Extending P0 with Explicit Deallocation

By:Baikai Wang, Ziqing Xu, Yujing Chen
April 2020

Introduction

The goal of this project is to extend P0 (for WebAssembly) with heap-allocated objects that can be explicitly deallocated when no longer needed since the P0 compiler can only support variables allocated on the stack or are global. The project is implemented in python while the documentation and testings are done in Jupyter notebook. At this stage, our group accomplished the extensions to P0 grammar and WebAssembly Code Generator.

Result

This is example of one of our test case:

```
program p;
    type a = ^record f : boolean end;
    type b = ^array [1..4] of boolean;
    var x : a;
    var z : b;
 begin
    new(x);
    new(z);
    z[2]^ := true;
    x.f^ := false;
    z[3]^ := x.f^;
    write(z[2]^); {print 1, means true}
    write(z[3]^); {print 0, means false}
    dispose(x)
 end
```

(Left:: P0 source code
Right: generated code in WebAssembly

```
(func $program
i32.const 0
i32.const 8
i32.store
i32.const 4
i32.const 12
i32.store
i32.const 4
i32.load
i32.const 16
i32.add
i32.const 1
i32.store
i32.const 0
i32.load
i32.const 0
i32.store
i32.const 4
i32.load
i32.const 32
i32.add
i32.const 0
i32.load
i32.load
i32.store
i32.const 4
i32.load
i32.const 16
i32.add
i32.load
call $write
i32.const 4
i32.load
i32.const 32
i32.add
i32.load
call $write
i32.const 8
i32.const 0
i32.store
i32.const 0
i32.const 20000
i32.store
)
(memory 1)
(start $program)
```

- The P0 source code is compiled to generated code in WebAssembly. Run the generated code, the result is 1, 0.
- The heap memory is shown as the information of the linked list in the screenshot below. As it is shown, x is deallocated successfully
- More complex test cases are included in the Porject_test.ipynon

```
adr is 8, size is 4, is_alloc: False, nxt:
adr is 12, size is 16, is_alloc: True, nxt:
None
```

Conclusion

Extending P0 with explicit allocation and deallocation is tough but well worth challenging. At this point, P0 is extended with explicit allocation, deallocation and basic mathematical calculation for heap-allocated object. Also, the assignment statement and expression are extended for heap-allocated object. The documented source code, testing and a README are pushed to the git repository.

Implementations

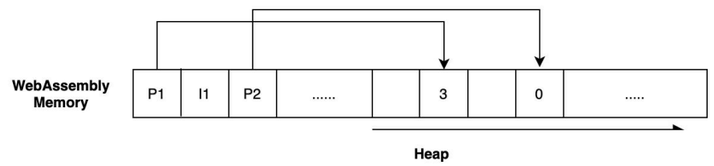
```
type ::= ident |
    "array" "[" expression ".." expression "]" "of" type |
    "record" typedIds {";" typedIds} "end" |
    "^" type
selector ::= {"." ident | "[" expression "]"}["^"].
```

Grammar:

- The right hand side shows which part of the P0 grammar is modified for the purpose of implementing pointer type and accessing heap-allocated object pointed by a pointer.

Pointer Type:

- P0 is extended with pointer type, which is implemented based on the language Pascal. A pointer is defined by prefixing the symbol ^ with the base type.
- The pointer stores the address of a heap-allocated object.



new(p) (Explicit allocation)

- Extend P0 with stdProc New, allocating the object pointed by p on the heap. Parameter p, is a pointer variable.
- Return the address of the heap-allocated object on the heap and the value is assigned to p after calling new(p)
- P0 parser and code generator is modified for implementing 'new(p)``

```
program p;
  type p = array [1..3] of ^integer;
  var v : p;
  begin
     new(v[1]);
     dispose(v[1])
  end
```

(a) Above is the P0 source code for new and dispose

dispose(p) (Explicit deallocation)

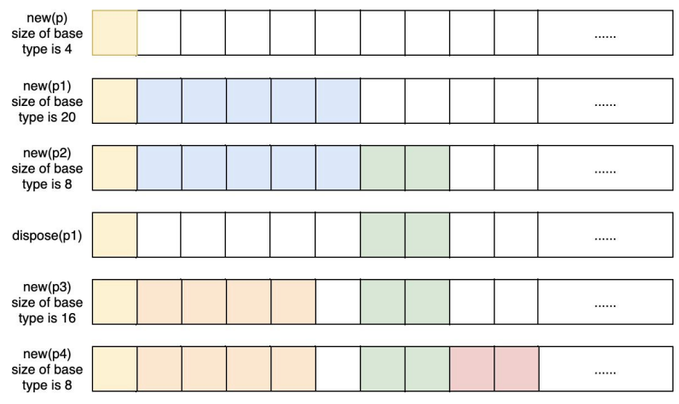
- Extend P0 with stdProc Dispose, deallocating the object pointed by p on the heap.
- Parameter p, pointer variable.
- The memory occupied by heap-allocated object is pointed by p will be freed.
- P0 parser and code generator is modified for this purpose

```
(func $program
i32.const 0   ;; adr of the pointer v
i32.const 12  ;; adr of the allocated object pointed by v
i32.store     ;; store adr of the allocated object
i32.const 12  ;; adr of the allocated object pointed by v
i32.const 0   ;; 0
i32.store     ;; reset the memory block
i32.const 0   ;; adr of the pointer v
i32.const 20000 ;; very large number to ensure no value is set at that adr
i32.store     ;; store to v
)
(memory 1)
(start $program)
)
```

(b) Above is the generated code in WebAssembly, compiled from (a)

Heap Management:

- Heap manager is used to handle the free memory block and the allocated memory block on the heap.
- We are using First-Fit algorithm to allocating object on the heap.



Each square is a block of size 4 on the heap resided in the WebAssembly Memory. The block with color is allocated and the white block is not allocated.

Challenges

Handling the allocation of complex array or record object pointed by a pointer type variable was a challenge in this project. For example, type p = ^array [1..4] of record t : array [1..3] of integer end; var x : p; new(x). Creating many test cases to make sure all the situations are covered is time consuming. For each test case, we create the P0 source code, compile the source code, and run the generated code to ensure the correct output.

Statistics

Lines of coding:

Category	Lines of coding (added)
SC	2
P0	270
ST	6
HM	120
Code Generator	110
Total	508

Number of test cases

Category	Number of Tests
Syntax	8
ST	3
Type Checking	12
HM	14
Code Generator	18
Total	55

Reference

[1] Course material for the P0 compiler
[2] "WebAssembly Specification¶," WebAssembly Specification - WebAssembly 1.0. https://webassembly.github.io/spec/core/index.html.
[3]"Pascal - Pointers," Tutorialspoint. https://www.tutorialspoint.com/pascal/pascal_pointers.htm.
[4]"Pascal - Memory Management," Tutorialspoint. https://www.tutorialspoint.com/pascal/pascal_memory.htm