# 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 1, 2017

# Everything is multi-core


## How to reason about concurrent programs?

# Owicki-Gries Approach

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

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

# Owicki-Gries Approach

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

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

- Interference freedom: every assertion used in the local verification is shown not invalidated by the execution of the other processes.

# Owicki-Gries Approach Example

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

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

- Program: $\{x = 0\}\ x := x + 1 \mid\mid x := x + 2\ \{x = 3\}$
- $P_1 : (x = 0 \vee x = 2)$ 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 Example

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

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

- Program: $\{x = 0\}\ x := x + 1 \ ||\ x := x + 2\ \{x = 3\}$
- $P_1 : (x = 0 \vee x = 2)$ 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 Example

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

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

- Program: $\{x = 0\}\ x := x + 1 \mid\mid x := x + 2\ \{x = 3\}$
- $P_1 : (x = 0 \vee x = 2)$ 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 decribing the interference that it imposes on the other threads of the system.

# 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 decribing the interference that it imposes on the other threads of the system.

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

($\dagger$): The precondition $P_i$ is stable under rely condition $R_i$.

# Rely Guarantee Example

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

Program: $\{x = 0\}\ x := x + 1 \mid\mid x := x + 2\ \{x = 3\}$
New form: $x := x + 1 \mid\mid x := x + 2 \quad \models (x = 0, x' = x, \mathit{True}, x' = 3)$
with **True** denotes the action that changes the state arbitrarily.

# Rely Guarantee Example

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

Program: $\{x = 0\}\ x := x + 1 \;||\; x := x + 2\ \{x = 3\}$
New form: $x := x + 1 \;||\; x := x + 2 \quad \models (x = 0, x' = x, \mathit{True}, x' = 3)$
with **True** denotes the action that changes the state arbitrarily.

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

# Rely Guarantee Example

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

Program: $\{x = 0\}\ x := x + 1 \mid\mid x := x + 2\ \{x = 3\}$
New form: $x := x + 1 \mid\mid x := x + 2 \quad \models (x = 0, x' = x, \mathit{True}, x' = 3)$
with **True** denotes the action that changes the state arbitrarily.

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

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

# Concurrent Separation Logic - CSL

- Interference-free concurrency

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

# Concurrent Separation Logic - CSL

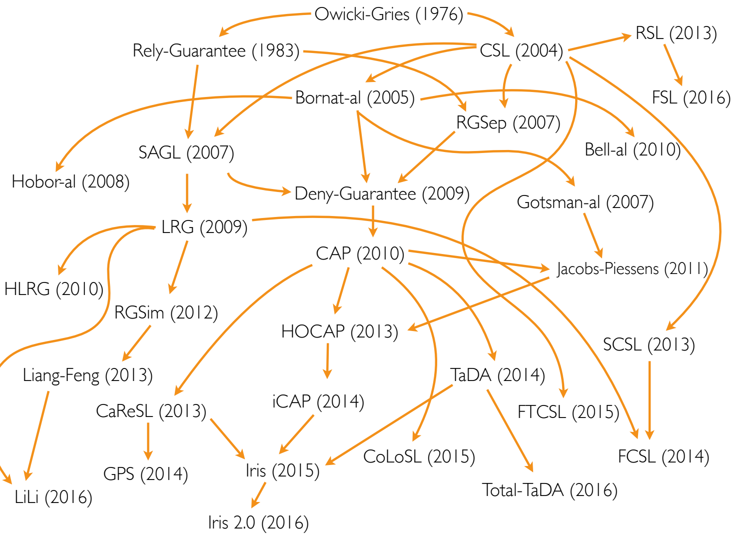
- Interference-free concurrency

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

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

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

# CSL tree



Owicki-Gries (1976)
Rely-Guarantee (1983)
CSL (2004)
RSL (2013)
FSL (2016)
Bornat-al (2005)
RGSep (2007)
Bell-al (2010)
SAGL (2007)
Hobor-al (2008)
Deny-Guarantee (2009)
Gotsman-al (2007)
LRG (2009)
CAP (2010)
Jacobs-Piessens (2011)
HLRG (2010)
RGSim (2012)
HOCAP (2013)
SCSL (2013)
Liang-Feng (2013)
TaDA (2014)
CaReSL (2013)
iCAP (2014)
FTCSL (2015)
CoLoSL (2015)
FCSL (2014)
GPS (2014)
Iris (2015)
Total-TaDA (2016)
LiLi (2016)
Iris 2.0 (2016)

# Automated tools for CSL?

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

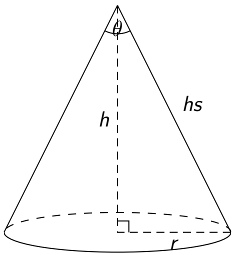
# Automated tools for CSL?

- Great successes of sequential separation logic tools, e.g. Infer of Facebook, and SLAyer of Microsoft.
- Only 2 CSL tools appear *this year*, Caper and Starling.
- Our contributions:
  - The first formal verification for `CountDownLatch` by using abstract predicates.
  - Ensure race-freedom and deadlock-freedom.
  - An automated prototype which can verify a library implemenation of `CountDownLatch`.

# Motivation Example

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

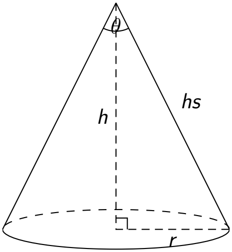
$$\left(\begin{array}{l|l|l} \texttt{\# } {\color{blue}T_1} & \texttt{\# } {\color{blue}T_2} & \texttt{\# } {\color{blue}T_3} \\ \texttt{h=hs*cosine(}\theta\texttt{/2);} & \texttt{r=hs*sine(}\theta\texttt{/2);} & \\ \texttt{countDown(c);} & \texttt{countDown(c);} & \texttt{await(c);} \\ & & \texttt{V=(1/3)*}\pi\texttt{*power(r,2)*h;} \end{array}\right)$$

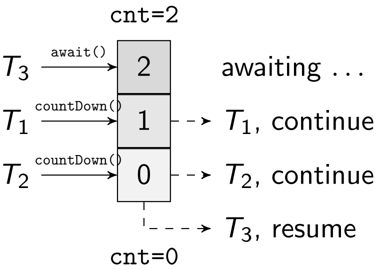


# Motivation Example

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

$$
\left(
\begin{array}{l}
\texttt{\# } T_1 \\
\texttt{h=hs*cosine(}\theta\texttt{/2);} \\
\texttt{countDown(c);} \\
\\
\end{array}
\;\middle\|\;
\begin{array}{l}
\texttt{\# } T_2 \\
\texttt{r=hs*sine(}\theta\texttt{/2);} \\
\texttt{countDown(c);} \\
\\
\end{array}
\;\middle\|\;
\begin{array}{l}
\texttt{\# } T_3 \\
\\
\texttt{await(c);} \\
\texttt{V=(1/3)*}\pi\texttt{*power(r,2)*h;}
\end{array}
\right)
$$



# Motivation Example

- Resources can be split and sent to multiple receiving threads.

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

$$\left( \begin{array}{l|l|l} \texttt{\# send P*Q} & \texttt{await(c);} & \texttt{await(c);} \\ \texttt{countDown(c);} & \texttt{\# receive P} & \texttt{\# receive Q} \\ \texttt{...} & \texttt{...} & \texttt{...} \end{array} \right)$$

# Motivation Example

- Model the barrier synchronization.

$$c = \texttt{create\_latch(2);}$$

$$\left( \begin{array}{l} \texttt{...} \\ \texttt{\# owns P} \\ \texttt{countDown(c); await(c);} \\ \texttt{\# owns Q} \\ \texttt{...} \end{array} \middle\| \begin{array}{l} \texttt{...} \\ \texttt{\# owns Q} \\ \texttt{countDown(c); await(c);} \\ \texttt{\# owns P} \\ \texttt{...} \end{array} \right)$$

# 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)∧n>0
 ensures CNT(i,n-1);
 requires CNT(i,-1)
 ensures CNT(i,-1);
```

# Resource Predicates

```
void countDown(CountDownLatch i)
 requires LatchIn(i,P)*P*CNT(i,n)∧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);
```

# Splitting Lemmas

$$[\underline{\textsf{SPLIT}-1}] : \texttt{LatchOut}(\texttt{i}, \texttt{P*Q}) \longrightarrow \texttt{LatchOut}(\texttt{i}, \texttt{P})\texttt{*}\texttt{LatchOut}(\texttt{i}, \texttt{Q})$$
$$[\underline{\textsf{SPLIT}-2}] : \texttt{LatchIn}(\texttt{i}, \texttt{P*Q}) \longrightarrow \texttt{LatchIn}(\texttt{i}, \texttt{P})\texttt{*}\texttt{LatchIn}(\texttt{i}, \texttt{Q})$$
$$[\underline{\textsf{SPLIT}-3}] : \texttt{CNT}(\texttt{c}, \texttt{n}) \wedge \texttt{n1,n2} \geq 0 \wedge \texttt{n} = \texttt{n1+n2} \longrightarrow \texttt{CNT}(\texttt{c}, \texttt{n1})\texttt{*}\texttt{CNT}(\texttt{c}, \texttt{n2})$$

# Splitting Lemmas

$$[\textsf{SPLIT}-1] : \texttt{LatchOut}(\texttt{i},\texttt{P*Q}) \longrightarrow \texttt{LatchOut}(\texttt{i},\texttt{P})\texttt{*LatchOut}(\texttt{i},\texttt{Q})$$
$$[\textsf{SPLIT}-2] : \texttt{LatchIn}(\texttt{i},\texttt{P*Q}) \longrightarrow \texttt{LatchIn}(\texttt{i},\texttt{P})\texttt{*LatchIn}(\texttt{i},\texttt{Q})$$
$$[\textsf{SPLIT}-3] : \texttt{CNT}(\texttt{c},\texttt{n}) \wedge \texttt{n1,n2}\geq 0 \wedge \texttt{n}=\texttt{n1+n2} \longrightarrow \texttt{CNT}(\texttt{c},\texttt{n1})\texttt{*CNT}(\texttt{c},\texttt{n2})$$

```
c = create_latch(2) with P*Q;
# LatchOut(c,P*Q)*LatchIn(c,P*Q)*CNT(c,2)
# LatchOut(c,P*Q)*LatchIn(c,P)*LatchIn(c,Q)*CNT(c,0)*CNT(c,1)*CNT(c,1)
```

| | | |
|---|---|---|
| ... | ...create P... | ...create Q... |
| #LatchOut(c,P*Q)*CNT(c,0) | #P*LatchIn(c,P)*CNT(c,1) | #Q*LatchIn(c,Q)*CNT(c,1) |
| await(c); | countDown(c); | countDown(c); |
| # P*Q*CNT(c,-1) | # CNT(c,0) | # CNT(c,0) |
| ...use P*Q... | ... | ... |

# Race Condition

$$\texttt{c = create\_latch(1) with } \color{blue}{\texttt{P*Q}}\texttt{;}$$

$$\left(\begin{array}{l} \cdots \\ \texttt{await(c);} \\ \cdots\texttt{use P*Q}\cdots \end{array} \middle\| \begin{array}{l} \cdots\texttt{create P}\cdots \\ \texttt{countDown(c);} \\ \cdots \end{array} \middle\| \begin{array}{l} \cdots\texttt{create Q}\cdots \\ \texttt{skip();} \\ \cdots \end{array}\right)$$

- The first thread uses `Q` while it's not ready for use.

# Race Error

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

# Race Error

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

```
c = create_latch(1) with P*Q;
```

\# $\texttt{LatchOut}(c, P*Q)*\texttt{LatchIn}(c, P*Q)*\texttt{CNT}(c, 1)$

\# $\texttt{LatchOut}(c, P*Q)*\texttt{LatchIn}(c, P)*\texttt{LatchIn}(c, Q)*\texttt{CNT}(c, 0)*\texttt{CNT}(c, 1)*\texttt{CNT}(c, 0)$

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

\# $P*Q*\texttt{CNT}(c, -1)\ *\texttt{CNT}(c, 0)\ *Q*\texttt{LatchIn}(c, Q)*\texttt{CNT}(c, 0)$

\# $P*Q*\texttt{CNT}(c, -1)\ *Q*\texttt{LatchIn}(c, Q)$

\# RACE-ERROR detected by [**ERR−1**]

# Race Error

$$\texttt{c = create\_latch(1) with } \color{blue}{\texttt{P*Q}}\texttt{;}$$
$$\texttt{\# } \color{blue}{\texttt{LatchOut}(c,\texttt{P*Q})*\texttt{LatchIn}(c,\texttt{P*Q})*\texttt{CNT}(c,1)}$$
$$\texttt{\# } \color{blue}{\texttt{LatchOut}(c,\texttt{P*Q})*\texttt{LatchIn}(c,\texttt{P})*\texttt{LatchIn}(c,\texttt{Q})*\texttt{CNT}(c,0)*\texttt{CNT}(c,1)*\texttt{CNT}(c,0)}$$

$$\left(\begin{array}{l|l|l} \cdots & \cdots\texttt{create P}\cdots & \cdots\texttt{create Q}\cdots \\ \#\,\color{blue}{\texttt{LatchOut}(c,\texttt{P*Q})*\texttt{CNT}(c,0)} & \#\,\color{blue}{\texttt{P*LatchIn}(c,\texttt{P})*\texttt{CNT}(c,1)} & \#\,\color{blue}{\texttt{Q*LatchIn}(c,\texttt{Q})*\texttt{CNT}(c,0)} \\ \texttt{await(c);} & \texttt{countDown(c);} & \texttt{countDown(c);} \\ \#\,\color{blue}{\texttt{P*Q*CNT}(c,-1)} & \#\,\color{blue}{\texttt{CNT}(c,0)} & \#\ \color{red}{\texttt{RACE-ERROR}} \end{array}\right)$$

- The precondition of `countDown` cannot be `CNT(c,0)`

# Race Error

$$\texttt{c = create\_latch(2) with } \color{blue}{\texttt{P}*\texttt{Q}}\texttt{;}$$

$$\texttt{\#}\ \color{blue}{\texttt{LatchOut}(c, \texttt{P}*\texttt{Q})*\texttt{LatchIn}(c, \texttt{P}*\texttt{Q})*\texttt{CNT}(c, 2)}$$

$$\texttt{\#}\ \color{blue}{\texttt{LatchOut}(c, \texttt{P}*\texttt{Q})*\texttt{LatchIn}(c, \texttt{P})*\texttt{LatchIn}(c, \texttt{Q})*\texttt{CNT}(c, 0)*\texttt{CNT}(c, 1)*\texttt{CNT}(c, 1)}$$

$$\left(\begin{array}{l|l|l}
\cdots & \cdots\texttt{create P}\cdots & \cdots\texttt{create Q}\cdots \\
\texttt{\#}\,\color{blue}{\texttt{LatchOut}(c, \texttt{P}*\texttt{Q})*\texttt{CNT}(c, 0)} & \texttt{\#}\,\color{blue}{\texttt{P}*\texttt{LatchIn}(c, \texttt{P})*\texttt{CNT}(c, 1)} & \texttt{\#}\,\color{blue}{\texttt{Q}*\texttt{LatchIn}(c, \texttt{Q})*\texttt{CNT}(c, 1)} \\
\texttt{await(c);} & \texttt{countDown(c);} & \texttt{countDown(c);} \\
\texttt{\#}\,\color{blue}{\texttt{P}*\texttt{Q}*\texttt{CNT}(c, -1)} & \texttt{\#}\,\color{blue}{\texttt{CNT}(c, 0)} & \texttt{\#}\,\color{blue}{\texttt{CNT}(c, 0)} \\
\cdots\texttt{use P*Q}\cdots & \cdots & \cdots
\end{array}\right)$$

$$\texttt{\#}\ \color{blue}{\texttt{P}*\texttt{Q}*\texttt{CNT}(c, -1)\ *\texttt{CNT}(c, 0)\ *\texttt{CNT}(c, 0)}$$

$$\texttt{\#}\ \color{blue}{\texttt{P}*\texttt{Q}*\texttt{CNT}(c, -1)\ *\texttt{CNT}(c, 0)}$$

$$\texttt{\#}\ \color{blue}{\texttt{P}*\texttt{Q}*\texttt{CNT}(c, -1)}$$

# Race Error

$$\texttt{c = create\_latch(2) with } P{*}Q;$$
$$\#\ \texttt{LatchOut}(c, P{*}Q){*}\texttt{LatchIn}(c, P{*}Q){*}\texttt{CNT}(c, 2)$$
$$\#\ \texttt{LatchOut}(c, P{*}Q){*}\texttt{LatchIn}(c, P){*}\texttt{LatchIn}(c, Q){*}\texttt{CNT}(c, 0){*}\texttt{CNT}(c, 1){*}\texttt{CNT}(c, 1)$$

$$\left(\begin{array}{l||l||l} \cdots & \cdots\texttt{create P}\cdots & \cdots\texttt{create Q}\cdots \\ \#\,\texttt{LatchOut}(c, P{*}Q){*}\texttt{CNT}(c, 0) & \#\,P{*}\texttt{LatchIn}(c, P){*}\texttt{CNT}(c, 1) & \#\,Q{*}\texttt{LatchIn}(c, Q){*}\texttt{CNT}(c, 1) \\ \texttt{await(c);} & \texttt{countDown(c);} & \texttt{countDown(c);} \\ \#\,P{*}Q{*}\texttt{CNT}(c, -1) & \#\,\texttt{CNT}(c, 0) & \#\,\texttt{CNT}(c, 0) \\ \cdots\texttt{use P*Q}\cdots & \cdots & \cdots \end{array}\right)$$

$$\#\ P{*}Q{*}\texttt{CNT}(c, -1)\ {*}\texttt{CNT}(c, 0)\ {*}\texttt{CNT}(c, 0)$$
$$\#\ P{*}Q{*}\texttt{CNT}(c, -1)\ {*}\texttt{CNT}(c, 0)$$
$$\#\ P{*}Q{*}\texttt{CNT}(c, -1)$$

$$[\underline{\textsf{NORM}-1}] : \texttt{CNT}(c, n){*}\texttt{CNT}(c, -1) \wedge n \leq 0 \longrightarrow \texttt{CNT}(c, -1)$$
$$[\underline{\textsf{NORM}-2}] : \texttt{CNT}(c, n1){*}\texttt{CNT}(c, n2) \wedge n{=}n1{+}n2 \wedge n1, n2 \geq 0 \longrightarrow \texttt{CNT}(c, n)$$

# Deadlock

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

$\Big($ `countDown(c);` $\Big\|$ `await(c);` $\Big)$ ;

# Deadlock

```
c = create_latch(2);
( countDown(c);  ||  await(c);) ;
```

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

# Deadlock

```
c = create_latch(2);
( countDown(c); || await(c);) ;
```

[ERR−2]: CNT(c,a)*CNT(c,-1)∧a>0 ⟶ DEADLOCK-ERROR

```
c = create_latch(2);
# CNT(c,2) ⟶ CNT(c,2)*CNT(c,0)

( # CNT(c,2)         || # CNT(c,0)   ) ;
( countDown(c);      || await(c);    )
( # CNT(c,1)         || # CNT(c,-1)  )

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

# Inter-latch Deadlock

```
c1 = create_latch(1); c2 = create_latch(1);
```

$$\left(\begin{array}{l} \texttt{await(c2);} \\ \texttt{countDown(c1);} \end{array} \;\middle\|\; \begin{array}{l} \texttt{await(c1);} \\ \texttt{countDown(c2);} \end{array}\right)$$

# Inter-latch Deadlock

```
c1 = create_latch(1); c2 = create_latch(1);
```

$$\left( \begin{array}{l} \texttt{await(c2);} \\ \texttt{countDown(c1);} \end{array} \middle\| \begin{array}{l} \texttt{await(c1);} \\ \texttt{countDown(c2);} \end{array} \right)$$

$\underline{[\textbf{WAIT}-\textbf{1}]}$: $\texttt{WAIT}(\texttt{S}) \wedge \neg\texttt{isCyclic(S)} \longrightarrow \texttt{WAIT}(\{\})$

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

$\underline{[\textbf{WAIT}-\textbf{3}]}$: $\texttt{WAIT}(S_1) * \texttt{WAIT}(S_2) \longleftrightarrow \texttt{WAIT}((S_1 \cup S_2))$

$\underline{[\textbf{ERR}-\textbf{3}]}$: $\texttt{WAIT}(\texttt{S}) \wedge \texttt{isCyclic(S)} \longrightarrow \texttt{DEADLOCK-ERROR}$

# Inter-latch Deadlock

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

| | |
|---|---|
| # CNT(c1,1)*CNT(c2,0) | # CNT(c2,1)*CNT(c1,0) |
| await(c2); | await(c1); |
| # CNT(c1,1)*CNT(c2,-1) | # CNT(c2,1)*CNT(c1,-1) |
| # * WAIT({c2 → c1}) | # * WAIT({c1 → c2}) |
| countDown(c1); | countDown(c2); |
| # CNT(c1,0)*CNT(c2,-1) | # CNT(c2,0)*CNT(c1,-1) |
| # * WAIT({c2 → c1}) | # * WAIT({c1 → c2}) |

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

# Implementation

**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: spin lock, ticket lock, CAS, etc.
- Build on recent logics such as Views, Iris, and LiLi
- Compare with state-of-the-art tools Caper and Starling

# Future work

- Verify fundamental concurrent algorithms: spin lock, ticket lock, CAS, etc.
- Build on recent logics such as Views, Iris, and LiLi
- Compare with state-of-the-art tools Caper and Starling
- Build on the recent work of "tree share" model

# Thank you for your attention!

**Questions ?**

# Interference free

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

# Interference free

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

# Interpretations for Abstract Predicates

$$\mathtt{LatchOut}(\mathtt{i},\mathtt{P}) \quad \widehat{=} \quad \boxed{\mathtt{i}\mapsto\mathtt{0}} \longrightarrow \mathtt{P}$$

$$\mathtt{LatchIn}(\mathtt{i},\mathtt{P}) \quad \widehat{=} \quad (\mathtt{P} \mathbin{-\!\circledast} \mathtt{emp}) * [\mathtt{DEC}] * \boxed{\mathtt{i}\mapsto\mathtt{m}} \wedge \mathtt{m}>0$$

$$\mathtt{DEC}: \quad \boxed{\mathtt{i}\mapsto\mathtt{n}} \wedge \mathtt{n}>0 \quad \leadsto \boxed{\mathtt{i}\mapsto\mathtt{n-1}}$$

# Interpretations for Abstract Predicates

$$\texttt{LatchOut}(\texttt{i},\texttt{P}) \quad \widehat{=} \quad \boxed{\texttt{i}\mapsto\texttt{0}} \longrightarrow \texttt{P}$$

$$\texttt{LatchIn}(\texttt{i},\texttt{P}) \quad \widehat{=} \quad (\texttt{P} -\!\circledast \texttt{emp}) * [\texttt{DEC}] * \boxed{\texttt{i}\mapsto\texttt{m}} \wedge \texttt{m>0}$$

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

$$\{\!|\texttt{i}\mapsto\texttt{n}|\!\} \longrightarrow \boxed{\texttt{i}\mapsto\texttt{m}} \wedge \texttt{m}\geq\texttt{n}\geq\texttt{0}$$

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

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

# Interpretations for Abstract Predicates

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

$$\mathtt{LatchIn(i,P)} \;\hat{=}\; (\mathtt{P} -\!\!\circledast \mathtt{emp}) * [\mathtt{DEC}] * \boxed{\mathtt{i}\mapsto\mathtt{m}} \wedge \mathtt{m>0}$$

$$\mathtt{DEC}: \;\boxed{\mathtt{i}\mapsto\mathtt{n}} \wedge \mathtt{n>0} \;\leadsto\; \boxed{\mathtt{i}\mapsto\mathtt{n-1}}$$

$$\{\!|\mathtt{i}\mapsto\mathtt{n}|\!\} \longrightarrow \boxed{\mathtt{i}\mapsto\mathtt{m}} \wedge \mathtt{m}\geq\mathtt{n}\geq\mathtt{0}$$

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

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

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

# Core Language

$$
\begin{array}{rcl}
\mathit{Prog} & ::= & \overline{\mathit{data}}\ \ \overline{\texttt{proc}} \\
\texttt{datat} & ::= & \textbf{data}\ \texttt{C}\ \{\overline{t\ \ f}\} \\
\texttt{proc} & ::= & t\ \ pn(\overline{t\ \ v})\ \ \overline{spec}\{\ e\ \} \\
\texttt{spec} & ::= & \texttt{requires}\ \ \Phi_{pr}\ \ \texttt{ensure}\ \ \Phi po; \\
\texttt{t} & ::= & \texttt{void} \mid \texttt{int} \mid \texttt{bool} \mid \texttt{CountDownLatch} \mid \texttt{C} \\
\texttt{e} & ::= & \texttt{v} \mid \texttt{v.f} \mid \texttt{k} \mid \texttt{new C(}\overline{v}\texttt{)} \mid e_1\,\texttt{;}\,e_2 \mid e_1\,\texttt{||}\,e_2 \mid \texttt{<e>} \\
 & & \textbf{create\_latch}\texttt{(n)}\ \ \textbf{with}\ \kappa \wedge \pi \mid \textbf{coutnDown}(n) \\
 & & \textbf{await}(v) \mid pn(\overline{v}) \mid \texttt{if}\ v\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \mid \ldots
\end{array}
$$

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

$k \in$ *integer constants* $\quad \texttt{v} \in$ *variables*, $\bar{\texttt{v}} \equiv \texttt{v}_1, .., \texttt{v}_n \quad \texttt{C} \in$ *data names*

$\texttt{V} \in$ *resource variables* $\quad \texttt{R} \in$ *resource pred. names* $\quad \epsilon \in (0, 1]$