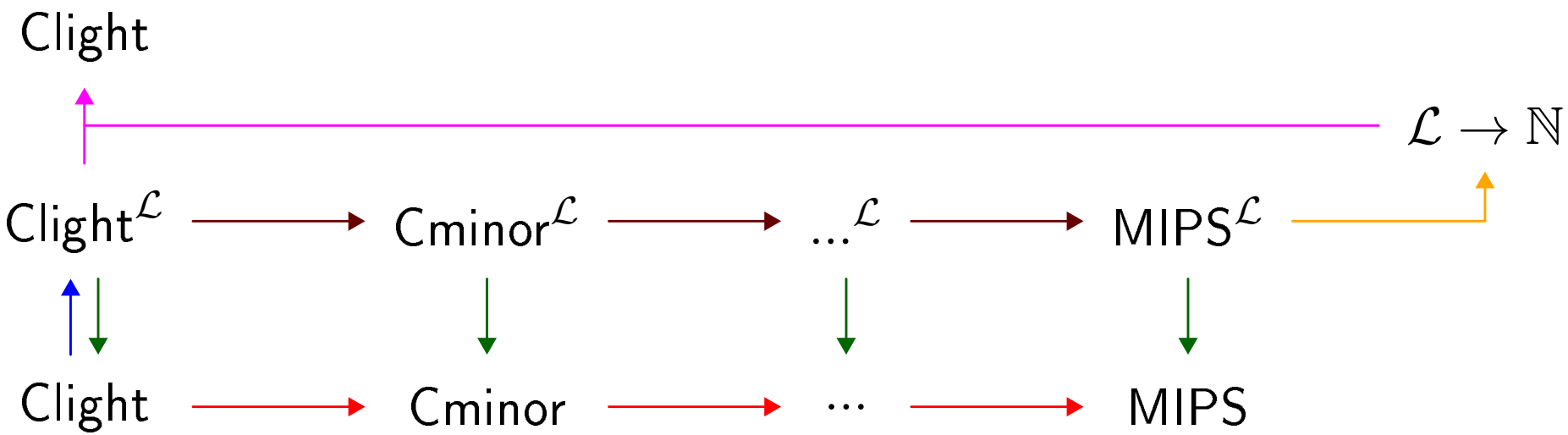
CerCo Work Package 2 : The Untrusted Compiler Prototype

(part 2)

Yann Régis-Gianas

March 11 2011

Architecture of the Compiler



Labelling

Erasure

Compilation

Labelled compilation

Cost deduction

Instrumentation

Porting from MIPS to the 8051 microprocessor

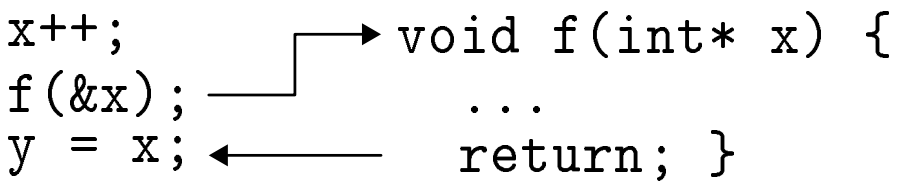
Targetting the 8051 microprocessor raised the following issues:

- How to represent 32bits values in an 8bits architecture?
- How to deal with heterogeneous representation of pointers and integers? (Words are 8bits long whereas memory addresses are stored using 16bits.)
- How to select instruction for this microprocessor?
- What calling convention to use?

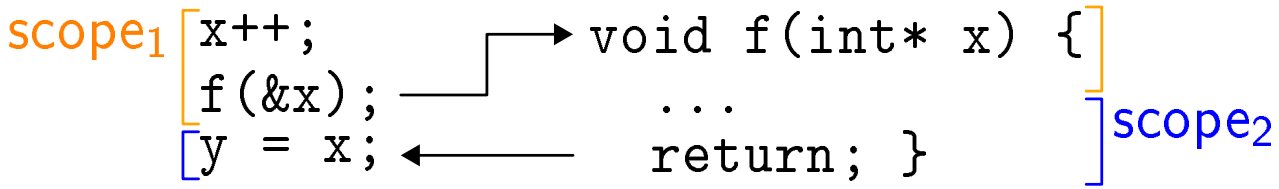
Where was the difficulty in the prototype implementation?

The main issue in scaling our approach from the toy compiler to the C compiler was **function calls** because they add an extra complexity in the labelling process.

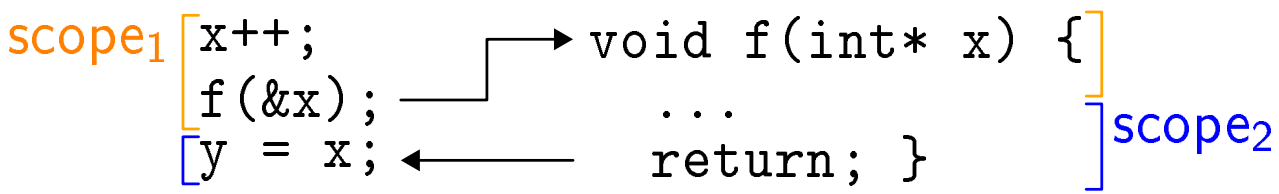
How to cover the control flow with cost labels?



How to cover the control flow with cost labels?



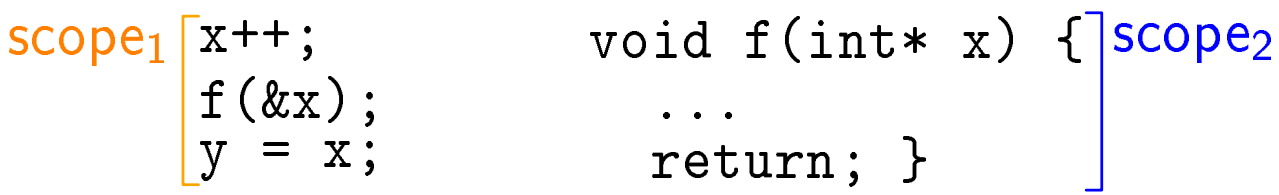
How to cover the control flow with cost labels?



Function pointer: statically unresolvable destination

Each function should handle its cost. [a]

[a] Notice that the proof of the compiler will provide the invariant that function pointers always only contain valid addresses to code generated using the same compiler.



A glimpse on the compiler passes

```
char search (char tab[], char size, char to_find) {
  char low = 0, high = size-1, i;

  while (high ≥ low) {
    i = (high+low) / 2;
    if (tab[i] == to_find) return i;
    if (tab[i] > to_find) high = i-1;
    if (tab[i] < to_find) low = i+1;
  }

  return (-1);
}
```

A glimpse on the compiler passes : Labelling in Clight

```
unsigned char search(unsigned char *tab, unsigned char size, unsigned char to_find)
{
  unsigned char low, high, i;
   _cost8:
  low = (unsigned char)0;
  high = (unsigned char)((int)size - 1);
  while ((int)high ≥ (int)low) {
     _cost6:
    i = (unsigned char)(((int)high + (int)low) / 2);
    if ((int)tab[i] == (int)to_find) {
         _cost4:  return i;
    } else { _cost5: }
    if ((int)tab[i] > (int)to_find) {
       _cost2:  high = (unsigned char)((int)i - 1);
    } else { _cost3:  }
    if ((int)tab[i] < (int)to_find) {
       _cost0:  low = (unsigned char)((int)i + 1);
    } else { _cost1:  }
  }
   _cost7:
  return (unsigned char)(-1);
}
```

A glimpse on the compiler passes : RTL_{abs}

```
"search"([%9 ; %8], [%2], [%3]):   ptr → int → int → int
    locals:   ...
    result:   [%10]
    stacksize:   0
    entry:   search40
    exit:   search0

    search9:   lt [%13], [%3] -→ search8, search5
    search8:   emit _cost0 -→ search7
    search7:   imm [%12], imm_int 1 -→ search6
    search6:   add [%5], [%7], [%12] -→ search4
    search5:   emit _cost1 -→ search4
    search40:   emit _cost8 -→ search39
    search4:   -→ search36
    // ...
```

A glimpse on the compiler passes : 8051

```
// ...
317:   nop ;; 1 _cost4
318:   mov 002h, #000h ;; 3
321:   mov A, 002h ;; 1
323:   mov 005h, A ;; 1
325:   mov A, 009h ;; 1
327:   mov 004h, A ;; 1
329:   mov A, 000h ;; 1
331:   push 0E0h ;; 2
333:   mov A, 001h ;; 1
335:   push 0E0h ;; 2
337:   mov 0E0h, #004h ;; 3
340:   add A, 006h ;; 1
342:   mov 006h, A ;; 1
344:   mov 0E0h, #000h ;; 3
347:   addc A, 007h ;; 1
349:   mov 007h, A ;; 1
351:   mov A, 005h ;; 1
353:   mov 083h, A ;; 1
355:   mov A, 004h ;; 1
357:   mov 082h, A ;; 1
359:   ret ;; 2
```

Benchmarks

	gcc -O0	acc	gcc -O1
badsort	55.93	34.51	12.96
fib	76.24	34.28	45.68
mat_det	163.42	156.20	54.76
min	12.21	16.25	3.95
quicksort	27.46	17.95	9.41
search	463.19	623.79	155.38

Future work

- Prototype maintenance, validation and testing.
- Integration of the 8051 specification (recently provided as deliverable 4.1).
- Integration of the non-standard extensions of the C language consisting of directives that specifies storage location (given that their semantics have been addressed in deliverable 3.1).
- Integration of a preprocessor to encode 16 bits and 32 bits integers into records of 8 bits integers.
- Improvement of instruction selection (but we will not sacrifice conceptual simplicity to keep mechanized proofs manageable).
- Development of a `Frama-C` plugin that will embed the compiler as well as an algorithm to produce synthetic information on the execution of C functions from the current cost annotations (which only give information about constant time portions of code).