Tackling Real-Life Relaxed Concurrency with FSL++

Marko Doko Viktor Vafeiadis

Max Planck Institute for Software Systems
(MPI-SWS)

ESOP
2017-04-26

Weak memory

- memory models weaker than sequential consistency (SC)
- gives us better performance

Logics for weak memory

- iCAP-TSO, OGRA, GPS, RSL, FSL

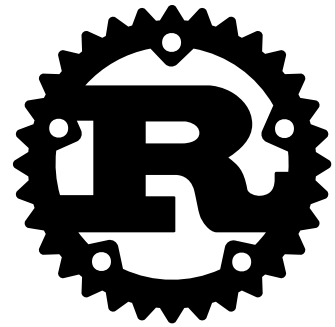
Current state of verification

- simplified algorithms & toy examples

In this talk

- **first verification of a non-simplified real-world algorithm**

Atomic Reference Counter (ARC)



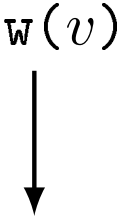
- part of the Rust standard library
- allows concurrent reads of a shared resource
- uses advanced weak memory primitives

How is ARC used?

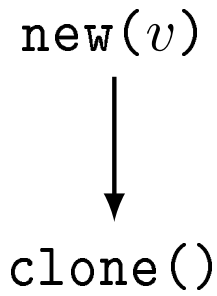
How is ARC used?

`new(`v`)`

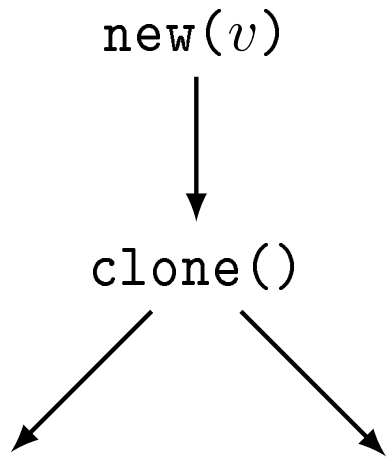
How is ARC used?



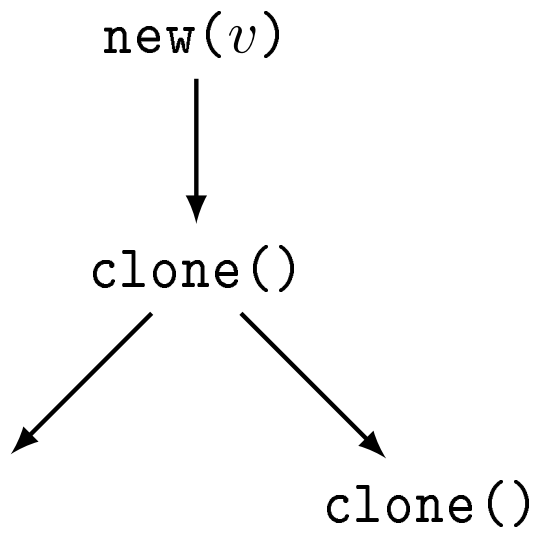
How is ARC used?



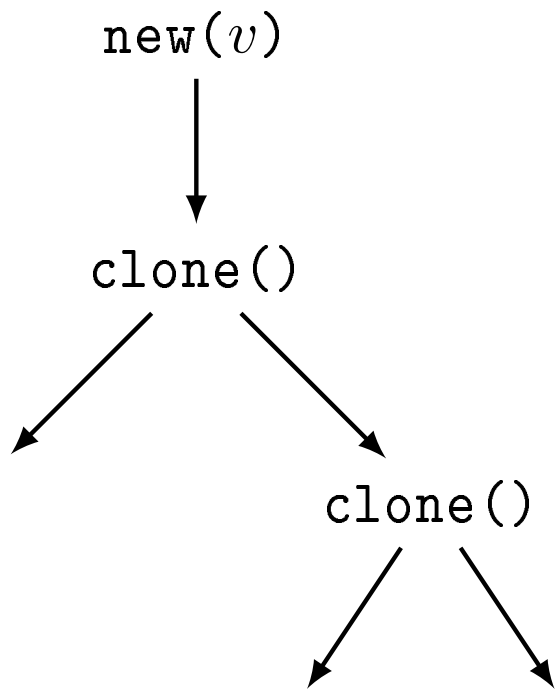
How is ARC used?



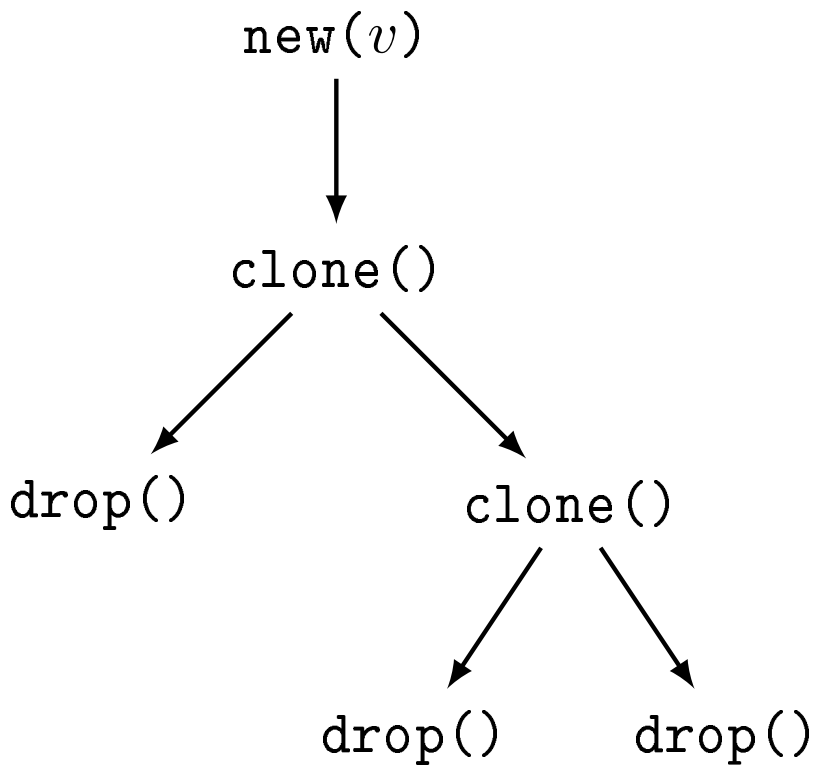
How is ARC used?



How is ARC used?



How is ARC used?



$$\begin{array}{rll}
\{\mathsf{emp}\} & a \texttt{ = new}(v) & \{\mathsf{ARC}(a, v)\} \\
\{\mathsf{ARC}(a, v)\} & y \texttt{ = read}(a) & \{y = v \wedge \mathsf{ARC}(a, v)\} \\
\{\mathsf{ARC}(a, v)\} & \texttt{clone}(a) & \{\mathsf{ARC}(a, v) * \mathsf{ARC}(a, v)\} \\
\{\mathsf{ARC}(a, v)\} & \texttt{drop}(a) & \{\mathsf{emp}\}
\end{array}$$

$$\begin{array}{rcl}
\{\mathsf{emp}\} & a = \texttt{new}(v) & \{\mathsf{ARC}(a, v)\} \\
\{\mathsf{ARC}(a, v)\} & y = \texttt{read}(a) & \{y = v \wedge \mathsf{ARC}(a, v)\} \\
\{\mathsf{ARC}(a, v)\} & \texttt{clone}(a) & \{\mathsf{ARC}(a, v) * \mathsf{ARC}(a, v)\} \\
\{\mathsf{ARC}(a, v)\} & \texttt{drop}(a) & \{\mathsf{emp}\}
\end{array}$$

```
new(v){
  a = alloc();
  a.data = v;
  a.count = 1;
  return a;
}

read(a){
  return a.data;
}
```

```
clone(a){
  FADD(a.count, +1);
}

drop(a){
  t = FADD(a.count, -1);
  if(t == 1){
    free(a);
  }
}
```

FADD = fetch_and_add

$$\{\mathsf{emp}\}\ a = \texttt{new}(v)\ \{\mathsf{ARC}(a, v)\}$$
$$\{\mathsf{ARC}(a, v)\}\ y = \texttt{read}(a)\ \{y = v \wedge \mathsf{ARC}(a, v)\}$$
$$\{\mathsf{ARC}(a, v)\}\ \texttt{clone}(a)\ \{\mathsf{ARC}(a, v) * \mathsf{ARC}(a, v)\}$$
$$\{\mathsf{ARC}(a, v)\}\ \texttt{drop}(a)\ \{\mathsf{emp}\}$$

```
new(v){
  a = alloc();
  a.data = v;
  a.count_rlx = 1;
  return a;
}

read(a){
  return a.data;
}
```

```
clone(a){
  FADD_rlx(a.count, +1);
}

drop(a){
  t = FADD_rel(a.count, -1);
  if(t == 1){
    fence_acq;
    free(a);
  }
}
```

FADD = fetch_and_add

FSL (Fenced Separation Logic) [VMCAI '16]

- ✓ supports **rel**, **acq**, and **rlx** accesses
- ✓ supports memory fences

Too weak to verify ARC

- ✗ concurrent plain (non-atomic) reads
  - SOLUTION: partial permissions
- ✗ `fetch_and_add` instructions
  - SOLUTION: new rules
- ✗ not expressive enough
  - SOLUTION: ghost state

$$\{\quad\}\quad \texttt{FADD}_{\mathbf{acq\_rel}}(x,t)\quad \{\quad\}$$

$\{\mathsf{U}(x, \mathcal{Q}) * P\}$ $\texttt{FADD}_{\textbf{acq\_rel}}(x, t)$ $\{\quad\}$

$\{\mathsf{U}(x, \mathcal{Q}) * P\}$ $\texttt{FADD}_{\textbf{acq\_rel}}(x, t)$ $\{$ $\}$

$\mathcal{Q}$: $Val \to Assn$ is invariant for x:
x has value $c \Rightarrow$ the invariant owns $\mathcal{Q}(c)$

$$\frac{\forall c.\ \mathcal{Q}(c) \Rightarrow R \qquad \forall c.\ P \Rightarrow \mathcal{Q}(c+t)}{\{\mathsf{U}(x,\mathcal{Q}) * P\}\quad \texttt{FADD}_{\textbf{acq\_rel}}(x,t)\quad \{\mathsf{U}(x,\mathcal{Q}) * R\}}$$

$\mathcal{Q}$: $Val \to Assn$ is invariant for x:
x has value $c \Rightarrow$ the invariant owns $\mathcal{Q}(c)$

Updating the value of x from c to $c+t$:
(1) get $\mathcal{Q}(c)$ out of the invariant
(2) put $\mathcal{Q}(c+t)$ back into the invariant

$$\frac{\forall c.\ \mathcal{Q}(c) \ \Rightarrow R * T \qquad \forall c.\ T * P \Rightarrow \mathcal{Q}(c+t)}{\{\mathsf{U}(x, \mathcal{Q}) * P\} \quad \texttt{FADD}_{\textbf{acq\_rel}}(x, t) \quad \{\mathsf{U}(x, \mathcal{Q}) * R\}}$$

$\mathcal{Q}$: $Val \to Assn$ is invariant for x:
x has value $c \Rightarrow$ the invariant owns $\mathcal{Q}(c)$

Updating the value of x from c to $c + t$:
(1) get $\mathcal{Q}(c)$ out of the invariant
(2) put $\mathcal{Q}(c + t)$ back into the invariant

$$\frac{\begin{array}{l}\forall c.\ \mathcal{Q}(c) \ \Rightarrow R * T \\ \forall c.\ T * P \Rightarrow \mathcal{Q}(c+t)\end{array}}{\begin{array}{lll}\{\mathsf{U}(x,\mathcal{Q}) * P\} & \texttt{FADD}_{\mathbf{acq\_rel}}(x,t) & \{\mathsf{U}(x,\mathcal{Q}) * R\} \\ \{\quad\quad\quad\quad\} & \texttt{FADD}_{\mathbf{rel}}(x,t) & \{\quad\quad\quad\quad\}\end{array}}$$

```
drop(a){
  t = FADD_rel(a.count, -1);
  if(t == 1){
    fence_acq;
    free(a);
  }
}
```

$$\frac{\begin{array}{c}\forall c.\ \mathcal{Q}(c) \ \Rightarrow R * T \\ \forall c.\ T * P \Rightarrow \mathcal{Q}(c+t)\end{array}}{\begin{array}{ccc}\{\mathsf{U}(x,\mathcal{Q}) * P\} & \texttt{FADD}_{\mathbf{acq\_rel}}(x,t) & \{\mathsf{U}(x,\mathcal{Q}) * R\} \\ \{\mathsf{U}(x,\mathcal{Q}) * P\} & \texttt{FADD}_{\mathbf{rel}}(x,t) & \{\mathsf{U}(x,\mathcal{Q}) * \bigtriangledown R\}\end{array}}$$

```
drop(a){
  t = FADD_rel(a.count, -1);
  if(t == 1){
    fence_acq;
    free(a);
  }
}
```

$$\frac{\forall c.\ \mathcal{Q}(c) \Rightarrow R * T \qquad \forall c.\ T * P \Rightarrow \mathcal{Q}(c+t)}{\begin{array}{ccc} \{\mathsf{U}(x,\mathcal{Q}) * P\} & \texttt{FADD}_{\mathbf{acq\_rel}}(x,t) & \{\mathsf{U}(x,\mathcal{Q}) * R\} \\ \{\mathsf{U}(x,\mathcal{Q}) * P\} & \texttt{FADD}_{\mathbf{rel}}(x,t) & \{\mathsf{U}(x,\mathcal{Q}) * \triangledown R\} \end{array}}$$

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

```
drop(a){
  t = FADD_rel(a.count, -1);
  if(t == 1){
    fence_acq;
    free(a);
  }
}
```

$$\frac{\forall c.\ \mathcal{Q}(c) \ \Rightarrow R * T \qquad \forall c.\ T * P \Rightarrow \mathcal{Q}(c+t)}{\begin{array}{ccc} \{\mathsf{U}(x,\mathcal{Q}) * P\} & \mathtt{FADD}_{\mathbf{acq\_rel}}(x,t) & \{\mathsf{U}(x,\mathcal{Q}) * R\} \\ \{\mathsf{U}(x,\mathcal{Q}) * P\} & \mathtt{FADD}_{\mathbf{rel}}(x,t) & \{\mathsf{U}(x,\mathcal{Q}) * \bigtriangledown R\} \\ \{\quad\quad\quad\quad\} & \mathtt{FADD}_{\mathbf{rlx}}(x,t) & \{\quad\quad\quad\quad\} \end{array}}$$

```
drop(a){
  t = FADD_rel(a.count, -1);
  if(t == 1){
    fence_acq;
    free(a);
  }
}
```

```
clone(a){
  FADD_rlx(a.count, +1);
}
```

$$\frac{\forall c.\ \mathcal{Q}(c) \Rightarrow R * T \qquad \forall c.\ T * P \Rightarrow \mathcal{Q}(c+t)}{\begin{array}{lll} \{\mathsf{U}(x,\mathcal{Q}) * P\} & \mathtt{FADD}_{\mathbf{acq\_rel}}(x,t) & \{\mathsf{U}(x,\mathcal{Q}) * R\} \\ \{\mathsf{U}(x,\mathcal{Q}) * P\} & \mathtt{FADD}_{\mathbf{rel}}(x,t) & \{\mathsf{U}(x,\mathcal{Q}) * \triangledown R\} \\ \{\ \ \ \ \ \ \ \ \ \ \} & \mathtt{FADD}_{\mathbf{rlx}}(x,t) & \{\mathsf{U}(x,\mathcal{Q}) * \triangledown R\} \end{array}}$$

$\{\triangledown P\}\ \mathtt{fence}_{\mathbf{acq}}\ \{P\}$

```
drop(a){
  t = FADD_rel(a.count, -1);
  if(t == 1){
    fence_acq;
    free(a);
  }
}
```

```
clone(a){
  FADD_rlx(a.count, +1);
}
```

$$\frac{\forall c.\ \mathcal{Q}(c) \Rightarrow R * T \qquad \forall c.\ T * P \Rightarrow \mathcal{Q}(c+t)}{\begin{array}{ccc} \{\mathsf{U}(x,\mathcal{Q}) * P\} & \mathtt{FADD}_{\mathbf{acq\_rel}}(x,t) & \{\mathsf{U}(x,\mathcal{Q}) * R\} \\ \{\mathsf{U}(x,\mathcal{Q}) * P\} & \mathtt{FADD}_{\mathbf{rel}}(x,t) & \{\mathsf{U}(x,\mathcal{Q}) * \bigtriangledown R\} \\ \{\mathsf{U}(x,\mathcal{Q}) * \bigtriangleup P\} & \mathtt{FADD}_{\mathbf{rlx}}(x,t) & \{\mathsf{U}(x,\mathcal{Q}) * \bigtriangledown R\} \end{array}}$$

$\{P\}\,\mathtt{fence}_{\mathbf{rel}}\,\{\bigtriangleup P\}$

$\{\bigtriangledown P\}\,\mathtt{fence}_{\mathbf{acq}}\,\{P\}$

```
drop(a){
  t = FADD_rel(a.count, -1);
  if(t == 1){
    fence_acq;
    free(a);
  }
}
```

```
clone(a){
  FADD_rlx(a.count, +1);
}
```

$$\frac{\forall c.\ \mathcal{Q}(c) \Rightarrow R * T \qquad \forall c.\ T * P \Rightarrow \mathcal{Q}(c+t)}{\begin{array}{lll} \{\mathsf{U}(x,\mathcal{Q}) * P\} & \texttt{FADD}_{\mathbf{rel}}(x,t) & \{\mathsf{U}(x,\mathcal{Q}) * \bigtriangledown R\} \\ \{\mathsf{U}(x,\mathcal{Q}) * \bigtriangleup P\} & \texttt{FADD}_{\mathbf{rlx}}(x,t) & \{\mathsf{U}(x,\mathcal{Q}) * \bigtriangledown R\} \end{array}}$$

What is $\mathsf{ARC}(a, v)$?

Which invariant to choose for the counter $a.\texttt{count}$?

$\{\mathsf{ARC}(a, v)\}$

```
drop(a){
  t = FADD_rel(a.count, -1);
  if(t == 1){
    fence_acq;
    free(a);
  }}
```

$\{\mathsf{emp}\}$

$\{\mathsf{ARC}(a, v)\}$

```
clone(a){
  FADD_rlx(a.count, +1);
}
```

$\{\mathsf{ARC}(a, v) * \mathsf{ARC}(a, v)\}$

$$\frac{\forall c.\ \mathcal{Q}(c) \Rightarrow R * T \qquad \forall c.\ T * P \Rightarrow \mathcal{Q}(c+t)}{\begin{array}{lll} \{\mathsf{U}(x,\mathcal{Q}) * P\} & \texttt{FADD}_{\mathbf{rel}}(x,t) & \{\mathsf{U}(x,\mathcal{Q}) * \bigtriangledown R\} \\ \{\mathsf{U}(x,\mathcal{Q}) * \bigtriangleup P\} & \texttt{FADD}_{\mathbf{rlx}}(x,t) & \{\mathsf{U}(x,\mathcal{Q}) * \bigtriangledown R\} \end{array}}$$

What is $\mathsf{ARC}(a, v)$?

$$\mathsf{ARC}(a,v) = \mathsf{U}(a.\texttt{count}, \mathcal{Q}) * \exists q \in \langle 0, 1]. \ a.\texttt{data} \overset{q}{\mapsto} v$$

Which invariant to choose for the counter $a.\texttt{count}$?

$\{\mathsf{ARC}(a,v)\}$

```
drop(a){
  t = FADD_rel(a.count, -1);
  if(t == 1){
    fence_acq;
    free(a);
  }}
```

$\{\mathsf{emp}\}$

$\{\mathsf{ARC}(a,v)\}$

```
clone(a){
  FADD_rlx(a.count, +1);
}
```

$\{\mathsf{ARC}(a,v) * \mathsf{ARC}(a,v)\}$

$$\frac{\forall c.\ \mathcal{Q}(c) \Rightarrow R * T \qquad \forall c.\ T * P \Rightarrow \mathcal{Q}(c+t)}{\begin{array}{lll} \{\mathsf{U}(x,\mathcal{Q}) * P\} & \texttt{FADD}_{\mathbf{rel}}(x,t) & \{\mathsf{U}(x,\mathcal{Q}) * \bigtriangledown R\} \\ \{\mathsf{U}(x,\mathcal{Q}) * \bigtriangleup P\} & \texttt{FADD}_{\mathbf{rlx}}(x,t) & \{\mathsf{U}(x,\mathcal{Q}) * \bigtriangledown R\} \end{array}}$$

What is $\mathsf{ARC}(a, v)$?

$$\mathsf{ARC}(a,v) = \mathsf{U}(a.\texttt{count}, \mathcal{Q}) * \exists q \in \langle 0, 1]. \ a.\texttt{data} \overset{q}{\mapsto} v * ???$$

Which invariant to choose for the counter $a.\texttt{count}$?

$\{\mathsf{ARC}(a,v)\}$

```
drop(a){
  t = FADD_rel(a.count, -1);
  if(t == 1){
    fence_acq;
    free(a);
  }}
```

$\{\mathsf{emp}\}$

$\{\mathsf{ARC}(a,v)\}$

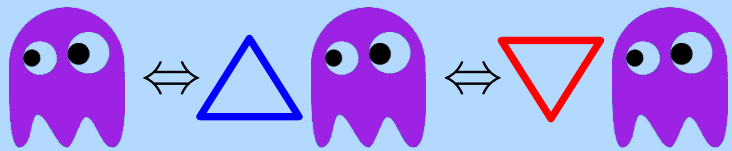
```
clone(a){
  FADD_rlx(a.count, +1);
}
```

$\{\mathsf{ARC}(a,v) * \mathsf{ARC}(a,v)\}$

$$\frac{\forall c.\ \mathcal{Q}(c) \Rightarrow R * T \qquad \forall c.\ T * P \Rightarrow \mathcal{Q}(c+t)}{\begin{array}{lll} \{\mathsf{U}(x,\mathcal{Q}) * P\} & \texttt{FADD}_{\mathbf{rel}}(x,t) & \{\mathsf{U}(x,\mathcal{Q}) * \bigtriangledown R\} \\ \{\mathsf{U}(x,\mathcal{Q}) * \bigtriangleup P\} & \texttt{FADD}_{\mathbf{rlx}}(x,t) & \{\mathsf{U}(x,\mathcal{Q}) * \bigtriangledown R\} \end{array}}$$

Modalities ($\bigtriangleup$ and $\bigtriangledown$) prevent data races.

Ghost state is not accessed $\Rightarrow$ no races on ghosts!



$\{\mathsf{ARC}(a,v)\}$

```
drop(a){
  t = FADD_rel(a.count, -1);
  if(t == 1){
    fence_acq;
    free(a);
  }}
```

$\{\mathsf{emp}\}$

$\{\mathsf{ARC}(a,v)\}$

```
clone(a){
  FADD_rlx(a.count, +1);
}
```

$\{\mathsf{ARC}(a,v) * \mathsf{ARC}(a,v)\}$

$$\frac{\begin{array}{l}\forall c.\ \mathcal{Q}(c) \Rightarrow R * T \\ \forall c.\ T * P \Rightarrow \mathcal{Q}(c+t)\end{array}}{\begin{array}{rcl}\{\mathsf{U}(x,\mathcal{Q}) * P\} & \texttt{FADD}_{\mathbf{rel}}(x,t) & \{\mathsf{U}(x,\mathcal{Q}) * \bigtriangledown R\} \\ \{\mathsf{U}(x,\mathcal{Q}) * \bigtriangleup P\} & \texttt{FADD}_{\mathbf{rlx}}(x,t) & \{\mathsf{U}(x,\mathcal{Q}) * \bigtriangledown R\}\end{array}}$$

What is $\mathsf{ARC}(a, v)$?

$$\mathsf{ARC}(a,v) = \mathsf{U}(a.\texttt{count}, \mathcal{Q}) * \exists q \in \langle 0, 1]. \ a.\texttt{data} \overset{q}{\mapsto} v * ???$$

Which invariant to choose for the counter $a.\texttt{count}$?

$\{\mathsf{ARC}(a,v)\}$

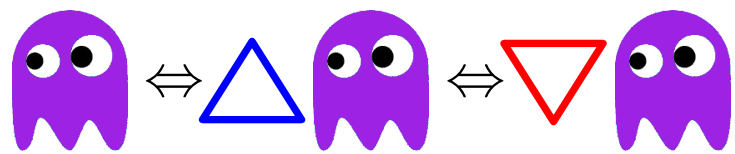
```
drop(a){
  t = FADD_rel(a.count, -1);
  if(t == 1){
    fence_acq;
    free(a);
  }}
```

$\{\mathsf{emp}\}$

$\{\mathsf{ARC}(a,v)\}$

```
clone(a){
  FADD_rlx(a.count, +1);
}
```

$\{\mathsf{ARC}(a,v) * \mathsf{ARC}(a,v)\}$



$$\frac{\forall c.\ \mathcal{Q}(c) \Rightarrow R * T \qquad \forall c.\ T * P \Rightarrow \mathcal{Q}(c+t)}{\begin{array}{lll} \{\mathsf{U}(x,\mathcal{Q}) * P\} & \texttt{FADD}_{\mathbf{rel}}(x,t) & \{\mathsf{U}(x,\mathcal{Q}) * \bigtriangledown R\} \\ \{\mathsf{U}(x,\mathcal{Q}) * \bigtriangleup P\} & \texttt{FADD}_{\mathbf{rlx}}(x,t) & \{\mathsf{U}(x,\mathcal{Q}) * \bigtriangledown R\} \end{array}}$$

What is $\mathsf{ARC}(a, v)$?

$$\mathsf{ARC}(a,v) = \mathsf{U}(a.\texttt{count}, \mathcal{Q}) * \exists q \in \langle 0, 1]. \ a.\texttt{data} \overset{q}{\mapsto} v * ???$$

Which invariant to choose for the counter $a.\texttt{count}$?

$$\mathcal{Q}(c) \iff \mathcal{Q}(c+1) * \text{👻}$$

$\{\mathsf{ARC}(a,v)\}$

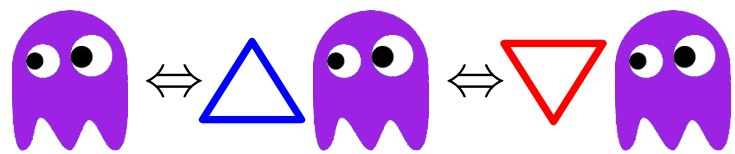
```
drop(a){
  t = FADD_rel(a.count, -1);
  if(t == 1){
    fence_acq;
    free(a);
  }}
```

$\{\mathsf{emp}\}$

$\{\mathsf{ARC}(a,v)\}$

```
clone(a){
  FADD_rlx(a.count, +1);
}
```

$\{\mathsf{ARC}(a,v) * \mathsf{ARC}(a,v)\}$



$$\frac{\forall c.\ \mathcal{Q}(c) \Rightarrow R * T \qquad \forall c.\ T * P \Rightarrow \mathcal{Q}(c+t)}{\begin{array}{lll} \{\mathsf{U}(x,\mathcal{Q}) * P\} & \texttt{FADD}_{\mathbf{rel}}(x,t) & \{\mathsf{U}(x,\mathcal{Q}) * \bigtriangledown R\} \\ \{\mathsf{U}(x,\mathcal{Q}) * \bigtriangleup P\} & \texttt{FADD}_{\mathbf{rlx}}(x,t) & \{\mathsf{U}(x,\mathcal{Q}) * \bigtriangledown R\} \end{array}}$$

What is $\mathsf{ARC}(a, v)$?

$$\mathsf{ARC}(a,v) = \mathsf{U}(a.\texttt{count}, \mathcal{Q}) * \exists q \in \langle 0,1]. \ a.\texttt{data} \overset{q}{\mapsto} v \ * \ (1-q)$$ 

Which invariant to choose for the counter $a.\texttt{count}$?

$$\mathcal{Q}(c) \iff \mathcal{Q}(c+1) *$$ 

$\{\mathsf{ARC}(a,v)\}$

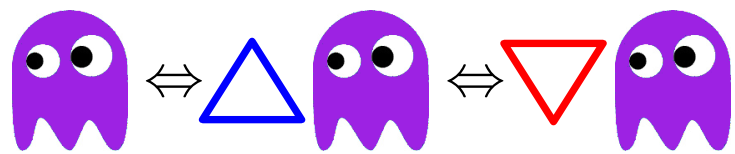
```
drop(a){
  t = FADD_rel(a.count, -1);
  if(t == 1){
    fence_acq;
    free(a);
  }}
```

$\{\mathsf{emp}\}$

$\{\mathsf{ARC}(a,v)\}$

```
clone(a){
  FADD_rlx(a.count, +1);
}
```

$\{\mathsf{ARC}(a,v) * \mathsf{ARC}(a,v)\}$



$$\frac{\forall c.\ \mathcal{Q}(c) \Rightarrow R * T \qquad \forall c.\ T * P \Rightarrow \mathcal{Q}(c+t)}{\begin{array}{lll} \{\mathsf{U}(x,\mathcal{Q}) * P\} & \texttt{FADD}_{\mathbf{rel}}(x,t) & \{\mathsf{U}(x,\mathcal{Q}) * \bigtriangledown R\} \\ \{\mathsf{U}(x,\mathcal{Q}) * \bigtriangleup P\} & \texttt{FADD}_{\mathbf{rlx}}(x,t) & \{\mathsf{U}(x,\mathcal{Q}) * \bigtriangledown R\} \end{array}}$$

What is $\mathsf{ARC}(a,v)$?

$$\mathsf{ARC}(a,v) * \text{👻} \Rightarrow \mathsf{ARC}(a,v) * \mathsf{ARC}(a,v)$$

Which invariant to choose for the counter $a.\texttt{count}$?

$$\mathcal{Q}(c) \iff \mathcal{Q}(c+1) * \text{👻}$$

$\{\mathsf{ARC}(a,v)\}$

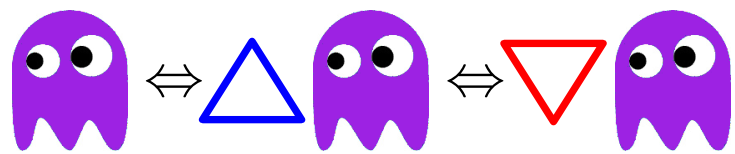
```
drop(a){
  t = FADD_rel(a.count, -1);
  if(t == 1){
    fence_acq;
    free(a);
  }}
```

$\{\mathsf{emp}\}$

$\{\mathsf{ARC}(a,v)\}$

```
clone(a){
  FADD_rlx(a.count, +1);
}
```

$\{\mathsf{ARC}(a,v) * \mathsf{ARC}(a,v)\}$



Summary:

- ARC: simple (but interesting) algorithm with advanced weak memory constructs
- FSL++ = FSL + partial permissions
  + rules for atomic updates (CAS, fetch & add)
  + ghost state
- ARC verification using FSL++ formalized in Coq
  `http://plv.mpi-sws.org/fsl/`

Future work:

- verify more examples
- adapt FSL++ for new memory models
  (e.g. promising semantics [Kang et al. POPL '17])

```
clone(a){
  FADD_rlx(a.count, +1);
}
```

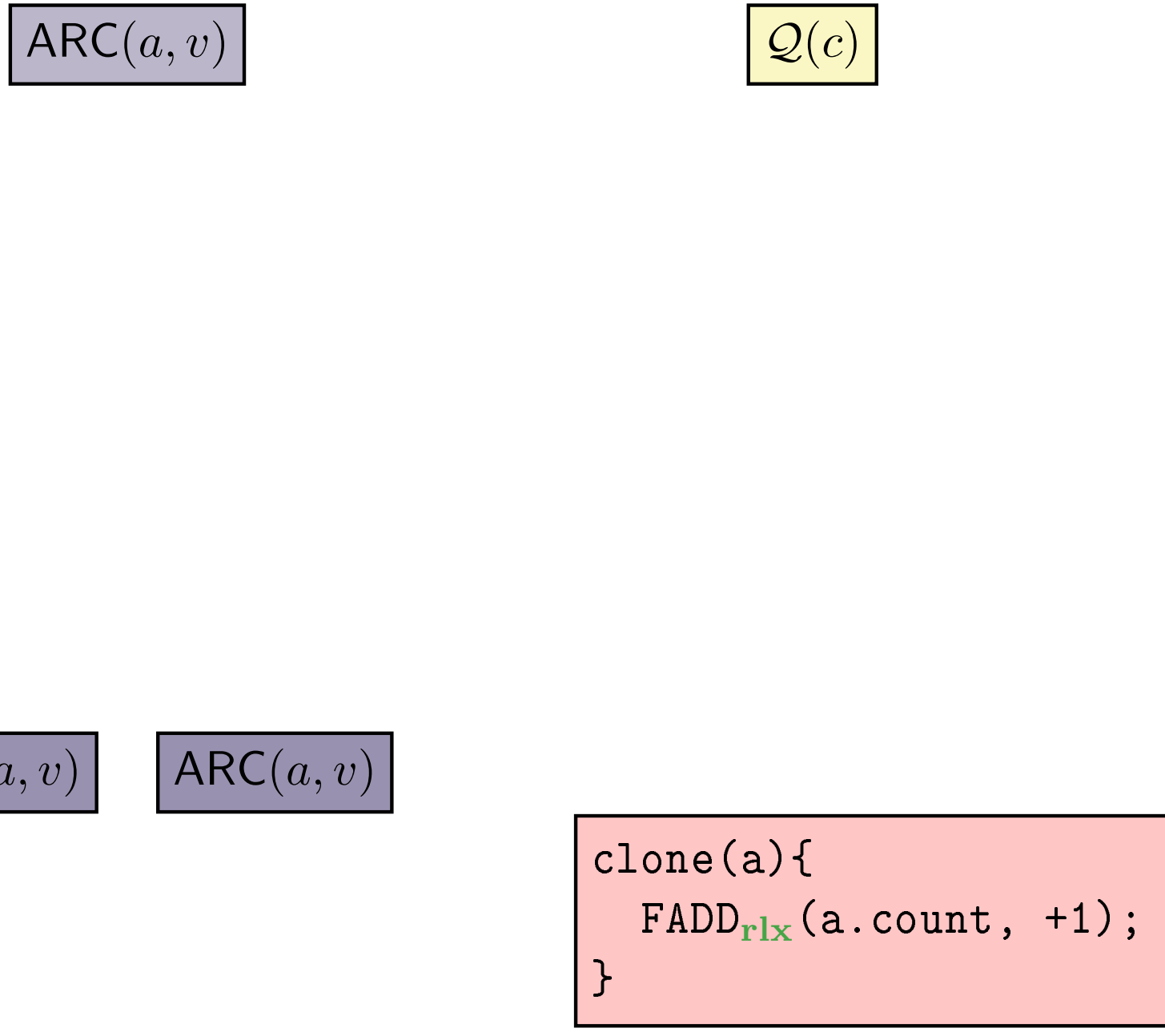
ARC(a, v)

```
clone(a){
  FADD_rlx(a.count, +1);
}
```

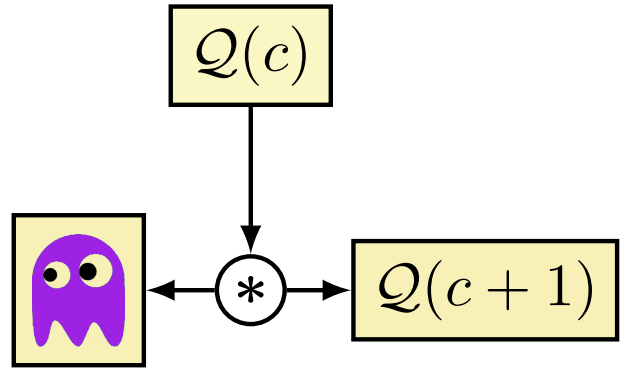
ARC(a, v)

ARC(a, v) ARC(a, v)

```
clone(a){
  FADD_rlx(a.count, +1);
}
```


ARC(a, v)
$\mathcal{Q}(c)$
ARC(a, v)
ARC(a, v)
clone(a){
FADD$_{\mathbf{rlx}}$(a.count, +1);
}

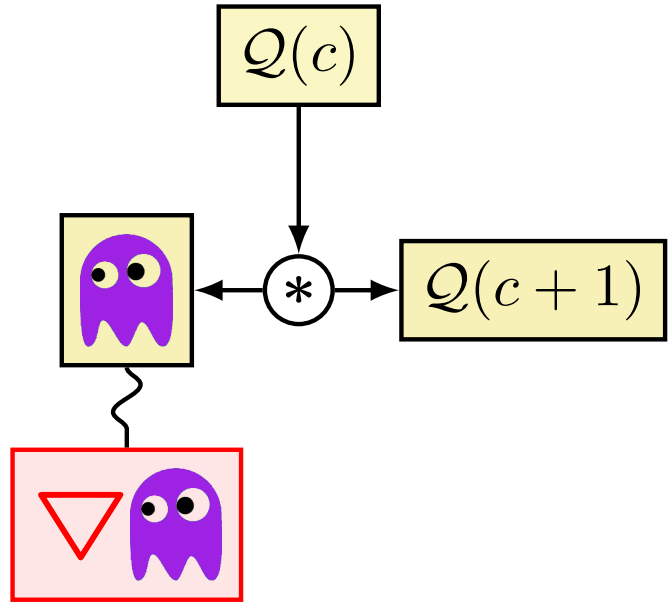
ARC(a, v)



ARC(a, v) ARC(a, v)

```
clone(a){
  FADD_rlx(a.count, +1);
}
```

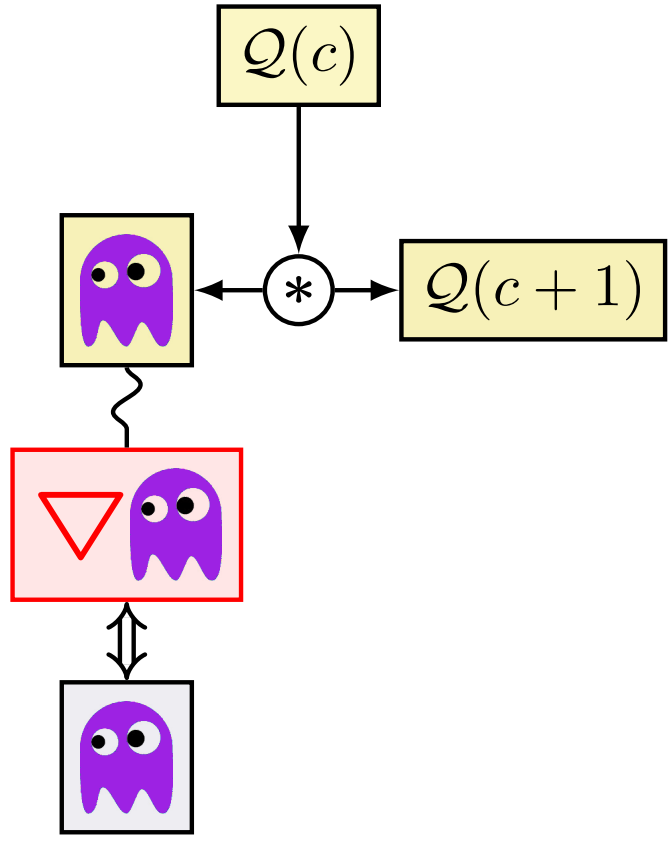
ARC(a, v)



ARC(a, v) ARC(a, v)

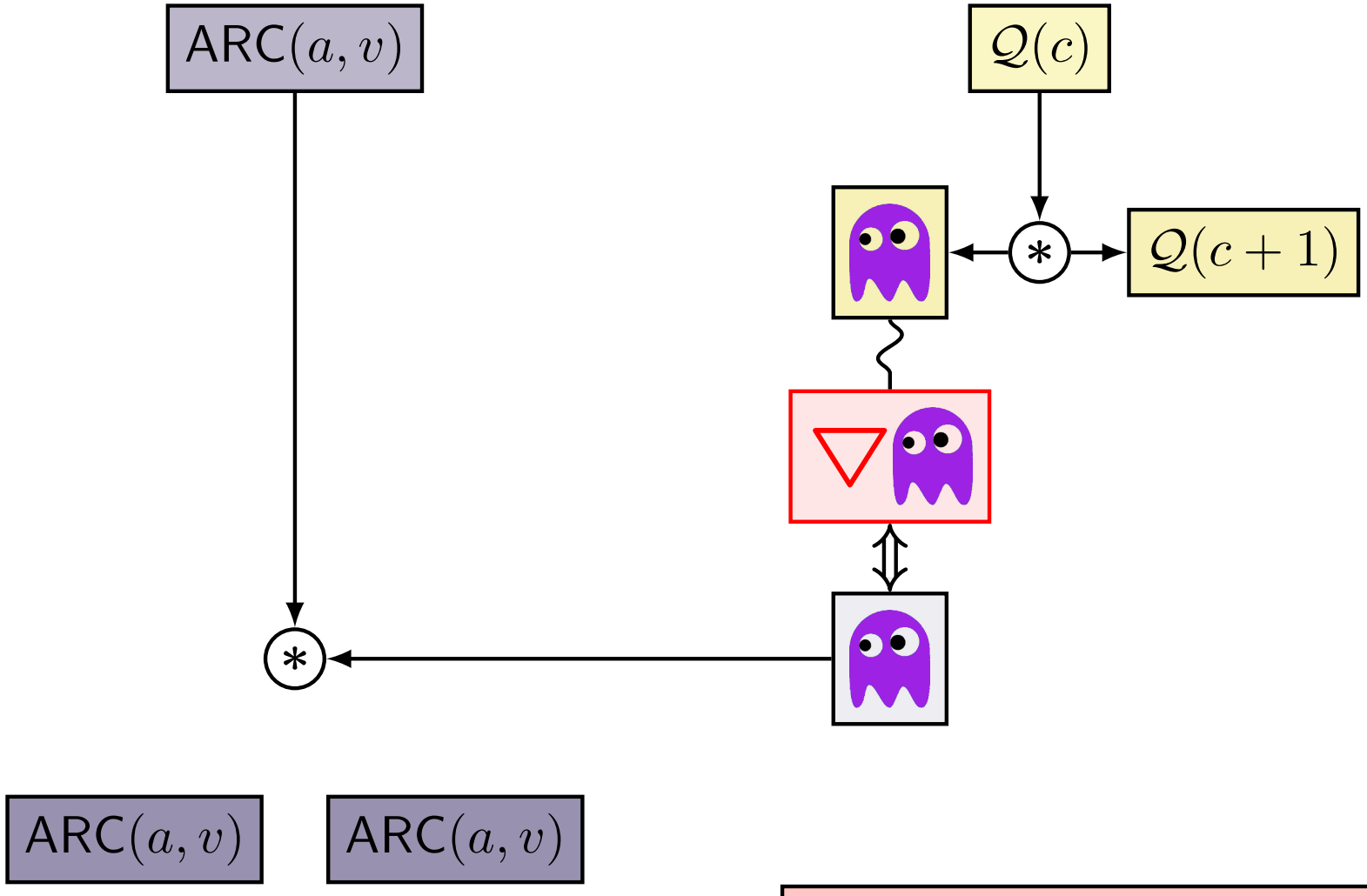
```
clone(a){
  FADD_rlx(a.count, +1);
}
```

ARC(a, v)

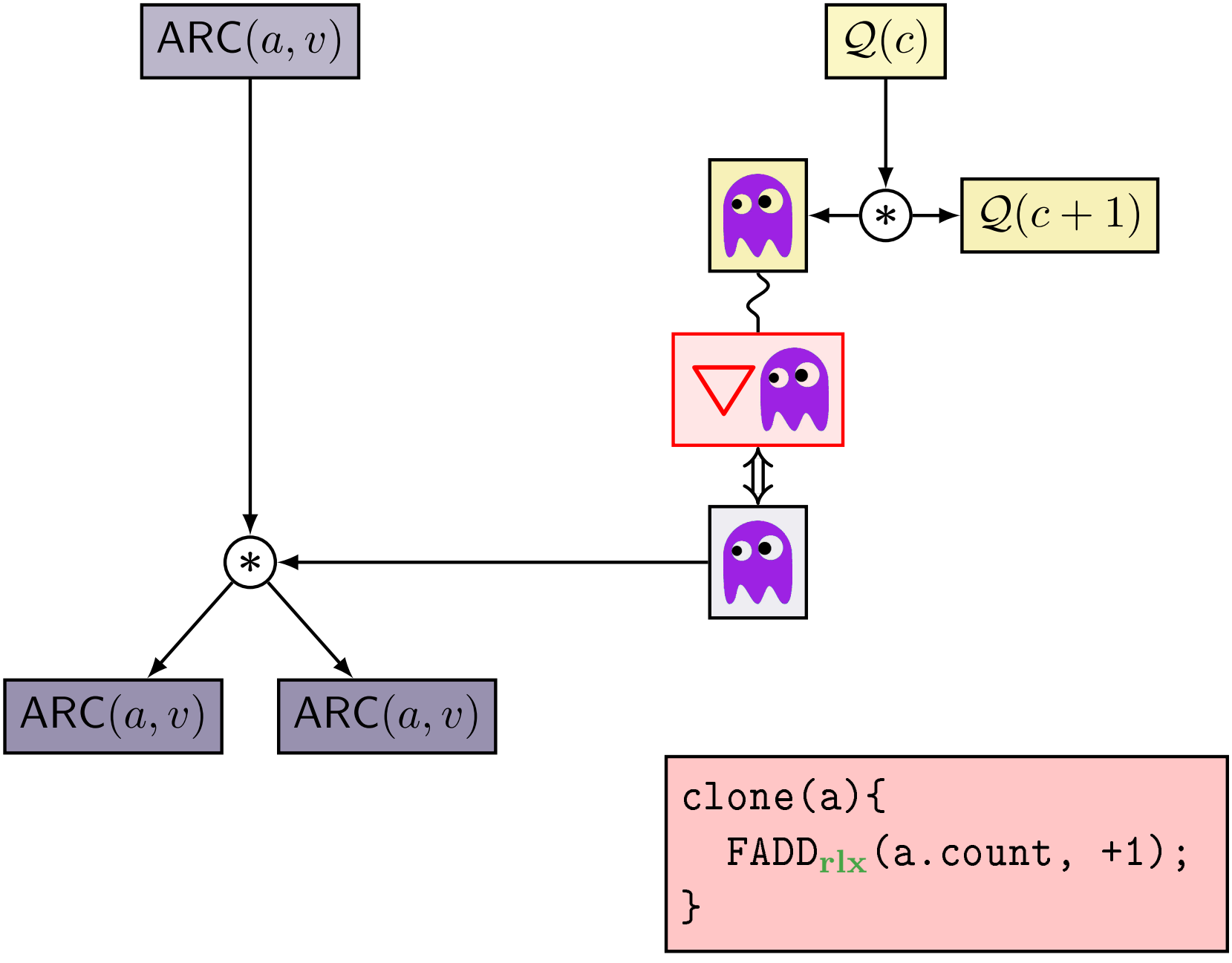


ARC(a, v) ARC(a, v)

```
clone(a){
  FADD_rlx(a.count, +1);
}
```




```
clone(a){
  FADD_rlx(a.count, +1);
}
```


ARC(a, v)
Q(c)
*
Q(c + 1)
*
ARC(a, v)
ARC(a, v)
clone(a){
FADD_rlx(a.count, +1);
}

```
drop(a){
  t = FADD_rel(a.count, -1);
  if(t == 1){
    fence_acq;
    free(a);
  }
}
```

ARC(a, v)

```
drop(a){
  t = FADD_rel(a.count, -1);
  if(t == 1){
    fence_acq;
    free(a);
  }
}
```

Decrementing the counter from $c > 1$:

ARC(a, v)

```
drop(a){
  t = FADD_rel(a.count, -1);
  if(t == 1){
    fence_acq;
    free(a);
  }
}
```

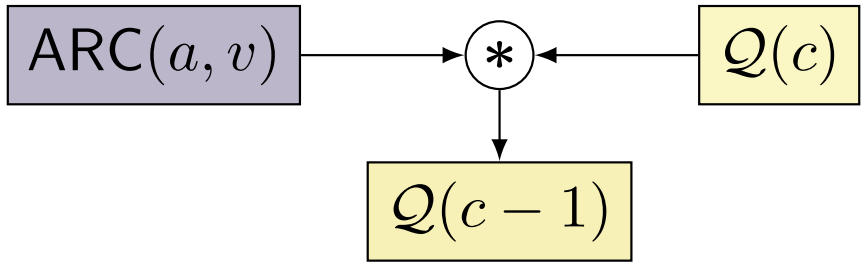
Decrementing the counter from $c > 1$:

ARC(a, v)

$\mathcal{Q}(c)$

```
drop(a){
  t = FADD_rel(a.count, -1);
  if(t == 1){
    fence_acq;
    free(a);
  }
}
```

Decrementing the counter from $c > 1$:




```
drop(a){
  t = FADD_rel(a.count, -1);
  if(t == 1){
    fence_acq;
    free(a);
  }
}
```

Decrementing the counter from $c = 1$:

ARC(a, v)

```
drop(a){
  t = FADD_rel(a.count, -1);
  if(t == 1){
    fence_acq;
    free(a);
  }
}
```

Decrementing the counter from $c = 1$:

ARC(a, v)

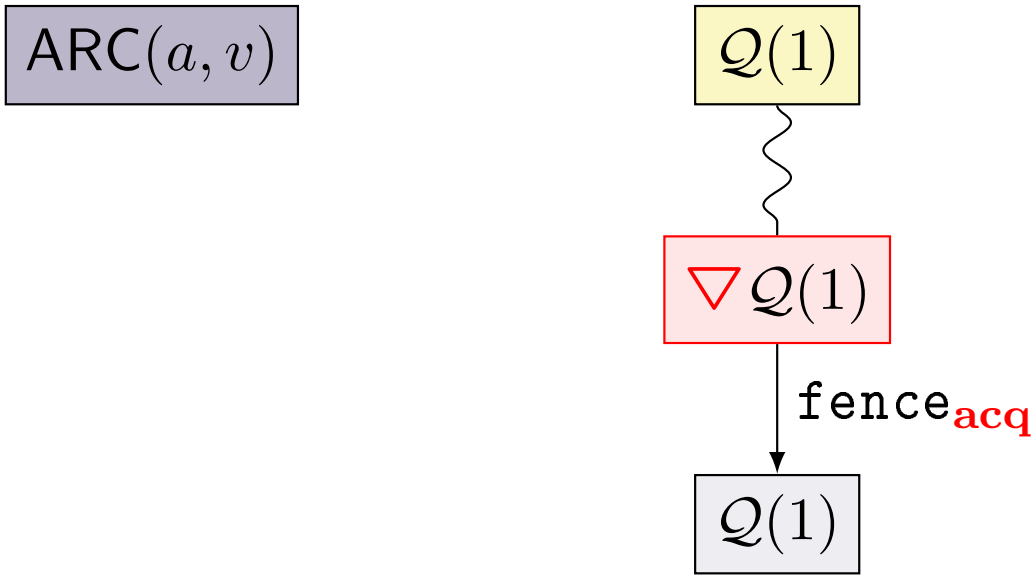
$\mathcal{Q}(1)$

```
drop(a){
  t = FADD_rel(a.count, -1);
  if(t == 1){
    fence_acq;
    free(a);
  }
}
```

Decrementing the counter from $c = 1$:

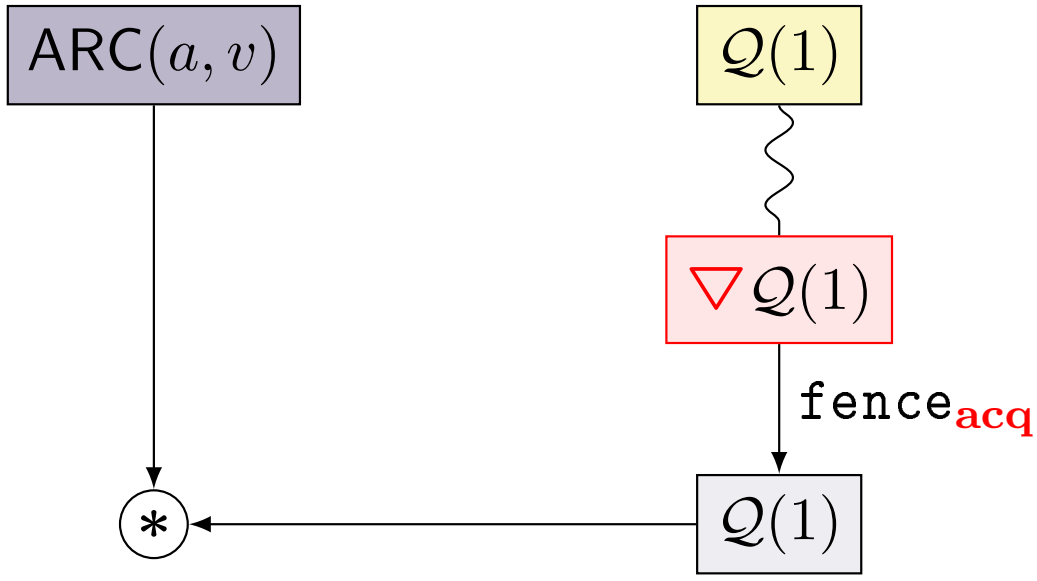

```
drop(a){
  t = FADD_rel(a.count, -1);
  if(t == 1){
    fence_acq;
    free(a);
  }
}
```

Decrementing the counter from $c = 1$:



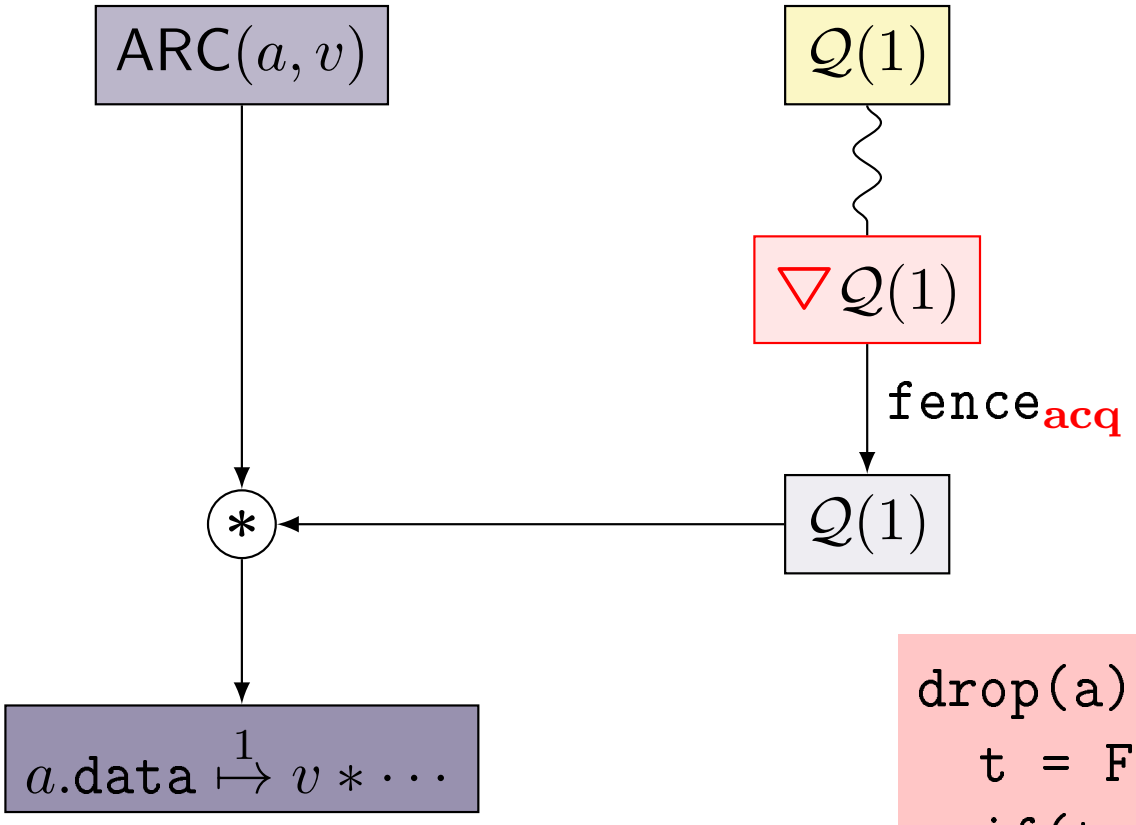

```
drop(a){
  t = FADD_rel(a.count, -1);
  if(t == 1){
    fence_acq;
    free(a);
  }
}
```

Decrementing the counter from $c = 1$:



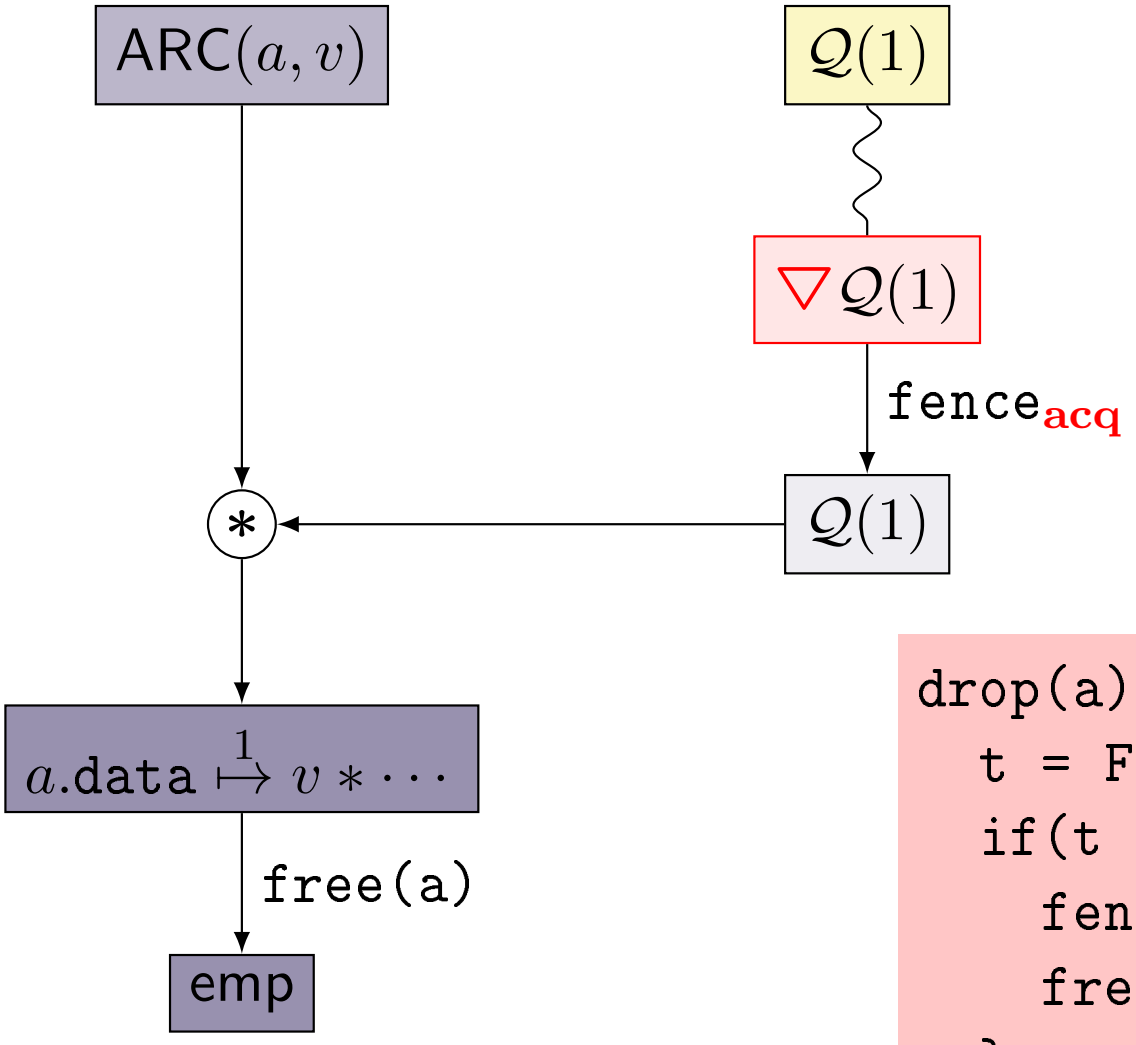

```
drop(a){
  t = FADD_rel(a.count, -1);
  if(t == 1){
    fence_acq;
    free(a);
  }
}
```

Decrementing the counter from $c = 1$:




```
drop(a){
  t = FADD_rel(a.count, -1);
  if(t == 1){
    fence_acq;
    free(a);
  }
}
```

Decrementing the counter from $c = 1$:




```
drop(a){
  t = FADD_rel(a.count, -1);
  if(t == 1){
    fence_acq;
    free(a);
  }
}
```

$$\mathcal{Q} \stackrel{\text{def}}{=} \lambda c.\, \textbf{if } c = 0 \textbf{ then } \text{  }:0 * \text{  }:0$$

$$\textbf{else } \exists f \in [0,1].\, a.\texttt{data} \overset{f}{\mapsto} v * \text{  }:(c - 1 + f) * \text{  }:(1 - f)$$

$$\mathsf{ARC}(a, v) \stackrel{\text{def}}{=} \mathsf{U}(a.\,\texttt{count}, \mathcal{Q}(a.\texttt{data})) *$$

$$\exists q \in \langle 0,1].\, a.\texttt{data} \overset{q}{\mapsto} v * (1-q)\cdot \text{  } * q \cdot \text{  }$$

$$p \cdot \text{  } * q \cdot \text{  }^{+} \iff (p+q) \cdot \text{  }$$

$$\text{  }:p * \text{  }:q \iff \mathsf{false}$$

$$p \cdot \text{  } * \text{  }:q \iff \text{  }:q * p \cdot \text{  } \iff \begin{cases} \text{  }:(q-p) & \text{if } q - p \geqslant 0 \\ \mathsf{false} & \text{otherwise} \end{cases}$$