Seminar Configurable Systems

Prof. Zeller / Christian Lindig — Cons

Assignment

1. Read the manual page for *Cons*. While this is not a scientific paper it addresses a lot of important points and tries to offer a practical solution. It is not important to understand every little detail but to *identify the new ideas with respect to Make*. Cons as a tool was superceded by SCons, where Perl is replaced by Python. However, the documentation for SCons lacks the design rationale that is present in the Cons documentation.

   Unfortunately, there are only few good scientific articles about software build tools. A good albeit long one is *Hierarchical Modularity* by Blume and Appel in ACM Transactions on Programming Languages and Systems (21) 4, 1999. We are going to read a different one soon and will read more scientific papers in the second half of the seminar.

2. Prepare a one-page document that summarizes Cons. You should devote about half of your document to the summary of the paper *in your own words*. The rest you can use to judge the paper's strengths and weaknesses, and to offer some opinion.

   Here are some generic questions that you might want to answer in your summary:

   (a) What is the problem being solved or discussed?
   (b) What are the main ideas or concepts presented?
   (c) Do you find the paper convincing? What are the main claims of the author and does he provide evidence for them?

   Note all questions about the paper that you cannot answer yourself; we will try to answer them in class.

   All writing assignments may be completed in German or English. You will receive extra credit for writing in a foreign language.

   Your summary is always due the Tuesday before the next meeting, 11:30am at my office (307/45). Please provide me with a printout of your summary and don't forget to put your name on it.

3. In general, I do not accept submissions by email. However, if you cannot stop by my office and therefore decide to send me an email with your submission, send it as a PDF or plain text file.

4. Try out the following Makefile with GNU Make after creating a file `foo.a`. A generic rules `%.x: %.y` is equivalent to a suffix rule `.y.x` in classic Make: it tells how to make a file `*.x` from a file `*.y`. What does the result tell you about rules?

```
all: foo.c

%.c: %.b
touch $@

%.b: %.a
touch $@
```

What happens, if you add a rule `%.c: %.a`? Does this illustrate a potential problem? You don't need to include the answer to these questions into your write-up.