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

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

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

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

Owicki-Gries approach

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

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

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

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

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

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

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

Owicki-Gries approach

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

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

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

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

Rely Guarantee Example

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

Rely Guarantee Example

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

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

Rely Guarantee Example

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

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

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

Concurrent Separation Logic - CSL

- Interfence-free concurrency

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

Concurrent Separation Logic - CSL

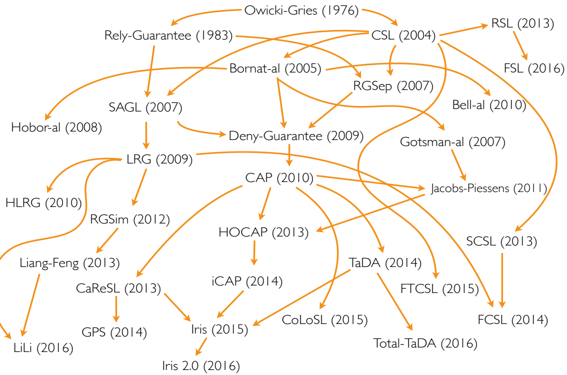
- Interfence-free concurrency

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

- 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)
GPS (2014)
Iris (2015)
CoLoSL (2015)
FCSL (2014)
LiLi (2016)
Total-TaDA (2016)
Iris 2.0 (2016)

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 implemenation of `CountDownLatch`.

Motivation Example

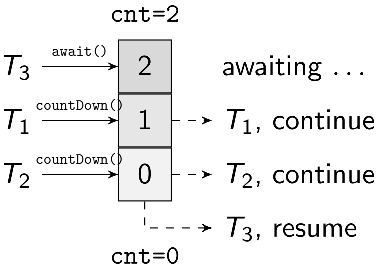
```
c = create_latch(2);
```

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

Motivation Example

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

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



Motivation Example

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

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

Motivation Example

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

```
( # send P*Q      ||  await(c);    ||  await(c);
  countDown(c);   ||  # receive P  ||  # receive Q
  ...             ||  ...          ||  ...        )
```

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

```
( ...                           ||  ...
  # owns P                      ||  # owns Q
  countDown(c); await(c);       ||  countDown(c); await(c);
  # owns Q                      ||  # owns P
  ...                           ||  ...                     )
```

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);
```

```
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

$$[\underline{\textsf{NORM}-1}] : \mathtt{CNT}(\mathtt{c},\mathtt{n}) * \mathtt{CNT}(\mathtt{c},\mathtt{-1}) \wedge \mathtt{n} \leq \mathtt{0} \longrightarrow \mathtt{CNT}(\mathtt{c},\mathtt{-1})$$
$$[\underline{\textsf{NORM}-2}] : \mathtt{CNT}(\mathtt{c},\mathtt{n1}) * \mathtt{CNT}(\mathtt{c},\mathtt{n2}) \wedge \mathtt{n} = \mathtt{n1+n2} \wedge \mathtt{n1,n2} \geq \mathtt{0} \longrightarrow \mathtt{CNT}(\mathtt{c},\mathtt{n})$$
$$[\underline{\textsf{NORM}-3}] : \mathtt{LatchOut}(\mathtt{c},\mathtt{P}) * \mathtt{CNT}(\mathtt{c},\mathtt{-1}) \longrightarrow \mathtt{CNT}(\mathtt{c},\mathtt{-1}) * \mathtt{P}$$

Normalization Rules

$[\underline{\textbf{NORM}-1}]$: $\mathtt{CNT(c,n)*CNT(c,\text{-}1)}\wedge\mathtt{n}\leq\mathtt{0}\longrightarrow\mathtt{CNT(c,\text{-}1)}$
$[\underline{\textbf{NORM}-2}]$: $\mathtt{CNT(c,n1)*CNT(c,n2)}\wedge\mathtt{n}{=}\mathtt{n1+n2}\wedge\mathtt{n1,n2}\geq\mathtt{0}\longrightarrow\mathtt{CNT(c,n)}$
$[\underline{\textbf{NORM}-3}]$: $\mathtt{LatchOut(c,P)*CNT(c,\text{-}1)}\longrightarrow\mathtt{CNT(c,\text{-}1)*P}$

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

Race Error

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

Race Error

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

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

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

$$\texttt{\# P*Q*CNT}(\texttt{c},-1)\ *\texttt{CNT}(\texttt{c},0)\ *\texttt{Q*LatchIn}(\texttt{c},\texttt{Q})*\texttt{CNT}(\texttt{c},0)$$
$$\texttt{\# P*Q*CNT}(\texttt{c},-1)\ *\texttt{Q*LatchIn}(\texttt{c},\texttt{Q})$$
$$\texttt{\# RACE-ERROR detected by } [\underline{\textbf{ERR}-\textbf{1}}]$$

Deadlock

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

Deadlock

[**ERR−2**]: `CNT(c,a)*CNT(c,-1)` $\wedge$ `a>0` $\longrightarrow$ `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

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

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

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

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

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)
```

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

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

Interpretations for Abstract Predicates

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

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

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

Interpretations for Abstract Predicates

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

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

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

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

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

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

Interpretations for Abstract Predicates

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

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

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

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

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

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

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

Core Language

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

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*	$::=$	$\texttt{pred}\ \ \mathtt{R}(\mathtt{self},\bar{\mathtt{V}},\bar{\mathtt{v}})\ [\equiv\ \ \Phi]\ [\textbf{inv}\ \pi]$
Action	*act*	$::=$	$\texttt{action I}\ \equiv\ \ \overline{\iota_1 \wedge \pi_1 \rightsquigarrow \iota_2 \wedge \pi_2}$
Disj. formula	Φ	$::=$	$\bigvee(\exists\,\bar{\mathtt{v}}\cdot\eta{*}\kappa\wedge\pi)$
Non-Resource	η	$::=$	$[\mathtt{I}]_\xi \mid \mathtt{WAIT}(\overline{v_1{\rightarrow}v_2})_\xi \mid \boxed{\mathtt{v}{\mapsto}\mathtt{C}(\bar{\mathtt{v}})} \mid \eta_1{*}\eta_2$
Sep. formula	κ	$::=$	$\iota \mid \mathtt{V} \mid \mathtt{R}(\mathtt{v},\overline{\Phi_{\mathtt{f}}},\bar{\mathtt{v}}) \mid \kappa_1{*}\kappa_2$
Simple heap	ι	$::=$	$\mathtt{emp} \mid \mathtt{v}\mapsto\mathtt{C}(\bar{\mathtt{v}}) \mid \{\!\mid \mathtt{v}{\mapsto}\mathtt{C}(\bar{\mathtt{v}}) \mid\!\} \mid \iota_1{*}\iota_2$
Perms	ξ	$::=$	$\epsilon \mid 1$
Pure formula	π	$::=$	$\alpha \mid \pi_1{\wedge}\pi_2 \mid \pi_1{\vee}\pi_2 \mid \neg\pi \mid \exists\mathtt{v}{\cdot}\pi \mid \forall\mathtt{v}{\cdot}\pi$
Arith. formula	α	$::=$	$\alpha^t_1{=}\alpha^t_2 \mid \alpha^t_1{\neq}\alpha^t_2 \mid \alpha^t_1{<}\alpha^t_2 \mid \alpha^t_1{\leq}\alpha^t_2$
Arith. term	α^t	$::=$	$\mathtt{k} \mid \mathtt{v} \mid \mathtt{k}\times\alpha^t \mid \alpha^t_1+\alpha^t_2 \mid -\alpha^t$

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

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

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 ?