Enhancing Lisp Instruction with RCXLisp and Robotics

Frank Klassner
Department of Computing Sciences
Villanova University
Villanova, PA 19085
610-519-5671
Frank.Klassner@villanova.edu

ABSTRACT

This paper presents the RCXLisp library, an extension to Common Lisp that allows students to investigate a broad range of artificial intelligence and computer science topics using the LEGO MindStorms® platform. The library has two features that distinguish it from other third-party packages and languages designed by academics and hobbyists for programming the MindStorms platform. The first is that it supports both remote control and on-board programming of MindStorms robots. The second is that it supports targeted communication between multiple LEGO robots and command-center desktops. The package is also the only one to be integrated into most popular Common Lisp programming environments. This paper also summarizes student experiences with the package over the years 2000-2003 in an Artificial Intelligence course.

Categories and Subject Descriptors

D.3.3 [**Programming Languages**]: Language Constructs and Features

General Terms

Languages

Keywords

Common Lisp, Robotics, Artificial Intelligence, Programming Languages, LEGO MindStorms

1. INTRODUCTION

In the Fall of 1997 the Introduction to Artificial Intelligence (AI) course at Villanova University was reorganized around the concept of agent-oriented design [7]. As described in previous published work [3], this approach was augmented in the Fall 1999 offering with LEGO MindStorms in order to give students the opportunity to explore the relationships among hardware, environment, and software organization in agent design. The course traditionally used Common Lisp in its programming projects, but at the time of MindStorms' adoption, there was no Common Lisp development environment for MindStorms. This situation led to a course design in which Not Quite C (NQC) [1] was used for robotics projects and Common Lisp was retained for all other projects. Course surveys indicated that the burden of working with (and often learning on the fly) two different languages distracted students from learning the AI concepts of the course.

Based on this observation and my experiences with programming MindStorms for AI projects, I decided to develop a Common Lisp library, RCXLisp, for programming the MindStorms' platform. This paper describes how the RCXLisp library augments not only augments the MindStorms platform as a tool for AI pedagogy but also enhances students' interest in the Lisp language itself. The second section of this paper describes the MindStorms platform's design insofar as it affected the design of RCXLisp. The paper's third section describes the RCXLisp library and support firmware. The fourth section presents observations on student experiences with the package over the four semesters (2000-2003) that it has been used in the AI course. The fifth section concludes with plans for future use and development for RCXLisp.

2. MINDSTORMS BACKGROUND

2.1 Basic System Design

The RCX is the programmable brick at the heart of the LEGO MindStorms kit. Figure 1 shows an RCX in a simple roving robot.

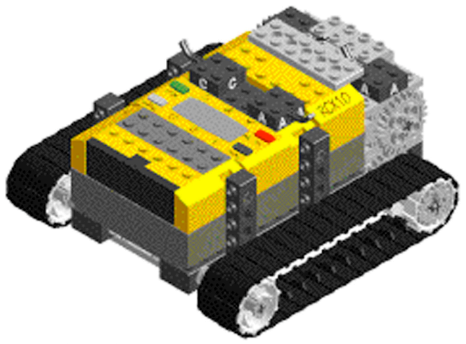

Figure 1. RCX unit. IR port is "in front," at lower left. Sensor ports are gray squares behind the IR port. Output ports are dark squares located beyond the LCD display. Figure courtesy David Baum.

The RCX has a 16MHz CPU (Hitachi H8/3292 microcontroller), 16KB RAM (and another 16K of ROM routines), and houses an infrared (IR) transmitter/receiver for sending and receiving data and commands from a desktop PC or from other RCXs. The IR transceiver has a range of 15-25 feet, depending on lighting conditions and reflectivity of walls. Version 1.0 of the platform used a serial-port IR Tower for broadcasting messages and software to the RCX from a desktop. Later versions (the current version in 2003 is 2.0) replaced the serial device with a USB

SIGCSE'04, March 3–7, 2004, Norfolk, Virginia, USA.

Tower whose IR receiving range is more limited: 8-12 feet. The RCX unit has three input ports (labeled 1,2 and 3), three output ports (labeled A, B, and C), and a 5-"digit" LED display. LEGO manufactures touch, light, temperature, and angle (rotation) sensors that can be connected to the RCX's input ports; several third-party vendors offer magnetic compass sensors, ultrasonic distance sensors, and infrared distance sensors. Motors, LEDs, and infrared emitters can be attached to the output ports.

The RCX's "operating system" is a replaceable firmware that models a primitive virtual machine. The RCX's firmware can be used in autonomous mode (the robot's behavior depends only on the program loaded into its memory) or in direct-control mode (a control program on a desktop computer broadcasts a series of instructions to the robot for on-board execution). When used in autonomous mode, the firmware supports 32 sixteen-bit global integer variables that can be shared by up to 10 threads. Each thread can allocate up to 16 private variables. There is no firmware support for dynamic memory allocation. Only integer arithmetic is supported by the standard firmware. There is no firmware support for call stacks. With regard to direct-control mode, it should be noted that the standard RCX firmware supports a broadcast protocol through its IR transceiver.

2.2 MindStorms AI Shortcomings

From the hardware perspective, I do not believe that the available set of sensors for MindStorms is any more limiting for AI or robotics work at the collegiate level than that available at considerable greater cost for other robotics platforms such as the HandyBoard or Khepera [4]. However, the RCX's 16KB of onboard RAM, does represent a serious obstacle for projects involving large-footprint real-time schedulers and planners that are often used in robotics in the study of Intelligent System design.

Another shortcoming of MindStorms is that the platform does not support point-to-point wireless protocols over its built-in IR port—all IR communication is broadcast. If one has three RCXs in the same vicinity, two of them cannot exchange messages without the third inadvertently receiving the messages. Furthermore, it is not possible to have a desktop application coordinate several robots of a team without a mechanism to address the robots' RCXs individually. The standard Lego firmware, however, is limited to using a broadcast protocol that does not support such targeted message-passing, and therefore so is software based on the Lego firmware, such as Dave Baum's NQC [1] and Wick et al.'s Lego/Scheme compiler [8].

In the category of software shortcomings, the standard firmware's lack of support for a call stack limits on-board programs' use of abstraction because nested function or procedure calls are not possible. The firmware does not support dynamic memory allocation that is the basis of modern languages such as Java, C++, and Lisp.

A final MindStorms issue occurs with respect to machine learning. Machine learning algorithms often require a state vector in which the elements must be measured more or less simultaneously. If one assumes a ML algorithm running on a desktop, it might build up a state vector by sequentially querying the RCX for registers, sensors-value buffers, timer registers, motor setting, etc. Examination of the RCX and its standard firmware show that there could be as many as 60 queries made in order to construct a state vector. Each IR message and response takes about 7ms, so it can take quite a while to build up the state vector, possibly violating the assumption of simultaneity.

3. RCXLISP

3.1 Extended Firmware

Both as a means of overcoming the problems cited in section 2.2 and as part of a larger project aimed at improving the MindStorms platform's usefulness in collegiate computer science curricula, I have developed the RCXLisp programming libraries. This package allows one to work with the RCX unit from the LEGO MindStorms® kit using Common Lisp.

The RCXLisp package is compatible with LEGO's firmware. It is, however, designed primarily to work with extended firmware my students and I designed that supports wireless networking and most of the bytecodes from version 1.0 of LEGO's firmware. This extended firmware is called "Mnet firmware," and supports directed IR communication by adding source and target fields to the basic LEGO protocol and by allowing each RCX to set a 1-byte ID value for itself. Mnet firmware supports a bytecode that allows an RCX to restrict what sources (i.e. what other RCXs) it will accept messages from.

The Mnet firmware also supports a bytecode for requesting all of the state information of an RCX, which is useful for monitoring and statistical analysis as well as for supplying state vectors for neural network training and for other machine learning applications.

Although the first version of this firmware does not support dynamic memory allocation, future versions are expected to include this capability (along with garbage collection) in order to support a larger subset of Common Lisp's functionality. I argue that this lack of functionality is mitigated by RCXLisp's ability (discussed later) to integrate desktop remote control with on-board autonomous programs. Problems requiring recursive or nested function calls can be solved on a desktop Lisp environment and the results can be communicated to the RCX's on-board control program.

3.2 Language Design

Moving from hardware-oriented issues, let us now discuss the design and organization of the RCXLisp language itself. Specifically, RCXLisp lets one

- remotely control the RCX from a Common Lisp program running on a desktop computer,
- write RCXLisp programs to run on the RCX,
- create and compile RCXLisp programs for downloading to RCXs "on the fly," from within Common Lisp desktop environments,
- simultaneously control more than one RCX from a single MindStorms infrared transceiver tower
- set up a network of RCX units that can communicate with each other in a targeted manner (as opposed to the "broadcast manner" supported by LEGO's kit).

The RCX libraries support both the older serial-port infrared transceivers and the newer USB-port towers that LEGO is currently shipping. It is also possible to use the libraries to control more than one tower, opening up the possibility of extending the remote-control radius of a desktop through strategic placement of multiple IR towers in one room.

RCXLisp is actually two related languages. The first is "Remote RCXLisp," which is a collection of macros, variables, and functions for remotely controlling RCX units from a desktop. The second language is "RCXLisp" proper, which is a subset of Common Lisp that can be cross-compiled to run on RCX firmware for autonomous control of the unit. The next two subsections provide details on the two parts of the language library, with the goals of demonstrating both the tight integration among Lisp environment, RCX, and desktop control processes the library makes possible, and the Common Lisp language features that the library encourages students to learn about as they set up robotics-inspired projects.

3.2.1 *Remote RCXLisp*

Since "Remote RCXLisp" is intended to run within a desktop Common Lisp environment, the design goal of this language was to adhere as closely as possible to the Common Lisp standard in Guy Steele's text, "Common Lisp: The Language," 2nd edition (aka "CLTL2").

The "Remote RCXLisp" library provides users with the **with-open-com-port** and **with-open-rcx-stream** macros to set up communication environments for contacting the RCX units. These macros are modeled closely on Common Lisp's "with-open-stream" macro. **With-open-com-port** is used to specify the communication port (serial or USB) over which an RCX is to be contacted, and **with-open-rcx-stream** is used to define the RCX-unit-specific data stream that will use a port stream. The code in Figure 2 shows how the macros are used and provides an example of the functions that can be invoked to control an RCX from within the macros' communication environment. There are 45 functions defined for RCX control. Built-in Common Lisp functions are capitalized in the figure for clarity. Lowercased function names should be assumed to be part of the RCXLisp language.

The body of the "full-speed-ahead" function contains the following examples of "Remote RCXLisp" functions for controlling an RCX: **set-effector-state** (for initializing and controlling motors), **set-sensor-state** (for initializing how an input sensor port will be used to gather data), and **sensor** (for accessing the current value of a sensor port). The testing function uses **alivep** to determine if the RCX is in range and responding. It will be useful to note for later discussion that the language includes a **var** function which allows a desktop program to query an RCX asynchronously for the value stored in a given variable register.

All "Remote RCXLisp" functions take an optional final stream argument ("r" in full-speed-ahead, and "rcx10" in testing). If one wishes to forego the repetitious typing of the stream argument, one can use the **using-rcx** macro as in figure 2 to define a default stream for enclosed RCXLisp functions. This macro is closely modeled on Common Lisp's "using-slots" macro for object-oriented programming. It also emphasizes for programmers the conceptual role of the desktop as a director telling some particular RCX what to do next.

```
(DEFUN full-speed-ahead  (r s dir)
"This will make the rcx in R go at speed S in direction DIR until touch sensor
on its '2' port returns 1."
(LET ((result 0))
     (set-effector-state '(:A :B :C) :power :off r)
            ;in case things are in an inconsistent state,
            ;turn everything off first
      (set-effector-state '(:A :C) :speed s  r)
      (set-effector-state '(:A :C) :direction dir  r)
               ; dir is eq  to  :forward, :backward, or :toggle
              ; no motion will occur until the
              ; next call to set-effector-state
      (set-sensor-state 2 :type :touch :mode :boolean  r)
      (set-effector-state '(:A :C) :power :on r)
      (LOOP ;this loop will repeat forever until sensor 2 returns a 1
           (SETF result (sensor 2 r))
           (WHEN (AND (NUMBERP result)
                          ;needed to keep = from causing error if
                          ;sensor function returns nil.
                         (= result 1))
                 (RETURN)))
      (set-effector-state '(:A :C) :power :float r))))


(DEFUN testing ()
(with-open-com-port (port :LEGO-USB-TOWER)
  (with-open-rcx-stream (rcx10 port :timeout-interval 80 :rcx-unit 10)
     ; increase/decrease  serial timeout value of 80 ms depending on
    ;environmental factors like ambient  light.
    (WHEN (alivep rcx10)
        (full-speed-ahead rcx10  5 :forward)))))
```

Figure 2. Sample "Remote RCXLisp" code

```
(DEFUN full-speed-ahead  (r s dir)
"This will make the rcx in R go at speed S in direction DIR until touch sensor
on its '2' port returns 1."
(LET ((result 0))
    (using-rcx  r
     (set-effector-state '(:A :B :C) :power :off)
            ;in case things are in an inconsistent state,
            ;turn everything off first
      (set-effector-state '(:A :C) :speed s )
      (set-effector-state '(:A :C) :direction dir )
               ; dir is eq  to  :forward, :backward, or :toggle
              ; no motion will occur until the
              ; next call to set-effector-state
      (set-sensor-state 2 :type :touch :mode :boolean)
      (set-effector-state '(:A :C) :power :on )
      (LOOP ;this loop will repeat forever until sensor 2 returns a 1
           (SETF result (sensor 2 ))
           (WHEN (AND (NUMBERP result)
                          ;needed to keep = from causing error if
                          ;sensor function returns nil.
                         (= result 1))
                 (RETURN)))
      (set-effector-state '(:A :C) :power :float ))))
```

Figure 3. Using 'using-rcx to clean up "Remote RCXLisp" code.

Programs that are intended to be executed on an RCX are first compiled within the RCXLisp desktop environment and then downloaded through the IR Tower to the RCX. Firmware is also loaded from the desktop Lisp environment. These two actions are accomplished with the **download-firmware** and **rcx-compile-**

and-download functions. It is important to note that these functions *are native to the Lisp environment*. That is, no non-Lisp mechanism is needed for these actions. Both LeJOS [6] (the MindStorms Java virtual machine developed originally by Jose Solarzano) and Wick et al.'s Lego/Scheme compiler [8] require a separate command-line system program to download compiled code or firmware into the RCX. The recursive-descent parser+compiler in **rcx-compile-and-download** is implemented in Lisp.

3.2.2 "RCXLisp" Proper

RCXLisp is the subset of Common Lisp (with a few non-standard additions) that can be compiled and downloaded to run on an RCX unit autonomously. As with "Remote RCXLisp," the design goal was to follow CLTL2's standard as closely as possible, and to maintain compatibility with Lego firmware as well as Mnet extended firmware. However, because even the extended firmware does not yet support indirect addressing or call stacks, some of the "functional" nature of Common Lisp is still missing. Some RCXLisp functions cannot accept variable values for some arguments; they can only accept constants.

For consistency with "Remote RCXLisp," and to make it more straightforward to transfer desktop Lisp code to the RCX, RCXLisp implements all of the RCX-control functions in "Remote RCXLisp." In RCXLisp however, rcx-control functions like **set-sensor-state** do not have an optional final stream argument since it is assumed that the Lisp code will only be executed on the RCX unit itself. If an RCX needs to control the behavior of another RCX unit, it does not download programs into the other RCX. Rather, it sends integer-valued messages that the other RCX must interpret to determine what action to take.

RCXLisp supports analogs to the following subset of Common Lisp control expressions, along with their standard semantics defined in CLTL2: DOTIMES, COND, IF, LOOP, PROGN, RETURN, and WHEN. The RCXLisp language supports 16-bit signed integer arithmetic with the following operators: +, -, *, and / (integer division). RCXLisp provides the >, >=, <, <=, =, /=, and EQUAL Common Lisp comparison operators in their full functionality. Just as Common Lisp allows one to use comparison invocations such as "(< 2 x 6)" to test for when the value of x is between 2 and 6, so too does RCXLisp. It also provides a limited version of the Common Lisp RANDOM function.

RCXLisp also supports the **and**, **or,** and **not** Boolean operators, along with their CLTL2 semantics (including the "Boolean short-circuit"). RCXLisp does not support floating point arithmetic, but it does support the Boolean data type (i.e. T and NIL) and certain keyword constants (although new keywords <u>cannot</u> be defined).

Constants can be declared in RCXLisp programs with **defconstant**, which follows the semantics of the Common Lisp DEFCONSTANT form, and global variables can be declared with **defvar**, whose semantics are only partially the same as those of the Common Lisp DEFVAR form. Values (signed 16-bit integers and T and NIL) can be stored into variables using **setq**, which is similar to the Common Lisp SETQ form. Note that currently there is <u>no</u> analog in RCXLisp to the Common Lisp SETF macro.

Since general function calls are not supported, RCXLisp does not have an analog to the Common Lisp DEFUN form. In an effort to support *some* kind of code abstraction, the language design borrows inspiration from NQC's emphasis on macros for code abstraction and includes a **defmacro** form that follows the complete semantics of Common Lisp's DEFMACRO form. RCXLisp also borrows from Rodney Brooks' much earlier "L" language [2] a desire for simplicity (many advanced Common Lisp functions are not available in that language) and memory-conservation that is necessary for squeezing as much programming as possible into the small memories available on most robot platforms even today.

The language also provides two special-purpose forms that are neither in the "Remote RCXLisp" language nor the Common Lisp language. The first form is **defregister**, which is used to bind symbolic variable names to particular RCX variable registers. **Defregister** allows a programmer to tie a symbolic variable name to a given register so that a "Remote RCXLisp" program on a desktop using **var** to query a register can be guaranteed to access the intended variable value.

The second non-standard form is **defthread**, which is used to define RCX threads to run on an RCX unit. Calling this form "non-standard," however, is less of an indictment of RCXLisp than of the Common Lisp spec itself since as of 2003 no progress has been made in formalizing threading in the language!

```
(defconstant *receiver* 1)
(defregister 4 *LIMIT* 16)


(defthread (signaller) ()
  (loop
    (send-message 78)
    (sleep 15) ;; this is to leave the IR port silent for a
            ;; short time in case a desktop is sending a message.
    ))

(defthread  (alpha :primary t)   ( )
  (let  ((diff1 0)
         (diff2 0))
    (set-sensor-state *receiver* :type :light :mode :raw)
    (setq diff1 (abs (- (sensor *receiver* :raw)
                        (sensor *receiver* :raw))))
    (setq diff2 (abs (- (sensor *receiver* :raw)
                        (sensor *receiver* :raw))))
    (start-rcx-thread signaller)
    (loop
      (when (>= (abs (- diff1 diff2)) *LIMIT*)
          (play-tone 500 1))
      (setq diff1 diff2)
      (setq diff2 (abs (- (sensor *receiver* :raw)
                          (sensor *receiver* :raw)))))))
```

Figure 4. Multi-threaded RCXLisp Sample Code

Figure 3 shows a sample RCXLisp program that illustrates many of the features described above. The program will beep whenever the RCX is carried too close to a reflective object. This code makes the IR port on an RCX work together with a light sensor on sensor port 1 to implement a simple proximity sensor. The "signaler" thread repeatedly sends out an arbitrary integer message through the RCX's IR port. When the front of the RCX gets too close to a tall obstacle, the IR signal from the IR port will reflect back, and the light sensor will pick this echo up. As the reflections increase in intensity, the light sensor's value will jump more wildly. The value of *LIMIT* may need to be experimented with. It is declared as a register variable because this allows it to be modified dynamically by a "Remote RCXLisp" program on a desktop, by using **var** to access and set register 4.

3.3 Library Installation

The RCXLisp libraries are supported on the Allegro (Franz), MCL (Digitool), and Xanalys Common Lisp environments, on both Windows (98, 2000, and XP) and Mac OS X. RCXLisp is straightforward to install. The first step is to copy a port-communication library file (.dll for Windows, .dynlib for Mac) into the same directory as the Lisp environment program. The second step is to make sure that the LEGO USB driver is installed if one plans to use the LEGO USB Tower. The final step is to load the Lisp files that define the RCXLisp environment.

4. STUDENT EXPERIENCE

Both "Remote RCXlisp" and "RCXLisp" make extensive use of keyword and optional arguments, as well as streams and macros – concepts not often explored in depth in Common Lisp in courses like Programming Languages (PL) and almost never in Lisp-based AI courses simply because of a lack of motivating material. Students who I have taught in both courses (the PL course is generally taken as a requirement before the AI elective) have commented on how the RCXLisp environment's use of extended function argument capabilities helped them understand the benefits (rapid prototyping) and pitfalls (confusion over argument ordering) of these features better than when they briefly encountered them in the PL course.

Both my students and I noticed the reduced overhead in learning how to program the RCXs. Since RCXLisp is just an extension of the Common Lisp they were already using the AI and PL courses, they could spend more time on the application problem rather than on learning yet another language like robotics-oriented language like NQC.

The library's support for "on-the-fly" generation and download of programs also helped students appreciate the power of Common Lisp's lack of differentiation between code and data. Since Common Lisp function declarations are themselves merely formatted linked lists, students could generate plans as linked lists within a planner and then download the same data to the RCX as an immediately executable form. I hasten to add that I have emphasized to PL and AI students that there is nothing "magical" about this ability of Lisp. As long as a language has a built-in interpreter like EVAL and a built-in compiler like **rcx-compile-and-download**, then it too can do what Lisp lets them do.

Students have also commented on the immediacy of working with RCXs via the Common Lisp Listener: simply by invoking a function in the Listener, an RCX can be made to respond. This has helped them understand the differences and similarities between compiled and interpreted code, as well as given them a more concrete example of the concept of side-effect.

The library's integration with low-level system functions such as infrared USB communication helps students get past the uninteresting details of port communication and instead concentrate on the AI-oriented problems of environmental noise, sensor sensitivity, and environmental nondeterminism.

5. CONCLUSIONS AND FUTURE WORK

RCXLisp is the first open source Lisp approach for programming physical robots that supports both remote control and on-board programming of robots as well as targeted communication between multiple robots and command-center desktops. Coupled with the low cost and adaptability of MindStorms, RCXLisp should make it easier for cost-conscious schools to add robotics-inspired projects to AI and PL courses, without having to turn computer science students into mechanical or computer engineers.

The library has separate API functions for accessing serial ports and USB ports, making it a useful basis for designing Lisp solutions for interfacing with other serial and USB devices.

Improvements and optimizations still remain. One goal is to add call stack support and memory management support to the Mnet firmware. This would extend the Lisp functionality of the on-board language for RCXs (e.g add DEFUN and list-manipulation functions). Lest the lack of garbage collection seem too limiting for RCXLisp right now, it should be noted that leJOS also does not currently implement a garbage collector in an effort to keep the JVM footprint as small as possible. A related possibility would be to eliminate firmware and target the H8 processor directly, as LegOS does [5].

Another goal is to integrate the LEGO Vision Command Camera into the RCXLisp library. This would give students a low-cost yet powerful tool for exploring machine vision problems at the undergraduate level.

6. ACKNOWLEDGMENTS

"LEGO MindStorms" and "RCX" are trademarks of the LEGO Group, which does not sponsor, authorize, or endorse any of the third-party work cited in this article. The author of this article has no financial relationship with the LEGO Group.

I thank Andrew Chang for his graduate independent study work that led to extending the MindStorms' firmware, Drew Jacobs for his ongoing work in porting this framework to Java and C++ platforms, and my AI students for their patience and enthusiasm in vetting the RCXLisp libraries.

This material is based upon work supported by the National Science Foundation under Grant No. 0088884 and Grant No. 0306096. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

7. REFERENCES

[1] Baum, D. Not Quite C (NQC), Sept. 2003, http://www.baumfamily.com/nqc/

[2] Brooks, R. A. (1993), L: A Subset of Common Lisp, Technical report, Massachusetts Institute of TechnologyArtificial Intelligence Lab.

[3] Klassner, F., *A Case Study of LEGO Mindstorms'^{TM} Suitability for Artificial Intelligence and Robotics Courses at the College Level,* in *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education* (2002).

[4] Martin, F., The MIT HandyBoard Project, September 2003. http://lcs.www.media.mit.edu/groups/el/Projects/handy-board

[5] Noga, M. LegOS, September 2003, http://www.noga.de/legOS

[6] Solorzano, J. LejOS, Sep 2003, http://lejos.sourceforge.com

[7] Russell, S. and Norvig, P. *Artificial Intelligence: A Modern Approach.* 2nd edition, Prentice Hall, 2003.

[8] Wick, A., Klipsch, K., and Wagner, M. LEGO/Scheme compiler, http://www.cs.indiana.edu/~mtwagner/legoscheme