Design Specification

Memory Management Tools

Anvil Memory

Author: UniquesKernel

Date: March 31, 2025

Contents

1 Memory Management

Let a program $P \equiv (i_n)_{n\in\mathbb{N}}$ be an ordered sequence of instructions. Let $A \subset P$ denote the set of allocation instructions and $B \subset P$ the deallocation instructions.

Definition 1.1 Memory-safe Program**.**

A program P is *memory-safe* if for all execution paths $\pi \in \Pi(P)$:

1. Exists bijection $T_\pi : A_\pi \to B_\pi$ where:
   - $\forall a_i \in A_\pi,\ T_\pi(a_i) = b_i$ is unique.
   - a_i lexically precedes b_i on π.
2. The deallocation graph $G_\pi = (V_\pi, E_\pi)$ is acyclic.
3. $\text{mem}(a_i) \cap \text{mem}(a_j) = \emptyset$ for $i \neq j$.

1.1 Scope-Centralized Memory Management

Definition 1.2 Memory Arena**.**

A *memory arena* $\mathcal{A} \equiv (\mathcal{M}, F, \Lambda)$ where:

- $\mathcal{M} = \{\mathcal{M}_1, ..., \mathcal{M}_k\}$ - is a finite set of disjoint memory blocks..
- $F \subseteq \bigcup_{i=1}^{k} \mathcal{M}_i$ - is the set of free addresses.
- Λ - is an allocator with allocation strategy σ.

The *allocation function* $H : \mathcal{A} \rightharpoonup \mathcal{A} \times \texttt{addr}(\mathcal{M})$ is:

$$H((\mathcal{M}, F, \Lambda)) = \begin{cases} ((\mathcal{M}, F \setminus \{a\}, \Lambda), a) & \text{if } a \in F \text{ via } \sigma \\ \text{undefined} & \text{otherwise} \end{cases}$$

Subject to invariants:

1. Disjointness: $\mathcal{M}_i \cap \mathcal{M}_j = \emptyset\ \forall i \neq j$.
2. No individual deallocations: $\nexists b_i \in B$ for $a_i \in \mathcal{A}$.
3. Mass reclamation: $\bigcup \mathcal{M}_\pi \subseteq F_\pi^{\text{final}}$ at scope exit.

For control-flow graph G with scope entry/exit nodes G_a, G_b:

- Arena instantiation at G_a creates $\mathcal{A} = (\mathcal{M}, F, \Lambda)$.
- All paths $S_{a\to b}$ allocate via $H((\mathcal{M}, F, \Lambda))$.
- Destruction at G_b enforces $F^{\text{final}} = \bigcup \mathcal{M}$, satisfying Definition 1.1.

This formalism reduces verification complexity from $O(2^{|C|})$ paths to $O(1)$ scope-level invariants. The allocator Λ implements σ through H while maintaining:

$$\forall \pi \in \Pi(P), 1.\ T_\pi \text{ bijection} \Rightarrow F^{\text{final}} = \bigcup \mathcal{M}$$
$$2.\ \text{Lexical precedence} \Rightarrow \text{Scope nesting}$$
$$3.\ \text{Disjointness} \Rightarrow \mathcal{M}_i \cap \mathcal{M}_j = \emptyset$$

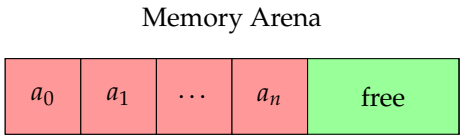

Figure 1: Memory arena structure showing sequential allocations within contiguous blocks.

1.2 Initialization-Centralized Memory Management

We now present a further centralization of memory management through what we shall term the "Defer Pattern," which couples allocation and deallocation at the point of initialization.

Utilizing compiler facilities such as the cleanup attribute in GCC, one may specify:

```
MemoryArena *arena DEFER(memory_arena_destroy) = memory_arena_create(...);
```

This construction guarantees deallocation at scope exit, irrespective of the control flow path that leads to said exit.

The Defer Pattern exhibits the following properties:

1. **Spatial Coupling**: The allocation and its corresponding deallocation are textually adjacent in the source code, providing immediate verification of the appropriate $T_\pi(a_i) = b_i$ relationship.
2. **Compiler-Enforced Temporal Sequencing**: The compiler ensures that deallocation occurs at scope exit, enforcing the lexical ordering requirement that a_i precedes b_i.
3. **Single-Point Specification**: Memory management decisions are made once, at the point of initialization, rather than distributed throughout the control flow graph.

This approach effectively centralizes memory management responsibilities at initialization points, delegating enforcement to the compiler. The programmer need only ensure proper declaration of cleanup behaviors at allocation sites, substantially reducing the cognitive burden of tracking allocation-deallocation pairs across the program's control flow structure.

We thus observe a progression from decentralized path-based verification, to scope-centralized arena management, to initialization-centralized specification—each stage reducing the cognitive distance between allocation and deallocation logic, and consequently reducing the opportunity for memory management errors.

For compiler-enforced cleanup:

$$\underbrace{\text{MemoryArena}^* \mathcal{A}}_{\text{Allocation}} \xrightarrow[\text{Deallocation}]{\text{DEFER}} \mathcal{A} \mapsto (\mathcal{M}, F^{\text{final}}, \Lambda)_{\text{Scope exit}}$$

where the DEFER macro enforces:

$$\mathcal{A} \mapsto (\emptyset, \bigcup \mathcal{M}, \Lambda) \; \forall \pi \in \Pi(P)$$

References