1.0 SVEC 3.1a Action Item 11 - ERR-4

Disposition of automatic variables in the presence of:

```
fork/join_none
fork/join_any
```

SV 3.1 added the fork/join_none and fork/join_any constructs.

It has since been discovered that there is a hole in the SV 3.1 specification with regards to automatic variables that are used by sub-processes spawned by fork/join_none and fork/join_any. These automatic variables may cease to exist while sub-processes that use them continue to exist.

Various proposals and ideas for dealing with this issue have been circulated on the reflector and discussed in the SVEC bi-weekly meetings. This email contains a summary of the various issues and a set of options for dealing with each of them. A list of pros and cons is given for each proposed option.

1.1 Example of the problem

This example has since been referred to as the "looping problem".

The statements spawned by join_none can potentially outlive the return of the containing task. It doesn't matter whether the fork/join_none is contained within a task that is automatic or static, or even inside a task at all. It's the automatic looping variable j that can cause problems.

Code Example 1-1 Looping Problem

```
initial
for(int j=1; j<=3; ++j)
   fork
      #1 $display( "j=%0d", j);
join_none
```

The values of j used within the calls to $display could be undefined since the automatic variable j may go out of scope before any of the sub-processes have had a chance to start.

In this example the intended behavior would be for each $display to receive a different value of j (e.g. 1,2,3).

1.2 There are 3 main points to consider in this situation.

- Sharing of automatic variables

  Should sub-processes spawned by fork/join_any and fork/join_none be allowed to share automatic variables that are contained within the scope which spawns those sub-processes?

- Lifetime of automatic variables

  When should the automatic variables within the containing scope be allowed to cease to exist?

- Local copies of automatic variables

  Should local copies of automatic variables be passed to sub-processes spawned by fork/join_none and fork/join_any?

1.3 Example of user-specified local copies of automatic variables

The following examples show two potential ways for a set of automatic variables to be made available to sub-processes.

Code Example 1-2 Local copies of automatic variables

The fork specifies the local variables

```
fork (a,b,c,d)  // list of vars
   begin
      ...
   end
join_none
```

A local automatic variable is specified

```
int j;
...
fork
   begin
      automatic int a=j;
      ...
   end
join_none
```

The first alternative was suggested in one of the SVEC conference calls. In this approach the fork statement is modified to allow a set of arguments to be specified. Each of the arguments specifies the name of an automatic variable that is currently defined within the local scope. Specifying the name of an existing automatic variable as an argument to the fork statement causes a copy of that automatic variable to be created within the spawned sub-process. The automatic variable created within the sub-process is initialized to the value of the currently existing automatic variable when the sub-process is created.

The second alternative uses the declaration of an automatic variable local to the spawned sub-processes. These local variables are created and initialized when the sub-processes get created. This approach is consistent with V2K (section 10.2.3). This alternative solves the looping problem using an existing V2K construct.

1.4 Lifetime of automatic variables

A point that was not raised in previous discussions within the SVEC is that the information contained in the activation record enclosing the fork...join is needed to implement the “disable fork” and “wait fork” constructs. This requires that the activation record must be kept around until all sub-processes forked by a task terminate regardless of the lifetime of the task itself.

Requiring the lifetime of automatic variables contained within sub-processes to exist until the sub-process terminates appears to be consistent with this. This interpretation specifies the notion of a re-entrant process.

1.5 Voting

Instead of voting on the individual points and thus potentially ending up with an inconsistent set of rules it would be preferable to vote for one of the following 7 scenarios.

Scenarios:

1 1.a
2 1.b
3 1.c

4 2.a.1
5 2.a.2

6 2.b.1
7 2.b.2

1. Sharing of automatic variables is disallowed for join_none and join_any

   + avoids race conditions
   + automatic variables can't prematurely go out of scope
   - inconsistent with V2K fork/join access to automatic variables

   a. A local copy of used automatic variables is made (for each sub-process)

      This is the original proposal made in ERR-4.

      + provides a simple solution to the "looping problem"
      - inconsistent with V2K automatic variables

   b. A local copy is made only for user-specified automatic variables

      + consistent with V2K automatic variables, adds a new semantic on top of what is already there
      + provides a simple solution to the "looping problem"
      - a bit more verbose

   c. No local copy support is provided.

      + consistent with V2K automatic variables, no new semantics
      - it would be very cumbersome to work around the "looping problem"

2. Sharing of automatic variables is allowed

+ consistent with V2K fork/join access to automatic variables
- possible race conditions (consistent with V2K)

a. The lifetimes of shared automatic variables depend on when sub-processes stop using them.

+ automatic variables can't prematurely go out of scope
- inconsistent with V2K automatic variables
- could be difficult to implement
- could cause a hit in simulation performance

i. Users have the option of specifying a set of automatic variables that are to be locally copied.

This is the way Vera works today.

+ provides a simple solution to the "looping problem"

ii. No local copy support is provided.

- it would be very cumbersome to work around the "looping problem"

b. The lifetimes of automatic variables end when they go out of scope

+ no changes to V2K automatic variables
- automatic variables can prematurely go out of scop

i. Users have the option of specifying a set of automatic variables that are to be locally copied.

+ provides a simple solution to the "looping problem"
+ allows automatic variables to be prevented from prematurely going out of scope

ii. No local copy support is provided.

- it would be very cumbersome to work around the "looping problem"