

Theory exercises

Program verification with types and logic (NWI-IMC060)

Week 12

1. This exercise is about linear type systems.

(a) Consider the typing rule for linear function types:

$$\frac{\Gamma, x : A \vdash e : B \quad x \notin \text{dom } \Gamma}{\Gamma \vdash \lambda x. e : A \multimap B}$$

Assume we were to omit the side condition $x \notin \text{dom } \Gamma$ and instead “overwrite” the type of x in Γ . That is, we would use the following modified rule:

$$\frac{(\text{delete } x \Gamma), x : A \vdash e : B}{\Gamma \vdash \lambda x. e : A \multimap B}$$

Does this modified rule preserve the strong type safety theorem of linear type systems? That is, does it (1) forbid use-after-free and double-free errors (2) forbid memory leaks? If yes, explain why (you do not need to give a proof). If no, give a counterexample.

(b) Assume we were to add recursive functions with the following typing rule:

$$\frac{\Gamma, f : A \multimap B, x : A \vdash e : B \quad x \notin \text{dom } \Gamma \quad f \notin \text{dom } \Gamma}{\Gamma \vdash \text{rec } f x. e : A \multimap B}$$

Does this rule preserve our strong type safety theorem? That is, does it (1) forbid use-after-free and double-free errors (2) forbid memory leaks? If yes, explain why (you do not need to give a proof). If no, give a counterexample and explain how you could repair this rule.

2. In this exercise we consider a linear type system with a typing judgment $\Gamma_1 \vdash e : A \dashv \Gamma_2$ with two contexts. Here, Γ_1 is the *pre-typing context* and Γ_2 is the *post-typing context*. The idea is that the post-typing context Γ_2 contains the variables from Γ_1 that are not used by the expression e . Examples of typing rules are:

$$\frac{n \in \mathbb{N}}{\Gamma \vdash n : \mathbf{nat} \dashv \Gamma} \quad \frac{}{\Gamma, x : A \vdash x : A \dashv \Gamma} \quad \frac{\Gamma_1 \vdash e_1 : A \multimap B \dashv \Gamma_2 \quad \Gamma_2 \vdash e_2 : A \dashv \Gamma_3}{\Gamma_1 \vdash e_1 e_2 : B \dashv \Gamma_3}$$

(a) State a theorem that formalizes the expected equivalence between the standard linear typing judgment $\Gamma \vdash e : A$ with a single context (from week 11), and the judgment $\Gamma_1 \vdash e : A \dashv \Gamma_2$ with two contexts.

(b) Give the typing rule for `let $x = e_1$ in e_2` using the typing judgment with two contexts.

- (c) Assume that you have to implement a type checker for a linear type system, *i.e.*, a function/algorithm that given an expression computes its type. Explain why the typing judgment $\Gamma \vdash e : A$ with one context is not directly suitable for implementing a type checking function/algorithm, and how the typing judgment $\Gamma_1 \vdash e : A \dashv \Gamma_2$ with two contexts helps. Clearly describe the type signature of the type checking functions that you consider. (Note: You do not have to worry about other challenging aspects of type checking that also appear when considering ordinary/unrestricted programming languages, like the inference of function types of λ -expressions/types of λ -bound variables.)
- (d) Give the semantic interpretation of the judgment $\Gamma_1 \vdash e : A \dashv \Gamma_2$ in separation logic. You are allowed to use the parallel substitution `subst_map` $\vec{v} e$, and the semantic context typing `ctx_typed` $\Gamma \vec{v}$, where \vec{v} is a finite map from variables names to values.
3. This question is about linear type systems.
- (a) Linear type systems ensure two properties (a) data is not used after it has been deallocated, and (b) all data is eventually deallocated. Explain how linear type systems ensure these two properties.
- (b) Consider the following typing rules for linear references:

$$\frac{\Gamma \vdash e : A}{\Gamma \vdash \text{alloc } e : \mathbf{ref}(A)} \qquad \frac{\Gamma \vdash e : \mathbf{ref}(A)}{\Gamma \vdash \text{free } e : A}$$

$$\frac{\Gamma \vdash e : \mathbf{ref}(A)}{\Gamma \vdash \text{load } e : \mathbf{ref}(\mathbf{moved}) \times A} \qquad \frac{\Gamma_1 \vdash e_1 : \mathbf{ref}(A) \quad \Gamma_2 \vdash e_2 : A}{\Gamma_1 ++ \Gamma_2 \vdash \text{store } e_1 \ e_2 : \mathbf{ref}(A)}$$

There is a bug in the rule for `store`. Explain which of the two aforementioned properties of linear type systems break. Give a counterexample that exhibits the problem.

- (c) Give a correct typing rule for `store`, and explain how it solves the bug.
- (d) Explain the difference between a linear type system and an affine type system.
- (e) When creating a semantic model of a linear type system, we model each type A as a function $A : \text{val} \rightarrow \text{sepProp}$. Give the semantic definition of $\mathbf{ref}(A)$ as such a predicate.