A Nanopass Framework for Commercial Compiler Development

Andrew W. Keep and R. Kent Dybvig
Traditional Compilers

• Composed of a few large passes
• Each pass performs several tasks
• Difficult to maintain
• Difficult to add new optimizations
Nanopass Compilers

- Composed of many small passes
- Each pass performs a single task
- Easier to maintain
- Easier to add new optimizations
Potential Dangers

• Large languages can lead to large passes (lots of boilerplate)

• Tracking language changes can be difficult

• The nanopass framework helps address this
Nanopass Framework

- Embedded DSL for writing compilers
- Languages are formally defined
- Passes operate over languages
- Sarkar, Waddell, and Dybvig ICFP ’04
Defining a language

(define-language Lsrc
  (terminals
    (datum     (d))
    (primitive (pr))
    (uvar      (x)))
  (Expr (e body)
    x
    (quote d)
    (if e0 e1 e2)
    (begin e* ... e)
    (lambda (x* ...) body)
    (let ([x* e*] ...) body)
    (letrec ([x* e*] ...) body)
    (set! x e)
    (pr e* ...)
    (call e e* ...) => (e e* ...)))
Extending a language

(define-language L1
  (extends Lsrc)
  (terminals
    (¬ (datum (d)))
    (+ (constant (c))))
  (Expr (e body)
    (¬ (quote d))
    (+ (quote c)))))
Defining a pass

(define-pass convert-complex-datum : Lsrc (e) -> L1 ()
  (definitions
   (define const-x* '())
   (define const-e* '())
   (define datum->expr ---))
  (Expr : Expr (e) -> Expr ()
   [,x x]
   [(quote ,d)
    (if (constant? d)
     `(quote ,d)
      (let ([t (unique-name 't)])
       (set! const-x* (cons t const-x*))
       (set! const-e* (cons (datum->expr d) const-e*))
       t))]
   [(if ,e0 ,e1 ,e2) `(if ,(Expr e0) ,(Expr e1) ,(Expr e2))]
   [(begin ,e* ... ,e) `(begin ,(map Expr e*) ... ,(Expr e))]
   [(lambda ([,x* ...) ,body) `(lambda ([,x* ...) ,(Expr body))]
   [(let ([,x* ,e*] ...) ,body)
    `(let ([,x* ,(map Expr e*)] ...) ,(Expr body))]
   [(letrec ([,x* ,e*] ...) ,body)
    `(letrec ([,x* ,(map Expr e*)] ...) ,(Expr body))]
   [(set! ,x ,e) `(set! ,x ,(Expr e))]
   [(,pr ,e* ...) `(,pr ,(map Expr e*) ...)]
   [(call ,e ,e* ...) `(call ,(Expr e) ,(map Expr e*) ...)]
   [else (errorf who "invalid Expr form ~s" e))]
  (let ([e (Expr e)])
   (if (null? const-x*)
    e
    `(let ([,const-x* ,const-e*] ...) ,e))))
Defining a pass

(define-pass convert-complex-datum : Lsrc (e) -> L1 ()
  (definitions
   (define const-x* '())
   (define const-e* '())
   (define datum->expr ---))
  (Expr : Expr (e) -> Expr ()
   [(quote ,d)
    (guard (not (constant? d)))
    (let ([t (unique-name 't)])
     (set! const-x* (cons t const-x*))
     (set! const-e* (cons (datum->expr d) const-e*))
     t))]
  (let ([e (Expr e)])
   (if (null? const-x*)
    e
    `(~ 'let ([,const-x* ,const-e*] ...) ,e)')))

Monday, October 7, 13
Defining a pass

(define-pass convert-complex-datum : Lsrc (e) -> L1 ()
  (definitions
   (define const-x* '())
   (define const-e* '())
   (define datum->expr ----))
  (Expr : Expr (e) -> Expr ()
    [(quote ,d)
     (guard (not (constant? d)))
     (let ([t (unique-name 't)])
      (set! const-x* (cons t const-x*))
      (set! const-e* (cons (datum->expr d) const-e*))
      t))]
  (let ([e (Expr e)])
    (if (null? const-x*)
      e
      `(let ([,const-x* ,const-e*] ...) ,e))))
Question

Is the nanopass methodology suitable for use in commercial compiler development?
Approach

• Replace the compiler for Chez Scheme
• Support identical feature set
• Improve the nanopass framework (as needed)
• Graph coloring register allocator
• Improve and add optimizations
Comparing compilers

Front end:

Back end:
Comparing compilers
Evaluation

- Three sets of benchmarks
  - R6RS Benchmarks
  - Chez Scheme benchmarks
  - Source optimizer benchmarks
## Compile-time factors

<table>
<thead>
<tr>
<th>Optimize level</th>
<th>x86</th>
<th>x86_64</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.38</td>
<td>1.44</td>
</tr>
<tr>
<td>3</td>
<td>1.26</td>
<td>1.38</td>
</tr>
</tbody>
</table>

**Average factor of compile-time slow down**
## Run-time Performance

<table>
<thead>
<tr>
<th>Optimize level</th>
<th>x86</th>
<th>x86_64</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.72</td>
<td>0.76</td>
</tr>
<tr>
<td>3</td>
<td>0.76</td>
<td>0.83</td>
</tr>
</tbody>
</table>

Average improvement of benchmark run times
Improved Framework

- Error checking
- Robustness
- Functionality
- Performance
Conclusion

• Commercial nanopass compiler
  • Supports identical feature set
  • Modest compile-time penalty
  • 17%-28% average run-time speed-up
  • Has proved easier to extend
Nanopass Framework

- [github.com/akeep/nanopass-framework](https://github.com/akeep/nanopass-framework)
- Projects using the nanopass framework
  - Chez Scheme
  - Harlan - [github.com/eholk/harlan](https://github.com/eholk/harlan)