Codegen #3 : Garbage collector

Горбань Игорь

24 апреля 2018

g.i.b@list.ru

Again

```
$ make -f /usr/class/cs143/assignments/PA5/Makefile
$ wget http://spark-university.s3.amazonaws.com/stanford-compilers/scripts/pa4-grading.pl
$ vim cgen.cc cgen.h
  ...
$ make cgen
$ cp /usr/class/cs143/examples/hello_world.cl .
$ ./mycoolc hello_world.cl
$ /usr/class/cs143/bin/spim hello_world.s
SPIM Version 6.5 of January 4, 2003
Copyright 1990-2003 by James R. Larus (larus@cs.wisc.edu).
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/class/cs143/lib/trap.handler
Hello, World.
COOL program successfully executed
Stats -- #instructions : 154
#reads : 27 #writes 22 #branches 28 #other 77
```

Programm segments, mips types

- **.data** - начало сегмента данных (обычно располагается по адресу 0x10010000)
- **.text** - начало сегмента кода (обычно располагается по адресу 0x00400000)
- **[^:#]*:** - метки
- **#.*** - комментарии

- **.byte 0** - 8 бит
- **.half 0** - 16 бит
- **.word 0** - 32 бита
- **.dword 0** - 64 бита

Registers mappings, syscall.

$0 = $zero

$1 = $at

$2,$3 = $v0,$v1

$4-$7 = $a0-$a3

$8-$15 = $t0-$t7

$16-$23=$s0-$s7

$24,$25 = $t8,$t9

Syscall: значение передается через $v0

- 1,2,3,4 - вывести на экран значение из $a0 (число int/float (**$f12**)/double (**$f12**) или строку)
- 5,6,7,8 - считать значение (число int, float, double или строку) и положить в $v0, $f0, $f0, $a0(аддрес буффера)
- 10 - выход из программы
- 11 - вывести символ $a0
- 12 - считать символ в $a0
- 13,14,15,16 - открыть/читать/писать/закрыть файл.

Memory errors

- Ошибки явного управления памятью
- Ошибки при работе с памятью возникают редко и потому трудно находимы
- Проблема освобождения ресурса — система не знает, как освободить сетевой ресурс или снять блокировку в базе данных
- Необходимо различать уничтожение памяти (уничтожение объекта/ уничтожение путей доступа) и утилизацию памяти (сборка мусора)
- Проблема отслеживания различных путей доступа к структуре (различные указатели на одно и то же место, передача параметром в процедуру и т.д.), поэтому утилизация памяти обычно проблематична

Memory manager (cool-specific)

- Когда создается объект, выделяется память.
- После выделения большого количества памяти - она заканчивается 🤔
- Некоторая память хранит объекты, которые никогда не будут использованы

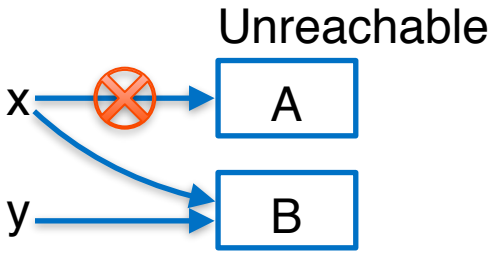
Как узнать, что объект не используется ?

Ex : let x A : <- new A in { x<- y; ...}

Объект x доступен тогда и только тогда:

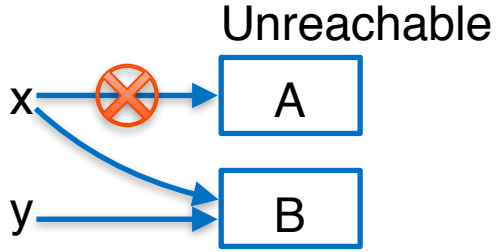
- регистры хранят указатель на x

- другой доступный объект хранит указатель на x



Memory manager #2

Таким образом можно найти все доступные объекты, используя регистры и все указатели. Все недоступные объекты - мусор.



После x<-y:

- первый обьект A - недоступен
- обьект B - доступен (через x)
- поэтому B - не мусор и не удаляется

Ex :
```
x <- new A;
y <- new B;
x <- y;
```

GC
```
if alwaysTrue() then x <- new A
else x.foo() fi;
```

Увы, runtime не ловит такие вещи

без self-modified

Memory manager search

Cool использует accumulator:

- он может указывать на обьект
- этот обьект в свою очередь может указывать на другие обьекты

Так же Cool использует stack pointer:

- stack frame может хранить указатели на обьекты (параметры)
- может хранить *не указатели* на обьекты (return address)
- если знать расположение stack-frame, можно найти указатели внутри них

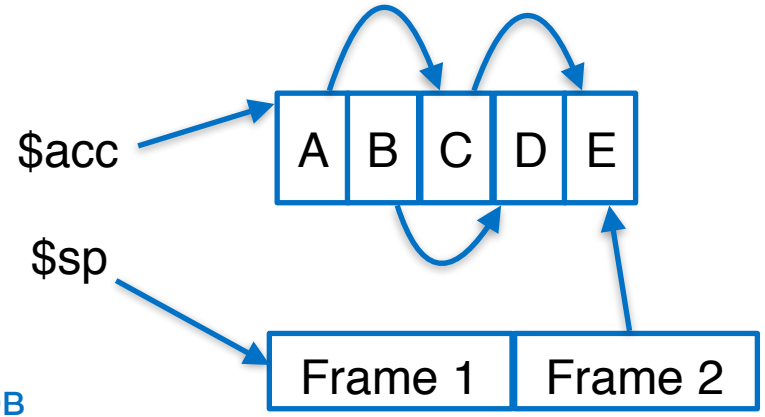
Если runtime знает какой метод вызывается, то он может указать, какие указатели на обьекты находятся в AR (activation record)

Memory manager example

B и D - недоступны из acc и sp, поэтому мы можем освободить их.

Каждая стратегия сборщика мусора делает следующие шаги:

1. Выделяет память, нужную для новых обьектов
2. Когда память кончается:
    A. Рассчитывает, какие обьекты достижимы
    B. Освобождает память, которая не найдена в (A).



Mark and Sweep

Простой алгоритм определения достижимых объектов, «алгоритм пометок» (Mark and Sweep), заключается в следующем:

- для каждого объекта хранится бит, указывающий, достижим ли этот объект из программы или нет;
- изначально все объекты, кроме корневых, помечаются как недостижимые;
- рекурсивно просматриваются и помечаются как достижимые объекты, ещё не помеченные, и до которых можно добраться из корневых объектов по ссылкам;
- те объекты, у которых бит достижимости не был установлен, считаются недостижимыми.

Следует обратить внимание, что, согласно данному алгоритму, если два или более объектов ссылаются друг на друга, но ни на один из этих объектов нет других ссылок, то имеет место циклическая ссылка, и вся группа считается недостижимой.

Stop & copy

Алгоритм работает с двумя областями памяти:

- old space - используется для алокации в программе
- new space - вспомогательная область, используемая при сборке мусора

heap pointer - указывает на следующее пустое слово в old space.

GC - стартует, когда old space - переполняется.

1. Копирование всех достижимых обьектов из old space new space (мусор остается в old space)
2. При копировании в new space содержится меньше обьектов, чем в old до сборки мусора
3. После копирования роли old и new space меняются и так до следующего переполнения

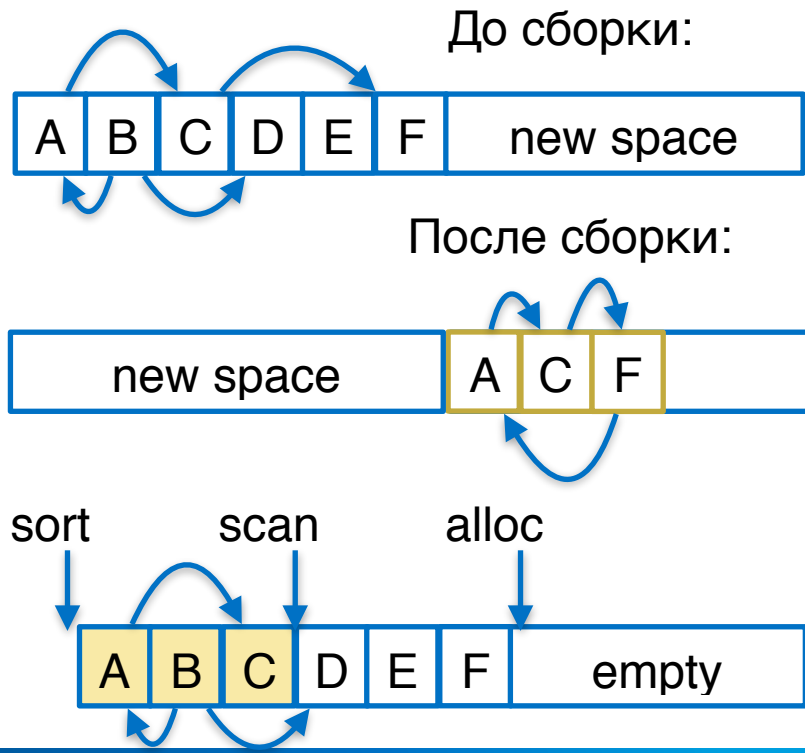
Stop & copy #2

Проблемы:

1. Нужно найти все достижимые обьекты.
2. Нужно скопировать все обьекты и обновить все ссылки в них

Во время копирования мы можем помечать обьекты ссылками на новое положение в памяти(forwarding pointer). Таким образом при повторном попадании на тот же обьект мы будем знать, что он уже скопирован.

Для того, чтобы не хранить таблицу обьектов, которые находятся в процессе - добавляют указатель на регионы.

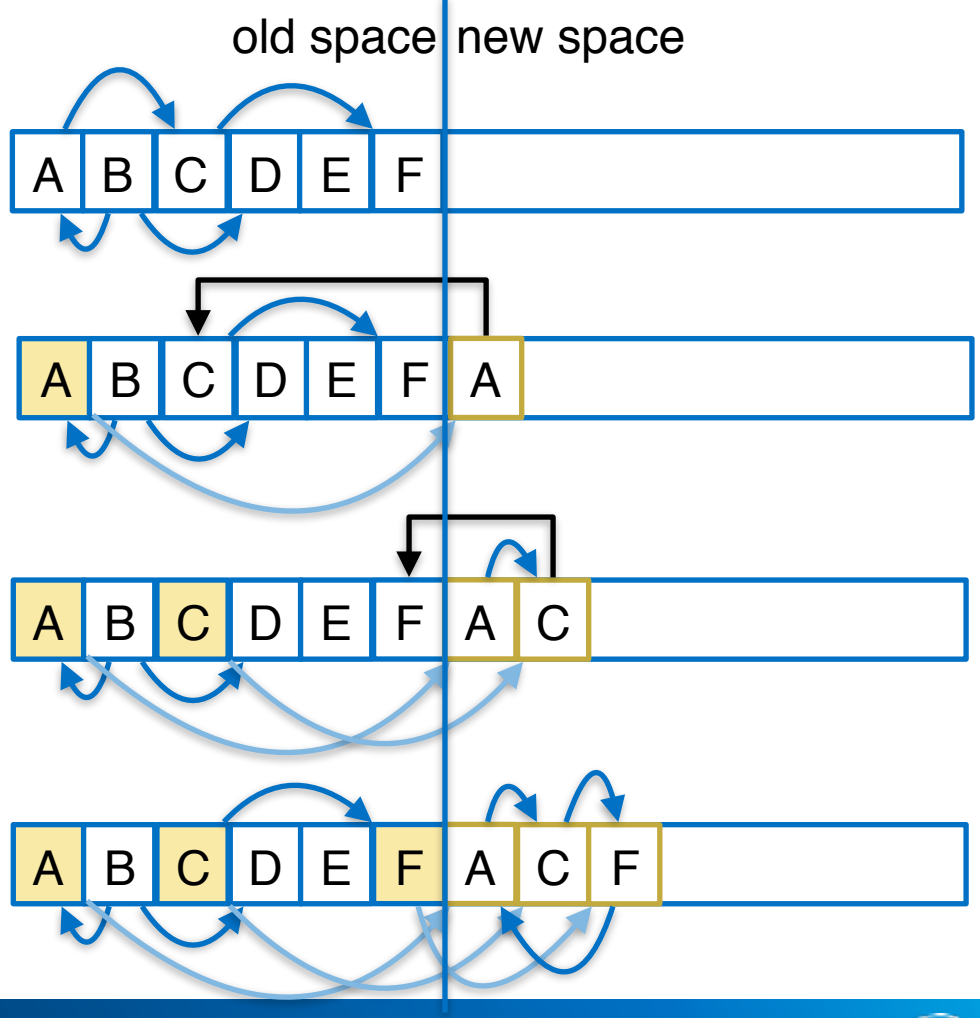


Stop & copy Example

Pointer

Wrong pointer

Forward pointer



Stop & copy #3

s&c - очень быстрый алгоритм

алокация дешева

сбор относительно дешев (зависит от количества мусора). Один проход для каждого объекта + не трогает мусор

Стоимость s&c ~ O(| live objects |)

Стоимость m&s ~ O(| Objects num |)

Однако не применим для языков типа C/C++.

Reference counting

Вместо того, чтобы ждать, пока память переполнится попробуем найти обьекты, на которых нет ссылок.

Для любого обьекта храним количество указателей на него (счетчик ссылок)

Каждое присваивание изменяет этот счетчик.

Т.е. каждое выражение вида x <- y преобразуется в

rc(Py) <- rc(Py) + 1

rc(Px) <- rc(Px) - 1

if (rc(Px) == 0) then free Px

x <- y

Reference counting #2

+ Прост в исполнении

+ Сборка мусора не останавливает программу

- Нет возможности очистить циклические зависимости

- Замедляет общее исполнение программы

Устраняет контроль программиста над памятью (delete). Все-же возможны утечки памяти, которые сложно отловить.

Программист должен очищать обнулять указатели на большие структуры, которые больше не использует

C++ : ***unique_ptr, weak_ptr, shared_ptr***

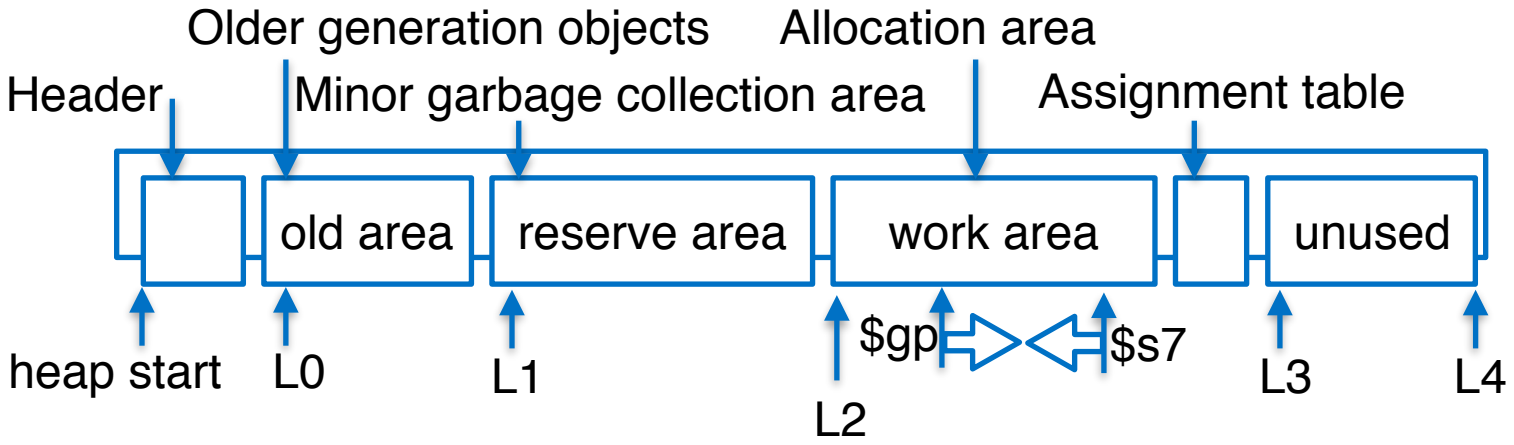
Cool Garbage collector.

Усовершенствованная модель s&c

(Simple Generational Garbage Collection Andrew W. Appel)

Документация находится в trap.handler стр. 1200

Этот файл загружается в симулятор вместо OS.



Trap handler

В Cool общение с пользователем происходит через пре-определенные классы: Int, Bool, String, IO, Object.

Описаны в cool-tour.pdf "7.4 Labels Expected"

В trap.handler на мипсовом ассемблере реализованы:

equality_test (сравнение строк),

Object.copy

Object.abort

Object.type_name

Scratch registers $v0,$v1,

$a0–$a2,

$t0–$t4

Heap pointer $gp

Limit pointer $s7

Naming agreement

Методы и метки, обязательные к добавлению в генерированный файл.

Имя метки	Содержание	Секция
Main_protObj	Прототип обьекта, класса Main	Data
Main_init	Функция инициализации обьекта класса Main, указатель на который передан в $a0	**Code**
Main.main	Основной метод, класса Main. $a0 - указатель на обьект Main.	**Code**
Int_protObj	Прототип обьекта класса Int	Data
Int_init	Код, инициализирующий обьект класса Int, указатель на который находится в $a0	**Code**
String_protObj	Прототип обьекта класса String	Data
String_init	Код, инициализирующий обьект класса String, указатель на который находится в $a0	**Code**
_int_tag	Слово (word), содержащее идентификатор класса Int.	Data
_bool_tag	Слово (word), содержащее идентификатор класса Bool.	Data
_string_tag	Слово (word), содержащее идентификатор класса String.	Data
class_nameTab	Таблица, содержащая список имен классов в соответствии с идентификатором (tag). В таблице хранятся указатели на обьекты класса String	Data
bool_const0	Обьект типа Bool, содержащий логическое значение false	Data

Trap handler methods

Методы обьектов, обьявленные в trap.handler файле

Object.copy	Процедура, возвращающая новую копию объекта, переданного в $a0. Результат будет в $a0
Object.abort	Процедура, которая выводит имя класса объекта в $a0. Завершает выполнение программы
Object.type_name	Возвращает имя класса объекта, переданного в $ a0, как строковый объект. Использует тег класса и класс таблицы nameTab
IO.out_string	Печать строки на терминал, которая положена на стек. Не меняет $a0.
IO.out_int	Печать целого числа на терминал, берет значение из стека. Не меняет $a0.
IO.in_string	Читает строку из терминала и возвращает обьект типа String через $a0. (перевод строки - не является ее частью)
IO.in_int	Читает число с терминала и возвращает обьект типа Int через $a0
String.length	Считает размер строкового обьекта, переданного через $a0. Кладет результат в $a0
String.concat	Совмещает два обьекта типа String. Обьект, передаваемый через стек объединяется с обьектом, передаваемым через $a0. Результат кладется в $a0
String.substr	Берет подстроку от обьекта String, переданного через $a0. Начальный индекс и длинна подстроки передаются через стек(индекс под длинной). Результат в $a0

Trap.handler + exception handlers

Вспомогательная функция

equality_test	Проверяет, являются ли переданные примитивные обьекты (Int, String, Bool) одного и того же типа и хранят одно и то же значение. Обьекты передаются через $t0 и $t1. Ожидается обьект true в $a0 и false в $a1. Результат - в $a0 (не меняется, если обьекты совпадают и меняет его на значение в $a1, если обьекты разные).

Обработчики исключений (! не прерываний)

_dispatch_abort	Вызывается, при попытке вызвать пустой(void) обьект. Печатает номер строки из $t1 и имя файла из $a0, из которого происходит попытка вызова. Прерывает исполнение программы.
_case_abort	Вызывается, когда аргументу case не находится соответствия. Печатает имя класса в $a0 и прерывает программу.
_case_abort2	Вызывается, когда оператору case переданный пустой(void) обьект. Печатает номер строки из $t1 и имя файла из $a0 из которого происходит попытка. Прерывает исполнение программы.

+ методы MemMgr:

_MemMgr_Init

_MemMgr_Alloc

_MemMgr_QAlloc

_MemMgr_Test

+ методы GC

_GenGC_Init

_GenGC_Assign

_GenGC_Collect

...

_NoGC_Init

_NoGC_Collect

Layout of Cool object

"-1" - зарезервировано GC

offset -4 Garbage Collector Tag

Метка (Label) обьекта находится ниже

Objname_const:

offset 0 Class tag

offset 4 Object size (in 32-bit words)

offset 8 Dispatch pointer

offset 12. . . Attributes

```
        .align      2
        .word       -1
str_const1:
        .word       3
        .word       6
        .word       String_dispTab
        .word       int_const9
        .ascii      "x is "
        .byte       0
```

Завершающий 0

Filling sections

В кодгене необходимо для каждого класса заполнить:

- Таблицу имен методов (Dispatch table) каждого класса (включая конструкторы)
- Таблицу констант
- Прототипы обьектов каждого класса

```
A_dispTab:
        .word       Object.abort
        .word       Object.type_name
        .word       Object.copy
        .word       A.printX
        .word       A.new_st
        .word       A.bump
```

```
        .word       -1
A_protObj:
        .word       2
        .word       4
        .word       A_dispTab
        .word       int_const1
```

```
class A {
  x : Int <- x + 1;
  printX():Object {{
    (new IO).out_string("x is ");
    (new IO).out_int(x);
    (new IO).out_string("\n");
  }};
  new_st():A { ... };
  bump():Object {  ...  };
};
```

см. стр.22

```
A.printX:
        sw          $fp 0($sp)
        addiu       $sp $sp -4
        sw          $s0 0($sp)
        addiu       $sp $sp -4
        sw          $ra 0($sp)
        addiu       $sp $sp -4
        addiu       $fp $sp 4
        move        $s0 $a0
        # begin of expression in method
        la          $a0 str_const1
        sw          $a0 0($sp)
        addiu       $sp $sp -4
        # code for new
        la          $a0 IO_protObj
        jal         Object.copy
        jal         IO_init
...
```

```
void typcase_class::code(Environment *env) {
  std::map<int, branch_class*> branch_map;
    for (int i = cases->first(); cases->more(i); i = cases->next(i)) {
        branch_class* b = (branch_class*) cases->nth(i);
        CgenNodeP n =
                env->cgen_table->get_node_by_name(b->type_decl);
        int branch_tag = n->get_tag();  branch_map[branch_tag] = b;
    }
    expr->code(env);
    emit_push(ACC, s);    env->cur_exp_oft += -4;
    int label_begin = env->get_label_cnt();
    emit_bne(ACC, ZERO, label_begin, s);
    s << LA << ACC << "\t";
    std::string filename = env->cur_class->get_filename()-
>get_string();
    stringtable.lookup_string((char*) filename.c_str())->code_ref(s);
    s << endl;
    //li $t1 <line number>
    emit_load_imm(T1, line_number, s);
    emit_jal("_case_abort2", s);
    emit_label_def(label_begin, s);
    emit_load(T2, 0, ACC, s);
   int label_end = env->get_label_cnt();
  for (auto iter = branch_map.rbegin(); iter != branch_map.rend(); iter++) {
       branch_class * b = iter->second;
      env->sym_table.enterscope();  int class_tag = iter->first;
      int max_tag = env->cgen_table->get_max_descen_tag(b->type_decl);
      int label_next = env->get_label_cnt();
      emit_blti(T2, class_tag, label_next, s);    emit_bgti(T2, max_tag, label_next, s);
      env->sym_table.addid(b->name, "($fp)");
      b->expr->code(env);
       emit_pop(s);
       emit_branch(label_end, s);
       emit_label_def(label_next, s);
       env->sym_table.exitscope();
    }
    emit_jal("_case_abort", s);
    emit_label_def(label_end, s);
    env->cur_exp_oft += 4;
}
```