Check which version of GNU Make is running

Because GNU Make is regularly updated and new features are added (notably the `$(eval ...)` function in GNU Make 3.80) it's sometimes important to know the version of GNU Make that's processing a Makefile, or even whether a specific GNU Make feature is available.

There are two ways to do that: look at the `MAKE_VERSION` variable (which was introduced in GNU Make 3.68.5 in 1993---if you are using an older GNU Make then it's time to upgrade!), or look in the `.FEATURES` variable (added in GNU Make 3.81). It's also possible to check for specific features, like `$(eval...)`.

MAKE_VERSION

The `MAKE_VERSION` contains the version number of the GNU Make that's processing the Makefile in which `MAKE_VERSION` is referenced. Here's an example Makefile that prints the version of GNU Make and stops.

```
.PHONY: all
all: ; @echo $(MAKE_VERSION)
```

And here's the output generated when GNU Make 3.80 parses that Makefile

```
$ make
3.80
```

A common task is to determine if *at least* a specific version of GNU Make is running. For example, you might want to determine that the Makefile is being handled by at least version 3.79.1.

If you assume that GNU Make version numbers are always in the form X.YY.Z or X.YY (where X is a single digit and YY is always a pair) then the following fragment will set the `ok` variable to non-empty if the version mentioned in `need` is equal to or less than the running version of GNU Make.

```
need := 3.79.1
ok := $(filter $(need),$(firstword $(sort $(MAKE_VERSION) \
                                          $(need))))
```

If `ok` is not blank (in fact if it's not blank it'll have the same value as `$(need)`) then at least the required version of GNU Make is being used; if it's blank then the version of GNU Make is too old.

This works by creating a space-separated list from the version of GNU Make that's running (from `MAKE_VERSION`) and the version required (from `need`) and sorting that list. The first word of the sorted list will be `$(need)` if the `$(MAKE_VERSION)` is greater than or equal to `$(need)`. If it's not `$(need)` then the `$(filter...)` will remove it and `ok` will be blank.

Note: this fragment won't work correctly with versions of GNU Make starting at 10.01 (but that's a long way off!).

`.FEATURES`

GNU Make 3.81 introduced the `.FEATURES` default variable that contains a list features supported by GNU Make. At the time of writing there are seven features listed (for GNU Make 3.81):

1. `archives`: archive (ar) files are supported using the archive(member) syntax;
2. `check-symlink`: The `-L` and `--check-symlink-times` flags are supported;
3. `else-if`: Else branches in the non-nested form `else ifX` are supported;
4. `jobserver`: Parallel building using the job server is supported;
5. `second-expansion`: Double expansion of prerequisite lists is supported;
6. `order-only`: Has order-only prerequisite support;
7. `target-specific`: Target-specific and pattern-specific variables are supported.

To check if a specific feature is available the following `is_feature` function can be used. `is_feature` returns `T` if the requested feature is supported; if the feature is missing `is_feature` returns an empty string.

```
is_feature = $(if $(filter $1,$(.FEATURES)),T)
```

For example, the following Makefile uses `is_feature` to echo whether the `archives` feature is available:

```
.PHONY: all
all: ; @echo archives are $(if $(call is_feature,archives),,not)
available
```

And here's the output using GNU Make 3.81:

```
$ make
archives are available
```

If you want to check whether the `.FEATURES` variable is even supported then either use `MAKE_VERSION` as described above, or simply expand `.FEATURES` and see if it's empty or not.

This Makefile fragment results in `has_features` being `T` if the `.FEATURES` variable is present and contains any features:

```
has_features := $(if $(filter default,$(origin .FEATURES)),$(if
$(.FEATURES),T))
```

The fragment first checks that the `.FEATURES` variable is a default variable (using `$(origin...)`) so that `has_features` is not fooled if someone had defined `.FEATURES` in the Makefile. If it is, then the second `$(if...)` just determines whether `.FEATURES` is blank or not.

Detecting `$(eval...)`

`$(eval...)` is a powerful GNU Make feature that was added in version 3.80. If you use `$(eval...)` it is important to check that the feature is available in the version of GNU Make reading your Makefile.

You could use `MAKE_VERSION` as described above to check for 3.80, or the following fragment of code that sets the `eval_available` variable to `T` is `$(eval...)` is implemented and an empty string if not:

```
eval_available :=
$(eval eval_available := T)
```

This works because if `$(eval...)` is present it sets `eval_available` to `T` by evaluating `eval_available := T`, and if `$(eval...)` is not available then GNU Make does nothing when it encounters `$(eval eval_available := T)`.

(Actually, in versions prior to 3.80 when GNU Make sees `$(eval eval_available := T)` it tries to see if there's a variable called `eval_available := T` and get its value, which, of course, is empty.)

You can then use `eval_available` with `ifdef` to generate a fatal error if `$(eval...)` isn't implemented.

```
ifneq ($(eval_available),T)
$(error This Makefile only works with a Make program that
supports $$(eval))
endif
```

`eval_available` is especially useful if you can't check `MAKE_VERSION`. For example, if your Makefile might be run using a Make tools that emulates GNU Make (such as *clearmake* or *emake*) checking for a specific feature like `$(eval...)` is easiest with the trick above.