Enabling cross-library optimization and compile-time error checking in the presence of procedural macros

Andrew W. Keep R. Kent Dybvig

INDIANA UNIVERSITY

SCHOOL OF INFORMATICS AND COMPUTING

Bloomington

Library Groups

Andrew W. Keep R. Kent Dybvig

INDIANA UNIVERSITY
SCHOOL OF INFORMATICS AND COMPUTING
Bloomington

Goals

- Cross-library optimizations
- Type checking across library boundaries
- Single binary for multiple libraries
- Unchanged development process

3

Library Groups

- Explicitly combine libraries
- Optionally add a top-level program
- A new form: `library-group`

4

Example

```
(library (tree)
(export make-tree ---)
(import (rnrs))
(define make-tree ---)
---)
```


5

Example

```
(library (tree constants)
  (export quote-tree t0 ---)
  (import (rnrs) (tree))
  (define-syntax quote-tree
    --- (make-tree ---) ---)
  (define t0 (quote-tree))
  ---)
```


6

Example

```
(import (rnrs) (tree) (tree constants))
(define tree->list ---)
(tree->list t0)
(tree-value (tree-children t2))
(tree->list (quote-tree 5 (7 9)))
```


7

Example

```
(library-group
  (library (tree)
    (export make-tree ---)
    (import (rnrs))
    (define make-tree ---)
    ---)
  (library (tree constants)
    (export quote-tree t0 ---)
    (import (rnrs) (tree))
    (define-syntax quote-tree
      --- (make-tree ---) ---)
    (define t0 (quote-tree))
    ---)
(import (rnrs) (tree) (tree constants))
(define tree->list ---)
(tree->list t0)
(tree-value (tree-children t2))
(tree->list (quote-tree 5 (7 9))))
```


8

Example

```
(library-group
  (include “tree.sls”)
  (include “tree/constants.sls”)
  (include “app.sps”))
```


9

Library Group Syntax

```
library-group -> (library-group lglib* lgprog)
                 | (library-group lglib*)
lglib         -> library | (include filename)
lgprog        -> program | (include filename)
```


10

Challenges

- Achieving proper phasing
- Handling cyclic dependencies
- Enabling cross-library optimization/checking

11

Implementation: Libraries

- Visit code, invoke code, metadata
- Import dependencies form a DAG
- Invoke code body uses `letrec*` semantics

12

Example

```
(letrec* ([make-tree ---]
          ---)
 (set-top-level! $make-tree make-tree)
 ---)
```


13

Example

```
(letrec* ([t0 tree-constant]
          ---)
 (set-top-level! $t0 t0)
 ---)
```


14

Example

```
(letrec* ([tree->list ---])
  (tree->list $t0)
  ($tree-value ($tree-children $t2))
  (tree->list tree-constant))
```


15

Implementation: Library Groups

- Combine `letrec*` expressions
- Preserve existing library exports
- Invoke libraries needed during expansion

16

Library Group 1

```
(lambda (uid)
  (case uid
    [(tree) (letrec* ([make-tree —]
                      ---)
                ---)]
    [(constants) (letrec* ([t0 ---] ---)
                   ---)]
    [else (letrec* ([tree->list ---])
            (tree->list $t0)
            ($tree-value
              (car ($tree-children $t2)))
            (tree->list ---))])))
```


17

Library Group 1

- Advantages:
  - Single output binary
    - Matches existing library semantics
- Disadvantages:
  - Hinders cross-library optimizations

18

Library Group 2

```
(letrec* ([make-tree ---] ---)
  (set-top-level! $make-tree make-tree)
  ---
  (letrec* ([t0 tree-constant] ---)
    (set-top-level! $t0 t0)
    ---
    (letrec* ([tree->list ---])
      (tree->list $t0)
      ($tree-value
        (car ($tree-children $t2))))
      (tree->list tree-constant)))
```

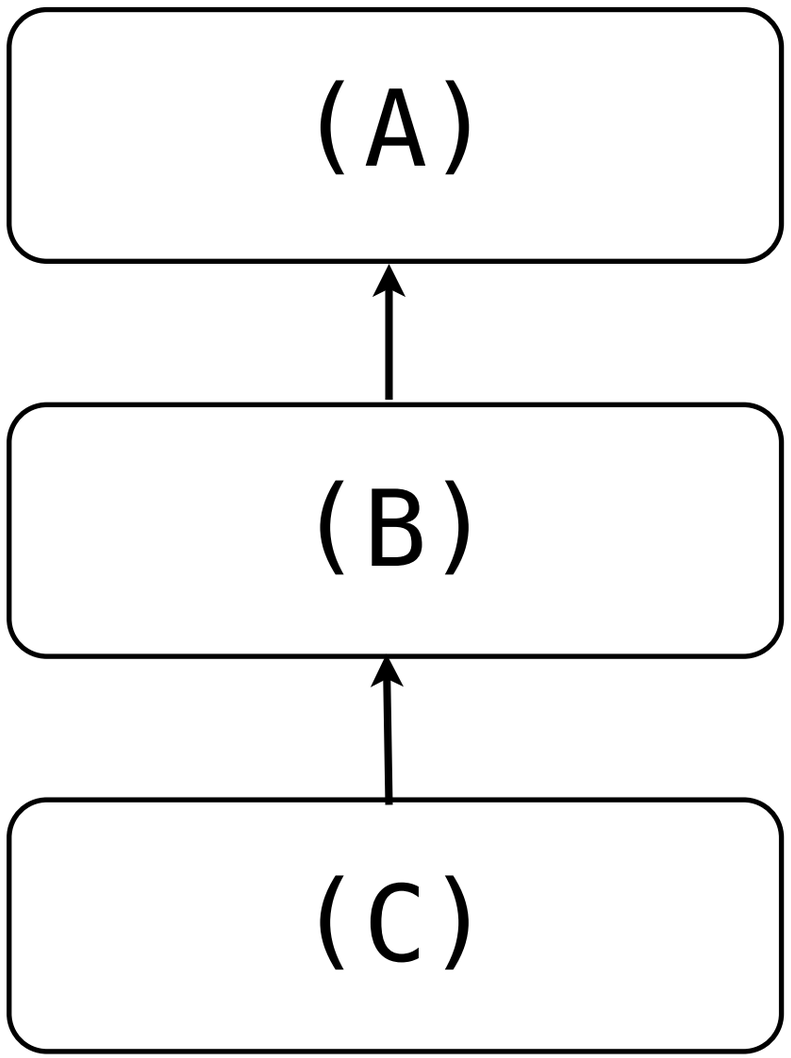

19

Library Group 2

- Advantages:
  - Creates a single invoke code
    - Allows optimizations and checking
- Disadvantage:
  - Causes dependency problems

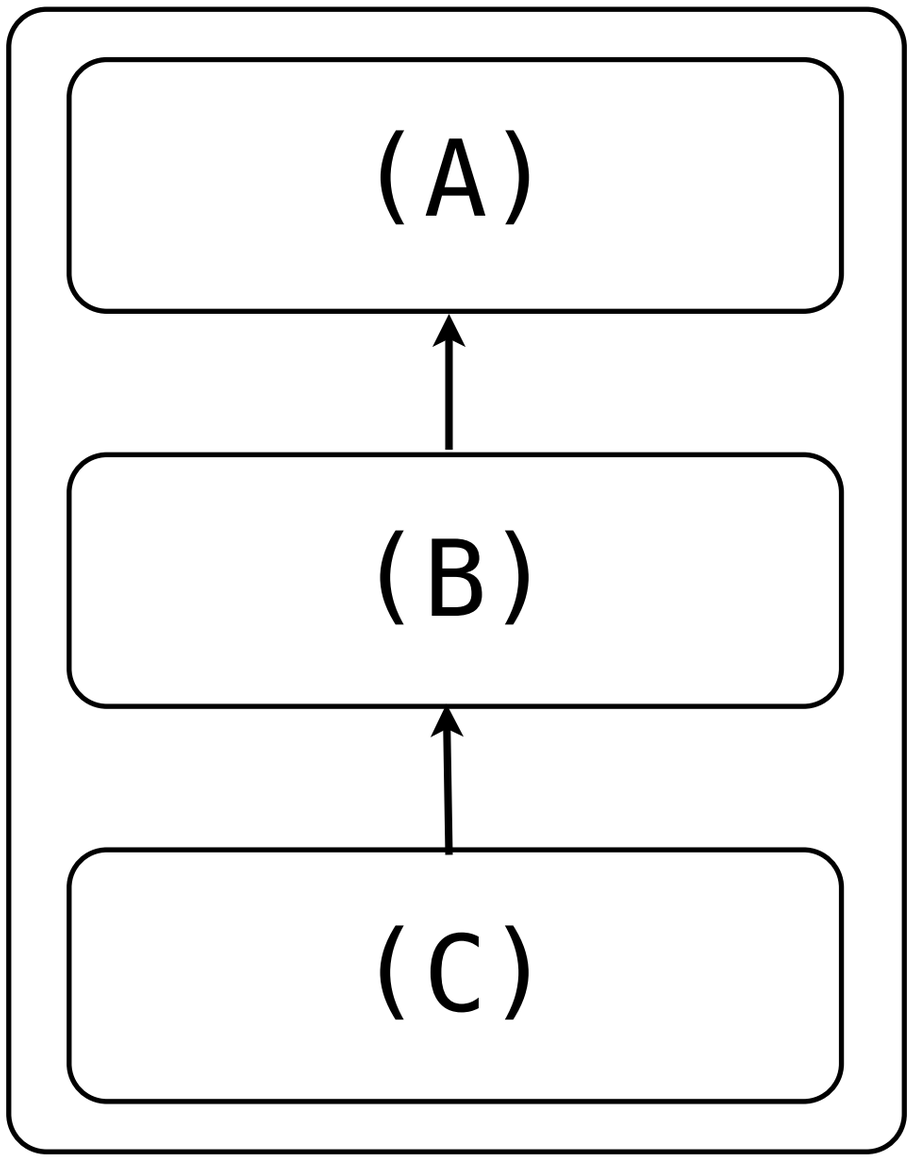
20

Dependency Problems



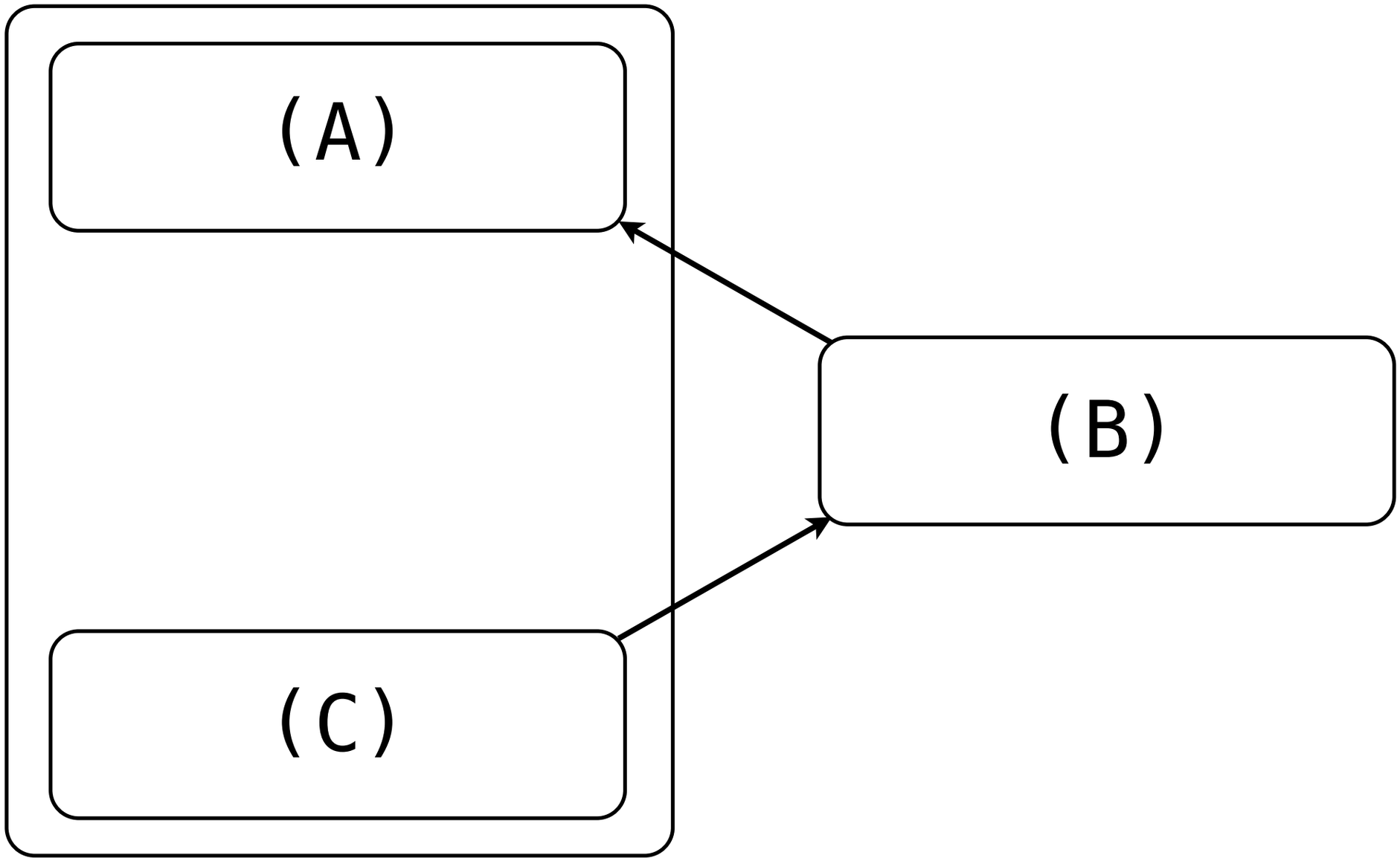
21

Dependency Problems



21

Dependency Problems



21

Library Group 3

```
(lambda (uid)
  (letrec* ([make-tree ---] ---)
    ---
    (mark-invoked! 'tree)
    (let ([nested-lib
           (lambda (uid)
             (letrec* ([t0 ---] ---)
               ---
               (mark-invoked! 'constants)
               (let ([nested-lib program code])
                 (if (eq? uid 'constants)
                     nested-lib
                     (nested-lib uid))))))])
      (if (eq? uid 'tree)
          nested-lib
          (nested-lib uid)))))
```


22

Library Group 3

- Advantages:
  - Avoids synthetic cycles
  - Allows optimization and checking
  - Single output binary

23

Caveat: Dynamic Dependencies

- Arises from use of `eval` in init expressions
- Library groups allow explicit ordering
- Work arounds
  - Transform into import dependency
  - Move into initialization function

24

Fixing Dynamic Dependencies

- Start with `case`-based `library-group`
- Lift "simple" `letrec*` bindings
- Requires `letrec*` style optimization

25

Library Phasing

- Retain phasing between libraries in group
- Cannot simply recompile from source
- Relatively straightforward solution

26

Summary

- Library groups meet our goals:
  - Cross-library optimization
    - Type checking across library boundaries
    - Single output binary
- Maintains proper phasing order
- Avoids synthetic import dependency cycles

27

Thanks

Questions?

INDIANA UNIVERSITY

SCHOOL OF INFORMATICS AND COMPUTING

Bloomington

28