

Automated Verification of Concurrent Programs

CountDownLatch Mechanism Case Study

Student: Nguyen Thanh Toan - A0154686R

Supervisor: Assoc. Prof. Chin Wei Ngan

National University of Singapore

November 3, 2017

Everything is multi-core



How to reason about concurrent programs?

Owicki-Gries approach

$$\frac{\{P_1\} \ C_1 \ \{Q_1\} \quad \{P_2\} \ C_2 \ \{Q_2\}}{\{P_1 \wedge P_2\} C_1 \parallel C_2 \ \{Q_1 \wedge Q_2\}}$$

if the proofs $\{P_1\} \ C_1 \ \{Q_1\}$ and $\{P_2\} \ C_2 \ \{Q_2\}$ are **interference free**.

Owicki-Gries approach

$$\frac{\{P_1\} C_1 \{Q_1\} \quad \{P_2\} C_2 \{Q_2\}}{\{P_1 \wedge P_2\} C_1 \parallel C_2 \{Q_1 \wedge Q_2\}}$$

if the proofs $\{P_1\} C_1 \{Q_1\}$ and $\{P_2\} C_2 \{Q_2\}$ are **interference free**.

Example:

- ▶ Program: $\{x = 0\} x := x + 1 \parallel x := x + 2 \{x = 3\}$

Owicki-Gries approach

$$\frac{\{P_1\} C_1 \{Q_1\} \quad \{P_2\} C_2 \{Q_2\}}{\{P_1 \wedge P_2\} C_1 \parallel C_2 \{Q_1 \wedge Q_2\}}$$

if the proofs $\{P_1\} C_1 \{Q_1\}$ and $\{P_2\} C_2 \{Q_2\}$ are **interference free**.

Example:

- ▶ Program: $\{x = 0\} x := x + 1 \parallel x := x + 2 \{x = 3\}$
- ▶ $P_1 : (x = 0 \vee x = 2)$ and $Q_1 : (x = 1 \vee x = 3)$
- ▶ $P_2 : (x = 0 \vee x = 1)$ and $Q_2 : (x = 2 \vee x = 3)$

Owicki-Gries approach

$$\frac{\{P_1\} C_1 \{Q_1\} \quad \{P_2\} C_2 \{Q_2\}}{\{P_1 \wedge P_2\} C_1 \parallel C_2 \{Q_1 \wedge Q_2\}}$$

if the proofs $\{P_1\} C_1 \{Q_1\}$ and $\{P_2\} C_2 \{Q_2\}$ are **interference free**.

Example:

- ▶ Program: $\{x = 0\} x := x + 1 \parallel x := x + 2 \{x = 3\}$
- ▶ $P_1 : (x = 0 \vee x = 2)$ and $Q_1 : (x = 1 \vee x = 3)$
- ▶ $P_2 : (x = 0 \vee x = 1)$ and $Q_2 : (x = 2 \vee x = 3)$

Need to prove:

- ▶ $\{P_1\} x := x + 1 \{Q_1\}$
- ▶ $\{P_2\} x := x + 2 \{Q_2\}$

Owicki-Gries approach

$$\frac{\{P_1\} C_1 \{Q_1\} \quad \{P_2\} C_2 \{Q_2\}}{\{P_1 \wedge P_2\} C_1 \parallel C_2 \{Q_1 \wedge Q_2\}}$$

if the proofs $\{P_1\} C_1 \{Q_1\}$ and $\{P_2\} C_2 \{Q_2\}$ are **interference free**.

Example:

- ▶ Program: $\{x = 0\} x := x + 1 \parallel x := x + 2 \{x = 3\}$
- ▶ $P_1 : (x = 0 \vee x = 2)$ and $Q_1 : (x = 1 \vee x = 3)$
- ▶ $P_2 : (x = 0 \vee x = 1)$ and $Q_2 : (x = 2 \vee x = 3)$

Need to prove:

- ▶ $\{P_1\} x := x + 1 \{Q_1\}$
- ▶ $\{P_2\} x := x + 2 \{Q_2\}$
- ▶ $\{P_1 \wedge P_2\} x := x + 2 \{P_1\}$ and $\{P_2 \wedge P_1\} x := x + 1 \{P_2\}$
- ▶ $\{Q_1 \wedge P_2\} x := x + 2 \{Q_1\}$ and $\{Q_2 \wedge P_1\} x := x + 1 \{Q_2\}$

Rely Guarantee Reasoning (P,R,G,Q)

- ▶ Precondition P and Postcondition Q .
- ▶ The rely condition R models all the atomic actions of the environment, describing the interference the program can tolerate from its environment.
- ▶ The guarantee condition G models the atomic actions of the program, and hence describing the interference that it imposes on the other threads of the system.

Rely Guarantee Reasoning (P,R,G,Q)

- ▶ Precondition Q and Postcondition Q.
- ▶ The rely condition R models all the atomic actions of the environment, describing the interference the program can tolerate from its environment.
- ▶ The guarantee condition G models the atomic actions of the program, and hence describing the interference that it imposes on the other threads of the system.

$$\frac{C_1 \models (p_1, R_1, G_1, q_1) \quad C_2 \models (p_2, R_2, G_2, p_2)}{C_1 \parallel C_2 \models (p_1 \wedge p_2, R_1 \wedge R_2, G_1 \vee G_2, q_1 \wedge q_2)}$$

Rely Guarantee Example

Program: $\{x = 0\} \ x := x + 1 \ \parallel \ x := x + 2 \ \{x = 3\}$

Rely Guarantee Example

Program: $\{x = 0\} \ x := x + 1 \parallel x := x + 2 \ \{x = 3\}$

$$\begin{aligned} x := x + 1 \models & (x = 0 \vee x = 2, \\ & (x = 0 \wedge x' = 2) \vee (x = 1 \wedge x' = 3), \\ & (x = 0 \wedge x' = 1) \vee (x = 2 \wedge x' = 3), \\ & (x = 0 \wedge x' = 1) \vee (x = 2 \wedge x' = 3)) \end{aligned}$$

Rely Guarantee Example

Program: $\{x = 0\} \ x := x + 1 \parallel x := x + 2 \ \{x = 3\}$

$$\begin{aligned} x := x + 1 \models & (x = 0 \vee x = 2, \\ & (x = 0 \wedge x' = 2) \vee (x = 1 \wedge x' = 3), \\ & (x = 0 \wedge x' = 1) \vee (x = 2 \wedge x' = 3), \\ & (x = 0 \wedge x' = 1) \vee (x = 2 \wedge x' = 3)) \end{aligned}$$

$$\begin{aligned} x := x + 2 \models & (x = 0 \vee x = 1, \\ & (x = 0 \wedge x' = 1) \vee (x = 2 \wedge x' = 3), \\ & (x = 0 \wedge x' = 2) \vee (x = 1 \wedge x' = 3), \\ & (x = 0 \wedge x' = 2) \vee (x = 1 \wedge x' = 3)) \end{aligned}$$

Concurrent Separation Logic

- Interference-free concurrency

$$\frac{\{P_1\} \ C_1 \ \{Q_1\}}{\{P_1 * P_2\} \ C_1 \ || \ C_2 \ \{Q_1 * Q_2\}}$$

Concurrent Separation Logic

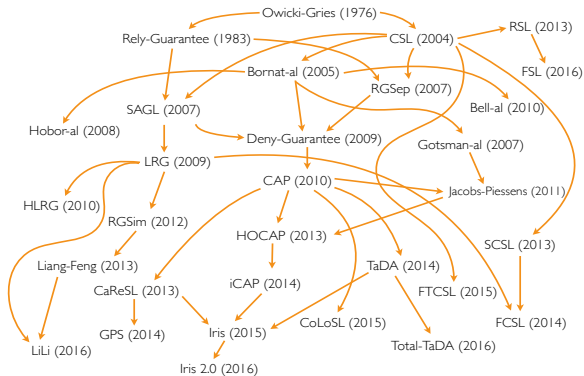
- Interference-free concurrency

$$\frac{\{P_1\} \ C_1 \ \{Q_1\}}{\{P_1 * P_2\} \ C_1 \ || \ C_2 \ \{Q_1 * Q_2\}}$$

- A resource invariant RI_r is associated with each resource. Acquiring the resource imports the resource invariant in the local scope.

$$\frac{\{(P * RI_r) \wedge S\} \ C \ \{Q * RI_r\}}{\{P\} \ \text{with } r \ \text{when } S \ \text{do } C \ \{Q\}}$$

CSL tree



Automated tools for CSL?

- ▶ Great successes of sequential separation logic tools, e.g. Infer
- ▶ Only 2 CSL tools appear *this year*, Caper and Starling.

Automated tools for CSL?

- ▶ Great successes of sequential separation logic tools, e.g. Infer
- ▶ Only 2 CSL tools appear *this year*, Caper and Starling.
- ▶ Our contributions:
 - ▶ The first formal verification for `CountDownLatch` by using abstract predicates.
 - ▶ A modular solution to the counter mechanism by combining a *thread-local* abstraction and the *global* view.
 - ▶ Ensure race-freedom and deadlock-freedom.
 - ▶ An automated prototype which can verify a library implementation of `CountDownLatch`.

Motivation Example

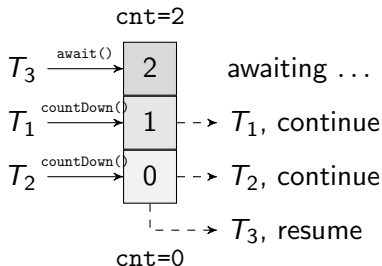
```
c = create_latch(2);
```

$$\left(\begin{array}{l} \# T_1 \\ \# h := h_s \cos \frac{\theta}{2}; \\ \text{countDown}(c); \end{array} \parallel \begin{array}{l} \# T_2 \\ \# r := h_s \sin \frac{\theta}{2}; \\ \text{countDown}(c); \end{array} \parallel \begin{array}{l} \# T_3 \\ \text{await}(c); \\ \# V := \frac{1}{3} \pi r^2 h; \end{array} \right)$$

Motivation Example

```
c = create_latch(2);
```

$$\left(\begin{array}{l} \# T_1 \\ \# h := h_s \cos \frac{\theta}{2}; \\ \text{countDown}(c); \end{array} \parallel \begin{array}{l} \# T_2 \\ \# r := h_s \sin \frac{\theta}{2}; \\ \text{countDown}(c); \end{array} \parallel \begin{array}{l} \# T_3 \\ \text{await}(c); \\ \# V := \frac{1}{3} \pi r^2 h; \end{array} \right)$$



Motivation Example

```
c = create_latch(1);
```

$$\left(\begin{array}{l} \# \text{ send } P*Q \\ \text{countDown}(c); \\ \dots \end{array} \parallel \begin{array}{l} \text{await}(c); \\ \# \text{ receive } P \\ \dots \end{array} \parallel \begin{array}{l} \text{await}(c); \\ \# \text{ receive } Q \\ \dots \end{array} \right)$$

Motivation Example

```
c = create_latch(1);
```

$$\left(\begin{array}{l} \# \text{ send } P*Q \\ \text{countDown}(c); \\ \dots \end{array} \parallel \begin{array}{l} \text{await}(c); \\ \# \text{ receive } P \\ \dots \end{array} \parallel \begin{array}{l} \text{await}(c); \\ \# \text{ receive } Q \\ \dots \end{array} \right)$$

```
c = create_latch(2);
```

$$\left(\begin{array}{l} \dots \\ \# \text{ owns } P \\ \text{countDown}(c); \text{ await}(c); \\ \# \text{ owns } Q \\ \dots \end{array} \parallel \begin{array}{l} \dots \\ \# \text{ owns } Q \\ \text{countDown}(c); \text{ await}(c); \\ \# \text{ owns } P \\ \dots \end{array} \right)$$

Resource predicates

```
CountDownLatch create_latch(n) with P
  requires n>0
  ensures LatchIn(res,P)*LatchOut(res,P)*CNT(res,n);
  requires n=0
  ensures CNT(res,-1);
```

Resource Predicates

```
void countDown(CountDownLatch i)
  requires LatchIn(i,P)*P*CNT(i,n) $\wedge$ n>0
  ensures CNT(i,n-1);
  requires CNT(i,-1)
  ensures CNT(i,-1);
```

```
void await(CountDownLatch i)
  requires LatchOut(i,P)*CNT(i,0)
  ensures P*CNT(i,-1);
  requires CNT(i,-1)
  ensures CNT(i,-1);
```


Normalization rules

NORM-1 : $\text{CNT}(c, n) * \text{CNT}(c, -1) \wedge n \leq 0 \longrightarrow \text{CNT}(c, -1)$

NORM-2 : $\text{CNT}(c, n1) * \text{CNT}(c, n2) \wedge n = n1 + n2 \wedge n1, n2 \geq 0 \longrightarrow \text{CNT}(c, n)$

NORM-3 : $\text{LatchOut}(c, P) * \text{CNT}(c, -1) \longrightarrow \text{CNT}(c, -1) * P$

Normalization rules

[NORM-1] : $\text{CNT}(c, n) * \text{CNT}(c, -1) \wedge n \leq 0 \longrightarrow \text{CNT}(c, -1)$

[NORM-2] : $\text{CNT}(c, n1) * \text{CNT}(c, n2) \wedge n = n1 + n2 \wedge n1, n2 \geq 0 \longrightarrow \text{CNT}(c, n)$

[NORM-3] : $\text{LatchOut}(c, P) * \text{CNT}(c, -1) \longrightarrow \text{CNT}(c, -1) * P$

[SPLIT-1] : $\text{LatchOut}(i, P * Q) \longrightarrow \text{LatchOut}(i, P) * \text{LatchOut}(i, Q)$

[SPLIT-2] : $\text{LatchIn}(i, P * Q) \longrightarrow \text{LatchIn}(i, P) * \text{LatchIn}(i, Q)$

[SPLIT-3] : $\text{CNT}(c, n) \wedge n1, n2 \geq 0 \wedge n = n1 + n2 \longrightarrow \text{CNT}(c, n1) * \text{CNT}(c, n2)$

Race Error Case

[ERR-1]: $\text{LatchIn}(c, P) * \text{CNT}(c, -1) \longrightarrow \text{RACE-ERROR}$

Race Error Case

[ERR-1]: LatchIn(c,P)*CNT(c,-1) \longrightarrow RACE-ERROR

c = create_latch(1) with P*Q;

LatchOut(c,P*Q)*LatchIn(c,P*Q)*CNT(c,1)

LatchOut(c,P*Q)*LatchIn(c,P)*LatchIn(c,Q)*CNT(c,0)*CNT(c,1)*CNT(c,0)

$\left(\begin{array}{l} \dots \\ \# \text{LatchOut}(c, P*Q)*\text{CNT}(c, 0) \\ \text{await}(c); \\ \# P*Q*\text{CNT}(c, -1) \\ \dots \text{use } P*Q \dots \end{array} \right)$	\parallel	$\left(\begin{array}{l} \dots \text{create } P \dots \\ \# P*\text{LatchIn}(c, P)*\text{CNT}(c, 1) \\ \text{countDown}(c); \\ \# \text{CNT}(c, 0) \\ \dots \end{array} \right)$	\parallel	$\left(\begin{array}{l} \dots \text{create } Q \dots \\ \# Q*\text{LatchIn}(c, Q)*\text{CNT}(c, 0) \\ \text{skip}(); \\ \# Q*\text{LatchIn}(c, Q)*\text{CNT}(c, 0) \\ \dots \end{array} \right)$
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

P*Q*CNT(c,-1) *CNT(c,0) *Q*LatchIn(c,Q)*CNT(c,0)

P*Q*CNT(c,-1) *Q*LatchIn(c,Q)

RACE-ERROR detected by [ERR-1]

Deadlock

ERR-2: $\text{CNT}(c, a) * \text{CNT}(c, -1) \wedge a > 0 \longrightarrow \text{DEADLOCK-ERROR}$

Deadlock

[ERR-2]: $\text{CNT}(c, a) * \text{CNT}(c, -1) \wedge a > 0 \longrightarrow \text{DEADLOCK-ERROR}$

```
c = create_latch(2);  
# CNT(c, 2)  $\longrightarrow$  CNT(c, 2) * CNT(c, 0)
```

```
 $\left( \begin{array}{l} \text{\# CNT}(c, 2) \\ \text{countDown}(c); \\ \text{\# CNT}(c, 1) \end{array} \parallel \begin{array}{l} \text{\# CNT}(c, 0) \\ \text{await}(c); \\ \text{\# CNT}(c, -1) \end{array} \right);$ 
```

```
    # CNT(c, 1) * CNT(c, -1)  
# DEADLOCK-ERROR detected by [ERR-2]
```

Inter-latch Deadlock

WAIT-1: $\text{WAIT}(S) \wedge \neg \text{isCyclic}(S) \longrightarrow \text{WAIT}(\{\})$

WAIT-2: $\text{CNT}(c_1, a) * \text{CNT}(c_2, -1) * \text{WAIT}(S) \wedge a > 0 \longrightarrow$
 $\text{CNT}(c_1, a) * \text{CNT}(c_2, -1) \wedge a > 0 * \text{WAIT}(S \cup \{c_2 \rightarrow c_1\})$

WAIT-3: $\text{WAIT}(S_1) * \text{WAIT}(S_2) \longleftrightarrow \text{WAIT}(S_1 \cup S_2)$

ERR-3: $\text{WAIT}(S) \wedge \text{isCyclic}(S) \longrightarrow \text{DEADLOCK-ERROR}$

Inter-latch Deadlock

```
c1 = create_latch(1); c2 = create_latch(1);  
# WAIT{} * CNT(c1,1) * CNT(c2,1)  $\implies$   
# CNT(c1,1) * CNT(c2,0) * CNT(c2,1) * CNT(c1,0)
```

$\left(\begin{array}{l} \# \text{CNT}(c1,1)*\text{CNT}(c2,0) \\ \text{await}(c2); \\ \# \text{CNT}(c1,1)*\text{CNT}(c2,-1) \\ \# * \text{WAIT}(\{c2 \rightarrow c1\}) \\ \text{countDown}(c1); \\ \# \text{CNT}(c1,0)*\text{CNT}(c2,-1) \\ \# * \text{WAIT}(\{c2 \rightarrow c1\}) \end{array} \right $	$\begin{array}{l} \# \text{CNT}(c2,1)*\text{CNT}(c1,0) \\ \text{await}(c1); \\ \# \text{CNT}(c2,1)*\text{CNT}(c1,-1) \\ \# * \text{WAIT}(\{c1 \rightarrow c2\}) \\ \text{countDown}(c2); \\ \# \text{CNT}(c2,0)*\text{CNT}(c1,-1) \\ \# * \text{WAIT}(\{c1 \rightarrow c2\}) \end{array} \right)$
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

```
# CNT(c1,-1)*CNT(c2,-1) * WAIT({c2  $\rightarrow$  c1, c1  $\rightarrow$  c2})  
# DEADLOCK-ERROR detected by [ERR-3]
```


Interpretations for Abstract Predicates

$$\text{LatchOut}(i, P) \hat{=} \boxed{i \mapsto 0} \longrightarrow P$$

$$\text{LatchIn}(i, P) \hat{=} (P \multimap \text{emp}) * [\text{DEC}] * \boxed{i \mapsto m} \wedge m > 0$$

$$\text{DEC} : \boxed{i \mapsto n} \wedge n > 0 \rightsquigarrow \boxed{i \mapsto n-1}$$

Interpretations for Abstract Predicates

$$\text{LatchOut}(i, P) \hat{=} \boxed{i \mapsto 0} \longrightarrow P$$

$$\text{LatchIn}(i, P) \hat{=} (P \multimap \text{emp}) * [\text{DEC}] * \boxed{i \mapsto m} \wedge m > 0$$

$$\text{DEC} : \boxed{i \mapsto n} \wedge n > 0 \rightsquigarrow \boxed{i \mapsto n-1} \quad \{i \mapsto n\} \longrightarrow \boxed{i \mapsto m} \wedge m \geq n \geq 0$$

$$\{i \mapsto n\} \wedge a, b \geq 0 \wedge n = a + b \longleftrightarrow \{i \mapsto a\} * \{i \mapsto b\}$$

$$\boxed{i \mapsto a} * \boxed{i \mapsto b} \longleftrightarrow \boxed{i \mapsto a} \wedge a = b$$

Interpretations for Abstract Predicates

$$\text{LatchOut}(i, P) \hat{=} \boxed{i \mapsto 0} \longrightarrow P$$

$$\text{LatchIn}(i, P) \hat{=} (P \multimap \text{emp}) * [\text{DEC}] * \boxed{i \mapsto m} \wedge m > 0$$

$$\text{DEC} : \boxed{i \mapsto n} \wedge n > 0 \rightsquigarrow \boxed{i \mapsto n-1} \quad \{i \mapsto n\} \longrightarrow \boxed{i \mapsto m} \wedge m \geq n \geq 0$$

$$\{i \mapsto n\} \wedge a, b \geq 0 \wedge n = a + b \longleftrightarrow \{i \mapsto a\} * \{i \mapsto b\}$$

$$\boxed{i \mapsto a} * \boxed{i \mapsto b} \longleftrightarrow \boxed{i \mapsto a} \wedge a = b$$

$$\text{CNT}(i, n) \hat{=} \{i \mapsto n\} \wedge n \geq 0 \vee \boxed{i \mapsto 0} \wedge n = -1$$

$Prog ::= \overline{data} \ \overline{proc}$
 $datat ::= \mathbf{data} \ C \ \{ \overline{t} \ \overline{f} \}$
 $proc ::= t \ pn(\overline{t} \ \overline{v}) \ \overline{spec} \{ e \}$
 $spec ::= \text{requires } \Phi_{pr} \ \text{ensure } \Phi_{po};$
 $t ::= \text{void} \mid \text{int} \mid \text{bool} \mid \text{CountDownLatch} \mid C$
 $e ::= v \mid v.f \mid k \mid \text{new } C(\overline{v}) \mid e_1; e_2 \mid e_1 \parallel e_2 \mid \langle e \rangle$
 $\quad \mathbf{create_latch}(n) \ \mathbf{with} \ \kappa \wedge \pi \mid \mathbf{countDown}(n)$
 $\quad \mathbf{await}(v) \mid pn(\overline{v}) \mid \text{if } v \text{ then } e_1 \text{ else } e_2 \mid \dots$

Core Language Example

```
data CDL{}
data cell{int val;}
pred_prim LatchIn{-%P@Split }<>
pred_prim LatchOut{+%P@Split }<>
pred_prim CNT<n:int>
  inv n >= (-1)

lemma "split" self::CNT<n> & a>=0 & b>=0 & n=a+b
  -> self::CNT<a> * self::CNT<b>;

// Normalization lemmas
lemma_prop "idemp-CNT" self::CNT<a> * self::CNT<(-1)>
  -> self::CNT<(-1)>;

lemma_prop "combine-CNT" self::CNT<a> * self::CNT<b> & a,b>=0
  -> self::CNT<a+b>;
```

Core Language Example

```
void main()
  requires emp ensures emp;
{
  cell h, r;
  int v;
  CDL c = create_latch(2) with h'::cell<1> * r'::cell<2> * @full[h, r];
  par h, r, v, c@L
  {
    case h, c@L c'::LatchIn- h'::cell<1> * @full[h]<> * c'::CNT<(1)> ->
      h = new cell(1);
      countDown(c);
  ||
    case r, c@L c'::LatchIn- r'::cell<2> * @full[r]<> * c'::CNT<(1)> ->
      r = new cell(2);
      countDown(c);
  ||
    case v, c@L c'::LatchOut+ h'::cell<1> * r'::cell<2> * @full[h, r]<> * c'::CNT<0> ->
      await(c);
      v = h.val + r.val;
  }
  assert h'::cell<1> * r'::cell<2> & v' = 3;
}
```

Core Specification Language

FA Pred.	$rpred$	$::=$	$\text{pred } R(\text{self}, \bar{v}, \bar{v}) [\equiv \Phi] [\text{inv } \pi]$
Action	act	$::=$	$\text{action } I \equiv \overline{\iota_1 \wedge \pi_1} \rightsquigarrow \iota_2 \wedge \pi_2$
Disj. formula	Φ	$::=$	$\bigvee (\exists \bar{v} \cdot \eta * \kappa \wedge \pi)$
Non-Resource	η	$::=$	$[I]_\xi \mid \text{WAIT}(\overline{v_1 \rightarrow v_2})_\xi \mid \boxed{v \mapsto C(\bar{v})} \mid \eta_1 * \eta_2$
Sep. formula	κ	$::=$	$\iota \mid V \mid R(v, \Phi_f, \bar{v}) \mid \kappa_1 * \kappa_2$
Simple heap	ι	$::=$	$\text{emp} \mid v \mapsto C(\bar{v}) \mid \{ \{ v \mapsto C(\bar{v}) \} \} \mid \iota_1 * \iota_2$
Perms	ξ	$::=$	$\epsilon \mid 1$
Pure formula	π	$::=$	$\alpha \mid \pi_1 \wedge \pi_2 \mid \pi_1 \vee \pi_2 \mid \neg \pi \mid \exists v \cdot \pi \mid \forall v \cdot \pi$
Arith. formula	α	$::=$	$\alpha_1^t = \alpha_2^t \mid \alpha_1^t \neq \alpha_2^t \mid \alpha_1^t < \alpha_2^t \mid \alpha_1^t \leq \alpha_2^t$
Arith. term	α^t	$::=$	$k \mid v \mid k \times \alpha^t \mid \alpha_1^t + \alpha_2^t \mid -\alpha^t$

$k \in \text{integer constants}$ $v \in \text{variables}, \bar{v} \equiv v_1, \dots, v_n$ $C \in \text{data names}$
 $V \in \text{resource variables}$ $R \in \text{resource pred. names}$ $\epsilon \in (0, 1]$

Let's watch the demo

Conclusions

- ▶ Formal verification of `CountDownLatch` mechanism
- ▶ Detect deadlock and race cases
- ▶ Implement an automated verifier for `CountDownLatch` programs

Future work

- ▶ Verify fundamental concurrent algorithms: locks, etc.
- ▶ Compare with state-of-the-art tools Caper and Starling
- ▶ Build on the recent work of “tree share” model
- ▶ Build on recent logics such as Views, Iris, and LiLi

Thank you for your attention!

Questions ?