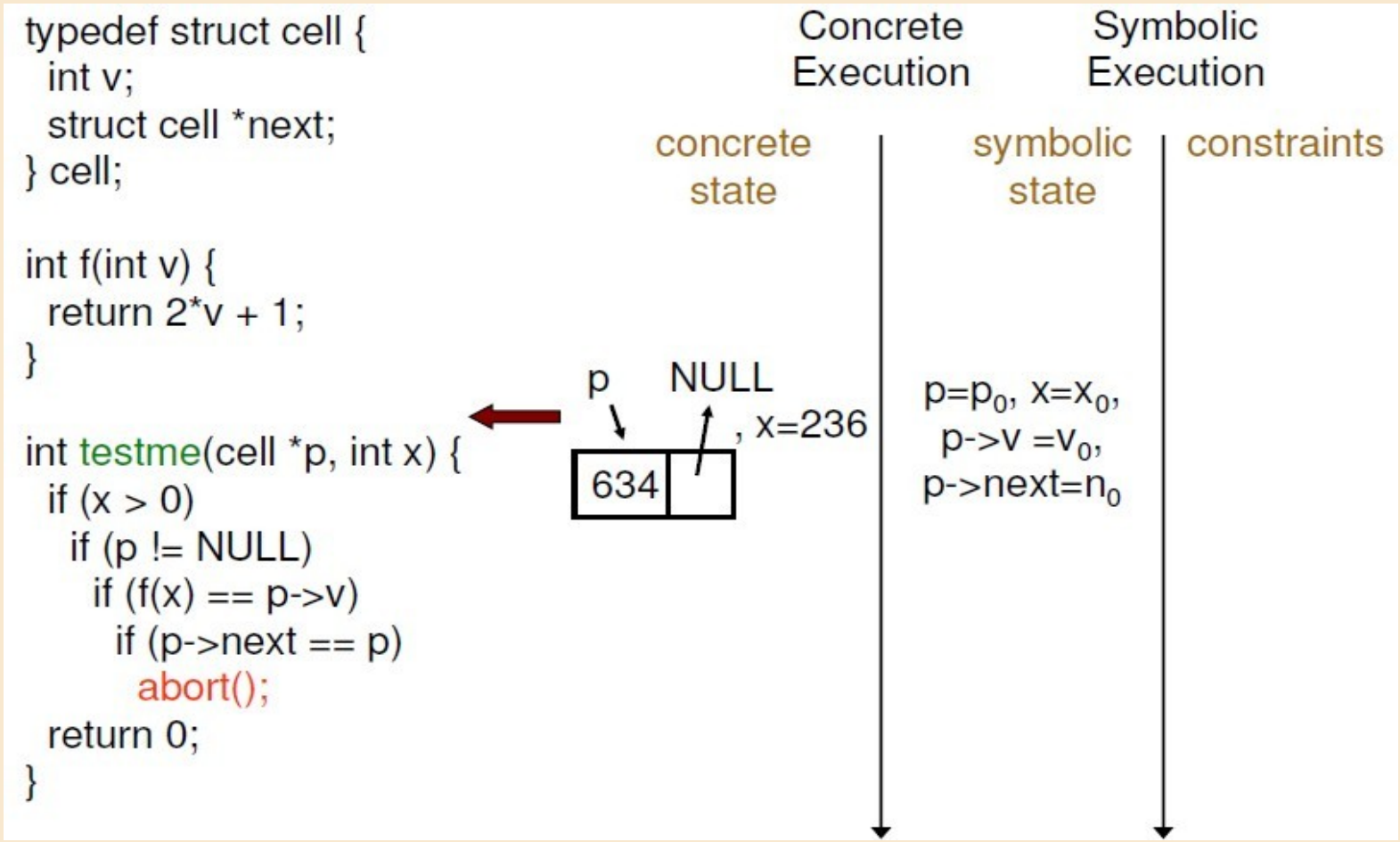
An example


typedef struct cell {
int v;
struct cell *next;
} cell;
int f(int v) {
return 2*v + 1;
}
int testme(cell *p, int x) {
if (x > 0)
if (p != NULL)
if (f(x) == p->v)
if (p->next == p)
abort();
return 0;
}
Concrete Execution
Symbolic Execution
concrete
symbolic
constraints
solve: $x_0>0$ and $p_0 \neq NULL$
$x_0>0$
$p_0=NULL$
p
NULL
, x=236
$p=p_0$, $x=x_0$

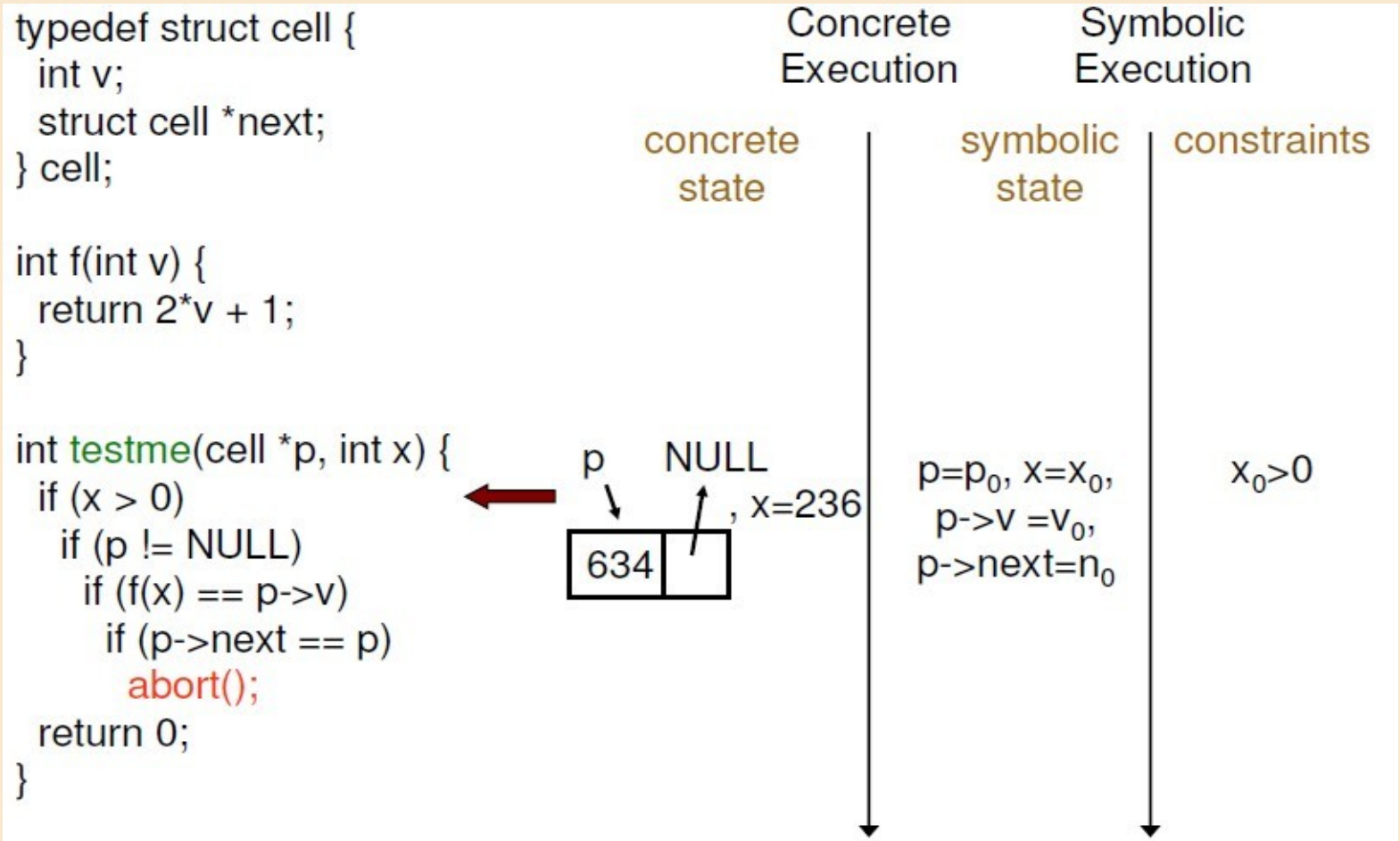
An example


typedef struct cell {
int v;
struct cell *next;
} cell;
int f(int v) {
return 2*v + 1;
}
int testme(cell *p, int x) {
if (x > 0)
if (p != NULL)
if (f(x) == p->v)
if (p->next == p)
abort();
return 0;
}
Concrete Execution
Symbolic Execution
concrete
symbolic
constraints
solve: x0>0 and p0≠NULL
x0=236, p0
NULL
634
x0>0
p0=NULL
p
NULL
, x=236
p=p0, x=x0

An example

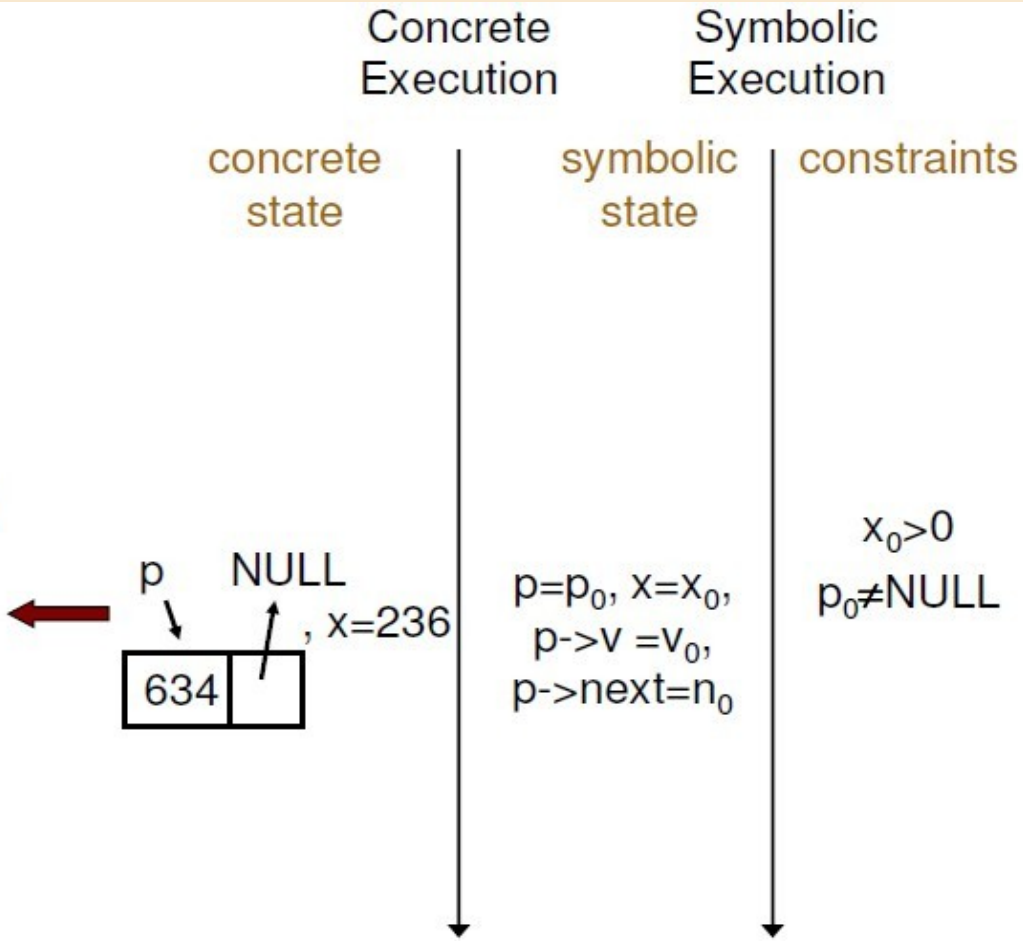

typedef struct cell {
int v;
struct cell *next;
} cell;
int f(int v) {
return 2*v + 1;
}
int testme(cell *p, int x) {
if (x > 0)
if (p != NULL)
if (f(x) == p->v)
if (p->next == p)
abort();
return 0;
}
Concrete Execution
Symbolic Execution
concrete state
symbolic state
constraints
p
NULL
634
, x=236
p=p0, x=x0,
p->v =v0,
p->next=n0

An example


typedef struct cell {
int v;
struct cell *next;
} cell;
int f(int v) {
return 2*v + 1;
}
int testme(cell *p, int x) {
if (x > 0)
if (p != NULL)
if (f(x) == p->v)
if (p->next == p)
abort();
return 0;
}
Concrete Execution
Symbolic Execution
concrete state
symbolic state
constraints
p
NULL
, x=236
634
p=p0, x=x0,
p->v =v0,
p->next=n0
x0>0

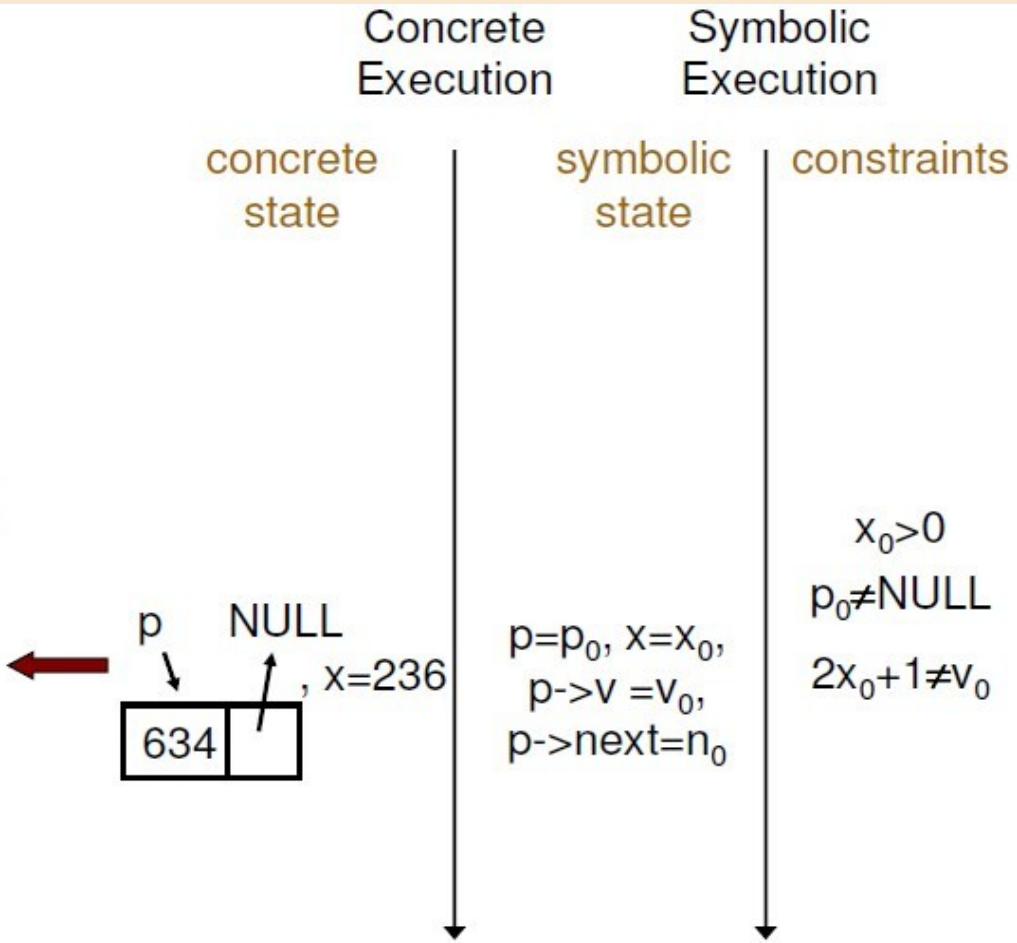
An example

```
typedef struct cell {
  int v;
  struct cell *next;
} cell;

int f(int v) {
  return 2*v + 1;
}

int testme(cell *p, int x) {
  if (x > 0)
    if (p != NULL)
      if (f(x) == p->v)
        if (p->next == p)
          abort();
  return 0;
}
```

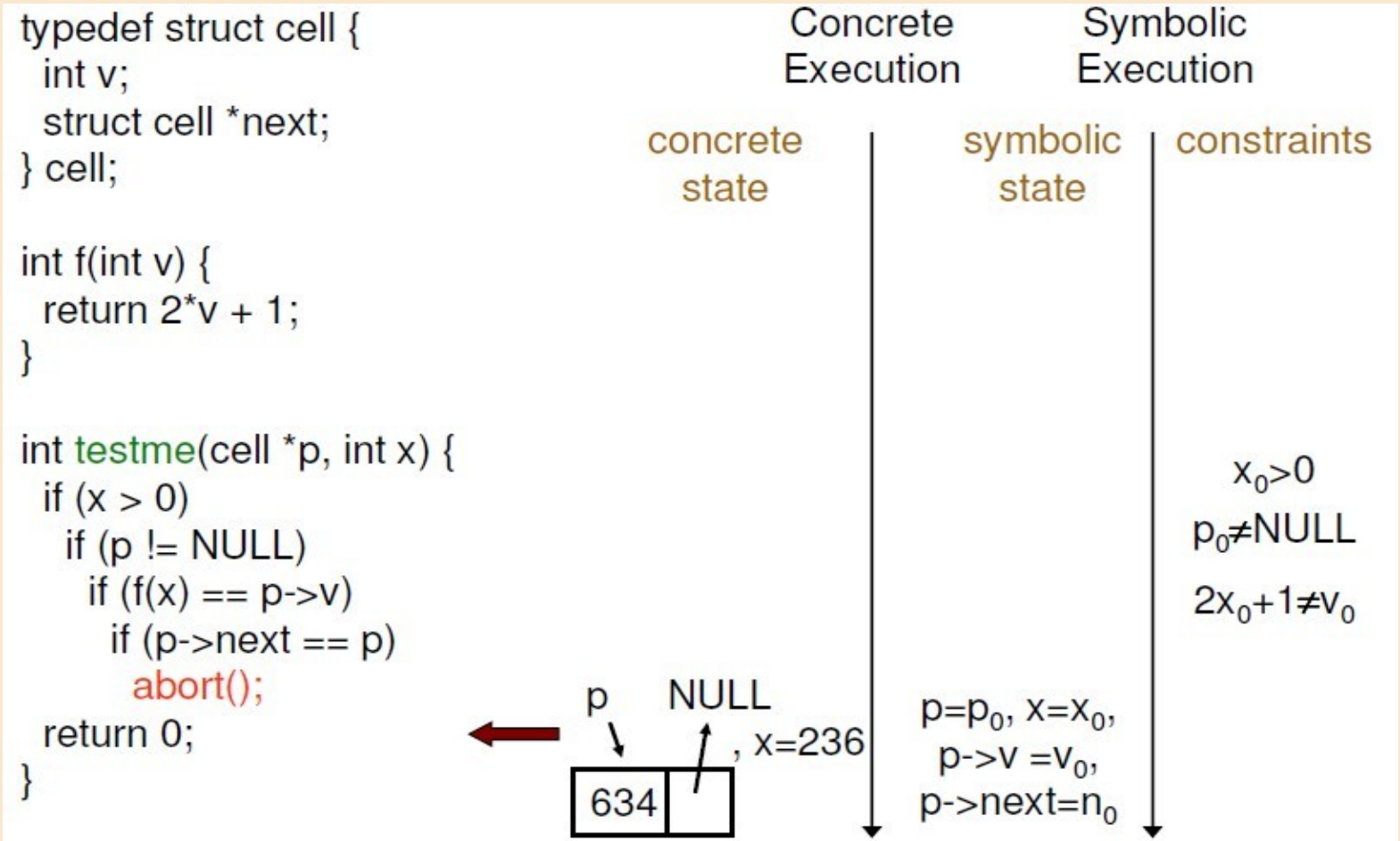


An example

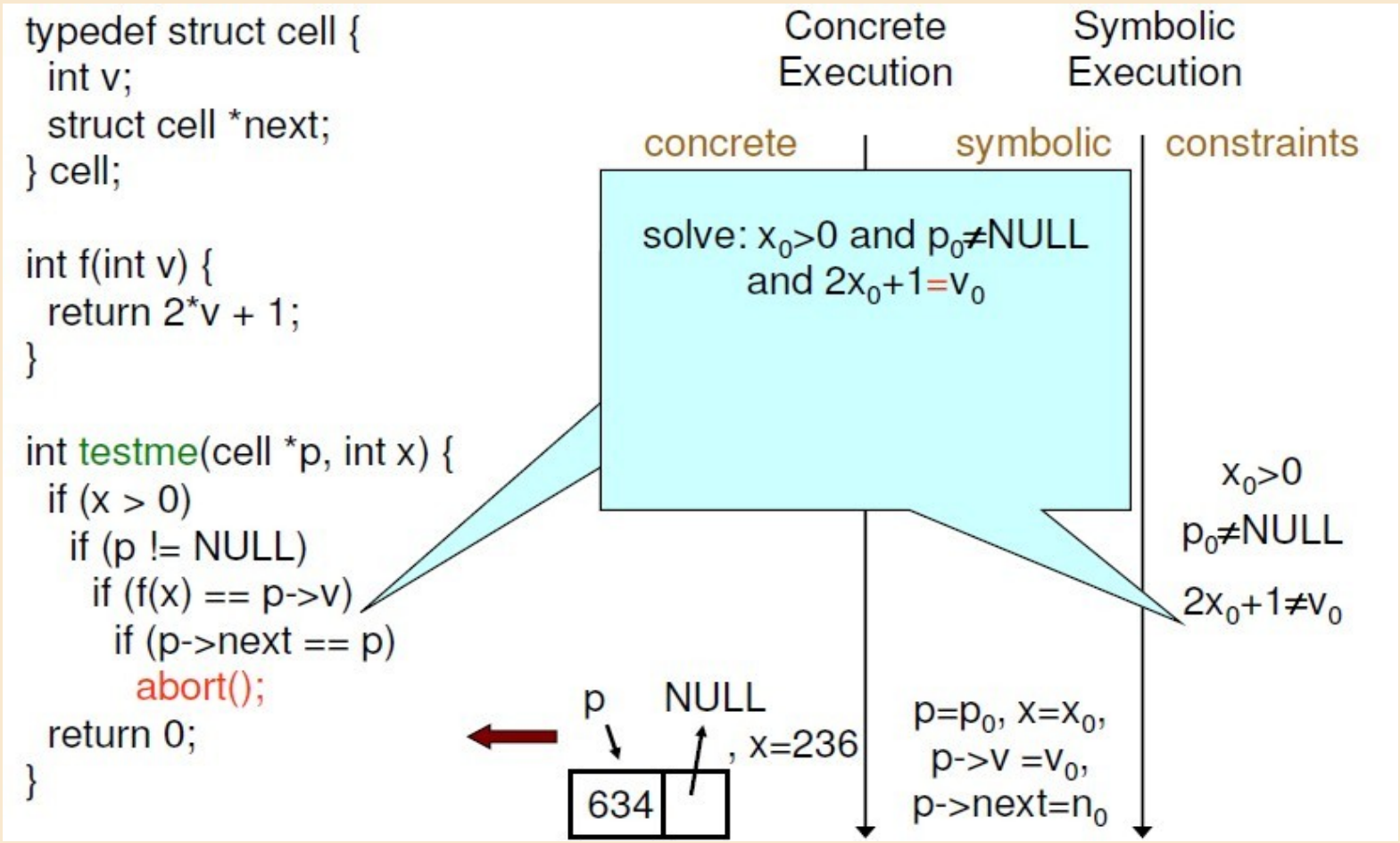
```
typedef struct cell {
  int v;
  struct cell *next;
} cell;

int f(int v) {
  return 2*v + 1;
}

int testme(cell *p, int x) {
  if (x > 0)
    if (p != NULL)
      if (f(x) == p->v)
        if (p->next == p)
          abort();
  return 0;
}
```



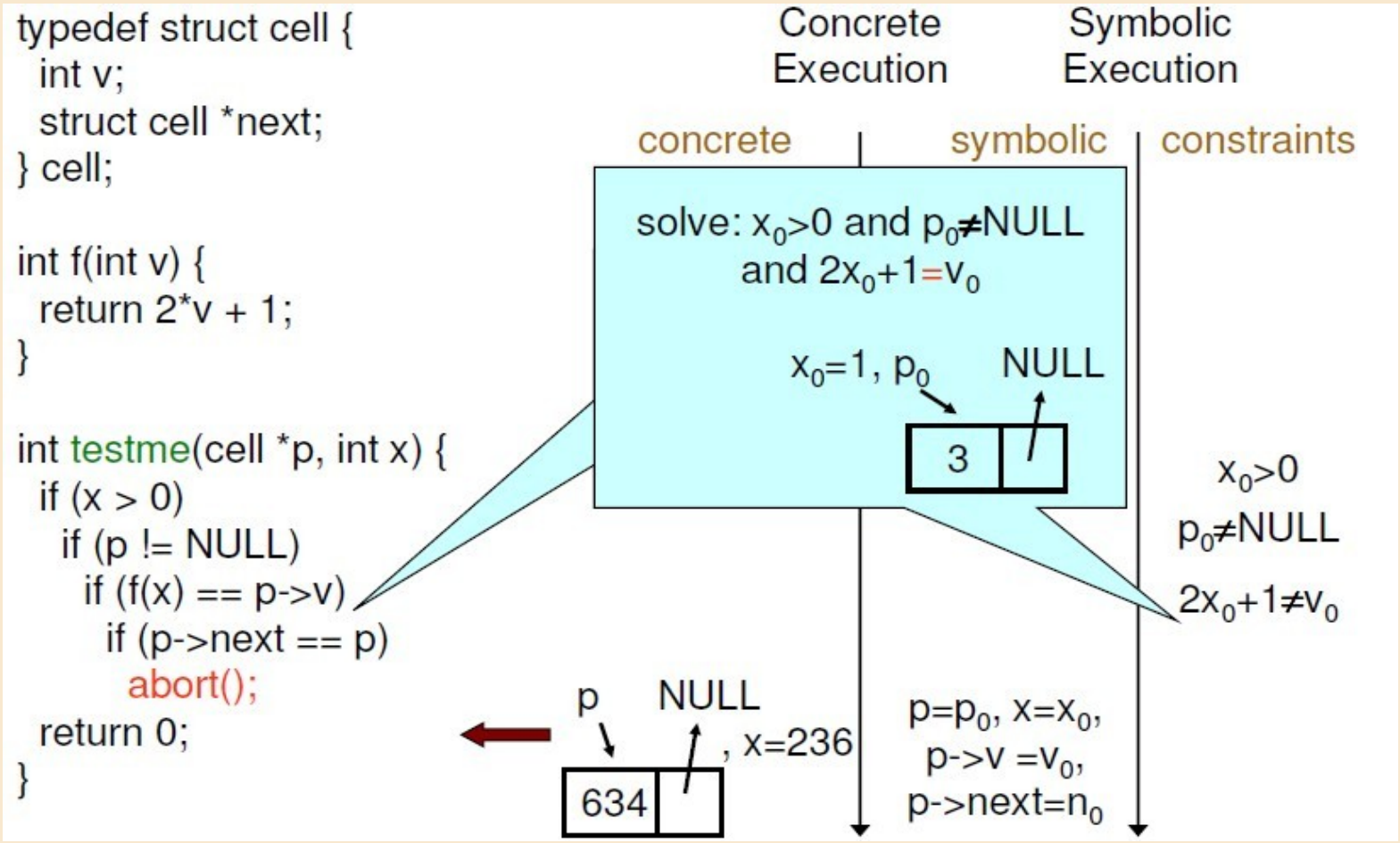
An example


typedef struct cell {
int v;
struct cell *next;
} cell;
int f(int v) {
return 2*v + 1;
}
int testme(cell *p, int x) {
if (x > 0)
if (p != NULL)
if (f(x) == p->v)
if (p->next == p)
abort();
return 0;
}
Concrete Execution
Symbolic Execution
concrete state
symbolic state
constraints
$x_0>0$
$p_0 \neq NULL$
$2x_0+1 \neq v_0$
p
NULL
, x=236
634
$p=p_0, x=x_0,$
$p->v = v_0,$
$p->next=n_0$

An example

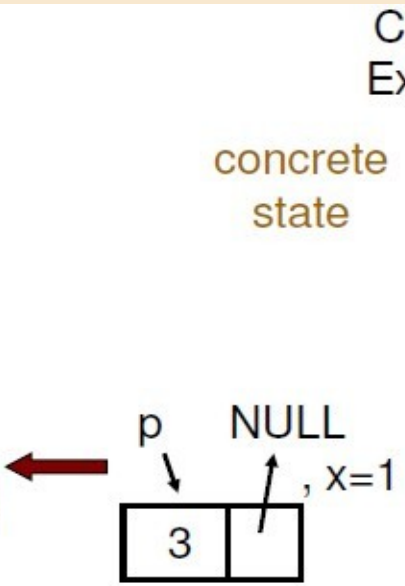

typedef struct cell {
int v;
struct cell *next;
} cell;
int f(int v) {
return 2*v + 1;
}
int testme(cell *p, int x) {
if (x > 0)
if (p != NULL)
if (f(x) == p->v)
if (p->next == p)
abort();
return 0;
}
Concrete Execution
Symbolic Execution
concrete
symbolic
constraints
solve: $x_0>0$ and $p_0\neq$NULL and $2x_0+1=v_0$
$x_0>0$
$p_0\neq$NULL
$2x_0+1\neq v_0$
p
NULL
, x=236
634
$p=p_0$, $x=x_0$,
p->v $=v_0$,
p->next$=n_0$

An example


typedef struct cell {
int v;
struct cell *next;
} cell;
int f(int v) {
return 2*v + 1;
}
int testme(cell *p, int x) {
if (x > 0)
if (p != NULL)
if (f(x) == p->v)
if (p->next == p)
abort();
return 0;
}
Concrete Execution
Symbolic Execution
concrete
symbolic
constraints
solve: $x_0>0$ and $p_0 \neq NULL$ and $2x_0+1=v_0$
$x_0=1$, p_0
NULL
3
$x_0>0$
$p_0 \neq NULL$
$2x_0+1 \neq v_0$
p
NULL
, x=236
634
$p=p_0$, $x=x_0$,
p->v $=v_0$,
p->next$=n_0$

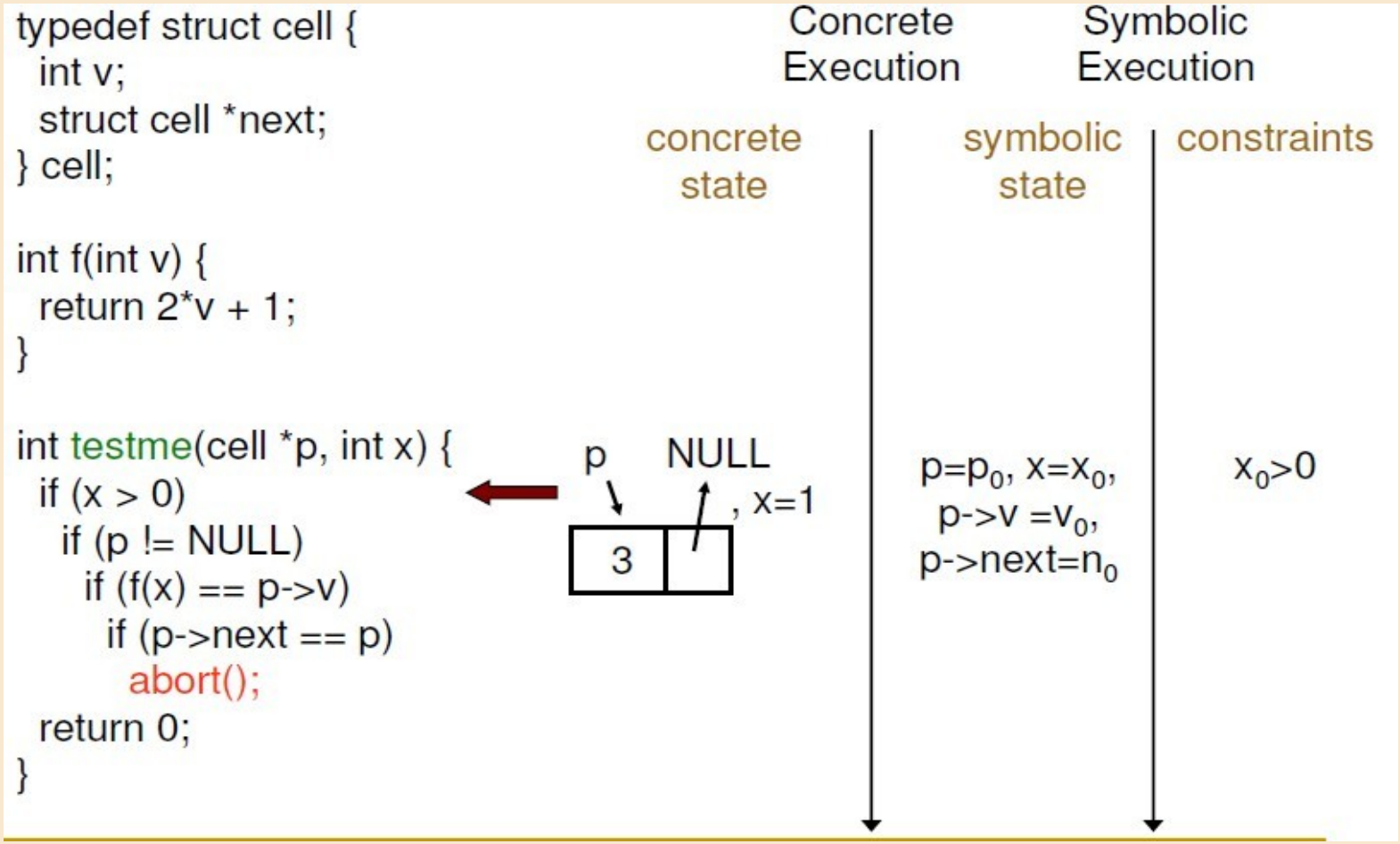
An example

```
typedef struct cell {
  int v;
  struct cell *next;
} cell;

int f(int v) {
  return 2*v + 1;
}

int testme(cell *p, int x) {
  if (x > 0)
    if (p != NULL)
      if (f(x) == p->v)
        if (p->next == p)
          abort();
  return 0;
}
```



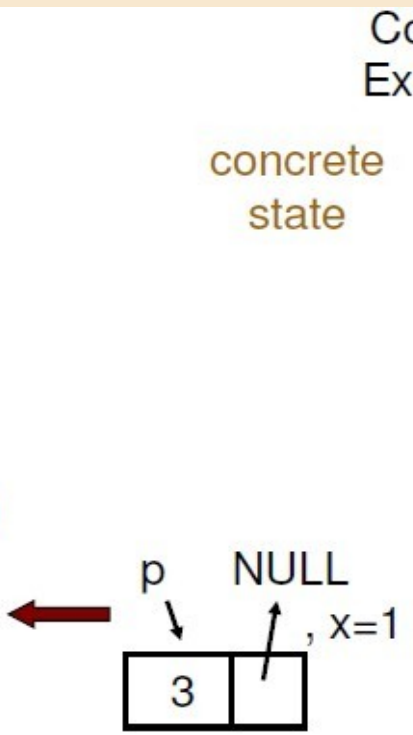
Concrete Execution		Symbolic Execution	
concrete state		symbolic state	constraints
		$p=p_0$, $x=x_0$, $p->v=v_0$, $p->next=n_0$	

An example


typedef struct cell {
int v;
struct cell *next;
} cell;
int f(int v) {
return 2*v + 1;
}
int testme(cell *p, int x) {
if (x > 0)
if (p != NULL)
if (f(x) == p->v)
if (p->next == p)
abort();
return 0;
}
Concrete Execution
Symbolic Execution
concrete state
symbolic state
constraints
p
NULL
, x=1
3
p=p0, x=x0,
p->v =v0,
p->next=n0
x0>0

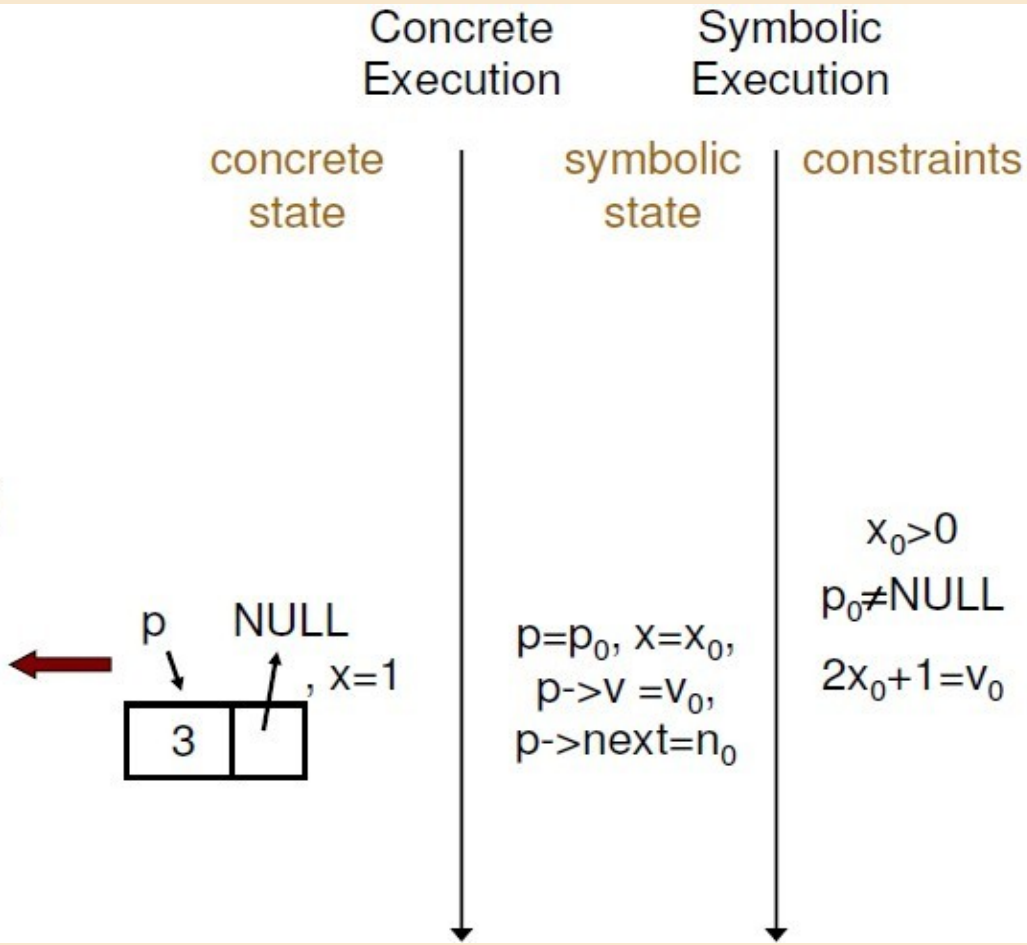
An example

```
typedef struct cell {
  int v;
  struct cell *next;
} cell;

int f(int v) {
  return 2*v + 1;
}

int testme(cell *p, int x) {
  if (x > 0)
    if (p != NULL)
      if (f(x) == p->v)
        if (p->next == p)
          abort();
  return 0;
}
```



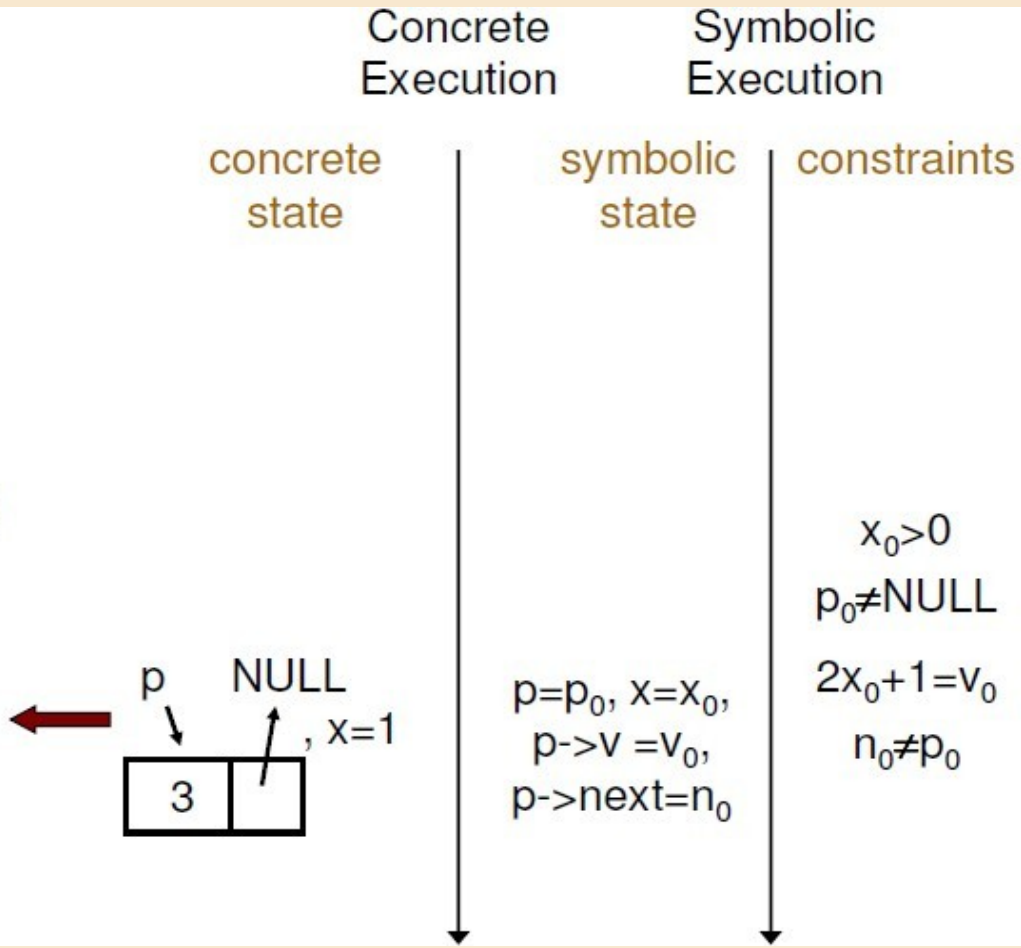
An example

```
typedef struct cell {
  int v;
  struct cell *next;
} cell;

int f(int v) {
  return 2*v + 1;
}

int testme(cell *p, int x) {
  if (x > 0)
    if (p != NULL)
      if (f(x) == p->v)
        if (p->next == p)
          abort();
  return 0;
}
```



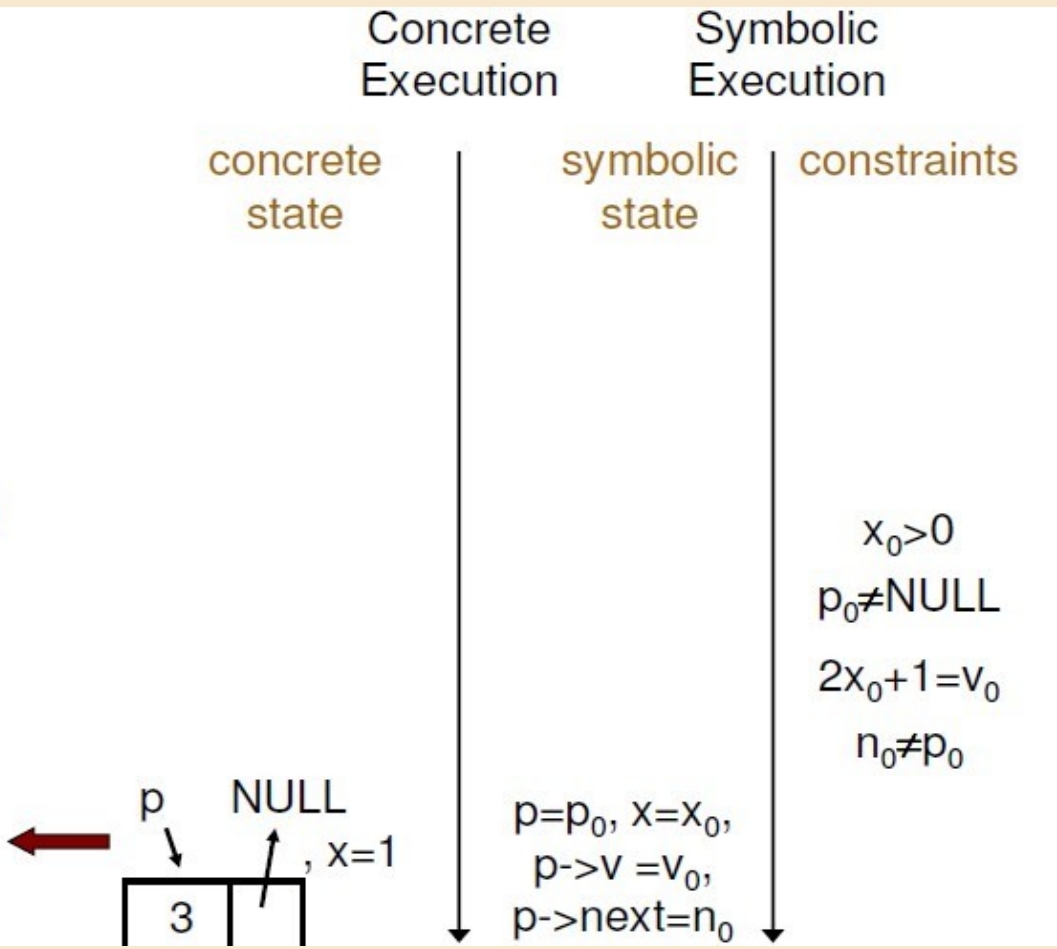
An example

```
typedef struct cell {
  int v;
  struct cell *next;
} cell;

int f(int v) {
  return 2*v + 1;
}

int testme(cell *p, int x) {
  if (x > 0)
    if (p != NULL)
      if (f(x) == p->v)
        if (p->next == p)
          abort();
  return 0;
}
```

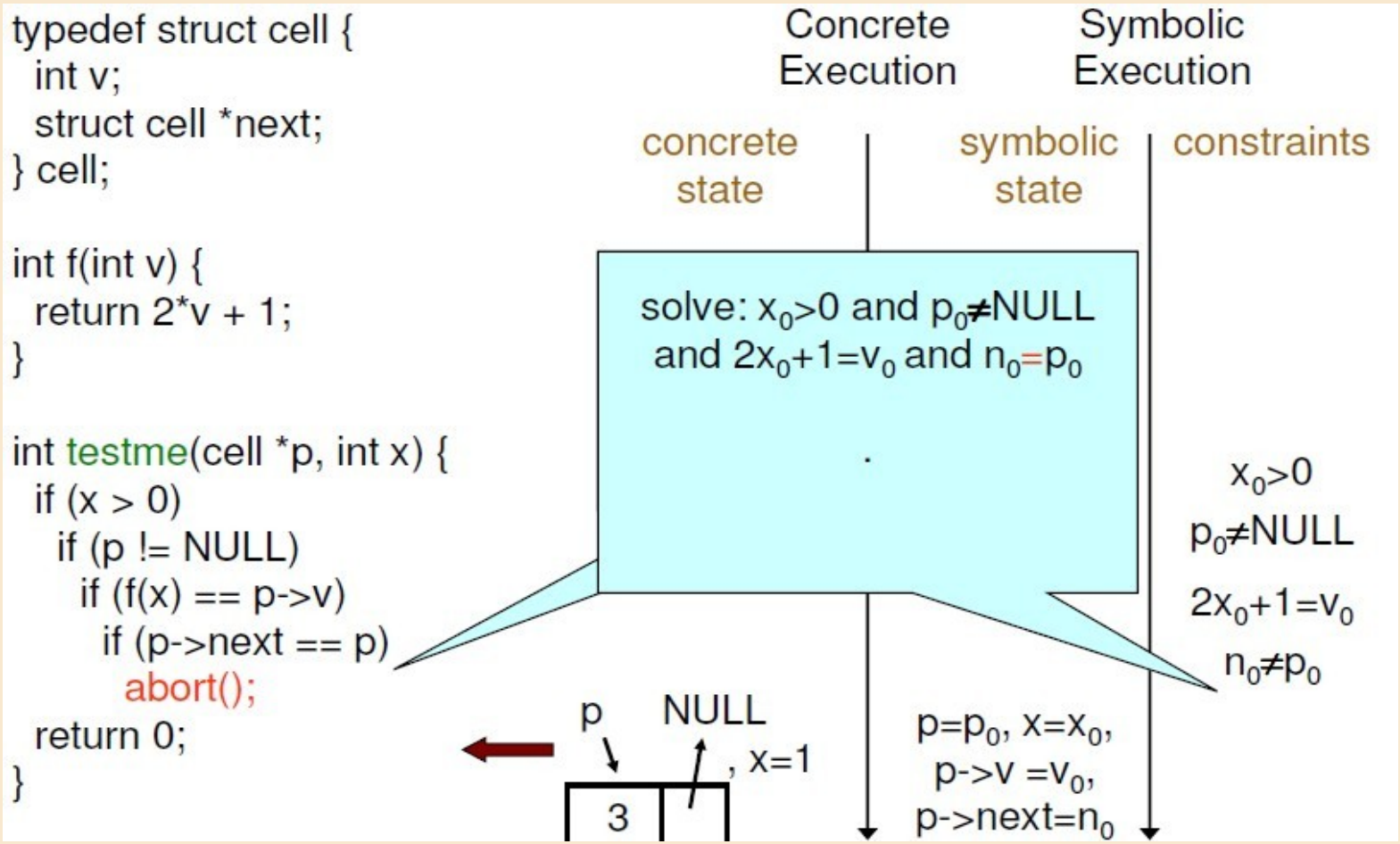


An example

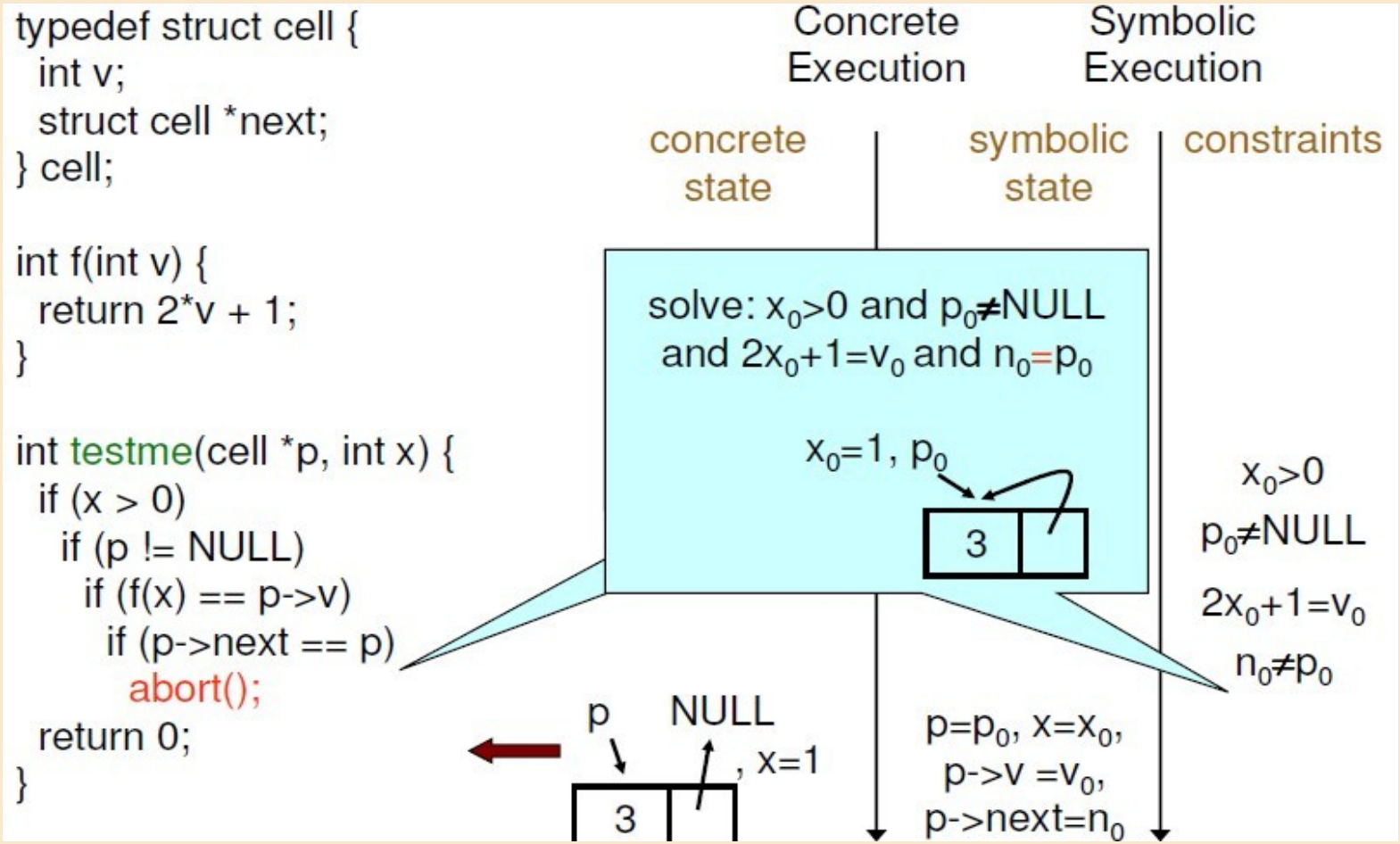
```
typedef struct cell {
  int v;
  struct cell *next;
} cell;

int f(int v) {
  return 2*v + 1;
}

int testme(cell *p, int x) {
  if (x > 0)
    if (p != NULL)
      if (f(x) == p->v)
        if (p->next == p)
          abort();
  return 0;
}
```



An example

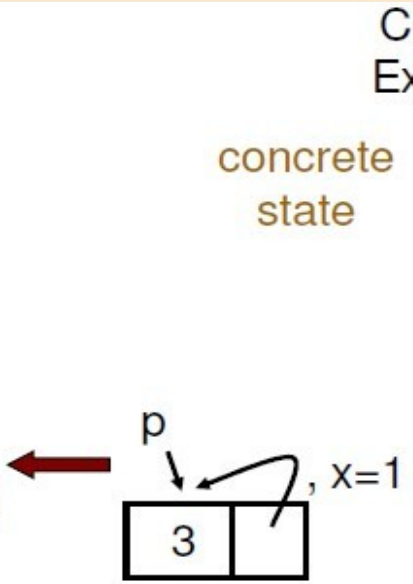

typedef struct cell {
int v;
struct cell *next;
} cell;
int f(int v) {
return 2*v + 1;
}
int testme(cell *p, int x) {
if (x > 0)
if (p != NULL)
if (f(x) == p->v)
if (p->next == p)
abort();
return 0;
}
Concrete Execution
Symbolic Execution
concrete state
symbolic state
constraints
solve: $x_0>0$ and $p_0 \neq NULL$ and $2x_0+1=v_0$ and $n_0=p_0$
$x_0>0$
$p_0 \neq NULL$
$2x_0+1=v_0$
$n_0 \neq p_0$
p
NULL
, x=1
3
$p=p_0$, $x=x_0$,
$p->v = v_0$,
$p->next=n_0$

An example


typedef struct cell {
int v;
struct cell *next;
} cell;
int f(int v) {
return 2*v + 1;
}
int testme(cell *p, int x) {
if (x > 0)
if (p != NULL)
if (f(x) == p->v)
if (p->next == p)
abort();
return 0;
}
Concrete Execution
Symbolic Execution
concrete state
symbolic state
constraints
solve: $x_0>0$ and $p_0\neq$NULL and $2x_0+1=v_0$ and $n_0=p_0$
$x_0=1$, p_0
3
$x_0>0$
$p_0\neq$NULL
$2x_0+1=v_0$
$n_0\neq p_0$
p NULL
, x=1
3
$p=p_0$, $x=x_0$,
$p->v=v_0$,
$p->next=n_0$

An example

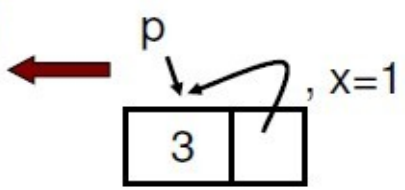
```
typedef struct cell {
  int v;
  struct cell *next;
} cell;

int f(int v) {
  return 2*v + 1;
}

int testme(cell *p, int x) {
  if (x > 0)
    if (p != NULL)
      if (f(x) == p->v)
        if (p->next == p)
          abort();
  return 0;
}
```



Concrete Execution	Symbolic Execution	
concrete state	symbolic state	constraints
	$p=p_0$, $x=x_0$, $p->v=v_0$, $p->next=n_0$	

An example

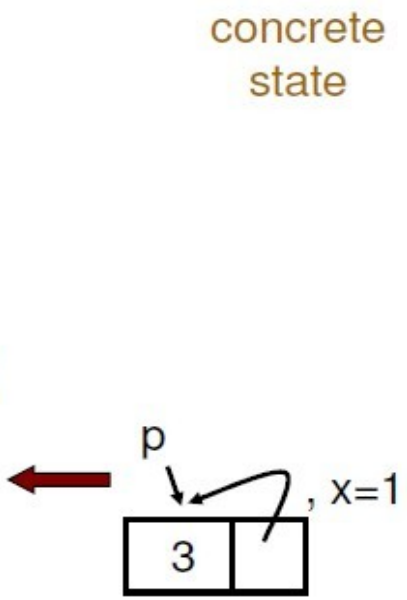
```
typedef struct cell {
  int v;
  struct cell *next;
} cell;

int f(int v) {
  return 2*v + 1;
}

int testme(cell *p, int x) {
  if (x > 0)
    if (p != NULL)
      if (f(x) == p->v)
        if (p->next == p)
          abort();
  return 0;
}
```



Concrete Execution	Symbolic Execution	
concrete state	symbolic state	constraints
	$p=p_0$, $x=x_0$, $p->v=v_0$, $p->next=n_0$	$x_0>0$

An example

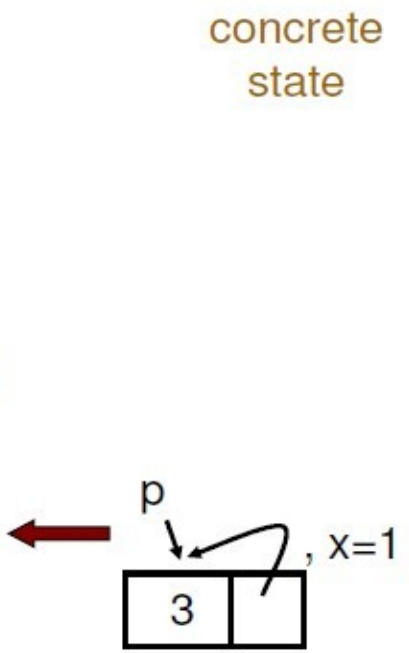
```
typedef struct cell {
  int v;
  struct cell *next;
} cell;

int f(int v) {
  return 2*v + 1;
}

int testme(cell *p, int x) {
  if (x > 0)
    if (p != NULL)
      if (f(x) == p->v)
        if (p->next == p)
          abort();
  return 0;
}
```



Concrete Execution	Symbolic Execution	
concrete state	symbolic state	constraints
	$p=p_0$, $x=x_0$, $p->v=v_0$, $p->next=n_0$	$x_0>0$, $p_0 \neq NULL$

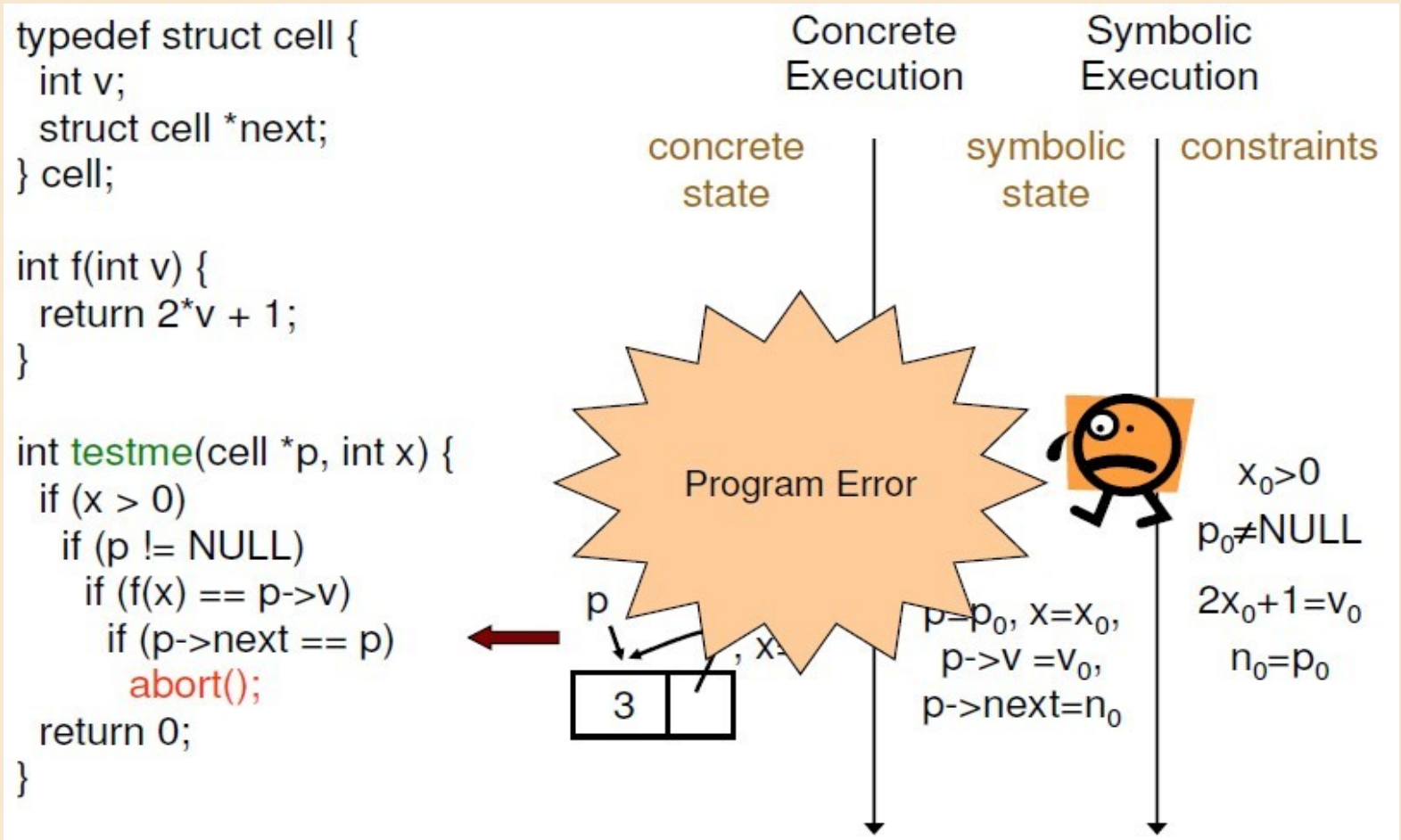
An example

```
typedef struct cell {
  int v;
  struct cell *next;
} cell;

int f(int v) {
  return 2*v + 1;
}

int testme(cell *p, int x) {
  if (x > 0)
    if (p != NULL)
      if (f(x) == p->v)
        if (p->next == p)
          abort();
  return 0;
}
```



Concrete Execution		Symbolic Execution	
concrete state		symbolic state	constraints
		$p=p_0$, $x=x_0$, $p\text{->}v=v_0$, $p\text{->}next=n_0$	$x_0>0$, $p_0 \neq NULL$, $2x_0+1=v_0$

An example


typedef struct cell {
int v;
struct cell *next;
} cell;
int f(int v) {
return 2*v + 1;
}
int testme(cell *p, int x) {
if (x > 0)
if (p != NULL)
if (f(x) == p->v)
if (p->next == p)
abort();
return 0;
}
Concrete Execution
Symbolic Execution
concrete state
symbolic state
constraints
Program Error
p
3
p=p0, x=x0,
p->v =v0,
p->next=n0
x0>0
p0≠NULL
2x0+1=v0
n0=p0

Outline

Static Analysis

- Previous Work

Static Analysis

- Previous Work
  - LCLint (USENIX Security 01)

Static Analysis

- Previous Work
  - LCLint (USENIX Security 01)
  - CSSV (PLDI 03)

Static Analysis

- Previous Work
  - LCLint (USENIX Security 01)
  - CSSV (PLDI 03)
  - ESC/Java (PLDI 02)

Static Analysis

- Previous Work
  - LCLint (USENIX Security 01)
  - CSSV (PLDI 03)
  - ESC/Java (PLDI 02)
  - ARCHER (FSE 03)

Static Analysis

- Previous Work
  - LCLint (USENIX Security 01)
  - CSSV (PLDI 03)
  - ESC/Java (PLDI 02)
  - ARCHER (FSE 03)
- Drawbacks

Runtime Check

- Previous Work

Runtime Check

- Previous Work
  - Purify (Winter USENIX Conf 1992)

Runtime Check

- Previous Work
  - Purify (Winter USENIX Conf 1992)
  - CCured (POPL 02)

Runtime Check

- Previous Work
  - Purify (Winter USENIX Conf 1992)
  - CCured (POPL 02)
  - CERD (NDSS 04)

Runtime Check

- Previous Work
  - Purify (Winter USENIX Conf 1992)
  - CCured (POPL 02)
  - CERD (NDSS 04)
- Drawbacks

Symbolic Execution

- Previous Work

Symbolic Execution

- Previous Work
  - DART(Random Testing) (PLDI 05)

Symbolic Execution

- Previous Work
  - DART(Random Testing) (PLDI 05)
  - CUTE (FSE 05)

Symbolic Execution

- Previous Work
  - DART(Random Testing) (PLDI 05)
  - CUTE (FSE 05)
  - KLEE (OSDI 08)

Symbolic Execution

- Previous Work
  - DART(Random Testing) (PLDI 05)
  - CUTE (FSE 05)
  - KLEE (OSDI 08)
  - Length Abstraction (ISSTA 08)

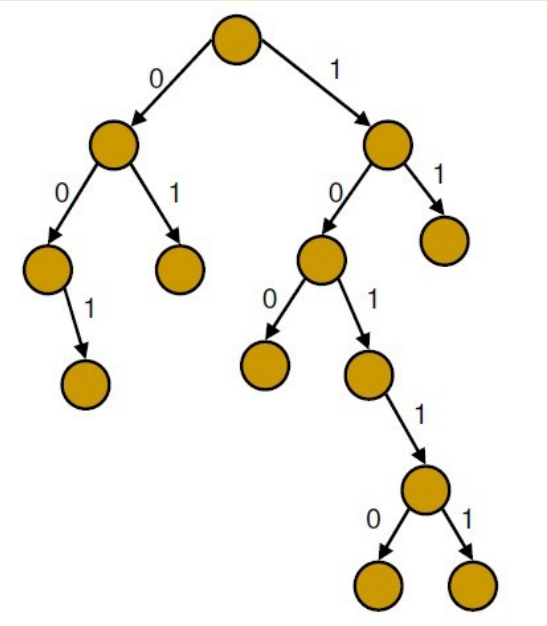
Symbolic Execution

- Previous Work
  - DART(Random Testing) (PLDI 05)
  - CUTE (FSE 05)
  - KLEE (OSDI 08)
  - Length Abstraction (ISSTA 08)
- Drawbacks

Outline

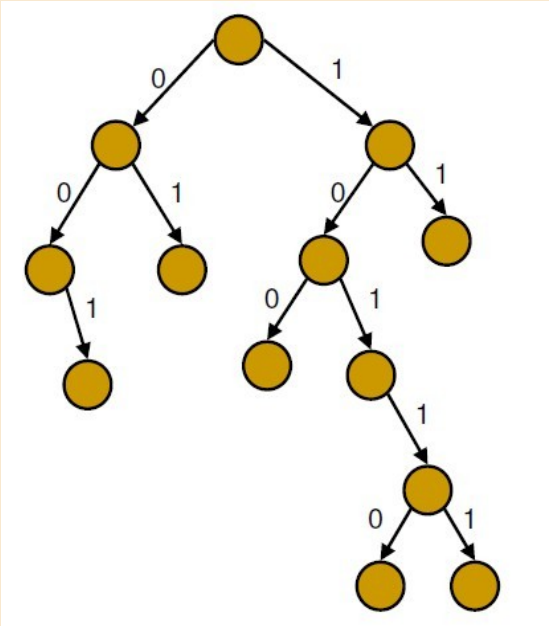
Motivation

- Motivation



Motivation

- Motivation



- Three forms of Buffer overflow

Improvement

- Compared with dynamic analysis

Improvement

- Compared with dynamic analysis
  - symbolic execution

Improvement

- Compared with dynamic analysis
  - symbolic execution
  - runtime check

Improvement

- Compared with dynamic analysis
  - symbolic execution
  - runtime check
- Compared with static analysis

Improvement

- Compared with dynamic analysis
  - symbolic execution
  - runtime check
- Compared with static analysis
- Compared with Length Abstraction (Unsure -,-)

Key Idea (by example)

- Sendmail 8.12.7 (daemon.c)

```
    /*      char buf[50];
            char *p=&buf[0];
            char *s;
            char ibuf[50];
            int i,nleft;        */

    nleft=sizeof(ibuf)-1;
    while((i=read(s,p,nleft))>0)
    {
        p+=i;
        nleft-=i;
        *p='\0';
        if(strchr(ibuf, '\n') != NULL || nleft <= 0)
            break;
    }
    close(s);
    if(i<0||p==&ibuf[0])
        goto noident;
    if(*--p=='\n' && *--p == '\r')
        p--;
    *++p = '\n';
noident:
    return 0;
```

Outline

Q & A

- Q & A

Q & A

- Q & A
- Thanks