

# The WISL Memory Model

14th Summer School on Formal Techniques

Menlo College

May, 2025

# What is WISL?

- WISL: While with a Simplified C memory model (block-offset)
  - Uses pointers and pointer arithmetic
  - Support for more language errors (e.g. Use After Free)
  - C-style deallocation
  - Used in Gillian

# The Syntax of WISL

## Values

$$p \in \text{Ptr} \stackrel{\text{def}}{=} \mathcal{L} \times \mathbb{N}$$

$$v \in \text{Val} \supseteq \mathbb{N} \cup \text{Bool} \cup \{\text{null}\} \cup \text{Ptr}$$

## Expressions

$$E, E_1, E_2, \dots \in \text{Exp} \stackrel{\text{def}}{=} v \in \text{Val} \mid x \in \text{PVar} \mid \ominus E \mid E \oplus E$$

## Commands

$$C \in \mathcal{C}_W \stackrel{\text{def}}{=} \text{skip} \mid x := E \mid x := f(\vec{E})$$

$$\mid x := [E] \mid [E] := E \mid x := \text{new}(E) \mid \text{free}(E)$$

$$\mid C; C \mid \text{if } (E) C \text{ else } C \mid \text{while } (E) C$$

Basic

Memory management

Control flow



# The Semantics of WISL: What's new?

## Pointers

- Block-offset pairs:  $(l, n)$
- Allocation creates a new block  $b$  of a given size  $n$  and returns the pair  $(b, 0)$
- Deallocation can free entire blocks only

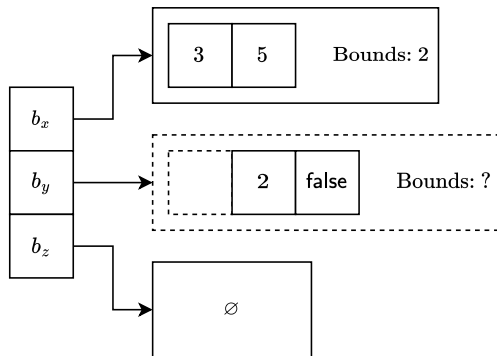
## Partial Memory

$$h : \mathcal{L} \rightarrow_{\text{fin}} ((\mathbb{N} \rightarrow_{\text{fin}} \mathcal{V}_W) \times \mathbb{N}^?)_{\emptyset}$$

# The Memory Model Visualised

## Partial Memory

$$h : \mathcal{L} \rightarrow_{\text{fin}} ((\mathbb{N} \rightarrow_{\text{fin}} \mathcal{V}_W) \times \mathbb{N}^?)_{\emptyset}$$



# A Tractable Assertion Language for Tools

**Pure:**  $P_\pi, Q_\pi \in \mathbb{A}_\pi \stackrel{\text{def}}{=} \text{True} \mid \text{False} \mid E = E \mid E < E \mid \dots \mid \neg P_\pi \mid P_\pi \wedge P_\pi \mid P_\pi \vee P_\pi.$

**Spatial:**  $P_\sigma, Q_\sigma \in \mathbb{A}_\sigma \stackrel{\text{def}}{=} \text{emp} \mid P_\sigma \star P_\sigma \mid E \mapsto E \mid \text{bound}(E, E) \mid E \mapsto \emptyset$

**All:**  $P, Q \in \mathbb{A} : \mathbb{A}_\pi \times \mathbb{A}_\sigma$

## What are the main differences?

- Conjunction and negation of pure formulae only
- No *explicit* existential quantification

# WISL-Specific Assertions

## WISL-Specific Assertions

$$E \mapsto E \mid \text{bound}(E, E) \mid E \mapsto \emptyset$$

- $E_1 \mapsto E_2$  describes the single cell, as in While
- $\text{bound}(E_1, E_2)$  states that  $E_1$  is a block pointer, pointing to a block of length  $E_2$
- $E \mapsto \emptyset$  states that  $E$  is a block pointer and its entire block has been freed

# WISL-Specific Assertions

## WISL-Specific Assertions

$$E \mapsto E \mid \text{bound}(E, E) \mid E \mapsto \emptyset$$

- $E_1 \mapsto E_2$  describes the single cell, as in While
- $\text{bound}(E_1, E_2)$  states that  $E_1$  is a block pointer, pointing to a block of length  $E_2$
- $E \mapsto \emptyset$  states that  $E$  is a block pointer and its entire block has been freed

## Derived Predicates

- $E \mapsto E_1, \dots, E_n$  describes  $n$  consecutive cells in a memory block, as usual
- $E \mapsto_b E_1, \dots, E_n \stackrel{\text{def}}{=} E \mapsto E_1, \dots, E_n \star \text{bound}(E, n)$  describes a complete block of size  $n$  at  $E$



# WISL-Specific Assertions

## WISL-Specific Assertions

$$E \mapsto E \mid \text{bound}(E, E) \mid E \mapsto \emptyset$$

- $E_1 \mapsto E_2$  describes the single cell, as in While
- $\text{bound}(E_1, E_2)$  states that  $E_1$  is a block pointer, pointing to a block of length  $E_2$
- $E \mapsto \emptyset$  states that  $E$  is a block pointer and its entire block has been freed

## Derived Predicates

- $E \mapsto E_1, \dots, E_n$  describes  $n$  consecutive cells in a memory block, as usual
- $E \mapsto_b E_1, \dots, E_n \stackrel{\text{def}}{=} E \mapsto E_1, \dots, E_n \star \text{bound}(E, n)$  describes a complete block of size  $n$  at  $E$

## Some Interesting Properties

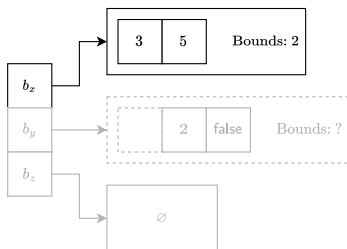
- $\vdash E + m \mapsto - \star \text{bound}(E, E_1) \vdash m < E_1$
- $\vdash E \mapsto_b E_1, \dots, E_n \star E + m \mapsto - \Rightarrow \text{False}$

# WISL-Specific Assertions

## WISL-Specific Assertions

$$E \mapsto E \mid \text{bound}(E, E) \mid E \mapsto \emptyset$$

$$b_x \mapsto_b 3, 5 \quad * \quad b_y + 1 \mapsto 2, \text{false} \quad * \quad b_z \mapsto \emptyset$$

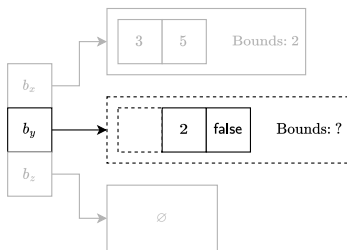


# WISL-Specific Assertions

## WISL-Specific Assertions

$$E \mapsto E \mid \text{bound}(E, E) \mid E \mapsto \emptyset$$

$$b_x \mapsto_b 3, 5 \quad * \quad b_y + 1 \mapsto 2, \text{false} \quad * \quad b_z \mapsto \emptyset$$

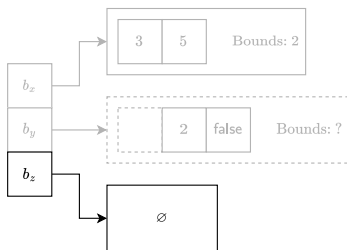


# WISL-Specific Assertions

## WISL-Specific Assertions

$$E \mapsto E \mid \text{bound}(E, E) \mid E \mapsto \emptyset$$

$$b_x \mapsto_b 3, 5 \quad * \quad b_y + 1 \mapsto 2, \text{false} \quad * \quad b_z \mapsto \emptyset$$



# Predicate definitions

$$\begin{aligned} \text{list}(x, \alpha) &\stackrel{\text{def}}{=} \\ &(x = \text{null} \star \alpha = []) \vee \\ &(\exists a, z, \beta. x \mapsto a, z \star \alpha = a:\beta \star \text{list}(z, \beta)) \end{aligned}$$

$$\begin{aligned} \text{lseg}(x, y, \alpha) &\stackrel{\text{def}}{=} \\ &(x = y \star \alpha = \text{null}) \vee \\ &(\exists a, z, \beta. x \mapsto a, z \star \alpha = a:\beta \star \text{lseg}(z, y, \beta)) \end{aligned}$$

# Predicate definitions

$$\begin{aligned} \text{list}(x, \alpha) &\stackrel{\text{def}}{=} \\ &(x = \text{null} \star \alpha = []) \vee \\ &(\exists a, z, \beta. x \mapsto_b a, z \star \alpha = a:\beta \star \text{list}(z, \beta)) \end{aligned}$$

$$\begin{aligned} \text{lseg}(x, y, \alpha) &\stackrel{\text{def}}{=} \\ &(x = y \star \alpha = \text{null}) \vee \\ &(\exists a, z, \beta. x \mapsto_b a, z \star \alpha = a:\beta \star \text{lseg}(z, y, \beta)) \end{aligned}$$

# Predicate definitions

$$\text{list}(x, \alpha) \stackrel{\text{def}}{=} \\ (x = \text{null} \star \alpha = []) \vee \\ (\exists a, z, \beta. x \mapsto_b a, z \star \alpha = a:\beta \star \text{list}(z, \beta))$$

$$\text{lseg}(x, y, \alpha) \stackrel{\text{def}}{=} \\ (x = y \star \alpha = \text{null}) \vee \\ (\exists a, z, \beta. x \mapsto_b a, z \star \alpha = a:\beta \star \text{lseg}(z, y, \beta))$$

```

predicate list(+x, alpha) {
  (x == null) * (alpha == []);
  (x -b> #a, #z) * (alpha == #a :: #beta) * list(#z, #beta)
}

predicate listseg(+x, y, alpha) {
  (x == y) * (alpha == []);
  (x -b> #a, #z) * (alpha == #a :: #beta) * listseg(#z, y, #beta)
}

```

# Verifying WISL in Gillian

list\_length\_rec.wisl

Verify llen | Symbolic-debug llen

**VARIABLES**

- Program store
  - $x = \#x$
- Heap
- Predicates
- Pure formulae
  - $\#alpha == []$
  - $\#x == \text{null}$
- Types
  - $\#alpha = \text{List}$
  - $\#x = \text{Null}$
- Axioms

```

6  { (x = #x) * list(#x, #alpha) }
7  function llen(x) {
8      if (x = null) {
9          n := 0
10     } else {
11         t := [x+1];
12         n := llen(t);
13         n := n + 1
14     };
15     return n
16 }
17 { list(#x, #alpha) * (ret = len(#alpha)) }
18

```

Gillian Debugging

llen

```

graph TD
    llen[llen] --> if{if ((x = null))}
    if -- true --> n0[n := 0]
    n0 --> t1(( ))
    if -- false --> t2(( ))
    style t1 fill:none,stroke:none
    style t2 fill:none,stroke:none
    
```



Stay tuned for the lab tomorrow!