Appendix A: Adapt BNF

The following is a complete definition of the Adapt language in the Extended Backus-Naur form. Square brackets enclose optional items, braces enclose items that may be repeated zero or more times, and vertical bars separate alternatives. These special symbols do not have their special meaning when they are enclosed in single quotes.

program ::= procedure { procedure }

procedure ::= *proc_decl* IS *var_decl* *adapt_body* END [*id*] ;
| *proc_decl* [IS *var_decl* BEGIN *stmts* END [*id*]] ;
| *func_decl* [IS *var_decl* BEGIN *stmts* END [*id*]] ;

proc_decl ::= PROCEDURE *id* *proc_args*

func_decl ::= FUNCTION *id* *proc_args* RETURN *elementary_type*

adapt_body ::= [*initial_sect*] *raster_sect* [*combining_sect*] [*final_sect*]

proc_args ::= ([{ *arg_decl* , } *arg_decl*])

arg_decl ::= *arg_list* : *arg_type*

arg_list ::= { *id* , } *id*

arg_type ::= IN [TRANSPOSED] IMAGE *type*
[BORDER *border_style*]
[SAMPLE (*sample_style*)]
| OUT [TRANSPOSED] IMAGE *type*
| [IN | OUT | IN OUT] *type*

type ::= [ARRAY (*dimension_range_list*) OF] *elementary_type*

elementary_type ::= SIGNED_BYTE | UNSIGNED_BYTE | BYTE |
SHORT | UNSIGNED_SHORT |
SIGNED_INTEGER | UNSIGNED_INTEGER | INTEGER |
FLOAT | LONG_FLOAT

dimensions_range_list ::= { [*range*] , } *range*

range ::= *expr* .. *expr*

border_style ::= *expr* | MIRRORED | REPLICATED

sample_style ::= { *expr* , } *expr*

var_decl ::= { [*var_info*] ; } [*var_info*]

var_info ::= { *id* , } *id* : *type*

initial_sect ::= FIRST *var_decl* BEGIN *stmts* END FIRST ;

raster_sect ::= NEXT *var_decl* BEGIN *stmts* END NEXT ;

combining_sect ::= COMBINE *var_decl* BEGIN *stmts* END COMBINE ;

final_sect ::= LAST *var_decl* BEGIN *stmts* END LAST ;

stmts ::= { *statement* ; } *statement* ;

statement ::= *assign* | *if* | *loop* | *exit* | *case* | *call* | *return*

assign ::= [_]*scalar_var* : *expr*

scalar_var ::= *id* [(*args*)]

call ::= *id* (*args*)

args ::= { *expr* , } *expr*

if ::= IF *bool_expr* THEN *stmts* { *elsif* } [ELSE *stmts*] END IF

elsif ::= ELSIF *bool_expr* THEN *stmts*

loop ::= [*id* :] [*loop_decl*] LOOP *stmts* END LOOP

loop_decl ::= FOR *id* IN [REVERSE] *expr* .. *expr*
| WHILE *bool_expr*

exit ::= EXIT [*id*] WHEN *bool_expr*

case ::= CASE *expr* IS { *when_clause* } [*when_other*] END CASE

when_clause ::= WHEN { *expr* '|' } *expr* => *stmts*

when_other ::= WHEN OTHERS => *stmts*

return ::= RETURN *expr*

bool_expr ::= *conjunction* | *disjunction* | *simple_bool_expr*

simple_bool_expr ::= { NOT } (*bool_expr*)
| { NOT } *expr* *bool_op* *expr*

bool_op ::= >= | <= | /= | = | < | >

conjunction ::= *simple_bool_expr* AND [THEN] *conjunction*
| *simple_bool_expr* AND [THEN] *simple_bool_expr*

disjunction ::= *simple_bool_expr* OR [ELSE] *disjunction*
| *simple_bool_expr* OR [ELSE] *simple_bool_expr*

expr	::=	*expr* (+ | – | * | / | MOD | REM | **) *expr*
	|	*expr* (LAND | LOR | LXOR | RSH | LSH) *expr*
	|	(*expr*)
	|	(– | LNOT | *type*) *expr*
	|	MAX (*expr* , *expr*)
	|	MIN (*expr* , *expr*)
	|	[–] *inum*
	|	' *char* '
	|	*fnum*
	|	*scalar_var*
	|	*_call*
	|	*_id*
id	::=	*letter*[{_ | *letter* | *digit*}[*letter* | *digit*]
fnum	::=	[–][*inum*].*inum*[e[[+|–]]*inum*]
	|	[–]*inum*[.*inum*][e[[+|–]]*inum*]
inum	::=	*digit*{*digit*}
char	::=	any character except '
digit	::=	0|1|2|3|4|5|6|7|8|9
letter	::=	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| 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