Assisting interface for efficient compilation of programs

Thibault Raffaillac <traf@kth.se>

Outline

Compiling a program nowadays is simple. Once the source code itself is written, a single action is needed to turn it to an executable file. With an IDE, it is synonym with clicking on a "Compile" button. With command-line tools such as GCC, it is computed with a single command. Still, such user-click-compiler-execute interaction is scarce. Apart from errors and warnings, compilers provide very little feedback on their operation.

On the other hand, programming languages are growing to higher and higher levels. As this brings ease to express increasingly complex needs, it makes it harder to foresee how the low level will eventually look. Moreover, as compilers become smarter and smarter, it is more difficult to guess *how* some sequence of instructions will be optimized. There are often several ways to achieve the same output, and the programmer wants to find the one the compiler (or all compilers) will "understand" the best.

There should be a way for compilers to "discuss" how well they performed on a source program. The user could then have the possibility to refine the input, to increase the efficiency of the output. An obvious benefit is performance of the executable files, but this would not be the only gain. Educationally, it would promote better coding practices, bring little known standards aspects to light and provide general knowledge on the operation of different architectures. On the security and reliability side, it could warn about alleged weaknesses like overflows or use of non-portable functions, and suggest alternate strategies. Last but not least, it would improve transparency of the compilation process, thus foster the trust granted by the user.

My goal for this Degree Project is to investigate the idea of an assistant handling three tasks of interaction with the programmer:

- *inform*: tells how well a portion of code was compiled, relates a compilation technique, promote a coding practice, introduces a feature from the standard, etc.
- *warn*: incites the user to correct a supposed flaw, notifies about a potential vulnerability which could arise with further lack of attention. As opposed to the previous task, here we demand some code to be fixed.
- *ask*: inquires a clarification about a portion of code. The answer can sometimes be expressed by updating the code, as when the compiler suspects a variable to be constant. Sometimes it also cannot be expressed with code, as for choosing the implementation of a tree structure (map in C++), or the character set (Latin-9, UTF-8, etc.) of a string object.

Having an embodied assistant is an option which will be considered. However, my main motivation being transparency, it shall *not* have an obscure operation similar to artificial intelligence. For the implementation I presently think of a stack of pending tasks, sorted by their importance. The user could iteratively ask for the next one, and the assistant would remember the remarks already made to further lower their importance.

The integration of the assistant should be possible for both graphical and command-line tools, thus the two cases will be considered. A priori, the target users are novice as well as professional programmers. I will try to involve serendipitous learning in the assistant tasks, and avoid gamification design so as to preserve the consideration that programming is a serious activity.

Work plan

As with the course DH2626 Interaction Design for which I had done a 2-months project, I intend to follow a work plan derived from the book "designing for interaction" by Dan Saffer:

- Define the design approach and clarify the problem (1 week).
- Develop a design strategy while conducting preliminary interviews (5 weeks). This consists in gathering and studying the existing works on the subject, and identifying the users' needs.
- Investigate as many ideas as possible to solve the problem (1 week).
- Refine the concepts selected, focusing on the details (2 weeks).
- Create a prototype and test its usability (3 weeks).
- Iterate on the prototype to comply with the usability observations (1 week).

The delays presented amount to half of the actual available time, to keep a margin for possible delays, and the redaction of the report. Additionally, in parallel with the work steps I will work on enumerating many possible messages from the assistant, and structure them by domain and architecture. I have already started working on this task through many technical courses, and intend to benefit from the help of teachers and professionals.