Test suite

Sparse has a number of test cases in its validation directory. The test-suite script aims at making automated checking of these tests possible. It works by embedding tags in C comments in the test cases.

Tag's syntax

`check-name:`

Name of the test. This is the only mandatory tag.

`check-description:`

A description of what the test checks.

`check-command:`

There are different kinds of tests. Some can validate the sparse preprocessor, while others will use sparse, cgcc, or even other backends of the library. check-command allows you to give a custom command to run the test-case. The `$file` string is special. It will be expanded to the file name at run time. It defaults to `sparse $file`.

`check-arch-ignore:`

`check-arch-only:`

Ignore the test if the current architecture (as returned by `uname -m`) matches or not one of the archs given in the pattern.

`check-assert:`

Ignore the test if the given condition is false when evaluated as a static assertion (`_Static_assert`).

`check-cpp-if:`

Ignore the test if the given condition is false when evaluated by sparse's pre-processor.

`check-exit-value:`

The expected exit value of check-command. It defaults to 0.

`check-timeout:`

The maximum expected duration of check-command, in seconds. It defaults to 1.

`check-output-start` / `check-output-end`

The expected output (stdout and stderr) of check-command lies between those two tags. It defaults to no output.

`check-output-ignore` / `check-error-ignore`

Don't check the expected output (stdout or stderr) of check-command (useful when this output is not comparable or if you're only interested in the exit value). By default this check is done.

`check-known-to-fail`

Mark the test as being known to fail.

`check-output-contains:` *pattern*

Check that the output (stdout) contains the given pattern. Several such tags can be given, in which case the output must contains all the patterns.

`check-output-excludes:` *pattern*

Similar than the above one, but with opposite logic. Check that the output (stdout) doesn't contain the given pattern. Several such tags can be given, in which case the output must contains none of the patterns.

`check-output-pattern(` *nbr* `):` *pattern*

`check-output-pattern(` *min* `,` *max* `):` *pattern*

Similar to the contains/excludes above, but with full control of the number of times the pattern should occur in the output. If *min* or *max* is `-` the corresponding check is ignored.

`check-output-match(` *start* `):` *pattern*

Check that in the output (stdout) all lines starting with the first pattern also contains the second pattern. This should be reserved for matching IR instructions since the '.$size' suffix is ignored in the first pattern but is expected to be followed by a space character.

`check-output-returns:` *value*

Check that in the output (stdout) all IR return instructions have the given value.

Using test-suite

The test-suite script is called through the check target of the Makefile. It will try to check every test case it finds (`find validation -name '*.c'`). It can be called to check a single test with:

```
$ cd validation
$ ./test-suite single preprocessor/preprocessor1.c
     TEST     Preprocessor #1 (preprocessor/preprocessor1.c)
preprocessor/preprocessor1.c passed !
```

Writing a test

The test-suite comes with a format command to make a test easier to write:

```
test-suite format [-a] [-l] [-f] file [name [cmd]]
```

name: **check-name value**

If no name is provided, it defaults to the file name.

cmd: **check-command value**

If no cmd is provided, it defaults to `sparse $file`.

The output of the test-suite format command can be redirected into the test case to create a test-suite formatted file.:

```
$ ./test-suite format bad-assignment.c Assignment >> bad-assignment.c
$ cat !$
cat bad-assignment.c
/*
 * check-name: bad assignment
 *
 * check-command: sparse $file
 * check-exit-value: 1
 *
 * check-output-start
bad-assignment.c:3:6: error: Expected ; at end of statement
bad-assignment.c:3:6: error: got \
 * check-output-end
 */
```

The same effect without the redirection can be achieved by using the `-a` option.

You can define the check-command you want to use for the test.:

```
$ ./test-suite format -a validation/preprocessor2.c "Preprocessor #2" \
                "sparse -E \$file"
$ cat !$
cat validation/preprocessor2.c
/*
 * This one we happen to get right.
 *
 * It should result in a simple
 *
 *      a + b
 *
 * for a proper preprocessor.
 */
#define TWO a, b

#define UNARY(x) BINARY(x)
#define BINARY(x, y) x + y

UNARY(TWO)
/*
 * check-name: Preprocessor #2
 *
 * check-command: sparse -E $file
 * check-exit-value: 0
 *
 * check-output-start

a + b
 * check-output-end
 */
```