# Programming Logics and Software Verification

Automating Verification:

Procedures,

Hierarchical Data Structures,

Arithmetic Strengthening

Josh Berdine

# Procedures

# Interprocedural analysis

- For each procedure, create a table that records all the past analysis results.
- Table for create_list():

| Precond. | Postcondition |
|---|---|
| emp | $ret=0 \wedge emp$,  $ret \mapsto 0$,  $ls(ret,0)$ |
| ... | ... |

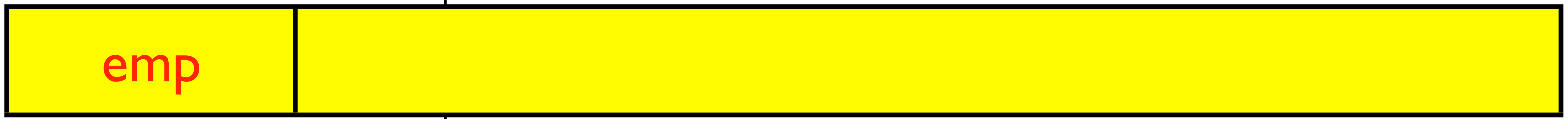
- Use the table whenever possible.

# Creation of two lists

```
let create_list() ={...}

in

      emp

 x=create_list();

 y=create_list();
```

# Creation of two lists

let create_list() ={...}




in

emp

x=create_list();

y=create_list();

# Creation of two lists

let create_list() ={...}

| emp | ret=0 $\wedge$ emp, ret$\mapsto$0, ls(ret,0) |
| --- | --- |

in

emp

x=create_list();

y=create_list();

# Creation of two lists

let create_list() ={...}

| emp | ret=0 ∧ emp, ret↦0, ls(ret,0) |
|---|---|

in

emp

x=create_list();

x=0∧emp    x↦0    ls(x,0)

y=create_list();

# Creation of two lists

let create_list() ={...}

| | |
|---|---|
| emp | $ret=0 \wedge emp$, $ret \mapsto 0$, $ls(ret,0)$ |
| $x \mapsto 0$ | |
| ls x 0 | |

in

emp

x=create_list();

$x=0 \wedge emp$ $x \mapsto 0$ $ls(x,0)$

y=create_list();

# Creation of two lists

let create_list() ={...}

| | |
|---|---|
| emp | ret=0 ∧ emp, ret↦0, ls(ret,0) |
| x↦0 | ret=0 ∧ x↦0, x↦0 * ret↦0, x↦0 * ls(ret,0) |
| ls x 0 | ret=0 ∧ ls(x,0), ls(x,0) * ret↦0, ls(x,0) * ls(ret,0) |

in

emp

x=create_list();

x=0∧emp    x↦0    ls(x,0)

y=create_list();

# Creation of two lists

let create_list() ={...}

| | |
|---|---|
| emp | $ret=0 \land emp$, $ret \mapsto 0$, $ls(ret,0)$ |
| $x \mapsto 0$ | $ret=0 \land x \mapsto 0$, $x \mapsto 0 * ret \mapsto 0$, $x \mapsto 0 * ls(ret,0)$ |
| ls x 0 | $ret=0 \land ls(x,0)$, $ls(x,0) * ret \mapsto 0$, $ls(x,0) * ls(ret,0)$ |

in

emp

x=create_list();

$x=0 \land emp$ $x \mapsto 0$ $ls(x,0)$

y=create_list();

$x=0 \land y=0 \land emp$ $x=0 \land y \mapsto 0$ ........ $x \mapsto 0 * y \mapsto 0$ ........ $ls(x,0) * ls(y,0)$

# Creation of two lists

let create_list() ={...}

| | |
|---|---|
| emp | ret=0 ∧ emp, ret↦0, ls(ret,0) |
| x↦0 | ret=0 ∧ x↦0, x↦0 * ret↦0, x↦0 * ls(ret,0) |
| ls x 0 | ret=0 ∧ ls(x,0), ls(x,0) * ret↦0, ls(x,0) * ls(ret,0) |

3 entries, 9 results

in

emp

x=create_list();

x=0∧emp    x↦0    ls(x,0)

y=create_list();

x=0∧y=0∧emp    x=0∧y↦0    ........    x↦0*y↦0    ........    ls(x,0) * ls(y,0)

# Creation of two lists

let create_list() ={...}

| | |
|---|---|
| emp | ret=0 ∧ emp, ret↦0, ls(ret,0) |
| x↦0 | ret=0 ∧ x↦0, x↦0 * ret↦0, x↦0 * ls(ret,0) |
| ls x 0 | ret=0 ∧ ls(x,0), ls(x,0) * ret↦0, ls(x,0) * ls(ret,0) |

3 entries, 9 results

in

emp

The verifier constructs proofs of two Hoare triples for create_list unnecessarily.

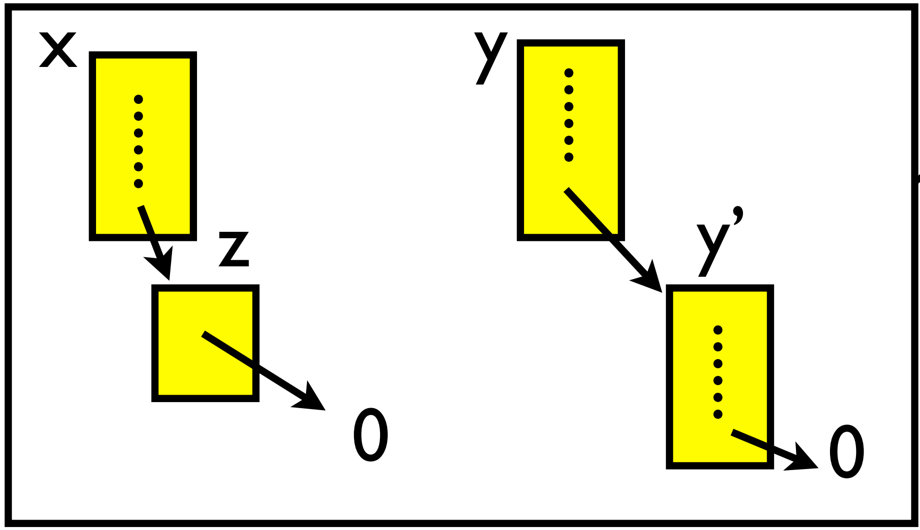
y=create_list();

x=0∧y=0∧emp    x=0∧y↦0 ........ x↦0*y↦0 ........ ls(x,0) * ls(y,0)

# Optimisation by the frame rule

- Pass & change only the part of a symbolic heap, that is reachable from the parameters. [Rinetzky et al., Gotsman et al.]
- E.g.

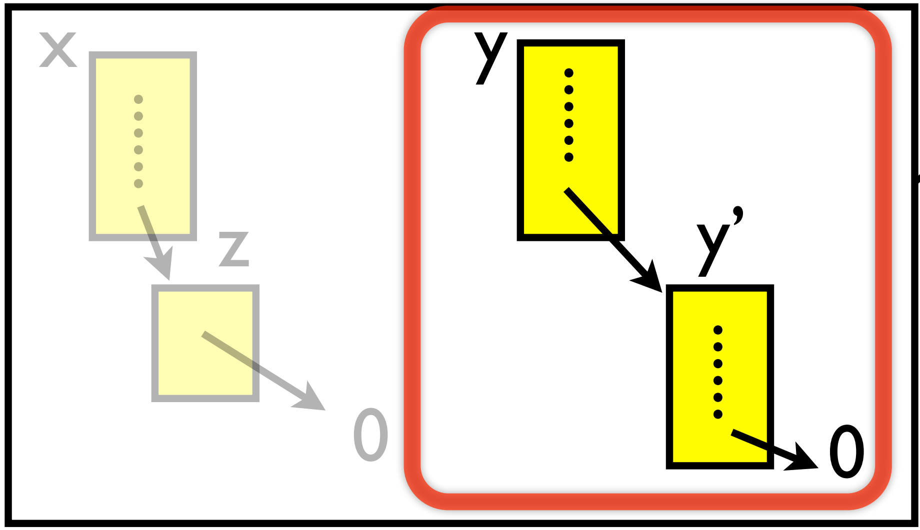
  $\{\exists y'.\ ls(x,z) * z{\mapsto}0 * ls(y,y') * ls(y',0)\}$ dispose_list(y)




# he frame rule

- Pass & change only the part of a symbolic heap, that is reachable from the parameters. [Rinetzky et al., Gotsman et al.]
- E.g.

$$\{\exists y'.\ ls(x,z) * z \mapsto 0 * ls(y,y') * ls(y',0)\}\ \ dispose\_list(y)$$




# he frame rule

- Pass & change only the part of a symbolic heap, that is reachable from the parameters. [Rinetzky et al., Gotsman et al.]
- E.g.

$$\{\exists y'.\ ls(x,z) * z \mapsto 0 * ls(y,y') * ls(y',0)\}\ \ dispose\_list(y)$$

# Creation of 2 Lists

```
let create_list() ={...}

in

      emp
 x=create_list();

 y=create_list();
```

# Creation of 2 Lists

let create_list() ={...}

emp

in

emp

x=create_list();

y=create_list();

# Creation of 2 Lists

let create_list() ={...}

| emp | ret=0 ∧ emp, ret↦0, ls(ret,0) |
|---|---|

in

emp

x=create_list();

y=create_list();

# Creation of 2 Lists

let create_list() ={...}

| emp | ret=0 ∧ emp, ret↦0, ls(ret,0) |
|---|---|

in

emp

x=create_list();

x=0∧emp    x↦0    ls(x,0)

y=create_list();

# Creation of 2 Lists

let create_list() ={...}

| emp | ret=0 ∧ emp, | ret↦0, | ls(ret,0) |
| --- | --- | --- | --- |

in

emp

x=create_list();

x=0∧emp  x↦0  ls(x,0)

y=create_list();

# Creation of 2 Lists

let create_list() ={...}

| emp | ret=0 ∧ emp, ret↦0, ls(ret,0) |
|---|---|

in

emp

x=create_list();

x=0∧emp    x↦0    ls(x,0)

y=create_list();

# Creation of 2 Lists

let create_list() ={...}

| emp | ret=0 ∧ emp, ret↦0, ls(ret,0) |
|---|---|

in

emp

x=create_list();

x=0∧emp    x↦0    ls(x,0)

y=create_list();

x=0∧y=0∧emp    x=0∧y↦0    ........    x↦0*y↦0    ........    ls(x,0) * ls(y,0)

# Creation of 2 Lists

let create_list() ={...}

| emp | ret=0 ∧ emp, ret↦0, ls(ret,0) |
|---|---|

~~3 entries, 9 results~~

1 entries, 3 results

in

emp

x=create_list();

x=0∧emp    x↦0    ls(x,0)

y=create_list();

x=0∧y=0∧emp    x=0∧y↦0    ........    x↦0*y↦0    ........    ls(x,0) * ls(y,0)

# Challenges for interprocedural analyses – efficiency

- Underlying intraprocedural analysis should be efficient
- Interprocedural analyses compute summaries and reuse them:

| | |
|---|---|
| $\{\mathsf{ls}(x, \mathsf{nil}) * \mathsf{ls}(y, \mathsf{nil})\}$ | $\{\mathsf{ls}(u, \mathsf{nil}) * \mathsf{ls}(v, \mathsf{nil})\}$ |
| `append(x, y)` | `append(u, v)` |
| $\{\mathsf{ls}(x, y) * \mathsf{ls}(y, \mathsf{nil})\}$ | $\{\mathsf{ls}(u, v) * \mathsf{ls}(v, \mathsf{nil})\}$ |

- Efficiency $\Rightarrow$ more reusable summaries needed:

| | |
|---|---|
| $\{\mathsf{ls}(x, \mathsf{nil}) * \mathsf{ls}(y, \mathsf{nil})\}$ | $\{\mathsf{ls}(x, \mathsf{nil}) * \mathsf{ls}(y, \mathsf{nil}) * \mathsf{ls}(z, \mathsf{nil})\}$ |
| `append(x, y)` | `append(x, y)` |
| $\{\mathsf{ls}(x, y) * \mathsf{ls}(y, \mathsf{nil})\}$ | $\{?\}$ |

- Procedures should be analyzed on local heaps: $\mathsf{ls}(x, \mathsf{nil})$

# Challenges for interprocedural analyses – precision

Summary: $\{\mathsf{ls}(x, \mathsf{nil}) * \mathsf{ls}(y, \mathsf{nil})\}\ \mathtt{append}(\mathtt{x}, \mathtt{y})\ \{\mathsf{ls}(x, y) * \mathsf{ls}(y, \mathsf{nil})\}$

$\{\mathsf{ls}(x, \mathsf{nil}) * \mathsf{ls}(y, \mathsf{nil}) * \mathsf{ls}(z, \mathsf{nil})\}$

```
append(x, y)
```

$\{\mathsf{ls}(x, y) * \mathsf{ls}(y, \mathsf{nil}) * \mathsf{ls}(z, \mathsf{nil})\}$

```
append(x, z)
```

$\{?\}$

- $y$ – (heap) cutpoint (Rinetzky et al., POPL'05)

# Challenges for interprocedural analyses – precision

Summary: $\{\mathsf{ls}(x, \mathsf{nil}) * \mathsf{ls}(y, \mathsf{nil})\}$ $\mathtt{append}(\mathtt{x}, \mathtt{y})$ $\{\mathsf{ls}(x, y) * \mathsf{ls}(y, \mathsf{nil})\}$

$\{\mathsf{ls}(x, \mathsf{nil}) * \mathsf{ls}(y, \mathsf{nil}) * \mathsf{ls}(z, \mathsf{nil})\}$
`append(x, y)`
$\{\mathsf{ls}(x, y) * \mathsf{ls}(y, \mathsf{nil}) * \mathsf{ls}(z, \mathsf{nil})\}$
`append(x, z)`
$\{\mathsf{ls}(x, y) * \mathsf{ls}(y, z) * \mathsf{ls}(z, \mathsf{nil})\}$

- $y$ – (heap) cutpoint (Rinetzky et al., POPL'05)
- Handling cutpoints precisely and efficiently is difficult

# Handling procedure calls and returns

- Need to compute local heap,
- ...analyze procedure on the local heap,
- ...and recombine the result with the rest of the heap.

# Handling procedure calls and returns

- Need to compute local heap,
- ...analyze procedure on the local heap,
- ...and recombine the result with the rest of the heap.
- Idea: use the FRAME rule

$$\textsc{Frame}\quad \frac{\{P\}\ C\ \{Q\}}{\{P * R\}\ C\ \{Q * R\}}\ C \text{ does not modify variables in } R$$

- Example:
  $\{\mathsf{ls}(x, \mathsf{nil}) * \mathsf{ls}(y, \mathsf{nil})\}\ \mathtt{append}(\mathtt{x}, \mathtt{y})\ \{\mathsf{ls}(x, y) * \mathsf{ls}(y, \mathsf{nil})\}$
  $\Rightarrow$
  $\{\mathsf{ls}(x, \mathsf{nil}) * \mathsf{ls}(y, \mathsf{nil}) * \mathsf{ls}(z, \mathsf{nil})\}$
  `append(x, y)`
  $\{\mathsf{ls}(x, y) * \mathsf{ls}(y, \mathsf{nil}) * \mathsf{ls}(z, \mathsf{nil})\}$

# Local procedure call rule

$$\frac{\textsc{LocalProcCall} \quad S \models P\sigma * R \qquad Q\sigma * R \models T}{\Gamma\,, \{P\}\ f(\vec{x})\ \{Q\} \vdash \{S\}\ f(\vec{x}\sigma)\ \{T\}}$$

# Local procedure call rule

$$\frac{S \models P\sigma * R \qquad Q\sigma * R \models T}{\Gamma\,, \{P\}\ f(\vec{x})\ \{Q\} \vdash \{S\}\ f(\vec{x}\sigma)\ \{T\}}\ \textsc{LocalProcCall}$$

1. Given a heap $S$ at the call-site of $f(\vec{x}\sigma)$

# Local procedure call rule

$$\frac{\textsc{LocalProcCall} \quad S \vDash P\sigma * R \qquad Q\sigma * R \vDash T}{\Gamma\,, \{P\}\ f(\vec{x})\ \{Q\} \vdash \{S\}\ f(\vec{x}\sigma)\ \{T\}}$$

1. Given a heap $S$ at the call-site of $f(\vec{x}\sigma)$
2. Split $S$ into: $P\sigma$ (a local heap) and $R$ (a frame)

# Local procedure call rule

$$\frac{\textsc{LocalProcCall} \quad S \models P\sigma * R \qquad Q\sigma * R \models T}{\Gamma, \{P\}\ f(\vec{x})\ \{Q\} \vdash \{S\}\ f(\vec{x}\sigma)\ \{T\}}$$

1. Given a heap $S$ at the call-site of $f(\vec{x}\sigma)$
2. Split $S$ into: $P\sigma$ (a local heap) and $R$ (a frame)
3. Express the pre-heap $P\sigma$ in terms of the formal parameters: $P$

# Local procedure call rule

$$\dfrac{\substack{\textsc{LocalProcCall}\\ S \vDash P\sigma * R \qquad Q\sigma * R \vDash T}}{\Gamma\,,\ \{P\}\ f(\vec{x})\ \{Q\} \vdash \{S\}\ f(\vec{x}\sigma)\ \{T\}}$$

1. Given a heap $S$ at the call-site of $f(\vec{x}\sigma)$
2. Split $S$ into: $P\sigma$ (a local heap) and $R$ (a frame)
3. Express the pre-heap $P\sigma$ in terms of the formal parameters: $P$
4. Compute the post-heap of the procedure call on $P$: $Q$

# Local procedure call rule

$$\frac{\textsc{LocalProcCall} \quad S \models P\sigma * R \qquad Q\sigma * R \models T}{\Gamma, \{P\}\ f(\vec{x})\ \{Q\} \vdash \{S\}\ f(\vec{x}\sigma)\ \{T\}}$$

1. Given a heap $S$ at the call-site of $f(\vec{x}\sigma)$
2. Split $S$ into: $P\sigma$ (a local heap) and $R$ (a frame)
3. Express the pre-heap $P\sigma$ in terms of the formal parameters: $P$
4. Compute the post-heap of the procedure call on $P$: $Q$
5. Express $Q$ in terms of the actual parameters: $Q\sigma$

# Local procedure call rule

$$\frac{\textsc{LocalProcCall} \quad S \models P\sigma * R \qquad Q\sigma * R \models T}{\Gamma\,,\{P\}\ f(\vec{x})\ \{Q\} \vdash \{S\}\ f(\vec{x}\sigma)\ \{T\}}$$

1. Given a heap $S$ at the call-site of $f(\vec{x}\sigma)$
2. Split $S$ into: $P\sigma$ (a local heap) and $R$ (a frame)
3. Express the pre-heap $P\sigma$ in terms of the formal parameters: $P$
4. Compute the post-heap of the procedure call on $P$: $Q$
5. Express $Q$ in terms of the actual parameters: $Q\sigma$
6. $*$-conjoin $Q\sigma$ with the frame $R$, yielding the post-heap: $T$

# Local procedure call rule

$$\dfrac{\textsc{LocalProcCall} \quad S \models P\sigma * R \qquad Q\sigma * R \models T}{\Gamma\,,\{P\}\ f(\vec{x})\ \{Q\} \vdash \{S\}\ f(\vec{x}\sigma)\ \{T\}}$$

$$\{\mathsf{ls}(u, \mathsf{nil}) * \mathsf{ls}(v, \mathsf{nil}) * \mathsf{ls}(w, \mathsf{nil})\}\ \mathtt{append}(\mathtt{u}, \mathtt{v})\ \{?\}$$

# Local procedure call rule

$$\frac{\textsc{LocalProcCall} \quad S \models P\sigma * R \qquad Q\sigma * R \models T}{\Gamma\,,\{P\}\ f(\vec{x})\ \{Q\} \vdash \{S\}\ f(\vec{x}\sigma)\ \{T\}}$$

$$\{\mathsf{ls}(u,\mathsf{nil}) * \mathsf{ls}(v,\mathsf{nil}) * \mathsf{ls}(w,\mathsf{nil})\}\ \texttt{append}(\texttt{u},\texttt{v})\ \{?\}$$

$$\mathsf{ls}(u,\mathsf{nil}) \text{ and } \mathsf{ls}(v,\mathsf{nil}) * \mathsf{ls}(w,\mathsf{nil})$$

# Local procedure call rule

$$\dfrac{\textsc{LocalProcCall} \quad S \models P\sigma * R \qquad Q\sigma * R \models T}{\Gamma\,, \{P\}\ f(\vec{x})\ \{Q\} \vdash \{S\}\ f(\vec{x}\sigma)\ \{T\}}$$

$$\{\mathsf{ls}(u, \mathsf{nil}) * \mathsf{ls}(v, \mathsf{nil}) * \mathsf{ls}(w, \mathsf{nil})\}\ \texttt{append}(\texttt{u}, \texttt{v})\ \{?\}$$

$$\mathsf{ls}(u, \mathsf{nil}) \text{ and } \mathsf{ls}(v, \mathsf{nil}) * \mathsf{ls}(w, \mathsf{nil})$$

$$\mathsf{ls}(x, \mathsf{nil})$$

# Local procedure call rule

$$\frac{\textsc{LocalProcCall} \quad S \models P\sigma * R \qquad Q\sigma * R \models T}{\Gamma\,, \{P\}\ f(\vec{x})\ \{Q\} \vdash \{S\}\ f(\vec{x}\sigma)\ \{T\}}$$

$$\{\mathsf{ls}(u, \mathsf{nil}) * \mathsf{ls}(v, \mathsf{nil}) * \mathsf{ls}(w, \mathsf{nil})\}\ \texttt{append}(\texttt{u}, \texttt{v})\ \{?\}$$

$$\mathsf{ls}(u, \mathsf{nil}) \text{ and } \mathsf{ls}(v, \mathsf{nil}) * \mathsf{ls}(w, \mathsf{nil})$$

$$\mathsf{ls}(x, \mathsf{nil})$$

$$\mathsf{ls}(x, y)$$

# Local procedure call rule

$$\frac{\begin{array}{c}\textsc{LocalProcCall} \\ S \models P\sigma * R \qquad Q\sigma * R \models T\end{array}}{\Gamma\,,\{P\}\; f(\vec{x})\;\{Q\} \vdash \{S\}\; f(\vec{x}\sigma)\;\{T\}}$$

$$\{\mathsf{ls}(u,\mathsf{nil}) * \mathsf{ls}(v,\mathsf{nil}) * \mathsf{ls}(w,\mathsf{nil})\}\ \texttt{append}(\texttt{u},\texttt{v})\ \{?\}$$

$\mathsf{ls}(u,\mathsf{nil})$ and $\mathsf{ls}(v,\mathsf{nil}) * \mathsf{ls}(w,\mathsf{nil})$

$\mathsf{ls}(x,\mathsf{nil})$

$\mathsf{ls}(x,y)$

$\mathsf{ls}(u,v)$

# Local procedure call rule

$$\frac{\textsc{LocalProcCall} \quad S \models P\sigma * R \qquad Q\sigma * R \models T}{\Gamma\,, \{P\}\ f(\vec{x})\ \{Q\} \vdash \{S\}\ f(\vec{x}\sigma)\ \{T\}}$$

$$\{\mathsf{ls}(u, \mathsf{nil}) * \mathsf{ls}(v, \mathsf{nil}) * \mathsf{ls}(w, \mathsf{nil})\}\ \texttt{append}(\texttt{u}, \texttt{v})\ \{?\}$$

$$\mathsf{ls}(u, \mathsf{nil}) \text{ and } \mathsf{ls}(v, \mathsf{nil}) * \mathsf{ls}(w, \mathsf{nil})$$

$$\mathsf{ls}(x, \mathsf{nil})$$

$$\mathsf{ls}(x, y)$$

$$\mathsf{ls}(u, v)$$

$$\mathsf{ls}(u, v) * \mathsf{ls}(v, \mathsf{nil}) * \mathsf{ls}(w, \mathsf{nil})$$

# Choice of heap splittings

$$\frac{\textsc{LocalProcCall}\quad S \models P\sigma * R \qquad Q\sigma * R \models T}{\Gamma\,, \{P\}\ f(\vec{x})\ \{Q\} \vdash \{S\}\ f(\vec{x}\sigma)\ \{T\}}$$

- $\{\mathsf{ls}(u, \mathsf{nil}) * \mathsf{ls}(v, \mathsf{nil}) * \mathsf{ls}(w, \mathsf{nil})\}\ \mathtt{append(u, v)}\ \{?\}$
- Split $S$ into a local heap and a frame:
  - emp and $\mathsf{ls}(u, \mathsf{nil}) * \mathsf{ls}(v, \mathsf{nil}) * \mathsf{ls}(w, \mathsf{nil})$
  - $\mathsf{ls}(u, \mathsf{nil})$ and $\mathsf{ls}(v, \mathsf{nil}) * \mathsf{ls}(w, \mathsf{nil})$
  - $\mathsf{ls}(u, \mathsf{nil}) * \mathsf{ls}(v, \mathsf{nil})$ and $\mathsf{ls}(w, \mathsf{nil})$
  - $\mathsf{ls}(u, \mathsf{nil}) * \mathsf{ls}(v, \mathsf{nil}) * \mathsf{ls}(w, \mathsf{nil})$ and emp
- All splittings are sound
- Local heap = the portion of the heap reachable from actual parameters:

  $$\mathsf{ls}(u, \mathsf{nil}) * \mathsf{ls}(v, \mathsf{nil}) \text{ and } \mathsf{ls}(w, \mathsf{nil})$$

# Cutpoints

$$\dfrac{\text{LocalProcCall} \quad S \models P\sigma * R \qquad Q\sigma * R \models T}{\Gamma\,, \{P\}\; f(\vec{x})\; \{Q\} \vdash \{S\}\; f(\vec{x}\sigma)\; \{T\}}$$

$$\{\mathsf{ls}(u, c) * \mathsf{ls}(c, \mathsf{nil}) * \mathsf{ls}(v, \mathsf{nil}) * \mathsf{ls}(w, \mathsf{nil})\}\ \mathtt{append}(\mathtt{u}, \mathtt{v})\ \{?\}$$

# Cutpoints

$$\frac{\textsc{LocalProcCall} \quad S \models P\sigma * R \qquad Q\sigma * R \models T}{\Gamma\ , \{P\}\ f(\vec{x})\ \{Q\} \vdash \{S\}\ f(\vec{x}\sigma)\ \{T\}}$$

$$\{\mathsf{ls}(u, c) * \mathsf{ls}(c, \mathsf{nil}) * \mathsf{ls}(v, \mathsf{nil}) * \mathsf{ls}(w, \mathsf{nil})\}\ \mathtt{append}(\mathtt{u}, \mathtt{v})\ \{?\}$$

$$\mathsf{ls}(u, c) * \mathsf{ls}(c, \mathsf{nil}) * \mathsf{ls}(v, \mathsf{nil}) \text{ and } \mathsf{ls}(w, \mathsf{nil})$$

# Cutpoints

$$\frac{\overset{\textsc{LocalProcCall}}{S \models P\sigma * R \qquad Q\sigma * R \models T}}{\Gamma\,, \{P\}\ f(\vec{x})\ \{Q\} \vdash \{S\}\ f(\vec{x}\sigma)\ \{T\}}$$

$$\{\mathsf{ls}(u, c) * \mathsf{ls}(c, \mathsf{nil}) * \mathsf{ls}(v, \mathsf{nil}) * \mathsf{ls}(w, \mathsf{nil})\}\ \texttt{append}(\texttt{u}, \texttt{v})\ \{?\}$$

$$\mathsf{ls}(u, c) * \mathsf{ls}(c, \mathsf{nil}) * \mathsf{ls}(v, \mathsf{nil}) \text{ and } \mathsf{ls}(w, \mathsf{nil})$$

$$\mathsf{ls}(x, ?) * \mathsf{ls}(?, \mathsf{nil}) * \mathsf{ls}(y, \mathsf{nil})$$

# Cutpoints

$$\frac{\text{LocalProcCall} \quad S \models P\sigma * R \qquad Q\sigma * R \models T}{\Gamma\,, \{P\}\ f(\vec{x})\ \{Q\} \vdash \{S\}\ f(\vec{x}\sigma)\ \{T\}}$$

$$\{\mathsf{ls}(u, c) * \mathsf{ls}(c, \mathsf{nil}) * \mathsf{ls}(v, \mathsf{nil}) * \mathsf{ls}(w, \mathsf{nil})\}\ \mathtt{append}(\mathtt{u}, \mathtt{v})\ \{?\}$$

$$\mathsf{ls}(u, c) * \mathsf{ls}(c, \mathsf{nil}) * \mathsf{ls}(v, \mathsf{nil}) \text{ and } \mathsf{ls}(w, \mathsf{nil})$$

$$\mathsf{ls}(x, X) * \mathsf{ls}(X, \mathsf{nil}) * \mathsf{ls}(y, \mathsf{nil})$$

# Cutpoints

$$\dfrac{\textsc{LocalProcCall} \quad S \models P\sigma * R \qquad Q\sigma * R \models T}{\Gamma\,, \{P\}\ f(\vec{x})\ \{Q\} \vdash \{S\}\ f(\vec{x}\sigma)\ \{T\}}$$

$$\{\mathsf{ls}(u, c) * \mathsf{ls}(c, \mathsf{nil}) * \mathsf{ls}(v, \mathsf{nil}) * \mathsf{ls}(w, \mathsf{nil})\}\ \mathtt{append(u, v)}\ \{?\}$$

$$\mathsf{ls}(u, c) * \mathsf{ls}(c, \mathsf{nil}) * \mathsf{ls}(v, \mathsf{nil}) \text{ and } \mathsf{ls}(w, \mathsf{nil})$$

$$\mathsf{ls}(x, X) * \mathsf{ls}(X, \mathsf{nil}) * \mathsf{ls}(y, \mathsf{nil})$$

$$\mathsf{ls}(x, X) * \mathsf{ls}(X, y) * \mathsf{ls}(y, \mathsf{nil})$$

# Cutpoints

$$\frac{S \models P\sigma * R \qquad Q\sigma * R \models T}{\Gamma\,, \{P\}\ f(\vec{x})\ \{Q\} \vdash \{S\}\ f(\vec{x}\sigma)\ \{T\}}\ \textsc{LocalProcCall}$$

$$\{\mathsf{ls}(u,c) * \mathsf{ls}(c,\mathsf{nil}) * \mathsf{ls}(v,\mathsf{nil}) * \mathsf{ls}(w,\mathsf{nil})\}\ \mathtt{append(u,v)}\ \{?\}$$

$$\mathsf{ls}(u,c) * \mathsf{ls}(c,\mathsf{nil}) * \mathsf{ls}(v,\mathsf{nil}) \text{ and } \mathsf{ls}(w,\mathsf{nil})$$

$$\mathsf{ls}(x,X) * \mathsf{ls}(X,\mathsf{nil}) * \mathsf{ls}(y,\mathsf{nil})$$

$$\mathsf{ls}(x,X) * \mathsf{ls}(X,y) * \mathsf{ls}(y,\mathsf{nil})$$

$$\mathsf{ls}(u,c) * \mathsf{ls}(c,v) * \mathsf{ls}(v,\mathsf{nil})$$

# Cutpoints

$$\frac{S \models P\sigma * R \qquad Q\sigma * R \models T}{\Gamma\,,\{P\}\ f(\vec{x})\ \{Q\} \vdash \{S\}\ f(\vec{x}\sigma)\ \{T\}}\ \textsc{LocalProcCall}$$

$$\{\mathsf{ls}(u,c) * \mathsf{ls}(c,\mathsf{nil}) * \mathsf{ls}(v,\mathsf{nil}) * \mathsf{ls}(w,\mathsf{nil})\}\ \mathtt{append}(\mathtt{u},\mathtt{v})\ \{?\}$$

$$\mathsf{ls}(u,c) * \mathsf{ls}(c,\mathsf{nil}) * \mathsf{ls}(v,\mathsf{nil}) \text{ and } \mathsf{ls}(w,\mathsf{nil})$$

$$\mathsf{ls}(x,X) * \mathsf{ls}(X,\mathsf{nil}) * \mathsf{ls}(y,\mathsf{nil})$$

$$\mathsf{ls}(x,X) * \mathsf{ls}(X,y) * \mathsf{ls}(y,\mathsf{nil})$$

$$\mathsf{ls}(u,c) * \mathsf{ls}(c,v) * \mathsf{ls}(v,\mathsf{nil})$$

$$\mathsf{ls}(u,c) * \mathsf{ls}(c,v) * \mathsf{ls}(v,\mathsf{nil}) * \mathsf{ls}(w,\mathsf{nil})$$

# Handling cutpoints

- Problem: number of unquantified variables is no longer bounded
  - abstract domain will be infinite!
- Solution:
  - bound the number of cutpoints
  - quantify the excess

# Handling cutpoints

$$\frac{\textsc{LocalProcCallCut}\quad S \models P' * R \qquad \exists \vec{c}.\, P' \models P\sigma \qquad Q\sigma * R \models T}{\Gamma\,, \{P\}\ f(\vec{x})\ \{Q\} \vdash \{S\}\ f(\vec{x}\sigma)\ \{T\}}$$

1. Given a heap $S$ at the call-site of $f(\vec{x}\sigma)$
2. Split $S$ into: $P\sigma$ (a local heap) and $R$ (a frame)
3. Quantify extra cutpoints and abstract
4. Express the pre-heap $P\sigma$ in terms of the formal parameters: $P$
5. Compute the post-heap of the procedure call on $P$: $Q$
6. Express $Q$ in terms of the actual parameters: $Q\sigma$
7. $*$-conjoin $Q\sigma$ with the frame $R$, yielding the post-heap: $T$

# Handling cutpoints

$$\textsc{LocalProcCallCut}\quad \frac{S \models P' * R \qquad \exists \vec{c}.\, P' \models P\sigma \qquad Q\sigma * R \models T}{\Gamma\,,\{P\}\; f(\vec{x})\;\{Q\} \vdash \{S\}\; f(\vec{x}\sigma)\;\{T\}}$$

$$\{\mathsf{ls}(u,c) * \mathsf{ls}(c,\mathsf{nil}) * \mathsf{ls}(v,\mathsf{nil}) * \mathsf{ls}(w,\mathsf{nil})\}\ \mathtt{append}(\mathtt{u},\mathtt{v})\ \{?\}$$

# Handling cutpoints

$$\frac{\textsc{LocalProcCallCut} \quad S \models P' * R \qquad \exists \vec{c}.\, P' \models P\sigma \qquad Q\sigma * R \models T}{\Gamma\,,\{P\}\; f(\vec{x})\;\{Q\} \vdash \{S\}\; f(\vec{x}\sigma)\;\{T\}}$$

$$\{\mathsf{ls}(u, c) * \mathsf{ls}(c, \mathsf{nil}) * \mathsf{ls}(v, \mathsf{nil}) * \mathsf{ls}(w, \mathsf{nil})\}\ \mathtt{append}(\mathtt{u}, \mathtt{v})\ \{?\}$$

$\mathsf{ls}(u, c) * \mathsf{ls}(c, \mathsf{nil}) * \mathsf{ls}(v, \mathsf{nil})$ and $\mathsf{ls}(w, \mathsf{nil})$

# Handling cutpoints

$$\frac{\textsc{LocalProcCallCut} \quad S \models P' * R \qquad \exists \vec{c}.\, P' \models P\sigma \qquad Q\sigma * R \models T}{\Gamma\,, \{P\}\ f(\vec{x})\ \{Q\} \vdash \{S\}\ f(\vec{x}\sigma)\ \{T\}}$$

$$\{\mathsf{ls}(u, c) * \mathsf{ls}(c, \mathsf{nil}) * \mathsf{ls}(v, \mathsf{nil}) * \mathsf{ls}(w, \mathsf{nil})\}\ \texttt{append}(\texttt{u}, \texttt{v})\ \{?\}$$

$$\mathsf{ls}(u, c) * \mathsf{ls}(c, \mathsf{nil}) * \mathsf{ls}(v, \mathsf{nil}) \text{ and } \mathsf{ls}(w, \mathsf{nil})$$

$$\mathsf{ls}(u, c') * \mathsf{ls}(c', \mathsf{nil}) * \mathsf{ls}(v, \mathsf{nil})$$

# Handling cutpoints

$$\frac{\textsc{LocalProcCallCut} \quad S \models P' * R \qquad \exists \vec{c}.\, P' \models P\sigma \qquad Q\sigma * R \models T}{\Gamma\,,\{P\}\; f(\vec{x})\;\{Q\} \vdash \{S\}\; f(\vec{x}\sigma)\;\{T\}}$$

$$\{\mathsf{ls}(u,c) * \mathsf{ls}(c,\mathsf{nil}) * \mathsf{ls}(v,\mathsf{nil}) * \mathsf{ls}(w,\mathsf{nil})\}\ \mathtt{append(u,v)}\ \{?\}$$

$$\mathsf{ls}(u,c) * \mathsf{ls}(c,\mathsf{nil}) * \mathsf{ls}(v,\mathsf{nil}) \text{ and } \mathsf{ls}(w,\mathsf{nil})$$

$$\mathsf{ls}(u,c') * \mathsf{ls}(c',\mathsf{nil}) * \mathsf{ls}(v,\mathsf{nil})$$

$$\mathsf{ls}(u,\mathsf{nil}) * \mathsf{ls}(v,\mathsf{nil})$$

# Handling cutpoints

$$\frac{\overset{\textsc{LocalProcCallCut}}{S \models P' * R \qquad \exists \vec{c}.\, P' \models P\sigma \qquad Q\sigma * R \models T}}{\Gamma\, ,\, \{P\}\ f(\vec{x})\ \{Q\} \vdash \{S\}\ f(\vec{x}\sigma)\ \{T\}}$$

$$\{\mathsf{ls}(u, c) * \mathsf{ls}(c, \mathsf{nil}) * \mathsf{ls}(v, \mathsf{nil}) * \mathsf{ls}(w, \mathsf{nil})\}\ \mathtt{append(u,v)}\ \{?\}$$

$$\mathsf{ls}(u, c) * \mathsf{ls}(c, \mathsf{nil}) * \mathsf{ls}(v, \mathsf{nil}) \text{ and } \mathsf{ls}(w, \mathsf{nil})$$

$$\mathsf{ls}(u, c') * \mathsf{ls}(c', \mathsf{nil}) * \mathsf{ls}(v, \mathsf{nil})$$

$$\mathsf{ls}(u, \mathsf{nil}) * \mathsf{ls}(v, \mathsf{nil})$$

$$\mathsf{ls}(x, \mathsf{nil}) * \mathsf{ls}(y, \mathsf{nil})$$

# Handling cutpoints

$$\frac{\textsc{LocalProcCallCut} \quad S \models P' * R \qquad \exists \vec{c}.\, P' \models P\sigma \qquad Q\sigma * R \models T}{\Gamma\,,\{P\}\ f(\vec{x})\ \{Q\} \vdash \{S\}\ f(\vec{x}\sigma)\ \{T\}}$$

$$\{\mathsf{ls}(u, c) * \mathsf{ls}(c, \mathsf{nil}) * \mathsf{ls}(v, \mathsf{nil}) * \mathsf{ls}(w, \mathsf{nil})\}\ \mathtt{append(u, v)}\ \{?\}$$

$$\mathsf{ls}(u, c) * \mathsf{ls}(c, \mathsf{nil}) * \mathsf{ls}(v, \mathsf{nil}) \text{ and } \mathsf{ls}(w, \mathsf{nil})$$

$$\mathsf{ls}(u, c') * \mathsf{ls}(c', \mathsf{nil}) * \mathsf{ls}(v, \mathsf{nil})$$

$$\mathsf{ls}(u, \mathsf{nil}) * \mathsf{ls}(v, \mathsf{nil})$$

$$\mathsf{ls}(x, \mathsf{nil}) * \mathsf{ls}(y, \mathsf{nil})$$

$$\mathsf{ls}(x, y) * \mathsf{ls}(y, \mathsf{nil})$$

# Handling cutpoints

$$\frac{\textsc{LocalProcCallCut} \quad S \models P' * R \qquad \exists \vec{c}.\, P' \models P\sigma \qquad Q\sigma * R \models T}{\Gamma\,,\{P\}\; f(\vec{x})\; \{Q\} \vdash \{S\}\; f(\vec{x}\sigma)\; \{T\}}$$

$\{\mathsf{ls}(u,c) * \mathsf{ls}(c,\mathsf{nil}) * \mathsf{ls}(v,\mathsf{nil}) * \mathsf{ls}(w,\mathsf{nil})\}\ \mathtt{append(u,v)}\ \{?\}$

$\mathsf{ls}(u,c) * \mathsf{ls}(c,\mathsf{nil}) * \mathsf{ls}(v,\mathsf{nil})$ and $\mathsf{ls}(w,\mathsf{nil})$

$\mathsf{ls}(u,c') * \mathsf{ls}(c',\mathsf{nil}) * \mathsf{ls}(v,\mathsf{nil})$

$\mathsf{ls}(u,\mathsf{nil}) * \mathsf{ls}(v,\mathsf{nil})$

$\mathsf{ls}(x,\mathsf{nil}) * \mathsf{ls}(y,\mathsf{nil})$

$\mathsf{ls}(x,y) * \mathsf{ls}(y,\mathsf{nil})$

$\mathsf{ls}(u,v) * \mathsf{ls}(v,\mathsf{nil})$

# Handling cutpoints

$$\frac{\overset{\textsc{LocalProcCallCut}}{S \models P' * R \qquad \exists \vec{c}.\, P' \models P\sigma \qquad Q\sigma * R \models T}}{\Gamma\ , \{P\}\ f(\vec{x})\ \{Q\} \vdash \{S\}\ f(\vec{x}\sigma)\ \{T\}}$$

$$\{\mathsf{ls}(u, c) * \mathsf{ls}(c, \mathsf{nil}) * \mathsf{ls}(v, \mathsf{nil}) * \mathsf{ls}(w, \mathsf{nil})\}\ \mathtt{append}(\mathtt{u}, \mathtt{v})\ \{?\}$$

$\mathsf{ls}(u, c) * \mathsf{ls}(c, \mathsf{nil}) * \mathsf{ls}(v, \mathsf{nil})$ and $\mathsf{ls}(w, \mathsf{nil})$

$$\mathsf{ls}(u, c') * \mathsf{ls}(c', \mathsf{nil}) * \mathsf{ls}(v, \mathsf{nil})$$

$$\mathsf{ls}(u, \mathsf{nil}) * \mathsf{ls}(v, \mathsf{nil})$$

$$\mathsf{ls}(x, \mathsf{nil}) * \mathsf{ls}(y, \mathsf{nil})$$

$$\mathsf{ls}(x, y) * \mathsf{ls}(y, \mathsf{nil})$$

$$\mathsf{ls}(u, v) * \mathsf{ls}(v, \mathsf{nil})$$

$$\mathsf{ls}(u, v) * \mathsf{ls}(v, \mathsf{nil}) * \mathsf{ls}(w, \mathsf{nil})$$

# Local variables

- Problem: callee may have local variables
- Solution: existentially quantify local variables at return and abstract

# Local variables

- Problem: callee may have local variables
- Solution: existentially quantify local variables at return and abstract
- Example:

$$\mathsf{ls}(x, t) * t \mapsto y * \mathsf{ls}(y, \mathsf{nil})$$

# Local variables

- Problem: callee may have local variables
- Solution: existentially quantify local variables at return and abstract
- Example:

$$\mathsf{ls}(x, t) * t \mapsto y * \mathsf{ls}(y, \mathsf{nil})$$

$$\mathsf{ls}(x, t') * t' \mapsto y * \mathsf{ls}(y, \mathsf{nil})$$

# Local variables

- Problem: callee may have local variables
- Solution: existentially quantify local variables at return and abstract
- Example:

$$\mathsf{ls}(x, t) * t \mapsto y * \mathsf{ls}(y, \mathsf{nil})$$

$$\mathsf{ls}(x, t') * t' \mapsto y * \mathsf{ls}(y, \mathsf{nil})$$

$$\mathsf{ls}(x, y) * \mathsf{ls}(y, \mathsf{nil})$$

# Overall analysis — interprocedural analysis

- Bound on the number of cutpoints handled at a time
- Procedure calls and returns through LocalProcCallCut
- Reps-Horwitz-Sagiv algorithm for tabulating summaries:
  - start from the first statement of `main()`
  - go forward and top-down
  - if the function isn't analyzed for a heap

    obtain a local heap and analyze it for this heap
  - else

    reuse the existing summary
- Elements of abstract domain: tables of path edges
- Abstract domain is finite $\Rightarrow$ RHS terminates

# Soundness

- Via compilation to separation logic
- Run of the analysis $\Rightarrow$ collection of proofs
- Summary $\Rightarrow$ valid Hoare triple in separation logic
- Proof by induction on the structure of the program
  - uses the rule LocalProcCallCut

# Interprocedural Analysis Summary

- Optimizing using Frame rule beneficial because
  - Most procedures modify only a small subset of the heap
  - Their effect is local
  - Separated heap abstractions mirror this locality
- Structural rules in separation logic often enable sound optimizations
  - Frame rule for interprocedural analysis
  - Concurrency rules and thread-local analysis
- Separation logic proof rules resolve tricky semantic issues
  - Allow optimized symbolic execution

# Hierarchical Data Structures

Microsoft Development Environment [design] - 1394diag.c [Read Only]

File Edit View Debug Tools Window Help

Start Page | 1394diag.c

```
void
t1394Diag_CancelIrp(
    IN PDEVICE_OBJECT   DeviceObject,
    IN PIRP             Irp
    )
{
    KIRQL               Irql;
    PBUS_RESET_IRP      BusResetIrp;
    PDEVICE_EXTENSION   deviceExtension;

    ENTER("t1394Diag_CancelIrp");

    deviceExtension = DeviceObject->DeviceExtension;

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

    BusResetIrp = (PBUS_RESET_IRP) deviceExtension->BusResetIrps.Flink;

    TRACE(TL_TRACE, ("Irp = 0x%x\n", Irp));

    while (BusResetIrp) {

        TRACE(TL_TRACE, ("Cancelling BusResetIrp->Irp = 0x%x\n", BusResetIrp->Irp));

        if (BusResetIrp->Irp == Irp) {

            RemoveEntryList(&BusResetIrp->BusResetIrpList);
            ExFreePool(BusResetIrp);
            break;
        }
        else if (BusResetIrp->BusResetIrpList.Flink == &deviceExtension->BusResetIrps) {
            break;
        }
        else
            BusResetIrp = (PBUS_RESET_IRP)BusResetIrp->BusResetIrpList.Flink;
    }
```

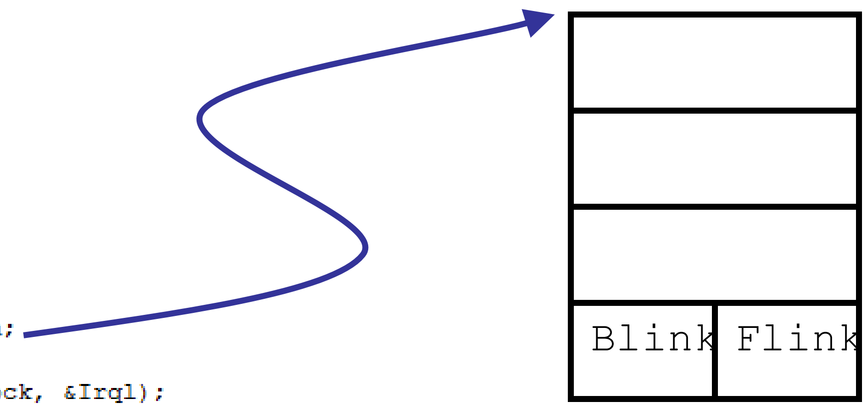
Ready | Ln 90 | Col 1 | Ch 1 | INS

Microsoft Development Environment [design] - 1394diag.c [Read Only]

File Edit View Debug Tools Window Help

Start Page 1394diag.c

```
void
t1394Diag_CancelIrp(
    IN PDEVICE_OBJECT   DeviceObject,
    IN PIRP             Irp
    )
{
    KIRQL               Irql;
    PBUS_RESET_IRP      BusResetIrp;
    PDEVICE_EXTENSION   deviceExtension;

    ENTER("t1394Diag_CancelIrp");

    deviceExtension = DeviceObject->DeviceExtension;

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

    BusResetIrp = (PBUS_RESET_IRP) deviceExtension->BusResetIrps.Flink;

    TRACE(TL_TRACE, ("Irp = 0x%x\n", Irp));

    while (BusResetIrp) {

        TRACE(TL_TRACE, ("Cancelling BusResetIrp->Irp = 0x%x\n", BusResetIrp->Irp));

        if (BusResetIrp->Irp == Irp) {

            RemoveEntryList(&BusResetIrp->BusResetIrpList);
            ExFreePool(BusResetIrp);
            break;
        }
        else if (BusResetIrp->BusResetIrpList.Flink == &deviceExtension->BusResetIrps) {
            break;
        }
        else
            BusResetIrp = (PBUS_RESET_IRP)BusResetIrp->BusResetIrpList.Flink;
    }
```



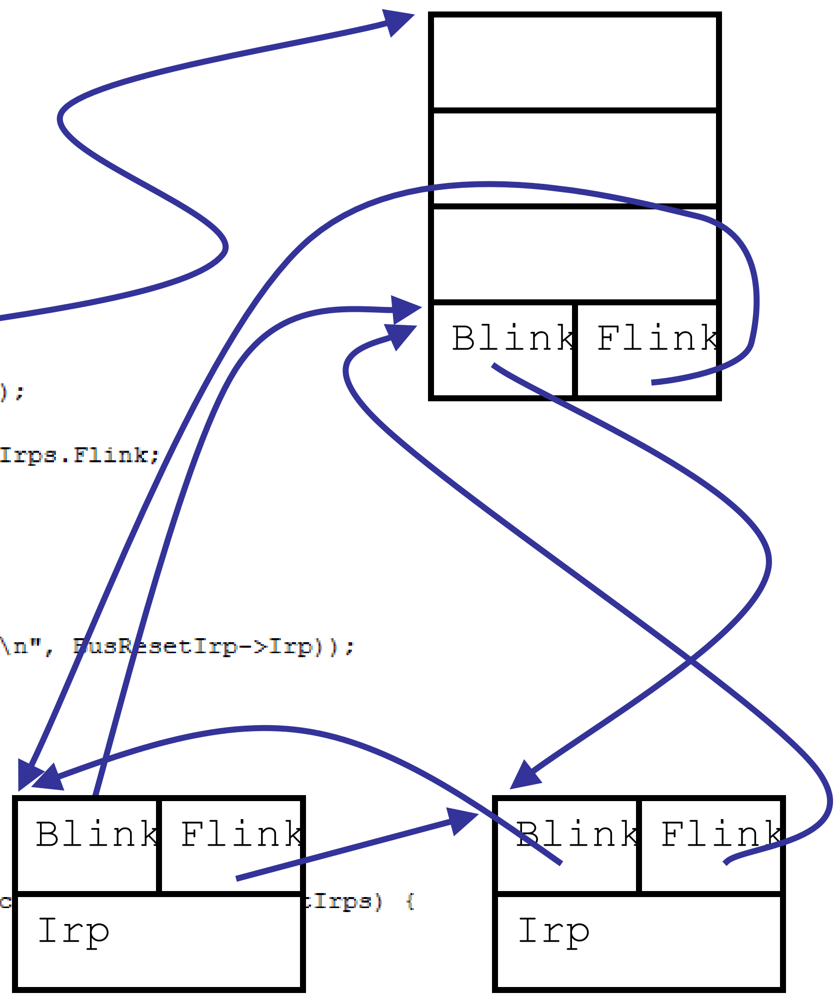

Ready Ln 90 Col 1 Ch 1 INS

Microsoft Development Environment [design] - 1394diag.c [Read Only]

File Edit View Debug Tools Window Help

Start Page 1394diag.c

```c
void
t1394Diag_CancelIrp(
    IN PDEVICE_OBJECT    DeviceObject,
    IN PIRP              Irp
    )
{
    KIRQL                Irql;
    PBUS_RESET_IRP       BusResetIrp;
    PDEVICE_EXTENSION    deviceExtension;

    ENTER("t1394Diag_CancelIrp");

    deviceExtension = DeviceObject->DeviceExtension;

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

    BusResetIrp = (PBUS_RESET_IRP) deviceExtension->BusResetIrps.Flink;

    TRACE(TL_TRACE, ("Irp = 0x%x\n", Irp));

    while (BusResetIrp) {

        TRACE(TL_TRACE, ("Cancelling BusResetIrp->Irp = 0x%x\n", BusResetIrp->Irp));

        if (BusResetIrp->Irp == Irp) {

            RemoveEntryList(&BusResetIrp->BusResetIrpList);
            ExFreePool(BusResetIrp);
            break;
        }
        else if (BusResetIrp->BusResetIrpList.Flink == &devic           Irps) {
            break;
        }
        else
            BusResetIrp = (PBUS_RESET_IRP)BusResetIrp->BusResetIrpList.Flink;
    }
```



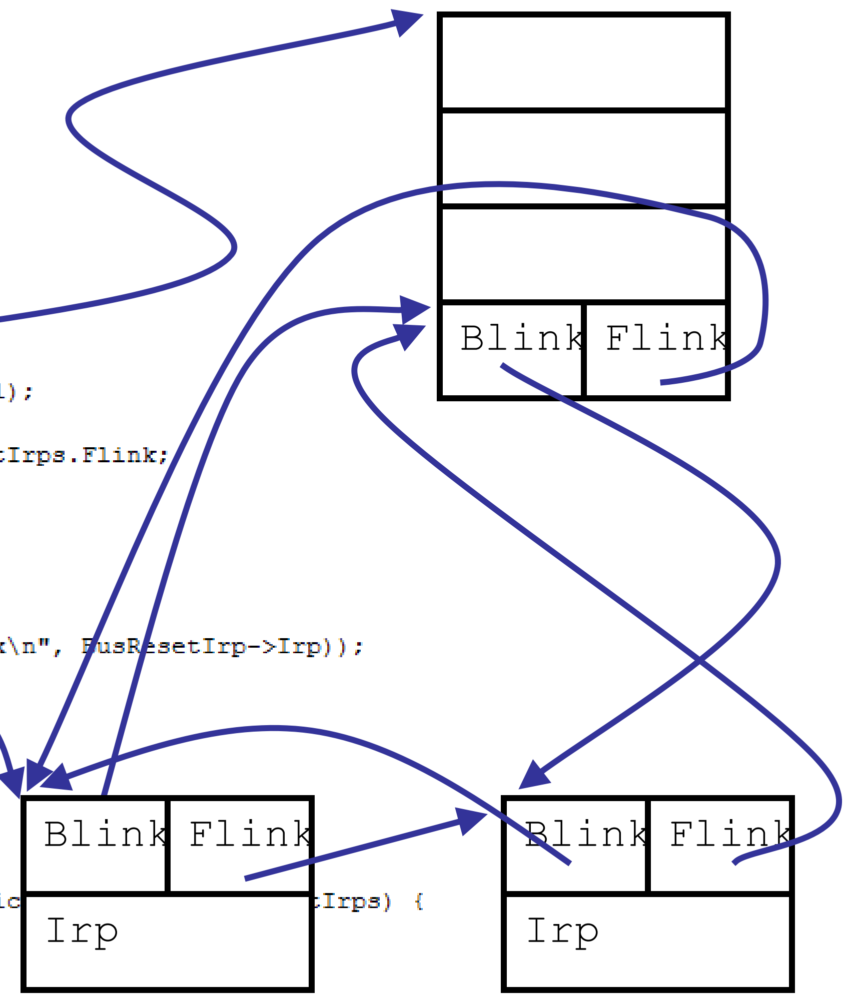

Ready | Ln 90 | Col 1 | Ch 1 | INS

Microsoft Development Environment [design] - 1394diag.c [Read Only]

File Edit View Debug Tools Window Help

Start Page | 1394diag.c

```c
void
t1394Diag_CancelIrp(
    IN PDEVICE_OBJECT   DeviceObject,
    IN PIRP             Irp
    )
{
    KIRQL               Irql;
    PBUS_RESET_IRP      BusResetIrp;
    PDEVICE_EXTENSION   deviceExtension;

    ENTER("t1394Diag_CancelIrp");

    deviceExtension = DeviceObject->DeviceExtension;

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

    BusResetIrp = (PBUS_RESET_IRP) deviceExtension->BusResetIrps.Flink;

    TRACE(TL_TRACE, ("Irp = 0x%x\n", Irp));

    while (BusResetIrp) {

        TRACE(TL_TRACE, ("Cancelling BusResetIrp->Irp = 0x%x\n", BusResetIrp->Irp));

        if (BusResetIrp->Irp == Irp) {

            RemoveEntryList(&BusResetIrp->BusResetIrpList);
            ExFreePool(BusResetIrp);
            break;
        }
        else if (BusResetIrp->BusResetIrpList.Flink == &devic[illegible]Irps) {
            break;
        }
        else
            BusResetIrp = (PBUS_RESET_IRP)BusResetIrp->BusResetIrpList.Flink;
    }
```



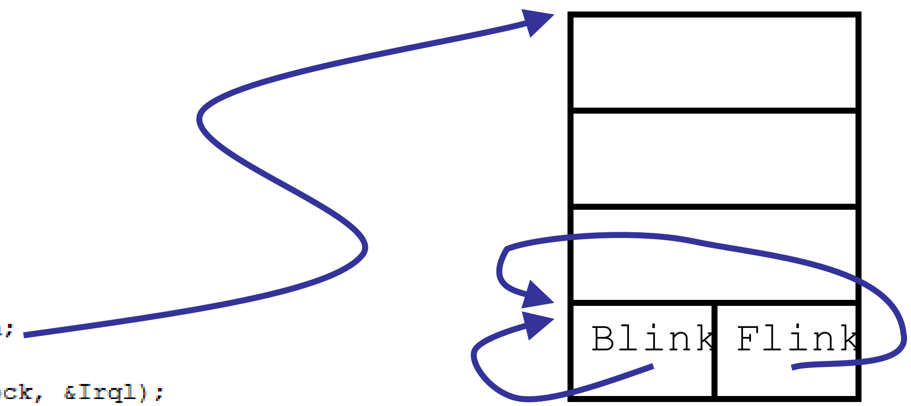

Ready | Ln 90 | Col 1 | Ch 1 | INS

Microsoft Development Environment [design] - 1394diag.c [Read Only]

File Edit View Debug Tools Window Help

Start Page | 1394diag.c

```c
void
t1394Diag_CancelIrp(
    IN PDEVICE_OBJECT   DeviceObject,
    IN PIRP             Irp
    )
{
    KIRQL               Irql;
    PBUS_RESET_IRP      BusResetIrp;
    PDEVICE_EXTENSION   deviceExtension;

    ENTER("t1394Diag_CancelIrp");

    deviceExtension = DeviceObject->DeviceExtension;

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

    BusResetIrp = (PBUS_RESET_IRP) deviceExtension->BusResetIrps.Flink;

    TRACE(TL_TRACE, ("Irp = 0x%x\n", Irp));

    while (BusResetIrp) {

        TRACE(TL_TRACE, ("Cancelling BusResetIrp->Irp = 0x%x\n", BusResetIrp->Irp));

        if (BusResetIrp->Irp == Irp) {

            RemoveEntryList(&BusResetIrp->BusResetIrpList);
            ExFreePool(BusResetIrp);
            break;
        }
        else if (BusResetIrp->BusResetIrpList.Flink == &deviceExtension->BusResetIrps) {
            break;
        }
        else
            BusResetIrp = (PBUS_RESET_IRP)BusResetIrp->BusResetIrpList.Flink;
    }
```



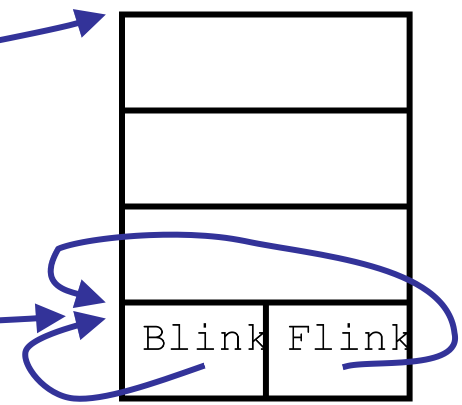

Ready | Ln 90 | Col 1 | Ch 1 | INS

Microsoft Development Environment [design] - 1394diag.c [Read Only]

File Edit View Debug Tools Window Help

Start Page 1394diag.c

```
void
t1394Diag_CancelIrp(
    IN PDEVICE_OBJECT   DeviceObject,
    IN PIRP             Irp
    )
{
    KIRQL               Irql;
    PBUS_RESET_IRP      BusResetIrp;
    PDEVICE_EXTENSION   deviceExtension;

    ENTER("t1394Diag_CancelIrp");

    deviceExtension = DeviceObject->DeviceExtension;

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

    BusResetIrp = (PBUS_RESET_IRP) deviceExtension->BusResetIrps.Flink;

    TRACE(TL_TRACE, ("Irp = 0x%x\n", Irp));

    while (BusResetIrp) {

        TRACE(TL_TRACE, ("Cancelling BusResetIrp->Irp = 0x%x\n", BusResetIrp->Irp));

        if (BusResetIrp->Irp == Irp) {

            RemoveEntryList(&BusResetIrp->BusResetIrpList);
            ExFreePool(BusResetIrp);
            break;
        }
        else if (BusResetIrp->BusResetIrpList.Flink == &deviceExtension->BusResetIrps) {
            break;
        }
        else
            BusResetIrp = (PBUS_RESET_IRP)BusResetIrp->BusResetIrpList.Flink;
    }
```



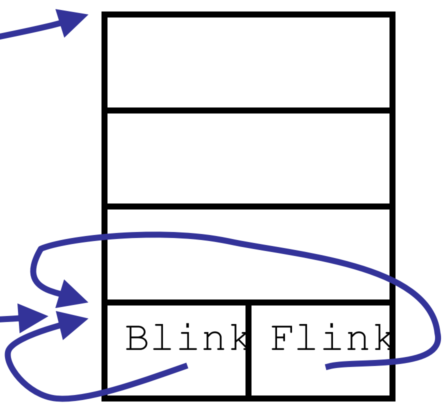

Ready Ln 90 Col 1 Ch 1 INS

Microsoft Development Environment [design] - 1394diag.c [Read Only]

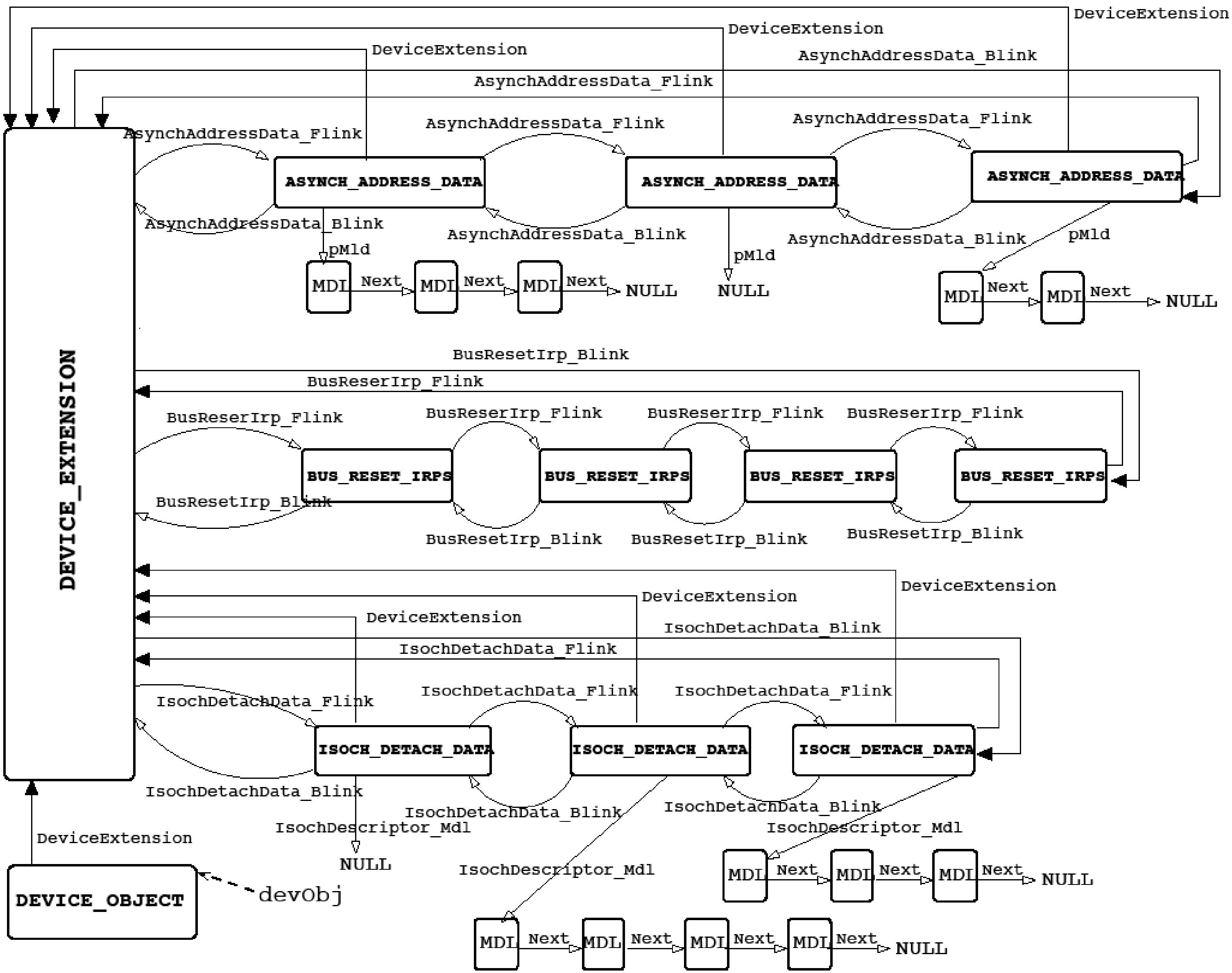
File Edit View Debug Tools Window Help

Start Page | 1394diag.c

```c
void
t1394Diag_CancelIrp(
    IN PDEVICE_OBJECT   DeviceObject,
    IN PIRP             Irp
    )
{
    KIRQL               Irql;
    PBUS_RESET_IRP      BusResetIrp;
    PDEVICE_EXTENSION   deviceExtension;

    ENTER("t1394Diag_CancelIrp");

    deviceExtension = DeviceObject->DeviceExtension;

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

    BusResetIrp = (PBUS_RESET_IRP) deviceExtension->BusResetIrps.Flink;

    TRACE(TL_TRACE, ("Irp = 0x%x\n", Irp));

    while (BusResetIrp) {

        TRACE(TL_TRACE, ("Cancelling BusResetIrp->Irp = 0x%x\n", BusResetIrp->Irp));

        if (BusResetIrp->Irp == Irp) {

            RemoveEntryList(&BusResetIrp->BusResetIrpList);
            ExFreePool(BusResetIrp);
            break;
        }
        else if (BusResetIrp->BusResetIrpList.Flink == &deviceExtension->BusResetIrps) {
            break;
        }
        else
            BusResetIrp = (PBUS_RESET_IRP)BusResetIrp->BusResetIrpList.Flink;
    }
```




Ready | Ln 90 | Col 1 | Ch 1 | INS

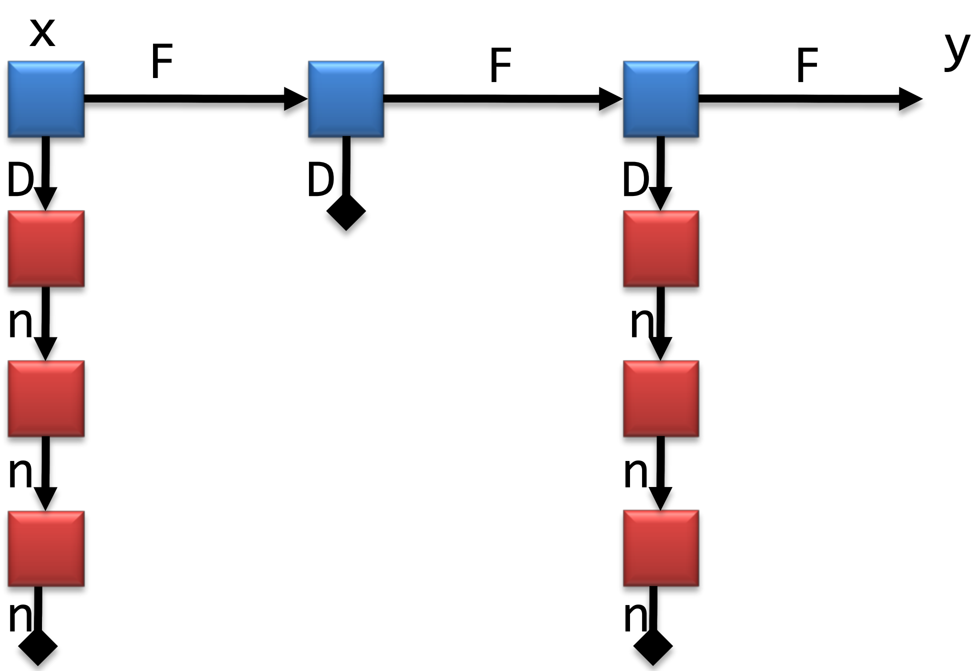

DeviceExtension
DeviceExtension
DeviceExtension
AsynchAddressData_Blink
AsynchAddressData_Flink
AsynchAddressData_Flink
AsynchAddressData_Flink
AsynchAddressData_Flink
ASYNCH_ADDRESS_DATA
ASYNCH_ADDRESS_DATA
ASYNCH_ADDRESS_DATA
AsynchAddressData_Blink
AsynchAddressData_Blink
AsynchAddressData_Blink
pMld
pMld
pMld
MDL
Next
NULL
DEVICE_EXTENSION
BusResetIrp_Blink
BusReserIrp_Flink
BusReserIrp_Flink
BusReserIrp_Flink
BusReserIrp_Flink
BusReserIrp_Flink
BUS_RESET_IRPS
BusResetIrp_Blink
BusResetIrp_Blink
BusResetIrp_Blink
BusResetIrp_Blink
DeviceExtension
DeviceExtension
DeviceExtension
IsochDetachData_Blink
IsochDetachData_Flink
IsochDetachData_Flink
IsochDetachData_Flink
IsochDetachData_Flink
ISOCH_DETACH_DATA
IsochDetachData_Blink
IsochDetachData_Blink
IsochDetachData_Blink
IsochDescriptor_Mdl
IsochDescriptor_Mdl
IsochDescriptor_Mdl
DeviceExtension
DEVICE_OBJECT
devObj

# Second-Order Linked List Segments

- $\mathrm{hls}(\Lambda, E, F) \;\Leftrightarrow\; (E{=}F \wedge \mathrm{emp})$
  $\qquad\qquad\vee\; (\exists y'.\ \Lambda(E,y') * \mathrm{hls}(\Lambda, y', F))$

- For: $\Lambda\mathrm{ls}(x,y) = \exists z'.\ (x \mapsto \{F{:}y,\ D{:}z'\}) * \mathrm{ls}\,(z', \mathrm{null})$

$\mathrm{hls}(\Lambda\mathrm{ls}, x, y)$

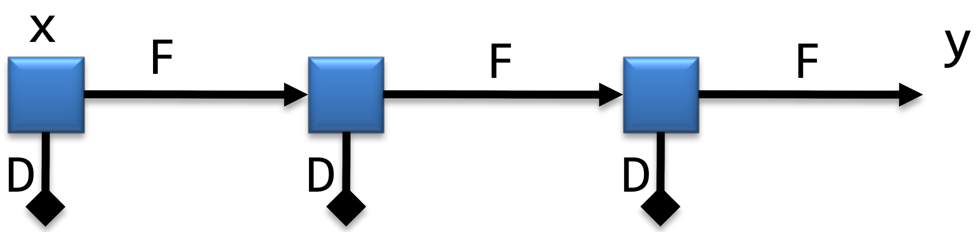


# Second-Order Linked List Segments

- $\text{hls}(\Lambda, E, F) \Leftrightarrow (E{=}F \wedge \text{emp})$
  $\vee (\exists y'.\ \Lambda(E,y') * \text{hls}(\Lambda, y', F))$

- For: $\Lambda\text{emp}(x,y) = \exists z'.\ (x \mapsto \{F{:}y,\ D{:}z'\})$

hls(Λemp, x, y)



# Symbolic Execution with HO lists

```
trim(x,y) {
```

$\{hls(\Lambda ls, x, y)\}$

```
  if (x!=y) {
```

$\{\exists w',z'. (x \mapsto \{F{:}w', D{:}z'\}) * ls(z', null) * hls(\Lambda ls, w', y)\}$

```
    free_list(x->D);
```

$\{\exists w',z'. (x \mapsto \{F{:}w', D{:}z'\}) * emp * hls(\Lambda ls, w', y)\}$

$\{\exists w',z'. (x \mapsto \{F{:}w', D{:}z'\}) * hls(\Lambda ls, w', y)\}$

```
    trim(x->F,y);
```

$\{\exists w',z'. (x \mapsto \{F{:}w', D{:}z'\}) * hls(\Lambda emp, w', y)\}$

```
  }
```

$\{hls(\Lambda emp, x, y)\}$

```
}
```

Supposing: $\{ls(x, null)\}$ `free_list(x);` $\{emp\}$

# Hierarchical Data Structures

- Adaptive shape analysis
  - build in induction principles, rather than particular data structures
  - automatic recognition of many complex variations on linked lists:
    - singly-linked list segments
    - …of non-empty doubly-linked lists
    - …with back-pointers to the head node
    - …of cyclic doubly-linked lists
    - …