Computing Typ

Daniel Weise
Erik Ruf

D

Te

Computin

Introduction

A *program specializer* (also cal
the possible inputs of the prog
two argument function could b
being any value at all. Progra
compilers and compiler generat
up computations by two order
programs [6].

Program specializers opera
sions that can be reduced are
program.[1] A major goal of spe
gram as possible. The accurac
are both limited by the amoun
izers often ignore or throw awa

will prove, for example, that t
and a store of the form ((a .
will return a store of the same
`if`-expressions. Similarly, our
an unknown set of updates and
`(c . <some-value>))` will re
point analysis. The appendices
for a very simple language.

The three key mechanisms
a runtime value, and which co
and a code expression indicati
create a new symbolic value rep
computations for computing t
been presented before to remed
type propagation, and overly

```
(define (update-store store
  (if (eq? (caar store) name
      (cons (cons (caar stor
      (cons (car store) (upd

(define (process-updates upd
  (if (null? updates)
      store
      (process-updates
        (cdr updates)
        (update-store store

(define (lookup-in-store nam
  (if (eq? name (caar store)
      (cdar store)
      (lookup-in-store name
```

lookups. Our system, which res

the specializer [24]. Finally, his

due to delaying side-effects unt

to face.

This paper has 7 sections. W

approximations. The second s

whose programs will be specia

lynchpin of our method (the res

The specializer and generalizat

the specializer's termination st

Section 7 discusses concrete ab

1 Program Speciali

The user of a specializer speci

Our definition of program
evaluator as a program that
who describes partial evaluatio
specialized program only requi
time. There is little fundame
the distinction between our for
formulation of Abstract Interp
[23]. We simply find our app
differences will become more p

Our prior definition of prog
solely of code, *i.e.*, was an expre
and a type attribute, so we need
specializer (TPS) produces spe
Let the function SF_{code} return
return its type attribute. Infor

```
(define (eval exp env)
  (cond ((boolean? exp) exp)
        ((nat? exp) exp)
        ((var? exp) (lookup
        ((if? exp) (if (eval
                       (eval
                       (eval
        ((lambda? exp) (crea
        ((call? exp) (apply


        ((letrec? exp) (eval


(define (apply f args)
  (cond ((primitive-proc? f)
```

XNum = N

XBool = t +

XPair = S

Nil = ni

XFunc = ex

XVAL = XV

$\langle$`a - natural, a`$\rangle$

$\langle 4, 4\rangle$

Example symbolic values in
is used to express the code in a
the code attribute is the value
cell will not appear in the spec
at partial evaluation time but i
when V itself must be construc
specialized. The code specifyin
the formal parameters of the p

The code attribute specifies
attribute v and code attribute
to. We will notate a symbolic v
evaluation the `car` operation
a subset of V, the value attribu

```
(define pe-+
  (lambda (x y)
    (let ((x-val (sv-val x))
      (cond ((no-value? x) x
            ((no-value? y) y
            ((and (xnum? x-v
             (if (and (known
                 (known-valu
                 (make-sv 'a
            ((and (possible-
             (make-sv (join
            (else (type-erro

(define pe-cons
  (lambda (x y)
    (if (and (known-value? x
        (make-sv (cons x y)
```

```
(define pe
  (lambda (exp env stack)
    (cond ((constant? exp) e
          ((var? exp) (looku
          ((if? exp)
           (let ((pred (pe (
             (cond ((true? p
                   ((false?
                   (else
                     (let ((
                           (
                       (gene
          ((lambda? exp) (cr
          ((call? exp)
           (papply (pe (call
```

```
(define (generalize <v1, c1>
  (cond ((eqv? c1 c2) <v1, c
        ((and (pair? v1) (pa
         (generalize-pairs v
        (else <(join v1 v2),

(define (generalize-pairs (c
  (let ((new-car (generalize
        (new-cdr (generalize
    <(cons new-car new-cdr),

(define (join xval1 xval2)
  (cond ((eqv? xval1 xval2)
        ((bottom? xval1) xva
        ((bottom? xval2) xva
        ((top? xval1) xval1)
```

```
(define papply
  (lambda (f args stack)
    (cond ((primitive-proced
          ((lookup-specializ
           (make-residual-ca
          ((specialize? f ar
           (let ((stack-and-
             (let ((sf (spec
               (make-residua
          (else (apply-closu

;; The real version of this
(define specialize-closure
  (lambda (cl arg stack)
    (let ((sc (make-speciali
                (gen-name)
                arg
```

[25].
done. Too much abstraction, a
termination properties of a pro
is then based upon the abstra
point domain **unknown** (which c
number of values generated du
possibilities, this will not occur
infinite number of specializatio
must make it into the cache^6 a

For program specialization
could be returned by invocatio
symbolic value represents not
a freshly created name are colle
arguments. When specializatio
difference between unfolding a

```
(define *initial-closure-bod

(define (specialize-closure
  (let ((sc (make-specialize
    (cache! (cons cl args) s
    (find-fixed-point cl arg

(define (find-fixed-point cl
  (let ((new-body (apply-clo
    (set-specialized-closure
    (if
```