Script generated by TTT

Title: Petter: Virtual Machines (18.06.2019)

Date: Tue Jun 18 10:18:27 CEST 2019

Duration: 86:55 min

Pages: 8

39 Garbage Collection

- Both during execution of a MaMa- as well as a WiM-programs, it may happen that some objects can no longer be reached through references.
- Obviously, they cannot affect the further program execution. Therefore, these objects are called garbage.
- Their storage space should be freed and reused for the creation of other objects.

Caveat

The WiM provides some kind of heap de-allocation. This, however, only frees the storage of failed alternatives !!!

Discussion

- We adopt the C++ perspective on classes and objects.
- We extend our implementation of C. In particular ...
- Classes are considered as extensions of structs. They may comprise:
  - ⇒ attributes, i.e., data fields;
  - ⇒ constructors;
  - ⇒ member functions which either are virtual, i.e., are called depending on the run-time type or non-virtual, i.e., called according to the static type of an object.
  - ⇒ static member functions which are like ordinary functions.
- We ignore visibility restrictions such as **public**, **protected** or **private** but simply assume general visibility.
- We ignore multiple inheritance.

For every class C we assume that we are given an address environment ρ_C.

ρ_C maps every identifier x visible inside C to its decorated relative address a. We distinguish:

global variable	(G, a)
local variable	(L, a)
attribute	(A, a)
virtual function	(V, b)
non-virtual function	(N, a)
static function	(S, a)

For virtual functions x , we do not store the starting address of the code — but the relative address b of the field of x inside the VFT.

Discussion

- Besides storing the current object pointer inside the stack frame, we could have additionally used a specific register COP.
- This register must updated before calls to non-static member functions and restored after the call.
- We have refrained from doing so since
  - → Only some functions are member functions.
  - → We want to reuse as much of the C-machine as possible.

41 Calling Member Functions

Static member functions are considered as ordinary functions.

For non-static member functions, we distinguish two forms of calls:

(1) directly: $f\,(e_2, \ldots, e_n)$

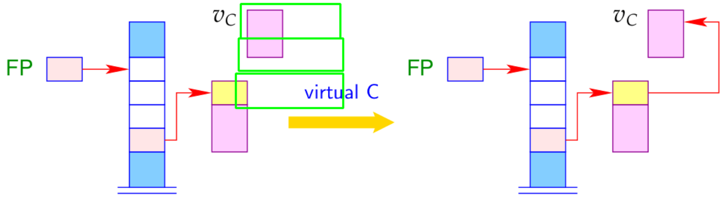
(2) relative to an object: $e_1.f\,(e_2, \ldots, e_n)$

Idea

- The case (1) is considered as an abbreviation of $\mathbf{this}.f\,(e_2, \ldots, e_n)$.
- The object is passed to f as an implicit first argument.
- If f is non-virtual, proceed as with an ordinary call of a function.
- If f is virtual, insert an indirect call via the VFT.

a table with static address v_C.

Then:



$$\text{S[S[FP-3]]} = v_C$$

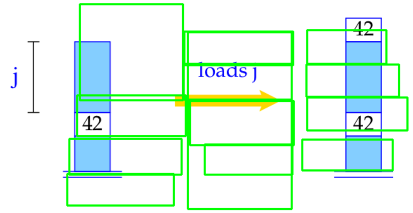
For every class C we assume that we are given an address environment ρ_C.

ρ_C maps every identifier x visible inside C to its decorated relative address a. We distingish:

global variable	(G, a)
local variable	(L, a)
attribute	(A, a)
virtual function	(V, b)
non-virtual function	(N, a)
static function	(S, a)

For virtual functions x, we do not store the starting address of the code — but the relative address b of the field of x inside the VFT.

The instruction loads j loads relative to the stack pointer:



S[SP+1] = S[SP−j];
SP++;