Challenge Set I

SNU 4190.310, 2006 가을

이 광근

Due: 12/18 10:30

한 문제라도 올바른 답을 제출한 사람은 최종 성적이 상승합니다.

Challenge 1 (1 단계 상승)"S80 = SM5 + GC"

SM5 메모리에서는 무한히 많은 새로운 주소가 샘솟을 수 없다.

이제, SM5의 메모리는 8K(2^{13})개의 주소만 있다고 하자. 위의 문제에서 주어진 모듈 `Sm5`를 뜯어 고쳐서, `malloc`할 것이 더이상 없을 때 메모리를 재활용하는 함수 gc를 장착한다. 즉,

$$(S, M, E, \texttt{malloc} :: C, K) \;\Rightarrow\; (l :: S, M, E, C, K) \quad \text{new } l$$

이 아래와 같이 변경될 것이다:

$$\begin{array}{lcll} (S, M, E, \texttt{malloc} :: C, K) & \Rightarrow & (l :: S, M, E, C, K) & \text{new } l, \text{ if } |\text{dom}M| < 2^{13} \\ (S, M, E, \texttt{malloc} :: C, K) & \Rightarrow & (l :: S, M', E, C, K) & \text{recycled } l, \text{ if } |\text{dom}M| = 2^{13} \end{array}$$

위에서 M'은 E와 K를 통해서 M의 메모리중에서 사용될 수 있는 부분만을 가지고 다시 구성한 메모리가 된다. 이러한 일을 하는 함수 `gc`를 구현한다:

$$\texttt{gc} : \mathit{SM5} \rightarrow \mathit{Memory}$$

메모리 재활용 함수 `gc`는 현재의 SM5 기계 상태를 받아서, 앞으로 접근할 수 있는 메모리만을 간추려서 돌려준다. 함수 `gc`는 실제 구현보다 훨씬 간단하다. 현재 메모리에서 미래에 사용할 부분만을 모으면 될 것이다. □

Challenge 2 (1단계 상승) "Concurrent GC"

메모리 재활용이 대상 프로그램을 멈추지 않고 진행될 수 있다. 프로그램 실행은 실행대로, 메모리 재활용은 재활용대로 서로 동시에 진행할 수 있다. 특히 CPU가 두 개라면 그 두개가 실재로도 동시에 가능하게 된다. 이러한 알고리즘이 그동안 많이 연구되어 왔는데, 이제 비로소 누구나 사용할 수 있는 환경에 적용될 때가 되었다 ("Dual-Core CPU!").

"Concurrent GC"라는 키워드로 관련 연구논문을 알아보고 정리해서 리포트로 제출한다. Survey논문은 구체적이어야 하고, 될 수 있으면 많은 논문을 모으고 조사한 후 정리해야 한다. ACM 이나 IEEE Digital Library, Google Scholar등을 통해서 쉽게 탐색을 시작할 수 있을 것이다. 신뢰도 높은 연구결과들이 발표되는 곳은 명망있는 저널이나 학회들이다. ACM Symposium on Principles of Programming Languages, ACM Symposium on Programming Language Design and Implementation, ACM Symposium on Memory Management 등이 관련 연구가 발표되는 프리미어 학회들이다. 그 학회들의 학회록(proceedings)를 뒤지게 될 것입니다.

□

Challenge 3 (2단계 상승) "확인하자"

메모리 재활용기(garbage collector)는 올바른가?

강의에서 논의한대로, 자동 메모리 재활용기(garbage collector)의 원리는 이렇습니다:

- 원칙: 프로그램의 진행을 멈추고 나서, 지금까지 할당된 메모리중에서 미래에 사용할 수 있는 메모리를 제외하고 나머지는 모두 재활용해야 한다.
- 사실: 재활용을 완전하게(빠뜨리지 않고) 할 수 있는 방법은 없다. 있다면 그런 재활용기를 이용해서 *Halting Problem*을 풀 수 있기 때문이다.
- 양보: 그러나 재활용을 안전하게는(빠뜨리는 것은 있으나) 할 수 있는 방법은 있다. 뭔고하니,
  - ⋆ 실행할 식 E의 현재 환경(environment)으로부터 현재의 메모리를 통해 다다를 수 있는 모든 주소들은 앞으로 E를 실행중에 다시 사용될 가능성이 있다. 이것들만 빼고 재활용하자. 즉, 그렇게 해서 다다를 수 없는 메모리 주소들은, 과거에 할당되어서 사용되었으나 앞으로의 E를 실행하는데는 사용되지 않을 것이 분명하므로, 재활용해도 된다.

모든 현대 언어들의 메모리 재활용기는 위의 방식으로 구현되어 있습니다 (Java, ML, Scheme, Haskell, C#, Prolog, etc.)

이번 숙제에서는 위의 주장 $\star$ 가 옳은지를 확인하는 것입니다.

위의 사실이 어떻게 엄밀한 정리(theorem)로 표현되는지를 살펴보시기 바랍니다. 우선, 그 정리에서 사용하는 용어를 정확히 정의해야 합니다. 프로그램은 어떤 언어로 짠 프로그램을 말하는지, 프로그램의 실행(semantics)은 무엇인지, 환경에서 다다를 수 있는 메모리는 무엇을 뜻하는 지에 대한 정의들.

대상 프로그래밍 언어는 D 라는 언어로 합시다. D 프로그램 실행의 정확한 정의는 아래와 같습니다.

[Now on in English]

e	::=	n	integer
	$\mid$	x	variable
	$\mid$	`{`x `:=` e`}`	record
	$\mid$	`{}`	nil record
	$\mid$	$e.x$	record field
	$\mid$	x`:=`e	assignment
	$\mid$	e`;`e	sequence
	$\mid$	`let` x e e	local block

x	$\in$	Var			variables
v	$\in$	Val	$=$	$Num + Record$	values
n	$\in$	Num			numbers
l	$\in$	Loc			locations
r	$\in$	$Record$	$=$	$Var \xrightarrow{\text{fin}} Loc$	records
σ	$\in$	Env	$=$	$Var \xrightarrow{\text{fin}} Loc$	environments
M	$\in$	$Memory$	$=$	$Loc \xrightarrow{\text{fin}} Val$	memories

Notation: $A \xrightarrow{\text{fin}} B$ is the set of functions from a finite subset of A to B. Let f be a function $\{a \mapsto 1, b \mapsto 2\}$, then we write $domf$ for the domain $\{a, b\}$ of f. We write $f[2/a]$ for a new function $\{a \mapsto 2, b \mapsto 2\}$, and $f[3/c]$ for $\{a \mapsto 1, b \mapsto 2, c \mapsto 3\}$.

The semantics rules precisely defines the execution of D expressions; they define how relations of the form

$$\sigma, M \vdash e \Downarrow v, M'$$

to be inferred. The relation is read "expression e computes value v under environment σ and memory M."

Definition 1 (Expression's semantics/execution) *An expression e's execution is defined to be the inference tree for* $(\sigma, M \vdash e \Downarrow v, M')$. *If there is no such* σ, M, v, *and* M', *then the expression has no meaning, no way to execute.*

In particular, program e's execution is the inference tree for $(\emptyset, \emptyset \vdash e \Downarrow v, M')$ *for some* v *and* M'.

[Int] $$\sigma, M \vdash n \Downarrow n, M$$

[Var] $$\frac{M(\sigma(x)) = v}{\sigma, M \vdash x \Downarrow v, M}$$

[Rec] $$\frac{\sigma, M \vdash e \Downarrow v, M' \qquad l \notin domM \cup domM'}{\sigma, M \vdash \texttt{\{}x \texttt{ := } e\texttt{\}} \Downarrow \{x \mapsto l\}, M'[v/l]}$$

[NilRec] $$\sigma, M \vdash \texttt{\{\}} \Downarrow \{\}, M$$

[Field] $$\frac{\sigma, M \vdash e \Downarrow \{x \mapsto l\}, M' \qquad M'(l) = v}{\sigma, M \vdash e.x \Downarrow v, M'}$$

[Assign] $$\frac{\sigma, M \vdash e \Downarrow v, M' \qquad \sigma(x) = l}{\sigma, M \vdash x \texttt{ := } e \Downarrow v, M'[v/l]}$$

[Seq] $$\frac{\sigma, M \vdash e_1 \Downarrow v_1, M' \qquad \sigma, M' \vdash e_2 \Downarrow v_2, M''}{\sigma, M \vdash e_1 \,;\, e_2 \Downarrow v_2, M''}$$

[Let] $$\frac{\sigma, M \vdash e_1 \Downarrow v_1, M' \qquad \sigma[l/x], M'[v_1/l] \vdash e_2 \Downarrow v_2, M'' \qquad l \notin domM \cup domM'}{\sigma, M \vdash \texttt{let } x \; e_1 \; e_2 \Downarrow v_2, M''}$$

Definition 2 *reach*(σ, M) *is the set of locations in* M *that are reachable from the entries in* σ. *It is the smallest set that satisfies the two rules:*

$$\sigma(x) \in reach(\sigma, M) \qquad \frac{l \in reach(\sigma, M) \qquad M(l) = \{x \mapsto l'\}}{l' \in reach(\sigma, M)}$$

Now the correctness of the garbage collector idea hinges on the following theorem.

Theorem 1 *In the inference of* $(\sigma, M \vdash e \Downarrow v, M')$, *the set of used (read or written) locations in* M *is included in* $reach(\sigma, M)$.

The proof of the theorem needs the following lemma:

Lemma 1 *If* $(\sigma, M \vdash e \Downarrow v, M')$ *is infered, then* $reach(\sigma, M) \supseteq reach(\sigma, M') \cap dom M$.

Both the proofs of the theorem and the lemma can be done by induction on expression e. You are challenged to complete the two proofs. □