Mutable Store

15-411/15-611 Compiler Design

Seth Copen Goldstein

October 20, 2020

Today

- Pointers
- The Heap and pointers
- Arrays
- Length & bounds checking
- Elaboration of +=, etc.

Adding a pointer

- Extend types

$$\tau ::= \mathsf{int} \mid \mathsf{bool} \mid \tau *$$

- Extend expressions
  - **alloc(τ)**: allocate a heap cell to hold a value of τ
  - ***e**: dereference a pointer to get value at *e*
  - **null**: special null pointer

$$e ::= \ldots \mid \mathsf{alloc}(\tau) \mid *e \mid \mathsf{null}$$

Typing rules

$$\frac{}{\Gamma \vdash \mathsf{alloc}(\tau) : \tau*}$$

- A freshly allocated cell has type “pointer to τ”

$$\frac{\Gamma \vdash e : \tau*}{\Gamma \vdash *e : \tau}$$

- if *e* has type “pointer to τ,” then **e* has type “τ”

$$\frac{}{\Gamma \vdash \mathsf{null} : \ ?}$$

- What type should null have?

The type of null?

- Desired behavior
  - allow any pointer to be compared to null
  - disallow pointer dereference to null

Equality for pointers?

- Can we compare τ* and σ*:
  - if $\tau = \sigma$
  - if $\tau \neq \sigma$
  - What about **`int* p;`** … **`if (p==null)`** …

- null is given type of "any*"
- And, implicitly converted to τ* as needed

The type of null?

- Desired behavior
  - allow any pointer to be compared to null
  - disallow pointer dereference to null
- Using the type “any*” along with subsumption
- Subsumption used for implicit coercion

$$\frac{}{\Gamma \vdash \text{null} : any*} \qquad \frac{\Gamma \vdash e : any*}{\Gamma \vdash e : \tau*}$$

- Have to make sure introducing any* is safe

The type of null?

- Desired behavior
  - allow any pointer to be compared to null
  - disallow pointer dereference to null
- Using the type "any*" along with subsumption
- Subsumption used for implicit coercion

$$\frac{}{\Gamma \vdash \text{null} : any*} \qquad \frac{\Gamma \vdash e : any*}{\Gamma \vdash e : \tau*}$$

- Can't allow **`*null`**

$$\frac{\Gamma \vdash e : \tau* \quad \Gamma \nvdash e : any*}{\Gamma \vdash *e : \tau}$$

Typing rules (revised)

$$\frac{}{\Gamma \vdash \mathsf{alloc}(\tau) : \tau*}$$

- A freshly allocated cell has type “pointer to τ”

$$\frac{\Gamma \vdash e : \tau* \quad \Gamma \not\vdash e : any*}{\Gamma \vdash *e : \tau}$$

- if *e* has type “pointer to τ,” and *e* isn’t null, then **e* has type “τ”

$$\frac{}{\Gamma \vdash \mathrm{null} \ : any*}$$

- null has the indefinite type

$$\frac{\Gamma \vdash e : any*}{\Gamma \vdash e : \tau*}$$

- Implicit coercion

Representing the Heap

Evaluation of expression e in the context of

- a Heap,
- Stack, and
- binding environment.

$$H; S; \eta \vdash e \rhd K$$

- alloc(τ) returns an unused address in H (the heap) which can store a value of τ

What is an address?

- How do we represent addresses, i.e., the result of the alloc operation?
- 64-bits? infinite?
- What happens when we run out of memory? How do we model this in the dynamic semantics?

What is an address?

- How do we represent addresses, i.e., the result of the alloc operation?
- 64-bits? infinite?
- What happens when we run out of memory? How do we model this in the dynamic semantics?
- Assume infinite address space, i.e., an address is in $\mathbb{N}$.
- Out of heap memory with generate an exception: “exception(mem)”

Using *H*

- alloc(τ) returns an address of proper size (or raises an exception)
- *H* must keep track of next free address.

$$H\colon (\mathbb{N} \cup \{\mathrm{next}\}) \to \mathrm{Val}$$

- Extend all old rules with H; which they leave unchanged, e.g.,

$$H; S; \eta \vdash e_1 \oplus e_2 \rhd K \longrightarrow H; S; \eta \vdash e_1 \rhd (\blacksquare \oplus e_2, K)$$

Pointers

- null evaluates to 0

$$H; S; \eta \vdash \text{null} \rhd K \;\longrightarrow H; S; \eta \vdash 0 \rhd K$$

- alloc(τ):
  - returns a fresh address *a*,
  - updates the next address in the heap
  - initializes the location to default for τ

$$H; S; \eta \vdash \text{alloc}(\tau) \rhd K \;\longrightarrow$$

$$H[a \mapsto \text{default}(\tau), \text{next} \mapsto a + |\tau|]; S; \eta \vdash a \rhd K$$
$$a = H(\text{next})$$

$$H; S; \eta \vdash \mathrm{alloc}(\tau) \rhd K \;\longrightarrow$$

$$H[a \mapsto \mathrm{default}(\tau), \mathrm{next} \mapsto a + |\tau|]; S; \eta \vdash a \rhd K$$
$$a = H(\mathrm{next})$$

- $\mathrm{default}(\tau)$: 0 for int, false for bool, null for ptr
- $|\tau|$ for x86-64:
  - |int| = 4
  - |bool|= 4
  - |τ*| = 8

Accessing Memory

- Dereferencing a pointer:

$$H; S; \eta \vdash * \, \mathrm{e} \rhd K \;\longrightarrow H; S; \eta \vdash e \rhd (* \blacksquare, K)$$

Accessing Memory

- Dereferencing a pointer:

$$H; S; \eta \vdash * \, \mathrm{e} \rhd K \;\longrightarrow\; H; S; \eta \vdash e \rhd (* \, \blacksquare, K)$$

- The interesting part:

$$H; S; \eta \vdash a \rhd K \;\longrightarrow\; H; S; \eta \vdash H(a) \rhd K \qquad a \neq 0$$

$$H; S; \eta \vdash a \rhd K \;\longrightarrow\; \mathrm{exception(mem)} \qquad a = 0$$

Writing to the heap

- l-values and r-values
- l-values or destinations:

$$d ::= x \mid *\, d$$

- Typing is the same for all destiniations:

$$\frac{\Gamma \vdash d : \tau \quad \Gamma \vdash e : \tau}{\Gamma \vdash \mathsf{assign}(d, e) : [\tau']}$$

recall, $[\tau']$, is the return type of the function.

Writing to the heap

- Distinguish between variables, *x*, which live on the stack,

$$
\begin{array}{lcl}
H \; ; S \; ; \eta \vdash \mathsf{assign}(x, e) \blacktriangleright K & \longrightarrow & H \; ; S \; ; \eta \vdash e \rhd (\mathsf{assign}(x, \_) \; , K) \\
H \; ; S \; ; \eta \vdash c \rhd (\mathsf{assign}(x, \_) \; , K) & \longrightarrow & H \; ; S \; ; \eta[x \mapsto c] \rhd \mathsf{nop} \blacktriangleright K
\end{array}
$$

Writing to the heap

- Distinguish between variables, *x*, which live on the stack,

$$
\begin{array}{lcl}
H \; ; S \; ; \eta \vdash \mathsf{assign}(x, e) \blacktriangleright K & \longrightarrow & H \; ; S \; ; \eta \vdash e \rhd (\mathsf{assign}(x, \_) \; , K) \\
H \; ; S \; ; \eta \vdash c \rhd (\mathsf{assign}(x, \_) \; , K) & \longrightarrow & H \; ; S \; ; \eta[x \mapsto c] \rhd \mathsf{nop} \blacktriangleright K
\end{array}
$$

- and other destinations which live in the heap.

$$
\begin{array}{lcll}
H \; ; S \; ; \eta \vdash \mathsf{assign}(*d, e) \blacktriangleright K & \longrightarrow & H \; ; S \; ; \eta \vdash d \rhd (\mathsf{assign}(*\_, e) \; , K) & \\
H \; ; S \; ; \eta \vdash a \rhd (\mathsf{assign}(*\_, e) \; , K) & \longrightarrow & H \; ; S \; ; \eta \vdash e \rhd (\mathsf{assign}(*a, \_) \; , K) & \\
H \; ; S \; ; \eta \vdash c \rhd (\mathsf{assign}(*a, \_) \; , K) & \longrightarrow & H[a \mapsto c] \; ; S \; ; \eta \vdash \mathsf{nop} \blacktriangleright K & (a \neq 0) \\
H \; ; S \; ; \eta \vdash c \rhd (\mathsf{assign}(*a, \_) \; , K) & \longrightarrow & \mathsf{exception}(\mathsf{mem}) & (a = 0)
\end{array}
$$

Writing to the heap

- left to right evaluation of address and rval

$$H \; ; S \; ; \eta \vdash \mathsf{assign}(*d, e) \blacktriangleright K \quad \longrightarrow \quad H \; ; S \; ; \eta \vdash d \rhd (\mathsf{assign}(*\_, e) \; , K)$$

$$H \; ; S \; ; \eta \vdash a \rhd (\mathsf{assign}(*\_, e) \; , K) \quad \longrightarrow \quad H \; ; S \; ; \eta \vdash e \rhd (\mathsf{assign}(*a, \_) \; , K)$$

- Then making assignment (if $a \neq 0$)

Writing to the heap

- left to right evaluation of address and rval

$$H \; ; S \; ; \eta \vdash \mathsf{assign}(*d, e) \blacktriangleright K \quad \longrightarrow \quad H \; ; S \; ; \eta \vdash d \rhd (\mathsf{assign}(*\_, e) \; , K)$$

$$H \; ; S \; ; \eta \vdash a \rhd (\mathsf{assign}(*\_, e) \; , K) \quad \longrightarrow \quad H \; ; S \; ; \eta \vdash e \rhd (\mathsf{assign}(*a, \_) \; , K)$$

- Then making assignment (if $a \neq 0$)

$$H \; ; S \; ; \eta \vdash c \rhd (\mathsf{assign}(*a, \_) \; , K) \quad \longrightarrow \quad H[a \mapsto c] \; ; S \; ; \eta \vdash \mathsf{nop} \blacktriangleright K \qquad (a \neq 0)$$

$$H \; ; S \; ; \eta \vdash c \rhd (\mathsf{assign}(*a, \_) \; , K) \quad \longrightarrow \quad \mathsf{exception}(\mathsf{mem}) \qquad (a = 0)$$

Proper eval order

- ```
  int* p = NULL;
  *p = 1/0;
  ```

- ```
  int**p = NULL;
  **p = 1/0;
  ```

Arrays

- Extend types, expressions, and destinations

$$
\begin{array}{lcl}
\tau & ::= & \ldots \mid \tau[\,] \\
e & ::= & \ldots \mid \mathsf{alloc\_array}(\tau, e) \mid e_1[e_2] \\
d & ::= & \ldots \mid d[e]
\end{array}
$$

- Need typing rules for alloc_array and $e_1[e_2]$

$$
\frac{\Gamma \vdash e : \mathsf{int}}{\Gamma \vdash \mathsf{alloc\_array}(\tau, e) : \tau[\,]}
\qquad
\frac{\Gamma \vdash e_1 : \tau[\,] \quad \Gamma \vdash e_2 : \mathsf{int}}{\Gamma \vdash e_1[e_2] : \tau}
$$

Allocating the array

$$H \; ; S \; ; \eta \vdash \mathsf{alloc\_array}(\tau, e) \rhd K$$
$$\longrightarrow \quad H \; ; S \; ; \eta \vdash e \rhd (\mathsf{alloc\_array}(\tau, \_) \; , K)$$

$$H \; ; S \; ; \eta \vdash n \rhd (\mathsf{alloc\_array}(\tau, \_) \; , K)$$

$$\begin{aligned} \longrightarrow \quad & H' \; ; S \; ; \eta \vdash a \rhd K & (n \geq 0) \\ & a = H(\mathsf{next}) \\ & H' = H[a + 0|\tau| \mapsto \mathsf{default}(\tau), \ldots, \\ & \qquad a + (n-1)|\tau| \mapsto \mathsf{default}(\tau), \mathsf{next} \mapsto a + n|\tau|] \end{aligned}$$

$$\longrightarrow \quad \mathsf{exception}(\mathsf{mem}) \qquad (n < 0)$$

Accessing the Array

- left to right evaluation of base address of array and index

$$
\begin{array}{lcl}
H \,;\, S \,;\, \eta \vdash e_1[e_2] \rhd K & \longrightarrow & H \,;\, S \,;\, \eta \vdash e_1 \rhd (\_[e_2]\,,\, K) \\
H \,;\, S \,;\, \eta \vdash a \rhd (\_[e_2]\,,\, K) & \longrightarrow & H \,;\, S \,;\, \eta \vdash e_2 \rhd (a[\_]\,,\, K)
\end{array}
$$

- Then, if in bounds, get the value

$$
H \,;\, S \,;\, \eta \vdash i \rhd (a[\_]\,,\, K) \longrightarrow H \,;\, S \,;\, \eta \vdash H(a + i|\tau|) \rhd K
$$

$$
a \neq 0, 0 \leq i < \mathsf{length}(a), a : \tau[\,]
$$

- Or, generate an exception

$$
\longrightarrow \quad \mathsf{exception}(\mathsf{mem})
$$

$$
a = 0 \text{ or } i < 0 \text{ or } i \geq \mathsf{length}(a)
$$

Accessing the Array

- left to right evaluation of base address of array and index

$$\begin{array}{lcl} H \; ; S \; ; \eta \vdash e_1[e_2] \rhd K & \longrightarrow & H \; ; S \; ; \eta \vdash e_1 \rhd (\_[e_2] \; , K) \\ H \; ; S \; ; \eta \vdash a \rhd (\_[e_2] \; , K) & \longrightarrow & H \; ; S \; ; \eta \vdash e_2 \rhd (a[\_] \; , K) \end{array}$$

- Then, if in bounds, get the value

$$H \; ; S \; ; \eta \vdash i \rhd (a[\_] \; , K) \quad \longrightarrow \quad H \; ; S \; ; \eta \vdash H(a + i\boxed{\tau}) \rhd K$$

$$a \neq 0, 0 \leq i < \text{length}(a), a : \boxed{\tau}[\,]$$

- Or, generate an exception

$$\longrightarrow \quad \text{exception(mem)}$$

$$a = 0 \text{ or } i < 0 \text{ or } i \geq \text{length}(a)$$

Accessing the Array

- left to right evaluation of base address of array and index

$$
\begin{array}{lcl}
H \,;\, S \,;\, \eta \vdash e_1[e_2] \rhd K & \longrightarrow & H \,;\, S \,;\, \eta \vdash e_1 \rhd (\_[e_2] \,,\, K) \\
H \,;\, S \,;\, \eta \vdash a \rhd (\_[e_2] \,,\, K) & \longrightarrow & H \,;\, S \,;\, \eta \vdash e_2 \rhd (a[\_] \,,\, K)
\end{array}
$$

- Then, if in bounds, get the value

$$
H \,;\, S \,;\, \eta \vdash i \rhd (a[\_] \,,\, K) \quad \longrightarrow \quad H \,;\, S \,;\, \eta \vdash H(a + i|\tau|) \rhd K
$$

$$
a \neq 0, 0 \leq i < \boxed{\mathsf{length}(a)}, a : \tau[\,]
$$

- Or, generate an exception

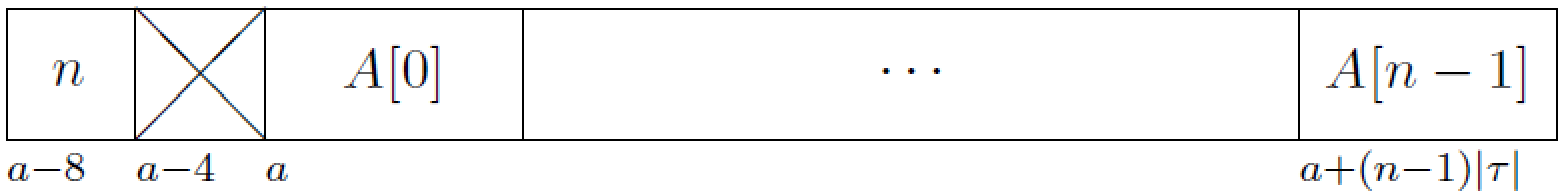
$$
\longrightarrow \quad \mathsf{exception}(\mathsf{mem})
$$

$$
a = 0 \text{ or } i < 0 \text{ or } i \geq \boxed{\mathsf{length}(a)}
$$

recall: alloc_array(τ,e)

Bounds checking

- Must store length in heap.



- Rationale for storing length at a-8?

Writing to the array

$$H \; ; S \; ; \eta \vdash \mathsf{assign}(d[e_2], e_3) \blacktriangleright K \quad \longrightarrow \quad H \; ; S \; ; \eta \vdash d \rhd (\mathsf{assign}(\_[e_2], e_3) \; , K)$$

$$H \; ; S \; ; \eta \vdash a \rhd (\mathsf{assign}(\_[e_2], e_3) \; , K) \quad \longrightarrow \quad H \; ; S \; ; \eta \vdash e_2 \rhd (\mathsf{assign}(a[\_], e_3) \; , K)$$

$$\begin{array}{lll} H \; ; S \; ; \eta \vdash i \rhd (\mathsf{assign}(a[\_], e_3) \; , K) & \longrightarrow & H \; ; S \; ; \eta \vdash e_3 \rhd (\mathsf{assign}(a + i|\tau|, \_) \; , K) \\ & & \qquad a \neq 0, 0 \leq i < \mathsf{length}(a), a : \tau[\,] \\ & \longrightarrow & \mathsf{exception}(\mathsf{mem}) \\ & & \qquad a = 0 \text{ or } i < 0 \text{ or } i \geq \mathsf{length}(a) \end{array}$$

$$H \; ; S \; ; \eta \vdash c \rhd (\mathsf{assign}(b, \_) \; , K) \quad \longrightarrow \quad H[b \mapsto c] \; ; S \; ; \eta \vdash \mathsf{nop} \blacktriangleright K$$

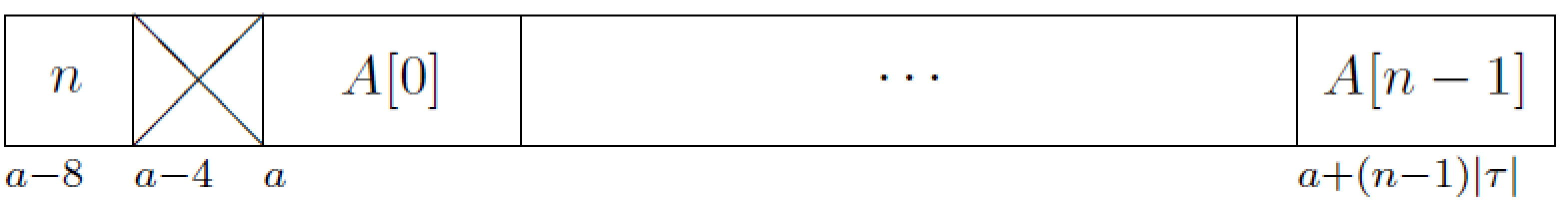
one caveat

$$H \;;\; S \;;\; \eta \vdash \mathsf{assign}(d[e_2], e_3) \blacktriangleright K \quad \longrightarrow \quad H \;;\; S \;;\; \eta \vdash d \rhd (\mathsf{assign}(\_[e_2], e_3) \;,\; K)$$

$$H \;;\; S \;;\; \eta \vdash a \rhd (\mathsf{assign}(\_[e_2], e_3) \;,\; K) \quad \longrightarrow \quad H \;;\; S \;;\; \eta \vdash e_2 \rhd (\mathsf{assign}(a[\_], e_3) \;,\; K)$$

$$\begin{array}{lll} H \;;\; S \;;\; \eta \vdash i \rhd (\mathsf{assign}(a[\_], e_3) \;,\; K) & \longrightarrow & H \;;\; S \;;\; \eta \vdash e_3 \rhd (\mathsf{assign}(a + i|\tau|, \_) \;,\; K) \\ & & \qquad a \neq 0, 0 \leq i < \mathsf{length}(a), a : \tau[\,] \\ & \longrightarrow & \mathsf{exception}(\mathsf{mem}) \\ & & \qquad a = 0 \text{ or } i < 0 \text{ or } i \geq \mathsf{length}(a) \end{array}$$

$$H \;;\; S \;;\; \eta \vdash c \rhd (\mathsf{assign}(b, \_) \;,\; K) \quad \longrightarrow \quad H[b \mapsto c] \;;\; S \;;\; \eta \vdash \mathsf{nop} \blacktriangleright K$$



Code Generation

- For access: $e_1[e_2]$ where $e_1{:}\tau[]$ and $|\tau|=k$

Code Generation

- For access: $e_1[e_2]$ where $e_1{:}\tau[]$ and $|\tau|=k$

$$
\begin{array}{ll}
\mathsf{cogen}(e_1, a) & (a \text{ new}) \\
\mathsf{cogen}(e_2, i) & (i \text{ new}) \\
a_1 \leftarrow a - 8 & \\
t_2 \leftarrow M[a_1] & \\
\mathsf{if}\ (i < 0)\ \mathsf{goto\ error} & \\
\mathsf{if}\ (i \geq t_2)\ \mathsf{goto\ error} & \\
a_3 \leftarrow i * \$k & \\
a_4 \leftarrow a + a_3 & \\
t_5 \leftarrow M[a_4] &
\end{array}
$$

Elaboration

- x = x + e is no longer always valid for x += e

Elaboration

- x = x + e is no longer always valid for x += e
- next time introduce &