Esterel to Lego E-Code Generator

Rahul Shah and Xuanming Dong

EE 290O Final Project

May 08, 2002

Outline

- Introduction
- Major steps for cross-compiler
- E-Machine platform
- Demo
- Conclusion

Project Description

- Cross compile programs written in Esterel to run on the Lego RCX module
- The Lego code uses the e-code paradigm used in the class
- Synthesized code should take into account parallelisms in Esterel and sequentialize them

Major steps for cross-compiler

- Parse the Esterel code, checking for valid tokens
- Write code in form of a concurrent control flow graph (CCFG)
- Synthesize a sequential control flow graph (SCFG)
- Synthesize E-Code from the SCFG

Step 1: Parse (using perl)

- Produce two output files:
  - Stripped version of the Esterel program, containing just the program code
  - A file of all the helper functions to be used with the “CALL” command of e-code
- Check all input and output signals for validity
- Limited subset of Esterel commands allowed

Step 2: Create a CCFG (using C)

- Shows the entire code in parallel form
- Identify dependencies between different forks in the program
- The CCFG is an intermediate representation of the program
- A 2-D array specifies the dependencies between commands

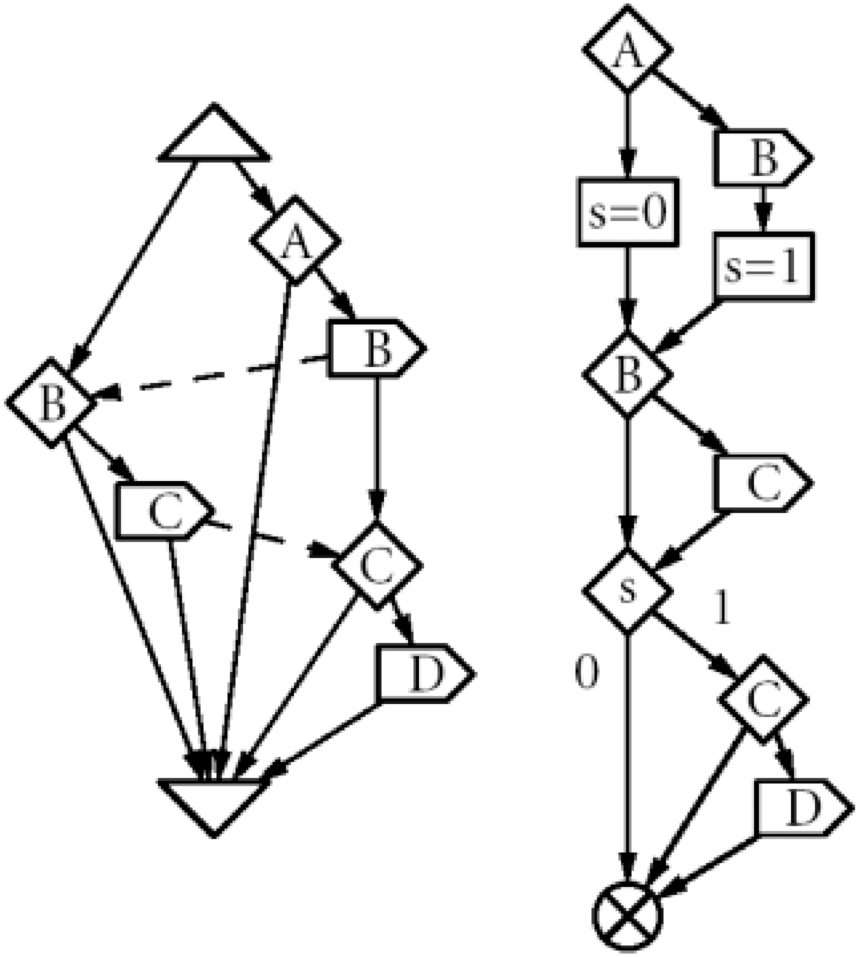
Array of dependencies

```
1 |  5 13 | 0 1 0 0 0 0 0 0
2 |  5 14 | 0 0 1 0 0 0 0 0
3 | 17  1 | 0 0 0 1 0 0 0 0
4 |  5 13 | 0 0 0 0 1 0 0 0
5 |  5 14 | 0 0 0 0 0 1 0 0
6 | 17  7 | 0 0 0 0 0 0 1 0
7 | 20  1 | 0 0 0 0 0 0 0 1
8 | 14  0 | 0 0 0 0 0 0 0 0
```

Step 3: Create a SCFG (using C)

- For sequential flow, no problem!
- For parallel forks, execute commands that do not depend on actions in the other fork
- When switching to another fork, it is necessary to save the state in the current fork

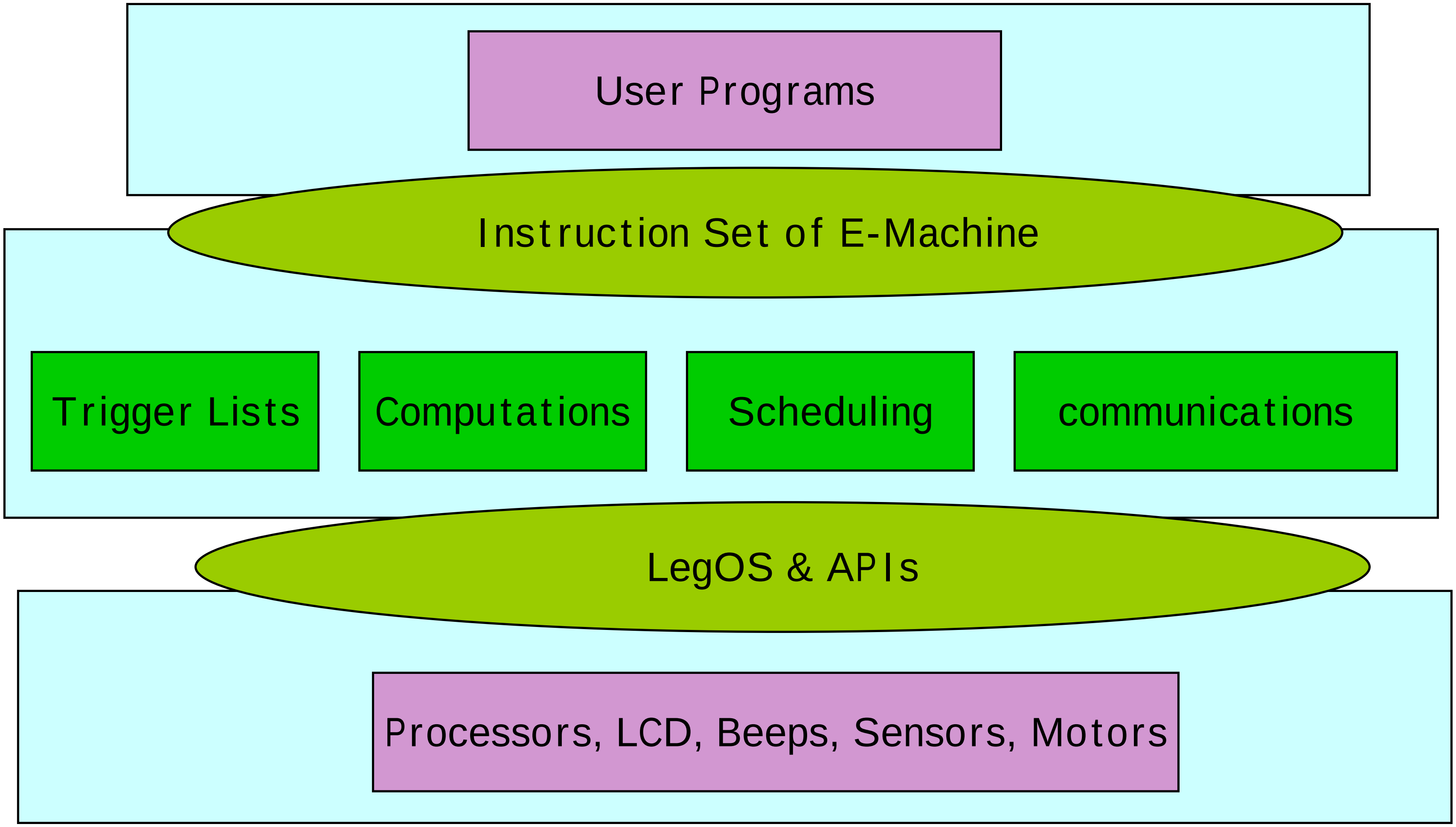
CCFG and SCFG



Step 4: Synthesize E-Code (using C)

- Using the sequential code of Step 3, translate Esterel commands into E-Code commands
- Need to do a double-pass to resolve cross-references between lines
- Write the output file with the e-code ready to compile using the LegOS compiler

System Architecture



Instruction Set of E-Machine

- NOP
- JMP
- CALL
- SCHEDULE
- FUTURE
- RETURN
- SLEEP

Triggers

- Classification
  - Sensor Triggers
    - `NULL_INPUT`
    - `TOUCH_INPUT_1, TOUCH_INPUT_2, TOUCH_INPUT_3,`
    - `LIGHT_LOW_1, LIGHT_HIGH_1, LIGHT_1_VALUE,`
`LIGHT_LOW_2, LIGHT_HIGH_2, LIGHT_2_VALUE,`
`LIGHT_LOW_3, LIGHT_HIGH_3, LIGHT_3_VALUE`
  - Timer Triggers
- APIs
  - Create_Trigger(signal, next, time)
  - Insert_Trigger(trigger)
  - Remove_Trigger(trigger)
  - Cancel_Trigger(trigger)
  - Evaluate_Trigger(trigger, predicate)

Project Demo

- The robot performs a simple task:
  - Move forward till it hits a wall or senses a dark surface below it
  - Reverse direction and continue till it again hits a wall or senses a dark surface
  - Continue this indefinitely….

The Esterel code

```
module project:
   input LIGHT_LOW_2;
   input TOUCH_INPUT_1;
   output MOTOR_A_SPEED(integer), MOTOR_A_DIR(integer);

   constant MOTOR_FWD, MOTOR_REV : integer;
   constant TICKS_PER_SECOND = 1000 : integer;

loop
   emit MOTOR_A_DIR(MOTOR_FWD);
   emit MOTOR_A_SPEED(100);
   await TOUCH_INPUT_1;
   emit MOTOR_A_DIR(MOTOR_REV);
   emit MOTOR_A_SPEED(100);
   await LIGHT_LOW_2;
end loop
```

Synthesized C code (I)

```
#include <stdlib.h>
#include <sys/em.h>
#include <conio.h>
#include <unistd.h>
#include <time.h>
#include <dsound.h>
#include <dmotor.h>
#include <dsensor.h>

void function1() {
   motor_a_dir(1);
}
```

```
void function2() {
   motor_a_speed(100);
}

void function3() {
   motor_a_dir(2);
}

void function4() {
   motor_a_speed(100);
}
```

Synthesized C code (II)

```c
int main() {
    inst_t eco[10];
    emachine_t *em;
    ds_active(&SENSOR_1);
    ds_active(&SENSOR_2);
    ds_active(&SENSOR_3);
    eco[0].opcode = CALL;
    eco[0].arg1 = (int) &function1;
    eco[1].opcode = CALL;
    eco[1].arg1 = (int) &function2;
    eco[2].opcode = FUTURE;
    eco[2].arg1 = (int) TOUCH_INPUT_1;
    eco[2].arg2 = (int) 4;
    eco[3].opcode = RETURN;
    eco[4].opcode = CALL;
    eco[4].arg1 = (int) &function3;
    eco[5].opcode = CALL;
    eco[5].arg1 = (int) &function4;
    eco[6].opcode = FUTURE;
    eco[6].arg1 = (int) LIGHT_LOW_2;
    eco[6].arg2 = (int) 8;
    eco[7].opcode = RETURN;
    eco[8].opcode = JMP;
    eco[8].arg1 = (int) 0;
    eco[9].opcode = NOP;
    em = (emachine_t *) malloc
    (sizeof (emachine_t));
    em->eco = eco;
    em->eco_size = 10;
    Emachine(em);
    return 0;
}
```

The cross-compiler...