Dracula Reborn:

ML-style modules, Racket macros,

and ACL2 theorem proving

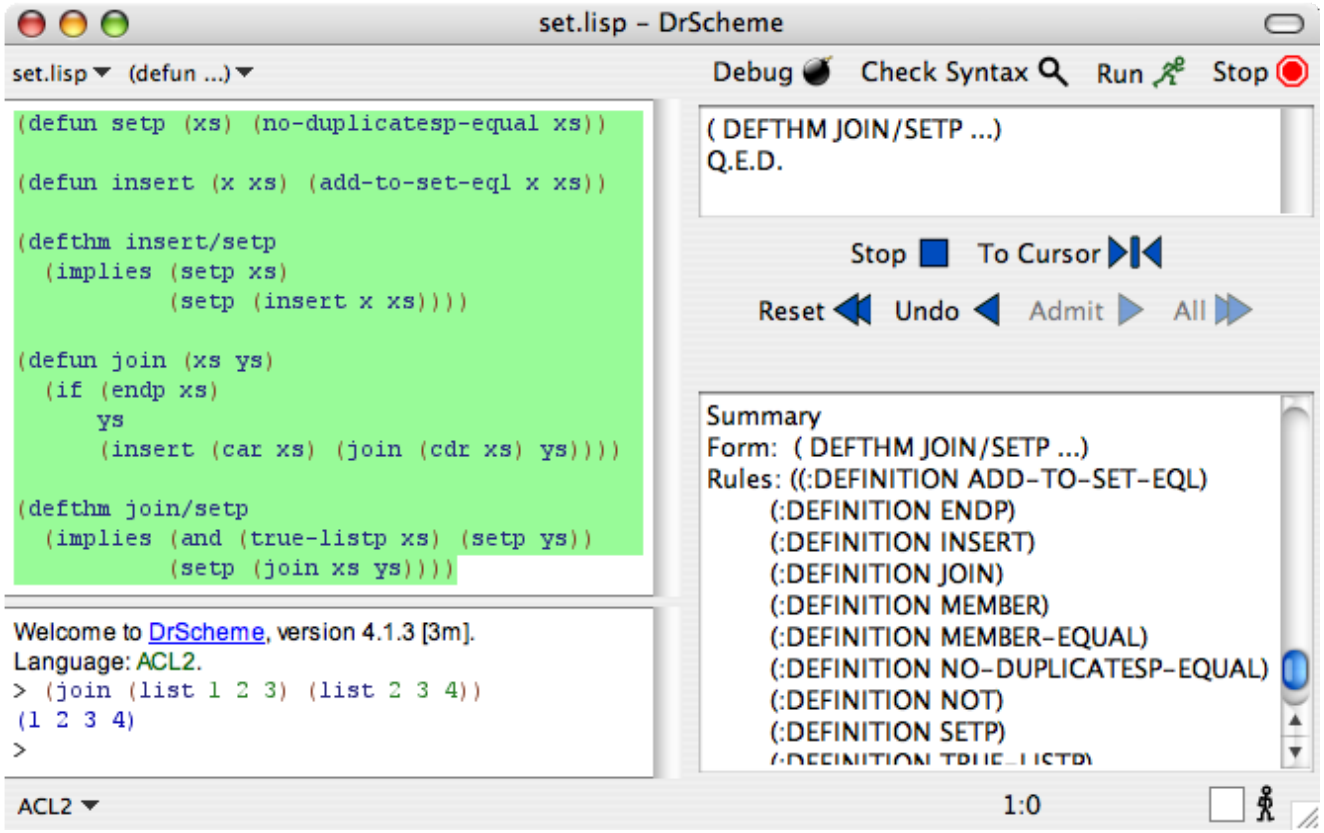
Carl Eastlund

Zoe Zhang

Matthias Felleisen

Northeastern University

Dracula


set.lisp – DrScheme
set.lisp
(defun ...)
Debug
Check Syntax
Run
Stop
(defun setp (xs) (no-duplicatesp-equal xs))
(defun insert (x xs) (add-to-set-eql x xs))
(defthm insert/setp
(implies (setp xs)
(setp (insert x xs))))
(defun join (xs ys)
(if (endp xs)
ys
(insert (car xs) (join (cdr xs) ys))))
(defthm join/setp
(implies (and (true-listp xs) (setp ys))
(setp (join xs ys))))
(DEFTHM JOIN/SETP ...)
Q.E.D.
Stop
To Cursor
Reset
Undo
Admit
All
Summary
Form: (DEFTHM JOIN/SETP ...)
Rules: ((:DEFINITION ADD-TO-SET-EQL)
(:DEFINITION ENDP)
(:DEFINITION INSERT)
(:DEFINITION JOIN)
(:DEFINITION MEMBER)
(:DEFINITION MEMBER-EQUAL)
(:DEFINITION NO-DUPLICATESP-EQUAL)
(:DEFINITION NOT)
(:DEFINITION SETP)
Welcome to DrScheme, version 4.1.3 [3m].
Language: ACL2.
> (join (list 1 2 3) (list 2 3 4))
(1 2 3 4)
>
ACL2
1:0

Modular ACL2

```
(interface TYPE
  (sig pred (x)))

(interface LIST-OF
  (extend TYPE)
  (sig list-of-p (x))
  (con list-of/nil (list-of-p nil))
  (con list-of/cons
    (iff (and (pred x) (list-of-p xs))
      (list-of-p (cons x xs)))))
```

Modular ACL2

```
(module List-of
  (import TYPE)
  (defun list-of-p (x)
    (cond
      ((atom x) (null x))
      (t (and (pred (car x))
              (list-of-p (cdr x))))))
  (export LIST-OF))

(link List-of-String
  (String List-of))
```

Racket bytecode verifier

```
(interface SOUNDNESS
  (extend STRUCTS)
  (extend BYTECODE-EXPR)
  (extend BYTECODE-VERIFY)
  (extend MACHINE-STATE)
  (extend MACHINE-EXECUTE)
  (con soundness
    (implies (and (bytecode-expr-p bc)
                  (verify-bytecode-program bc))
      (machine-state-p
        (machine-execute n
          (machine-initialize bc))))))
```

Top-down development

```
(module Soundness
  (import Bytecode-Soundness)
  (import Machine-Soundness)
  (defthm soundness
    (implies (and (bytecode-expr-p bc)
                  (verify-bytecode-program bc))
      (machine-state-p
        (machine-execute n
          (machine-initialize bc))))
    :hints (("Goal" ...)))
  (export SOUNDNESS))
```

Datatype abstractions

```
(interface STRUCTS
  (sig app (addr))
  (sig app-p (x))
  (sig app.fun (x))
  (sig app.args (x))
  (con app/predicate ...)
  (con app/constructor ...)
  (con app/selector ...)
  ...
  (con bytecode-expr/disjoint
    (and
      (implies (app-p x)
        (and (not (loc-p x)) (not (lam-p x))))
      ...)))
```

Datatype abstractions

```
(module Core-Datatype
  (import TYPE)
  (import LIST-OF)
  ...)

(link Datatype
  (String List-of Core-Datatype))
```

Dracula Reborn!

ML-inspired modules

```
(interface LIST-OF
  (mod elem : TYPE)
  (sig list-of-p (x))
  (con list-of/elem.pred ...)
  ...)

(module (List-of (Type : TYPE))
  : LIST-OF :where (Elem = Type)

  (defun list-of-p (x) ...))
```

ML-inspired modules

```
(module Datatype : DATATYPE
  (instance List-of-String
    (List-of String))
  (instance List-of-Number
    (List-of Number))
  ...)
```

Racket macros

```
(define-syntax cond
  (syntax-parser :literals (else)
    ((_ (else ~! default:expr)) #'default)
    ((_ (test:expr result:expr) . rest)
     #'(if test result (cond . rest)))))
```

Racket macros

```
(define-syntax datatype ...)

(datatype AST
  (:variants expr
    (var (name symbolp))
    (app (fun exprp)
         (args expr-listp))
    (lam (formals symbol-listp)
         (body exprp)))
  (:list-of expr-listp exprp))
```

Racket macros

```
(interface BINARY-OP
  (sig id-value ())
  (sig binary-++ (x y))
  (define-syntax ++
    (syntax-parser
      ((_) #'id-value)
      ((_ e:expr . rest)
       #'(binary-++ e (++ . rest))))))

(module Op : BINARY-OP ...)

(Op.++ 1 2 3 4)
```

To Do:

Implementation,

Experimentation,

and Dissertation.

Thank you!