APPENDIX A

BNF

General Notes

Dylan syntax can be parsed with an LALR(1) grammar.

This appendix uses some special notation to make the presentation of the grammar more readable.

- The $_{opt}$ suffix means that the preceding item is optional.
- A trailing ellipsis (...) is used in two different ways to signal possible repetition.
  - If there is only one item on the line preceding the ellipsis, the item may appear one or more times.
  - If more than one item precedes the ellipsis, the last of these items is designated a separator; the rest may appear one or more times, with the separator appearing after each occurrence but the last. (When only one item appears, the separator does not appear.)
- Identifiers for grammar rules are written with uppercase letters when the identifier is used in the phrase grammar but defined in the lexical grammar.
- The grammar does not use distinct identifiers for grammar rules that differ only in alphabetic case.

In the following grammar, some tokens are used multiple ways. For example - is punctuation, a unary operator, and a binary operator, and `method` is a BEGIN-WORD and a DEFINE-BODY-WORD. In some parsing implementations such multiple meanings of a token may not be possible. However this is just an implementation issue since the meaning of the grammar is clear. `method` is used as punctuation in *local-methods* and *method-definition*; since `method` is not a core reserved word, this typically has to be implemented by accepting any MACRO-NAME and checking semantically that the word used is "`method`." The grammar as presented is not obviously LALR(1), since the required changes would tend to obscure the readability for human beings (especially in macro

definitions and case-body). The grammar can be made LALR(1) through well-known standard transformations implemented by most parser generators.

Lexical Notes

In the lexical grammar, the various elements that come together to form a single token on the right-hand sides of rules must *not* be separated by white-space, so that the end result will be a single token. This is in contrast to the phrase grammar, where each element is already a complete token or a series of complete tokens.

Arbitrary white-space is permitted between tokens, but it is required only as necessary to separate tokens that might otherwise blend together.

Case is not significant except within character and string literals. The grammars do not reflect this, using one case or the other, but it is still true.

Lexical Grammar

Comments

comment:
    // …the rest of the line
    /* …everything even across lines, including nested comments… ***/**

Tokens

TOKEN:
    NAME
    SYMBOL
    NUMBER
    CHARACTER-LITERAL
    STRING
    UNARY-OPERATOR
    BINARY-OPERATOR

APPENDIX A

BNF

punctuation
#-word

punctuation:
one of **( ) , . ; [ ] { } :: - = == =>**
one of **#(#[## ? ?? ?= ...**

#-word:
one of **#t #f #next #rest #key #all-keys #include**

Reserved Words

reserved-word:
core-word
BEGIN-WORD
FUNCTION-WORD
DEFINE-BODY-WORD
DEFINE-LIST-WORD

core-word:
one of **define end handler let local macro otherwise**

The following reserved words are exported by the Dylan module:

BEGIN-WORD:
one of **begin block case for if method**
one of **select unless until while**

FUNCTION-WORD:
(none)

DEFINE-BODY-WORD:
one of **class library method module**

DEFINE-LIST-WORD:
one of **constant variable domain**

Names, Symbols and Keywords

NAME:
word

\ *word*
operator-name

UNRESERVED-NAME:
any *word* that is not also a *reserved-word*
\ *word*
operator-name

ORDINARY-NAME:
UNRESERVED-NAME
DEFINE-BODY-WORD
DEFINE-LIST-WORD

CONSTRAINED-NAME:
NAME **:** *word*
NAME **:** BINARY-OPERATOR
: *word*

operator-name:
\ UNARY-OPERATOR
\ BINARY-OPERATOR

MACRO-NAME:
ORDINARY-NAME
BEGIN-WORD
FUNCTION-WORD

NAME-NOT-END:
MACRO-NAME
one of **`define handler let local macro otherwise`**

SYMBOL:
*word***`:`**
STRING

word:
leading-alphabetic
leading-numeric alphabetic-character leading-alphabetic
leading-graphic leading-alphabetic

leading-alphabetic:
alphabetic-character
leading-alphabetic any-character

leading-numeric:
    numeric-character
    leading-numeric word-character-not-double-alphabetic

leading-graphic:
    graphic-character
    leading-graphic word-character-not-alphabetic

word-character-not-alphabetic:
    numeric-character
    graphic-character
    special-character

word-character-not-double-alphabetic:
    alphabetic-character word-character-not-alphabetic
    numeric-character
    graphic-character
    special-character

any-character:
    alphabetic-character
    numeric-character
    graphic-character
    special-character

alphabetic-character:
    one of **a b c d e f g h i j k l m n o p q r s t u v w x y z**

numeric-character:
    one of **0 1 2 3 4 5 6 7 8 9**

graphic-character:
    one of **! & * < > | ^ $ % @ _**

special-character:
    one of **- + ~ ? / =**

Operators

UNARY-OPERATOR:
    one of **- ~**

BINARY-OPERATOR:
    one of **+ - * / ^ = == ~= ~== < <= > >= & | :=**

APPENDIX A

BNF

Character and String Literals

CHARACTER-LITERAL:
    ' *character* **'**

character:
    any printing character (including space) except for **'** or ****
    **** *escape-character*

STRING*:*
    " *more-string*

more-string:
    string-character more-string
    "

string-character:
    any printing character (including space) except for **"** or ****
    **** *escape-character*

escape-character:
    one of **\ ' " a b e f n r t 0**
    < *hex-digits* **>**

Numbers

NUMBER:
    integer
    ratio
    floating-point

integer:
    binary-integer
    octal-integer
    $sign_{opt}$ *decimal-integer*
    hex-integer

binary-integer:
    #b *binary-digit*
    binary-integer binary-digit

APPENDIX A

BNF

octal-integer:
    #o *octal-digit*
    octal-integer octal-digit

decimal-integer:
    decimal-digit
    decimal-integer decimal-digit

hex-integer:
    #x *hex-digit*
    hex-integer hex-digit

hex-digits:
    hex-digit ...

binary-digit:
    one of **0 1**

octal-digit:
    one of **0 1 2 3 4 5 6 7**

decimal-digit:
    one of **0 1 2 3 4 5 6 7 8 9**

hex-digit:
    one of **0 1 2 3 4 5 6 7 8 9 A B C D E F**

ratio:
    $sign_{opt}$ *decimal-integer* **/** *decimal-integer*

floating-point:
    $sign_{opt}$ $decimal\text{-}integer_{opt}$ **.** *decimal-integer* $exponent_{opt}$
    $sign_{opt}$ *decimal-integer* **.** $decimal\text{-}integer_{opt}$ $exponent_{opt}$
    $sign_{opt}$ *decimal-integer exponent*

exponent:
    E $sign_{opt}$ *decimal-integer*

sign:
    one of **+ -**

APPENDIX A

BNF

Phrase Grammar

Program Structure

source-record:
    body$_{opt}$

body:
    constituents **;**$_{opt}$

constituents:
    constituent **;** …

constituent:
    definition
    local-declaration
    expression

macro:
    definition-macro-call
    statement
    function-macro-call

    parsed-macro-call

Property Lists

comma-property-list:
    , *property-list*

property-list:
    property **,** …

property:
    SYMBOL *value*

value:
    basic-fragment

APPENDIX A

BNF

Fragments

body-fragment:
    non-statement-body-fragment
    statement *non-statement-body-fragment*$_{opt}$

list-fragment:
    non-statement-list-fragment
    statement *non-statement-list-fragment*$_{opt}$

basic-fragment:
    non-statement-basic-fragment
    statement *non-statement-basic-fragment*$_{opt}$

non-statement-body-fragment:
    definition *semicolon-fragment*$_{opt}$
    local-declaration *semicolon-fragment*$_{opt}$
    simple-fragment *body-fragment*$_{opt}$
    `,` *body-fragment*$_{opt}$
    `;` *body-fragment*$_{opt}$

semicolon-fragment:
    `;` *body-fragment*$_{opt}$

non-statement-list-fragment:
    simple-fragment *list-fragment*$_{opt}$
    `,` *list-fragment*$_{opt}$

non-statement-basic-fragment:
    simple-fragment *basic-fragment*$_{opt}$

simple-fragment:
    variable-name
    constant-fragment
    BINARY-OPERATOR
    UNARY-OPERATOR
    bracketed-fragment
    function-macro-call
    #-word
    one of `. :: => ? ?? ?= ... ## otherwise`
    parsed-function-call
    parsed-macro-call

APPENDIX A

BNF

bracketed-fragment:
- **(** *body-fragment*$_{opt}$ **)**
- **[** *body-fragment*$_{opt}$ **]**
- **{** *body-fragment*$_{opt}$ **}**

constant-fragment:
- NUMBER
- CHARACTER-LITERAL
- STRING
- SYMBOL
- **#(** *constants* **.** *constant* **)**
- **#(** *constants*$_{opt}$ **)**
- **#[** *constants*$_{opt}$ **]**
- *parsed-list-constant*
- *parsed-vector-constant*

Definitions

definition:
- *definition-macro-call*
- **define macro** *macro-definition*
- *parsed-definition*

definition-macro-call:
- **define** *modifiers*$_{opt}$ DEFINE-BODY-WORD *body-fragment*$_{opt}$ *definition-tail*
- **define** *modifiers*$_{opt}$ DEFINE-LIST-WORD *list-fragment*$_{opt}$

modifier:
- UNRESERVED-NAME

modifiers:
- *modifier* …

definition-tail:
- **end**
- **end** MACRO-NAME
- **end** DEFINE-BODY-WORD MACRO-NAME

Local Declarations

local-declaration:
    let *bindings*
    let handler *condition* **=** *handler*
    local *local-methods*
    parsed-local-declaration

condition:
    type
    (*type comma-property-list* **)**

handler:
    expression

local-methods:
    method$_{opt}$ *method-definition* **,** …

bindings:
    variable **=** *expression*
    (*variable-list* **) =** *expression*

variable-list:
    variables
    variables **, #rest** *variable-name*
    #rest *variable-name*

variables:
    variable **,** …

variable:
    variable-name
    variable-name **::** *type*

variable-name:
    ORDINARY-NAME

type:
    operand

APPENDIX A

BNF

Expressions

expressions:
    expression **,** …

expression:
    binary-operand BINARY-OPERATOR …

expression-no-symbol:
    binary-operand-no-symbol
    binary-operand-no-symbol BINARY-OPERATOR *expression*

binary-operand-no-symbol:
    UNARY-OPERATOR$_{opt}$ *operand*

binary-operand:
    SYMBOL
    UNARY-OPERATOR$_{opt}$ *operand*

operand:
    operand **(** *arguments*$_{opt}$ **)**
    operand **[** *arguments*$_{opt}$ **]**
    operand **.** *variable-name*
    leaf

function-macro-call:
    FUNCTION-WORD **(** *body-fragment*$_{opt}$ **)**

leaf:
    literal
    variable-name
    (*expression* **)**
    function-macro-call
    statement
    parsed-function-call
    parsed-macro-call

arguments:
    argument **,** …

argument:
    SYMBOL *expression*
    expression-no-symbol
    SYMBOL

APPENDIX A

BNF

literal:
NUMBER
CHARACTER-LITERAL
string-literal
#t
#f
#(*constants* **.** *constant* **)**
#($constants_{opt}$ **)**
#[$constants_{opt}$ **]**
parsed-list-constant
parsed-vector-constant

string-literal:
STRING …

constants:
constant **,** …

constant:
literal
SYMBOL

Statements

statement:
BEGIN-WORD $body\text{-}fragment_{opt}$ *end-clause*

end-clause:
end BEGIN-WORD_{opt}

case-body:
cases $;_{opt}$

cases:
case-label $constituents_{opt}$ **;** …

case-label:
expressions **=>**
(*expressions* **,** *expressions* **) =>**
otherwise =>$_{opt}$

Methods

method-definition:
    variable-name parameter-list $body_{opt}$ **end method**$_{opt}$ $variable\text{-}name_{opt}$

parameter-list :
    ($parameters_{opt}$ **)** **;**$_{opt}$
    ($parameters_{opt}$ **)** **=>** *variable* **;**
    ($parameters_{opt}$ **)** **=>** **(** $values\text{-}list_{opt}$ **)** **;**$_{opt}$

parameters:
    required-parameters
    required-parameters **,** *next-rest-key-parameter-list*
    next-rest-key-parameter-list

next-rest-key-parameter-list:
    #next *variable-name*
    #next *variable-name* **,** *rest-key-parameter-list*
    rest-key-parameter-list

rest-key-parameter-list:
    #rest *variable-name*
    #rest *variable-name* **,** *key-parameter-list*
    key-parameter-list

key-parameter-list:
    #key $keyword\text{-}parameters_{opt}$
    #key $keyword\text{-}parameters_{opt}$ **, #all-keys**

required-parameters:
    required-parameter **,** …

required-parameter:
    variable
    variable-name **==** *expression*

keyword-parameters:
    keyword-parameter **,** …

keyword-parameter:
    SYMBOL$_{opt}$ *variable* $default_{opt}$

default:
    = *expression*

values-list:
    variables
    variables **, #rest** *variable*
    #rest *variable*

Macro Definitions

macro-definition:
    MACRO-NAME *main-rule-set aux-rule-sets*$_{opt}$ **end macro**$_{opt}$ MACRO-NAME$_{opt}$

main-rule-set:
    body-style-definition-rule ...
    list-style-definition-rule ...
    statement-rule ...
    function-rule ...

body-style-definition-rule:
    { define *definition-head*$_{opt}$ MACRO-NAME *pattern*$_{opt}$ **;**$_{opt}$ **end } =>** *rhs*

list-style-definition-rule:
    { define *definition-head*$_{opt}$ MACRO-NAME *pattern*$_{opt}$ **} =>** *rhs*

rhs:
    { *template*$_{opt}$ **};**$_{opt}$

definition-head:
    modifier-pattern ...

modifier-pattern:
    modifier
    pattern-variable

statement-rule:
    { MACRO-NAME *pattern*$_{opt}$ **;**$_{opt}$ **end } =>** *rhs*

function-rule:
    { MACRO-NAME **(** *pattern*$_{opt}$ **) } =>** *rhs*

Patterns

pattern:
    pattern-list **;** *...*

APPENDIX A

BNF

pattern-list:
    pattern-sequence
    property-list-pattern
    pattern-sequence **,** *pattern-list*

pattern-sequence:
    simple-pattern ...

simple-pattern:
    NAME-NOT-END
    =>
    bracketed-pattern
    binding-pattern
    pattern-variable

bracketed-pattern:
    ($pattern_{opt}$ **)**
    [$pattern_{opt}$ **]**
    { $pattern_{opt}$ **}**

binding-pattern:
    pattern-variable **::** *pattern-variable*
    pattern-variable **=** *pattern-variable*
    pattern-variable **::** *pattern-variable* **=** *pattern-variable*

pattern-variable:
    ? NAME
    ? CONSTRAINED-NAME
    ...

property-list-pattern:
    #rest *pattern-variable*
    #key $pattern\text{-}keywords_{opt}$
    #rest *pattern-variable* **, #key** $pattern\text{-}keywords_{opt}$

pattern-keywords:
    #all-keys
    pattern-keyword
    pattern-keyword **,** *pattern-keywords*

pattern-keyword:
    ? NAME $default_{opt}$
    ? CONSTRAINED-NAME $default_{opt}$

?? NAME *default*$_{opt}$
?? CONSTRAINED-NAME *default*$_{opt}$

Templates

template:
template-element ...

template-element:
NAME
SYMBOL
NUMBER
CHARACTER-LITERAL
STRING
UNARY-OPERATOR
separator
#-word
one of **. :: =>**
(*template*$_{opt}$ **)**
[*template*$_{opt}$ **]**
{ *template*$_{opt}$ **}**
#(*template*$_{opt}$ **)**
#[*template*$_{opt}$ **]**
parsed-list-constant
parsed-vector-constant
substitution

separator:
one of **; ,**
BINARY-OPERATOR

substitution:
name-prefix$_{opt}$ **?** *name-string-or-symbol name-suffix*$_{opt}$
?? NAME *separator*$_{opt}$ **...**
...
?= NAME

name-prefix:
STRING **##**

name-suffix:
STRING

APPENDIX A

BNF

name-string-or-symbol:
NAME
STRING
SYMBOL

Auxiliary Rule Sets

aux-rule-sets:
aux-rule-set ...

aux-rule-set:
SYMBOL *aux-rules*

aux-rules:
aux-rule ...

aux-rule:
{ $pattern_{opt}$ **}** **=>** *rhs*

Parsed Fragments

parsed-definition:
(no external representation)

parsed-local-declaration:
(no external representation)

parsed-function-call:
(no external representation)

parsed-macro-call:
(no external representation)

parsed-list-constant:
(no external representation)

parsed-vector-constant:
(no external representation)