Program Verification using Separation Logic

Lecture 0 : Course Introduction and Assertion Language

Hongseok Yang (Queen Mary, Univ. of London)

Dream

- Automatically verify the memory safety of systems software, such as OS kernels and web servers.
- The course is about the use of separation logic for realizing this dream.

```
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
```

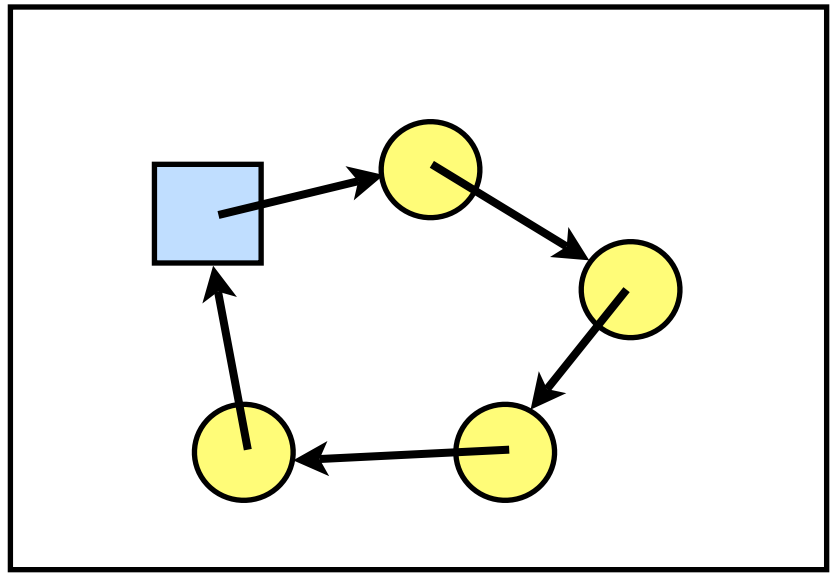
```
typedef struct BUS_RESET_IRP {
  struct BUS_RESET_IRP *Flink2;
  PIRP                 Irp;
} BUS_RESET_IRP, *PBUS_RESET_IRP;


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

Windows Firewire Driver
Q: Is this correct?

```
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;
```

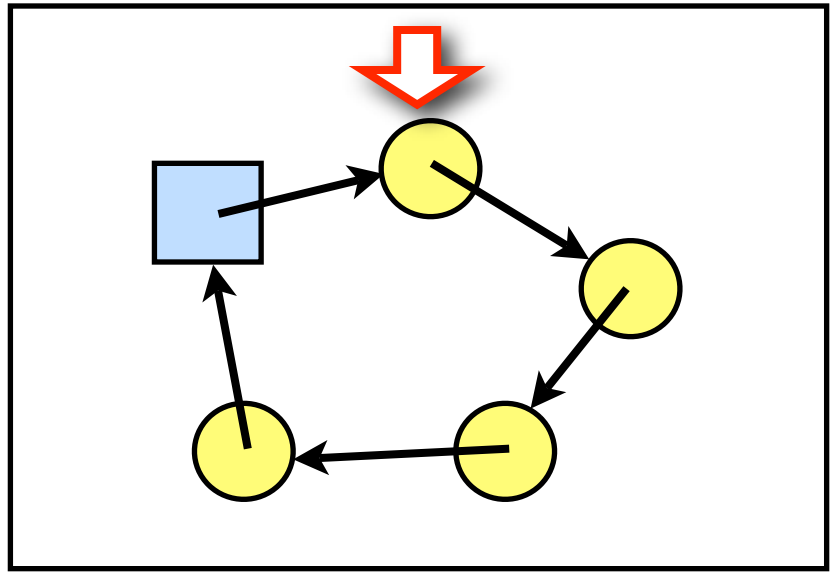
Windows Firewire Driver
Q: Is this correct?



```
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;
```

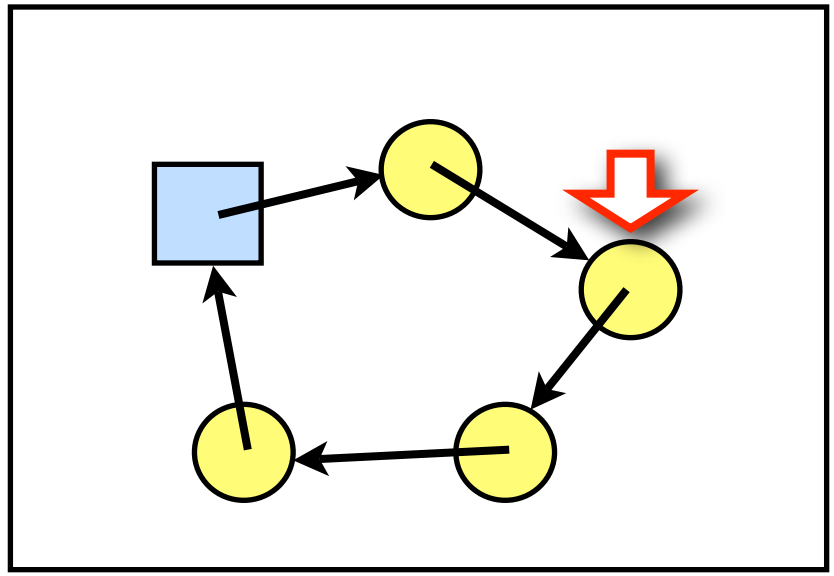
Windows Firewire Driver
Q: Is this correct?



```
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;
```

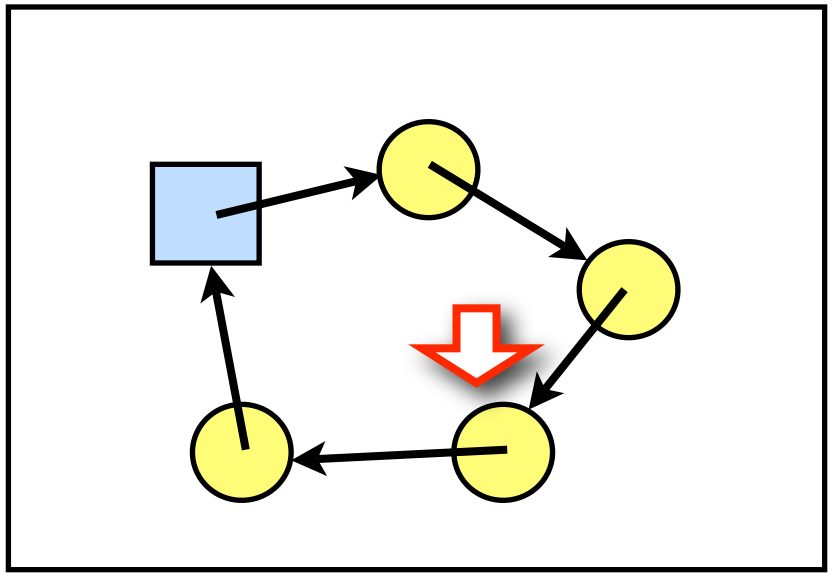
Windows Firewire Driver
Q: Is this correct?



```
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;
```

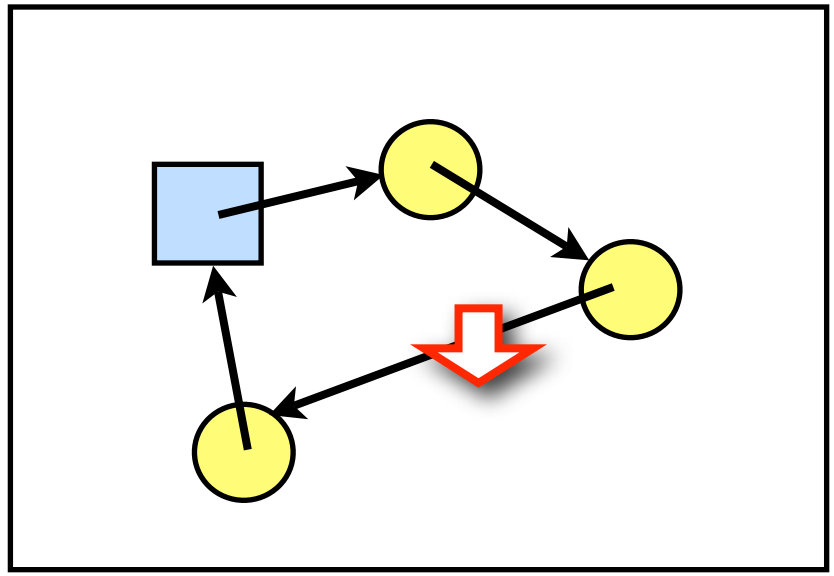
Windows Firewire Driver
Q: Is this correct?



```
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;
```

Windows Firewire Driver
Q: Is this correct?



Demo

```
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
```

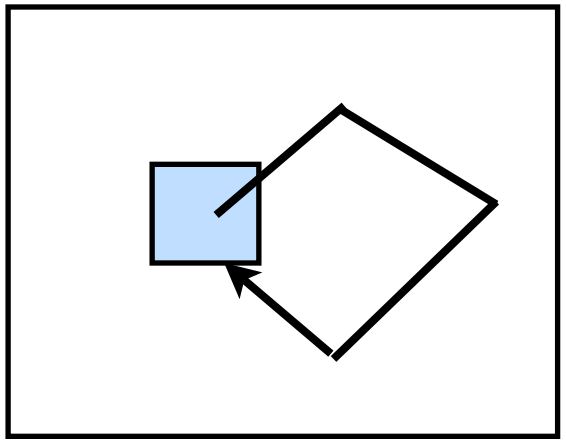
```
typedef struct BUS_RESET_IRP {
  struct BUS_RESET_IRP *Flink2;
  PIRP                 Irp;
} BUS_RESET_IRP, *PBUS_RESET_IRP;


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

Windows Firewire Driver
Q: Is this correct?

```
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;
```

Windows Firewire Driver
Q: Is this correct?

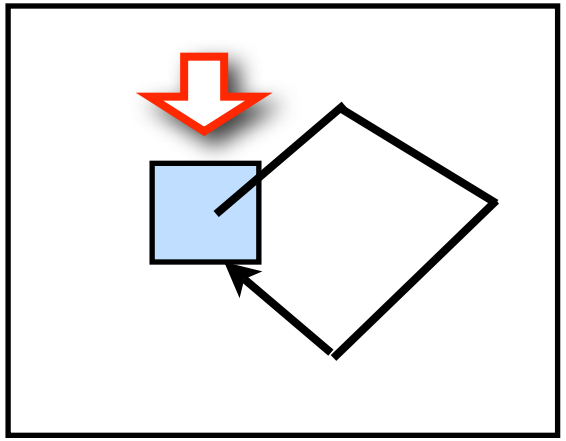


```
void t1394Diag_CancelIrp(PDEVICE_OBJECT DeviceObject, PIRP Irp)
{
  KIRQL                 Irql, CancelIrql;
  BUS_RESET_IRP         *BusResetIrp, *temp;
  PDEVICE_EXTENSION     deviceExtension;

  deviceExtension = DeviceObject->DeviceExtension;

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

  temp = (PBUS_RESET_IRP)deviceExtension;
  BusResetIrp=(...)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;
```

Windows Firewire Driver

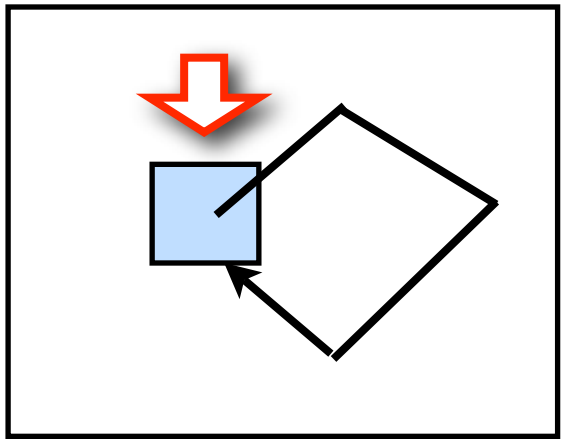
Q: Is this correct?



```
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;
```

Windows Firewire Driver
Q: Is this correct?

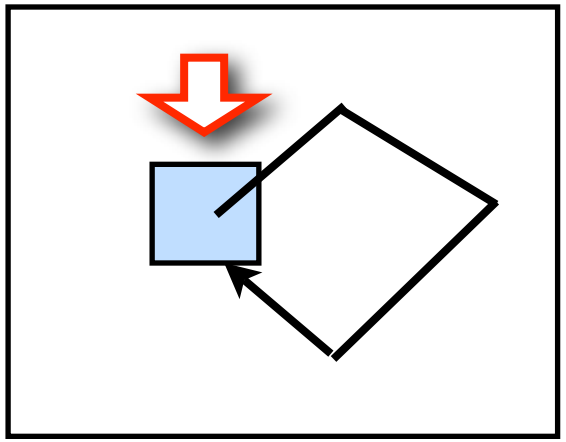


```
void t1394Diag_CancelIrp(PDEVICE_OBJECT DeviceObject, PIRP Irp)
{
  KIRQL                Irql, CancelIrql;
  BUS_RESET_IRP        *BusResetIrp, *temp;
  PDEVICE_EXTENSION    deviceExtension;

  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;
```

Windows Firewire Driver

Q: Is this correct?



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

Separation Logic

- A variant of Hoare logic that lets you verify pointer-manipulating programs effectively.
- Philosophy: formalize and exploit good disciplines used by top-ranked systems programmers.

Very Rough History

- 1995, Discoveries in categorical semantics.
- 1999, Logic of Bunched Implications.
- 2001, Separation logic.
- 2003~, Application to data abstraction.
- 2004~, Application to concurrency.
- 2005~, Application to program analysis.

Outcome of this Course

- Understand the basic principles of separation logic.
- Should be able to build a simple automatic verifier based on separation logic.

Schedule

- Lecture 0: Assertion language.
- Lecture 1: Proof rules in sep. logic.
- Lecture 2: Simple automatic verifier.
- Lecture 3: Frame rule.
- Lecture 4: Concurrency.

Starting Point

- Should be able to express properties at each program point effectively.

```
x = create_list();
y = create_list();
z = create_sorted_list();
x = append_list(x,y);
free_list(x);
traverse_list(z);
```

Starting Point

- Should be able to express properties at each program point effectively.

1. lists x,y,z.
2. lists x,y: disjoint.
3. lists y,z: disjoint.
4. lists x,z: disjoint.
5. list z sorted.

```
x = create_list();
y = create_list();
z = create_sorted_list();
x = append_list(x,y);
free_list(x);
traverse_list(z);
```

Starting Point

- Should be able to express properties at each program point effectively.

```
x = create_list();
y = create_list();
z = create_sorted_list();
x = append_list(x,y);
free_list(x);
traverse_list(z);
```

1. lists x,y,z.
2. lists x,y: disjoint.
3. lists y,z: disjoint.
4. lists x,z: disjoint.
5. list z sorted.

1. lists x,y,z.
2. list x contains y.
3. lists x,z: disjoint.
4. list z sorted.

Starting Point

- Should be able to express properties at each program point effectively.

```
x = create_list();
y = create_list();
z = create_sorted_list();
x = append_list(x,y);
free_list(x);
traverse_list(z);
```

1. lists x,y,z.
2. lists x,y: disjoint.
3. lists y,z: disjoint.
4. lists x,z: disjoint.
5. list z sorted.

1. lists x,y,z.
2. list x contains y.
3. lists x,z: disjoint.
4. list z sorted.

Express implicit/explicit, state-dependent separation effectively.

Toolkit of Separation-Logic Maniac

1. A class of (separation) properties to express.
2. Pick a storage model with separation built-in.
   - Means to find a partial commutative monoid.
3. Apply a general method for constructing an assertion language with separating connectives.

Toolkit of Separation-Logic Maniac

1. A class of (separation) properties to express.
2. Pick a storage model with separation built-in.
   - Means to find a partial commutative monoid.
3. Apply a general method for constructing an assertion language with separating connectives.

Storage Model with Separation

- Principle: Find a local description of state (or resource). Define a splitting operator.
- Technical, find a partial commutative monoid.
- E.g.
  - Global heap model: Heaps $\stackrel{def}{=}$ Locs $\rightarrow$ Vals
  - [1:0, 2:0, 3:0, ...], [1:3, 2:0, 3:4, 4:0, 5:0, ...]
  - Local heap model: Heaps $\stackrel{def}{=}$ Locs $\rightarrow_{\mathsf{fin}}$ Vals
  - [], [2:0, 5:0], [1:3, 3:4, 4:0], [1:3, 3:4, 4:0, 8:0, 9:8]

Partial Commutative Monoid

- A set M together with
  - a unit e in M and a partial binary operator * on M.
- Should satisfy:
  - identity: e * m = m * e = m
  - commutativity: m * m' = m' * m
  - associativity: m * (m' * m") = (m * m') * m"
- m = m' * m" implies that m' and m" are separate.

PCM Structure of Heaps

$$\text{Heaps} \stackrel{def}{=} \text{Locs} \rightarrow_{\text{fin}} \text{Vals}$$

- Separateness predicates h#h':

$$h_1 \# h_2 \quad \stackrel{def}{\Longleftrightarrow} \quad \text{dom}(h_1) \cap \text{dom}(h_2) = \emptyset$$

- Partial heap combining operator h*h':

$$h_1 * h_2 \quad \stackrel{def}{=} \quad \begin{cases} h_1 \cup h_2 & \text{if } h_1 \# h_2 \\ \text{undefined} & \text{otherwise} \end{cases}$$

- The empty heap [] is the unit for *.

1) [] * [11:98] = [11:98]
2) [6:11, 98:0] * [11:98] = [6:11, 11:98, 98:0]
3) [6:11, 11:0] * [11:98] = undefined
4) [11:98] * [11:98] = undefined

$$\text{Heaps} \stackrel{def}{=} \text{Locs} \rightarrow_{\text{fin}} \text{Vals}$$

- Separateness predicates h#h':

$$h_1 \# h_2 \quad \stackrel{def}{\Longleftrightarrow} \quad \text{dom}(h_1) \cap \text{dom}(h_2) = \emptyset$$

- Partial heap combining operator h*h':

$$h_1 * h_2 \quad \stackrel{def}{=} \quad \begin{cases} h_1 \cup h_2 & \text{if } h_1 \# h_2 \\ \text{undefined} & \text{otherwise} \end{cases}$$

- The empty heap [] is the unit for *.

Storage Model in Detail

$$\mathsf{Vars} \stackrel{def}{=} \{x, y, z, \ldots\}$$

$$\mathsf{Locs} \stackrel{def}{=} \{1, 2, 3, 4, \ldots\} \qquad \mathsf{Vals} \supseteq \mathsf{Locs}$$

$$\mathsf{Heaps} \stackrel{def}{=} \mathsf{Locs} \rightarrow_{\mathsf{fin}} \mathsf{Vals}$$

$$\mathsf{Stacks} \stackrel{def}{=} \mathsf{Vars} \rightarrow \mathsf{Vals}$$

$$\mathsf{States} \stackrel{def}{=} \mathsf{Stacks} \times \mathsf{Heaps}$$

Storage Model in Detail

$$\text{Vars} \stackrel{def}{=} \{x, y, z, \ldots\}$$

$$\text{Locs} \stackrel{def}{=} \{1, 2, 3, 4, \ldots\} \qquad \text{Vals} \supseteq \text{Locs}$$

$$\text{Heaps} \stackrel{def}{=} \text{Locs} \rightarrow_{\text{fin}} \text{Vals}$$

$$\text{Stacks} \stackrel{def}{=} \text{Vars} \rightarrow \text{Vals}$$

$$\text{States} \stackrel{def}{=} \text{Stacks} \times \text{Heaps}$$

1) Each state consists of stack s and heap h.

Storage Model in Detail

$$\mathsf{Vars} \stackrel{def}{=} \{x, y, z, \ldots\}$$

$$\mathsf{Locs} \stackrel{def}{=} \{1, 2, 3, 4, \ldots\} \qquad \mathsf{Vals} \supseteq \mathsf{Locs}$$

$$\mathsf{Heaps} \stackrel{def}{=} \mathsf{Locs} \rightarrow_{\mathsf{fin}} \mathsf{Vals}$$

$$\mathsf{Stacks} \stackrel{def}{=} \mathsf{Vars} \rightarrow \mathsf{Vals}$$

$$\mathsf{States} \stackrel{def}{=} \mathsf{Stacks} \times \mathsf{Heaps}$$

1) Each state consists of stack s and heap h.
2) Complete stacks, but partial finite "local" heaps.

Storage Model in Detail

$$\mathsf{Vars} \stackrel{def}{=} \{x, y, z, \ldots\}$$
$$\mathsf{Locs} \stackrel{def}{=} \{1, 2, 3, 4, \ldots\} \qquad \mathsf{Vals} \supseteq \mathsf{Locs}$$

$$\mathsf{Heaps} \stackrel{def}{=} \mathsf{Locs} \rightarrow_{\mathsf{fin}} \mathsf{Vals}$$
$$\mathsf{Stacks} \stackrel{def}{=} \mathsf{Vars} \rightarrow \mathsf{Vals}$$
$$\mathsf{States} \stackrel{def}{=} \mathsf{Stacks} \times \mathsf{Heaps}$$

1) Each state consists of stack s and heap h.
2) Complete stacks, but partial finite "local" heaps.
3) Allows pointer arithmetic, but not &x in C.

E.g.
1) ([x:0,y:0,...], [])
2) ([x:6,y:98,...], [6:11, 11:98, 98:0])
3) ([x:6,y:98,...], [6:11, 11:12, 12:0, 98:99, 99:100])

$$\text{Vars} \stackrel{def}{=} \{x, y, z, \ldots\}$$

$$\text{Locs} \stackrel{def}{=} \{1, 2, 3, 4, \ldots\} \qquad \text{Vals} \supseteq \text{Locs}$$

$$\text{Heaps} \stackrel{def}{=} \text{Locs} \rightarrow_{\text{fin}} \text{Vals}$$

$$\text{Stacks} \stackrel{def}{=} \text{Vars} \rightarrow \text{Vals}$$

$$\text{States} \stackrel{def}{=} \text{Stacks} \times \text{Heaps}$$

1) Each state consists of stack s and heap h.
2) Complete stacks, but partial finite "local" heaps.
3) Allows pointer arithmetic, but not &x in C.

Storage Model in Detail

$$\mathsf{Vars} \stackrel{def}{=} \{x, y, z, \ldots\}$$

$$\mathsf{Locs} \stackrel{def}{=} \{1, 2, 3, 4, \ldots\} \qquad \mathsf{Vals} \supseteq \mathsf{Locs}$$

$$\mathsf{Heaps} \stackrel{def}{=} \mathsf{Locs} \rightarrow_{\mathsf{fin}} \mathsf{Vals}$$

$$\mathsf{Stacks} \stackrel{def}{=} \mathsf{Vars} \rightarrow \mathsf{Vals}$$

$$\mathsf{States} \stackrel{def}{=} \mathsf{Stacks} \times \mathsf{Heaps}$$

Storage Model in Detail

General

$$\mathsf{Vars} \stackrel{def}{=} \{x, y, z, \ldots\}$$

$$\mathsf{Locs} \stackrel{def}{=} \{1, 2, 3, 4, \ldots\} \qquad \mathsf{Vals} \supseteq \mathsf{Locs}$$

$$\mathsf{Heaps} \stackrel{def}{=} \mathsf{Locs} \rightarrow_{\mathsf{fin}} \mathsf{Vals}$$

$$\mathsf{Stacks} \stackrel{def}{=} \mathsf{Vars} \rightarrow \mathsf{Vals}$$

$$\mathsf{States} \stackrel{def}{=} \mathsf{Stacks} \times \mathsf{Heaps}$$

Storage Model in Detail

General

$$\text{Vars} \stackrel{def}{=} \{x, y, z, \ldots\}$$

$$\text{Stacks} \stackrel{def}{=} \text{Vars} \rightarrow \text{Vals}$$

$$\text{States} \stackrel{def}{=} \text{Stacks} \times \text{Heaps}$$

Storage Model in Detail

General

$$\text{Vars} \stackrel{def}{=} \{x, y, z, \ldots\}$$

$$\text{Stacks} \stackrel{def}{=} \text{Vars} \rightarrow \text{Vals}$$

$$\text{States} \stackrel{def}{=} \text{Stacks} \times \text{Heaps}$$

$$\boxed{\text{States} \stackrel{def}{=} \text{Stacks} \times M}$$

Toolkit of Separation-Logic Maniac

1. A class of (separation) properties to express.
2. Pick a storage model with separation built-in.
   - Means to find a partial commutative monoid.
3. Apply a general method for constructing an assertion language with separating connectives.

General Method

- Constructs an assertion language and its semantics.
- Each assertion P specifies a set of states:

$$\text{“Meaning of } P\text{”} \subseteq \text{ States}$$

- Recipe:

1. Re-use all the connectives from classical logic.
2. Add separating connectives: emp, P * Q, P -* Q
3. Define semantics using the PCM structure.

General Method

- Constructs an assertion language and its semantics.
- Each assertion P specifies a set of states:

$$\text{“Meaning of } P\text{”} \subseteq \text{States}$$

- Recipe:

$$\text{States} \stackrel{def}{=} \text{Stacks} \times M$$

1. Re-use all the connectives from classical logic.
2. Add separating connectives: emp, P * Q, P -*- Q
3. Define semantics using the PCM structure.

General Method

- Constructs an assertion language and its semantics.
- Each assertion P specifies a set of states:

$$\text{“Meaning of } P\text{”} \subseteq \text{States}$$

- Recipe:

$$\text{States} \stackrel{def}{=} \text{Stacks} \times M$$

1. Re-use all the connectives from classical logic.
2. Add separating connectives: emp, P * Q, P -* Q
3. Define semantics using the PCM structure.

Ignore separating implication.
HW: Include the separating implication.

Assertion for Our Storage Model

- Specifies a set of states:

$$\text{States} \stackrel{def}{=} \text{Stacks} \times \text{Heaps}$$

$$\text{“Meaning of } P\text{”} \subseteq \text{States}$$

- Syntax of expressions E and assertions P

$$\begin{array}{lcll} E, F & ::= & x \mid n \mid E{+}F \mid -E \mid \ldots & \text{Heap-independent Exprs} \\ P, Q & ::= & E = F \mid E \geq F \mid E \mapsto F & \text{Atomic Predicates} \\ & \mid & \text{emp} \mid P * Q & \text{Separating Connectives} \\ & \mid & \text{true} \mid P \wedge Q \mid \neg P \mid \forall x.\, P & \text{Classical Logic} \end{array}$$

Assertion for Our Storage Model

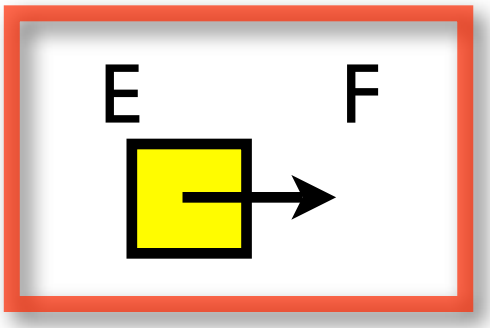
- Specifies a set of states:

$$\text{States} \stackrel{def}{=} \text{Stacks} \times \text{Heaps}$$

$$\text{“Meaning of } P\text{”} \subseteq \text{States}$$

- Syntax of expressions E and assertions P

$$\begin{array}{lcll} E, F & ::= & x \mid n \mid E{+}F \mid -E \mid \ldots & \text{Heap-independent Exprs} \\ P, Q & ::= & E = F \mid E \geq F \mid E \mapsto F & \text{Atomic Predicates} \\ & \mid & \text{emp} \mid P * Q & \text{Separating Connectives} \\ & \mid & \text{true} \mid P \wedge Q \mid \neg P \mid \forall x.\, P & \text{Classical Logic} \end{array}$$

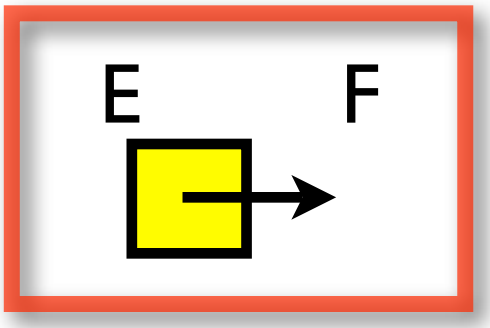


E.g. $1 \mapsto 0, \quad x \mapsto y$

([x:1,y:0,...], [1:0]),
([x:1,y:2,...], [1:0]),
([x:1,y:2,...], [1:2]),
([x:1,y:0,...], [1:0, 2:0])

- Syntax of expressions E and assertions P

$$
\begin{array}{rcll}
E, F & ::= & x \mid n \mid E{+}F \mid -E \mid \ldots & \text{Heap-independent Exprs} \\
P, Q & ::= & E = F \mid E \geq F \mid E \mapsto F & \text{Atomic Predicates} \\
 & \mid & \text{emp} \mid P * Q & \text{Separating Connectives} \\
 & \mid & \text{true} \mid P \wedge Q \mid \neg P \mid \forall x.\, P & \text{Classical Logic}
\end{array}
$$



Assertion for Our Storage Model

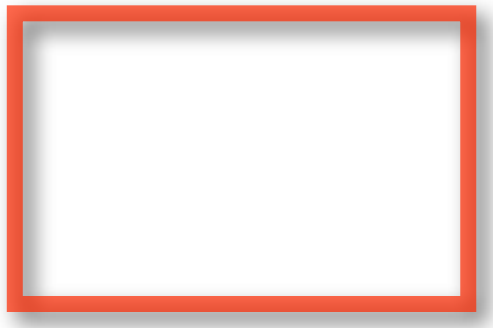
- Specifies a set of states:

$$\text{States} \stackrel{def}{=} \text{Stacks} \times \text{Heaps}$$

$$\text{“Meaning of } P\text{”} \subseteq \text{States}$$

- Syntax of expressions E and assertions P

$$\begin{array}{lcll} E, F & ::= & x \mid n \mid E{+}F \mid -E \mid \ldots & \text{Heap-independent Exprs} \\ P, Q & ::= & E = F \mid E \geq F \mid E \mapsto F & \text{Atomic Predicates} \\ & \mid & \text{emp} \mid P * Q & \text{Separating Connectives} \\ & \mid & \text{true} \mid P \wedge Q \mid \neg P \mid \forall x.\, P & \text{Classical Logic} \end{array}$$



Assertion for Our Storage Model

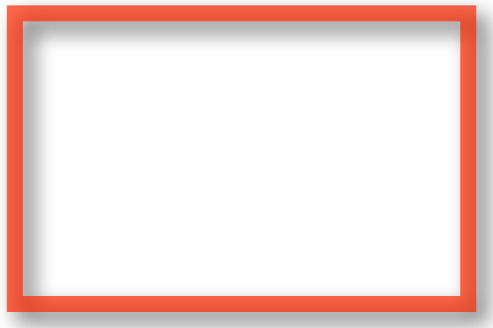
- Specifies a set of states:

$$\text{States} \stackrel{def}{=} \text{Stacks} \times \text{Heaps}$$

$$\text{“Meaning of } P\text{”} \subseteq \text{States}$$

- Syntax of expressions E and assertions P

$$\begin{array}{rcll} E, F & ::= & x \mid n \mid E{+}F \mid -E \mid \ldots & \text{Heap-independent Exprs} \\ P, Q & ::= & E = F \mid E \geq F \mid E \mapsto F & \text{Atomic Predicates} \\ & \mid & \text{emp} \mid P * Q & \text{Separating Connectives} \\ & \mid & \text{true} \mid P \wedge Q \mid \neg P \mid \forall x.\, P & \text{Classical Logic} \end{array}$$



E.g. emp
([x:1,y:0,...], []),
([x:1,y:1,...], [1:0])

Assertion for Our Storage Model

- Specifies a set of states:

$$\text{States} \stackrel{def}{=} \text{Stacks} \times \text{Heaps}$$

$$\text{“Meaning of } P\text{”} \subseteq \text{States}$$

- Syntax of expressions E and assertions P

$$\begin{array}{lcll} E, F & ::= & x \mid n \mid E{+}F \mid -E \mid \ldots & \text{Heap-independent Exprs} \\ P, Q & ::= & E = F \mid E \geq F \mid E \mapsto F & \text{Atomic Predicates} \\ & \mid & \text{emp} \mid P * Q & \text{Separating Connectives} \\ & \mid & \text{true} \mid P \wedge Q \mid \neg P \mid \forall x.\, P & \text{Classical Logic} \end{array}$$

Assertion for Our Storage Model

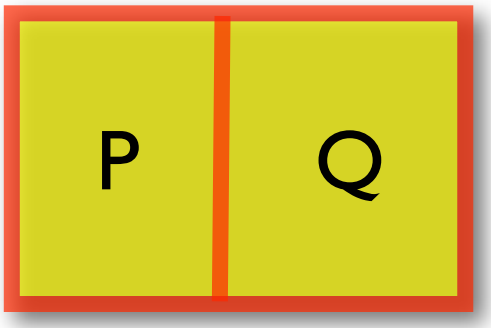
- Specifies a set of states:

$$\text{States} \stackrel{def}{=} \text{Stacks} \times \text{Heaps}$$

$$\text{“Meaning of } P\text{”} \subseteq \text{States}$$

- Syntax of expressions E and assertions P

$$\begin{array}{lcll} E, F & ::= & x \mid n \mid E{+}F \mid -E \mid \ldots & \text{Heap-independent Exprs} \\ P, Q & ::= & E = F \mid E \geq F \mid E \mapsto F & \text{Atomic Predicates} \\ & \mid & \text{emp} \mid P * Q & \text{Separating Connectives} \\ & \mid & \text{true} \mid P \wedge Q \mid \neg P \mid \forall x.\, P & \text{Classical Logic} \end{array}$$



E.g. $1 \mapsto 0 * 2 \mapsto 0, \quad x \mapsto y * y \mapsto x, \quad x \mapsto 0 * x \mapsto 0$

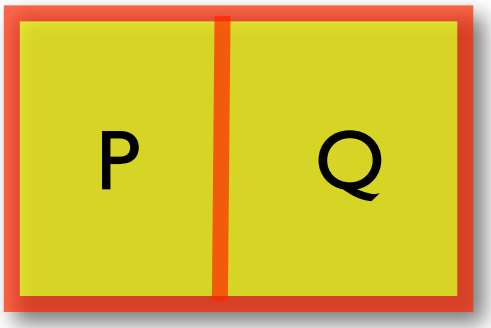
([x:1,y:2,...], [1:0, 2:0]),
([x:1,y:2,...], [1:2, 2:1]),
([x:2,y:1,...], [1:2, 2:1])

ps

"Meaning of P" $\subseteq$ States

- Syntax of expressions E and assertions P

$$
\begin{array}{lcll}
E, F & ::= & x \mid n \mid E{+}F \mid -E \mid \ldots & \text{Heap-independent Exprs} \\
P, Q & ::= & E = F \mid E \geq F \mid E \mapsto F & \text{Atomic Predicates} \\
 & \mid & \text{emp} \mid P * Q & \text{Separating Connectives} \\
 & \mid & \text{true} \mid P \wedge Q \mid \neg P \mid \forall x.\, P & \text{Classical Logic}
\end{array}
$$



E.g. $1 \mapsto 0 * 2 \mapsto 0$, $x \mapsto y * y \mapsto x$, $x \mapsto 0 * x \mapsto 0$

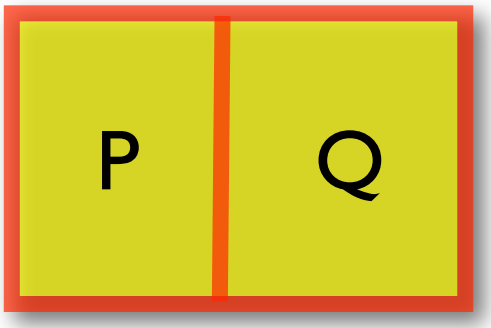
([x:1,y:2,...], [1:0, 2:0]),
([x:1,y:2,...], [1:2, 2:1]),
([x:2,y:1,...], [1:2, 2:1])

ps

"Meaning of P" $\subseteq$ States

- Syntax of expressions E and assertions P

$$
\begin{array}{rcll}
E, F & ::= & x \mid n \mid E{+}F \mid -E \mid \ldots & \text{Heap-independent Exprs} \\
P, Q & ::= & E = F \mid E \geq F \mid E \mapsto F & \text{Atomic Predicates} \\
 & \mid & \text{emp} \mid P * Q & \text{Separating Connectives} \\
 & \mid & \text{true} \mid P \wedge Q \mid \neg P \mid \forall x.\, P & \text{Classical Logic}
\end{array}
$$



Can express the separation of arbitrary formulas in a state-dependent manner.

Assertion for Our Storage Model

- Specifies a set of states:

$$\text{States} \stackrel{def}{=} \text{Stacks} \times \text{Heaps}$$

$$\text{“Meaning of } P\text{”} \subseteq \text{States}$$

- Syntax of expressions E and assertions P

$$\begin{array}{lcll} E, F & ::= & x \mid n \mid E{+}F \mid -E \mid \ldots & \text{Heap-independent Exprs} \\ P, Q & ::= & E = F \mid E \geq F \mid E \mapsto F & \text{Atomic Predicates} \\ & \mid & \text{emp} \mid P * Q & \text{Separating Connectives} \\ & \mid & \text{true} \mid P \wedge Q \mid \neg P \mid \forall x.\, P & \text{Classical Logic} \end{array}$$

1) Usual connectives from classical logic.
2) All other connectives are definable.
3) E.g. $(x \mapsto 0) \vee \exists x'.(x \mapsto x') * (x' \mapsto 0)$.

Semantics of Assertions

- Expressions mean maps from stacks to values.

$$[\![E]\!] \quad : \quad \text{Stacks} \rightarrow \text{Vals}$$

- Semantics of assertions given by satisfaction relation between states and assertions.

$$(s, h) \quad \models \quad P$$

- s in Stacks, h in Heaps.

Semantics of Assertions

$$
\begin{array}{lll}
(s,h) \models E \geq F & \text{iff} & [\![E]\!]s, [\![F]\!]s \in \mathsf{Integers} \text{ and } [\![E]\!]s \geq [\![F]\!]s \\
(s,h) \models E \mapsto F & \text{iff} & \mathsf{dom}(h) = \{[\![E]\!]s\} \text{ and } h([\![E]\!]s) = [\![F]\!]s \\
(s,h) \models \mathsf{emp} & \text{iff} & h = [] \text{ (i.e., } \mathsf{dom}(h) = \emptyset) \\
(s,h) \models P * Q & \text{iff} & \exists h_0 h_1.\ h_0 * h_1 = h,\ (s,h_0) \models P \text{ and } (s,h_1) \models Q \\
(s,h) \models \mathsf{true} & & \text{always} \\
(s,h) \models P \wedge Q & \text{iff} & (s,h) \models P \text{ and } (s,h) \models Q \\
(s,h) \models \neg P & \text{iff} & \text{not } ((s,h) \models P) \\
(s,h) \models \forall x.\, P & \text{iff} & \forall v \in \mathsf{Vals}.\ (s[x \mapsto v], h) \models P)
\end{array}
$$

Semantics of Assertions

$(s,h) \models E \geq F$ iff $[\![E]\!]s, [\![F]\!]s \in \mathsf{Integers}$ and $[\![E]\!]s \geq [\![F]\!]s$

$(s,h) \models E \mapsto F$ iff $\mathsf{dom}(h) = \{[\![E]\!]s\}$ and $h([\![E]\!]s) = [\![F]\!]s$

$(s,h) \models \mathsf{emp}$ iff $h = []$ (i.e., $\mathsf{dom}(h) = \emptyset$)

$(s,h) \models P * Q$ iff $\exists h_0 h_1.\ h_0 * h_1 = h,\ (s,h_0) \models P$ and $(s,h_1) \models Q$

$(s,h) \models \mathsf{true}$ always

$(s,h) \models P \wedge Q$ iff $(s,h) \models P$ and $(s,h) \models Q$

$(s,h) \models \neg P$ iff not $((s,h) \models P)$

$(s,h) \models \forall x.\, P$ iff $\forall v \in \mathsf{Vals}.\ (s[x \mapsto v], h) \models P$

$([x{:}1, y{:}2, ...],\ [1{:}2]) \models x \mapsto y$, but $([x{:}1, y{:}2, ...],\ [1{:}2, 2{:}1]) \not\models x \mapsto y$

$([x{:}1, y{:}2, ...],\ []) \models \mathsf{emp}$

$([x{:}1, y{:}2, ...],\ [1{:}2, 2{:}1]) \models x \mapsto y * y \mapsto x$

Assertions for Any Storage Models

~~Semantics of Assertions~~

$$
\begin{array}{lll}
(s,h) \models E \geq F & \text{iff} & [\![E]\!]s, [\![F]\!]s \in \mathsf{Integers} \text{ and } [\![E]\!]s \geq [\![F]\!]s \\
(s,h) \models E \mapsto F & \text{iff} & \mathsf{dom}(h) = \{[\![E]\!]s\} \text{ and } h([\![E]\!]s) = [\![F]\!]s \\
(s,h) \models \mathsf{emp} & \text{iff} & h = [] \text{ (i.e., } \mathsf{dom}(h) = \emptyset) \\
(s,h) \models P * Q & \text{iff} & \exists h_0 h_1.\ h_0 * h_1 = h,\ (s,h_0) \models P \text{ and } (s,h_1) \models Q \\
(s,h) \models \mathsf{true} & & \text{always} \\
(s,h) \models P \wedge Q & \text{iff} & (s,h) \models P \text{ and } (s,h) \models Q \\
(s,h) \models \neg P & \text{iff} & \text{not } ((s,h) \models P) \\
(s,h) \models \forall x.\, P & \text{iff} & \forall v \in \mathsf{Vals}.\ (s[x \mapsto v], h) \models P)
\end{array}
$$

Assertions for Any Storage Models

~~Semantics of Assertions~~

~~$(s,h) \models E \geq F$ iff $[\![E]\!]s, [\![F]\!]s \in \mathsf{Integers}$ and $[\![E]\!]s \geq [\![F]\!]s$~~

~~$(s,h) \models E \mapsto F$ iff $\mathsf{dom}(h) = \{[\![E]\!]s\}$ and $h([\![E]\!]s) = [\![F]\!]s$~~

$(s,h) \models \mathsf{emp}$ iff $h = []$ (i.e., $\mathsf{dom}(h) = \emptyset$)

$(s,h) \models P * Q$ iff $\exists h_0 h_1.\ h_0 * h_1 = h,\ (s,h_0) \models P$ and $(s,h_1) \models Q$

$(s,h) \models \mathsf{true}$ always

$(s,h) \models P \wedge Q$ iff $(s,h) \models P$ and $(s,h) \models Q$

$(s,h) \models \neg P$ iff not $((s,h) \models P)$

$(s,h) \models \forall x.\, P$ iff $\forall v \in \mathsf{Vals}.\ (s[x \mapsto v], h) \models P)$

Replace basic predicates appropriately.

Assertions for Any Storage Models

~~Semantics of Assertions~~

~~$(s,h) \models E \geq F$ iff $[\![E]\!]s, [\![F]\!]s \in$ Integers and $[\![E]\!]s \geq [\![F]\!]s$~~

~~$(s,h) \models E \mapsto F$ iff $\mathsf{dom}(h) = \{[\![E]\!]s\}$ and $h([\![E]\!]s) = [\![F]\!]s$~~

$(s,h) \models \mathsf{emp}$ iff ~~$h = []$ (i.e., $\mathsf{dom}(h) = \emptyset$)~~ $h = e$

$(s,h) \models P * Q$ iff $\exists h_0 h_1.\ h_0 * h_1 = h,\ (s,h_0) \models P$ and $(s,h_1) \models Q$

$(s,h) \models \mathsf{true}$ always

$(s,h) \models P \wedge Q$ iff $(s,h) \models P$ and $(s,h) \models Q$

$(s,h) \models \neg P$ iff not $((s,h) \models P)$

$(s,h) \models \forall x.\, P$ iff $\forall v \in \mathsf{Vals}.\ (s[x \mapsto v], h) \models P)$

Replace basic predicates appropriately.

Interpret emp and * by the PCM structure of the model.

Assertions for Any Storage Models

~~Semantics of Assertions~~

~~$(s,h) \models E \geq F$ iff $[\![E]\!]s, [\![F]\!]s \in$ Integers and $[\![E]\!]s \geq [\![F]\!]s$~~

~~$(s,h) \models E \mapsto F$ iff $\mathsf{dom}(h) = \{[\![E]\!]s\}$ and $h([\![E]\!]s) = [\![F]\!]s$~~

$(s,h) \models \mathsf{emp}$ iff ~~$h = []$ (i.e., $\mathsf{dom}(h) = \emptyset$)~~ $h = e$

$(s,h) \models P * Q$ iff $\exists h_0 h_1.\ h_0 * h_1 = h,\ (s,h_0) \models P$ and $(s,h_1) \models Q$

$(s,h) \models \mathsf{true}$ always

$(s,h) \models P \wedge Q$ iff $(s,h) \models P$ and $(s,h) \models Q$

$(s,h) \models \neg P$ iff not $((s,h) \models P)$

$(s,h) \models \forall x.\, P$ iff $\forall v \in \mathsf{Vals}.\ (s[x \mapsto v], h) \models P)$

Replace basic predicates appropriately.
Interpret emp and * by the PCM structure of the model.
Keep the rest.

Assertions for Any Storage Models

Semantics of Assertions

This recipe is standard, and used frequently.
E.g. Bornat&Parkinson's permission.
Vafeiadis&Parkinson's RGSep.

~~$(s,h) \models$~~
~~$(s,h) \models$~~

$(s,h) \models \mathsf{emp}$ iff ~~$h = []$ (i.e., $\mathsf{dom}(h) = \emptyset$)~~ $h = e$

$(s,h) \models P * Q$ iff $\exists h_0 h_1.\ h_0 * h_1 = h,\ (s,h_0) \models P$ and $(s,h_1) \models Q$

$(s,h) \models \mathsf{true}$ always

$(s,h) \models P \wedge Q$ iff $(s,h) \models P$ and $(s,h) \models Q$

$(s,h) \models \neg P$ iff not $((s,h) \models P)$

$(s,h) \models \forall x.\, P$ iff $\forall v \in \mathsf{Vals}.\ (s[x \mapsto v], h) \models P)$

Replace basic predicates appropriately.
Interpret emp and * by the PCM structure of the model.
Keep the rest.

Benefits

- Reminder:
  - Goal: Effective way to express separation.
  - Recipe: First, PCM. Next, separating connectives.
- Gives a very flexible way of expressing separation.
- Well-known proof rules for implication.
- Can re-use another, more important part of sep. logic, which is about program verification.

$$
\begin{array}{rcl}
P * \mathsf{emp} & \Leftrightarrow & P \\
P * Q & \Leftrightarrow & Q * P \\
P * (Q * R) & \Leftrightarrow & P * (Q * R) \\
P * (\bigvee_i Q_i) & \Leftrightarrow & \bigvee_i (P * Q_i) \\
P * (\exists x. Q(x)) & \Leftrightarrow & \exists x. P * Q(x)
\end{array}
$$

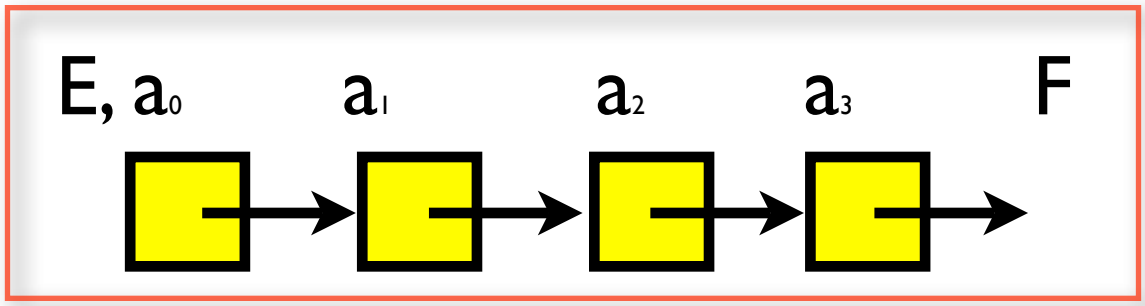
$$
\frac{P \Rightarrow P' \qquad Q \Rightarrow Q'}{P * Q \Rightarrow P' * Q'}
$$

- Reminder:
  - Goal: Effective way to express separation.
  - Recipe: First, PCM. Next, separating connectives.
- Gives a very flexible way of expressing separation.
- Well-known proof rules for implication.
- Can re-use another, more important part of sep. logic, which is about program verification.

List Segment Predicate

- Least fixpoint of the following equivalence:

$$\mathsf{ls}\ \alpha\ (E, F) \iff (\alpha = [] \wedge E = F \wedge \mathsf{emp}) \ \vee \ (\exists x' \alpha'.\ \alpha = E::\alpha' \wedge (E \mapsto x') * \mathsf{ls}\ \alpha'\ (x', F))$$



- The induction occurs positively, so it exists.
- Also, consider a variant that hides alpha:

$$\mathsf{ls}\ (E, F) \iff (E = F \wedge \mathsf{emp}) \ \vee \ (\exists x'.\ E \mapsto x' * \mathsf{ls}\ \alpha'\ (x', F))$$

Back to Original Example

- Express properties at each program point.

1. lists x,y,z.
2. lists x,y: disjoint.
3. lists y,z: disjoint.
4. lists x,z: disjoint.
5. list z sorted.

```
x = create_list();
y = create_list();
z = create_sorted_list();
x = append_list(x,y);
free_list(x);
traverse_list(z);
```

1. lists x,y,z.
2. list x contains y.
3. lists x,z: disjoint.
4. list z sorted.

Back to Original Example

$\mathsf{ls}(x, 0) * \mathsf{ls}(y, 0) * (\exists \alpha.\ \mathsf{sorted}(\alpha) \wedge \mathsf{ls}\,\alpha\,(z, 0))$ n point.

1. lists x,y,z.
2. lists x,y: disjoint.
3. lists y,z: disjoint.
4. lists x,z: disjoint.
5. list z sorted.

```
x = create_list();
y = create_list();
z = create_sorted_list();
x = append_list(x,y);
free_list(x);
traverse_list(z);
```

1. lists x,y,z.
2. list x contains y.
3. lists x,z: disjoint.
4. list z sorted.

Back to Original Example

$\mathsf{ls}(x,0) * \mathsf{ls}(y,0) * (\exists\alpha.\ \mathsf{sorted}(\alpha) \wedge \mathsf{ls}\,\alpha\,(z,0))$ n point.

1. lists x,y,z.
2. lists x,y: disjoint.
3. lists y,z: disjoint.
4. lists x,z: disjoint.
5. list z sorted.

```
x = create_list();
y = create_list();
z = create_sorted_list();
x = append_list(x,y);
free_list(x);
traverse_list(z);
```

1. lists x,y,z.
2. list x contains y.
3. lists x,z: disjoint.
4. list z sorted.

$\mathsf{ls}(x,y) * \mathsf{ls}(y,0) * (\exists\alpha.\ \mathsf{sorted}(\alpha) \wedge \mathsf{ls}\,\alpha\,(z,0))$

Separating Implication

- P -* Q is the weakest assertion A s.t.

$$P * A \Rightarrow Q$$

- Equivalently, P -* Q is the unique assertion s.t.

$$P * A \Rightarrow Q \quad \text{if and only if} \quad A \Rightarrow P \text{ -* } Q.$$

- Q1: Define the satisfaction relation for P -* Q.
- Q2: Define the satisfaction relation for ¬(P -* ¬Q).
- Q3: Give intuitive meanings of both connectives.

Reference

- Reynolds's LICS 2001 paper.
- Biering et al's ESOP 2005 paper.