# Syntax-Case

Kent Dybvig
May 2005

# Outline

Syntax binding forms

Transformers

`syntax-case` and `syntax`

`syntax-rules`

`identifier-syntax`

Built-in procedures

Expansion process

# Transformers

The RHS of a `define-syntax`, `let-syntax`, and `letrec-syntax` is a *transformer*

R5RS requires *transformer* to be a `syntax-rules` form

Change: require *transformer* to be an *expression*

A *transformer* must evaluate to a procedure of one argument

# syntax-case

Extend *expression*:

| | | |
|---|---|---|
| *expression* | $\longrightarrow$ | `(syntax-case` *exp* `(`*literal* `...)` *clause* `...)` |
| *literal* | $\longrightarrow$ | *identifier* |
| *clause* | $\longrightarrow$ | (*pattern output-expression*) |
| | $\mid$ | (*pattern fender output-expression*) |
| *fender* | $\longrightarrow$ | *expression* |

Notes:

- See TSPL3/CSUG7 for *pattern*
- Special treatment of underscore (`_`)

# syntax

Extend *expression*:

$$\begin{array}{lcl} \textit{expression} & \longrightarrow & \texttt{(syntax } \textit{template}\texttt{)} \\ & | & \texttt{\#'}\textit{template} \end{array}$$

Notes:

- See TSPL3/CSUG7 for *template*
- evaluates to an opaque `syntax-object`

# `syntax-rules`

Extend *expression*:

$$\begin{array}{lcl} \mathit{expression} & \longrightarrow & \texttt{(syntax-rules (}\mathit{literal}\ \texttt{...) }\mathit{clause}\ \texttt{...)} \\ \\ \mathit{clause} & \longrightarrow & (\mathit{pattern}\ \ \mathit{template}) \\ & \mid & (\mathit{pattern}\ \ \mathit{fender}\ \ \mathit{template}) \end{array}$$

Notes:

- straightforwardly expressed in terms of `syntax-case`

# identifier-syntax

(identifier-syntax *template*)
(identifier-syntax
  (*id* *template*)
  ((set! *id* *e*) *template*))

Notes:

- We will also have something more general

# Built-in procedures

(identifier? $obj$)

(bound-identifier? $id_1$ $id_2$)

(free-identifier? $id_1$ $id_2$)

(literal-identifier? $id_1$ $id_2$)

(literal-identifier? $id_1$ $id_2$)

(syntax-object->datum $syntax\text{-}object$)

(datum->syntax-object $identifier$ $obj$)

(generate-temporaries $ls$)

# Expansion Process

The expander is invoked once for each top-level form

When expander encounters a syntactic extension:

- invokes associated transformer
- repeats expansion process for resulting form

When expander encounters a core form:

- recursively processes subforms
- reconstructs form from expanded subforms

# Body Expansion Process

Forms processed from left to right; action based on form:

**syntactic extension:**

- invokes associated transformer
- repeats expansion process for resulting form

**variable definition:**

- identifier recorded as a variable
- expansion of rhs expression deferred

**syntax definition:**

- rhs expression expanded and evaluated
- keyword bound to resulting transformer

**begin form:**

- subforms added to list of forms to be processed

**core expression (nondefinition):**

- deferred forms expanded along with current and remaining form

# As yet unspecified

Ability to access to expand-time environment

Ability to add arbitrary bindings to expand-time environment

Meta definitions

Support for generalized identifier syntax

`syntax->list` operator