Typed Memory Management in a Calculus of Capabilities

Karl Crary
C
D
Cornell University
Greg Morrisett
Cornell University

A

A
g
c
p
provide an alternative to garbage collection by making mem-
o
not been clear how to use regions in low-level, type-safe code.
W
C
m
straightforward to compile to a typed assembly language.
Source languagesy be compiled to our language using
k
times need not be lexically sco6(-) d in our language, yet the
l
y
p
T
s
m
p
a

1

A
t
l
i

p
g

region types and lifetimes and how to implement their the-

1. Deallocation is explicit but provably safe.

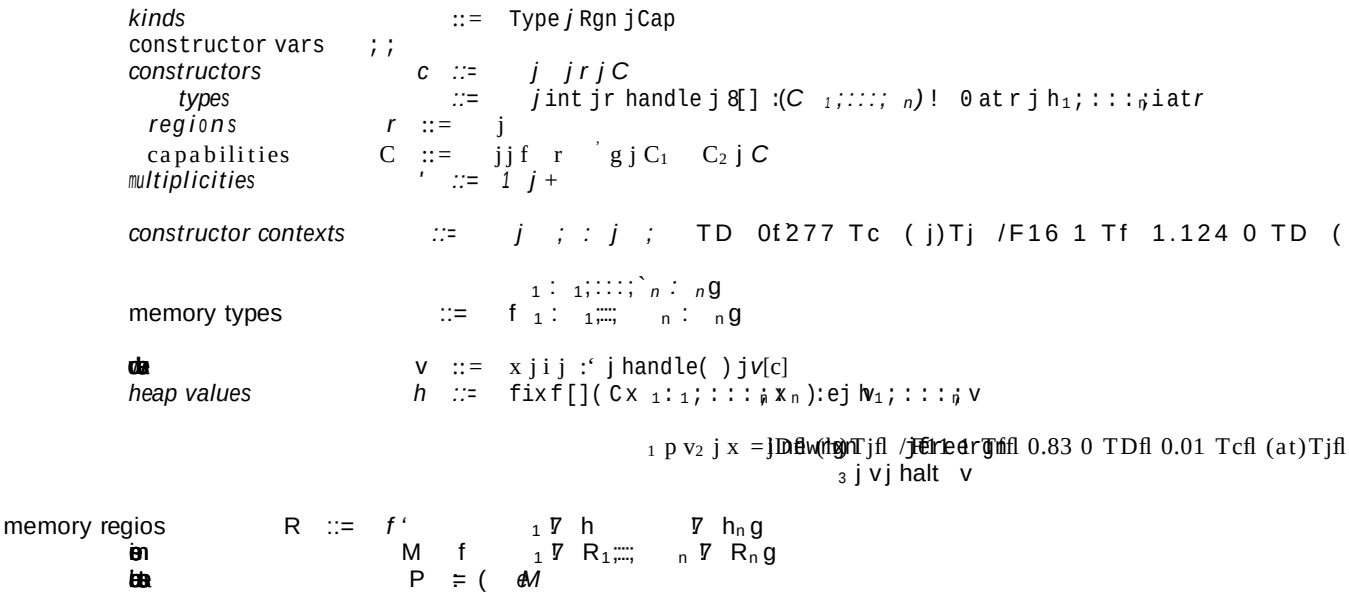
The Tofte-Talpin calculus uses a lexically scoped expres-

Both Birkedal *et al.* [4] and Aiken *et al.* [1] observ5(d)-373(t)-9(hat)-357(a)]TJfl -18.282 -1.097 TDfl -0.004 Tcfl [(compl)-13(e)-2(t)-6(e)-2(l)
use of memory n many cases. They proposed a seres dW]TJfl 0 -1.097 TDfl -0.005 Tcfl [(opt)-7
even mprove upn tradtional tracng garbage collecton n
some cases. Although ther optmizatons are safe, there s
no smple prdof- dr type-ccker that an untrustng cli5(n)29(t)]TJfl 0 -1.097 TDfl -0.002 Tcfl [(can)
straghtforward code-generaton requres that we stray from
the TdWte-Talpn framework and allow arbtrary separaton
f allocatn and deallocation points. Thereforewh re-
on nference brngs us half way to our goalr495()-13n order to
construct a strongly-typed regon-based assemly language
we must r5(-)-12(e)-5(xamy-7(e)my the fundamental qu5(s)-3(t)-9(i)-16(on:)]TJfl /F12 1 Tffl 21.628 0
it will nd longer be accessed. Operatg systems such as
Hydra [41] hauve solved the access co*nbility*94(p)16(e)-3(ob)6(l)-12(e)-1(m)-512(b)-21(ef)-10(ore)]TJ
object and requrin88(468(t)7(h)18(a)14(t)-475(t)7(h)18(e)-471(u)18(s)13(e)1t(r)-472(p)18(r)9(e)11
future access.

2 Overview of Contributions

In the rest of ths paperr412(w)18(e)-375(d)7(scr)be a strongly typed language called the Capability Calculus. Our language's type
prouvdes an e cen way to check the safet of explicit, arbrarily ordered region allocatin and deallocation instructions usin88(414(a)-387(n)18(o)14(t)7(i)0)n of capability. As in traditional capability systms, ur type system keeps track of capabilityp
t
the capabilits i our calculus are a purely stati concept and thus ther mplementatdn requres no run-tme overhead.

e have a purely



which states that (when memory has type Ψ, free constructor variables have kinds given by and free value variables have types given by Γ) it is legal to execute the term e, *provided that the capability C is held.* A related typing judgement is

$$\Psi; \quad ; \Gamma; C \; ` \; d \qquad {}^{0}; \Gamma^{0}; C^{0}$$

which states that if the capability C

Uniqueness

We solve this problem by using bounded quanti cation to relate $_1$, $_2$ and . Suppose h has type:

$$8[\ _1{:}\mathtt{Rgn};\ \ {:}\mathtt{Rgn};\ \ f\ \ ;\ \ {}^{+}_{2}\ \ 1\ \ {}^{0}) \ !\ 0\ \mathtt{at}\ r^{00}$$

If we hold capability $\mathsf{f}\, r^{1} g$, we may call h by instantiating $_1$ and $_2$ with r and instantiating with fr . This instantiation is permible because $r^{1}g$ $fr^{+}; r\ g$. As with g, the continuation will possess the capability r g, allowing it

$_1$ and $_2$, since $^{+}_{1}$ g.

Bounded quanti cation solves the problem by revealing some information about a capability , while still requiring the function to be parametric over . Hence, when the function calls its continuation we regain the stronger capability (to r), although that capability was temporarily hidden in order to duplicate r. More generally, bounded quanti cation allows us to hide some privileges when calling a function, and regain those privileges in its continuation. Thus, we support statically checkable attenuation and ampli cation of capabilities.

2.4 Formal Properties of the Calculus

The most important properties of the Capability Calculus are Type Soundness and Complete Collection. Each can be proven from the formal semantics in Appendix A.

) *then* (M^{0},e^{0}) *is notstuck.*

The proof of soundness is straightforward, making use

level operations such as the atomic allocation and initialization of data. In the companion technical report [5], we show that the capability constructs intract benignly with te process oftype-preserving compilation described by Morrisett *et al.* [24] and we use the techniques described in this paper t modify teir typed assmbly language t allow explicit dealloc(om)-1(t)-8(i)-15(on)-372(of)-334(dat)-8(a)-375(st)-8(r)-6(u)3(c)-4(t)-8(ur)-6(e)-4(s)-2(.)]TJfl 1.499 -1.097 TDfl -0.007 Tcfl [(I)-11(n)-3 but with a few changs we believe our capability apparatus may be used in a variety of othr settings as well. One potential application involvs re Tcfl [(‚)18(c)11(in)18(g)-280(t)7(h)18(e)-283(o)41(v)43-erhead of communication across the user-kert address space boundary in traditional operating systems. Typically, in such systems, when data in user space is presented to the kernel, the kernel must copy that data to ensure its integrity is preserved. However, if a user process hand-o a uniqu capbility for a region to the kernel,the kernel does not have to copyhat regon's Tcfl [((a)14(t)7(a)14(;)-321(w)2(it)7(h)18(o)14(u)18(t)-314(t)7(h)18(e)-310(c)11(a)14(p)18(a)14(b)18(ilit)34(y)96(,)-321(t)7-h)1 sive access to shared mutable data in a multi-threaded environment, by viewing loc(24(ks)-376(as)-403(anal)-I4(ogous)-403(t)-7(o)-401(r)-5(egi)-14(o)0(ns.)-576(I)-9(f)-387(w)15(e)]TJfl 0 -1.097 TDfl -0 little ofhe struc(-5(t)-9(ur)-7(e)-460(o)-2(f)-44s)-35he store. Anumber of other researchers [10, 7, 32] have studied static analyses that infer the shapes of data structures and the aliasng relationsips between them. We plan to nvestigate how to se these nergrained memory models to increase the flexibility of our type system.

A connection can also be drawn between capabilities and monadic type systems. Work relating e ects to monads [21, 28, 18, 8] has viewed e ectful functions as pure functionthat returntate tranformers. Th might be called

bu may alias oth regions. Second, afterwe dynamically check which of t1-8(h)3(e)-352(t)1b(w capability. Harper and Morrisett's typecase [13] mechanism developed forthe TIL com (p)1TD -0.0(l)-17(e)-6(r)-275(a)-3(nd)-320(f)-15(u r (e)-6(gi)-17(ons)-299(t)-10(o)-297(b)-26(e)-300(i)-17(nde)-6(p)-26(e)-6(nde forAikenanalyses,thatthose analyses may be used to produce type correct code in the apability Calculus.

Gay and Aiken [9] have developed a safe region ime-

6 Acknowledgements

We would like to thank Lars Birkedal, Martin Elsman, Dan Grossman, Chris Hawblitzel, Fred SmithmStephanie Weirich, Steve Zdancewic, and the anonymous reviewers for their comments and suggestions.

References

[1] Alexander AikenManuel F¨

ACM SIGPLAN Conference on Programming Languie Design and Implementation

[27] George Necula and Peter Lee. The design and implementation of a certifying compiler. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 333 { 344, Montreal, June 1998.

[28] Simon L. Peyton Jones and Philip Wadler. Imperative functional programming. In *Twentieth ACM Symposium on Principles of Programming Languages*, Charleston, South Carolina, January 1993.

[29] John C. Reynolds. De nitional interpreters for higher-order programming languages. In *Conference Record of the 25th Niional ACM Conference* , pages 717{740, Boston, August 1972.

[30] John C. Reynolds. Syntactic control of interferenl In *Fifth ACM Smposium on Principles of Programmin Languages*, pages 39{46, Tucson, Arizona, 1978.

[31] John C. Reynolds. Syntactic control of interferenl6(ce,)]TJfl 2.168 -1.097 TDfl -0.006 Tcfl [(par)-6(t)-409(2.)-684(I)-10(n)]TJfl /F12 analysis problems in languages witaldestructive updating. *ACM Transactions on Programming Languages and Systems*, 20(1):1{50, January 1996.

[33] J.-P. Talpin and P. Jouvelot. Polymorphictype, region, and e ect inferenl7(c)]TJfl /F12 1 Tffl 9.984 0 TDfl 0.01 Tcfl [(J)14(our)15(nal)-424(o)-1(f)-425(F)91(unctional)-397(P)10(r)68(o)53(g

[38] Philip Wadler. A taste of linear logic. In *Mathematical Foundations of Computer Science*, volume 711 of *LNCS*, Gdansk, Poland, August 1993. Springer-Verlag.

[39] Robert Wahbe, Steven Lucco, Thomas Anderson, and Susan Graham. E cient software-based fault isolation. In *Fourteenth ACM Symposium on Operating Systems Principles*, pages 203{216, Asheville, December 1993.

[40] Andrew K.Wright and Matthias Felleisen. A syntactic approach to type soundness. *Inmation an Computation*, 115(1):38{94, 1994.

[41] W. A. Wulf, R. Levin, and S. P. Harbison. *Hydra/C.mmp: An Experimental Computer System*. McGraw-Hill, New York, NY, 1981.

	$(M;e) \not\to P$
If $e =$	then $P =$
$\mathtt{let}\, x = v\, \mathtt{in}\, e'$	$(M; e'[v/x])$
$\mathtt{let}\, x = i\, p\, j\, \mathtt{in}\, e'$	$(M; e'$
$\mathtt{let}\, x = h\, \mathtt{at}\, (\mathtt{handle}($ $'$ and $\in Dom(M)$	($'$ where $\ell \in$
$\mathtt{let}\, x = {}_i(\ :`)\, \mathtt{in}\, e'$	$(M; e \quad /x])$
	$(M\, f \qquad [\ ;$ where $\in M$ and $\in e$
$\mathtt{let}\, \mathtt{freergn}($	$(M\, n\ ; e\ )$
$\mathtt{if0}\, 0\, \mathtt{then}\, e\, \mathtt{else}\, e$	$(M; e\ )$
$\mathtt{if0}\, i$	

aratidn d is well-fdC-4(e)-9(m)-9(e)-7(d)-321(a)-4(nd)-375(pr)-9(d)-30(duc)-7(e)-7(s)-38

Expression e is well-fdrmed.

$\Psi; \Delta; \Gamma \vdash \mathit{hat}\ r :$

$$\frac{\Psi; \Delta; \Gamma \vdash \mathit{fix}\ f[\alpha_1 : K_1, \ldots, \alpha_n : K_n](C; x_1 : T_1, \ldots, x_m : T_m) : \mathit{hat}\ r : T \quad \Psi; \Delta \vdash \mathit{hat}\ r : T \quad \alpha_1, \ldots, \alpha_n \in \mathit{Dom}(\Gamma)}{\Psi; \Delta; \Gamma \vdash \mathit{fix}\ f[\alpha_1 : K_1, \ldots, \alpha_n : K_n](C; x_1 : T_1, \ldots, x_m : T_m) : \mathit{hat}\ r : T}$$

$$\frac{\Psi; \Delta; \Gamma \vdash v_i : T_i \ (\text{for } 1 \le i \le n) \quad \Psi; \Delta \vdash r : \mathrm{Rgn}}{\Psi; \Delta; \Gamma \vdash \mathit{hat}\ v_1, \ldots, v_n : T_1, \ldots, T_n\ \mathit{at}\ r} \qquad \frac{\Psi; \Delta \vdash \mathit{hat}\ r : T \quad \Psi \vdash T = \mathrm{Type}}{\Psi; \Delta; \Gamma \vdash \mathit{hat}\ r : T}$$

$\Psi; \Delta; \Gamma \vdash v : T$

$$\frac{}{\Psi; \Delta; \Gamma, x : T \vdash x : T}$$