Automatic Verification of Heap Properties

Hongseok Yang (Queen Mary Univ. of London)

Joint with Calcagno, Distefano, O'Hearn
Berdine, Cook, Lee

Our Goal

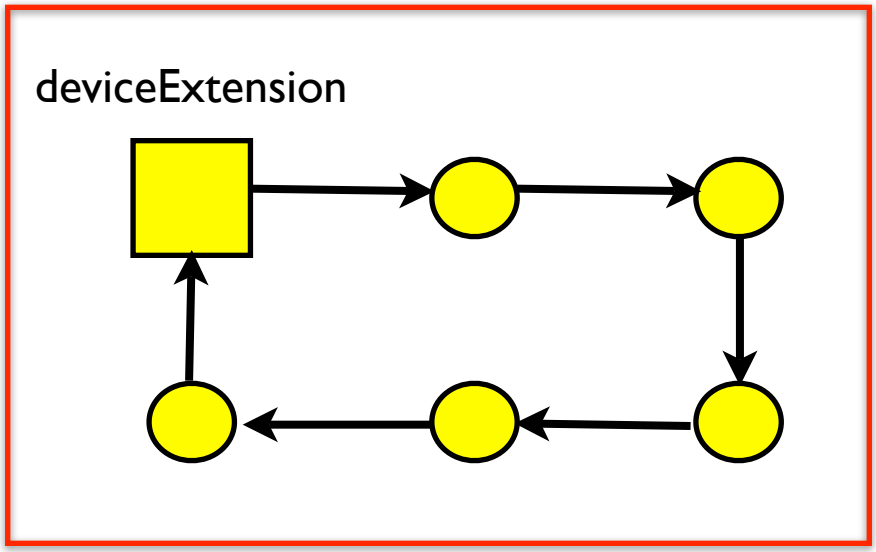
- Automatically verify memory safety of real-world systems code, such as Linux kernel, device drivers and Apache web servers.
- Dramatic improvement in the past 3 years:
  - In 2006, could verify up to 100 LOC.
  - In 2009, can verify up to 10000 LOC.

```c
void t1394Diag_CancelIrp(PDEVICE_OBJECT DeviceObject, PIRP Irp)
{
  KIRQL                 Irql, CancelIrql;
  BUS_RESET_IRP         *BusResetIrp, *temp;
  PDEVICE_EXTENSION     deviceExtension;

  deviceExtension = DeviceObject->DeviceExtension;

  KeAcquireSpinLock(&deviceExtension->ResetSpinLock, &Irql);

  temp = (PBUS_RESET_IRP)deviceExtension;
  BusResetIrp = (PBUS_RESET_IRP)deviceExtension->Flink2;

  while (BusResetIrp) {

    if (BusResetIrp->Irp == Irp) {
      temp->Flink2 = BusResetIrp->Flink2;
      free(BusResetIrp);
      break;
    }
    else if (BusResetIrp->Flink2 == (PBUS_RESET_IRP)deviceExtension) {
      break;
    }
    else {
      temp = BusResetIrp;
      BusResetIrp = (PBUS_RESET_IRP)BusResetIrp->Flink2;
    }
  }

  KeReleaseSpinLock(&deviceExtension->ResetSpinLock, Irql);

  IoReleaseCancelSpinLock(Irp->CancelIrql);
  Irp->IoStatus.Status = STATUS_CANCELLED;
  IoCompleteRequest(Irp, IO_NO_INCREMENT);
} // t1394Diag_CancelIrp
```

```c
typedef struct BUS_RESET_IRP {
  struct BUS_RESET_IRP *Flink2;
  PIRP                 Irp;
} BUS_RESET_IRP, *PBUS_RESET_IRP;


typedef struct {
  PBUS_RESET_IRP           Flink2;
} DEVICE_EXTENSION, *PDEVICE_EXTENSION;
```

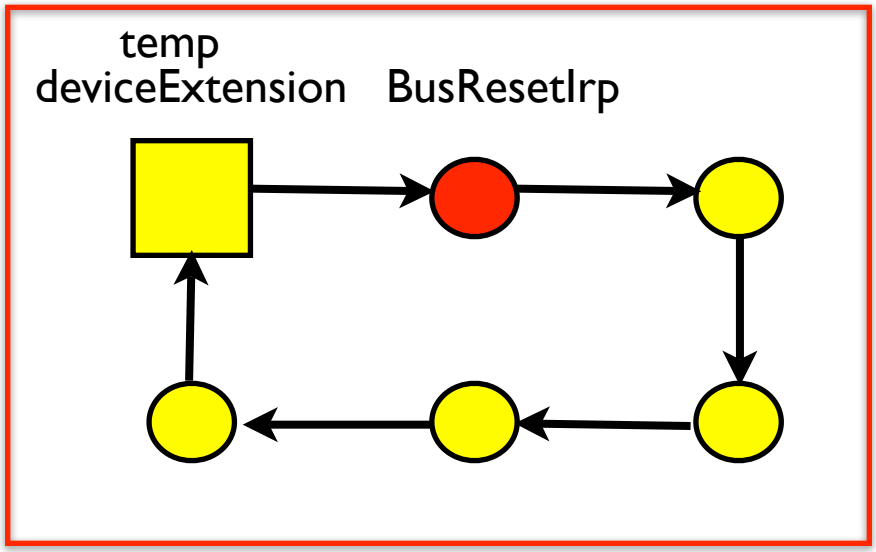
```
void t1394Diag_CancelIrp(PDEVICE_OBJECT DeviceObject, PIRP Irp)
{
  KIRQL                Irql, CancelIrql;
  BUS_RESET_IRP        *BusResetIrp, *temp;
  PDEVICE_EXTENSION    deviceExtension;

  deviceExtension = DeviceObject->DeviceExtension;

  KeAcquireSpinLock(&deviceExtension->ResetSpinLock, &Irql);

  temp = (PBUS_RESET_IRP)deviceExtension;
  BusResetIrp = (PBUS_RESET_IRP)deviceExtension->Flink2;

  while (BusResetIrp) {

    if (BusResetIrp->Irp == Irp) {
      temp->Flink2 = BusResetIrp->Flink2;
      free(BusResetIrp);
      break;
    }
    else if (BusResetIrp->Flink2 == (PBUS_RESET_IRP)deviceExtension) {
      break;
    }
    else {
      temp = BusResetIrp;
      BusResetIrp = (PBUS_RESET_IRP)BusResetIrp->Flink2;
    }
  }

  KeReleaseSpinLock(&deviceExtension->ResetSpinLock, Irql);

  IoReleaseCancelSpinLock(Irp->CancelIrql);
  Irp->IoStatus.Status = STATUS_CANCELLED;
  IoCompleteRequest(Irp, IO_NO_INCREMENT);
} // t1394Diag_CancelIrp
```

```
typedef struct BUS_RESET_IRP {
  struct BUS_RESET_IRP *Flink2;
  PIRP                 Irp;
} BUS_RESET_IRP, *PBUS_RESET_IRP;


typedef struct {
  PBUS_RESET_IRP           Flink2;
} DEVICE_EXTENSION, *PDEVICE_EXTENSION;
```



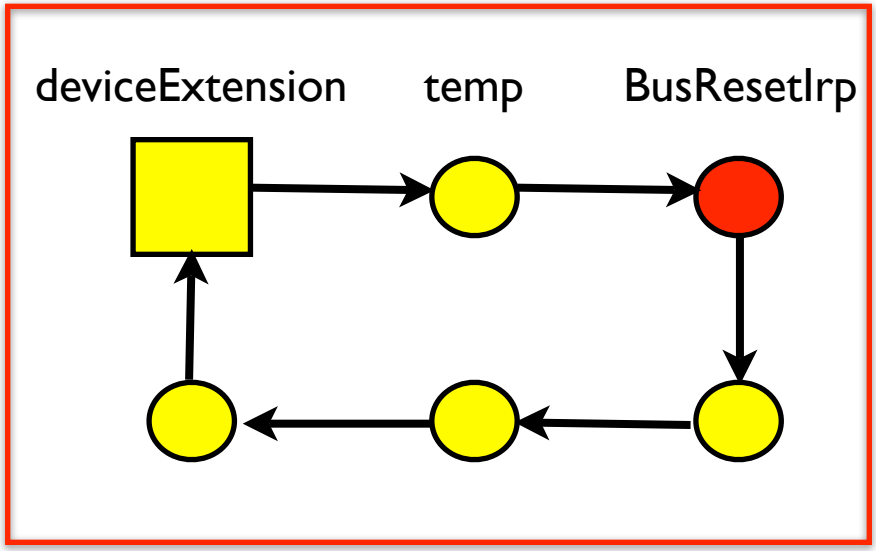
```
void t1394Diag_CancelIrp(PDEVICE_OBJECT DeviceObject, PIRP Irp)
{
  KIRQL                 Irql, CancelIrql;
  BUS_RESET_IRP         *BusResetIrp, *temp;
  PDEVICE_EXTENSION     deviceExtension;

  deviceExtension = DeviceObject->DeviceExtension;

  KeAcquireSpinLock(&deviceExtension->ResetSpinLock, &Irql);

  temp = (PBUS_RESET_IRP)deviceExtension;
  BusResetIrp = (PBUS_RESET_IRP)deviceExtension->Flink2;

  while (BusResetIrp) {

    if (BusResetIrp->Irp == Irp) {
      temp->Flink2 = BusResetIrp->Flink2;
      free(BusResetIrp);
      break;
    }
    else if (BusResetIrp->Flink2 == (PBUS_RESET_IRP)deviceExtension) {
      break;
    }
    else {
      temp = BusResetIrp;
      BusResetIrp = (PBUS_RESET_IRP)BusResetIrp->Flink2;
    }
  }

  KeReleaseSpinLock(&deviceExtension->ResetSpinLock, Irql);

  IoReleaseCancelSpinLock(Irp->CancelIrql);
  Irp->IoStatus.Status = STATUS_CANCELLED;
  IoCompleteRequest(Irp, IO_NO_INCREMENT);
} // t1394Diag_CancelIrp
```

```
typedef struct BUS_RESET_IRP {
  struct BUS_RESET_IRP *Flink2;
  PIRP                 Irp;
} BUS_RESET_IRP, *PBUS_RESET_IRP;


typedef struct {
  PBUS_RESET_IRP           Flink2;
} DEVICE_EXTENSION, *PDEVICE_EXTENSION;
```



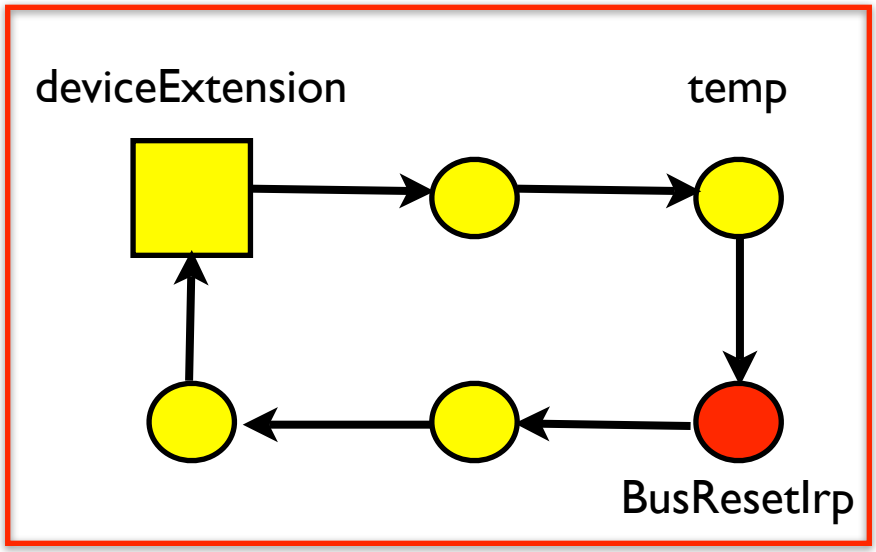
```
void t1394Diag_CancelIrp(PDEVICE_OBJECT DeviceObject, PIRP Irp)
{
  KIRQL                 Irql, CancelIrql;
  BUS_RESET_IRP         *BusResetIrp, *temp;
  PDEVICE_EXTENSION     deviceExtension;

  deviceExtension = DeviceObject->DeviceExtension;

  KeAcquireSpinLock(&deviceExtension->ResetSpinLock, &Irql);

  temp = (PBUS_RESET_IRP)deviceExtension;
  BusResetIrp = (PBUS_RESET_IRP)deviceExtension->Flink2;

  while (BusResetIrp) {

    if (BusResetIrp->Irp == Irp) {
      temp->Flink2 = BusResetIrp->Flink2;
      free(BusResetIrp);
      break;
    }
    else if (BusResetIrp->Flink2 == (PBUS_RESET_IRP)deviceExtension) {
      break;
    }
    else {
      temp = BusResetIrp;
      BusResetIrp = (PBUS_RESET_IRP)BusResetIrp->Flink2;
    }
  }

  KeReleaseSpinLock(&deviceExtension->ResetSpinLock, Irql);

  IoReleaseCancelSpinLock(Irp->CancelIrql);
  Irp->IoStatus.Status = STATUS_CANCELLED;
  IoCompleteRequest(Irp, IO_NO_INCREMENT);
} // t1394Diag_CancelIrp
```

```
typedef struct BUS_RESET_IRP {
  struct BUS_RESET_IRP *Flink2;
  PIRP                 Irp;
} BUS_RESET_IRP, *PBUS_RESET_IRP;


typedef struct {
  PBUS_RESET_IRP           Flink2;
} DEVICE_EXTENSION, *PDEVICE_EXTENSION;
```



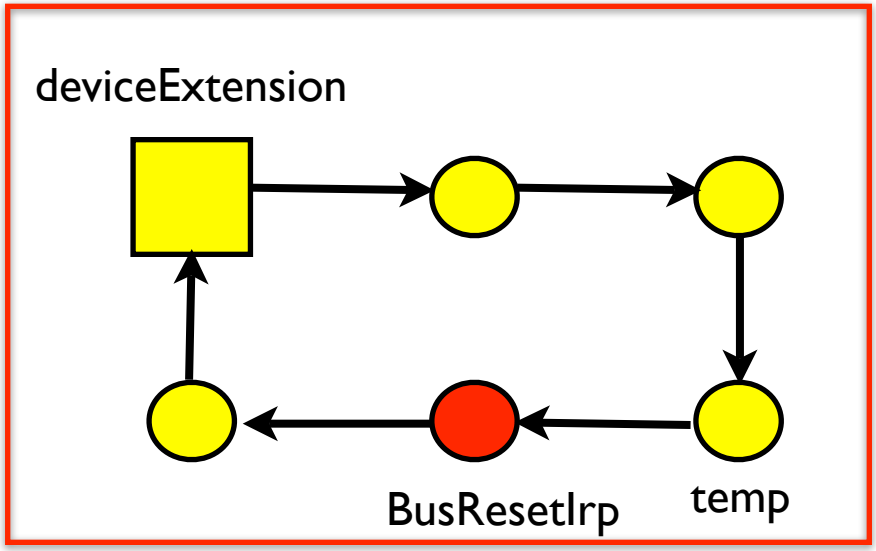
```
void t1394Diag_CancelIrp(PDEVICE_OBJECT DeviceObject, PIRP Irp)
{
  KIRQL                Irql, CancelIrql;
  BUS_RESET_IRP        *BusResetIrp, *temp;
  PDEVICE_EXTENSION    deviceExtension;

  deviceExtension = DeviceObject->DeviceExtension;

  KeAcquireSpinLock(&deviceExtension->ResetSpinLock, &Irql);

  temp = (PBUS_RESET_IRP)deviceExtension;
  BusResetIrp = (PBUS_RESET_IRP)deviceExtension->Flink2;

  while (BusResetIrp) {

    if (BusResetIrp->Irp == Irp) {
      temp->Flink2 = BusResetIrp->Flink2;
      free(BusResetIrp);
      break;
    }
    else if (BusResetIrp->Flink2 == (PBUS_RESET_IRP)deviceExtension) {
      break;
    }
    else {
      temp = BusResetIrp;
      BusResetIrp = (PBUS_RESET_IRP)BusResetIrp->Flink2;
    }
  }

  KeReleaseSpinLock(&deviceExtension->ResetSpinLock, Irql);

  IoReleaseCancelSpinLock(Irp->CancelIrql);
  Irp->IoStatus.Status = STATUS_CANCELLED;
  IoCompleteRequest(Irp, IO_NO_INCREMENT);
} // t1394Diag_CancelIrp
```

```
typedef struct BUS_RESET_IRP {
  struct BUS_RESET_IRP *Flink2;
  PIRP                 Irp;
} BUS_RESET_IRP, *PBUS_RESET_IRP;


typedef struct {
  PBUS_RESET_IRP           Flink2;
} DEVICE_EXTENSION, *PDEVICE_EXTENSION;
```



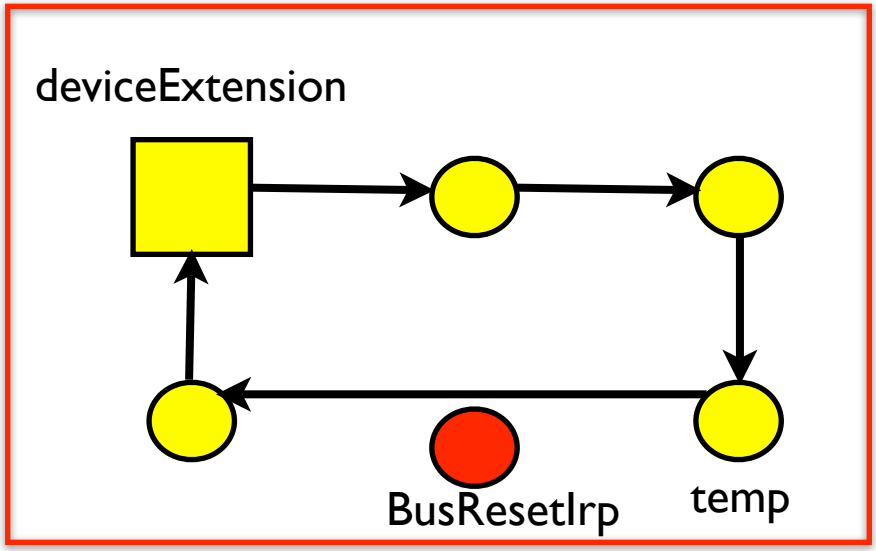
```c
void t1394Diag_CancelIrp(PDEVICE_OBJECT DeviceObject, PIRP Irp)
{
  KIRQL                Irql, CancelIrql;
  BUS_RESET_IRP        *BusResetIrp, *temp;
  PDEVICE_EXTENSION    deviceExtension;

  deviceExtension = DeviceObject->DeviceExtension;

  KeAcquireSpinLock(&deviceExtension->ResetSpinLock, &Irql);

  temp = (PBUS_RESET_IRP)deviceExtension;
  BusResetIrp = (PBUS_RESET_IRP)deviceExtension->Flink2;

  while (BusResetIrp) {

    if (BusResetIrp->Irp == Irp) {
      temp->Flink2 = BusResetIrp->Flink2;
      free(BusResetIrp);
      break;
    }
    else if (BusResetIrp->Flink2 == (PBUS_RESET_IRP)deviceExtension) {
      break;
    }
    else {
      temp = BusResetIrp;
      BusResetIrp = (PBUS_RESET_IRP)BusResetIrp->Flink2;
    }
  }

  KeReleaseSpinLock(&deviceExtension->ResetSpinLock, Irql);

  IoReleaseCancelSpinLock(Irp->CancelIrql);
  Irp->IoStatus.Status = STATUS_CANCELLED;
  IoCompleteRequest(Irp, IO_NO_INCREMENT);
} // t1394Diag_CancelIrp
```

```c
typedef struct BUS_RESET_IRP {
  struct BUS_RESET_IRP *Flink2;
  PIRP                 Irp;
} BUS_RESET_IRP, *PBUS_RESET_IRP;


typedef struct {
  PBUS_RESET_IRP           Flink2;
} DEVICE_EXTENSION, *PDEVICE_EXTENSION;
```



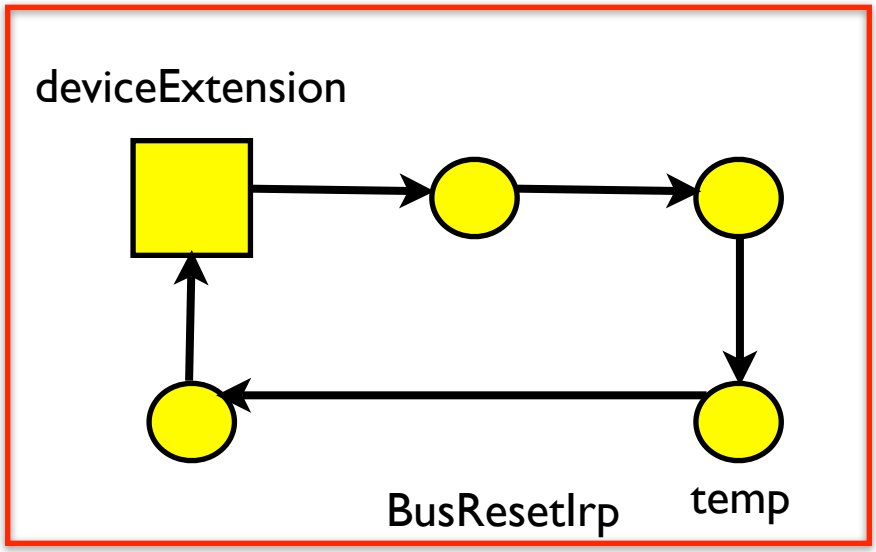
```
void t1394Diag_CancelIrp(PDEVICE_OBJECT DeviceObject, PIRP Irp)
{
  KIRQL                Irql, CancelIrql;
  BUS_RESET_IRP        *BusResetIrp, *temp;
  PDEVICE_EXTENSION    deviceExtension;

  deviceExtension = DeviceObject->DeviceExtension;

  KeAcquireSpinLock(&deviceExtension->ResetSpinLock, &Irql);

  temp = (PBUS_RESET_IRP)deviceExtension;
  BusResetIrp = (PBUS_RESET_IRP)deviceExtension->Flink2;

  while (BusResetIrp) {

    if (BusResetIrp->Irp == Irp) {
      temp->Flink2 = BusResetIrp->Flink2;
      free(BusResetIrp);
      break;
    }
    else if (BusResetIrp->Flink2 == (PBUS_RESET_IRP)deviceExtension) {
      break;
    }
    else {
      temp = BusResetIrp;
      BusResetIrp = (PBUS_RESET_IRP)BusResetIrp->Flink2;
    }
  }

  KeReleaseSpinLock(&deviceExtension->ResetSpinLock, Irql);

  IoReleaseCancelSpinLock(Irp->CancelIrql);
  Irp->IoStatus.Status = STATUS_CANCELLED;
  IoCompleteRequest(Irp, IO_NO_INCREMENT);
} // t1394Diag_CancelIrp
```

```
typedef struct BUS_RESET_IRP {
  struct BUS_RESET_IRP *Flink2;
  PIRP                 Irp;
} BUS_RESET_IRP, *PBUS_RESET_IRP;


typedef struct {
  PBUS_RESET_IRP           Flink2;
} DEVICE_EXTENSION, *PDEVICE_EXTENSION;
```



```
void t1394Diag_CancelIrp(PDEVICE_OBJECT DeviceObject, PIRP Irp)
{
  KIRQL                Irql, CancelIrql;
  BUS_RESET_IRP        *BusResetIrp, *temp;
  PDEVICE_EXTENSION    deviceExtension;

  deviceExtension = DeviceObject->DeviceExtension;

  KeAcquireSpinLock(&deviceExtension->ResetSpinLock, &Irql);

  temp = (PBUS_RESET_IRP)deviceExtension;
  BusResetIrp = (PBUS_RESET_IRP)deviceExtension->Flink2;

  while (BusResetIrp) {

    if (BusResetIrp->Irp == Irp) {
      temp->Flink2 = BusResetIrp->Flink2;
      free(BusResetIrp);
      break;
    }
    else if (BusResetIrp->Flink2 == (PBUS_RESET_IRP)deviceExtension) {
      break;
    }
    else {
      temp = BusResetIrp;
      BusResetIrp = (PBUS_RESET_IRP)BusResetIrp->Flink2;
    }
  }

  KeReleaseSpinLock(&deviceExtension->ResetSpinLock, Irql);

  IoReleaseCancelSpinLock(Irp->CancelIrql);
  Irp->IoStatus.Status = STATUS_CANCELLED;
  IoCompleteRequest(Irp, IO_NO_INCREMENT);
} // t1394Diag_CancelIrp
```

```
typedef struct BUS_RESET_IRP {
  struct BUS_RESET_IRP *Flink2;
  PIRP                 Irp;
} BUS_RESET_IRP, *PBUS_RESET_IRP;


typedef struct {
  PBUS_RESET_IRP           Flink2;
} DEVICE_EXTENSION, *PDEVICE_EXTENSION;
```



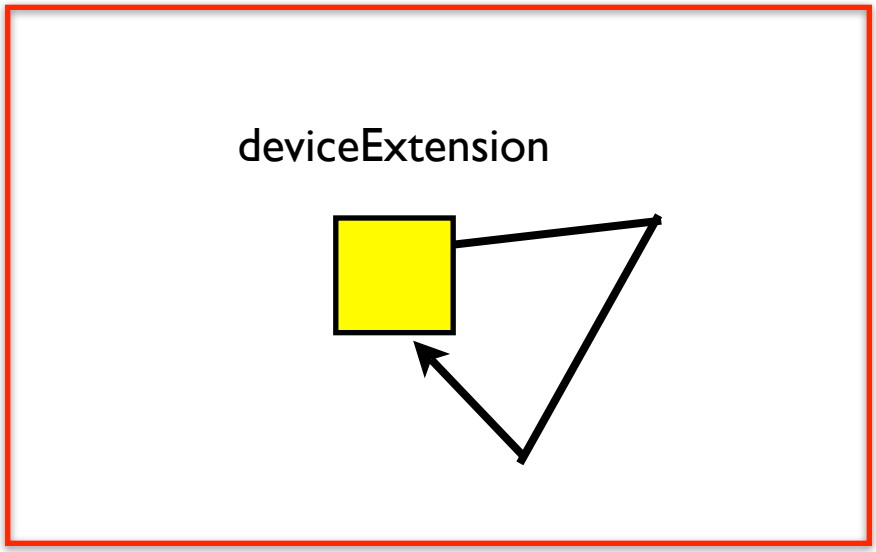
Demo

```c
void t1394Diag_CancelIrp(PDEVICE_OBJECT DeviceObject, PIRP Irp)
{
  KIRQL                 Irql, CancelIrql;
  BUS_RESET_IRP         *BusResetIrp, *temp;
  PDEVICE_EXTENSION     deviceExtension;

  deviceExtension = DeviceObject->DeviceExtension;

  KeAcquireSpinLock(&deviceExtension->ResetSpinLock, &Irql);

  temp = (PBUS_RESET_IRP)deviceExtension;
  BusResetIrp = (PBUS_RESET_IRP)deviceExtension->Flink2;

  while (BusResetIrp) {

    if (BusResetIrp->Irp == Irp) {
      temp->Flink2 = BusResetIrp->Flink2;
      free(BusResetIrp);
      break;
    }
    else if (BusResetIrp->Flink2 == (PBUS_RESET_IRP)deviceExtension) {
      break;
    }
    else {
      temp = BusResetIrp;
      BusResetIrp = (PBUS_RESET_IRP)BusResetIrp->Flink2;
    }
  }

  KeReleaseSpinLock(&deviceExtension->ResetSpinLock, Irql);

  IoReleaseCancelSpinLock(Irp->CancelIrql);
  Irp->IoStatus.Status = STATUS_CANCELLED;
  IoCompleteRequest(Irp, IO_NO_INCREMENT);
} // t1394Diag_CancelIrp
```

```c
typedef struct BUS_RESET_IRP {
  struct BUS_RESET_IRP *Flink2;
  PIRP                 Irp;
} BUS_RESET_IRP, *PBUS_RESET_IRP;


typedef struct {
  PBUS_RESET_IRP           Flink2;
} DEVICE_EXTENSION, *PDEVICE_EXTENSION;
```

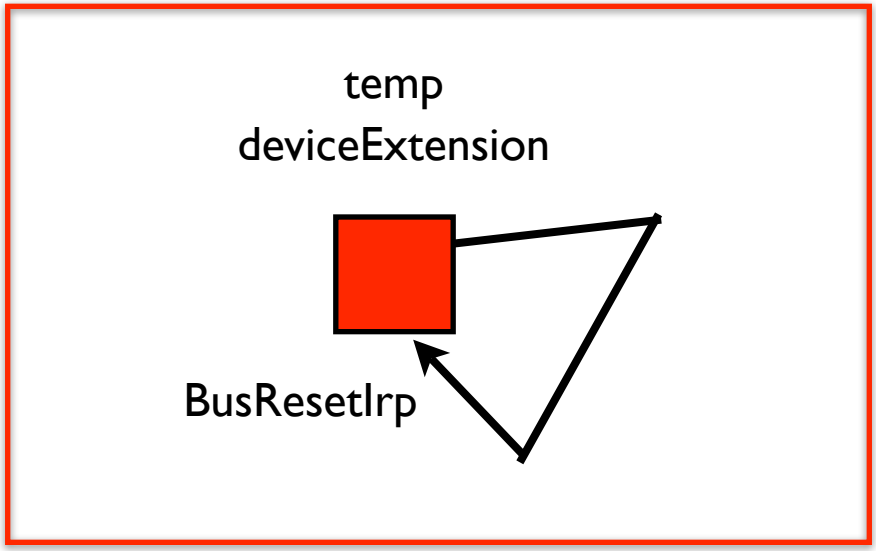
```
void t1394Diag_CancelIrp(PDEVICE_OBJECT DeviceObject, PIRP Irp)
{
  KIRQL                Irql, CancelIrql;
  BUS_RESET_IRP        *BusResetIrp, *temp;
  PDEVICE_EXTENSION    deviceExtension;

  deviceExtension = DeviceObject->DeviceExtension;

  KeAcquireSpinLock(&deviceExtension->ResetSpinLock, &Irql);

  temp = (PBUS_RESET_IRP)deviceExtension;
  BusResetIrp = (PBUS_RESET_IRP)deviceExtension->Flink2;

  while (BusResetIrp) {

    if (BusResetIrp->Irp == Irp) {
      temp->Flink2 = BusResetIrp->Flink2;
      free(BusResetIrp);
      break;
    }
    else if (BusResetIrp->Flink2 == (PBUS_RESET_IRP)deviceExtension) {
      break;
    }
    else {
      temp = BusResetIrp;
      BusResetIrp = (PBUS_RESET_IRP)BusResetIrp->Flink2;
    }
  }

  KeReleaseSpinLock(&deviceExtension->ResetSpinLock, Irql);

  IoReleaseCancelSpinLock(Irp->CancelIrql);
  Irp->IoStatus.Status = STATUS_CANCELLED;
  IoCompleteRequest(Irp, IO_NO_INCREMENT);
} // t1394Diag_CancelIrp
```

```
typedef struct BUS_RESET_IRP {
  struct BUS_RESET_IRP *Flink2;
  PIRP                 Irp;
} BUS_RESET_IRP, *PBUS_RESET_IRP;


typedef struct {
  PBUS_RESET_IRP           Flink2;
} DEVICE_EXTENSION, *PDEVICE_EXTENSION;
```



```
void t1394Diag_CancelIrp(PDEVICE_OBJECT DeviceObject, PIRP Irp)
{
  KIRQL                Irql, CancelIrql;
  BUS_RESET_IRP        *BusResetIrp, *temp;
  PDEVICE_EXTENSION    deviceExtension;

  deviceExtension = DeviceObject->DeviceExtension;

  KeAcquireSpinLock(&deviceExtension->ResetSpinLock, &Irql);

  temp = (PBUS_RESET_IRP)deviceExtension;
  BusResetIrp = (PBUS_RESET_IRP)deviceExtension->Flink2;

  while (BusResetIrp) {

    if (BusResetIrp->Irp == Irp) {
      temp->Flink2 = BusResetIrp->Flink2;
      free(BusResetIrp);
      break;
    }
    else if (BusResetIrp->Flink2 == (PBUS_RESET_IRP)deviceExtension) {
      break;
    }
    else {
      temp = BusResetIrp;
      BusResetIrp = (PBUS_RESET_IRP)BusResetIrp->Flink2;
    }
  }

  KeReleaseSpinLock(&deviceExtension->ResetSpinLock, Irql);

  IoReleaseCancelSpinLock(Irp->CancelIrql);
  Irp->IoStatus.Status = STATUS_CANCELLED;
  IoCompleteRequest(Irp, IO_NO_INCREMENT);
} // t1394Diag_CancelIrp
```

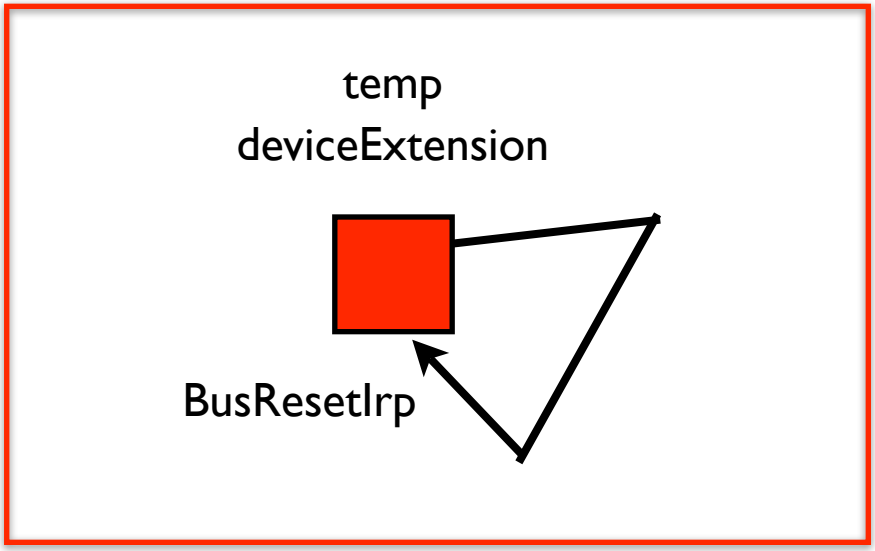
```
typedef struct BUS_RESET_IRP {
  struct BUS_RESET_IRP *Flink2;
  PIRP                 Irp;
} BUS_RESET_IRP, *PBUS_RESET_IRP;


typedef struct {
  PBUS_RESET_IRP           Flink2;
} DEVICE_EXTENSION, *PDEVICE_EXTENSION;
```



```
void t1394Diag_CancelIrp(PDEVICE_OBJECT DeviceObject, PIRP Irp)
{
  KIRQL                Irql, CancelIrql;
  BUS_RESET_IRP        *BusResetIrp, *temp;
  PDEVICE_EXTENSION    deviceExtension;

  deviceExtension = DeviceObject->DeviceExtension;

  KeAcquireSpinLock(&deviceExtension->ResetSpinLock, &Irql);

  temp = (PBUS_RESET_IRP)deviceExtension;
  BusResetIrp = (PBUS_RESET_IRP)deviceExtension->Flink2;

  while (BusResetIrp) {

    if (BusResetIrp->Irp == Irp) {
      temp->Flink2 = BusResetIrp->Flink2;
      free(BusResetIrp);
      break;
    }
    else if (BusResetIrp->Flink2 == (PBUS_RESET_IRP)deviceExtension) {
      break;
    }
    else {
      temp = BusResetIrp;
      BusResetIrp = (PBUS_RESET_IRP)BusResetIrp->Flink2;
    }
  }

  KeReleaseSpinLock(&deviceExtension->ResetSpinLock, Irql);

  IoReleaseCancelSpinLock(Irp->CancelIrql);
  Irp->IoStatus.Status = STATUS_CANCELLED;
  IoCompleteRequest(Irp, IO_NO_INCREMENT);
} // t1394Diag_CancelIrp
```

```
typedef struct BUS_RESET_IRP {
  struct BUS_RESET_IRP *Flink2;
  PIRP                 Irp;
} BUS_RESET_IRP, *PBUS_RESET_IRP;


typedef struct {
  PBUS_RESET_IRP           Flink2;
} DEVICE_EXTENSION, *PDEVICE_EXTENSION;
```

Pb: Dereferences a non-existing field “Irp”.



```
void t1394Diag_CancelIrp(PDEVICE_OBJECT DeviceObject, PIRP Irp)
{
  KIRQL                 Irql, CancelIrql;
  BUS_RESET_IRP         *BusResetIrp, *temp;
  PDEVICE_EXTENSION     deviceExtension;

  deviceExtension = DeviceObject->DeviceExtension;

  KeAcquireSpinLock(&deviceExtension->ResetSpinLock, &Irql);

  temp = (PBUS_RESET_IRP)deviceExtension;
  BusResetIrp = (PBUS_RESET_IRP)deviceExtension->Flink2;
   while (BusResetIrp) {

     if (BusResetIrp == (PBUS_RESET_IRP)deviceExtension) {
       break;
     }
     else if (BusResetIrp->Irp == Irp) {

       temp->Flink2 = BusResetIrp->Flink2;
       free(BusResetIrp);
       break;
     }
     else {
       temp = BusResetIrp;
       BusResetIrp = (PBUS_RESET_IRP)BusResetIrp->Flink2;
     }
   }
  KeReleaseSpinLock(&deviceExtension->ResetSpinLock, Irql);

  IoReleaseCancelSpinLock(Irp->CancelIrql);
  Irp->IoStatus.Status = STATUS_CANCELLED;
  IoCompleteRequest(Irp, IO_NO_INCREMENT);
} // t1394Diag_CancelIrp
```

```
typedef struct BUS_RESET_IRP {
  struct BUS_RESET_IRP *Flink2;
  PIRP                 Irp;
} BUS_RESET_IRP, *PBUS_RESET_IRP;


typedef struct {
  PBUS_RESET_IRP           Flink2;
} DEVICE_EXTENSION, *PDEVICE_EXTENSION;
```

Demo

Table from CAV'08

Program	LOC	Sec	MB	Memory leaks	Dereference errors
`scull.c`	1010	0.21	1.47	1	0
`class.c`	1983	6.68	8.36	2	1
`pci-driver.c`	2532	0.79	3.19	0	0
`ll_rw_blk.c`	5469	997.66	523.22	3	1
`cdrom.c`	6218	91.88	84.30	0	2
`md.c`	6635	1440.53	814.45	6	5
`t1394Diag.c`	10240	145.95	71.27	33	10

Table 1. Experimental Results. Performed on an Intel Core Duo 2GHz with 2GB.

Table from CAV'08

Program	LOC	Sec	MB	Memory leaks	Dereference errors
`scull.c`	1010	0.21	1.47	1	0
`class.c`	1983	6.68	8.36	2	1
`pci-driver.c`	2532	0.79	3.19	0	0
`ll_rw_blk.c`	5469	997.66	523.22	3	1
`cdrom.c`	6218	91.88	84.30	0	2
`md.c`	6635	1440.53	814.45	6	5
`t1394Diag.c`	10240	145.95	71.27	33	10

Table 1. Experimental Results. Performed on an Intel Core Duo 2GHz with 2GB.

- Programs of 1010 ~ 10240 LOC.
- Includes the full t1394Diag firewire driver.

Table from CAV'08

Program	LOC	Sec	MB	Memory leaks	Dereference errors
`scull.c`	1010	0.21	1.47	1	0
`class.c`	1983	6.68	8.36	2	1
`pci-driver.c`	2532	0.79	3.19	0	0
`ll_rw_blk.c`	5469	997.66	523.22	3	1
`cdrom.c`	6218	91.88	84.30	0	2
`md.c`	6635	1440.53	814.45	6	5
`t1394Diag.c`	10240	145.95	71.27	33	10

Table 1. Experimental Results. Performed on an Intel Core Duo 2GHz with 2GB.

- Found memory leaks and safety errors.
- Fixed those errors. Then, verified the integrity of pointer manipulation.

Table from CAV'08

{emp} t1394Diag_main() {emp}

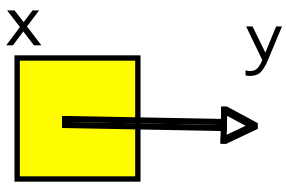
Program	LOC	Sec	MB	Memory leaks	Dereference errors
scull.c	1010	0.21	1.47	1	0
class.c	1983	6.68	8.36	2	1
pci-driver.c	2532	0.79	3.19	0	0
ll_rw_blk.c	5469	997.66	523.22	3	1
cdrom.c	6218	91.88	84.30	0	2
md.c	6635	1440.53	814.45	6	5
t1394Diag.c	10240	145.95	71.27	33	10

Table 1. Experimental Results. Performed on an Intel Core Duo 2GHz with 2GB.

- Found memory leaks and safety errors.
- Fixed those errors. Then, verified the integrity of pointer manipulation.

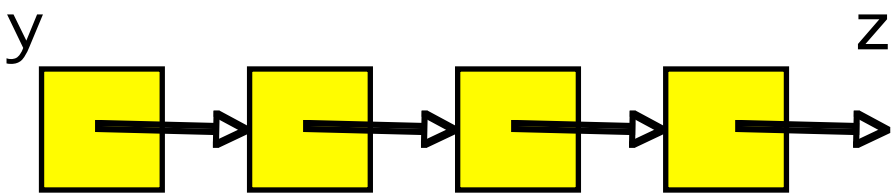
Separation Logic

$x \mapsto y$

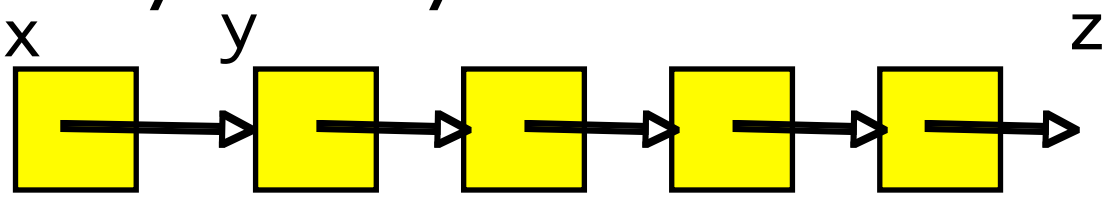


emp

lsne y z



$x \mapsto y * \text{lsne } y\ z$



$\exists w', v'.\ (z=y \wedge w' \neq v') \wedge (x \mapsto w' * \text{lsne } w'\ v' * \text{lsne } y\ v')$

Symbolic Heaps

Separation logic formulas of the form:

$$\exists w', v'.\ (y = z \wedge w' \neq v') \wedge (x \mapsto w' * \mathrm{lsne}\ w'\ v' * \mathrm{lsne}\ y\ v')$$

Symbolic Heaps

Separation logic formulas of the form:

$$(y=z \wedge w' \neq v') \wedge (x \mapsto w' * \mathrm{lsne}\ w'\ v' * \mathrm{lsne}\ y\ v')$$

Shape Analysis

- Run a program symbolically with symbolic heaps.
- Apply abstraction periodically to ensure termination.
- Repeat until the fixpoint is reached.

Function “Abs”

$$\mathrm{Abs} : \mathrm{SH} \rightarrow \mathrm{SH}$$

- Forgets the length of lists. Removes unnecessary primed vars.
- Ensures the termination of the analysis.
- E.g. $z!=x' \wedge (x \mapsto x' * x' \mapsto 0 * \mathrm{lsne}\ y\ y' * \mathrm{lsne}\ y'\ 0)$.

Function “Abs”

$$\mathrm{Abs} : \mathrm{SH} \rightarrow \mathrm{SH}$$

- Forgets the length of lists. Removes unnecessary primed vars.
- Ensures the termination of the analysis.
- E.g. z!=x’ $\wedge$ (lsne x 0 * lsne y y’ * lsne y’ 0).

Function “Abs”

$$\mathrm{Abs} : \mathrm{SH} \rightarrow \mathrm{SH}$$

- Forgets the length of lists. Removes unnecessary primed vars.
- Ensures the termination of the analysis.
- E.g. z!=x’ $\wedge$ (Isne x 0 * Isne y 0).

Function “Abs”

$$\mathrm{Abs} : \mathrm{SH} \rightarrow \mathrm{SH}$$

- Forgets the length of lists. Removes unnecessary primed vars.
- Ensures the termination of the analysis.
- E.g. (Isne x 0 * Isne y 0).

List Creation

```
L create() {
   L h=0;



   while (nondet)
   {



      L t;
      t = malloc();
      t->next=h;
      h=t;



   }

   return h;
}
```

List Creation

emp

```
L create() {
  L h=0;


  while (nondet)
  {


    L t;
    t = malloc();
    t->next=h;
    h=t;


  }

  return h;
}
```

List Creation

emp

```
L create() {
  L h=0;
```

h=0 ∧ emp

```
  while (nondet)
  {


    L t;
    t = malloc();
    t->next=h;
    h=t;



  }

  return h;
}
```

List Creation

```
emp
L create() {
  L h=0;
     h=0 ∧ emp
  while (nondet)
  {
     h=0 ∧ emp

     L t;
     t = malloc();
     t->next=h;
     h=t;


  }

  return h;
}
```

List Creation

```
emp
L create() {
  L h=0;
     h=0 ∧ emp
  while (nondet)
  {
     h=0 ∧ emp

     L t;
     t = malloc();
     t->next=h;
     h=t;
     t=h ∧ h↦0
  }

  return h;
}
```

List Creation

```
                                  emp
L create() {
   L h=0;
        h=0 ∧ emp
   while (nondet)
   {
        h=0 ∧ emp

      L t;
      t = malloc();
      t->next=h;
      h=t;
        t'=h ∧ h↦0
   }

   return h;
}
```

List Creation

```
emp
L create() {
  L h=0;
      h=0 ∧ emp
  while (nondet)
  {
      h=0 ∧ emp

    L t;
    t = malloc();
    t->next=h;
    h=t;
      h ↦ 0
  }

  return h;
}
```

List Creation

```
emp
L create() {
  L h=0;
    h=0 ∧ emp
  while (nondet)
  {
    h=0 ∧ emp

    L t;
    t = malloc();
    t->next=h;
    h=t;
      h ↦ 0
  }

  return h;
}
```

h ↦ 0

List Creation

```
emp
L create() {
  L h=0;
     h=0 ∧ emp
  while (nondet)
  {
     h=0 ∧ emp

     L t;
     t = malloc();
     t->next=h;
     h=t;
     h ↦ 0
  }

  return h;
}
```

h ↦ 0

t=h ∧ h↦h' * h'↦0

List Creation

```
emp
L create() {
  L h=0;
    h=0 ∧ emp
  while (nondet)
  {
    h=0 ∧ emp
      L t;
      t = malloc();
      t->next=h;
      h=t;
      h ↦ 0
  }
  return h;
}
```

h ↦ 0

t'=h ∧ h↦h' * h'↦0

List Creation

```
emp
L create() {
  L h=0;
    h=0 ∧ emp
  while (nondet)
  {
    h=0 ∧ emp

    L t;
    t = malloc();
    t->next=h;
    h=t;
    h ↦ 0
  }

  return h;
}
```

h ↦ 0

t'=h ∧ lsne h 0

List Creation

```
emp
L create() {
  L h=0;
    h=0 ∧ emp
  while (nondet)
  {
    h=0 ∧ emp

    L t;
    t = malloc();
    t->next=h;
    h=t;
    h ↦ 0
  }

  return h;
}
```

h ↦ 0

lsne h 0

List Creation

```
emp
L create() {
  L h=0;
    h=0 ∧ emp
  while (nondet)
  {
    h=0 ∧ emp

    L t;
    t = malloc();
    t->next=h;
    h=t;
    h ↦ 0
  }

  return h;
}
```

h ↦ 0

lsne h 0

lsne h 0

List Creation

```
emp
L create() {
  L h=0;
    h=0 ∧ emp
  while (nondet)
  {
    h=0 ∧ emp

      L t;
      t = malloc();
      t->next=h;
      h=t;
      h ↦ 0
  }

  return h;
}
```

h ↦ 0

lsne h 0

lsne h 0

t=h ∧ h↦h' * lsne h' 0

List Creation

```
emp
L create() {
  L h=0;
    h=0 ∧ emp
  while (nondet)
  {
    h=0 ∧ emp

      L t;
      t = malloc();
      t->next=h;
      h=t;
      h ↦ 0
  }

  return h;
}
```

h ↦ 0

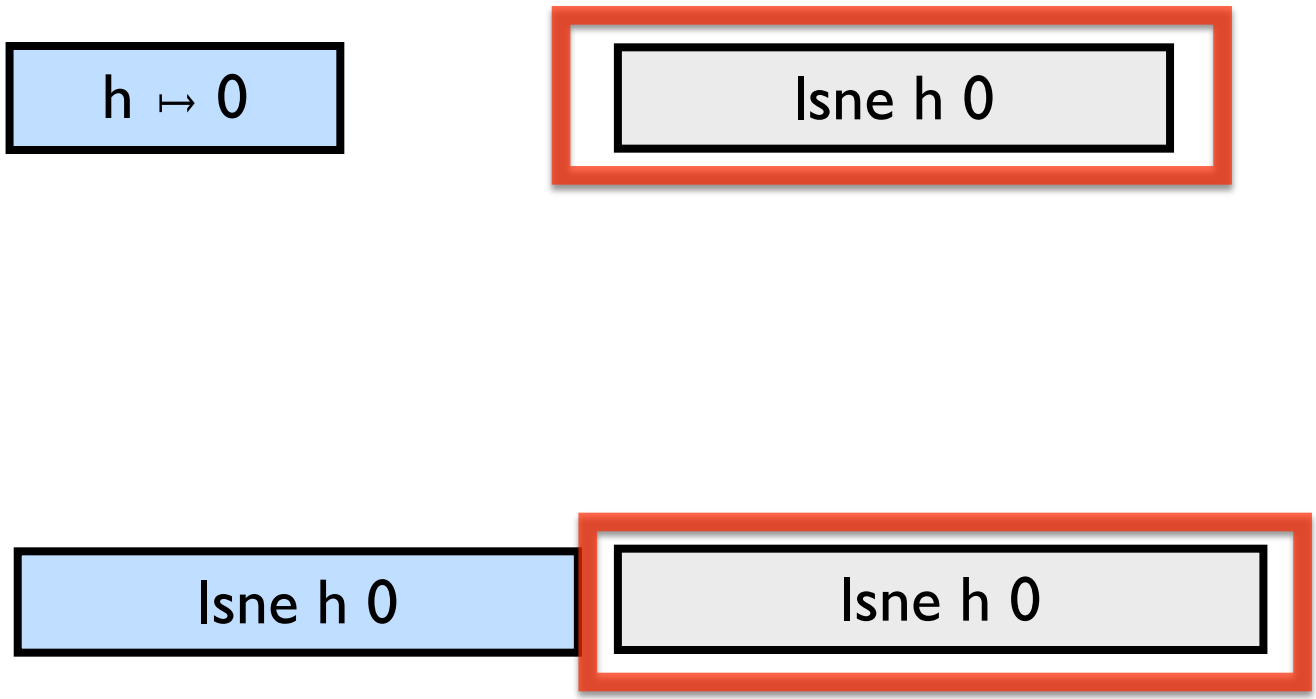
lsne h 0

lsne h 0

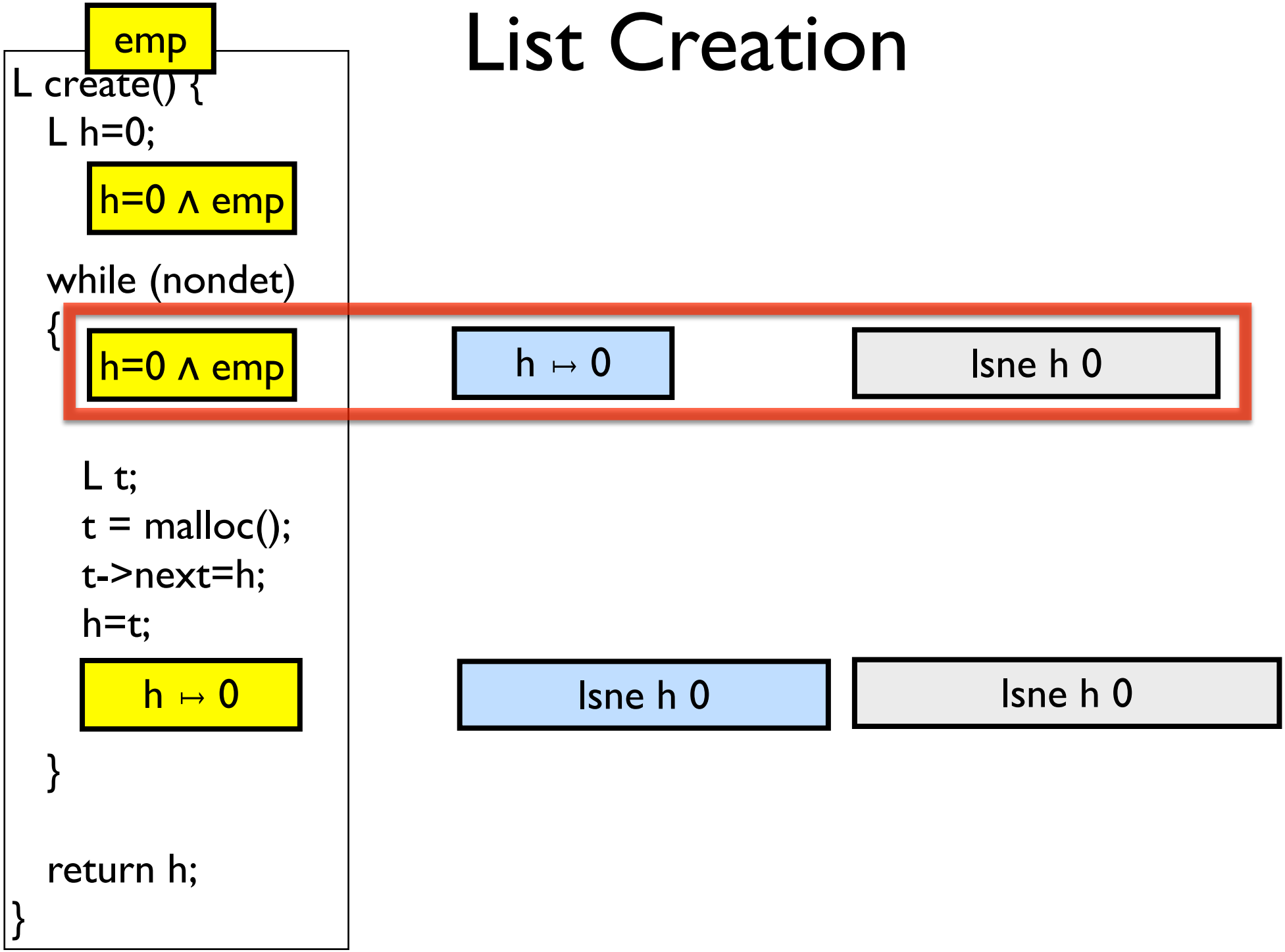
lsne h 0

List Creation

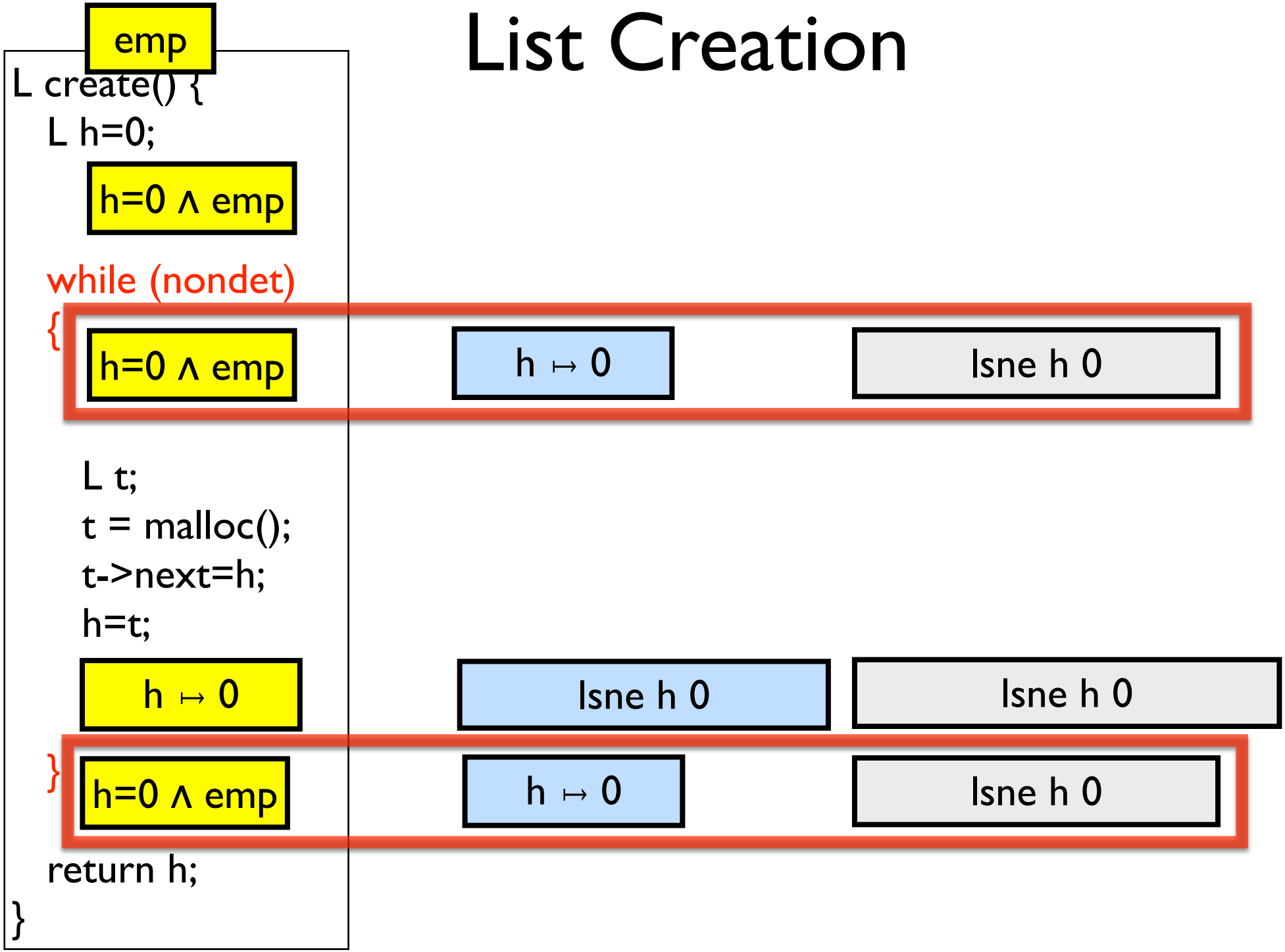
```
emp
L create() {
  L h=0;
    h=0 ∧ emp
  while (nondet)
  {
    h=0 ∧ emp

    L t;
    t = malloc();
    t->next=h;
    h=t;
    h ↦ 0
  }

  return h;
}
```



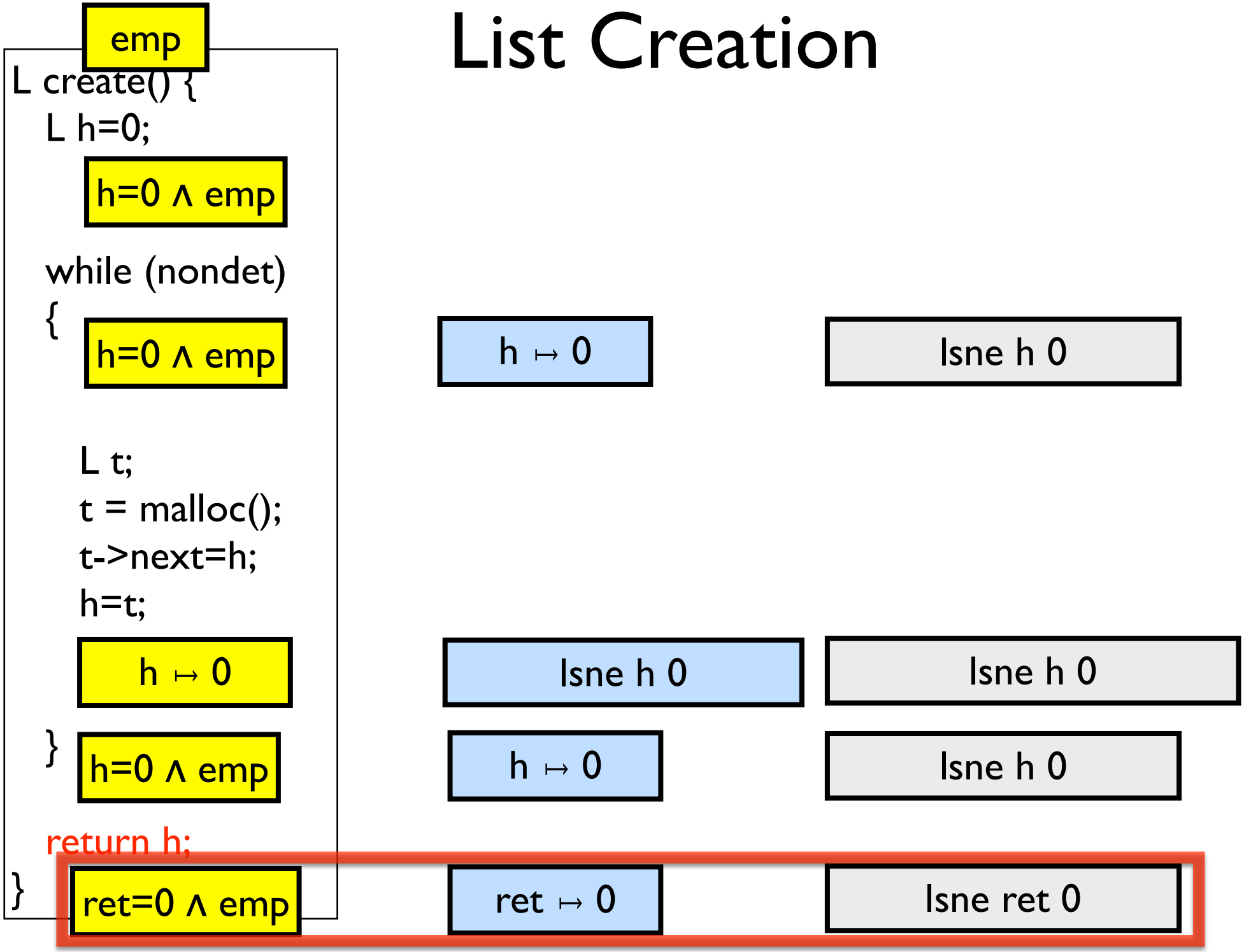
List Creation



List Creation



List Creation



Creation of Two Lists

```
L x = 0;
L y = 0;
while (nondet)
{

   L t;
   t = malloc();
   t->next=x;
   x=t;

   t = malloc();
   t->next=y;
   y=t;

}
```

Creation of Two Lists

```
L x = 0;
L y = 0;
while (nondet)
{
   L t;
   t = malloc();
   t->next=x;
   x=t;

   t = malloc();
   t->next=y;
   y=t;

}
```

$x=y=0 \wedge emp$ ... $x \mapsto 0 * y \mapsto 0$ $lsne\,x\,0 * lsne\,y\,0$ ... $lsne\,x\,0 * lsne\,y\,0$

- 9 entries.
- In Firewire, more than 5000 cases generated.

Symbolic Heaps with Possibly Empty List-seg lspe

$$
\begin{aligned}
e &::= x \mid x' \mid \mathsf{nil} \\
\Pi &::= \Pi \wedge \Pi \mid e{=}e \mid e{\neq}e \mid \mathsf{true} \\
\Sigma &::= \Sigma * \Sigma \mid \mathsf{emp} \mid (e \mapsto e) \mid \mathsf{lsne}\ e\ e \mid \mathsf{lspe}\ e\ e \mid \mathsf{true} \\
q &::= \Pi \wedge \Sigma
\end{aligned}
$$

Symbolic Heaps with Possibly Empty List-seg lspe

$$
\begin{aligned}
e &::= x \mid x' \mid \mathsf{nil} \\
\Pi &::= \Pi \wedge \Pi \mid e{=}e \mid e{\neq}e \mid \mathsf{true} \\
\Sigma &::= \Sigma * \Sigma \mid \mathsf{emp} \mid (e \mapsto e) \mid \mathsf{lsne}\ e\ e \mid \mathsf{lspe}\ e\ e \mid \mathsf{true} \\
q &::= \Pi \wedge \Sigma
\end{aligned}
$$

Partial Join Operator

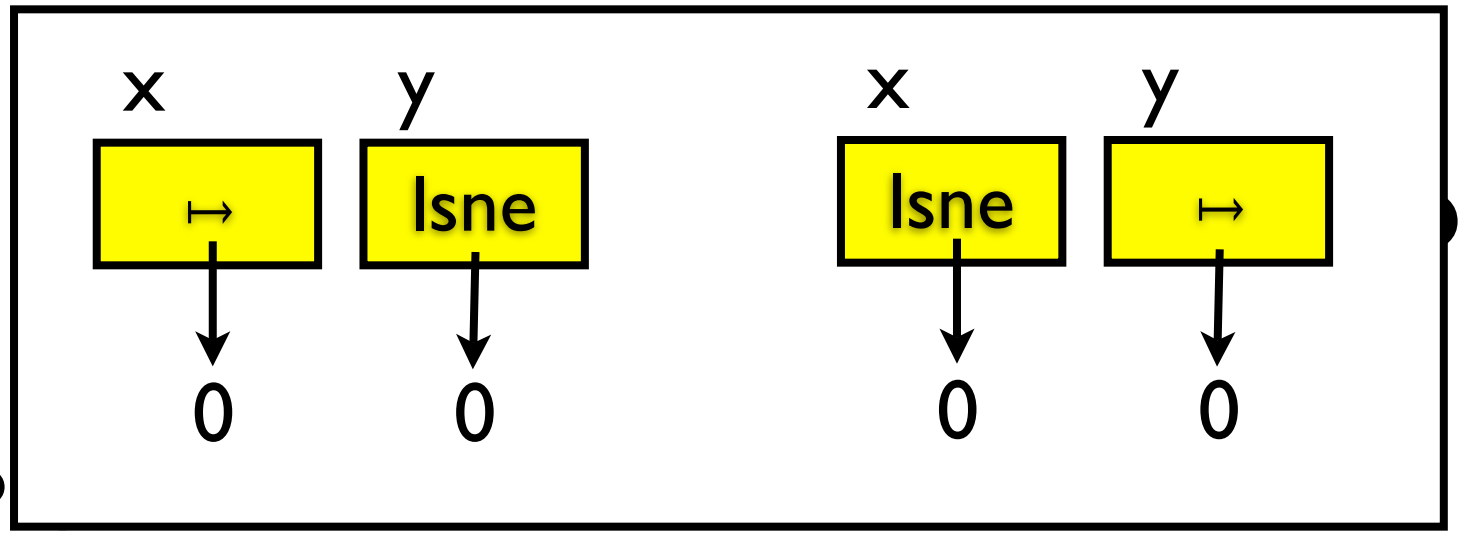
- pjoin : SH x SH $\rightharpoonup$ SH
- Overapproximation :

  If pjoin(q, q')=q", then $(q \vee q') \Rightarrow q"$
- E.g.

  pjoin($x \mapsto 0$ * lsne y 0, lsne x 0 * $y \mapsto 0$) = lsne x 0 * lsne y 0

  pjoin($x \mapsto y$ * lsne y 0, lsne x 0 * $y \mapsto 0$) = undefined.

  pjoin($y=0 \wedge$ lsne x y, $x \mapsto y$ * lsne y 0) = lsne x y * lspe y 0



- Overapproximation :

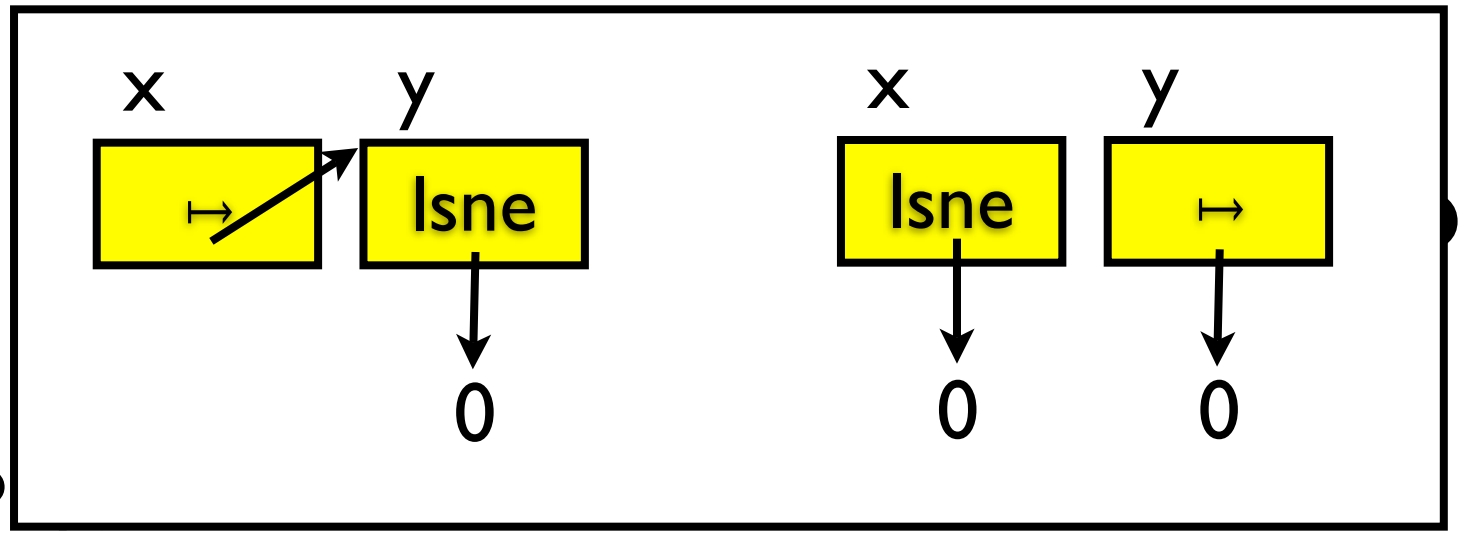
  If pjoin(q, q')=q", then $(q \vee q') \Rightarrow q$"

- E.g.

  pjoin(x↦0 * lsne y 0, lsne x 0 * y↦0) = lsne x 0 * lsne y 0

  pjoin(x↦y * lsne y 0, lsne x 0 * y↦0) = undefined.

  pjoin(y=0 ∧ lsne x y, x↦y * lsne y 0) = lsne x y * lspe y 0



- Overapproximation :

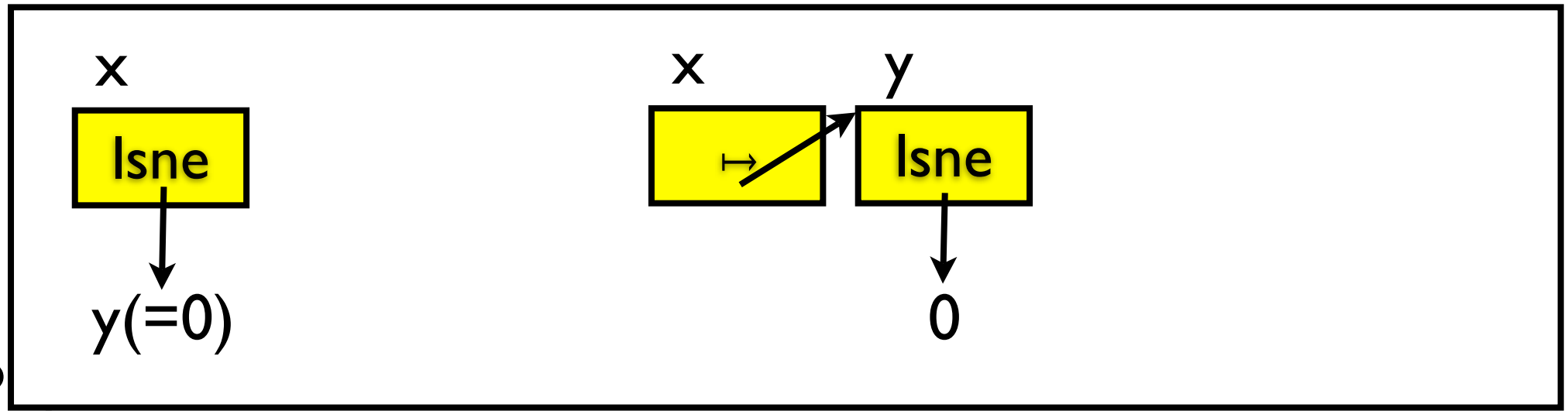
  If pjoin(q, q')=q", then (q ∨ q') ⟹ q"

- E.g.

  pjoin(x↦0 * lsne y 0, lsne x 0 * y↦0) = lsne x 0 * lsne y 0

  pjoin(x↦y * lsne y 0, lsne x 0 * y↦0) = undefined.

  pjoin(y=0 ∧ lsne x y, x↦y * lsne y 0) = lsne x y * lspe y 0



- Overapproximation :

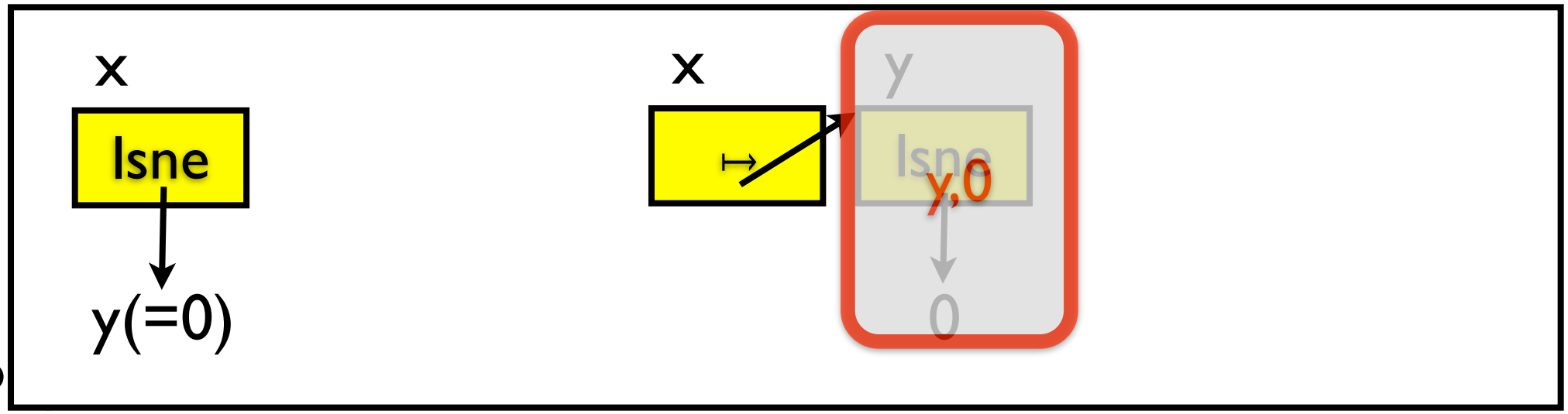
  If pjoin(q, q')=q", then $(q \vee q') \Longrightarrow q$"

- E.g.

  pjoin(x↦0 * lsne y 0, lsne x 0 * y↦0) = lsne x 0 * lsne y 0

  pjoin(x↦y * lsne y 0, lsne x 0 * y↦0) = undefined.

  pjoin(y=0 ∧ lsne x y, x↦y * lsne y 0) = lsne x y * lspe y 0



- Overapproximation :

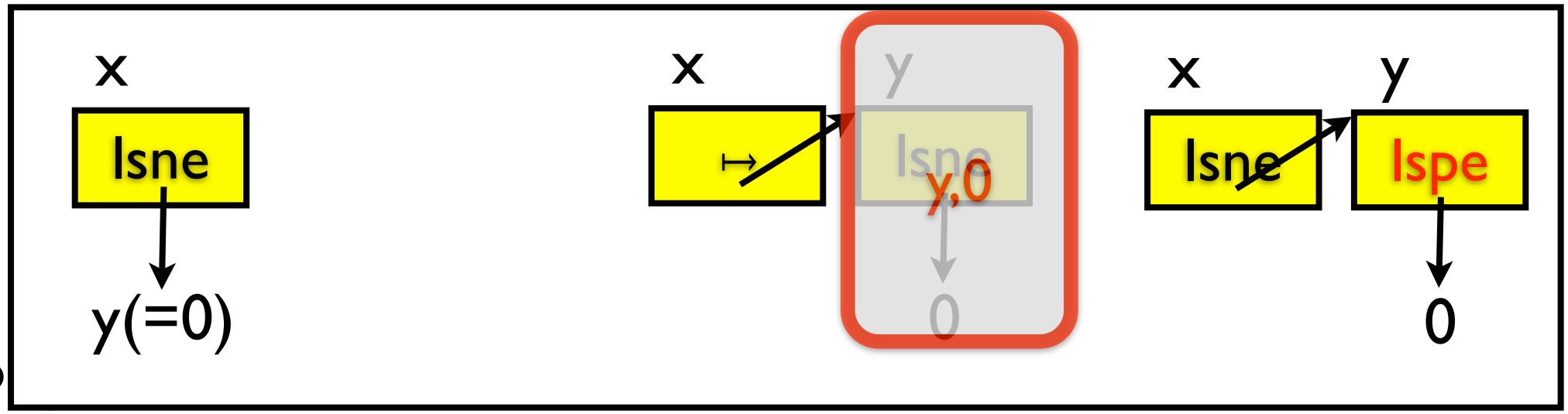
  If pjoin(q, q')=q", then $(q \vee q') \Rightarrow q$"

- E.g.

  pjoin(x↦0 * lsne y 0, lsne x 0 * y↦0) = lsne x 0 * lsne y 0

  pjoin(x↦y * lsne y 0, lsne x 0 * y↦0) = undefined.

  pjoin(y=0 ∧ lsne x y, x↦y * lsne y 0) = lsne x y * lspe y 0



- Overapproximation :

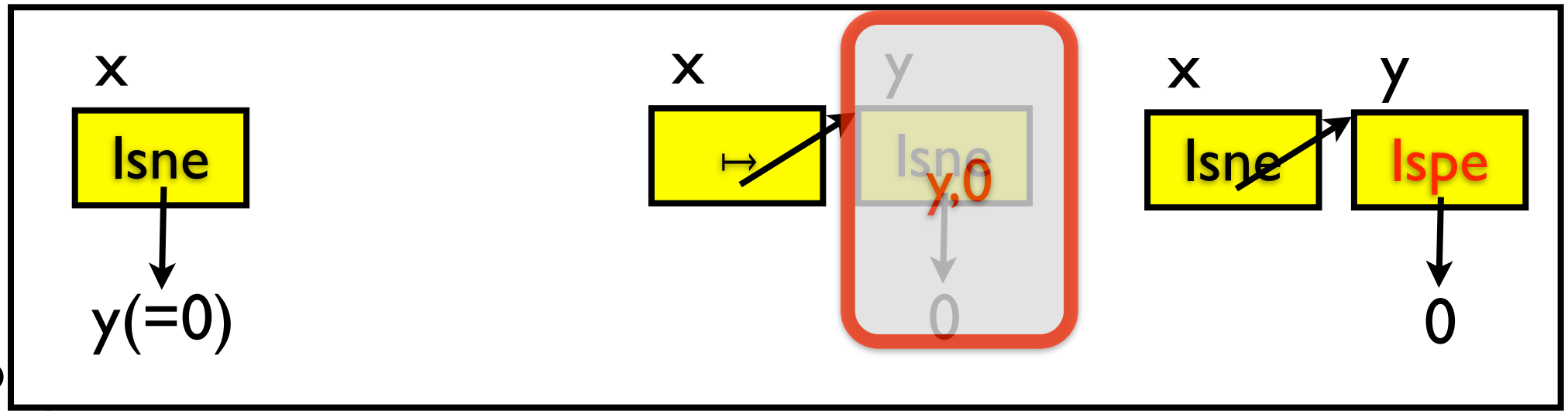
  If pjoin(q, q')=q", then $(q \vee q') \Rightarrow q$"

- E.g.

  pjoin(x↦0 * lsne y 0, lsne x 0 * y↦0) = lsne x 0 * lsne y 0

  pjoin(x↦y * lsne y 0, lsne x 0 * y↦0) = undefined.

  pjoin(y=0 ∧ lsne x y, x↦y * lsne y 0) = lsne x y * lspe y 0



1. Most tricky part.
2. Discovers, on the fly, which nodes to forget.

pjoin(x↦0 * lsne y 0, lsne x 0 * y↦0) = lsne x 0 * lsne y 0

pjoin(x↦y * lsne y 0, lsne x 0 * y↦0) = undefined.

pjoin(y=0 ∧ lsne x y, x↦y * lsne y 0) = lsne x y * lspe y 0

Creation of Two Lists

```
L x = 0;
L y = 0;
while (nondet)
{
   L t;
   t = malloc();
   t->next=x;
   x=t;

   t = malloc();
   t->next=y;
   y=t;

}
```

$x=y=0 \wedge emp$ ... $x \mapsto 0 * y \mapsto 0$ $lsne\,x\,0 * lsne\,y\,0$ ... $lsne\,x\,0 * lsne\,y\,0$

- 9 entries.
- In Firewire, more than 5000 cases generated.

Creation of Two Lists

```
L x = 0;
L y = 0;
while (nondet)
{
   L t;
   t = malloc();
   t->next=x;
   x=t;

   t = malloc();
   t->next=y;
   y=t;

}
```

$x=y=0 \wedge emp$ ... $x \mapsto 0 * y \mapsto 0$ $lsne\,x\,0 * lsne\,y\,0$ ... $lsne\,x\,0 * lsne\,y\,0$

- 9 entries.
- In Firewire, more than 5000 cases generated.

Creation of Two Lists

```
L x = 0;
L y = 0;
while (nondet)
{
   L t;
   t = malloc();
   t->next=x;
   x=t;

   t = malloc();
   t->next=y;
   y=t;

}
```

lspe x 0 * lspe y 0

- 9 entries.
- In Firewire, more than 5000 cases generated.

Creation of Two Lists

```
L x = 0;
L y = 0;
while (nondet)
{
   L t;
   t = malloc();
   t->next=x;
   x=t;

   t = malloc();
   t->next=y;
   y=t;

}
```

lspe x 0 * lspe y 0

- 9 entr
- In Fire
5000 ca

- 1 entry here.
- 1 case also in Firewire.

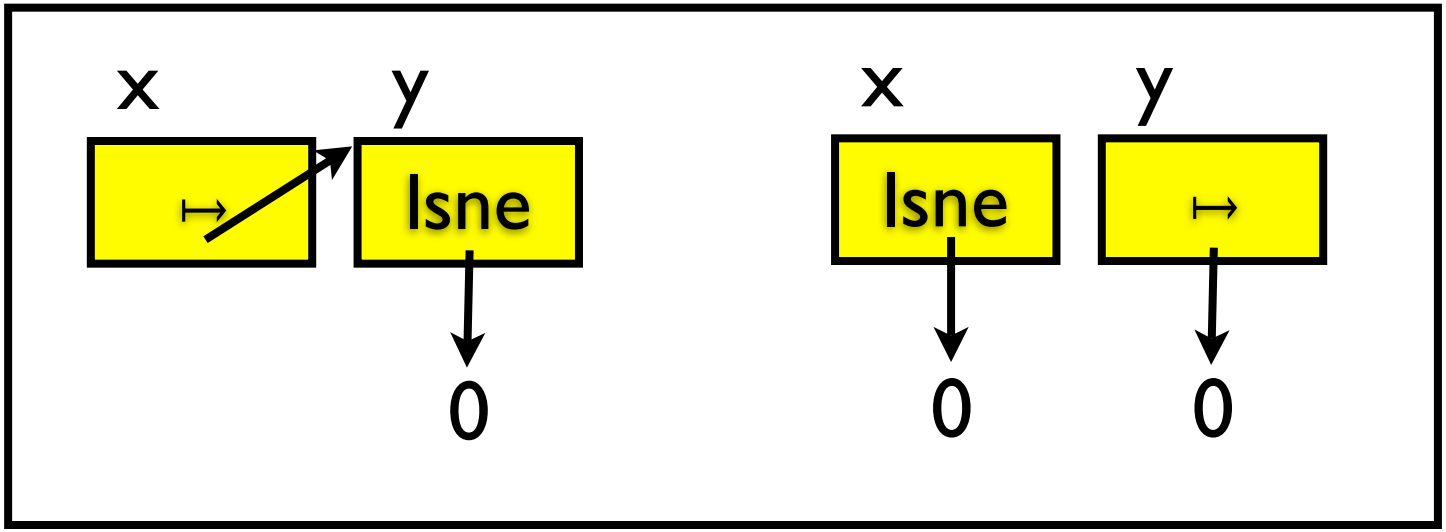
Why Partial Join?

Prevent the analysis from misusing existential variables and losing crucial shape information.

$$\{\ x \mapsto y * \mathrm{lsne}\ y\ 0 \ \vee\ \mathrm{lsne}\ x\ 0 * y \mapsto 0\ \}$$

```
free_list(x)
```

$$\{\ \mathrm{emp} \ \vee\ y \mapsto 0\ \}$$

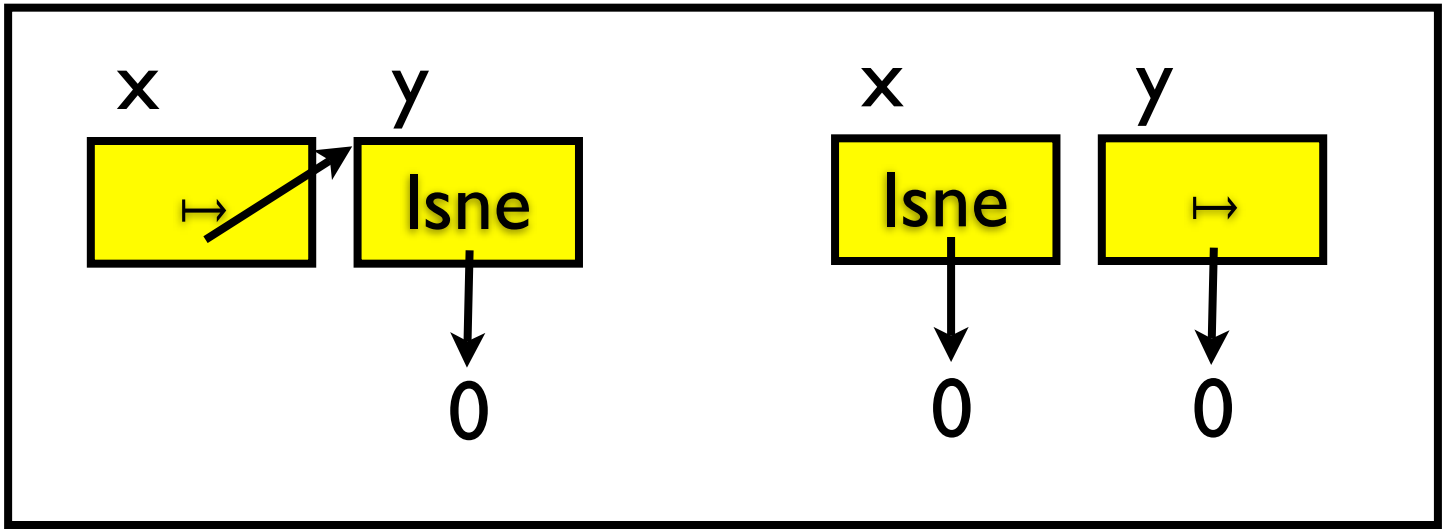


Prevent the analysis from misusing existential variables and losing crucial shape information.

$$\{\ x{\mapsto}y * \mathsf{lsne}\ y\ 0 \ \vee\ \mathsf{lsne}\ x\ 0 * y{\mapsto}0\ \}$$

```
free_list(x)
```

$$\{\ \mathsf{emp} \ \vee\ y{\mapsto}0\ \}$$

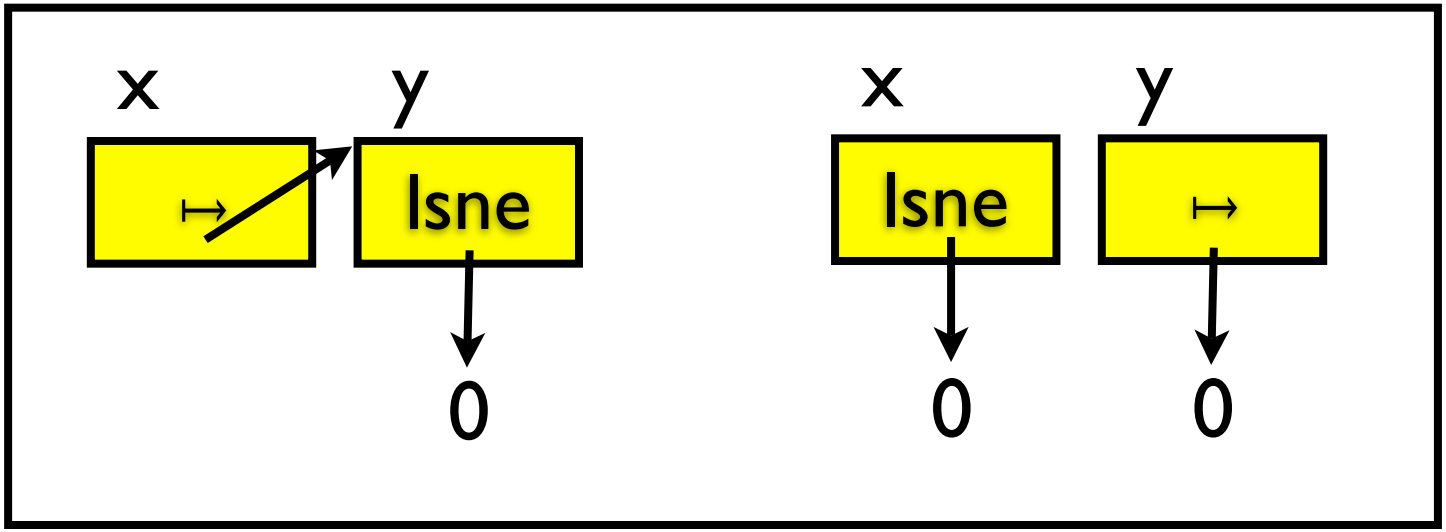


Prevent the analysis from misusing existential variables and losing crucial shape information.

$$\{\ x{\mapsto}y * \mathsf{lsne}\ y\ 0 \ \vee\ \mathsf{lsne}\ x\ 0 * y{\mapsto}0\ \}$$

```
free_list(x)
```

$$\{\ \mathsf{emp} \ \vee\ y{\mapsto}0\ \}$$

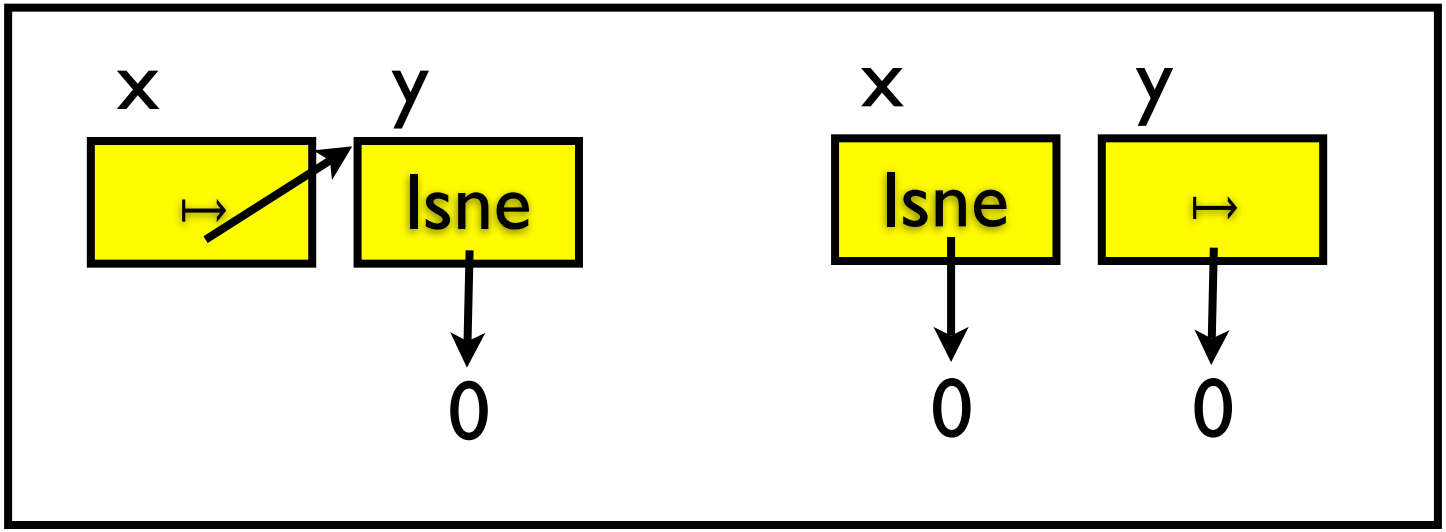


Prevent the analysis from misusing existential variables and losing crucial shape information.

{ lsne x x' * lsne y 0 }

```
free_list(x)
```

{ }



Prevent the analysis from misusing existential variables and losing crucial shape information.

{ lsne x x' * lsne y 0 }

```
free_list(x)
```

{ ???? }

Why Partial Join?

Keep important co-relation between value and shape.

$\{ x \mapsto 0 \}$

```
  y=t1394Diag_IoControl(x);
```

$\{ y=0 * emp \ \vee \ y!=0 * x \mapsto 0 \}$

```
  if (y!=0) free(x);
```

$\{ emp \}$

Why Partial Join?

Keep important co-relation between value and shape.

$\{ x \mapsto 0 \}$

```
  y=t1394Diag_IoControl(x);
```

$\{ y=0 * emp \ \vee \ y!=0 * x \mapsto 0 \}$

```
  if (y!=0) free(x);
```

$\{ emp \}$

Why Partial Join?

Keep important co-relation between value and shape.

{ x↦0 }

```
  y=t1394Diag_IoControl(x);
```

{ lspe x 0 }

```
  if (y!=0) free(x);
```

{ }

Why Partial Join?

Keep important co-relation between value and shape.

{ x↦0 }

```
  y=t1394Diag_IoControl(x);
```

{ lspe x 0 }

```
  if (y!=0) free(x);
```

{ ???? }

Many other optimizations are used in SpaceInvader.

Messages

- A dramatic progress in shape analysis in the past three years, from the analysis of 50 LOC to that of 10KLOC.
- Worthwhile to revisit old difficult problems in advanced compilation (such as automatic parallelization) with this new technology.