NIST

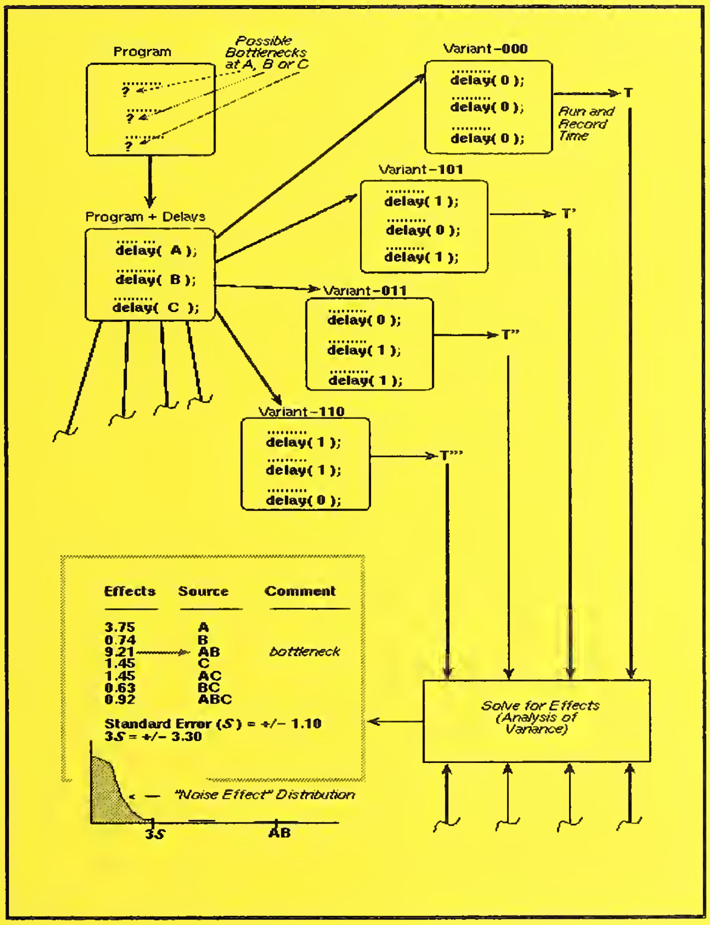
U.S. Department of Commerce
National Institute of Standards and Technology
High Performance Systems and Services Division
Scalable Parallel Systems and Applications Group

NISTIR 5789

Using S-Check

Alpha Release 1.0

Robert Snelick
Nathalie Drouin
John Antonishek



Effects	Source	Comment
3.75	A	
0.74	B	
9.21	AB	bottleneck
1.45	C	
1.45	AC	
0.63	BC	
0.92	ABC	

Standard Error (S) = +/- 1.10
3S = +/- 3.30

February 1996
Supported by NIST task number 40131 and ARPA task number 7066.

NISTIR 5789

Using S-Check

Alpha Release 1.0

Robert Snelick
Nathalie Drouin
John Antonishek

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
High Performance Systems and Services Division
Scalable Parallel Systems and Applications Group
Gaithersburg, MD 20899

February 1996
Supported by NIST task number 40131 and ARPA task number 7066

Preface

Today's multiprocessors provide unprecedented performance potential, yet all too often the actual performance obtained is far less impressive. The inherent complexity of parallel programs makes it far more difficult to capture true performance measurements on multiple-instruction stream, multiple-data stream (MIMD) architectures. In the absence of MIMD performance tools, obtaining reasonable parallel program performance is no small undertaking. The goal of S-Check is to provide a tool that gives the programmer useful performance information and is portable across machines as well as architectures.

S-Check automates the techniques of Synthetic Perturbation Screening (SPS). Synthetic Perturbation Screening systematically perturbs selected program code segments and determines performance sensitivities of these selected segments by using the statistical techniques of Design of Experiments (DEX). The resulting sensitivity analysis serves as a basis for performance evaluations. The name S-Check is derived from sensitivity analysis or Sensitivity **Check** (S-Check).

The concepts of Synthetic Perturbation for performance analysis of parallel programs were developed in the Parallel Processing Group at the National Institute of Standards and Technology (NIST) under the direction Dr. Gordon Lyon. In addition to Dr. Lyon and the authors, the following have contributed to the project: Dr. Raghu Kacker guided theoretical aspects of the statistical library developed for the tool; Dr. Joseph Ja'Ja' helped formulate some of the SPS techniques; Dominique Rodriguez wrote much of the tool's front end C parser and source code re-generator; Dr. James Filliben provided insight and examples for presenting statistical results graphically; Michel Courson and Arnaud Linz provided assistance in testing the techniques and tool prototypes.

S-Check 1.0 Alpha release works on Silicon Graphics and Sun parallel machines. S-Check analyzes any code that adheres to either Kernighan and Ritchie or ANSI C. The graphical interface is written with the widely available OSF Motif toolkit. We plan to port S-Check to other platforms in the near future. They include the IBM SP2 and Intel Paragon. When using this early release of S-Check, it is important to remember that many planned features have not been completed or implemented at all; we point these out in the manual.

Funding for the project is provided by NIST and the Advanced Research Projects Agency (ARPA), task number 7066.

S-Check is public domain. All or any parts of it can be used, modified, or incorporated into other systems without permission from NIST or the authors. However, the authors and NIST would appreciate credit if the tool or parts of it are used. There is no warranty, expressed or implied, on the capabilities of the code.

Send questions, comments, and bugs to **scheck-tool@www.scheck.nist.gov**. For information on how to obtain S-Check and for updates access the **http://www.scheck.nist.gov** Web site.

Robert Snelick
Nathalie Drouin
John Antonishek

Gaithersburg, MD.
September 1995

Using S-Check

Using S-Check

Robert Snelick
Nathalie Drouin
John Antonishek

What is S-Check?

S-Check is a software sensitivity checker designed to help you locate performance bottlenecks in parallel (and complex serial) programs. The tool S-Check provides the mechanisms to:

- determine the impact of computational code segments
- determine how well a program or code segment scales (not yet implemented)
- determine the cost of synchronization barriers (not yet implemented)
- detect interdependencies amongst code segments

S-Check employs and automates statistically designed experiments to identify sources of performance degradation. The host system can be a serial, parallel or networked (PVM-like) layout. The tool implements the techniques of Synthetic Perturbation Screening (SPS) developed at the National Institute of Standards and Technology (NIST). The methodology demands laborious experiment setup and execution procedures. S-Check automates much of the drudgery in an easy to use graphical user interface.

S-Check provides easy selection of test parameters, code instrumentation, experimental plan setup, experiment execution, calculation of results, and graphical presentation of results. S-Check is designed to accommodate users with varied SPS and design of experiment (DEX) skills. SPS proficient users personally control experiment details while novice users are given mechanisms for automatic setup and testing.

SPS introduces the notion of inserting artificial delays into the source code and capturing the effects of such delays by employing design of experiment techniques. Performance information takes the form of effects that correspond to source code segments or interactions among code segments. Effects are ranked by magnitude. Source code segments with the highest effects are likely candidates for bottlenecks. Based on variations of these techniques other performance information specific to a given architecture can also be obtained.

This document is provided as a user's guide to S-Check; it is not intended to describe or explain SPS techniques. However, a brief superficial overview of the process is given in the next section, *SPS Basics*. A detailed description of SPS can be found in the following publications: "Synthetic-perturbation tuning of MIMD programs"[1], "Synthetic perturbation techniques for screening shared-memory programs"[2], and "A simple scalability test for parallel code"[3].

Overview of an S-Check Experiment

S-Check's basic notion is an *experiment*. An experiment defines all the parameters needed to setup and run the SPS process. S-Check views an experiment as an object that can be created, opened, initialized, saved, executed, displayed, and modified. Multiple experiments may be instantiated in an S-Check session. The following list provides an overview of the basic steps that need to be performed (from the user's perspective) in an S-Check experiment:

- Create/Open an experiment
- Declare the target machine (not available in this release)
- Configure the test program
- Select the experiment type
- Select program test points
- Define a response interval
- Select an experimental plan
- Select experiment replication
- Run the experiment

- View experimental results

S-Check assists in or performs each of these steps.

In addition to providing the necessary functions to run an experiment, S-Check organizes experiments by providing maintenance functions such as saving the experiment configuration, control settings, and results. S-Check also provides status information of the experiment. It indicates the number of test parameters selected, number of runs required to complete the experiment for a given plan type and replication setting, an estimate on how long an experiment will run, an estimate of the delay magnitude, state of an experiment, and trial number of a running experiment. Results from experiments take the form of ranked order lists or graphical plots of effects. This data can be saved as postscript files.

S-Check's 1.0 Alpha Release

In this release, S-Check works on C programs on Silicon Graphics and Sun parallel machines. The next phase of the project is to link the tool to specific parallel machines. Since S-Check works at the source code level this only entails implementing procedures to build and run the test program on the target machine. S-Check analyzes any code that adheres to either Kernighan and Ritchie or ANSI C. The graphical interface is written with the widely available OSF Motif toolkit.

In this alpha release, the framework for some features exists in the interface but have not yet been implemented. These features are *grayed out*--they are inaccessible. For example, barrier and scaling tests are not yet implemented. Features that are not available in this release are nonetheless described.

SPS Basics

This section presents a perspective of the underlying technique that S-Check implements, which is Synthetic Perturbation Screening (SPS). A reader familiar with the technique can skip to the next section, *Getting Started.*

Synthetic Perturbation Screening (SPS) is a method of code investigation that helps identify which code segment matters most for performance improvement. It is especially suited for assaying MIMD parallel programs. Relying on the mathematical perspective of statistically designed experiments, program bottlenecks are directly identified as quantitative effects upon a response time. A MIMD program is treated as a black box with input parameters and an output. User-induced synthetic perturbations (or delays) are inserted in the source code at chosen locations suspected to be program bottlenecks. The added perturbations determine a set of input parameters with controllable settings (e.g., delay ON, delay OFF). Statistically designed experiments capture the effects of such controllable delays on the overall system behavior. Code segments that are highly sensitive to code perturbations have a detrimental effect on the system performance, so that source code improvement on these segments will likely yield increased performance. SPS is independent of the system's architecture and works for both shared and distributed memory machines. The technique is portable and scales well.

SPS step-by-step

The SPS technique involves:

- a test program P.
- a response to be improved (e.g., the running time of P)
- statistical techniques from Design of Experiments (DEX).
- code modification via Synthetic Perturbation.

A basic evaluation of a test program P involves the following steps:

Step 1: Select code locations to be tested. Any segment of code is a potential candidate location. Typically, it is a basic block construct (e.g., loops, functions, etc.), a critical section or a synchronization mechanism. These code locations (test parameters) are referred to as *factors.*

Step 2: Insert synthetic perturbations at the factor locations. Factors are treated with a delay/no delay option. No-delay option leaves the code unperturbed. A delay option adds a specified number of artificial instructions to the code. The perturbation occurs as a time delay and a uniform treatment setting is used (i.e., the duration of the delay does not vary with its location). The perturbation itself does not interfere with the logical state of the program.

Step 3: . Generate an experimental plan of analysis. An experimental plan can be developed once the factors have been chosen for study. A plan of experiment is a succession of trials with different patterns of synthetic delays. Each pattern involves a different combination (treatment) of delay settings. Different schemes are available to the user.

Step 4: Run the trials and record their responses. Each different pattern in the statistical plan corresponds to a different treated version of the program. Treatments are run in a random order and a response is recorded for each run. A typical response is the running (response) time of P.

Step 5: Perform analysis. The relative importance of the various delays is determined by an analysis of variance. Each location or interaction of delays is ranked quantitatively according to its sensitivity to the synthetic perturbation. Such a list is called an SPS rank.

Step 6: Tuning. SPS assumes that source code improvement at locations highly sensitive to code perturbation will yield increased performance. The tuning process is performed via iterative refinement (repeat steps 1 through 5) until satisfactory performance is obtained.

Within this framework, more specialized problems can be solved. These issues are quickly addressed in the next section.

Screening and Special SPS tests

Screening test. At an early stage of analysis, one is primarily concerned with identifying and discarding factors that have no significant effect on the program response. Screening is an investigation strategy that efficiently isolates important factors from a pool of candidates. Insignificant factors can then be removed from subsequent investigations, thus narrowing the scope of the analysis. Screening is an identification step that may be used to make a quick, preliminary assessment of a large application.

Barrier test. SPS can be used to address the issue of bottlenecks due to synchronization in the shared memory programming model. Processes that hit a barrier at widely dispersed times cause processors to idle for a significantly long period of time. The test objective is to identify barriers where such idle time overheads occur. Effects must be paired up and compared for analysis. In other words, two factors (or perturbations) are required per barrier. One is inserted immediately before the barrier, the other one immediately after. The barrier test requires a special factor treatment--not explained here. For each individual pair, the difference in the perturbations respective effects gives an indication of the cost associated with each synchronization. The synchronization cost increases with the difference in the two effects. If the paired effects are about the same the synchronization cost is marginal. Note that screening experiments and barrier tests must be conducted separately as factor treatments in these two cases are not comparable. See the Reference [2], "Synthetic perturbation techniques for screening shared-memory programs", for a thorough explanation of the technique.

Scaling test. Code scalability determines how well parallel code avoids becoming a bottleneck as its host computer is made larger. Statistically designed experiments handle program and system together as a single entity. The system size (i.e., the number of processors assigned to the program) comes as an additional factor to the regular set of input perturbations (i.e., segments of code suspected of being performance bottlenecks). A large negative number for the effect associated with the system size means the whole program is sensitive to scalability, since the response time decreases with the system size. To determine which code segment improvements best promote parallel speedup the effects of interactions between code segments and the system size are studied. A large negative number for these interaction effects means the code seems to be scalable, since the sensitivity to delays decreases with the system size.

Getting Started

S-Check's Directory/Experiment Layout

S-Check requires that all files needed to build the executable *test program* reside in the directory in which S-Check was started or if the target machine is remote, then the directory that is specified by the user in the Configuration Panel (**S-Check 1.0 Alpha release does not handle remote file access**). As illustrated below in Figure 1, an experiment is saved in a sub-directory of the starting directory, called the *experiment directory.* The first level of this hierarchy is named ".scheck". Subdirectories with the name of the experiment are then created under the ".scheck" directory. These directories contain internal S-Check information for an experiment. To open a saved experiment, you must start S-Check in the appropriate experiment directory.

To invoke S-Check, type the following command:

```
scheck
```

S-Check first displays a working dialog while it is performing system initializations. When this is completed the window for creating and opening experiments is launched.

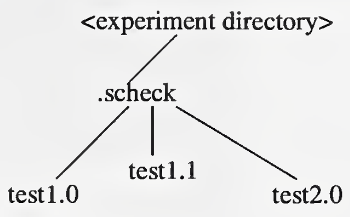


FIGURE 1. S-Check's directory hierarchy

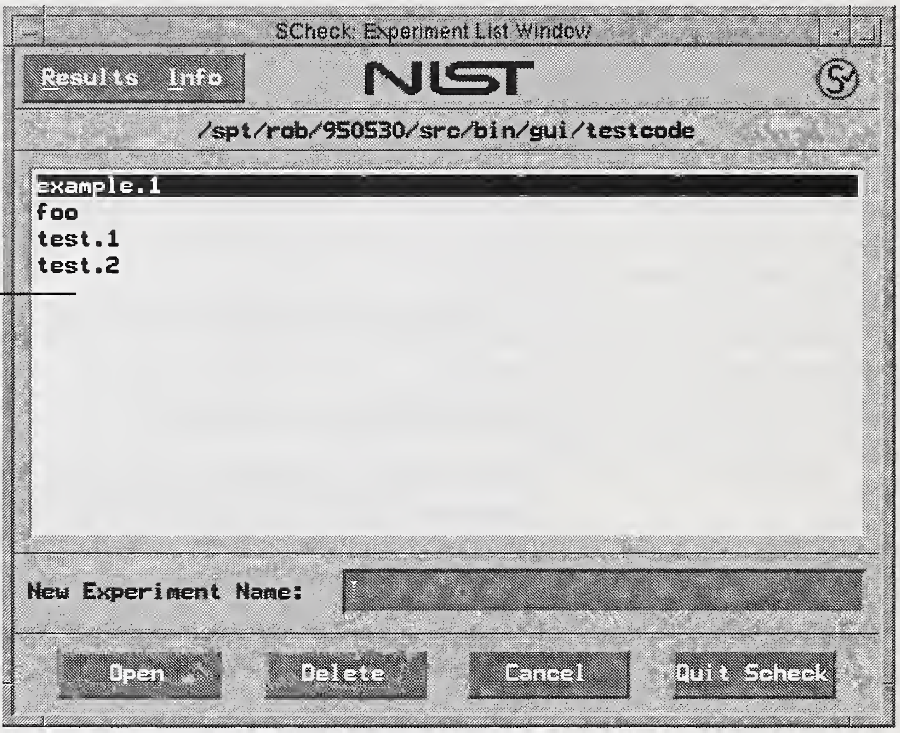
Creating/Opening an Experiment

S-Check starts by displaying the Experiment List Window, as seen in Figure 2. The experiment directory name is displayed near the top of the Experiment List Window. S-Check will look here for *work set* files if the experiment is to be performed on the host machine. The Experiment List Window displays previously defined experiments as well as an area for creating new experiments. To create a new experiment, enter the name of the experiment in the New Experiment Name field and select the *Open* button. To open a previously defined experiment double-click on the desired experiment in the experiment list.

Either of these actions brings up the Experiment Control Window (to be described shortly). This action also brings up the Configuration Window if the configuration for the experiment has not yet been defined. To delete an experiment, click on the experiment name and press *Delete.* To exit the Experiment List Window without requesting any tasks, select the *Cancel* button. To exit S-Check click on the *Quit S-Check* button. The *Cancel* button will also exit S-Check if no experiment is active.

Creating a new experiment from a previously defined experiment can really economize experiment setup time. Simply recall an experiment with a suitable configuration and use the *Save As* command on the Experiment Control Window menu. Modifications for the new experiment can now be made without having to re-initialize the bulk of experiment settings.

The Experiment List Window has two main menus: *Results* and *Info.* The menu selection *Multiple Display*, under *Results*, allows you to view results of previously saved experiments. The details of this are explained in *Viewing Results.* The *Info* menubar provides on-line information about S-Check.



/spt/rob/950530/src/bin/gui/testcode

example.1
foo
test.1
test.2

New Experiment Name:

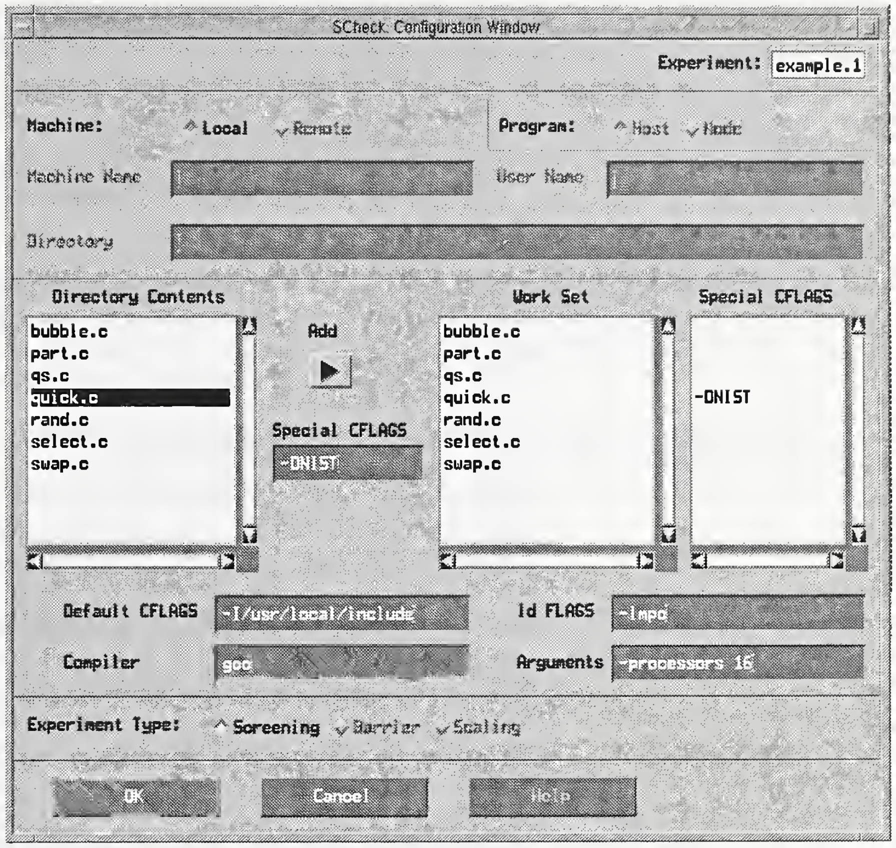
FIGURE 2. **Experiment List Window**

Experiment Configuration

The Configuration Window (Figure 3) supplies information to S-Check on how to build the executable program, how and where to run the experiments, and the type of the experiment.

The Configuration Window is invoked automatically upon creation of an experiment. Alternatively, it can be explicitly brought up from the Experiment Control Window under the *File* and *Configure* menu selections.

.



Experiment: example.1

Machine: [✓ Local] Remote

Program: Host Node

Machine Name

User Name

Directory

```
Directory Contents   Work Set      Special CFLAGS
bubble.c             bubble.c
part.c               part.c
qs.c                 qs.c
quick.c              quick.c       -DNIST
rand.c               rand.c
select.c             select.c
swap.c               swap.c
```

Special CFLAGS: -DNIST

Default CFLAGS: -I/usr/local/include

ld FLAGS: -lmpd

Compiler: gcc

Arguments: -processors 16

Experiment Type: [✓ Screening] Barrier Scaling

FIGURE 3. Configuration Window

Declaring the target machine

The first group of text fields in the Configuration Window request information about the *target machine* and associated information. First choose whether the target machine is the host (local) machine on which S-Check is running or whether the target machine is accessed remotely. The default setting is local. **Remote machine access is not available yet.** If the target machine is a remote machine, the following information must be provided:

- Machine Name of the remote machine (e.g., seq.ncsl.nist.gov)
- User Name on the remote machine (e.g., rob)

Note: this connection must be set up such that access to the machine can be gained without requiring the use of a password. Refer to Unix documentation on “.rhosts” or “host.equiv”.

- Directory (*working directory*) of the program that will be used in the experiment (e.g., /home/users/rob/parallel/quicksort)

  Note: that all files necessary to build the executable program must reside in this directory.

If a remote machine is declared, all target files must be located on the remote machine. There is no option of having the target files local with compiling and running on the remote machine.

Connecting your application to S-Check

The next step is to provide information so that S-Check can build your test program. S-Check requires that you select files and provide flags that are needed to compile your program. S-Check also provides an area to declare command line arguments for the executable. The list called Directory Contents names all allowable files from which an executable can be built. Select a file and enter any special C-flag(s) needed for that file. Push the *add* button to enter the file into the *work set*. The work set defines the files needed to build the executable. This list also defines the set of files that can be edited for choosing test locations (factors). The work set list appears on the Experiment Control Window. Double clicking on a file (under the Work Set list) will remove the file from the work set. Set default C-flags that apply to all C files in the Default CFLAGS text field. For example, to instruct S-Check to look in the include directory named /usr/local/include, enter

```
-I/usr/local/include
```

in the Default CFLAGS text field.

Likewise the loader/linker flags can be set in the ld FLAGS text field. To specify a compiler, other than the default, enter the name of the compiler (path, if necessary) in the Compiler text field. Command line arguments for the test program are set in the area named Arguments. Enter only the arguments. Do not enter the name of the executable. **File input/output redirection is not yet implemented.**

Selecting the experiment type

S-Check can perform basic screening tests to extract a sensitivity analysis of the test program. Specialized tests are also available. For shared memory architectures, the cost of synchronization barriers can be evaluated. Scaling is another test, one that determines how well a code segment scales as the size of the machine is increased.

The chosen experiment type dictates certain S-Check restrictions and requirements for additional input. For example, when performing barrier tests, only barriers can be selected as factors. Scaling tests require that the test program be capable of varying the number of processors. To select the experiment type, click on the desired test. Only one test can be selected.

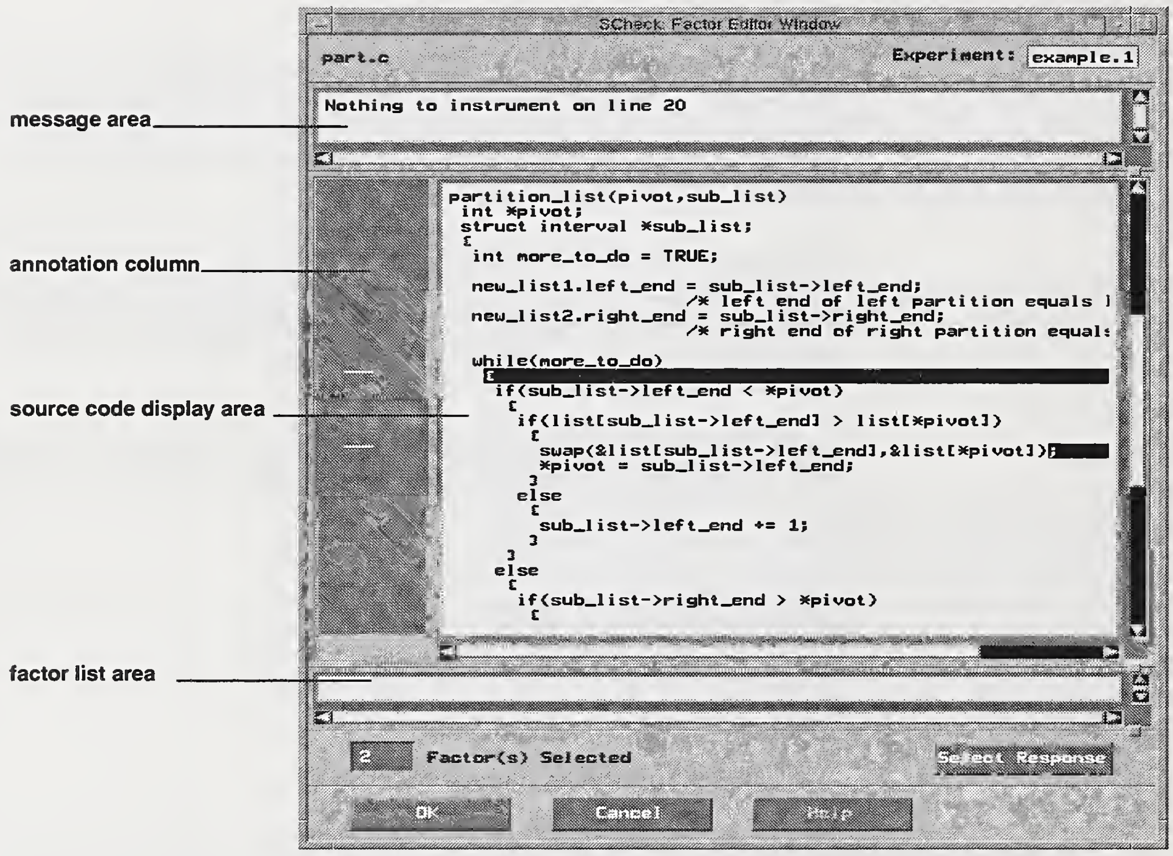
Pressing *OK* will record this information and pop up the Experiment Control Window. *Cancel* will bring down the window with no changes or selections recorded.

Instrumenting the Test Program

From your perspective instrumenting the test program involves two functions:

- selecting factors
- defining the response interval

Factor selection involves picking code segments in the program that you wish to test. Wherever a factor is selected, S-Check inserts code that can be activated to cause a delay at that location. You can choose factors and the response interval by launching Factor Editors. To start a Factor Editor double-click on the file you wish to edit in the Experiment Control Window (Figure 8). Starting a Factor Editor brings up a Factor Editor Window, loaded with the source code, and ready for factor selection. Figure 4 shows an example of a Factor Editor.



```
Nothing to instrument on line 20
```

```
partition_list(pivot,sub_list)
 int *pivot;
 struct interval *sub_list;
 {
  int more_to_do = TRUE;

  new_list1.left_end = sub_list->left_end;
                     /* left end of left partition equals l
  new_list2.right_end = sub_list->right_end;
                     /* right end of right partition equals

  while(more_to_do)
   {
    if(sub_list->left_end < *pivot)
     {
      if(list[sub_list->left_end] > list[*pivot])
       {
        swap(&list[sub_list->left_end],&list[*pivot]);
        *pivot = sub_list->left_end;
       }
      else
       {
        sub_list->left_end += 1;
       }
     }
    else
     {
      if(sub_list->right_end > *pivot)
       {
```

FIGURE 4. **Factor Editor Window**

Factor Editor Window

Through the Factor Editor you select factors and determine the response interval. The scrollable message area provides feedback during factor selection. For example, if you select an item that can't be instrumented, an appropriate message is displayed. The annotation column indicates if a factor is selected on a line by marking the corresponding annotation column line with a white horizontal dash. It also displays the start-stop indicators when the response interval is set. See discussion on Setting the Response Interval. The source code display area is where you define factors. Discussion on this topic is expanded in the next section, *Selecting Factors.*

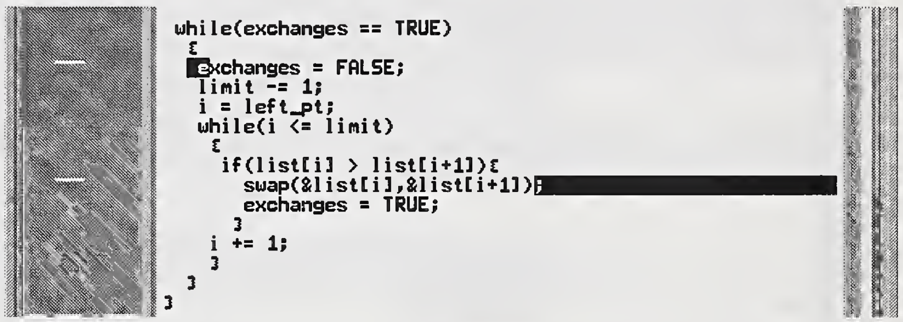
The factor list area provides a list of selected factors. The factors are displayed in sorted order depending on their line number within the file. **The factor list area is not yet available.** The factor count displays the total number of factors selected in the file that is being edited. A count of all factors selected for the experiment is displayed on the Experiment Control Window. To the right of the factor count is the *Select Response* button. The purpose of this button is described in *Setting the Response Interval.*

Press *OK* to dismiss the Factor Editor and record all changes. To lower the window while ignoring modifications, press *Cancel.* **The Help feature is currently unavailable.**

The size of the scrollable windows (i.e., message area, source code display area, and the factor list area) can be increased/decreased by moving the panes at the lower right hand side of each scrollable window.

Selecting Factors

Factors are test points in the code. Currently, you are responsible for selecting factors. Future versions of S-Check will provide limited automatic factor selection. The notion of a factor comes from the branch of statistics called design of experiments (DEX). In DEX, factors correspond to parameters that are varied in the experiment. To select a factor, click on the location where you want to define a test point. If this location is a valid factor that can be instrumented, the location of the instrumentation is indicated. The location is tagged with reverse video (Figure 5). The instrumentation will be inserted between the left most character and the right most character of the reverse video. Some exceptions to this rule exist and are pointed out below. Click on the location again to remove the factor. The factor will no longer be highlighted in reverse video. Once a factor has been selected the factor count is updated for the current file that is being edited. The global factor count for the experiment is maintained on the Experiment Control Window.



```
while(exchanges == TRUE)
 {
 exchanges = FALSE;
  limit -= 1;
  i = left_pt;
  while(i <= limit)
    {
     if(list[i] > list[i+1]){
       swap(&list[i],&list[i+1]);
       exchanges = TRUE;
       }
     i += 1;
     }
   }
}
```

FIGURE 5. Factor Selection, Default Interpretation.

Figure 5 shows a cut out of a Factor Editor. The first selection instruments the body of the **while()** loop. That is, the perturbation code is inserted as the first statement in the loop. The second selection instruments the code after the call to **swap()** and before the statement **exchanges = TRUE.**

Defining a code segment to be a factor will cause instrumentation code to be inserted at that location. The code instrumentation process occurs later in the SPS process. In most cases the instrumentation code is inserted between the left most and right most character displayed in reverse video. In a few situations this rule does not apply. The user must exercise caution when selecting factors to insure that the intended location for instrumentation is the desired location. The example above shows the default interpretation. The example in Figure 6 illustrates the special cases that apply to compound statements.

```
<F1> <D1> while <F2> (index > limit) <F3> { <F4>
            <D3> <F5> <D5>   stmt1 = setting1;
                             stmt2 = setting2;
            <D4>
            }
            <D2>
```

FIGURE 6. Factor Selection, Special Cases for Compound Statements.

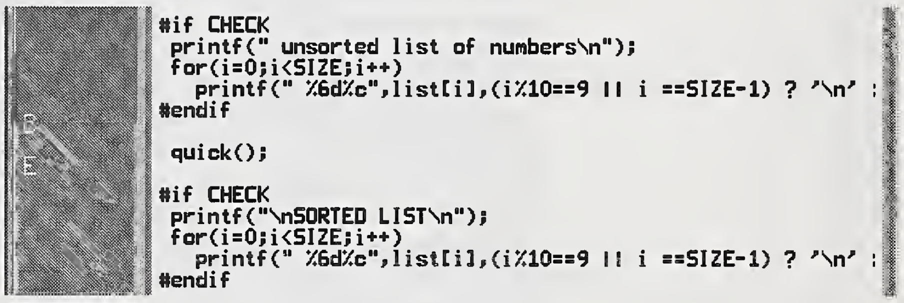
The symbols <F1> through <F5> indicate the test locations you selected. In S-Check these locations will be highlighted in reverse video. The symbols <D1> through <D5> indicate the locations where the delay treatment is inserted into the code. Factors <F1> and <F5> adhere to the default interpretation as the delay is inserted where the factor was chosen. The interpretation for factor <F2> is to instrument the code following the while() loop (<D2>). Clicking at location <F3> instruments the body of the loop, that is, the first statement in the loop. It is instrumented exactly like <F5>. If the location F4 is selected, then the last statement in the while loop is instrumented (<D4>). The impact of <F4> and <F3/F5> will differ if there is branching back to the beginning of the loop. Some of these special case options are implemented for future S-Check features. If you want to instrument the body of the loop, it is best to use the default interpretation as in <F5>.

If you select a location that S-Check cannot instrument, you are notified of this in the message area. Items that cannot be instrumented include lines that are encased in preprocessor commands which are not defined and therefore will not be part of the executable code. Locations that produce invalid code (e.g., at the end of a function) can not be instrumented.

Setting The Response Interval

The second item that needs to be set for instrumenting the code is the response interval. The *response interval* defines the start and stop locations for capturing the response time. The response time equals the time recorded at the stop location minus the time recorded at the start location. The response time is used in the calculation of effects. The start/stop locations can be set in any file and the response interval can span files. That is, the start/stop locations need not reside in the same file. The response interval should be set so that it encompasses all factors. Otherwise the effect for the factors not in the domain of the response interval are not accounted for. It is important that the response interval be set in a sequential part of the code or that it is guaranteed to be set by only one process. Results are otherwise undefined.

To select the response interval click on the *Select Response* button. This will turn off the factor selection mode and activate the select response mode. While in the select response mode, the cursor is modified to a down arrow indicator or to an up arrow indicator. Clicking in the source code display area while the down arrow cursor is active will cause a “B” to be placed in the annotation column corresponding to the selected location. Likewise, clicking in the source code display area while the up arrow cursor is active will cause an “E” to be placed in the annotation column corresponding to the selected location. The “B” indicates the start (begin) location and the “E” marks the stop (end) location.



```
#if CHECK
 printf(" unsorted list of numbers\n");
 for(i=0;i<SIZE;i++)
   printf(" %6d%c",list[i],(i%10==9 || i ==SIZE-1) ? '\n' :
#endif

 quick();

#if CHECK
 printf("\nSORTED LIST\n");
 for(i=0;i<SIZE;i++)
   printf(" %6d%c",list[i],(i%10==9 || i ==SIZE-1) ? '\n' :
#endif
```

FIGURE 7. **Defining the Response Interval.**

The first time the *Select Response* button is clicked, the down arrow indicator is active. You can change the selection mode to an up arrow (for defining the stop location) by selecting a start location or by clicking on the *Select Response* button. The latter option allows you to avoid setting or changing the start location. The former option will define the start location and automatically put the editor into the define stop location mode. Clicking twice on the *Select Response* button will return you to the factor selection mode (bypassing the stop location mode). The factor selection mode is automatically entered after the stop location is defined. Unlike factor selection, defining the response interval is line oriented. The resolution of the “B” and “E” (in this implementation) are somewhat coarse, so make sure the start and stop locations are clearly defined.

In Figure 7, the function **quick()** is defined to be the response interval.

S-Check does not check for multiple execution of start/stop locations. It does however check if the stop location is executed before the start location. Only one response interval can be set. Setting the start/stop locations when the response interval is already defined will overwrite any previous settings. The new start/stop locations will define the response interval.

Setting the response interval will automatically reset the delay value to an undefined state. It also changes the job status to Empty. A *built experiment* will have to be reconstructed.

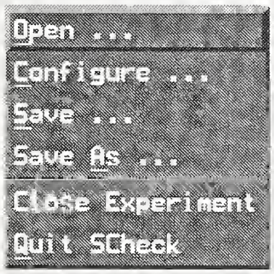
Experiment Control

The Experiment Control Window (Figure 8) involves:

- Experiment maintenance and convenience functions (*File* menubar)
- Launching factor editors
- Setting and showing the delay value (*Utilities* menubar)
- Selecting a DEX plan
- Selecting experiment replication
- Displaying progress and error messages from S-Check
- Displaying experiment setup information
- Running an experiment
- Displaying run status information
- Saving Results
- Launching result viewers (*Display* menubar)

The first seven topics are described in this section. The next two are explained in the section to follow, *Running an Experiment*. The last two items are covered in *Saving Results* and *Viewing Results*.

Experiment Maintenance and Convenience Functions



Experiment maintenance and convenience functions are accessed under the *File* menubar selection. *Open* gives you access to the Experiment List Window. For experiment configuration modifications select the *Configure* button to bring up the Configuration Window. To save the current experiment, select *Save*. This action brings up a box with toggle buttons named *Save Experiment* and *Save Results*. To save all of the settings of an experiment, select *Save Experiment* and hit *Save*. The saved file contains everything necessary to retrieve an experiment in its present state. These items include everything defined on the Configuration Window, Experiment Control Window and Factor Editors. It also saves the state of the interface. A saved experiment can be opened from the Experiment List Window. Results of an experiment can be saved by selecting the *Save Results* toggle button and hitting *Save*. This action pops up the Save Results Window. See the section on *Saving Results* for details. To save an experiment with a new name, select *Save As*. To close an experiment, select *Close*. *Quit S-Check* will exit the program.

Launching Factor Editors

The work set is the list of files defining the target program. This list is constructed in the Configuration Window. Double-click on the file you wish to edit. This action launches a factor editor, loaded with source code, and ready for factor selection. There is no limit on the number of factor editors you can launch. See *Instrumenting the Test Program* for details on using a factor editor.

S-Check Messages

S-Check uses the message area to inform you about its progress in instrumenting the target program. Typical status information includes:

- files being parsed
- compilation progress
- stages in determining the delay value
- state and value of the delay
- traces of response time and treatments for trials
- information for effect calculations
- warning and errors

TABLE 1. **Experiment Control Warnings and Error Messages**

Message	**Reason**	**Fix/Consequent**
delay value is no longer set	1. factor set was changed. 2. response interval was changed.	1. a built experiment has to be reconstructed because the previous delay value may not be appropriate for the new experiment settings.
unable to find suitable delay value in "*x*" tries	1. factors not significant enough, S-Check gave up looking for large delay value. 2. response times not linear with respect to delay value due to unusual program or varying system loads.	1. re-evaluate factor choices or set the delay manually. 2. look for unusual behavior in program or run when system load is stable.

TABLE 1. **Experiment Control Warnings and Error Messages**

Message	Reason	Fix/Consequent
resolution of delay value may be too coarse	1. S-Check's nominal delay value is too coarse for factor set.	1. response times may be unnecessarily long due the coarseness of S-Check's delay value. Future S-Check versions will allow for finer grain delay values.
runtime with no delay set is > runtime with delay	1. response interval does not encompass factors 2. selected factors are not significant enough to cause response time to increase significantly 3. program is very unusual	1. make sure the response interval encompasses all selected factors 2. re-evaluate factor set 3. check for severe communication contention or other anomalies which make the program run faster with delays. See [1] for an example.
variance computed with "*x*" degrees of freedom. You need at least 10 for a meaningful estimate.	1. plan does not provide enough information to obtain a meaningful estimate of the standard error	1. disregard and use normal probability plots to select significant factors. 2. choose new plan or run with replication.
S-Check can't handle saved suspended jobs. Job Status is set to built.	1. experiment was saved while job was suspended.	1. Current S-Check implementation can't handle suspended jobs. Job must be re-started.
Creation of instrumented source failed.	1. The program **CInstGen** failed.	1. Check the standard error message log and look for a work around. 2. This is a bug in S-Check and should be reported.
linking failed	1. unable to create executable	1. verify program code and linkage flags.

Table 1 gives a list of potential warning and error messages along with suggestions on how to resolve problems.

Setting the amount of delay

The delay value is either set by you manually or by S-Check automatically. By selecting *Build* (see *Running an Experiment*), S-Check will automatically determine an appropriate delay size for the experiment. It is part of the *build* process. Alternatively, you can set the delay value manually.

To set the delay value manually, select *Set Delay* from the *Utilities* menu on the Experiment Control Window. This pops up the Set Delay Window. After entering the delay value, select *Set*. A message confirming the delay value setting is printed in the message area on the Experiment Control Window. To get back to the default state of letting S-Check automatically determine the delay value, choose *Unset*.

The duration of the artificial delay is an important aspect. Ideally, the delay should be long enough so that it can be distinguished from experiment noise and short enough so as not to produce unnecessarily long program execution times. When setting the delay manually, it is up to you to select an appropriate value. When the delay is determined automatically, a few trial runs are performed until a satisfactory setting is found.

The delay itself is a function that performs artificial instructions. The delay value controls how many times a loop iterates performing the artificial instructions. With the current implementation of the delay function, it is recommended that S-Check tests be performed without compiler optimization. A future improvement to S-Check will incorporate a variety of artificial perturbation options.

The delay value can be queried at any time by selecting the *Show Delay* button under the *Utilities* menubar.

Selecting a DEX plan:

A design of experiment plan is a complete description of a minimum set of perturbation patterns needed to carry out a meaningful program investigation. A variety of schemes (briefly described below) is available in S-Check. It is important to note that there exists a direct correlation between the minimum number of runs in an experiment, the quantity of information provided by the investigation and the selected plan type. The trade off buys information with number of runs.

The user has six options for defining the experimental plan.

- Automatic
- Full factorial

- Half factorial
- Quarter Factorial
- Resolution IV
- Resolution III

The total number of factors under study is an important criterion in the choice of an experimental plan. For efficiency purposes, it is essential to maintain a balance between the quantity of information desired and the cost of the corresponding experiment. Any knowledge on interactions or the lack thereof should be put to use--there is no point looking for third-order interactions when they are known to be absent. Table 2 is intended to help you meet these requirements.

TABLE 2. Plan Selection

	Plan Selection		
	Up to S- Check	**Up to the user**	
Number of factors	<200	small(<12)	large(12<f<200)
Option	Automatic	Full Half Quarter	Resolution IV Resolution III

With *Automatic*, S-Check decides which plan type to use in investigating the program. Table 3 summarizes how choices are made.

TABLE 3. Automatic Selection

Number of Factors	**Plan Type**
up to 4	Full Factorial
5 or 6	Half Factorial
7 or 8	Quarter Factorial
9 to 31	Resolution IV
31 to 199	Resolution III

Full factorial designs consider every combination of factor levels, that is, they support inferences about all factor interactions. Be aware that the number of measurements (program runs) rapidly becomes prohibitive as the number of factors increases. A full factorial design is usually not appropriate for large numbers of factors. Note that high-order interactions are quite often of negligible magnitude when compared to main effects and low-order interactions; therefore when the number of factors increases, the desired information can be obtained by performing only a fraction of a full factorial experiment. Full factorial designs can currently be requested for up to 12 factors (2^{12} runs).

A *half factorial* design requires only half the runs of a complete full factorial. A minimum of 3 factors (theoretical limit) is necessary to request this type of plan. Such a design in 3 factors is of resolution III (explanation follows). To get an estimate of the standard deviation (called standard error) under this particular configuration you will need replicates. Other half factorial designs provide an estimate of the standard error without replication. Half factorial designs can be requested up to 9 factors (implementation limit).

A *quarter factorial* design requires only one-fourth the runs of full factorial (i.e., half the runs needed in a half-factorial design). A minimum of 5 factors (theoretical limit) is necessary to request such a design. This 5-factor configuration is of resolution III and as noted previously replicates are needed to get an estimate of the standard error. Other quarter factorial designs provide an estimate of the standard error without replication. This type of plan can be requested up to 10 factors (implementation limit).

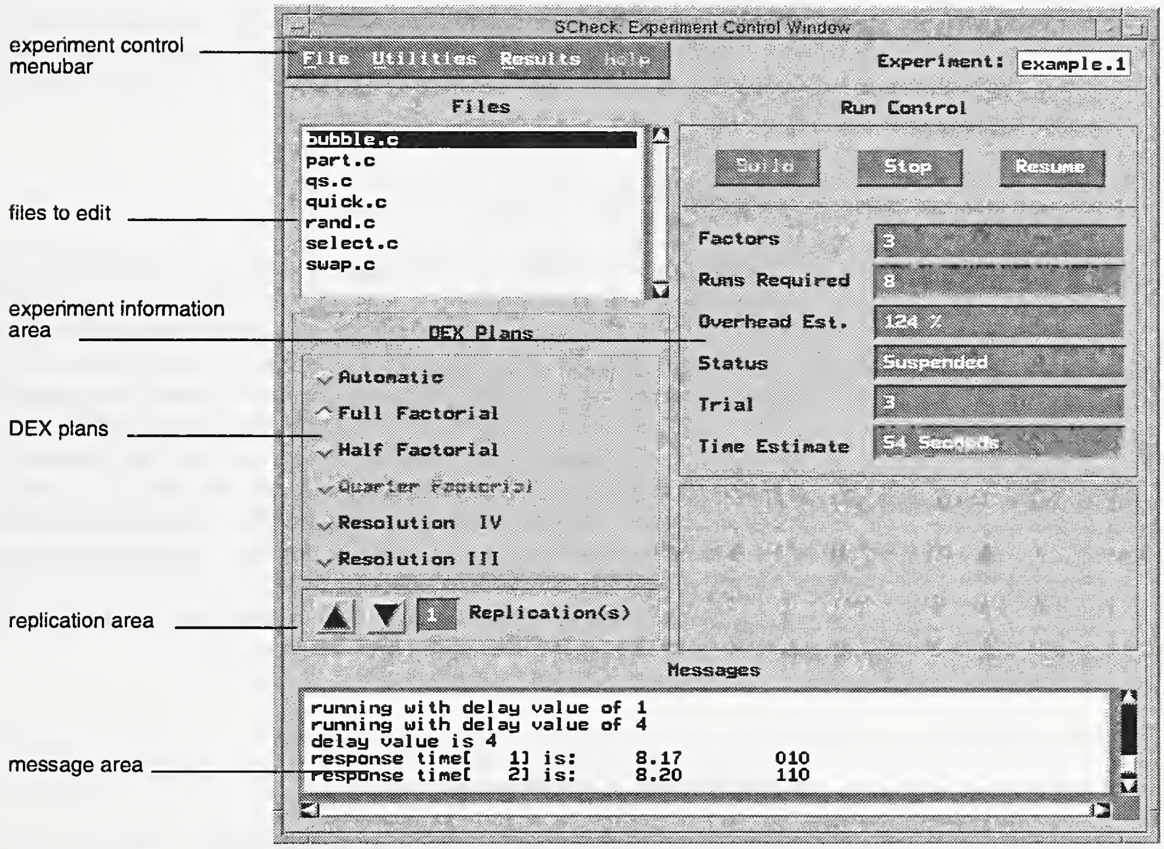
The resolution of a plan gives an immediate indication of the information capacity of the design. *Resolution III* plans confound (mix) results of first and second order effects, while *resolution IV* plans do not confuse first and second order effects, but instead confound first and third order effects. Ideally one should only use plans of resolution IV and up. Both resolution III and resolution IV plans are, however, appropriate for preliminary screening of a large number of factors, since in these cases many factors are insignificant.

Resolution IV designs may be requested for any number of factors between 2 and 199. They provide information on main (first order) effects only. In contrast to resolution III designs, no replication is needed to get an estimate of the standard error.

Investigations built on resolution III plans require half the runs of those built on resolution IV plans. Again, information is provided only on main effects and this time replications are needed to get an estimate of the standard error. Resolution III plans may be requested for any number of factors between 2 and 199.

Setting Replication

Replication indicates the number of times the experiment is performed. The default value is 1 (minimum), which means the experiment is run once. To duplicate the experiment, set replication to 2. The range of the replication value is currently 1 to 99.



```
running with delay value of 1
running with delay value of 4
delay value is 4
response time[  1] is:     8.17       010
response time[  2] is:     8.20       110
```

FIGURE 8. **Experiment Control Window**

Replication maybe necessary if anticipated variability or noise level of the experiment is high. Shared-memory systems generally have lower experiment noise than distributed-memory machines, with “PVM” style experiments being the worst. Replication may also be used to obtain a standard error for an experiment when a single run of the DEX plan cannot provide one. As mentioned, under some circumstances full factorial, half factorial, and quarter factorial plans do not provide a standard error. A single run of a resolution III experiment never provides a standard error (Normal probability plots allow some choices to be made without knowing the standard error). Refer to Appendix B for details on the availability and use of the standard error for an experiment.

To increase/decrease the number of replications, press the *up/down arrow* buttons to the left of the replication text field. For rapid movement hold the desired arrow button down. To reset replication to its default value click once in the replication text field.

Experiment Information Area

This area describes experiment parameters and attributes of the current experiment. Factors is the total number of factors selected for the experiment. Runs Required is the total number of times the test program is executed to complete the experiment. This value is affected by the number of factors, DEX plan, and the replication value. Overhead Est. (Overhead Estimate) gives an estimate (as a percentage) of how much longer the test program will run on average. Status indicates the state of the experiment: This topic is expanded in the next section. Trial indicates the current trial number S-Check is running (e.g., 17 of 32--the 17th run in a plan that requires 32 runs). Time Estimate shows how long the experiment will take to complete, based on preliminary measurements performed while determining the delay value and the number of runs in the experiment. This estimated value is updated during the execution of the experiment. If you set the delay value manually, then the Time Estimate is not given until after the first trial run.

Running an Experiment

There is a two step procedure for running an experiment. The first is an intermediate step called *build* (select *Build* on the Experiment Control Window). Building the experiment compiles the work set files and constructs an executable test program. The built program is instrumented with code that can activate/deactivate perturbation code at each factor location. Building the experiment also determines the size of the delay value (if not set manually).

At this point the experiment is ready for execution. You can select (if you haven't done so already) or change the DEX plan and number of replications without rebuilding the experiment. Changing the DEX plan or replication value can alter the number of runs required. **If the factor set is modified, the experiment must be rebuilt.**

To run an experiment, press *Start* on the Experiment Control Window. Starting an experiment will randomly execute each program version defined by the experiment parameters. The plan row and response time for a given trial is displayed in the message area. The series of 1's and 0's indicate whether or not a delay was executed for a given factor. If set to 1, the delay was executed. The code is not disturbed if set to 0. The factors are mapped to the plan row (left to right) and can be identified by correlating them with the id on the Effects List Window.

An experiment is suspended and restarted by selecting the *Suspend* and *Restart* buttons respectively. Suspending an experiment terminates the current running instance of the program. Data from completed trials are saved. Restarting an experiment first runs the trial that was terminated during the suspend operation and then continues the experiment. Response information is appended to data previously captured. Stop terminates the experiment with no saving of state. The job status is returned to **Built**, the experiment can be restarted. Start/Stop and Suspend/Restart occupy the same button region, therefore only one of the option pair is available at any given time.

Status indicates the current state of the experiment. An experiment can have the following states:

Empty	The experiment has not been built.
Building	The test program is being compiled with instrumentation code.
Determining Delay	Sample trial runs are executed for the purpose of finding a suitable delay value.
Built	The experiment was successfully built and the delay value is defined.
Running	The experiment is currently running.
Queued	The experiment is queued waiting for start-up. **This feature is currently unavailable.**

Suspended	The experiment is suspended. State information of previously run program instances are saved.
Terminated	The experiment has been Terminated. No experiment state information is saved.
Calculate Effects	All trials ran successfully. Results are being calculated.
Finished	The experiment has completed. Results of the experiment can be examined.

Saving Results

Results from the current experiment can be saved in a file by using the Save Results Window. The Save Results Window is accessed by selecting the *Save Results* button under the *Display* menubar. As mentioned earlier, it can also be accessed by selecting the *Save Results* toggle button from *Save* under the *File* menu. The Save Results Window can only be accessed when valid experimental results exist.

At the top of the Save Results Window are listed previously saved result files for the experiment. The text field (at the bottom of the window) provides an area to enter the name of the results file. Saved result files can be retrieved and viewed via List and Plot Windows through the Multiple Display Window. See the next section, *Viewing Results*, for details.

Viewing Results

There are two ways to view results. You can either obtain a rank-ordered list of the effects by launching a List Effects Window or observe the effects in various plot views of the Plot Effects Window. By default, data from the current experiment are displayed when either of these two windows are launched. To view results of other experiments you can bring up the Multiple Display Window. These options are found under the *Display* menu on the Experiment Control Window.

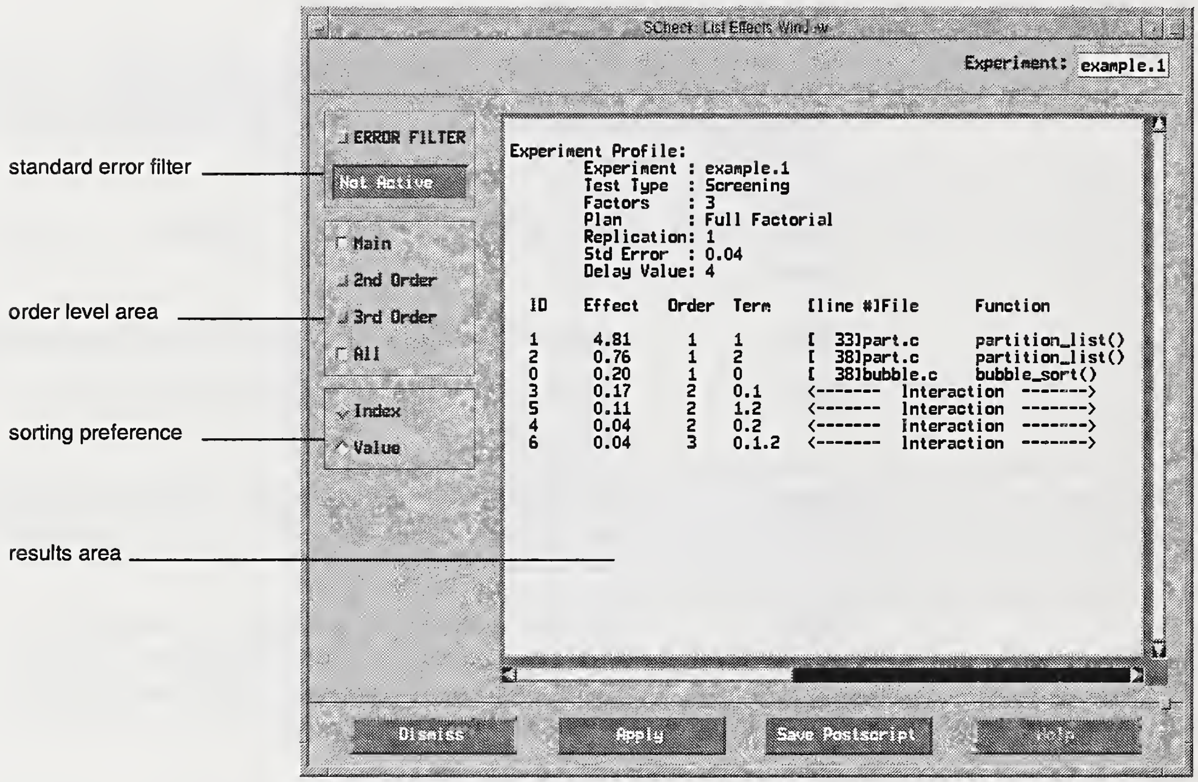
List Effects

The List Effects Window (Figure 9) displays results of the experiment. At the top of the results area an experiment profile is given. The profile includes experiment settings and the standard error. After this the effects for each factor and factor interaction are displayed. The list follows this format:

index effect order term [line #]file function text

where:

index	arbitrary identification assigned to a factor or factor interaction
effect	expresses importance of corresponding factor or factor interaction
order	describes the degree of interaction. An order of 1 indicates a main (standalone) effect. An order of 2 indicates a 2-factor interaction and so on.
term	describes the factor or factor interaction via an index. Index and term are the same for main effects. For effects with an order of two or more, the term is represented by combining the indexes of constituent factors. The indexes are delineated by periods (.).
[line #]file	indicates the file and line number for the factor. If the factor is an interaction then that is indicated instead.
function	is the function in which the factor resides.
text	displays the source code corresponding to the factor.



```
Experiment Profile:
        Experiment : example.1
        Test Type  : Screening
        Factors    : 3
        Plan       : Full Factorial
        Replication: 1
        Std Error  : 0.04
        Delay Value: 4

  ID    Effect   Order  Term    [line #]File      Function

  1      4.81      1    1       [  33]part.c      partition_list()
  2      0.76      1    2       [  38]part.c      partition_list()
  0      0.20      1    0       [  38]bubble.c    bubble_sort()
  3      0.17      2    0.1     <-------  Interaction  ------->
  5      0.11      2    1.2     <-------  Interaction  ------->
  4      0.04      2    0.2     <-------  Interaction  ------->
  6      0.04      3    0.1.2   <-------  Interaction  ------->
```

FIGURE 9. List Effects Window

The list control area gives the user the option to set preferences for displaying the effects. You can control a standard error filter, the level of the order to be displayed, and the sorting of effects by index or by value.

Setting the standard error filter determines how significant an effect should be to be displayed. For example, if the standard error is 0.25 and the standard error filter is set to 5, then only effects that are greater than 5 * 0.25 = 1.25 are displayed. Press *Apply* (or hit <return>) to activate the new setting. The default value for the standard error filter is one standard error (if activated).

The level of interaction, or order, of effects to be displayed is controlled by selecting the desired order in the order level area. Selecting the order will display effects of that order. More than one order may be selected. Once all your selections are final, press *Apply* to see the changes. The default displays only main effects, but in parallel processing, second order effects cannot be dismissed.

Effects are sorted either by their value or by their index. Select the desired ordering and press the *Apply* button to activate the changes. Sort-by-value is the default.

If you are changing more than one setting in the list control area, you can wait until all of your selections are made before pressing *Apply.*

List information can be saved in a postscript file. Select the *Save Postscript* button. This action brings up a dialog containing a list of files in the current working directory and a text entry field. Enter the name of file you wish to save the effects list in and click the *Save* button. The implementation saves the currently selected display. All postscript files are saved in the *experiment directory*, regardless of the experiment.

Plot Effects

Four plots are available to visualize S-Check analysis results (Figure 10):

- absolute effects plot
- mean plot
- normal probability plot
- half normal probability plot

By default, all four plots appear in the display area when the Plot Effects Window is popped up. Each plot can be requested individually by clicking on the plot or by using one of the available push buttons. Clicking again in the display area brings you back to the previous display.

The plot control area gives you the option to set preferences for displaying the effects. You can control a standard error filter, the level of the order to be displayed, and the sorting of effects by index or by value. Unless otherwise specified, these options apply only to the absolute value of effects and half effects plots.

Next and *Previous* allow you to browse through the results whenever a single screen cannot handle the complete set of results at once. A screen can plot up to 32 data points.

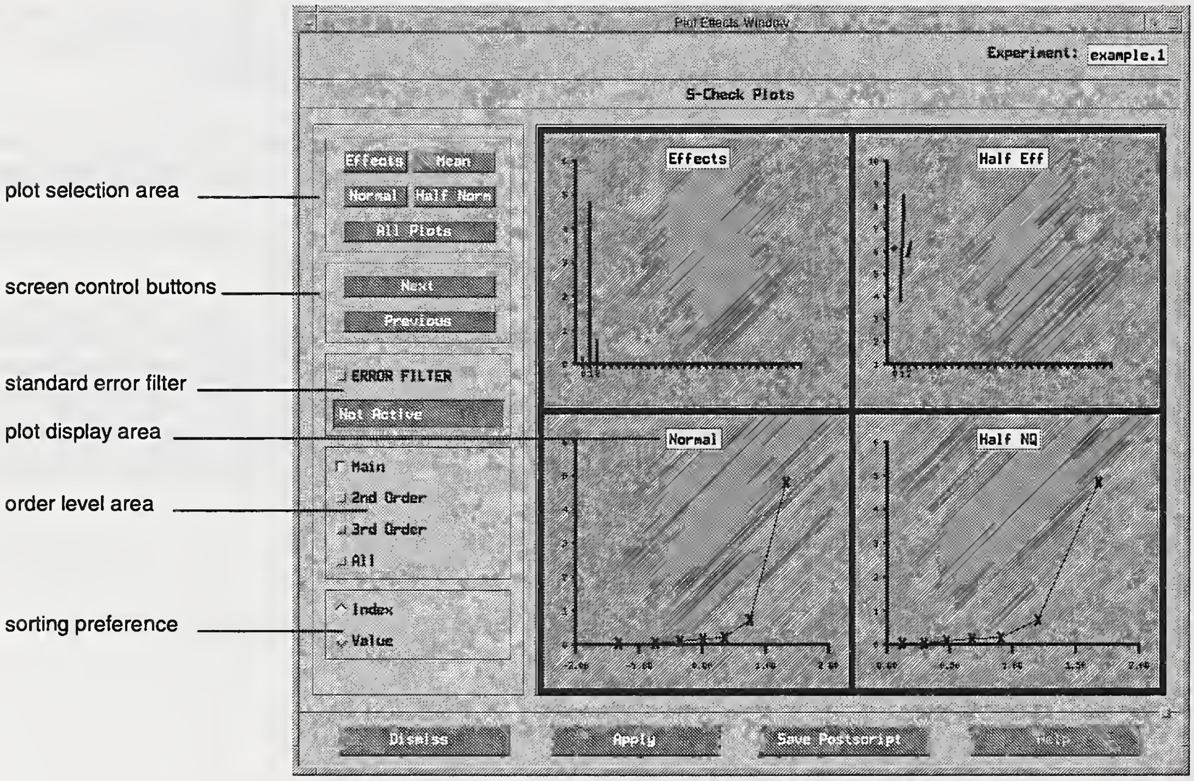


FIGURE 10. **Plot Effects Window**

The error filter applies only to the absolute value effects plot. It can be used to ignore factors that are not considered to be significant. To use this filter, select the *ERROR FILTER* toggle button. If a standard error exists, this will activate the text field immediately below the toggle button. Enter the standard error factor and hit <return>. This action will highlight effects that surpass the threshold set by the filter. An effect must be greater than the standard error multiplied by the standard error filter to be highlighted (displayed in another color). In addition, a highlighted horizontal line is drawn at the threshold limit. The default value for the standard error is one (if activated).

The level of interaction, or order, of effects to be displayed is controlled by selecting the desired order in the order level area. More than one order may be selected.

Once all your selections are final, press *Apply* to see the changes. The default displays only main effects.

Interaction availability is dependent on the experiment type. *Order* level toggle buttons are *grayed out* when interactions are not available.

Effects are sorted either by value or by index. Select the desired ordering and press the *Apply* button to activate the changes. In contrast to the List Panel, sort-by-index is the default.

If you are changing more than one setting in the plot control area, you can wait until all of your selections are made before pressing *Apply.*

A description of each plot follows:

Absolute Effects Plot

The absolute effects plot represents the absolute values of effects for the different factors under study and their interactions (when available). The taller the line, the larger the absolute value of the effect (y axis). Numbers on the x axis are arbitrary identifications assigned to factors. Use the rank-ordered list (List Effects Window) to link indexes with their corresponding term.

Mean Plot

The effect of a factor is given by the difference in the means for all high and all low settings of the factor. The mean plot is generated by drawing a line between the mean at the low and high setting of a factor for every factor and factor interaction (when available). The longer the line, the larger the effect. As with the absolute effects plot, factors are identified by an arbitrary index (x axis).

Normal Probability Plot

In a normal probability plot, effects (y axis) are arranged in ascending order of their value (i.e., from smallest to largest) and plotted against the theoretical standard normal percentiles (x axis). Theoretical standard normal percentiles are the cumulative values one would expect if the effects arose from a normal distribution centered about zero with a standard deviation of one. If the effects in an experiment are generated purely by noise, then the points would tend to fall on a line passing through the origin. Outliers from this line indicate significant effects and so highlight potential performance bottlenecks. Data points (i.e., effects) are arranged in ascending order of their value.

Half Normal Probability Plot

The half normal probability plot is similar to the normal probability plot, except that effect estimates are arranged by their absolute values (y axis) and plotted against the theoretical standard half-normal percentile (x axis).

The plots can be saved in a postscript file. Select the *Save Postscript* button. This action brings up a dialog containing a list of files in the current working directory and a text entry field. Enter the name of file you wish to save the plots in and click the *Save* button. The current implementation saves all four plots by default, there is no feature available yet to save individual plots. In addition to the plots a list correlating the factors indexes to their corresponding terms is provided. All post-script files are saved in the *experiment directory*, regardless of the experiment.

Multiple Displays

The Multiple Displays Window (Figure 11) allows the user to view and compare results from previously saved experiments without having to load each experiment separately. List and Plot panels can be launched for any results file listed in the results list. This list displays all the result files saved for all the experiments under the current experiment directory. Results can be saved by accessing the *Save* menu selection or the *Save Results* menu selection on the Experiment Control Window. If you want to display a results file, click in the corresponding toggle area and press List or Plot to obtain a display of the data. More than one results file may be selected at a given time. To see multiple results together, you must remember to move the displays around (they are stacked up initially) if your window manager automatically places windows.

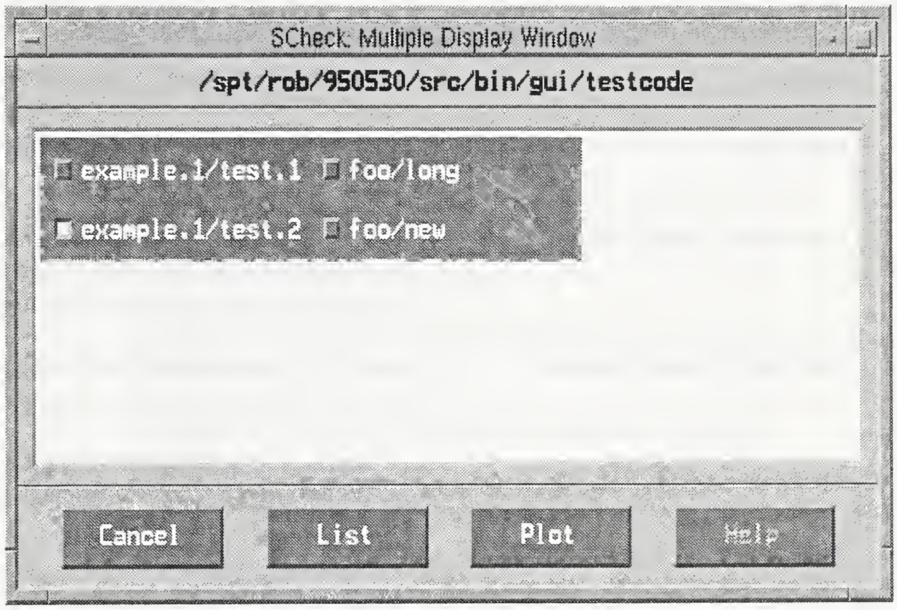

SCheck: Multiple Display Window
/spt/rob/950530/src/bin/gui/testcode
example.1/test.1
foo/long
example.1/test.2
foo/new
Cancel
List
Plot
Help

FIGURE 11. Multiple Displays Window

Glossary

analysis of variance (ANOVA)	statistical method used for the purpose of calculating effects.
effect	expresses importance of a factor or factor interaction.
experiment	an S-Check entity that defines parameters needed to perform an SPS analysis of the test program.
experiment configuration	items that define settings that allow S-Check to build (make) the test program. It also defines other setup information such as where to run the experiments and the type of the experiment.
experiment control settings	items that define settings directly associated with the experiment, e.g., factors, plan type, delay value, and replication.
experiment directory	the directory in which S-Check was started. Experiments are saved in this directory under the sub-directory named *.scheck.*
delay value	the amount of delay.
DEX	design of experiments.
factor	parameter that is being varied and tested (e.g., source code segments, synchronization barriers, etc.).
host machine	the machine on which S-Check is running.
replication	indicates the number of times an experiment is performed.
response interval	defines the start and stop locations for the purpose of capturing the response time.
response time	actual execution time for response interval. A response time is captured for each trial run.

Glossary

SPS rank	rank-ordered list of source code segments based on the relative sensitivity (effects) of the test program to synthetic delays associated with the code segments.
standard error	estimate of the standard deviation of observed effects.
target machine	the machine on which the S-Check experiments are performed.
test program	executable test program for the experiment.
treatment	instrumentation pattern (of delays) obtained from the experimental plan for a trial run.
trial	is an instance of the test program with a particular treatment.
work set	set of source code files needed to build the test program.
working directory	the directory in which files are accessed to build the test program. This directory will either be the directory in which S-Check was started (experiment will run on local machine) or the directory specified on the Configuration Window for remote machine access.

Warnings and Bugs

Cancel button on Factor Editors

The Cancel button on the Factors Editors is not implemented yet. It currently performs the same functions as the OK button.

Standard Error Messages

Messages sent to the standard error (stderr) from the test program are not displayed to the user.

C preprocessor anomalies

The alpha 1.0 version of S-Check takes advantage of the target machine's C processor by using the '-E' option to cc(1). Therefore, S-Check only parses the preprocessor output, not the original code. This can be confusing when selecting factors. For example:

```
#define TRUE 1
 foo = TRUE;
```

If the user clicks on the above statement, the highlighted result is 'RU' (since S-Check's input is 'foo = 1;'). The instrumented code will be correct, just shown to the user incorrectly. The user is advised to avoid selecting factors on macros that contain executable code. The results for the factor locations, in this case, are unknown.

Parsing the preprocessor output also causes another problem. Consider:

```
foo.h:
   #ifndef CORRECT
   you forgot to define CORRECT
   #endif
  foo.c:
   #define CORRECT 1
   #include <foo.h>
   main(){ }
```

Since S-Check never parses the macro CORRECT, the attempt to build an instrumented executable will fail. These problems will be corrected when a C preprocessor is integrated into S-Check. A work around is to add the #define to the special C-flags (i.e., -DCORRECT) for the file(s) in question on the Configuration Window.

Running experiments simultaneously

Results from running two or more experiments simultaneously are unpredictable. This feature has not been thoroughly tested and should be avoided. In addition, running experiments simultaneosly where the test program uses shared files (e.g., on SGI machines, shared arena) may cause problems.

Cannot allocate colormap entry for "color"

The server can't allocate the colormap entry for "color". Too many X-clients are currently running on the system. Exit one or more of these clients if you want to display the default S-Check colors.

Response interval tags (B and E) placement with source code modifications

If the response interval tags (B and E) are set and the source code is modified in which the response interval tags are set, the tags are no longer valid. Results are unpredictable. No warning or error message is provided to the user. The user should reset the response interval tags after source code modifications have been made. Modifying source code in which factors are selected is handled correctly.

Instrumenting switch statements

As mentioned in *Selecting Factors,* caution should be exercised when defining factors in compound statements. This is especially true when instrumenting `switch` statements. Selecting the `switch` statement itself will place the instrumentation code at the bottom of `switch` (by design), which under most circumstances is never reached. It is best to placed the factors in the individual cases that you want to instrument.

References

[1] G. Lyon, R. Snelick, R. Kacker.
Synthetic-perturbation tuning of MIMD programs,
The Journal of Supercomputing 8 (1) (1994), 5-28.

[2] R. Snelick, J. JaJa, R. Kacker and G. Lyon.
Synthetic-perturbation techniques for screening shared memory programs,
SOFTWARE--Practice and Experience 24 (8) (1994), 679-701.

[3] G. Lyon, R.Kacker, and A. Linz.
A simple scalability test for parallel code,
SOFTWARE--Practice and Experience, publication pending, June, 1995.
(Also appears in Proc., IEEE 2nd Int. Symposium on Software
Metrics, October, 1994, London, 54-60.)

Appendix A: error messages

Appendix A gives a list of possible error messages from S-Check. Errors of this sort are the result of invalid user input or input that is unrecognizable by S-Check. Adjustments (or input corrections) must be made before S-Check can proceed in the direction before the error occurred. Note that all other system functions are frozen until the user acknowledges the error by clicking on the OK button in the error popup dialog. The table below provides possible reasons and solutions to the errors. If you are unfamiliar with how S-Check instruments the source code, it may be useful to review Appendix C (Code Instrumentation). The information provided there will let you better understand the error messages.

Message	**Reason/Fix**
The program “X” is not in your path.	The executable “X” is not contained in any of the directories specified by your PATH variable. You must quit S-Check and correctly modify your PATH and restart S-Check.
The program “X” received a “Y” signal.	The program “X” received a termination signal “Y” from the UNIX operating system. If “X” was **CInstGen** or **Cparser**, this is an S-Check bug and should be reported. If “X” was **a.out**, a bug in the user’s test program exists. If “X” is **cc**, this is a compiler bug.
The program “X” received unknown signal “Y”.	The program “X” has received an unknown an unknown signal “Y” from the Unix operating system. “X” can be one of the following: **a.out**, **cc**, **Cparser**, or **CInstgen**.
“X” is not a regular file	S-Check expected a regular file. Check file type of file “X”.
The generator has returned unknown code “X”.	The program **CInstGen** has terminated with an exit status unknown to S-Check. This is a bug in S-Check and should be reported.
Invalid delay value: “X”, delay is not set. OR Invalid delay value: “X”, delay is still “Y”	The delay value entered manually does not fall in the range 0 to DELAY_MAX_VALUE. DELAY_MAX_VALUE can be modified when S-Check is installed. It’s default value is 1,000,000.
CTree_Load failed	Unable to load the “X.tree” file because it does not conform to a known format.

Message	Reason/Fix
Ran out of memory for source file load.	Ran out of memory (malloc() failed). Close other windows, stop other processes, increase swap space, add memory, etc.
bad read for file "X"	File "X" does not conform to a known format. If "X" is config then experiment save file is corrupted. You may not be able to savage the saved experiment.
bad read for response.scheck file, execution time unknown	The file response.scheck does not conform to a known format. The Begin (B) and End (E) markers were probably improperly placed. Check to see if the E was executed, or that E was executed before B.
bad factor kind found in config file	The config file does not conform to a known format. The file may be corrupted or the file may be incompatible with the current S-Check version.
illegal stmt index for factor in config file	The identified factor index is out of range. The config file is corrupted. The factor in question is removed from the factor list.
The parser has returned unknown code "X"	This is an S-Check bug and should be reported.
Invalid file name, file not saved.	An invalid file name was entered. The file was not saved, try another name.
Can't load results file: "X"	The results file "X" can not be read, contains corrupted data, or host system memory is low.

Appendix B: standard error table

The standard error of an experiment is used to assess the quality of S-Check's results (rank lists and plots). However, depending on the experiment, a standard error may not be provided or its accuracy may be limited. In these cases, valid conclusions can still be made from S-Check's results, although with diminishing confidence.

The table below shows availability of the standard error for an unreplicated experiment. The standard error for unreplicated plans is calculated by treating interaction effects of three or more as error. When third order interaction effects are not available, second order interaction effects are used for the standard error estimate. This is the case for resolution IV plans when the number of factors is ten or greater.

The plan and the number of factors determine the accuracy of the standard error. A "YES" indicates that the standard error is calculated with sufficient precision and can be used with confidence in analyzing the results. A "YES*" indicates that a standard error is provided, but it may not be very reliable. Typically, unreplicated plans are used when the number of factors is six or more. When the number of factors is less than six, it is generally possible to replicate the experiment and calculate the standard error from replication.

	Factors						
Plan	**2**	**3**	**4**	**5**	**6-7**	**8-9**	**10+**
Full Factorial	NO	YES*	YES*	YES	YES	YES	YES
Half Factorial	N/A	NO	NO	NO	YES	YES	YES
Quarter Factorial	N/A	N/A	N/A	NO	YES*	YES	YES
Resolution IV	NO	NO	NO	NO	NO	NO	YES
Resolution III	NO	NO	NO	NO	NO	NO	NO

NO: no standard error is possible for the plan

YES: the standard error is calculated with sufficient precision

YES*: the standard error is calculated with insufficient precision

N/A: the plan is not available for the specified number of factors

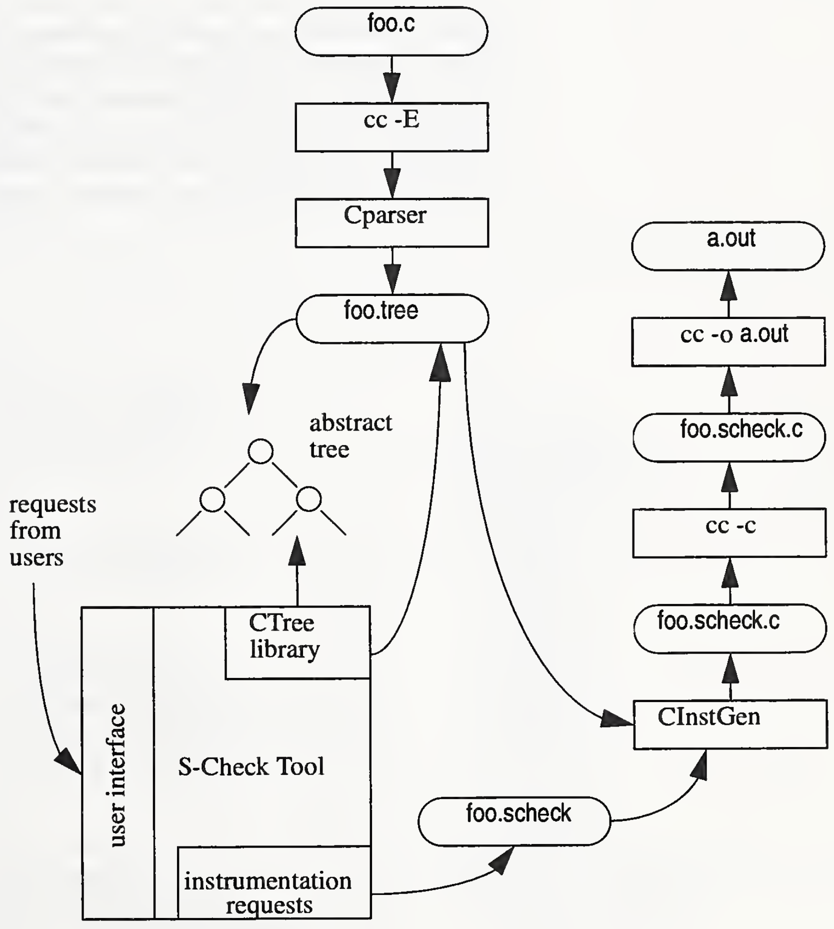
Note: a standard error is always available when the experiment is replicated.

A "NO" indicates that no standard error is calculated for the experiment. In the "YES*" and "NO" cases, inferences of the data can still be made but it should be done judiciously. For example, when the system load (and noise) is low, the estimate of the standard error may not be necessary. In such cases a resolution III plan may be used for screening. In contrast, investigating interactions between send/receive pairs in a highly congested (noisy) communication system may warrant a sound standard error estimate.

Decisions about the importance of the standard error for analyzing S-Check's results can be aided with information about the test program and host system. Knowledge from previously-run experiments is especially useful. In addition, the significance of a factor in relationship to other factors in lieu of adequate standard error information can still be determined through the use of normal probability plots. If significant outliers exist in the results, these plots highlight them as points not on a straight line.

Appendix C: Code Instrumentation

Appendix C explains the method in which S-Check instruments the source code program.



Given the file foo.c, S-Check runs the default C preprocessor for the purpose of including include files and expanding macros. The source code output from the C preprocessor is then analyzed with the **Cparser** program. The **Cparser** builds an abstract tree containing lexical and semantic information about the source code. This information is stored in the file foo.tree. A separate tree is built for each source file. S-Check (CTree library) uses the abstract tree to make inferences about the code and to re-generate the original source code with instrumentation. Inferences about the code allow for basic block factor selection, requests such as " instrument all while loops in function X()" are possible. This feature is not implemented yet.

To instrument and re-generate the source code, two inputs to the code generator program (**CInstGen**) are required: the abstract tree and the user's instrumentation requests. Instrumentation requests are gathered and translated by S-Check and are saved in foo.scheck. The files foo.scheck and foo.tree are fed as inputs to **CInstGen** which yields the instrumented source code (foo.scheck.c). This code is then compiled with a compiler of the user's choice to obtain object code (foo.scheck.o). Other object files are then linked together to create the executable program.

Index

How to install S-Check

S-Check software can be obtained via the ftp site **www.scheck.nist.gov**. You can also get to this location by accessing the **http://www.scheck.nist.gov** Web address. S-Check 1.0 Alpha release works on Silicon Graphics and Sun parallel machines. S-Check analyzes any code that adheres to either Kernighan and Ritchie or ANSI C. The graphical interface is written with the OSF Motif toolkit (version 1.2). You must have the Motif libraries to compile S-Check.

Steps:

1. Retrieve the compressed tar file *scheck_1.0A.tar.gz* from the scheck directory.
2. Use gunzip to un-compress the file.
   % **gunzip** *scheck_1.0A.tar.gz*
3. Use tar to extract the files.
   % **tar** xf *scheck_1.0A.tar*
4. Change directory to scheck_1.0A
   % **cd** scheck_1.0A
5. Configure the package for your system.
   % **configure**

   > This command automatically configures the software source code package for your particular system. Among other things, it builds the Makefiles necessary to create the executables.

6. Make the executables.
   % **make**
7. Make sure the executables (**CInstGen**, **Cparser**, and **scheck**) in the scheck_1.0A/bin directory are in your path.
8. Copy the resource file **Scheck** to your home directory on the machine your running scheck on.
9. Refer to the user's guide, *Using S-Check,* to get started with the tool.

Note: S-Check has a X-windows user interface, so remember to set your $DISPLAY environment variable to your local X-server and add the client machine to your xhost list. Use the command setenv DISPLAY your_X-server_machine:0 on the client machine to set the DISPLAY variable, and the command xhost +client_machine on your local X-server to allow the client machine to open a window on your console.

Send questions or comments to **scheck-tool@www.scheck.nist.gov**.

S-Check Quick-Reference
Alpha Release 1.0
(some transitions omitted)
Written by: Gordon Lyon

getting started
- put all ".c" parts of test program in same directory
- type the command scheck

SCheck: Experiment List Window
- select <name> from Experiment List Window to rerun or modify old experiment

or
- type in <new experiment name> to build completely new experiment

SCheck: Configuration Window
- select directory contents and add to work set. Set flags as needed for host parallel system. Exit via *OK* button.

SCheck: Experiment Control Window
- set *DEX Plans* on *automatic*, *Replication(s)* = 1 or 2
- double-click on files to establish or change response timing interval and code segments (factors) of interest

SCheck: Factor Editor Window
- click on code parts suspected as bottlenecks. A **black mark** will indicate selection. A second click deletes the selection. The window indicates the number of selected factors. Factor selection is always active unless you hit the response selection button. Exit via *OK*.
- the *Select Response* button in this window sets points *B-E* between which experiment timings will occur. This response is usually set to catch run time for the whole program, so the window margin markers of *B* and *E* should be set in the main driver. (The code being instrumented is indicated in the upper left-hand corner of the window.) The cursor will change to **red** and be directional during response point setting.
- exit of the window leads back to the Experiment Control Window, where other files can be set up or the experiment begun (see below).

setup con-tinues

setup stage done

SCheck: Experiment Control Window
- hit the *Build* button to initiate compilation and selection of delay size. Or, you can use the *Utilities* menu option to set the delay manually, should you wish to do so.
- when the *Start* button becomes enabled, hit it to begin (errors will prevent this)
- when the experiment is done, use the *Display* menu selection to plot or list results.

List or Plot Effects Window
- if there is an error estimate, the error filter can be set in multiples of the standard error (*e.g.*, 2SE is a 95% confidence acceptance level). Any effect less than the Standard Error (SE) is probably not significant. Even 2SE is marginal for SCheck Technology.