Programming Languages That Support Learning

Sebastian Thrun and Frank Pfenning

Problem: Today's computers a programmed using keyboards, by moving little plastic caps up and down. People are instructed by much richer means, involving instructions, demonstrations, explanations, trial-and-error learning, and so on. This project seeks to broaden the view of programming computers by integrating conventional programming with learning. In particular, it seeks to devise new programming languages that support software development by conventional programming, but that also support teaching and trial-and-error learning as a means to program computers.

Impact: The impact of this project, if successful could revolutionize the way we program computers. Software development could interleave phases of conventional programming with phases where we teach our software to do the right thing. Software, once written, could perpetually improve itself by adapting to changes in its environment. Our initial results, with a programming language called CES, demonstrate that the effort involved in developing software can, in selected cases, be reduced by two orders of magnitude or more.

State of the Art: The integration of programming and learning has barely been touched by the research community. We are not aware of any procedural programming language that would support learning. Conversely, the field of machine learning typically seeks to devise learning algorithms that can learn everything from scratch. Integrating procedural program code into a learning algorithm is, to our knowledge, an unexplored thing.

Approach: We have successfully developed a new prototype language, called CES (short for: C for Embedded Systems), which extends C by a learning mechanism. Software development in CES involves conventional programming and phases in which the program is taught. CES provides the necessary infrastructure to perform the credit assignment, necessary to train learning elements placed somewhere in the code.

CES also supports computing with probabilistic information. New data types are provided that represent probability distributions over existing data types, and new operations support computing with probabilistic information. This new way of computing is specifically tailored towards embedded systems, such as robots, where sensor limitations make it imperative to make decisions under uncertainty.

We successfully programmed our mobile robot Amelia to perform a gesture-driven mail-delivery task in our building, using as few as 144 lines of CES code. To make this program work, it was trained using hand-labeled data collected in less than one hour. The resulting program, which indeed bridges the gap all the way between raw sensor readings and direct motor commands, illustrates that software developed in CES is more than two orders of magnitude more efficient than with existing programming languages, such as C or C++. A second robotic application, which originally required more than 5,000 lines of code, was re-implemented on 54 lines. We believe that these results open a completely new way of programming computers much more efficiently than today's best practice.

Future Work: Currently, we are implementing a much cleaner version of our programming language on top of ML. We are also working on a new implementation of CES using a cross-compiler that compiles CES into C.

```
float     sonars[24];
probfloat turn, angle;
float     target_turn;
fa        mynet();

faconfigure(&mynet, NEURONET, 24, FLOAT, 1, PROBFLOAT, 10);
angle =  mynet(sonars) * M_PI;
turn  =  angle - (0.5 * M_PI);
turn  <- target_turn;
```

Table 1: CES code, taken from [1]. the data type `probfloat` represents a *probabilistic variable*, which contains a probability distributions over all floats. The function `mynet()` is a function approximator, configured as a neural network, which is trained using the `<-` operator. CES also offers a variety of new constructs that support common operations on probabilistic operations.

Selected References:

[1] S. Thrun. A framework for integrating programming and learning: Initial design and results. Technical Report CMU-CS-98-142, Carnegie Mellon University, Computer Science Department, Pittsburgh, PA, 1998.