Specification Of The
Array Semantics For Sisal 2.0

Yung-Syau Chen and Jean-Luc Gaudiot

CENG Technical Report 94-28

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California 90089-2562
(213)740-4484

July 1994
Specification of the Array Semantics for Sisal 2.0

Jean-Luc Gaudiot and Yung-Syau Chen

CENG Technical Report 94-28

Computer Engineering Division
Electrical Engineering - System Department
University of Southern California
Los Angeles, CA 90089-2562

July 1994
Abstract

We discuss in detail three important facilities of arrays in Sisal 2.0: array generation, reference to, and update of subarrays. We specify the semantics of them with Typol inference rules in the Centaur system.
In program *sisal* expression, three rules recognize the expressions for array generation, array reference, and array update according to the operators *array.gen*, *stream_ref_or_array_ref*, and *array.gen*:

```
array_generate(System, Env, Type_SPEC|- SIZE_DESCR_LIST, ARRAY_PART_LIST->VALUES)
-----------------------------------------------------------------------------
System, Env |- array_gen(Type, SIZE_DESCR_LIST, ARRAY_PART_LIST) : VALUES;
array_reference(System, Env |- ArrayName, ReferIndex -> VALUES)
-----------------------------------------------------------------------------------------------
System, Env |- stream_ref_or_array_ref(ArrayName, ReferIndex) : VALUES;
array_update(System, Env |- EXPRESSION, UPDATE_PART_LIST -> VALUES)
-----------------------------------------------------------------------------------------------
System, Env |- array_update(EXPRESSION, UPDATE_PART_LIST) : VALUES;
```

The semantics of array generation, array reference and array generate are described as follows.

1. The semantics of array generation can be written as follows:

```
build_array(System,Env,TYPE_SPEC|- SIZE_DESCR_LIST,ARRAY_PARTS->DEFARRAY,DIM) &
fill_array(System,Env,DEFARRAY|-ARRAY_PARTS->ARRAY')
-----------------------------------------------------------------------------------------------
System,Env,TYPE_SPEC|- SIZE_DESCR_LIST, ARRAY_PARTS-> values[ARRAY'];
```

First, we build the array with set *build.array*, specified as:

```
judgement System,Env,TYPE_SPEC|- SIZE_DESCR_LIST,ARRAY_PART_LIST->VALUE,integer;
otherwise(System, Env,TYPE_SPEC|- ARRAY_PART_LIST:VALUE) &
build(System,Env,0,VALUE|- SIZE_DESCR_LIST -> ARRAY,DIM)
-----------------------------------------------------------------------------------------------
System, Env, TYPE_SPEC |- SIZE_DESCR_LIST,ARRAY_PART_LIST -> ARRAY,DIM;
```

The first premise (*otherwise*) returns a default value specified in the otherwise placement.

If *SIZE_DESCR_LIST* is not empty, set *build* recognizes the dimension and the boundaries from the size description and then builds the array structure (*DEFARRAY*) with the *otherwise* value. If *SIZE_DESCR_LIST* is empty (e.g., array integer [10, 20, 30, 40]) set *build* uses expression_list to compute the number of array elements and then builds the array structure.

Second, set *fill_array* examines each array_part in sequence. It is specified as follows:

```
1
```
judgement SYSTEM,ENV,integer,VALUE| - ARRAY_PART_LIST->VALUE;

modify_array_with_array_part(SYSTEM,ENV,ARRAY_PART_COUNT,ARRAY| -ARRAY_PART->ARRAY')
 & plus1(ARRAY_PART_COUNT,ARRAY_PART_COUNT')
 & SYSTEM,ENV,ARRAY_PART_COUNT',ARRAY'| - ARRAY_PARTS->ARRAY''
-----------------------------------------------------------------------------
SYSTEM,ENV,ARRAY_PART_COUNT,ARRAY| - array_part_list[ARRAY_PART.ARRAY_PARTS]-> ARRAY'';

Variable ARRAY_PART_COUNT is used in case that we need to know the order of the array part in order to put the content in a proper position. For example, assume we are processing an array generation expression like:

A:= array integer [2..3, 7..8: 10, 20; 30, 40];;

Since set fill_array separately examines each array_part in sequence, we have to record that the processed array part. Set modify_array_with_array_part, called with variable ARRAY_PART_COUNT, examine one ARRAY_PART and returns the filled array. The set is specified as:

judgement SYSTEM,ENV,integer,VALUE| - ARRAY_PART->VALUE;

modify_array_with_opt_placement(SYSTEM,ENV,APC| -
 OPT_PLACEMENT,EXPRESSION_LIST,ARRAY->ARRAY')
-----------------------------------------------------------------------------
SYSTEM,ENV,APC,ARRAY| - array_part(OPT_PLACEMENT,EXPRESSION_LIST)->ARRAY'';

In this rule, set modify_array_with_opt_placement examines the placement son (OPT_PLACEMENT) and decides whether it is a no_placement().

(a) If the placement son is no_placement(), then the expression_list son is examined by set modify_array_with_no_placement, specified as:

judgement SYSTEM,ENV,integer| - EXPRESSION_LIST,VALUE->VALUE;

elt_to_elts(<- ARRAY->D,L,U,ELTS)
 & modify_elts_by_exp_list(SYSTEM,ENV,APC,L| -EXPRESSION_LIST,ELTS-> ELTS')
 & elts_to_elts(D,L,U,ELTS'->ARRAY')
-----------------------------------------------------------------------------
SYSTEM,ENV,APC| - EXPRESSION_LIST,ARRAY->ARRAY'';

The dimension, lower, upper and elements are extracted from the initial array (ARRAY); ELTS is modified by the values contained in the
EXPRESSION_LIST. Then the dimension, lower, upper and the modified ELTS together construct the new array (ARRAY') after processor the ARRAY_PART.

(b) If OPT PLACEMENT is a PLACEMENT with contents, then the EXPRESSION_LIST is handled by set modify_array_with_no_placement, specified as:

\[\text{extract_pattern_values(SYSTEM,ENV|\text{--} \text{PLACEMENT,EXPS \rightarrow PLIST,PATTERN,VALUES,ELTS}}\]
\& \text{update(ELTS|\text{--}PLIST,PATTERN,VALUES,ARRAY--\rightarrow ARRAY')}\]

\[\text{SYSTEM,ENV|\text{--} \text{PLACEMENT,EXPS,ARRAY-- \rightarrow ARRAY'}};\]

set extract_pattern_values distinguish the pattern of the array part by the selector_part_list inside the PLACEMENT, and get the values from the expression. There are three pattern:

i. If the indexes of all dimension are specified then the array part is pattern 1.
   For example, in an array generation expression like:
   \[A := \text{array real } [2..3, \ldots 8; [2,7] 1.0; [2,8] 2.0; \]
   \[3,\ldots ] 3.0; \]
   \[\text{otherwise} 0.0); \]
   \[3,1] 3.0; [3,2] 4.0];\]

Array part [2,7] 1.0 is pattern 1.

ii. If the indexes are only partially specified, then the array part is pattern 2. For example, in the previous array generation expression, Array part [3,..] 3.0 is pattern 2.

iii. If there is an index array in the selector_part_list then this array part is pattern 3.
   For example, in an array generation expression like:
   \[\text{array real } [1..3,1..4; [1,U] 11.1, 14.4, 12.2; \]
   \[2,U] 21.1, 24.4, 22.2; \]
   \[\text{otherwise} 99.9],\]

Array part [1,U] 11.1, 14.4, 12.2 is pattern 3.
Then, set update, called with these pattern and values, returns the final array.

2. The semantics of array reference can be written as follows:

\[\text{search_in_env(ArrName |- ENV: Array) \&} \]
\[\text{ExtractArrElementsBy1Ref(Ref1 |- Array \rightarrow arrayElement)}\]

\[\text{SYSTEM,ENV|- ArrName, int_const Ref1 \rightarrow values[arrayElement]];\]
search_in_env(ArrName | ENV: Array) &
eval_expression(SYSTEM, ENV | LOWER_EXP : values[int_const Ref1])
& eval_expression(SYSTEM, ENV | UPPER_EXP : values[int_const Ref2])
& ExtractArrElementsBy2Ref(Ref1, Ref2 | Array -> PartialArr)

SYSTEM, ENV | ArrName, triplet(LOWER_EXP, UPPER_EXP, ME) -> values[PartialArr];

search_in_env(ArrName | ENV: Array) &
search_in_env(IndexArrName | ENV: IndexArray) &
getIndexELTS( | IndexArray -> IndexELTS) &
modifyArrByIndexELTS(IndexELTS | Array -> PartialArr)

SYSTEM, ENV | ArrName, IndexArrName -> values[PartialArr];

Arrays can be referenced by an index integer(A1[3]), by a index triple
(A1[3..4]), or by an index integer array (A[U]). The type are distinguished
by the above 3 rules.

(a) For type 1 expression, set ExtractArrElementsBy1Ref (called in the
first rule,) called with the index, returns the referred array part.

(b) For type 2 expression: set ExtractArrElementsBy2Ref (called in the
second rule,) called with the lower and upper bound extracted from
the index triple, returns the referred array part.

(c) For type 3 expression: set modifyArrByIndexELTS (called in the
third rule,) called with index array, returns the referred array part.

3. The semantics of array update can be written as follows:

search_in_env(NAME | ENV: ARRAY)
& update_this_array(SYSTEM, ENV | ARRAY, UPDATE_PART_LIST -> ARRAY)

SYSTEM, ENV | NAME, UPDATE_PART_LIST -> values[ARRAY];

First, according to the array name, the content of the array (ARRAY)
is extracted. Then set update_this_array, is called with ARRAY and UP-
DATE_PART_LIST.

Each update part in the UPDATE_PART_LIST is examined in sequence by set
update_this_array. Called in set update_this_array, set update_any_dim_array
examines one update part. This set defined as:

judgement SYSTEM, ENV | UPDATE_PART, VALUE -> VALUE;

eval_expression_list(SYSTEM, ENV | EXPRESSION_LIST -> VALUES)
& process_selector(SYSTEM,ENV | SELECTOR, VALUES, ARRAY -> ARRAY')

SYSTEM,ENV| update_part(SELECTOR,EXPRESSION_LIST),ARRAY->ARRAY';

From the expression_list son, a list of values is generated. Then set process_selector, called with these values and the selector son, returns the updated array.