FSL: A Program Logic for C11 Memory Fences

Marko Doko Viktor Vafeiadis

Max Planck Institute for Software Systems
(MPI-SWS)

VMCAI 2016

Why C11?

Oddities of weak memory

```
        x = 0;
        y = 0;
x = 1;      || y = 1;
print y;    || print x;
```

Why C11?

Oddities of weak memory

```
        x = 0;
        y = 0;
x = 1;     || y = 1;
print y;   || print x;
```

Both threads can print 0!

Why C11?

Oddities of weak memory

```
      x = 0;
      y = 0;
x = 1;     || y = 1;
print y;   || print x;
```

Both threads can print 0!

W(x,0)

W(y,0)

W(x,1) W(y,1)

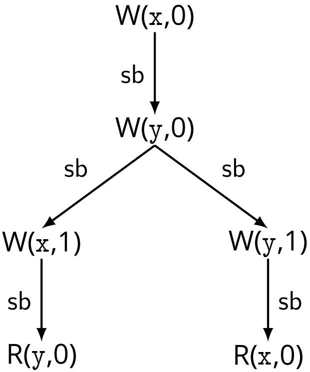
R(y,0) R(x,0)

Why C11?

Oddities of weak memory

```
        x = 0;
        y = 0;
x = 1;      ║ y = 1;
print y;    ║ print x;
```

Both threads can print 0!



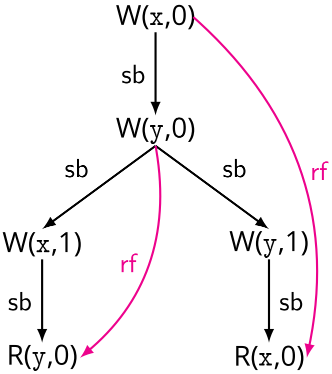
sb – sequenced-before

Why C11?

Oddities of weak memory

```
        x = 0;
        y = 0;
x = 1;     ║ y = 1;
print y;   ║ print x;
```

Both threads can print 0!



sb – sequenced-before
rf – reads-from

C11 model through examples

C11 model through examples

1

```
        int a = 0;
        int x = 0;
a = 42;  || if(x == 1){
x = 1;   ||     print(a);
         || }
```

C11 model through examples

1

```
         int a = 0;
         int x = 0;
a = 42; || if(x == 1){
x = 1;  ||    print(a);
        || }
```

race

C11 model through examples

1

```
     int a = 0;
     int x = 0;
a = 42; || if(x == 1){
x = 1;  ||     print(a);
        || }
```

race

2

```
     int a = 0;
 atomic_int x = 0;
a = 42;     || if(x_rlx == 1){
x_rlx = 1;  ||     print(a);
            || }
```

C11 model through examples

1

```
int a = 0;
int x = 0;
a = 42; || if(x == 1){
x = 1;  ||     print(a);
        || }
```

race

2

```
int a = 0;
atomic_int x = 0;
a = 42;     || if(x_rlx == 1){
x_rlx = 1;  ||     print(a);
            || }
```

race

C11 model through examples

1

```
int a = 0;
int x = 0;
a = 42;  ||  if(x == 1){
x = 1;   ||      print(a);
         ||  }
```

race

2

```
int a = 0;
atomic_int x = 0;
a = 42;      ||  if(x_rlx == 1){
x_rlx = 1;   ||      print(a);
             ||  }
```

race

3

```
int a = 0;
atomic_int x = 0;
a = 42;      ||  if(x_acq == 1){
x_rel = 1;   ||      print(a);
             ||  }
```

C11 model through examples

1

```
      int a = 0;
      int x = 0;
a = 42; || if(x == 1){
x = 1;  ||     print(a);
        || }
```

race

2

```
      int a = 0;
  atomic_int x = 0;
a = 42;      || if(x_rlx == 1){
x_rlx = 1;   ||     print(a);
             || }
```

race

3

```
      int a = 0;
  atomic_int x = 0;
a = 42;      || if(x_acq == 1){
x_rel = 1;   ||     print(a);
             || }
```

rf

C11 model through examples

1

```
        int a = 0;
        int x = 0;
a = 42; || if(x == 1){
x = 1;  ||   print(a);
        || }
```

race

2

```
        int a = 0;
     atomic_int x = 0;
a = 42;    || if(x_rlx == 1){
x_rlx = 1; ||   print(a);
           || }
```

race

3

```
        int a = 0;
     atomic_int x = 0;
a = 42;    || if(x_acq == 1){
x_rel = 1; ||   print(a);
           || }
```

rf

sync

C11 model through examples

1

```
      int a = 0;
      int x = 0;
a = 42;  || if(x == 1){
x = 1;   ||    print(a);
         || }
```

race

2

```
       int a = 0;
  atomic_int x = 0;
a = 42;     || if(x_rlx == 1){
x_rlx = 1;  ||    print(a);
            || }
```

race

3

```
       int a = 0;
  atomic_int x = 0;
a = 42;     || if(x_acq == 1){
x_rel = 1;  ||    print(a);
            || }
```

rf

sync

4

```
       int a = 0;
  atomic_int x = 0;
a = 42;       || if(x_rlx == 1){
fence_rel;    ||    fence_acq;
x_rlx = 1;    ||    print(a);
              || }
```

C11 model through examples

1

```
int a = 0;
int x = 0;
a = 42;  ||  if(x == 1){
x = 1;   ||    print(a);
         ||  }
```

race

2

```
int a = 0;
atomic_int x = 0;
a = 42;    ||  if(x_rlx == 1){
x_rlx = 1; ||    print(a);
           ||  }
```

race

3

```
int a = 0;
atomic_int x = 0;
a = 42;    ||  if(x_acq == 1){
x_rel = 1; ||    print(a);
           ||  }
```

rf

sync

4

```
int a = 0;
atomic_int x = 0;
a = 42;     ||  if(x_rlx == 1){
fence_rel;  ||    fence_acq;
x_rlx = 1;  ||    print(a);
            ||  }
```

rf

C11 model through examples

1

```
     int a = 0;
     int x = 0;
a = 42;  || if(x == 1){
x = 1;   ||    print(a);
         || }
```

race

2

```
       int a = 0;
  atomic_int x = 0;
a = 42;     || if(x_rlx == 1){
x_rlx = 1;  ||    print(a);
            || }
```

race

3

```
       int a = 0;
  atomic_int x = 0;
a = 42;     || if(x_acq == 1){
x_rel = 1;  ||    print(a);
            || }
```

rf

sync

4

```
       int a = 0;
  atomic_int x = 0;
a = 42;        || if(x_rlx == 1){
fence_rel;     ||    fence_acq;
x_rlx = 1;     ||    print(a);
               || }
```

rf

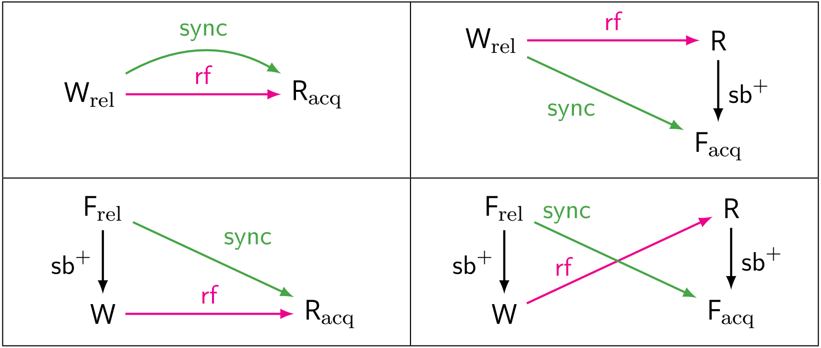
sync

C11 model through examples

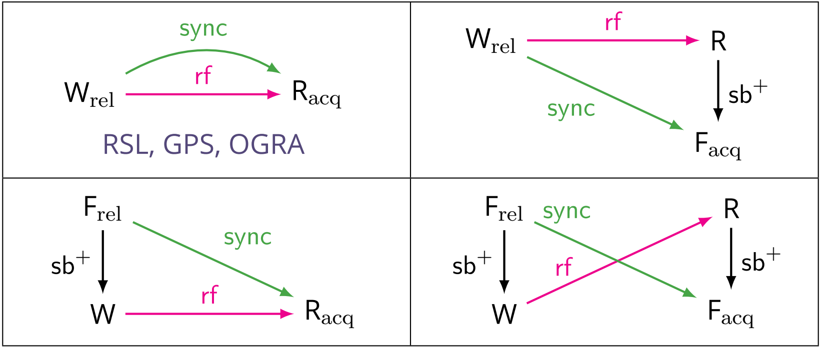
Why use fences?
Release and acquire constructs are expensive!

$$\begin{array}{c}
\texttt{int a = 0;} \\
\texttt{atomic_int x = 0;} \\
\begin{array}{l||l}
\texttt{a = 42;} & \boxed{\texttt{if(}\texttt{x}_{\mathrm{acq}}\texttt{ == 1)}}\texttt{\{} \\
\texttt{x}_{\mathrm{rel}}\texttt{ = 1;} & \quad\texttt{print(a);} \\
 & \texttt{\}}
\end{array}
\end{array}
\qquad
\begin{array}{c}
\texttt{int a = 0;} \\
\texttt{atomic_int x = 0;} \\
\begin{array}{l||l}
\texttt{a = 42;} & \texttt{if(}\texttt{x}_{\mathrm{rlx}}\texttt{ == 1)\{} \\
\texttt{fence}_{\mathrm{rel}}\texttt{;} & \quad\texttt{fence}_{\mathrm{acq}}\texttt{;} \\
\texttt{x}_{\mathrm{rlx}}\texttt{ = 1;} & \quad\texttt{print(a);} \\
 & \texttt{\}}
\end{array}
\end{array}$$

The synchronizes-with relation

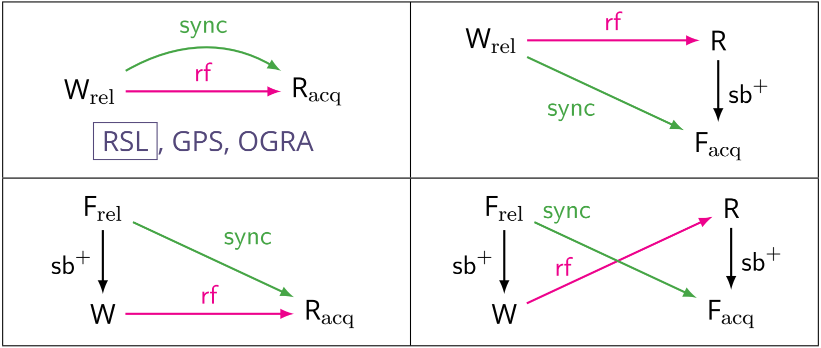


The synchronizes-with relation



RSL	Relaxed Separation Logic (V. Vafeiadis, C. Narayan; OOPSLA '13)
GPS	Ghosts, Protocols, and Separation (A. Turon, V. Vafeiadis, D. Dreyer; OOPSLA '14)
OGRA	Owicki-Gries for Release-Acquire (O. Lahav, V. Vafeiadis; ICALP '15)

The synchronizes-with relation



RSL Relaxed Separation Logic (V. Vafeiadis, C. Narayan; OOPSLA ’13)

GPS Ghosts, Protocols, and Separation (A. Turon, V. Vafeiadis, D. Dreyer; OOPSLA ’14)

OGRA Owicki-Gries for Release-Acquire (O. Lahav, V. Vafeiadis; ICALP ’15)

Relaxed Separation Logic (RSL)

V. Vafeiadis, C. Narayan (OOPSLA 2013)

$$\begin{tikzcd} W_{\mathrm{rel}} \arrow[r, "\mathsf{rf}"'] \arrow[r, bend left, "\mathsf{sync}"] & R_{\mathrm{acq}} \end{tikzcd}$$

$\texttt{int a = 0;}$

$\texttt{atomic\_int x = 0;}$

$\texttt{a = 42;}$	$\texttt{if(}\texttt{x}_{\mathrm{acq}}\texttt{ == 1)\{}$
$\texttt{x}_{\mathrm{rel}}\texttt{ = 1;}$	$\quad\texttt{print(a);}$
	$\texttt{\}}$

Relaxed Separation Logic (RSL)

V. Vafeiadis, C. Narayan (OOPSLA 2013)

$\{true\}$

```
int a = 0;

atomic_int x = 0;
```

`a = 42;`	`if(`x_{acq}` == 1){`
x_{rel} `= 1;`	`print(a);`
	`}`

Relaxed Separation Logic (RSL)

V. Vafeiadis, C. Narayan (OOPSLA 2013)

$\{true\}$

```
int a = 0;
```

$\{\&a \mapsto 0\}$

```
atomic_int x = 0;
```

`a = 42;`	$\texttt{if(x}_{\text{acq}}\texttt{ == 1)\{}$
$\texttt{x}_{\text{rel}}\texttt{ = 1;}$	`print(a);`
	`}`

Relaxed Separation Logic (RSL)

V. Vafeiadis, C. Narayan (OOPSLA 2013)

$\mathcal{Q} \stackrel{\text{def}}{=} \lambda v.\ (v = 0 \vee \&a \mapsto 42)$

$$\{\mathcal{Q}(v)\}$$
`atomic_int x =` v
$$\{\mathsf{Rel}(x, \mathcal{Q}) * \mathsf{Acq}(x, \mathcal{Q})\}$$

$\{true\}$
```
int a = 0;
```
$\{\&a \mapsto 0\}$
```
atomic_int x = 0;
```
$\{\&a \mapsto 0 * \mathsf{Rel}(x, \mathcal{Q}) * \mathsf{Acq}(x, \mathcal{Q})\}$

`a = 42;`	`if(`x_{acq}` == 1){`
x_{rel} `= 1;`	`print(a);`
	`}`

Relaxed Separation Logic (RSL)

V. Vafeiadis, C. Narayan (OOPSLA 2013)

$$\mathcal{Q} \stackrel{\text{def}}{=} \lambda v.\ (v = 0 \lor \&a \mapsto 42)$$

$$\{true\}$$

```
int a = 0;
```

$$\{\&a \mapsto 0\}$$

```
atomic_int x = 0;
```

$$\{\&a \mapsto 0 * \mathsf{Rel}(x, \mathcal{Q}) * \mathsf{Acq}(x, \mathcal{Q})\}$$

$\{\&a \mapsto 0 * \mathsf{Rel}(x, \mathcal{Q})\}$	
`a = 42;`	`if(`$\texttt{x}_{\text{acq}}$` == 1){`
$\texttt{x}_{\text{rel}}$` = 1;`	`print(a);`
	`}`

Relaxed Separation Logic (RSL)

V. Vafeiadis, C. Narayan (OOPSLA 2013)

$$\mathcal{Q} \stackrel{\mathsf{def}}{=} \lambda v.\ (v = 0 \vee \&a \mapsto 42)$$

$$\{true\}$$

```
int a = 0;
```

$$\{\&a \mapsto 0\}$$

```
atomic_int x = 0;
```

$$\{\&a \mapsto 0 * \mathsf{Rel}(x, \mathcal{Q}) * \mathsf{Acq}(x, \mathcal{Q})\}$$

$\{\&a \mapsto 0 * \mathsf{Rel}(x, \mathcal{Q})\}$	
`a = 42;`	`if(`$\mathtt{x}_{\mathrm{acq}}$` == 1){`
$\{\&a \mapsto 42 * \mathsf{Rel}(x, \mathcal{Q})\}$	
$\mathtt{x}_{\mathrm{rel}}$ `= 1;`	`print(a);`
	`}`

Relaxed Separation Logic (RSL)

V. Vafeiadis, C. Narayan (OOPSLA 2013)

$$\mathcal{Q} \stackrel{\text{def}}{=} \lambda v.\ (v = 0 \vee \&a \mapsto 42)$$

$$\{\mathsf{Rel}(x, \mathcal{Q}) * \mathcal{Q}(v)\}$$
$\texttt{x}_{\text{rel}}$ `=` v
$$\{\mathsf{Rel}(x, \mathcal{Q})\}$$

$\{true\}$

```
int a = 0;
```

$\{\&a \mapsto 0\}$

```
atomic_int x = 0;
```

$\{\&a \mapsto 0 * \mathsf{Rel}(x, \mathcal{Q}) * \mathsf{Acq}(x, \mathcal{Q})\}$

$\{\&a \mapsto 0 * \mathsf{Rel}(x, \mathcal{Q})\}$	
`a = 42;`	`if(`$\texttt{x}_{\text{acq}}$ `== 1){`
$\{\&a \mapsto 42 * \mathsf{Rel}(x, \mathcal{Q})\}$	
$\texttt{x}_{\text{rel}}$ `= 1;`	`print(a);`
$\{\mathsf{Rel}(x, \mathcal{Q})\}$	
$\{true\}$	`}`

Relaxed Separation Logic (RSL)

V. Vafeiadis, C. Narayan (OOPSLA 2013)

$$\mathcal{Q} \stackrel{\text{def}}{=} \lambda v.\ (v = 0 \lor \&a \mapsto 42)$$

$\{true\}$

```
int a = 0;
```

$\{\&a \mapsto 0\}$

```
atomic_int x = 0;
```

$\{\&a \mapsto 0 * \mathsf{Rel}(x, \mathcal{Q}) * \mathsf{Acq}(x, \mathcal{Q})\}$

$\{\&a \mapsto 0 * \mathsf{Rel}(x, \mathcal{Q})\}$	$\{\mathsf{Acq}(x, \mathcal{Q})\}$
`a = 42;`	`if(`$\mathtt{x}_{\mathrm{acq}}$ `== 1){`
$\{\&a \mapsto 42 * \mathsf{Rel}(x, \mathcal{Q})\}$	
$\mathtt{x}_{\mathrm{rel}}$ `= 1;`	`    print(a);`
$\{\mathsf{Rel}(x, \mathcal{Q})\}$	
$\{true\}$	`}`

Relaxed Separation Logic (RSL)

V. Vafeiadis, C. Narayan (OOPSLA 2013)

$$\mathcal{Q} \stackrel{\text{def}}{=} \lambda v.\ (v = 0 \vee \&a \mapsto 42)$$

$$\{\mathsf{Acq}(x, \mathcal{Q})\}\ t\ \texttt{=}\ \texttt{x}_{\text{acq}}\ \{\mathcal{Q}(t)\}$$

$\{true\}$

```
int a = 0;
```

$\{\&a \mapsto 0\}$

```
atomic_int x = 0;
```

$\{\&a \mapsto 0 * \mathsf{Rel}(x, \mathcal{Q}) * \mathsf{Acq}(x, \mathcal{Q})\}$

$\{\&a \mapsto 0 * \mathsf{Rel}(x, \mathcal{Q})\}$	$\{\mathsf{Acq}(x, \mathcal{Q})\}$
`a = 42;`	`if(`$\texttt{x}_{\text{acq}}$` == 1){`
$\{\&a \mapsto 42 * \mathsf{Rel}(x, \mathcal{Q})\}$	$\{\&a \mapsto 42\}$
$\texttt{x}_{\text{rel}}$ `= 1;`	`print(a);`
$\{\mathsf{Rel}(x, \mathcal{Q})\}$	
$\{true\}$	`}`

Relaxed Separation Logic (RSL)

V. Vafeiadis, C. Narayan (OOPSLA 2013)

$\mathcal{Q} \stackrel{\text{def}}{=} \lambda v.\ (v = 0 \lor \&a \mapsto 42)$

$\{true\}$

```
int a = 0;
```

$\{\&a \mapsto 0\}$

```
atomic_int x = 0;
```

$\{\&a \mapsto 0 * \mathsf{Rel}(x, \mathcal{Q}) * \mathsf{Acq}(x, \mathcal{Q})\}$

$\{\&a \mapsto 0 * \mathsf{Rel}(x, \mathcal{Q})\}$	$\{\mathsf{Acq}(x, \mathcal{Q})\}$
`a = 42;`	`if(`$\mathtt{x}_{\text{acq}}$` == 1){`
$\{\&a \mapsto 42 * \mathsf{Rel}(x, \mathcal{Q})\}$	$\{\&a \mapsto 42\}$
$\mathtt{x}_{\text{rel}}$` = 1;`	`print(a);`
$\{\mathsf{Rel}(x, \mathcal{Q})\}$	$\{true\}$
$\{true\}$	`}`

Relaxed Separation Logic (RSL)

V. Vafeiadis, C. Narayan (OOPSLA 2013)

$$\mathcal{Q} \stackrel{\text{def}}{=} \lambda v.\ (v = 0 \lor \&a \mapsto 42)$$

$\{true\}$

```
int a = 0;
```

$\{\&a \mapsto 0\}$

```
atomic_int x = 0;
```

$\{\&a \mapsto 0 * \mathsf{Rel}(x, \mathcal{Q}) * \mathsf{Acq}(x, \mathcal{Q})\}$

$\{\&a \mapsto 0 * \mathsf{Rel}(x, \mathcal{Q})\}$	$\{\mathsf{Acq}(x, \mathcal{Q})\}$
`a = 42;`	`if(`$\mathtt{x}_{\mathrm{acq}}$` == 1){`
$\{\&a \mapsto 42 * \mathsf{Rel}(x, \mathcal{Q})\}$	$\{\&a \mapsto 42\}$
$\mathtt{x}_{\mathrm{rel}}$ `= 1;`	`print(a);`
$\{\mathsf{Rel}(x, \mathcal{Q})\}$	$\{true\}$
$\{true\}$	`}`

$\{true\}$

Relaxed Separation Logic (RSL)

V. Vafeiadis, C. Narayan (OOPSLA 2013)

$$\mathcal{Q} \stackrel{\text{def}}{=} \lambda v.\ (v = 0 \vee \&a \mapsto 42)$$

$\{true\}$

```
int a = 0;
```

- no data races
- memory safety
- no reads of uninitialized locations

$\{\&a \mapsto$ …

$\{\&a \mapsto 0 * \mathsf{R}$ …

```
a = 42;
```

$\{\&a \mapsto 42 * \mathsf{Rel}(x, \mathcal{Q})\}$

$\mathtt{x}_{\mathrm{rel}}$ `= 1;`

$\{\mathsf{Rel}(x, \mathcal{Q})\}$

$\{true\}$

$\{\&a \mapsto 42\}$

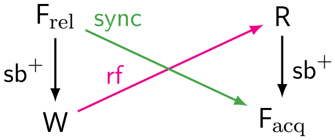
```
print(a);
```

$\{true\}$

```
}
```

$\{true\}$

Fenced Separation Logic (FSL)

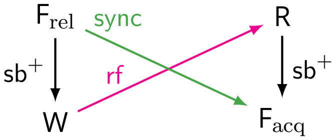



```
int a = 0;
atomic_int x = 0;
```

`a = 42;`	`if(`x_{rlx} `== 1){`
$\texttt{fence}_{rel}$`;`	$\quad\texttt{fence}_{acq}$`;`
x_{rlx} `= 1;`	`    print(a);`
	`}`

Fenced Separation Logic (FSL)

$\mathcal{Q} \stackrel{\text{def}}{=} \lambda v.\ (v = 0 \lor \&a \mapsto 42)$



$\{true\}$

`int a = 0;`

$\{\&a \mapsto 0\}$

`atomic_int x = 0;`

$\{\&a \mapsto 0 * \mathsf{Rel}(x, \mathcal{Q}) * \mathsf{Acq}(x, \mathcal{Q})\}$

$\{\&a \mapsto 0 * \mathsf{Rel}(x, \mathcal{Q})\}$	$\{\mathsf{Acq}(x, \mathcal{Q})\}$
`a = 42;`	`if(`$\mathtt{x}_{\mathrm{rlx}}$` == 1){`
$\{\&a \mapsto 42 * \mathsf{Rel}(x, \mathcal{Q})\}$	???
$\mathtt{fence}_{\mathrm{rel}}$`;`	$\mathtt{fence}_{\mathrm{acq}}$`;`
???	$\{\&a \mapsto 42\}$
$\mathtt{x}_{\mathrm{rlx}}$ `= 1;`	`print(a);`
$\{\mathsf{Rel}(x, \mathcal{Q})\}$	$\{true\}$
$\{true\}$	`}`

$\{true\}$

Fenced Separation Logic (FSL)

$\mathcal{Q} \stackrel{\text{def}}{=} \lambda v.\ (v = 0 \vee \&a \mapsto 42)$

$$\{P\}\ \texttt{fence}_{\text{rel}}\ \{\triangle P\}$$

$$\{\mathsf{Rel}(x,\mathcal{Q}) * \triangle \mathcal{Q}(v)\}$$
$$\texttt{x}_{\text{rlx}}\ \texttt{=}\ v$$
$$\{\mathsf{Rel}(x,\mathcal{Q})\}$$

$\{true\}$

`int a = 0;`

$\{\&a \mapsto 0\}$

`atomic_int x = 0;`

$\{\&a \mapsto 0 * \mathsf{Rel}(x,\mathcal{Q}) * \mathsf{Acq}(x,\mathcal{Q})\}$

$\{\&a \mapsto 0 * \mathsf{Rel}(x,\mathcal{Q})\}$	$\{\mathsf{Acq}(x,\mathcal{Q})\}$
`a = 42;`	$\texttt{if(x}_{\text{rlx}}\texttt{ == 1)\{}$
$\{\&a \mapsto 42 * \mathsf{Rel}(x,\mathcal{Q})\}$	???
$\texttt{fence}_{\text{rel}}\texttt{;}$	$\texttt{fence}_{\text{acq}}\texttt{;}$
$\{\triangle(\&a \mapsto 42) * \mathsf{Rel}(x,\mathcal{Q})\}$	$\{\&a \mapsto 42\}$
$\texttt{x}_{\text{rlx}}\texttt{ = 1;}$	`print(a);`
$\{\mathsf{Rel}(x,\mathcal{Q})\}$	$\{true\}$
$\{true\}$	`}`

$\{true\}$

Fenced Separation Logic (FSL)

$\mathcal{Q} \stackrel{\mathsf{def}}{=} \lambda v.\ (v = 0 \lor \&a \mapsto 42)$

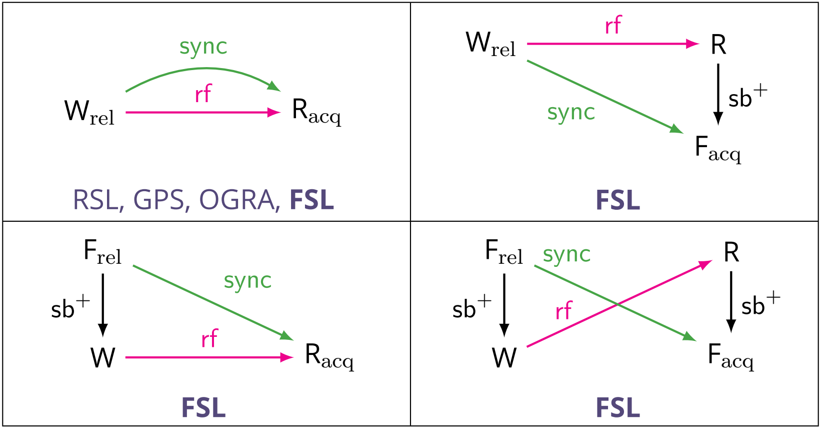
$\{\mathsf{Acq}(x, \mathcal{Q})\}$
$\texttt{t = x}_{\mathrm{rlx}}$
$\{\triangledown \mathcal{Q}(t)\}$

$\{\triangledown P\}\ \texttt{fence}_{\mathrm{acq}}\ \{P\}$

$\{\mathit{true}\}$
`int a = 0;`
$\{\&a \mapsto 0\}$
`atomic_int x = 0;`
$\{\&a \mapsto 0 * \mathsf{Rel}(x, \mathcal{Q}) * \mathsf{Acq}(x, \mathcal{Q})\}$

$\{\&a \mapsto 0 * \mathsf{Rel}(x, \mathcal{Q})\}$	$\{\mathsf{Acq}(x, \mathcal{Q})\}$
`a = 42;`	$\texttt{if(x}_{\mathrm{rlx}}\texttt{ == 1)\{}$
$\{\&a \mapsto 42 * \mathsf{Rel}(x, \mathcal{Q})\}$	$\quad\{\triangledown(\&a \mapsto 42)\}$
$\texttt{fence}_{\mathrm{rel}}\texttt{;}$	$\quad\texttt{fence}_{\mathrm{acq}}\texttt{;}$
$\{\triangle(\&a \mapsto 42) * \mathsf{Rel}(x, \mathcal{Q})\}$	$\quad\{\&a \mapsto 42\}$
$\texttt{x}_{\mathrm{rlx}}\texttt{ = 1;}$	$\quad$`print(a);`
$\{\mathsf{Rel}(x, \mathcal{Q})\}$	$\quad\{\mathit{true}\}$
$\{\mathit{true}\}$	`}`

$\{\mathit{true}\}$

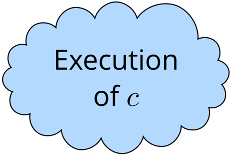

W_{rel}
sync
rf
R_{acq}
RSL, GPS, OGRA, **FSL**
W_{rel}
rf
R
sync
sb^+
F_{acq}
FSL
F_{rel}
sb^+
sync
W
rf
R_{acq}
FSL
F_{rel}
sync
R
sb^+
rf
sb^+
W
F_{acq}
FSL

The semantics of triples

Without a notion of state, what is the meaning of $\{P\}\, c\, \{Q\}$?

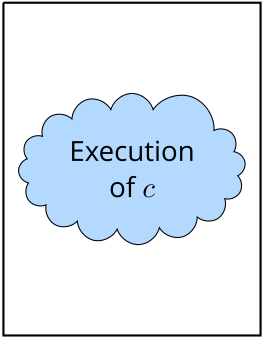
The semantics of triples

Without a notion of state, what is the meaning of $\{P\}\,c\,\{Q\}$?



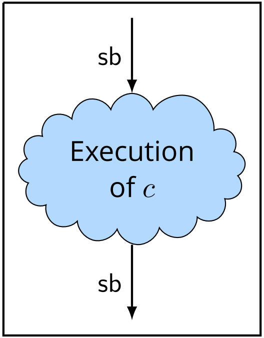
The semantics of triples

Without a notion of state, what is the meaning of $\{P\}\,c\,\{Q\}$?



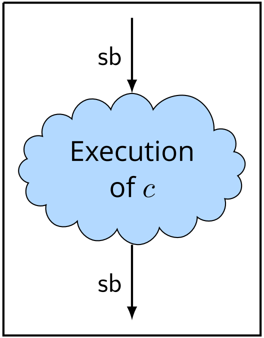
The semantics of triples

Without a notion of state, what is the meaning of $\{P\}\, c\, \{Q\}$?



The semantics of triples

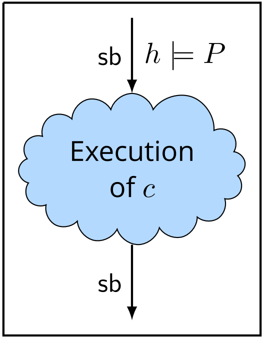
Without a notion of state, what is the meaning of $\{P\}\, c\, \{Q\}$?



Annotate heaps on sb and rf edges in the execution graph.

The semantics of triples

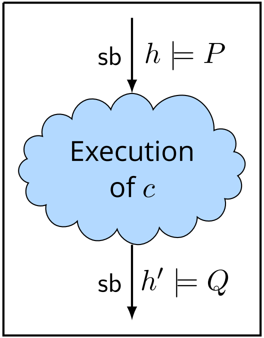
Without a notion of state, what is the meaning of $\{P\}\, c\, \{Q\}$?



Annotate heaps on sb and rf edges in the execution graph.

The semantics of triples

Without a notion of state, what is the meaning of $\{P\}\, c\, \{Q\}$?



Annotate heaps on sb and rf edges in the execution graph.

Local validity

a very simplified example

$$\{\triangle(\&a \mapsto 42) * \mathsf{Rel}(x, \mathcal{Q})\}\ \texttt{x}_{\mathrm{rlx}}\ \texttt{= 1}\ \{\mathsf{Rel}(x, \mathcal{Q})\}$$

Local validity

a very simplified example

$$\{\triangle(\&a \mapsto 42) * \mathsf{Rel}(x, \mathcal{Q})\}\ \mathtt{x}_{\mathrm{rlx}}\ \mathtt{=}\ \mathtt{1}\ \{\mathsf{Rel}(x, \mathcal{Q})\}$$

$$\mathsf{W}_{\mathrm{rlx}}(\mathtt{x}, 1)$$

Local validity

a very simplified example

$$\{\triangle(\&a \mapsto 42) * \mathsf{Rel}(x, \mathcal{Q})\}\ \mathtt{x}_{\mathrm{rlx}}\ \mathtt{=}\ \mathtt{1}\ \{\mathsf{Rel}(x, \mathcal{Q})\}$$

$$\downarrow$$

$$\mathsf{W}_{\mathrm{rlx}}(\mathtt{x}, 1)$$

Local validity

a very simplified example

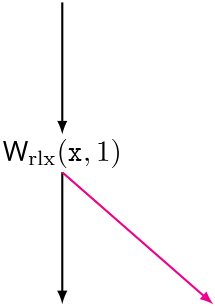
$$\{\triangle(\&a \mapsto 42) * \mathsf{Rel}(x, \mathcal{Q})\}\ \mathtt{x}_{\mathrm{rlx}}\ \mathtt{=}\ \mathtt{1}\ \{\mathsf{Rel}(x, \mathcal{Q})\}$$

$$\begin{tikzcd}
{} \arrow[d] \\
\mathsf{W}_{\mathrm{rlx}}(\mathtt{x}, 1) \arrow[d] \\
{}
\end{tikzcd}$$

Local validity

a very simplified example

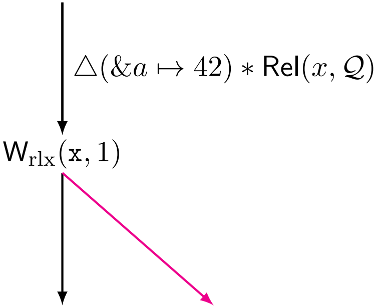
$$\{\triangle(\&a \mapsto 42) * \mathsf{Rel}(x, \mathcal{Q})\}\ \texttt{x}_{\mathrm{rlx}}\ \texttt{=}\ \texttt{1}\ \{\mathsf{Rel}(x, \mathcal{Q})\}$$



Local validity

a very simplified example

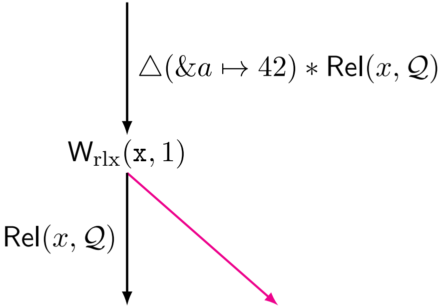
$$\{\triangle(\&a \mapsto 42) * \mathsf{Rel}(x, \mathcal{Q})\}\ \texttt{x}_{\mathrm{rlx}}\ \texttt{= 1}\ \{\mathsf{Rel}(x, \mathcal{Q})\}$$



Local validity

a very simplified example

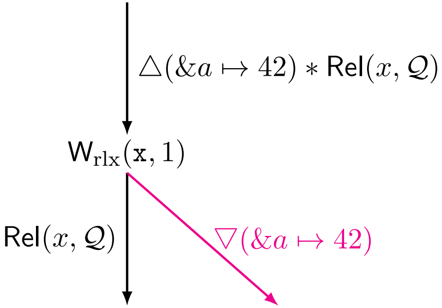
$$\{\triangle(\&a \mapsto 42) * \mathsf{Rel}(x, \mathcal{Q})\}\ \mathtt{x}_{\mathrm{rlx}}\ \mathtt{=}\ \mathtt{1}\ \{\mathsf{Rel}(x, \mathcal{Q})\}$$



Local validity

a very simplified example

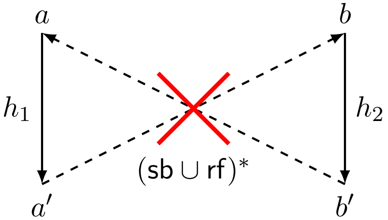
$$\{\triangle(\&a \mapsto 42) * \mathsf{Rel}(x, \mathcal{Q})\}\ \mathtt{x}_{\mathrm{rlx}}\ \mathtt{=}\ \mathtt{1}\ \{\mathsf{Rel}(x, \mathcal{Q})\}$$



Independent heap compatibility

Definition (Independent edges)

A set of edges $\mathcal{T}$ in an execution graph is *pairwise independent* if for all $(a, a'), (b, b') \in \mathcal{T}$, we have $\neg(\mathsf{sb} \cup \mathsf{rf})^*(a', b)$.



$\Longrightarrow$ $h_1 \oplus h_2$ is defined

Lemma (Independent heap compatibility)

For every validly annotated execution, and pairwise independent set of edges $\mathcal{T}$, heaps annotated on edges in $\mathcal{T}$ are combinable.

Data race freedom

Theorem

In a validly annotated execution, any two non-atomic accesses to the same location are ordered by $(\mathsf{sb} \cup \mathsf{sync})^+$ *(i.e. they are not racing).*

Data race freedom

Theorem

In a validly annotated execution, any two non-atomic accesses to the same location are ordered by $(\mathsf{sb} \cup \mathsf{sync})^+$ *(i.e. they are not racing).*

Data race freedom

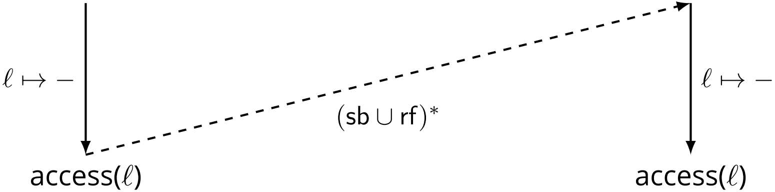
Theorem

In a validly annotated execution, any two non-atomic accesses to the same location are ordered by $(\mathsf{sb} \cup \mathsf{sync})^+$ *(i.e. they are not racing).*

Data race freedom

Theorem

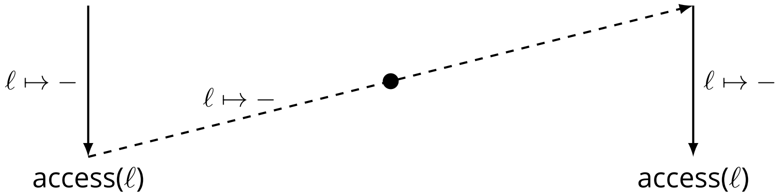
In a validly annotated execution, any two non-atomic accesses to the same location are ordered by $(\mathsf{sb} \cup \mathsf{sync})^+$ *(i.e. they are not racing).*



Data race freedom

Theorem

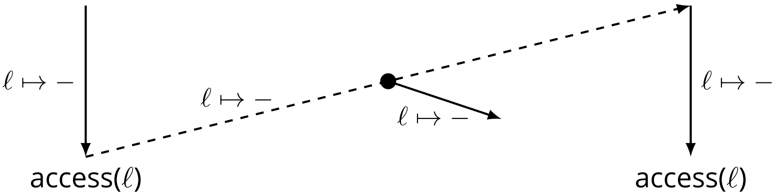
In a validly annotated execution, any two non-atomic accesses to the same location are ordered by $(\mathsf{sb} \cup \mathsf{sync})^+$ *(i.e. they are not racing).*



Data race freedom

Theorem

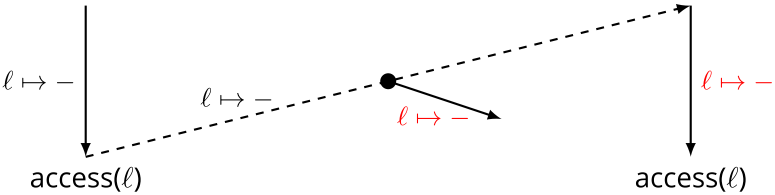
In a validly annotated execution, any two non-atomic accesses to the same location are ordered by $(\mathsf{sb} \cup \mathsf{sync})^+$ *(i.e. they are not racing).*



Data race freedom

Theorem

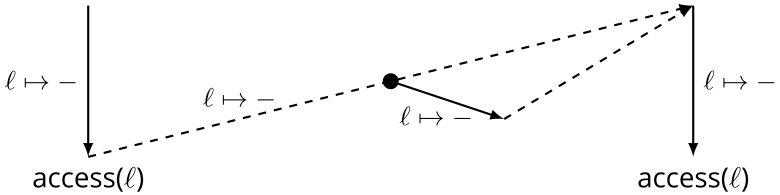
In a validly annotated execution, any two non-atomic accesses to the same location are ordered by $(\mathsf{sb} \cup \mathsf{sync})^+$ *(i.e. they are not racing).*



Data race freedom

Theorem

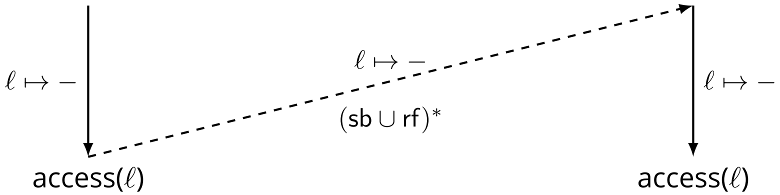
In a validly annotated execution, any two non-atomic accesses to the same location are ordered by $(\mathsf{sb} \cup \mathsf{sync})^+$ *(i.e. they are not racing).*



Data race freedom

Theorem

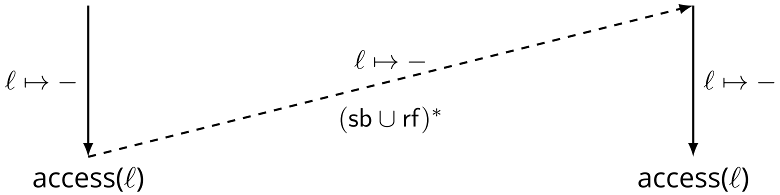
In a validly annotated execution, any two non-atomic accesses to the same location are ordered by $(\mathsf{sb} \cup \mathsf{sync})^+$ *(i.e. they are not racing).*

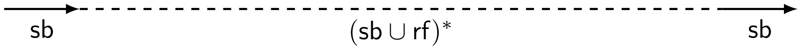


Data race freedom

Theorem

In a validly annotated execution, any two non-atomic accesses to the same location are ordered by $(\mathsf{sb} \cup \mathsf{sync})^+$ *(i.e. they are not racing).*

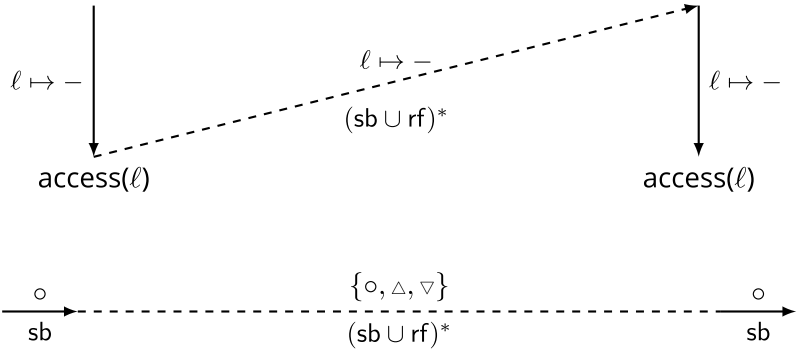




Data race freedom

Theorem

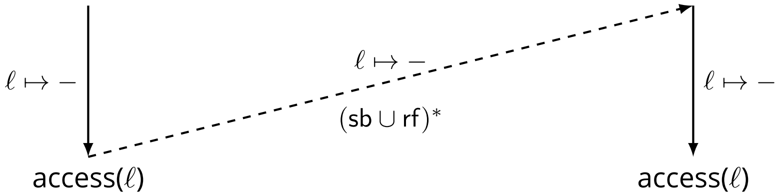
In a validly annotated execution, any two non-atomic accesses to the same location are ordered by $(\mathsf{sb} \cup \mathsf{sync})^+$ *(i.e. they are not racing).*

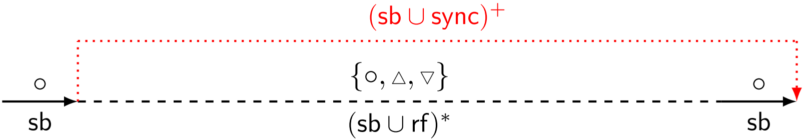


Data race freedom

Theorem

In a validly annotated execution, any two non-atomic accesses to the same location are ordered by $(\mathsf{sb} \cup \mathsf{sync})^+$ *(i.e. they are not racing).*

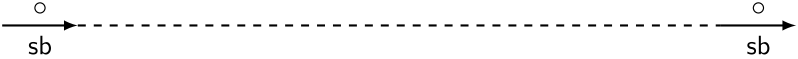




Data race freedom

Theorem

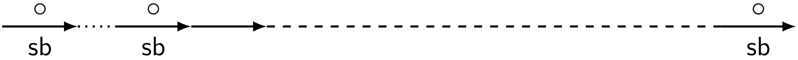
In a validly annotated execution, any two non-atomic accesses to the same location are ordered by $(\mathsf{sb} \cup \mathsf{sync})^+$ *(i.e. they are not racing).*



Data race freedom

Theorem

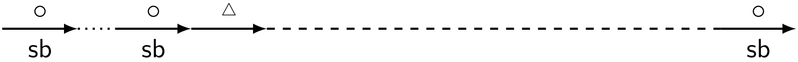
In a validly annotated execution, any two non-atomic accesses to the same location are ordered by $(\mathsf{sb} \cup \mathsf{sync})^+$ *(i.e. they are not racing).*



Data race freedom

Theorem

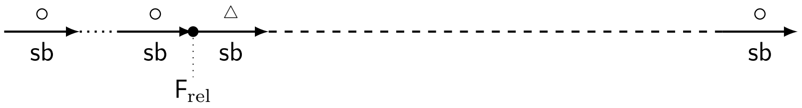
In a validly annotated execution, any two non-atomic accesses to the same location are ordered by $(\mathsf{sb} \cup \mathsf{sync})^+$ *(i.e. they are not racing).*



Data race freedom

Theorem

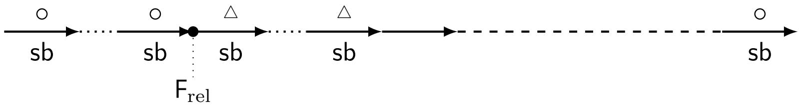
In a validly annotated execution, any two non-atomic accesses to the same location are ordered by $(\mathsf{sb} \cup \mathsf{sync})^+$ *(i.e. they are not racing).*



Data race freedom

Theorem

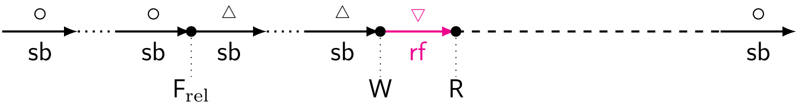
In a validly annotated execution, any two non-atomic accesses to the same location are ordered by $(\mathsf{sb} \cup \mathsf{sync})^+$ *(i.e. they are not racing).*



Data race freedom

Theorem

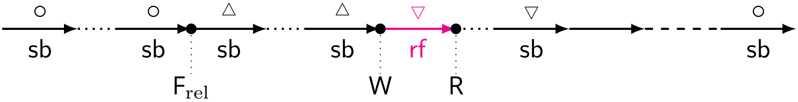
In a validly annotated execution, any two non-atomic accesses to the same location are ordered by $(\mathsf{sb} \cup \mathsf{sync})^+$ *(i.e. they are not racing).*



Data race freedom

Theorem

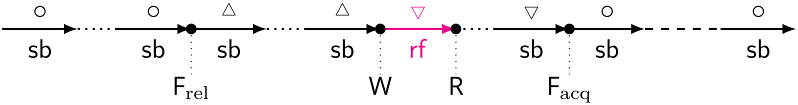
In a validly annotated execution, any two non-atomic accesses to the same location are ordered by $(\mathsf{sb} \cup \mathsf{sync})^+$ *(i.e. they are not racing).*



Data race freedom

Theorem

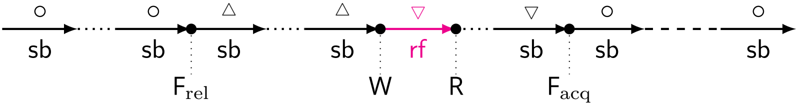
In a validly annotated execution, any two non-atomic accesses to the same location are ordered by $(\mathsf{sb} \cup \mathsf{sync})^{+}$ *(i.e. they are not racing).*

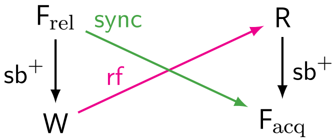


Data race freedom

Theorem

In a validly annotated execution, any two non-atomic accesses to the same location are ordered by $(\mathsf{sb} \cup \mathsf{sync})^+$ *(i.e. they are not racing).*

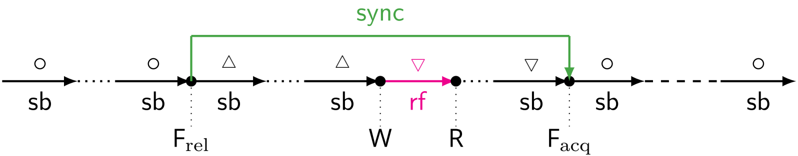


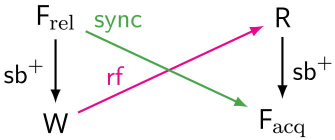


Data race freedom

Theorem

In a validly annotated execution, any two non-atomic accesses to the same location are ordered by $(\mathsf{sb} \cup \mathsf{sync})^+$ *(i.e. they are not racing).*

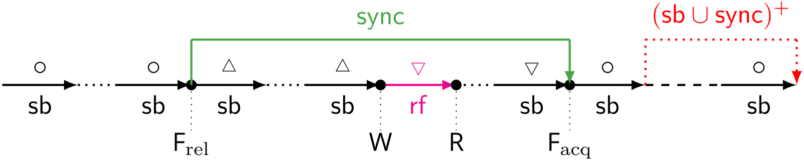


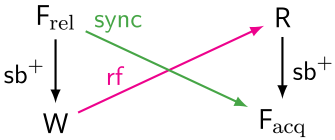


Data race freedom

Theorem

In a validly annotated execution, any two non-atomic accesses to the same location are ordered by $(\mathsf{sb} \cup \mathsf{sync})^+$ *(i.e. they are not racing).*





Summary and future work

Summary:

- FSL is the first logic that supports C11-style memory fences.
- FSL ensures
  - data race freedom,
  - memory safety, and
  - all reads read from initialized locations.
- Soundness proof is formalized in Coq:
  `http://plv.mpi-sws.org/fsl/`

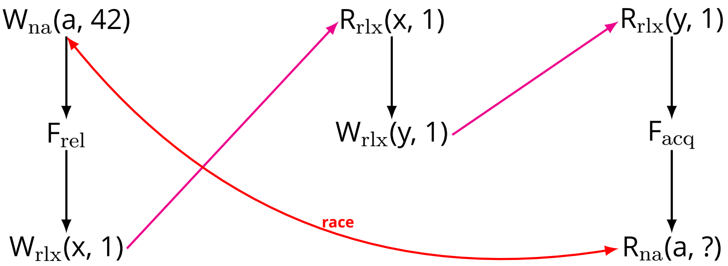
Future work:

- Support for CAS instructions and fractional permissions.
- Verify real-world algorithms (such as Rust's Arc).

Why the two modalities?

$$\mathcal{Q} \stackrel{\text{def}}{=} \lambda v.\ \textbf{if}\ v = 0\ \textbf{then}\ \mathsf{emp}\ \textbf{else}\ a \mapsto 42$$

$\{a \mapsto 0 * \mathsf{Rel}(x, \mathcal{Q})\}$		$\{\mathsf{Acq}(y, \mathcal{Q})\}$
`a = 42;`	$\{\mathsf{Acq}(x, \mathcal{Q})\}$	`while(`$\mathtt{y}_{\mathrm{rlx}}$` == 0);`
$\{a \mapsto 42 * \mathsf{Rel}(x, \mathcal{Q})\}$	`while(`$\mathtt{x}_{\mathrm{rlx}}$` == 0);`	$\{\Diamond a \mapsto 42\}$
$\mathtt{fence}_{\mathrm{rel}}$`;`	$\{\Diamond a \mapsto 42\}$	$\mathtt{fence}_{\mathrm{acq}}$`;`
$\{\Diamond a \mapsto 42 * \mathsf{Rel}(x, \mathcal{Q})\}$	$\mathtt{y}_{\mathrm{rlx}}$` = 1;`	$\{a \mapsto 42\}$
$\mathtt{x}_{\mathrm{rlx}}$` = 1;`	$\{\mathsf{true}\}$	`print(a);`
$\{\mathsf{true}\}$		$\{a \mapsto 42\}$



Some important properties of FSL assertions

- Release permissions are duplicable:

$$\mathsf{Rel}(\ell, \mathcal{Q}) \iff \mathsf{Rel}(\ell, \mathcal{Q}) * \mathsf{Rel}(\ell, \mathcal{Q})$$

- Acquire permissions are splittable:

$$\mathsf{Acq}(\ell, \mathcal{Q}_1) * \mathsf{Acq}(\ell, \mathcal{Q}_2) \iff \mathsf{Acq}(\ell, \lambda v.\, \mathcal{Q}_1(v) * \mathcal{Q}_2(v))$$

- Modalities ($\triangle$ and $\triangledown$) distribute over disjunction, conjunction, and separating conjunction:

$$\begin{aligned} \triangle(P \wedge Q) &\iff \triangle P \wedge \triangle Q & \qquad \triangledown(P \wedge Q) &\iff \triangledown P \wedge \triangledown Q \\ \triangle(P \vee Q) &\iff \triangle P \vee \triangle Q & \qquad \triangledown(P \vee Q) &\iff \triangledown P \vee \triangledown Q \\ \triangle(P * Q) &\iff \triangle P * \triangle Q & \qquad \triangledown(P * Q) &\iff \triangledown P * \triangledown Q \end{aligned}$$