Compilation Consistency Checks

D. Pleiter

31 January 2007

The first section of this document specifies the system which allows at each compilation step to add software specific information, to impose requests on options used for a particular software, and to check the consistency of information and requests at each stage of the compilation chain. The second section describes the implementation by a perl module.

1 Specification

1.1 `SW_INFO`

At each compilation step the involved software may insert information about themselves. There is a special information field indicated with the keyword `comment` which is not supposed to be used for checks and can be used to pass arbitrary user level information.

Note that it is recommended to only have the linker adding the prefix `id-<id>:`.

- **Revision:**
  Syntax: `[id-<id>:]<swname>:rev:<revision>`

`swname`	Name of software
`revision`	One or more integers separated by '.'

- **Options:**
  Syntax: `[id-<id>:]<swname>:opt<optid>:<value>`

`swname`	Name of software
`optid`	Option identifier
`value`	Integer value

- **User information:**
  Syntax: `[id-<id>:]<ns>:comment:<string>`

`ns`	User defined namespace (must be non-empty)
`string`	User defined string terminated by newline

1.2 SW_REQ

At each compilation step the involved software may insert request for certain conditions to be true. Each request may optionally be marked `global` or `local`. If neither of them is specified, `global` is implicitly assumed. If a request is marked `local` and an `<id>` has been defined, only those `SW_INFO` will be considered where `<id>` matches.

- **Revision:**

  Request the revision of a particular software to fulfill a particular condition, e.g. request a certain or more recent revision of this software.

Syntax:	`[{local[-<id>]|global}:]<swname>:rev:<relop>:<revision>`	
	`swname`	Name of software to be checked
	`relop`	Relational operator
	`revision`	One or more integers separated by '.'

- **Option:**

  Impose requests on options used for a particular software, e.g. check for mutually exclusive options.

Syntax:	`[{local[-<id>]|global}:]<swname>:opt<optid>:<relop>:<value>`	
	`swname`	Name of software to be checked
	`optid`	Option identifier
	`relop`	Relational operator
	`value`	Integer value

- **Require:**

  Before starting to execute a particular software, require another software to have been executed.

Syntax:	`[{local[-<id>]|global}:]<swname_a>:requires:<swname_b>`	
	`swname_a`	Name of software executed after
	`swname_b`	Name of software executed before

- **Check:**

  Request a particular software to perform a particular check.

Syntax:	`[{local[-<id>]|global}:]<swname>:check:<checkid>[:arg...]`	
	`swname`	Name of software to perform check
	`checkid`	Identifier of check to be performed
	`arg`	Optional arguments for check

The following relational operators are defined: `eq`, `ne`, `gt`, `ge`, `lt`, `le`.

1.3 Linker behaviour

The linker will perform the following operations:

1. Select or generate a UUID `<uuid>` for each module

2. Foreach SW_INFO that does not match the regular expression "^id-" add prefix "id-<uuid>".

3. Foreach SW_REQ that matches the regular expression "^local:" replace this substring by "local-<uuid>:".

1.4 Matching rules

1. Any combination of SW_INFO and SW_REQ will only be considered if any of the following rules apply:

   a) The SW_INFO did not contain an <id>.

   b) The SW_REQ did not contain a qualifier local and a corresponding <id>.

   c) The <id> of the SW_INFO and SW_REQ match.

2. In case of requests on revision or options any combination of SW_INFO and SW_REQ will only be considered if <sw_name> match.

3. In case of requests of type require at least one SW_INFO for <swname_a> must exist.

1.5 Example

In sasm or masm files the information and request fields may be kept in pragmas:

```
PRAGMA_SW_INFO os1:rev:0.9
PRAGMA_SW_REQ npsk:opt+a:ne:0
PRAGMA_SW_REQ nose:rev:ge:1.62 nose:opt-os7:ne:0

PRAGMA_SW_INFO mpp:rev:1.127 mpp:opt-r:1
PRAGMA_SW_REQ nose:opt-r:ne:0

PRAGMA_SW_INFO npsk:rev:0.992.2 npsk:opt+a:0
PRAGMA_SW_REQ npsk:requires:sofan
```

In above example the check for the requested shaker option '+a' and the check for execution of sofan should fail. Additionally, several warnings should be issued as the check for various nose options can not be performed.

In mem files the following syntax may be used:

```
!! SW_INFO: os1:rev:0.9
!! SW_REQ:  npsk:opt+a:ne:0
!! SW_REQ:  nose:rev:ge:1.62 nose:opt-os7:ne:0
```

The following example shows that in a linked module there may be conflicting "local" requests:

```
!! SW_INFO: id-a:mpp:rev:1.140
!! SW_INFO: id-a:os1:rev:0.9
!! SW_REQ: local-a:npsk:opt+a:ne:0
!! SW_REQ: global:nose:rev:ge:1.62 global:nose:opt-os7:ne:0
!!
!! SW_INFO: id-a:npsk:rev:0.992.2 id-a:npsk:opt+a:1

!! SW_INFO: id-b:mpp:rev:1.140
!! SW_INFO: id-b:os1:rev:0.9
!! SW_REQ: local-b:npsk:opt+a:eq:0
!!
!! SW_INFO: id-b:npsk:rev:0.992.2 id-b:npsk:opt+a:0

!! SW_INFO: nose:rev:1.63 nose:opt-os7:1
```

2 Implementation

2.1 Perl library

The perl library provides the following functions:

`parseLine()` Parse line provided on input for PRAGMA_SW_INFO and PRAGMA_SW_REQ and push content to arrays INFO and REQ.

`verify()` Verify checks: Evaluate pieces of information and requests stored in the hash arrays INFO and REQ. Returns a list of error and warn messages.

`verifyReq()` Parse line provided on input for PRAGMA_SW_REQ and verify software request using the information stored in INFO. Hash array REQ remains unchanged.

2.2 Executable

`swcheck [-q] file{.sasm|.masm|.mem}`

Program checks the consistency of all `SW_INFO` and `SW_REQ` entries in the input file and prints a list of errors and warnings if any. Output is surpressed when the option `-q` is used. The exit status may be used to obtain the number of detected errors.