Certifying cost annotations in compilers

Nicolas Ayache

Post-doctorate at PPS — Paris 7

18th november 2010

The CerCo project

3 years European project (FP7)

Laboratories

- University of Bologna — Claudio Sacerdoti Cohen
- University of Edinburgh — Randy Pollack
- **University of Paris Diderot (PPS)**

  Roberto Amadio (site leader)
  Nicolas Ayache (post-doc)
  Yann Régis-Gianas
  Ronan Saillard (Ph.D. student)

State of the art: Worst Case Execution Time

AbsInt (Abstract Interpretation)

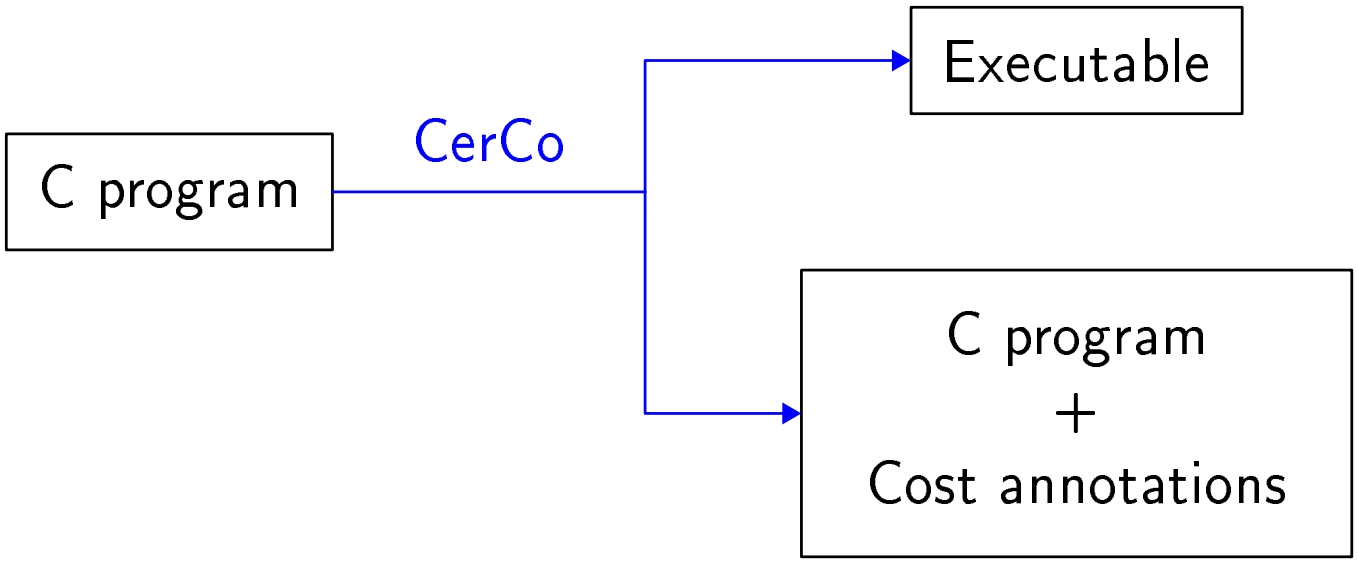
- ✓ Concrete time
- X Binary analysis
- X User interaction (loop iteration)
- X Not formally proven

Linear Logic

- ✓ Formal framework
- X Complexity classes
- X Verbose

Goal

Formally sound, cost annotating compiler



√ Overapproximated concrete complexity

CerCo's approach: problematic

What is the cost of evaluating `tab[i]+1` ?

Depends on:

- The variable final locations
- The way memory accesses are compiled
- The way operations are compiled

$\Rightarrow$ Depends on the compilation process

CerCo's approach: solution

Bad solution

Consider worst case scenarios

- X Too imprecise
- X Optimizations lost
- X Not modular

CerCo's approach: solution

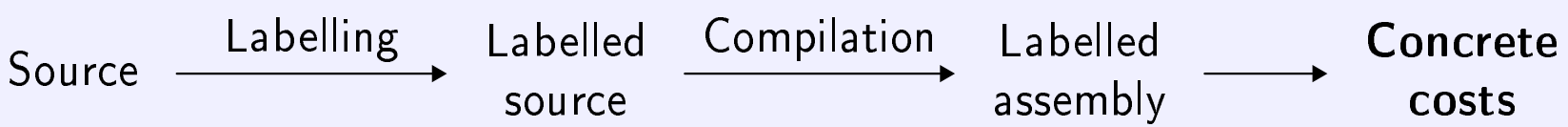
Bad solution

Consider worst case scenarios

- X Too imprecise
- X Optimizations lost
- X Not modular

CerCo's solution

Symbolic cost update: **label**



CerCo's approach: common considerations

Operational semantics

- *Without labels*: $\mathcal{P}(\text{Prog} \times \text{State} \times \text{State})$
- *With labels*: $\mathcal{P}(\text{Prog} \times \text{State} \times \text{label trace} \times \text{State})$

CerCo's approach: common considerations

Operational semantics

- *Without labels*: $\mathcal{P}(\text{Prog} \times \text{State} \times \text{State})$
- *With labels*: $\mathcal{P}(\text{Prog} \times \text{State} \times \text{label trace} \times \text{State})$

Annotation semantics

$\surd$ Annotation = Instruction of the source language

- Explicits cost annotations
- Analysis tools for cost synthesis

CerCo's approach: common considerations

Operational semantics

- *Without labels*: $\mathcal{P}(\text{Prog} \times \text{State} \times \text{State})$
- *With labels*: $\mathcal{P}(\text{Prog} \times \text{State} \times \text{label trace} \times \text{State})$

Annotation semantics

$\surd$ Annotation = Instruction of the source language

- Explicits cost annotations
- Analysis tools for cost synthesis

Demo

Outline

1. Toy compiler
   - Languages
   - Compilation
   - Labelling
   - Annotation
2. Realistic C compiler
   - Architecture
   - Labelling
   - Experiments
3. Future work

Outline

1. Toy compiler
   - Languages
   - Compilation
   - Labelling
   - Annotation
2. Realistic C compiler
   - Architecture
   - Labelling
   - Experiments
3. Future work

Overview

$$\textbf{Imp} \longrightarrow \textbf{VM} \longrightarrow \textbf{ASM}$$

- **Imp**: simple while language
- **VM**: virtual machine (stack operations)
- **ASM**: assembly

Overview

$$\textbf{Imp} \longrightarrow \textbf{VM} \longrightarrow \textbf{ASM}$$

- **Imp**: simple while language
- **VM**: virtual machine (stack operations)
- **ASM**: assembly

In the following...

- How to label the languages?
- How to adapt the proofs?
- How to keep the proofs modular?

Syntax

Imp

$e ::= id \mid n \in \mathbb{N} \mid e + e \qquad b ::= e < e$
$S ::= \text{skip} \mid id := e \mid S\ ;\ S \mid \text{if } b \text{ then } S \text{ else } S \mid \text{while } b \text{ do } S$
$P ::= \text{prog } S$

VM

$\mathcal{I} ::= \text{cnst}(n) \mid \text{var}(id) \mid \text{setvar}(id) \mid \text{add} \mid \text{branch}(k) \mid \text{bge}(k) \mid \text{halt}$
$P ::= \mathcal{I}\ \text{list}$

ASM

$\mathcal{I} ::= \text{loadi } R,n \mid \text{load } R,addr \mid \text{store } R,addr \mid \text{add } R,R,R$
$\mid \text{branch } k \mid \text{bge } R,R,k \mid \text{halt}$
$P ::= \mathcal{I}\ \text{list}$

Labelled syntax

$\text{Imp}^{\mathcal{L}}$

$e ::= id \mid n \in \mathbb{N} \mid e + e \qquad b ::= e < e$
$S ::= \text{skip} \mid id := e \mid S \,;\, S \mid \text{if } b \text{ then } S \text{ else } S \mid \text{while } b \text{ do } S \mid \ell : S$
$P ::= \text{prog } S$

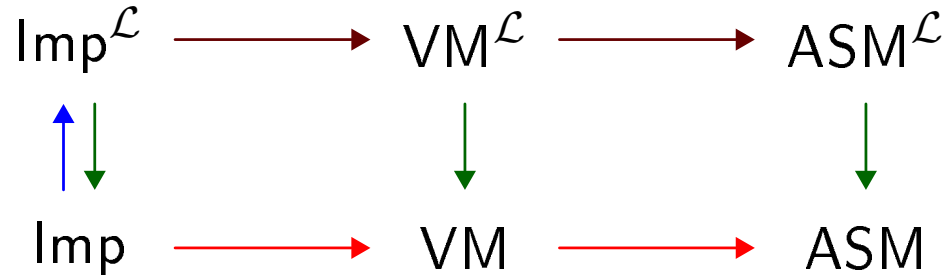
$\text{VM}^{\mathcal{L}}$

$\mathcal{I} ::= \text{cnst}(n) \mid \text{var}(id) \mid \text{setvar}(id) \mid \text{add} \mid \text{branch}(k) \mid \text{bge}(k) \mid \text{halt} \mid \text{emit}(\ell)$
$P ::= \mathcal{I}\ \text{list}$

$\text{ASM}^{\mathcal{L}}$

$\mathcal{I} ::= \text{loadi } R,n \mid \text{load } R,addr \mid \text{store } R,addr \mid \text{add } R,R,R$
$\quad \mid \text{branch } k \mid \text{bge } R,R,k \mid \text{halt} \mid \text{emit } \ell$
$P ::= \mathcal{I}\ \text{list}$

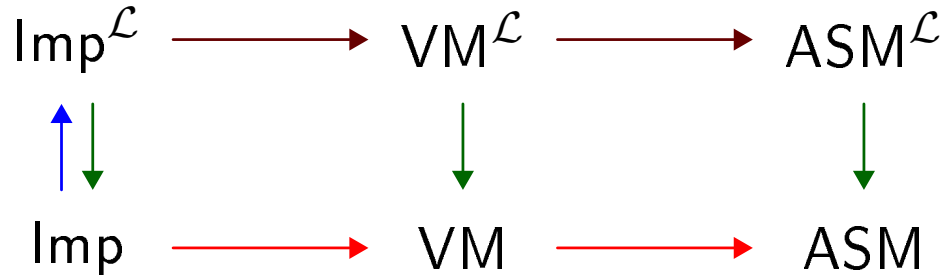
Simulation



Labelling Erasure

Compilation Labelled compilation

Simulation



Labelling — Erasure

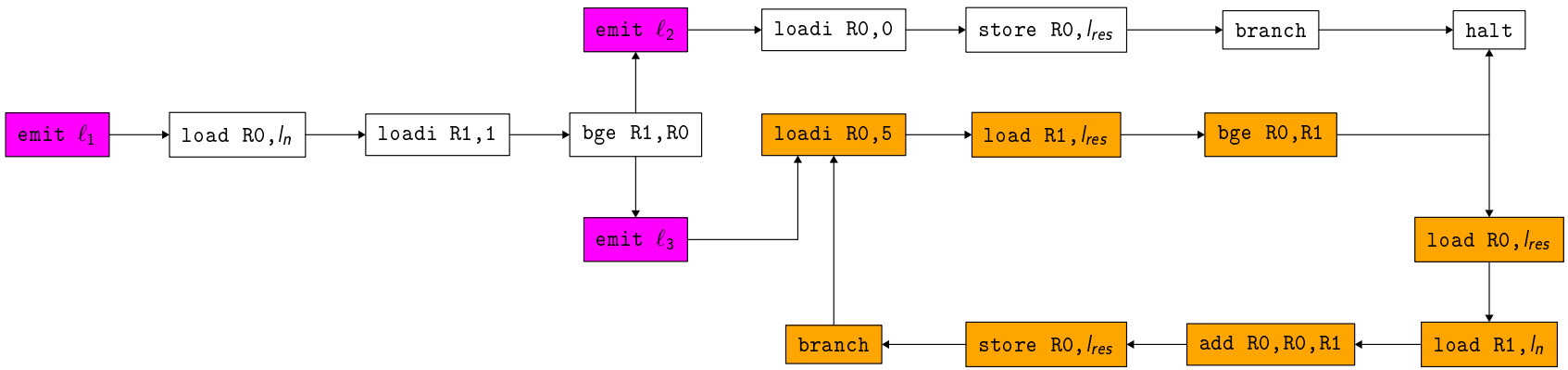
Compilation — Labelled compilation

Theorem: diagram commutativity

$$\left.\begin{array}{l} \mathcal{E}_{\mathsf{Imp}} \circ \mathcal{L}_{\mathsf{Imp}} = \mathsf{Id}_{\mathsf{Imp}} \\ \mathcal{C}_{\mathsf{Imp}} \circ \mathcal{E}_{\mathsf{Imp}} = \mathcal{E}_{\mathsf{VM}} \circ \mathcal{C}^{\mathcal{L}}_{\mathsf{Imp}} \\ \mathcal{C}_{\mathsf{VM}} \circ \mathcal{E}_{\mathsf{VM}} = \mathcal{E}_{\mathsf{ASM}} \circ \mathcal{C}^{\mathcal{L}}_{\mathsf{VM}} \end{array}\right\} \Rightarrow \mathcal{E}_{\mathsf{ASM}} \circ \mathcal{C}^{\mathcal{L}} \circ \mathcal{L}_{\mathsf{Imp}} = \mathcal{C}$$

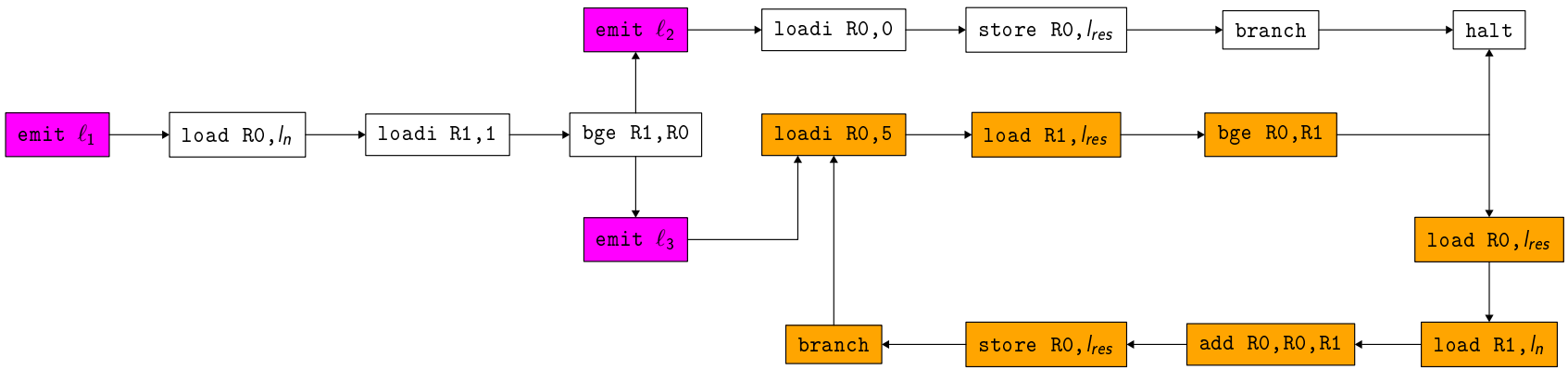
Example: soundness

```
prog
ℓ1:
if n < 1 then
  ℓ2:  res := 0
else
  ℓ3:
  while res < 5 do
        res := res + n
```



Example: soundness

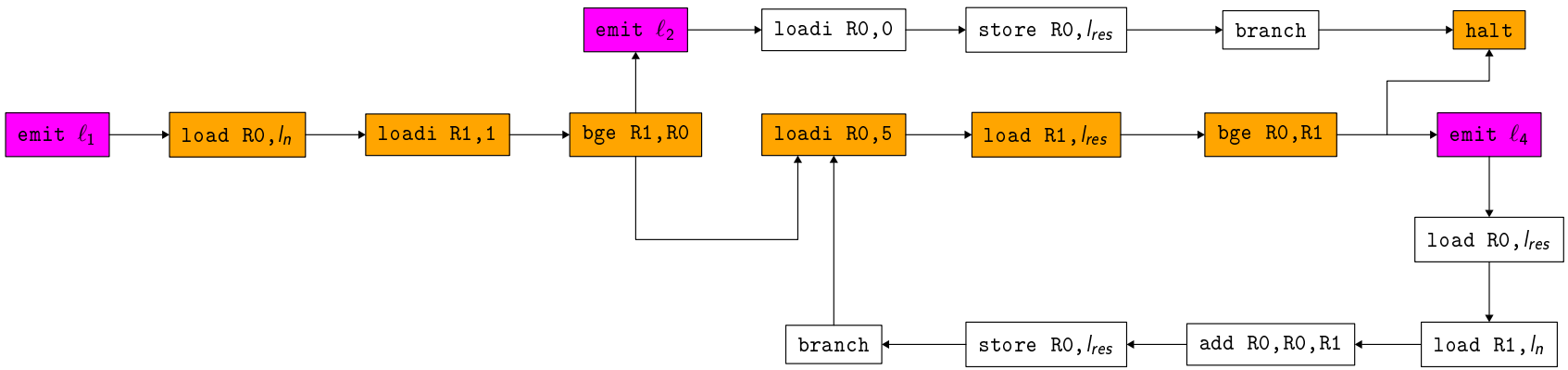
```
prog
ℓ1:
if n < 1 then
  ℓ2:  res := 0
else
  ℓ3:
  while res < 5 do
        res := res + n
```



Loop with no label $\Rightarrow$ not constant cost code

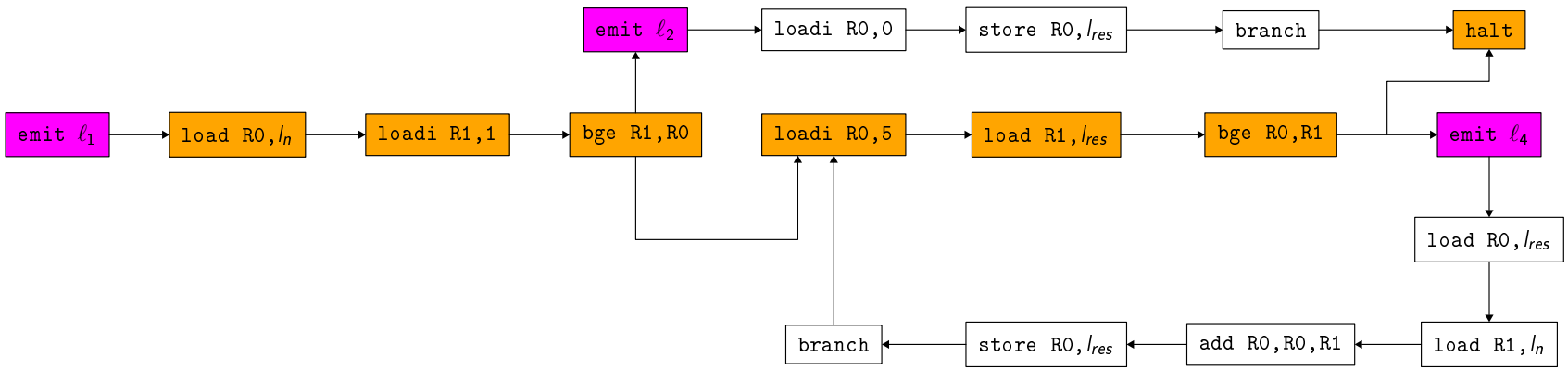
Example: precision

```
prog
ℓ1:
if n < 1 then
  ℓ2:  res := 0
else

  while res < 5 do
    ℓ4:  res := res + n
```



Example: precision

```
prog
ℓ1:
if n < 1 then
  ℓ2:  res := 0
else

  while res < 5 do
    ℓ4:  res := res + n
```



From emit ℓ_1: three paths with different costs $\Rightarrow$ imprecision

Labelling criteria

Soundness

Every reachable code is in the scope of a label.
At least one label inside each loop.

Precision

Two different paths to the next labels have the same cost.

Criteria

The labelling must be **sound** and **precise**.
(This can be syntatically checked on the assembly code.)

√ Nice plus is a reasonable **economy**: not too many labels.

Instrumentation

Cost deduction

Given: ϕ : ASM Instruction list $\to \mathbb{N}$ $\left(\begin{array}{c}\text{loop free}\\ \text{may overapproximate}\end{array}\right)$

Deduced: $\kappa : \mathcal{L} \to \mathbb{N}$

Instrumentation

Cost deduction

Given: ϕ : ASM Instruction list $\to \mathbb{N}$ $\left(\begin{array}{c}\text{loop free}\\ \text{may overapproximate}\end{array}\right)$

Deduced: $\kappa : \mathcal{L} \to \mathbb{N}$

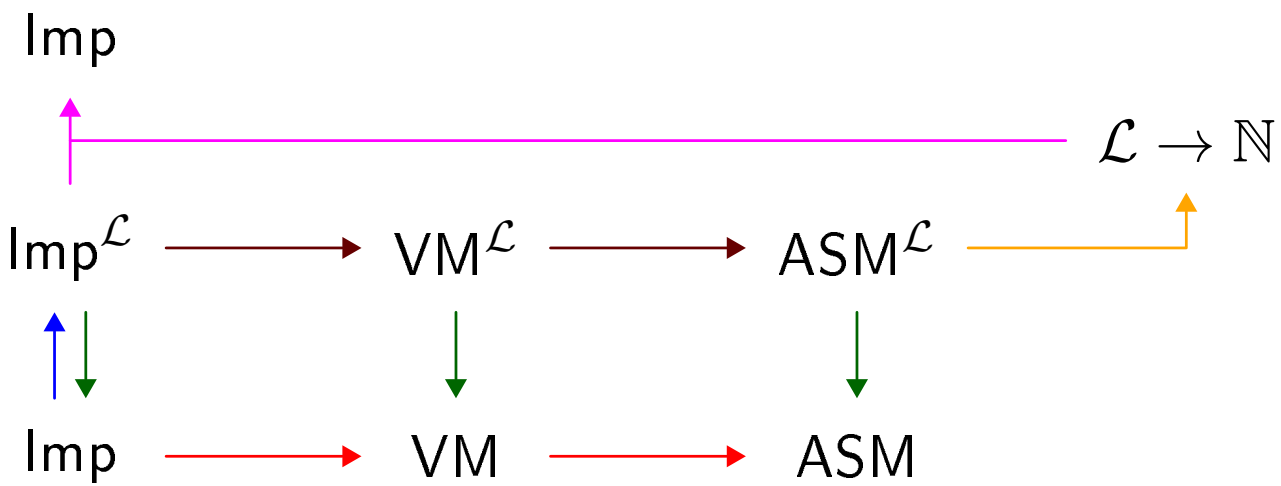
Instrumentation

- Use a fresh variable
- Initialize it to 0
- Replace labels with increments (following κ)

```
prog                          prog
ℓ1:                           _cost := 0;
while res < 5 do      ⟶       _cost := _cost + 2;
  ℓ2:                         while res < 5 do
  res := res + n                _cost := _cost + 4;
                                res := res + n
```

Annotation

Annotation = Instrumentation ∘ Labelling



Labelling — Erasure

Compilation — Labelled compilation

Cost deduction — Instrumentation

Outline

1. Toy compiler
   - Languages
   - Compilation
   - Labelling
   - Annotation
2. Realistic C compiler
   - Architecture
   - Labelling
   - Experiments
3. Future work

$$C \longrightarrow \text{Clight} \longrightarrow \text{Cminor} \longrightarrow \text{RTL}_{\text{abs}} \longrightarrow \text{RTL} \longrightarrow \text{ERTL} \longrightarrow \text{LTL} \longrightarrow \text{LIN} \longrightarrow \text{MIPS}$$

C to Cminor

- Inspired from **CompCert**
  (GNU GPL and INRIA Non-Commercial)
- **C to Clight**: CIL
- **Clight to Cminor**: manual port from Coq to OCaml

RTL_{abs}

C ⟶ Clight ⟶ Cminor ⟶ RTL_{abs} ⟶ RTL ⟶ ERTL ⟶ LTL ⟶ LIN ⟶ MIPS

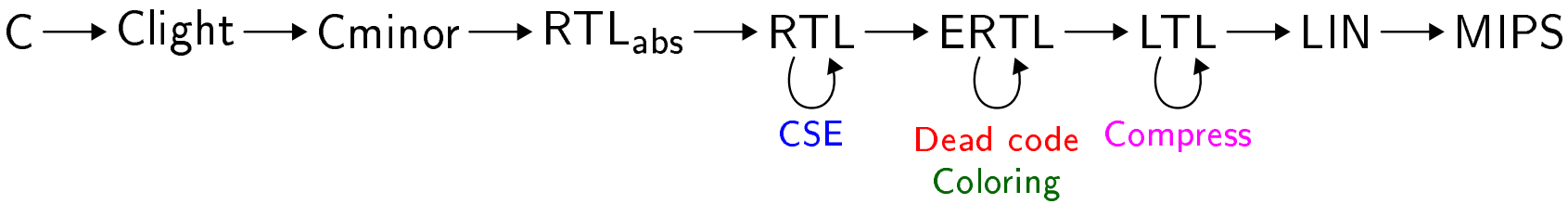
- Home made
- Architecture independent
- Retargetting simplified (inspired from Gimple in GCC)
  - √ Some common optimizations
  - X Some optimisations lost

RTL to MIPS

C ⟶ Clight ⟶ Cminor ⟶ RTL_{abs} ⟶ RTL ⟶ ERTL ⟶ LTL ⟶ LIN ⟶ MIPS

- Adapted from pedagogical Pseudo-Pascal compiler
  (François Pottier, Creative Commons)

Optimizations



- CSE: Common Subexpression Elimination
- Dead code elimination
- Coloring: variable locations
- Graph compression

Restrictions

C ⟶ Clight ⟶ Cminor ⟶ RTL_{abs} ⟶ RTL ⟶ ERTL ⟶ LTL ⟶ LIN ⟶ MIPS

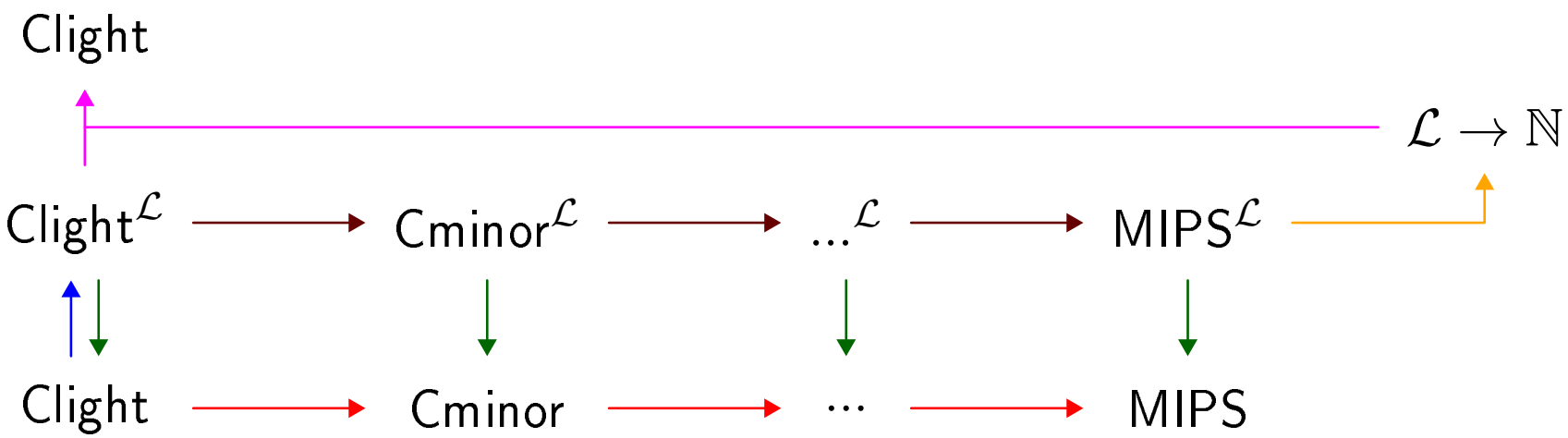
Clight

- X `long long` and `long double` types
- X `longjmp` and `setjmp` instructions
- X Unreasonable forms of `switch`
- X Unprototyped and variable-arity functions

RTL

- X `float`

Overview



Labelling

Erasure

Compilation

Labelled compilation

Cost deduction

Instrumentation

Differences Imp / C

Side effect expressions `y = x++;`

Ternary expressions `x ? y+2*z : z`

Labels and **Gotos** `lbl: ... goto lbl;`

Function calls `f(&x);`

Differences Imp / C

Side effect expressions `y = x++;`

eliminated by CIL $\rightarrow$ `_tmp = x; x = _tmp+1; y = _tmp;`

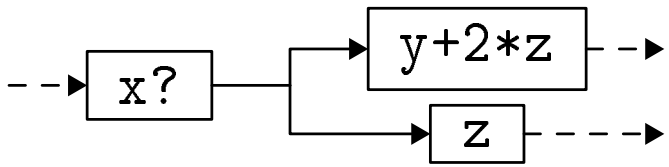
Ternary expressions `x ? y+2*z : z`

Labels and **Gotos** `lbl: ... goto lbl;`

Function calls `f(&x);`

Ternary expressions

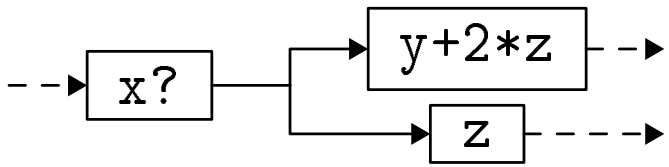
```
x?  (y+2*z) :  z
```



Branching ⇒ 1 label per branch for precision

Ternary expressions

```
x?  (y+2*z) :  z
```



Branching $\Rightarrow$ 1 label per branch for precision

Labelled expressions

```
x?  (ℓ1 :  y+2*z) :  (ℓ2 :  z)
```

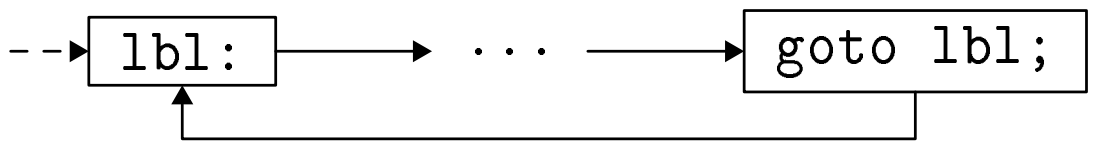
Instrumentation

1) Side effects inside expressions 2) Elimination by CIL

C $\xrightarrow{\text{CIL}}$ Clight $\xrightarrow{\text{Instrument}}$ C $\xrightarrow{\text{CIL}}$ Clight

Labels and Gotos

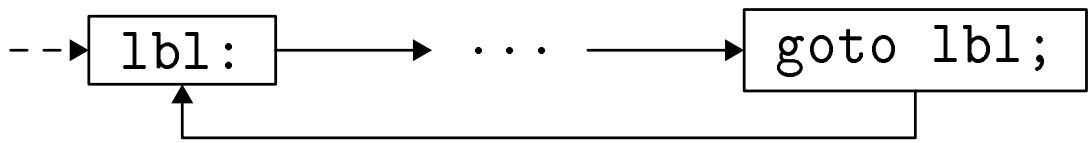
```
lbl:  ...  goto lbl;
```



Label $\Rightarrow$ potential loop $\Rightarrow$ 1 cost label needed

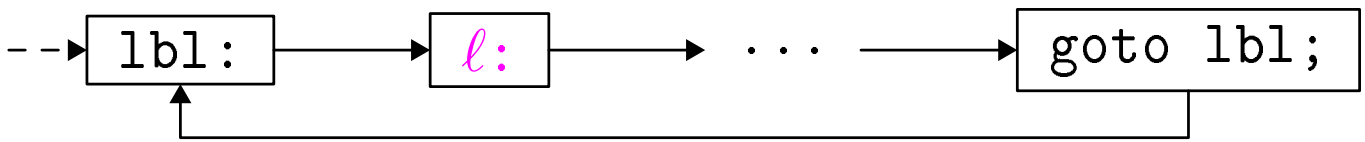
Labels and Gotos

```
lbl:  ...  goto lbl;
```



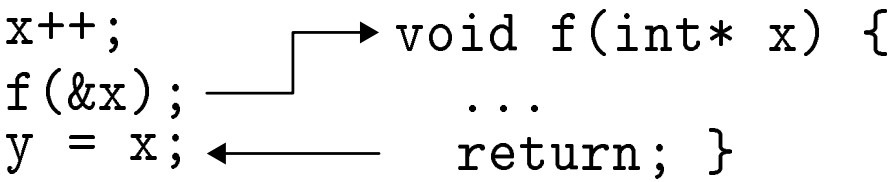
Label ⇒ potential loop ⇒ 1 cost label needed

```
lbl:  ℓ:  ...  goto lbl;
```



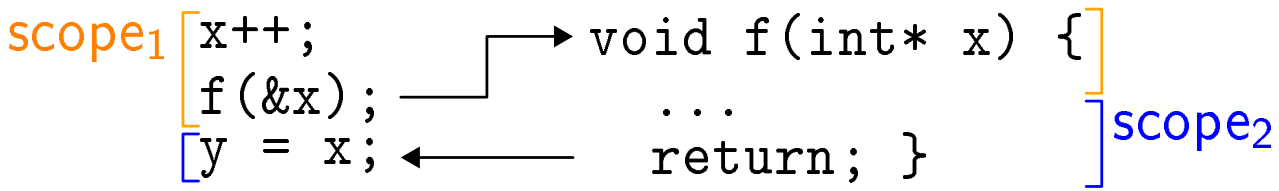
Function calls

Function call: sequential instruction



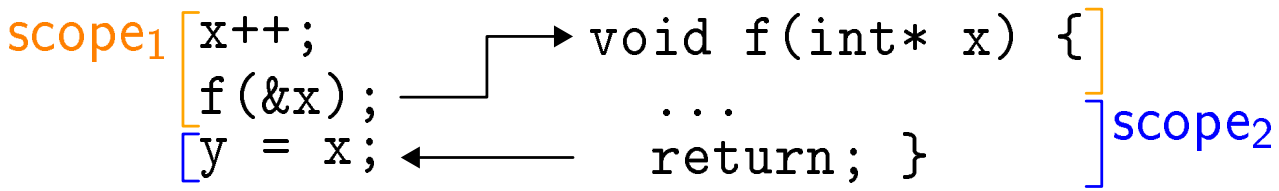
Function calls

Function call: sequential instruction



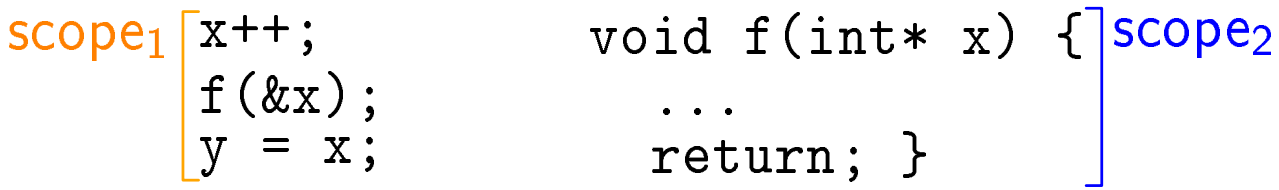
Function calls

Function call: sequential instruction



Function pointer: statically unresolvable destination

√ Each function handles its cost



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

Outline

1. Toy compiler
   - Languages
   - Compilation
   - Labelling
   - Annotation
2. Realistic C compiler
   - Architecture
   - Labelling
   - Experiments
3. Future work

Future work

- Formal proofs in Matita
- Real world example (Lustre)
- Frama-C plugin (demo)
- Retarget to 8051 (moduralize the target architecture)
- Functional languages

Conclusion

CerCo

- C sound compiler (untrusted for now...)
- Correct cost annotations (overapproximation)
- Symbolic cost labels
- Modular proofs

Thank you for your attention!

Questions?