Compilation vérifiée

David Pichardie - David Cachera

école normale supérieure rennes

Les logiciels critiques

Les logiciels critiques

Certains logiciels demandent un très haut niveau de confiance

Les logiciels critiques

Certains logiciels demandent un très haut niveau de confiance

- est-ce que ce programme est sûre ?

```
        .const
        .align  2
___stringlit_1:
        .ascii  "integr(square, 0.0, 1.0,
        .text
        .align  4
_square:
        subl    $12, %esp
        leal    16(%esp), %edx
        movl    %edx, 0(%esp)
        movl    0(%esp), %edx
        movsd   0(%edx), %xmm0
        mulsd   %xmm0, %xmm0
        subl    $8, %esp
        movsd   %xmm0, 0(%esp)
        fldl    0(%esp)
        addl    $8, %esp
        addl    $12, %esp
        ret
        .text
        .align  4
_integr:
        subl    $60, %esp
        leal    64(%esp), %edx
        movl    %edx, 8(%esp)
        movl    %ebx, 20(%esp)
        movl    %esi, 24(%esp)
        movl    8(%esp), %edx
        movl    0(%edx), %esi
        movl    8(%esp), %edx
        movsd   4(%edx), %xmm6
        movsd   %xmm6, 48(%esp)
        movl    8(%esp), %edx
        movsd   12(%edx), %xmm6
        movsd   %xmm6, 32(%esp)
        movl    8(%esp), %edx
        movl    20(%edx), %ebx
        movsd   32(%esp), %xmm6
        movsd   48(%esp), %xmm7
        subsd   %xmm7, %xmm6
        movsd   %xmm6, 32(%esp)
        cvtsi2sd %ebx, %xmm1
        movsd   32(%esp), %xmm6
        divsd   %xmm1, %xmm6
        movsd   %xmm6, 32(%esp)
        xorpd   %xmm6, %xmm6 # +0.0
        movsd   %xmm6, 40(%esp)
L100:
        cmpl    $0, %ebx
        jle     L101
        movsd   48(%esp), %xmm6
        movsd   %xmm6, 0(%esp)
        call    *%esi
        subl    $8, %esp
        fstpl   0(%esp)
        movsd   0(%esp), %xmm0
        addl    $8, %esp
        movsd   40(%esp), %xmm6
        addsd   %xmm0, %xmm6
        movsd   %xmm6, 40(%esp)
        leal    -1(%ebx), %ebx
        movsd   48(%esp), %xmm6
        movsd   32(%esp), %xmm7
        addsd   %xmm7, %xmm6
        movsd   %xmm6, 48(%esp)
        jmp     L100
L101:
        movsd   40(%esp), %xmm6
        movsd   32(%esp), %xmm7
        mulsd   %xmm7, %xmm6
        movsd   %xmm6, 40(%esp)
        fldl    40(%esp)
        movl    20(%esp), %ebx
        movl    24(%esp), %esi
        addl    $60, %esp
        ret
        .text
        .align  4
        .globl _test
_test:
        subl    $44, %esp
        leal    48(%esp), %edx
        movl    %edx, 24(%esp)
        movl    %ebx, 36(%esp)
        movl    24(%esp), %edx
        movl    0(%edx), %eax
        leal    _square, %ebx
        xorpd   %xmm2, %xmm2 # +0.0
        movsd   L102, %xmm1 # 1
        movl    %ebx, 0(%esp)
        movsd   %xmm2, 4(%esp)
        movsd   %xmm1, 12(%esp)
        movl    %eax, 20(%esp)
        call    _integr
        subl    $8, %esp
        fstpl   0(%esp)
        movsd   0(%esp), %xmm0
        addl    $8, %esp
        subl    $8, %esp
        movsd   %xmm0, 0(%esp)
        fldl    0(%esp)
        addl    $8, %esp
        movl    36(%esp), %ebx
        addl    $44, %esp
        ret
        .const_data
        .align  3
L102:   .quad   0x3ff0000000000000
        .text
        .align  4
        .globl _main
_main:
        subl    $44, %esp
        leal    48(%esp), %edx
        movl    %edx, 16(%esp)
        movl    %ebx, 28(%esp)
        movl    16(%esp), %edx
        movl    0(%edx), %ebx
        movl    16(%esp), %edx
        movl    4(%edx), %eax
        cmpl    $2, %ebx
        jge     L103
        movl    $10000000, %ebx
        jmp     L104
L103:
        movl    4(%eax), %eax
        movl    %eax, 0(%esp)
        call    _atoi
        movl    %eax, %ebx
L104:
        movl    %ebx, 0(%esp)
        call    _test
        subl    $8, %esp
        fstpl   0(%esp)
        movsd   0(%esp), %xmm0
        addl    $8, %esp
        leal    ___stringlit_1, %eax
        movl    %eax, 0(%esp)
        movl    %ebx, 4(%esp)
        movsd   %xmm0, 8(%esp)
        call    _printf
        xorl    %eax, %eax
        movl    28(%esp), %ebx
        addl    $44, %esp
        ret
```

critical_prog.ppc

Les logiciels critiques

Certains logiciels demandent un très haut niveau de confiance

- est-ce que ce programme est sûre ?

Deux options:

```
        .const
        .align  2
___stringlit_1:
        .ascii  "integr(square, 0.0, 1.0,
        .text
        .align  4
_square:
        subl    $12, %esp
        leal    16(%esp), %edx
        movl    %edx, 0(%esp)
        movl    0(%esp), %edx
        movsd   0(%edx), %xmm0
        mulsd   %xmm0, %xmm0
        subl    $8, %esp
        movsd   %xmm0, 0(%esp)
        fldl    0(%esp)
        addl    $8, %esp
        addl    $12, %esp
        ret
        .text
        .align  4
_integr:
        subl    $60, %esp
        leal    64(%esp), %edx
        movl    %edx, 8(%esp)
        movl    %ebx, 20(%esp)
        movl    %esi, 24(%esp)
        movl    8(%esp), %edx
        movl    0(%edx), %esi
        movl    8(%esp), %edx
        movsd   4(%edx), %xmm6
        movsd   %xmm6, 48(%esp)
        movl    8(%esp), %edx
        movsd   12(%edx), %xmm6
        movsd   %xmm6, 32(%esp)
        movl    8(%esp), %edx
        movl    20(%edx), %ebx
        movsd   32(%esp), %xmm6
        movsd   48(%esp), %xmm7
        subsd   %xmm7, %xmm6
        movsd   %xmm6, 32(%esp)
        cvtsi2sd %ebx, %xmm1
        movsd   32(%esp), %xmm6
        divsd   %xmm1, %xmm6
        movsd   %xmm6, 32(%esp)
        xorpd   %xmm6, %xmm6 # +0.0
        movsd   %xmm6, 40(%esp)
L100:
        cmpl    $0, %ebx
        jle     L101
        movsd   48(%esp), %xmm6
        movsd   %xmm6, 0(%esp)
        call    *%esi
        subl    $8, %esp
        fstpl   0(%esp)
        movsd   0(%esp), %xmm0
        addl    $8, %esp
        movsd   40(%esp), %xmm6
        addsd   %xmm0, %xmm6
        movsd   %xmm6, 40(%esp)
        leal    -1(%ebx), %ebx
        movsd   48(%esp), %xmm6
        movsd   32(%esp), %xmm7
        addsd   %xmm7, %xmm6
        movsd   %xmm6, 48(%esp)
        jmp     L100
L101:
        movsd   40(%esp), %xmm6
        movsd   32(%esp), %xmm7
        mulsd   %xmm7, %xmm6
        movsd   %xmm6, 40(%esp)
        fldl    40(%esp)
        movl    20(%esp), %ebx
        movl    24(%esp), %esi
        addl    $60, %esp
        ret
        .text
        .align  4
        .globl _test
_test:
        subl    $44, %esp
        leal    48(%esp), %edx
        movl    %edx, 24(%esp)
        movl    %ebx, 36(%esp)
        movl    24(%esp), %edx
        movl    0(%edx), %eax
        leal    _square, %ebx
        xorpd   %xmm2, %xmm2 # +0.0
        movsd   L102, %xmm1 # 1
        movl    %ebx, 0(%esp)
        movsd   %xmm2, 4(%esp)
        movsd   %xmm1, 12(%esp)
        movl    %eax, 20(%esp)
        call    _integr
        subl    $8, %esp
        fstpl   0(%esp)
        movsd   0(%esp), %xmm0
        addl    $8, %esp
        subl    $8, %esp
        movsd   %xmm0, 0(%esp)
        fldl    0(%esp)
        addl    $8, %esp
        movl    36(%esp), %ebx
        addl    $44, %esp
        ret
        .const_data
        .align  3
L102:   .quad   0x3ff0000000000000
        .text
        .align  4
        .globl _main
_main:
        subl    $44, %esp
        leal    48(%esp), %edx
        movl    %edx, 16(%esp)
        movl    %ebx, 28(%esp)
        movl    16(%esp), %edx
        movl    0(%edx), %ebx
        movl    16(%esp), %edx
        movl    4(%edx), %eax
        cmpl    $2, %ebx
        jge     L103
        movl    $10000000, %ebx
        jmp     L104
L103:
        movl    4(%eax), %eax
        movl    %eax, 0(%esp)
        call    _atoi
        movl    %eax, %ebx
L104:
        movl    %ebx, 0(%esp)
        call    _test
        subl    $8, %esp
        fstpl   0(%esp)
        movsd   0(%esp), %xmm0
        addl    $8, %esp
        leal    ___stringlit_1, %eax
        movl    %eax, 0(%esp)
        movl    %ebx, 4(%esp)
        movsd   %xmm0, 8(%esp)
        call    _printf
        xorl    %eax, %eax
        movl    28(%esp), %ebx
        addl    $44, %esp
        ret
```

critical_prog.ppc

Les logiciels critiques

Certains logiciels demandent un très haut niveau de confiance

- est-ce que ce programme est sûre ?

Deux options:

- *qualifier* le programme PPC comme s'il avait été écrit à la main
  (tests intensive, revue manuel de code...)

```
        .const
        .align  2
___stringlit_1:
        .ascii  "integr(square, 0.0, 1.0,
        .text
        .align  4
_square:
        subl    $12, %esp
        leal    16(%esp), %edx
        movl    %edx, 0(%esp)
        movl    0(%esp), %edx
        movsd   0(%edx), %xmm0
        mulsd   %xmm0, %xmm0
        subl    $8, %esp
        movsd   %xmm0, 0(%esp)
        fldl    0(%esp)
        addl    $8, %esp
        addl    $12, %esp
        ret
        .text
        .align  4
_integr:
        subl    $60, %esp
        leal    64(%esp), %edx
        movl    %edx, 8(%esp)
        movl    %ebx, 20(%esp)
        movl    %esi, 24(%esp)
        movl    8(%esp), %edx
        movl    0(%edx), %esi
        movl    8(%esp), %edx
        movsd   4(%edx), %xmm6
        movsd   %xmm6, 48(%esp)
        movl    8(%esp), %edx
        movsd   12(%edx), %xmm6
        movsd   %xmm6, 32(%esp)
        movl    8(%esp), %edx
        movl    20(%edx), %ebx
        movsd   32(%esp), %xmm6
        movsd   48(%esp), %xmm7
        subsd   %xmm7, %xmm6
        movsd   %xmm6, 32(%esp)
        cvtsi2sd %ebx, %xmm1
        movsd   32(%esp), %xmm6
        divsd   %xmm1, %xmm6
        movsd   %xmm6, 32(%esp)
        xorpd   %xmm6, %xmm6 # +0.0
        movsd   %xmm6, 40(%esp)
L100:
        cmpl    $0, %ebx
        jle     L101
        movsd   48(%esp), %xmm6
        movsd   %xmm6, 0(%esp)
        call    *%esi
        subl    $8, %esp
        fstpl   0(%esp)
        movsd   0(%esp), %xmm0
        addl    $8, %esp
        movsd   40(%esp), %xmm6
        addsd   %xmm0, %xmm6
        movsd   %xmm6, 40(%esp)
        leal    -1(%ebx), %ebx
        movsd   48(%esp), %xmm6
        movsd   32(%esp), %xmm7
        addsd   %xmm7, %xmm6
        movsd   %xmm6, 48(%esp)
        jmp     L100
L101:
        movsd   40(%esp), %xmm6
        movsd   32(%esp), %xmm7
        mulsd   %xmm7, %xmm6
        movsd   %xmm6, 40(%esp)
        fldl    40(%esp)
        movl    20(%esp), %ebx
        movl    24(%esp), %esi
        addl    $60, %esp
        ret
        .text
        .align  4
        .globl _test
_test:
        subl    $44, %esp
        leal    48(%esp), %edx
        movl    %edx, 24(%esp)
        movl    %ebx, 36(%esp)
        movl    24(%esp), %edx
        movl    0(%edx), %eax
        leal    _square, %ebx
        xorpd   %xmm2, %xmm2 # +0.0
        movsd   L102, %xmm1 # 1
        movl    %ebx, 0(%esp)
        movsd   %xmm2, 4(%esp)
        movsd   %xmm1, 12(%esp)
        movl    %ebx, 20(%esp)
        call    _integr
        subl    $8, %esp
        fstpl   0(%esp)
        movsd   0(%esp), %xmm0
        addl    $8, %esp
        subl    $8, %esp
        movsd   %xmm0, 0(%esp)
        fldl    0(%esp)
        addl    $8, %esp
        movl    36(%esp), %ebx
        addl    $44, %esp
        ret
        .const_data
        .align  3
L102:   .quad   0x3ff0000000000000
        .text
        .align  4
        .globl _main
_main:
        subl    $44, %esp
        leal    48(%esp), %edx
        movl    %edx, 16(%esp)
        movl    %ebx, 28(%esp)
        movl    16(%esp), %edx
        movl    0(%edx), %ebx
        movl    16(%esp), %edx
        movl    4(%edx), %eax
        cmpl    $2, %ebx
        jge     L103
        movl    $10000000, %ebx
        jmp     L104
L103:
        movl    4(%eax), %eax
        movl    %eax, 0(%esp)
        call    _atoi
        movl    %eax, %ebx
L104:
        movl    %ebx, 0(%esp)
        call    _test
        subl    $8, %esp
        fstpl   0(%esp)
        movsd   0(%esp), %xmm0
        addl    $8, %esp
        leal    ___stringlit_1, %eax
        movl    %eax, 0(%esp)
        movl    %ebx, 4(%esp)
        movsd   %xmm0, 8(%esp)
        call    _printf
        xorl    %eax, %eax
        movl    28(%esp), %ebx
        addl    $44, %esp
        ret
```

critical_prog.ppc

Les logiciels critiques

Certains logiciels demandent un très haut niveau de confiance

- est-ce que ce programme est sûre ?

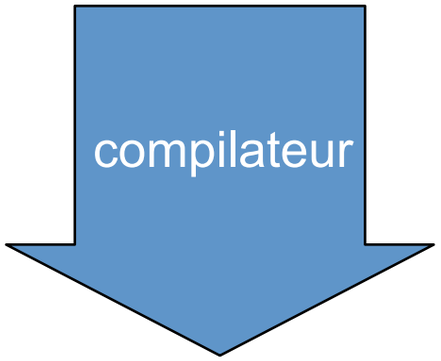
Deux options:

- *qualifier* le programme PPC comme s'il avait été écrit à la main
  (tests intensive, revue manuel de code...)

- *qualifier* le programme au niveau source
  (analyse statique, model checking, preuve de programme)

```c
#include <stdlib.h>
#include <stdio.h>

static double square(double x)
{
  return x * x;
}

static double integr(double (*f)(double), double low, double high, int n)
{
  double h, x, s;
  int i;

  h = (high - low) / n;
  s = 0;
  for (i = n, x = low; i > 0; i--, x += h) s += f(x);
  return s * h;
}

double test(int n)
{
  return integr(square, 0.0, 1.0, n);
}

int main(int argc, char ** argv)
{
  int n; double r;
  if (argc >= 2) n = atoi(argv[1]); else n = 10000000;
  r = test(n);
  printf("integr(square, 0.0, 1.0, %d) = %g\n", n, r);
  return 0;
}
```

critical_prog.c



critical_prog.ppc

Les logiciels critiques

Certains logiciels demandent un très haut niveau de confiance

- est-ce que ce programme est sûre ?

Deux options:

- *qualifier* le programme PPC comme s'il avait été écrit à la main
  (tests intensive, revue manuel de code...)

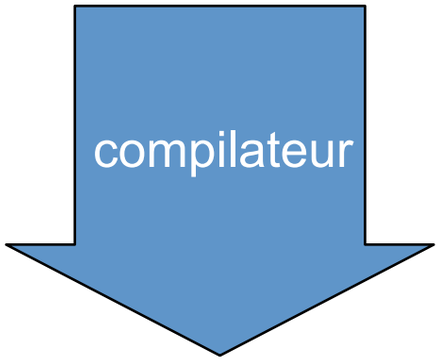
- *qualifier* le programme au niveau source
  (analyse statique, model checking, preuve de programme)

La seconde option est préférée en pratique

- comment faire confiance au compilateur ?
- cet exposé : comment vérifier le compilateur !

```c
#include <stdlib.h>
#include <stdio.h>

static double square(double x)
{
  return x * x;
}

static double integr(double (*f)(double), double low, double high, int n)
{
  double h, x, s;
  int i;

  h = (high - low) / n;
  s = 0;
  for (i = n, x = low; i > 0; i--, x += h) s += f(x);
  return s * h;
}

double test(int n)
{
  return integr(square, 0.0, 1.0, n);
}

int main(int argc, char ** argv)
{
  int n; double r;
  if (argc >= 2) n = atoi(argv[1]); else n = 10000000;
  r = test(n);
  printf("integr(square, 0.0, 1.0, %d) = %g\n", n, r);
  return 0;
}
```

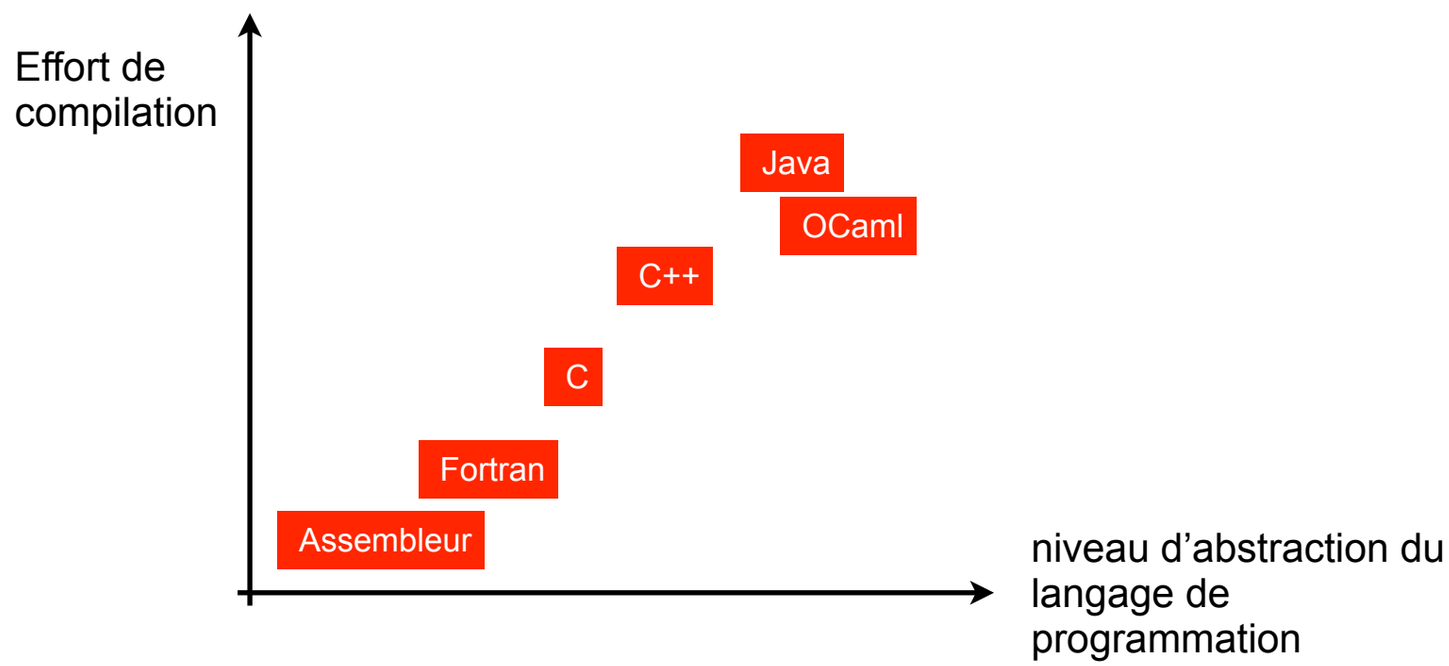
critical_prog.c



critical_prog.ppc

Compiler : est-ce difficile ?

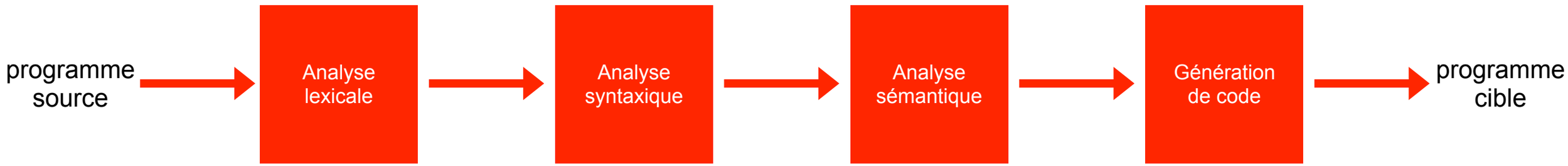
Les langages de haut-niveau d'abstraction diminuent le travail du programmeur mais augmentent celui du programmeur.



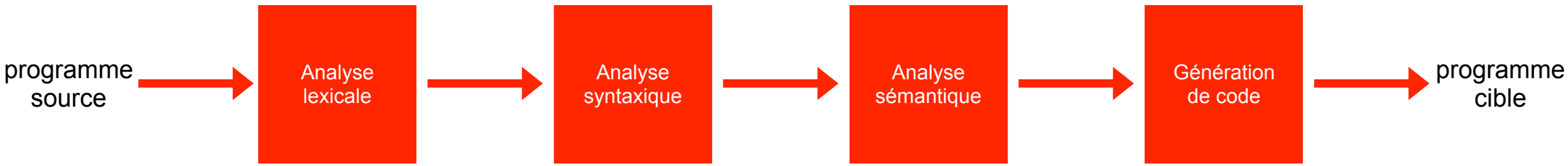
Autres difficultés

- il faut passer passer à l'échelle des programmes gigantesques
- les langages dynamiques (Java, Javascript) doivent être compilés pendant l'exécution : *Just-in-time compiler*
- il faut tenir compte des spécificités de chaque architecture cible

Compilation : les grandes étapes



Compilation : les grandes étapes



Théorie des automates finis et des langages rationnels

Théorie des automates à pile et des langages algébriques

Sémantique des langages de programmation

Interprétation abstraite

Algorithmique des problèmes NP-complets

Algorithmique parallèle

Architecture des ordinateurs

Compilation optimisante

Un exemple : le produit scalaire

```
double p = 0.0;
for (int i = 0; i < N, i++)
   p = p + A[i] * B[i];
return p;
```

Compilation optimisante

Passage en code intermédiaire *3-adresses*

```
double p = 0.0;
for (int i = 0; i < N, i++)
   p = p + A[i] * B[i];
return p;
```

```
p = 0.0;
for (int i = 0; i < N, i++)
   a = load(A + i * 8);
   b = load(B + i * 8);
   c = a * b;
   p = p + c;
}
...
```

Compilation optimisante

Les multiplications sont coûteuses...

```
p = 0.0;
for (int i = 0; i < N, i++)
   a = load(A + i * 8);
   b = load(B + i * 8);
   c = a * b;
   p = p + c;
}
...
```

```
p = 0.0;
for (int i = 0; i < N, i++)
   a = load(A);
   b = load(B);
   c = a * b;
   p = p + c;
   A = A + 8;
   B = B + 8;
}
...
```

Compilation optimisante

On regroupe l'itération i et i+1 dans un même corps de boucle

```
p = 0.0;
for (int i = 0; i < N, i++)
   a = load(A);
   b = load(B);
   c = a * b;
   p = p + c;
   A = A + 8;
   B = B + 8;
}
...
```

```
p = 0.0;
for (int i = 0; i < N-1, i += 2)
   a1 = load(A);
   b1 = load(B);
   c1 = a1 * b1;
   p = p + c1;
   a2 = load(A + 8);
   b2 = load(B + 8);
   c2 = a2 * b2;
   p = p + c2;
   A = A + 16;
   B = B + 16;
}
if (i < N) { ... }
```

Compilation optimisante

On réordonne certaines instructions pour profiter du pipeline d'instructions

```
p = 0.0;
for (int i = 0; i < N-1, i += 2)
   a1 = load(A);
   b1 = load(B);
   c1 = a1 * b1;
   p = p + c1;
   a2 = load(A + 8);
   b2 = load(B + 8);
   c2 = a2 * b2;
   p = p + c2;
   A = A + 16;
   B = B + 16;
}
if (i < N) { ... }
```

```
p = 0.0;
for (int i = 0; i < N-1, i += 2)
   a1 = load(A);
   b1 = load(B);
   a2 = load(A + 8);
   b2 = load(B + 8);

   c1 = a1 * b1;
   c2 = a2 * b2;
   A = A + 16;

   p = p + c1;
   B = B + 16;

   p = p + c2;
}
if (i < N) { ... }
```

Compilation optimisante

On effectue les lectures mémoires une itération en avance

```
p = 0.0;
for (int i = 0; i < N-1, i += 2)
   a1 = load(A);
   b1 = load(B);
   a2 = load(A + 8);
   b2 = load(B + 8);

   c1 = a1 * b1;
   c2 = a2 * b2;
   A = A + 16;

   p = p + c1;
   B = B + 16;

   p = p + c2;
}
if (i < N) { ... }
```

```
...
a3 = load(A); a4 = load(A + 8);
b3 = load(B); b4 = load(B + 8);
for (int i = 0; i < N-3, i += 2)
   a1 = a3; a3 = load(A + 16);
   b1 = b3; b3 = load(B + 16);
   c1 = a1 * b1;
   a2 = a4; a4 = load(A + 24);
   b2 = b4; b4 = load(B + 24);
   c2 = a2 * b2;
   A = A + 16;
   p = p + c1;
   B = B + 16;
   p = p + c2;
}
c1 = a3 * b3;
c2 = a4 * b4;
p = p + c1;
p = p + c2;
```

Compilation optimisante

Version complète

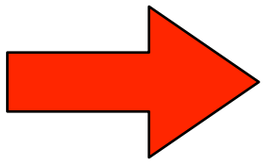
```
...
a3 = load(A); a4 = load(A + 8);
b3 = load(B); b4 = load(B + 8);
for (int i = 0; i < N-3, i += 2)
   a1 = a3; a3 = load(A + 16);
   b1 = b3; b3 = load(B + 16);
   c1 = a1 * b1;
   a2 = a4; a4 = load(A + 24);
   b2 = b4; b4 = load(B + 24);
   c2 = a2 * b2;
   A = A + 16;
   p = p + c1;
   B = B + 16;
   p = p + c2;
}
c1 = a3 * b3;
c2 = a4 * b4;
p = p + c1;
p = p + c2;
```

```
p = 0.0;
i = 0;
if (N >=4) {
  a3 = load(A); a4 = load(A + 8);
  b3 = load(B); b4 = load(B + 8);
  for (; i < N-3, i += 2)
    a1 = a3; a3 = load(A + 16);
    b1 = b3; b3 = load(B + 16);
    c1 = a1 * b1;
    a2 = a4; a4 = load(A + 24);
    b2 = b4; b4 = load(B + 24);
    c2 = a2 * b2;
    A = A + 16; p = p + c1;
    B = B + 16; p = p + c2;
  }
  c1 = a3 * b3;
  c2 = a4 * b4;
  p = p + c1; p = p + c2;
  A = A + 16; B = B + 16;
}
for (; i < N; i++) {
  a = load(A); A = A + 8;
  b = load(B); B = B + 8;
  c = a * b;
  p = p + c;
}
```

Compilation optimisante

Version complète

```
double p = 0.0;
for (int i = 0; i < N, i++)
   p = p + A[i] * B[i];
return p;
```



programme
2 à 3 fois plus
efficace

```
p = 0.0;
i = 0;
if (N >=4) {
  a3 = load(A); a4 = load(A + 8);
  b3 = load(B); b4 = load(B + 8);
  for (; i < N-3, i += 2)
    a1 = a3; a3 = load(A + 16);
    b1 = b3; b3 = load(B + 16);
    c1 = a1 * b1;
    a2 = a4; a4 = load(A + 24);
    b2 = b4; b4 = load(B + 24);
    c2 = a2 * b2;
    A = A + 16; p = p + c1;
    B = B + 16; p = p + c2;
  }
  c1 = a3 * b3;
  c2 = a4 * b4;
  p = p + c1; p = p + c2;
  A = A + 16; B = B + 16;
}
for (; i < N; i++) {
  a = load(A); A = A + 8;
  b = load(B); B = B + 8;
  c = a * b;
  p = p + c;
}
```

Les compilateurs se trompent-ils parfois ?

Les compilateurs se trompent-ils parfois ?

NULLSTONE isolated defects [in integer division] in twelve of twenty commercially available compilers that were evaluated.

http://www.nullstone.com/htmls/category/divide.htm

Les compilateurs se trompent-ils parfois ?

NULLSTONE isolated defects [in integer division] in twelve of twenty commercially available compilers that were evaluated.

http://www.nullstone.com/htmls/category/divide.htm

We tested thirteen production-quality C compilers and, for each, found situations in which the compiler generated incorrect code for accessing volatile variables.

E. Eide & J. Regehr, EMSOFT 2008

Les compilateurs se trompent-ils parfois ?

NULLSTONE isolated defects [in integer division] in twelve of twenty commercially available compilers that were evaluated.

http://www.nullstone.com/htmls/category/divide.htm

We tested thirteen production-quality C compilers and, for each, found situations in which the compiler generated incorrect code for accessing volatile variables.

E. Eide & J. Regehr, EMSOFT 2008

We created a tool that generates random C programs, and then spent two and a half years using it to find compiler bugs. So far, we have reported more than 290 previously unknown bugs to compiler developers. Moreover, every compiler that we tested has been found to crash and also to silently generate wrong code when presented with valid inputs.

J. Regehr et al, 2010

Le projet CompCert

Xavier Leroy et al.

Un compilateur développé et prouvé correct, dans l'assistant de preuve Coq

- langage source: un très grand sous-ensemble de C
- langage cible : assembleur PowerPC/ARM/x86
- génère du code *raisonnablement compacte et efficace*
  ⇒ génération de code minutieuse; quelques optimisations
- en cours de transfert pour l'industrie aéronautique...

Comment vérifier un compilateur ?

Comment vérifier un compilateur ?

Une idée simple :

Comment vérifier un compilateur ?

Une idée simple :

Il faut programmer et prouver le compilateur dans un même langage !

Comment vérifier un compilateur ?

Une idée simple :

Il faut programmer et prouver le compilateur dans un même langage !

Quel langage ?

Comment vérifier un compilateur ?

Une idée simple :

Il faut programmer et prouver le compilateur dans un même langage !

Quel langage ?

Vérifier un compilateur en Coq

La recette

Assistant de preuve

Vérifier un compilateur en Coq
La recette

Programmer le compilateur en Coq

```
Definition compiler
  (p:source) : option target := ...
```

Compilateur

Assistant de preuve

Vérifier un compilateur en Coq
La recette

Programmer le compilateur en Coq

```
Definition compiler
  (p:source) : option target := ...
```

Écrire son théorème de correction, en s'appuyant sur une description formelles des sémantiques des langages

```
Theorem compiler_is_correct :
  ∀ p, compiler p = Some p' →
       behaviors(p') ⊆ behaviors(p)
```

Compilateur

Sémantiques sources & cibles

Assistant de preuve

Vérifier un compilateur en Coq
La recette

Programmer le compilateur en Coq

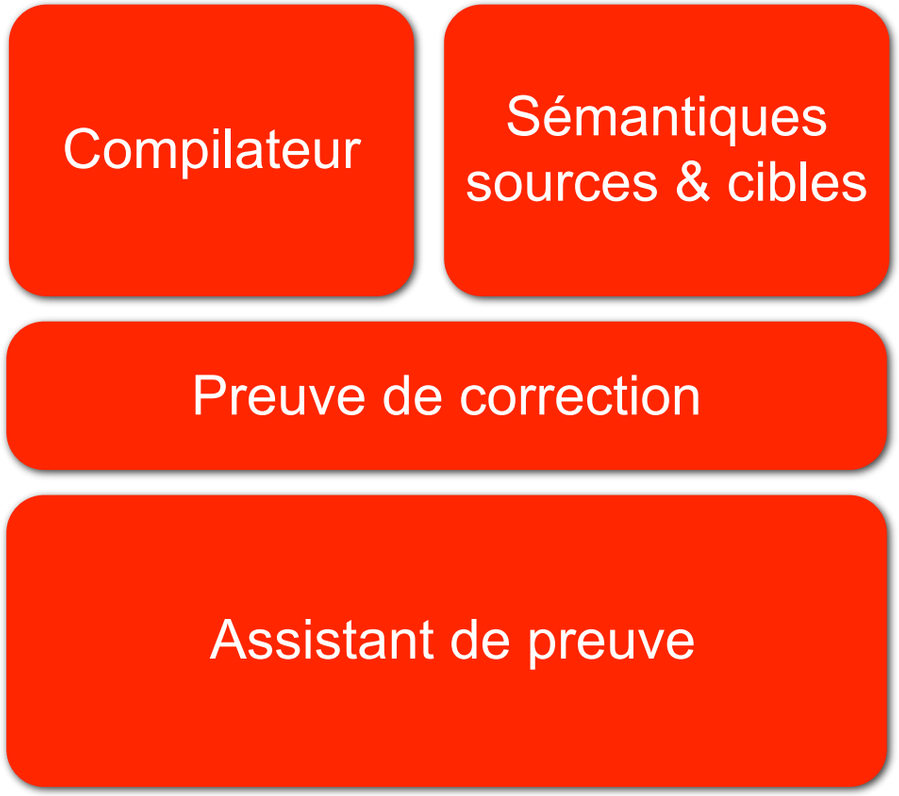
```
Definition compiler
  (p:source) : option target := ...
```

Écrire son théorème de correction, en s'appuyant sur une description formelles des sémantiques des langages

```
Theorem compiler_is_correct :
  ∀ p, compiler p = Some p' →
       behaviors(p') ⊆ behaviors(p)
```

Nous prouvons *interactivement* ce théorème

```
Proof. ... (* few days later *) ... Qed.
```



Vérifier un compilateur en Coq
La recette

Programmer le compilateur en Coq

```
Definition compiler
  (p:source) : option target := ...
```

Écrire son théorème de correction, en s'appuyant sur une description formelles des sémantiques des langages

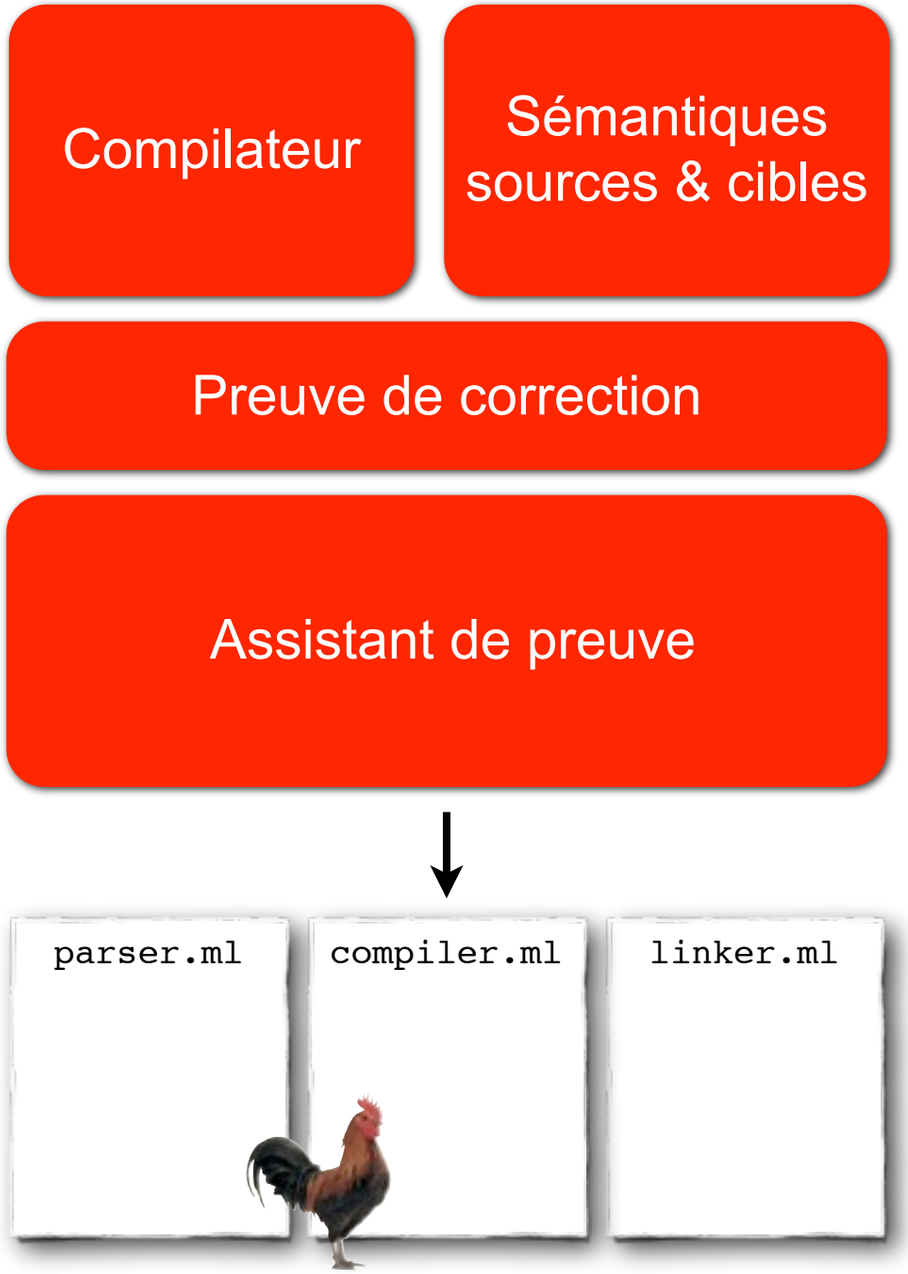
```
Theorem compiler_is_correct :
  ∀ p, compiler p = Some p' →
       behaviors(p') ⊆ behaviors(p)
```

Nous prouvons *interactivement* ce théorème

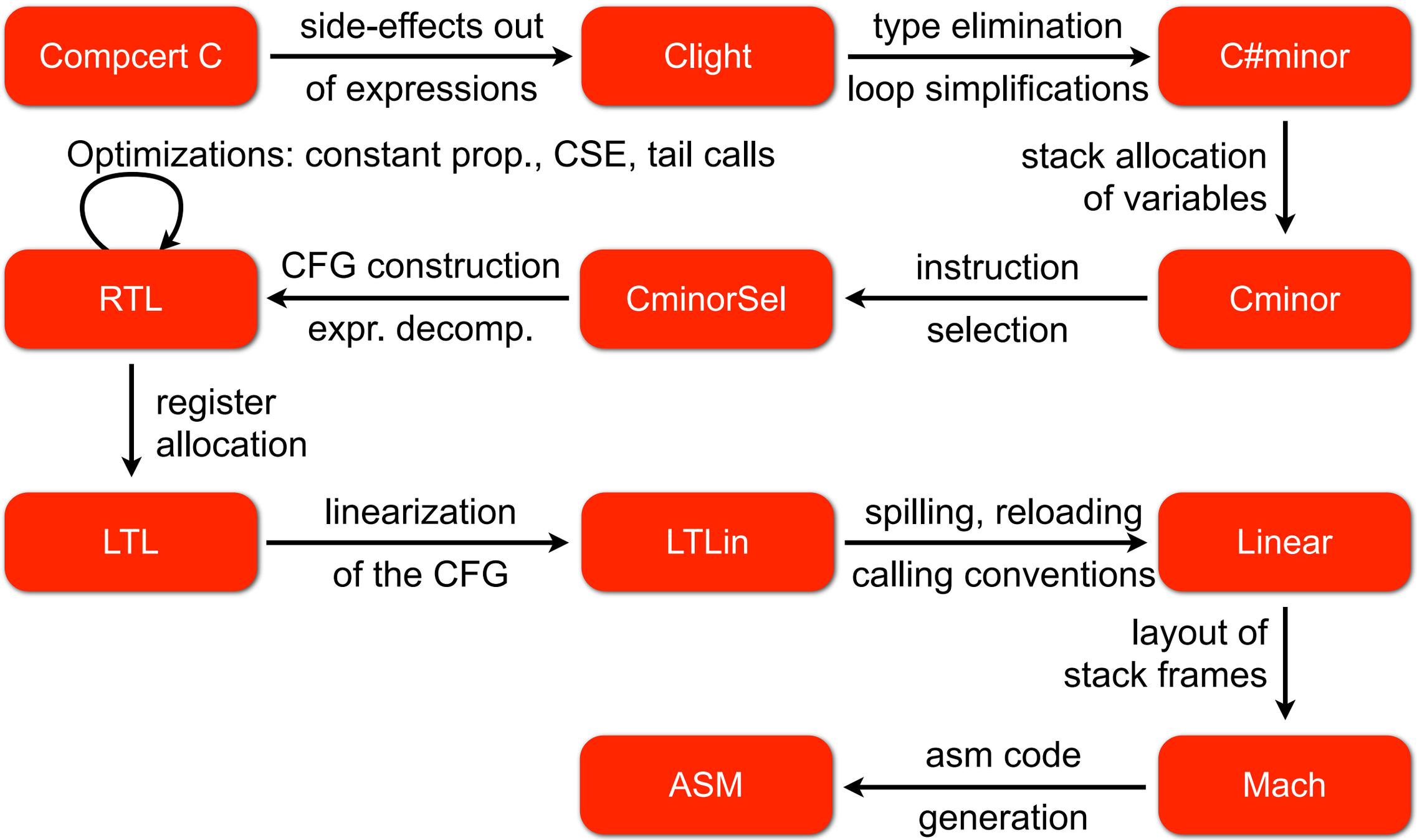
```
Proof. ... (* few days later *) ... Qed.
```

Nous utilisons l'*extraction* de Coq pour générer un compilateur écrit en OCaml

```
Extraction compiler.
```



Les *entrailles* d'un compilateur C vérifié



Le théorème principal

Après 50 000 lignes de Coq et 4 hommes/ans d'effort

```
Theorem transf_c_program_is_refinement:
  ∀ p tp,
  transf_c_program p = OK tp →
  (∀ beh, exec_C_program p beh → not_wrong beh) →
  (∀ beh, exec_asm_program tp beh → exec_C_program p beh).
```

Le comportement `beh` modélise la termination ou la divergence ou la fin abrupte (erreur) d'une exécution.

Compiler verification patterns

(for each pass)

Compiler verification patterns

(for each pass)

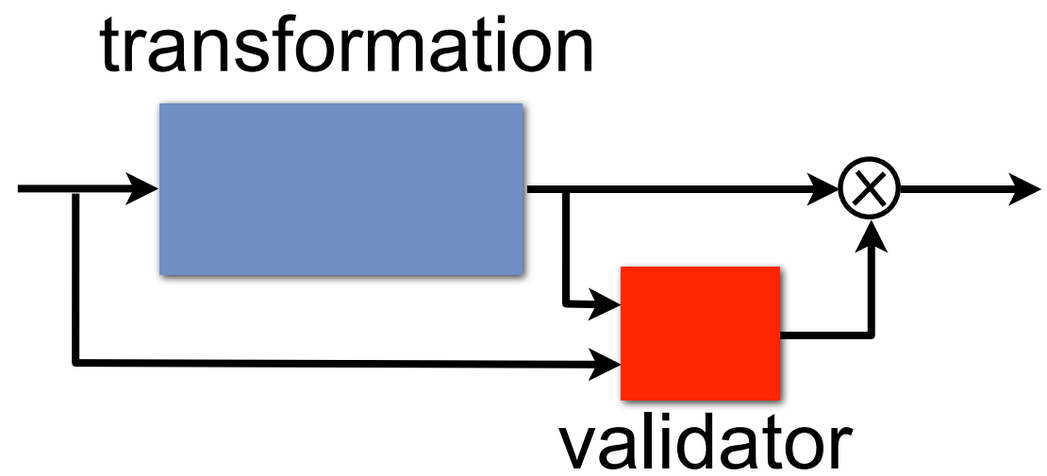
Verified transformation

Compiler verification patterns

(for each pass)

Verified transformation

Verified translation validation



= formally verified

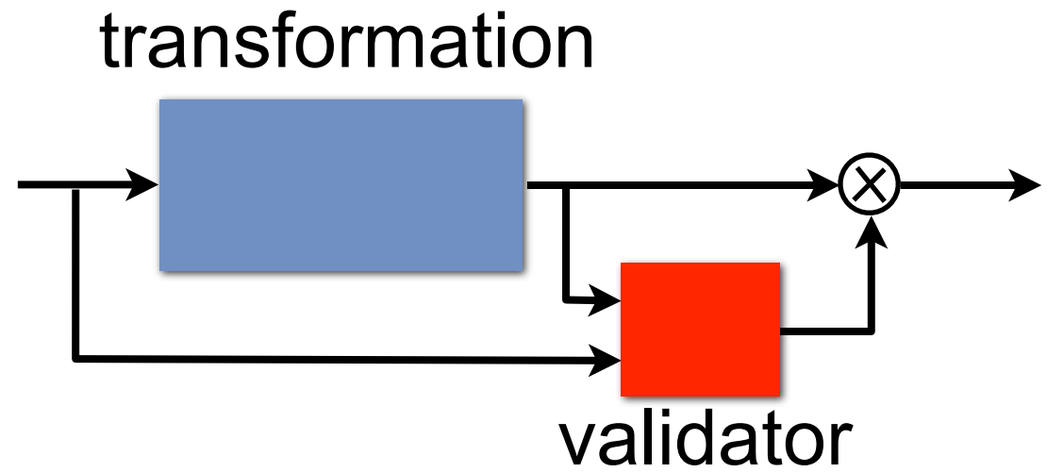
= not verified

Compiler verification patterns

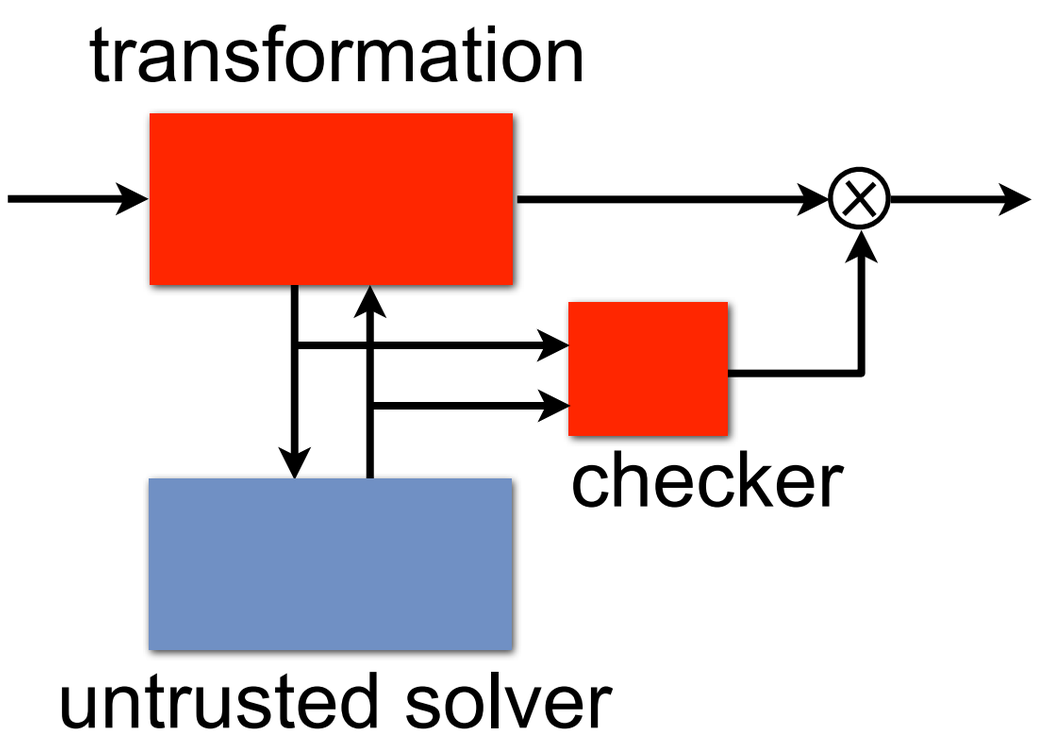
(for each pass)

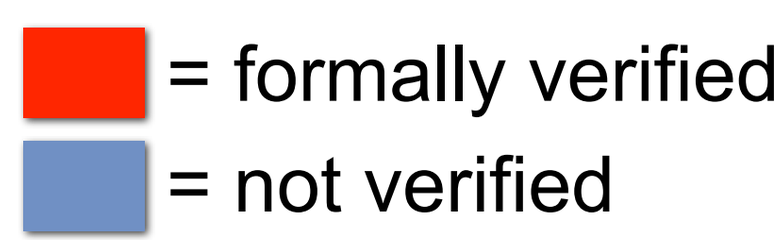
Verified transformation

Verified translation validation



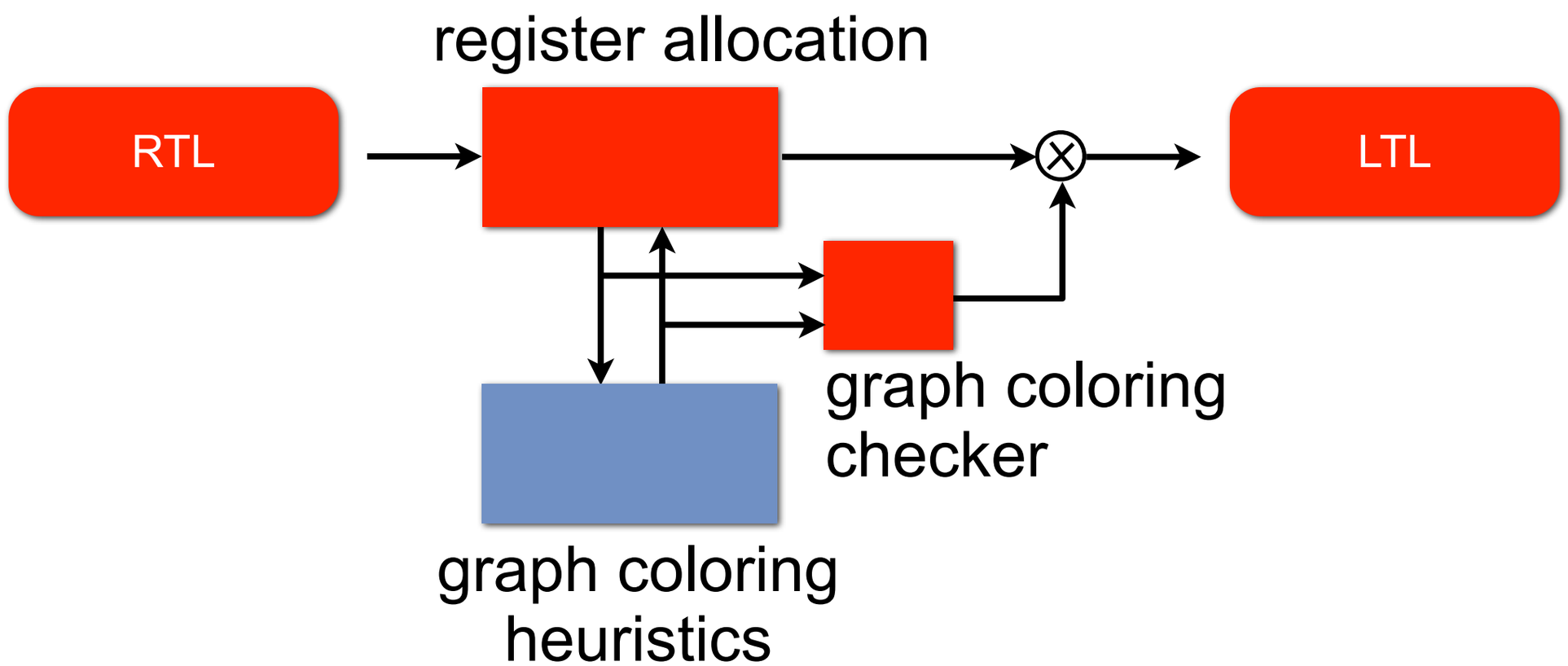
External solver with verified validation





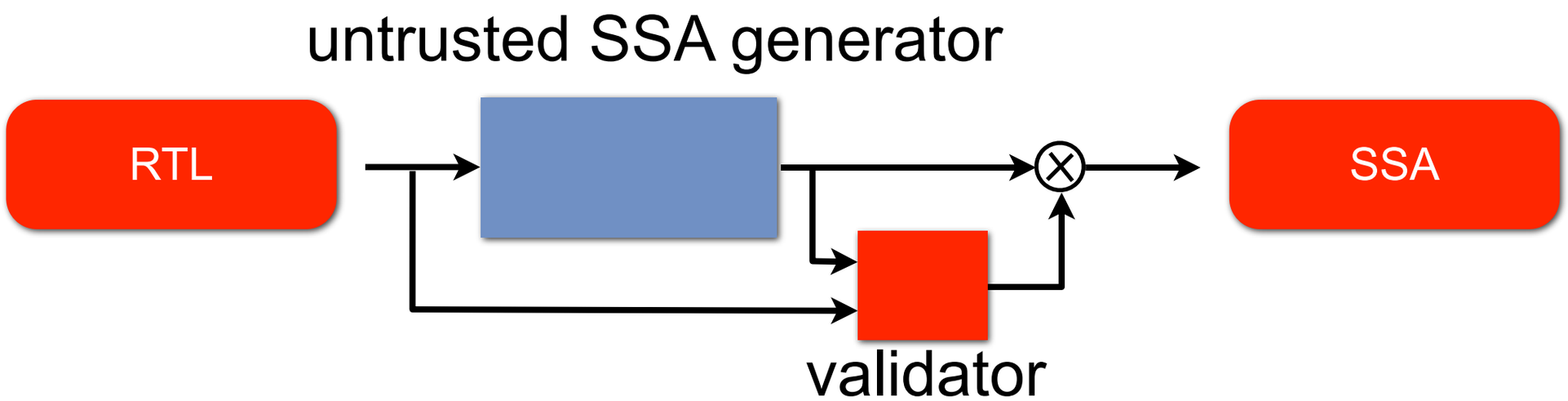
External solver with verified validation

Example: register allocation



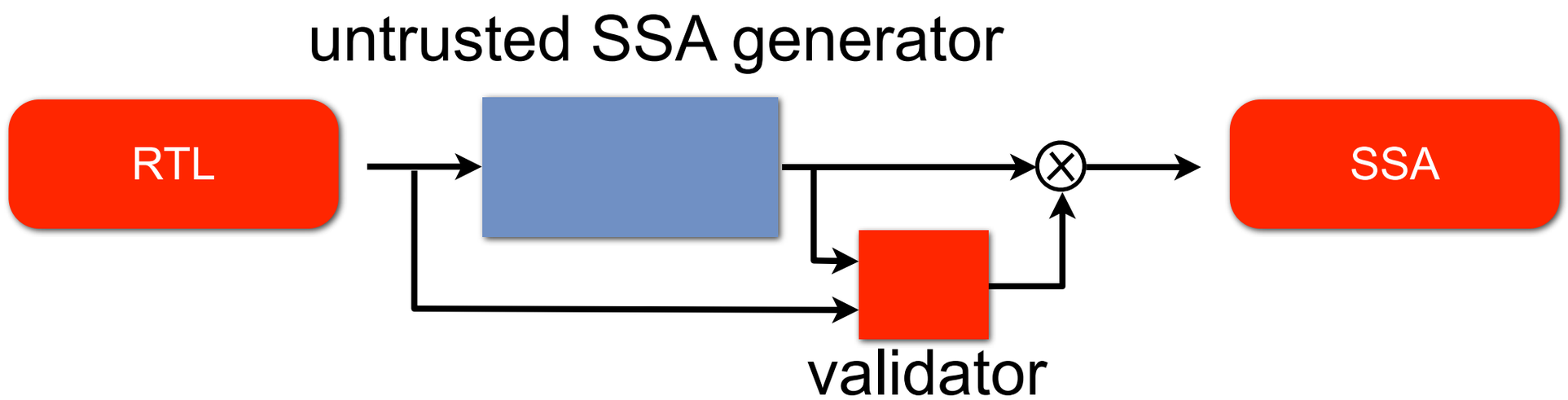
Verified translation validation

Example: SSA generation (in CompCert SSA extension)



Verified translation validation

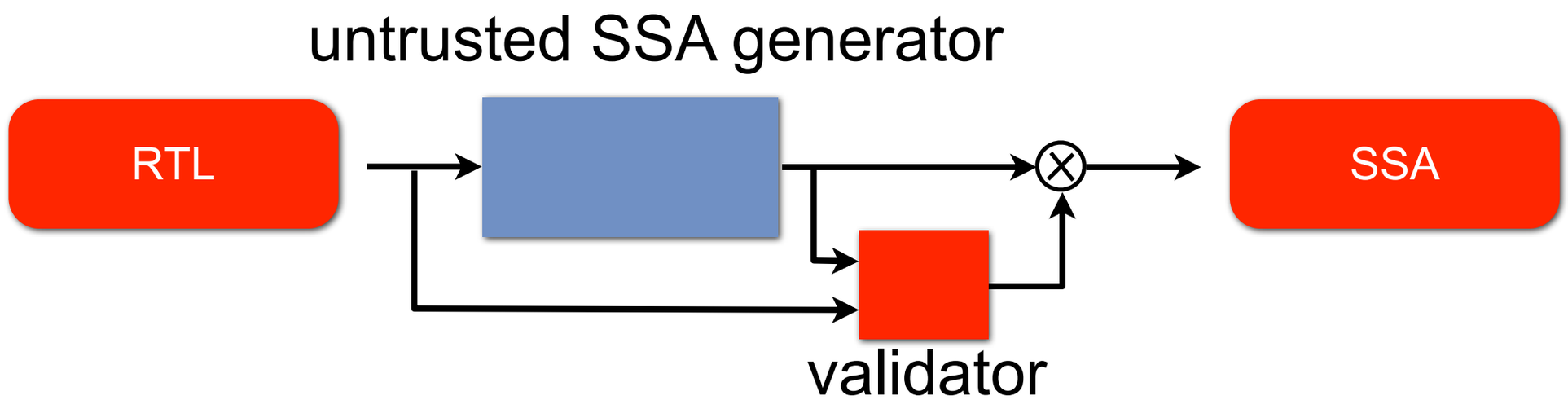
Example: SSA generation (in CompCert SSA extension)



The untrusted generator can rely on advanced graph algorithms as Lengauer and Tarjan's dominator tree construction and frontier dominance computation.

Verified translation validation

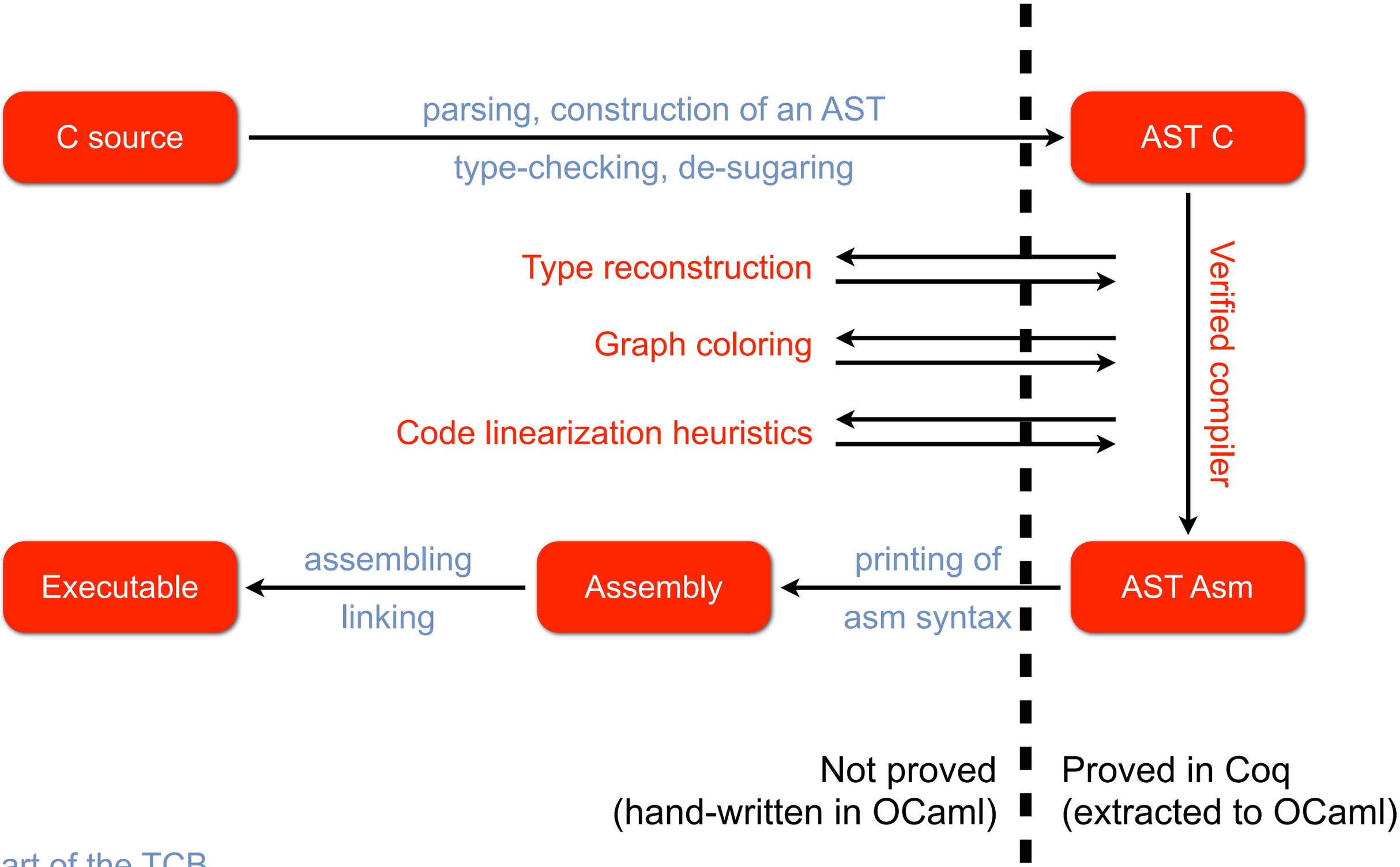
Example: SSA generation (in CompCert SSA extension)



The untrusted generator can rely on advanced graph algorithms as Lengauer and Tarjan's dominator tree construction and frontier dominance computation.

We prove the validator is *complete* with respect to this family of algorithms.

The whole CompCert compiler

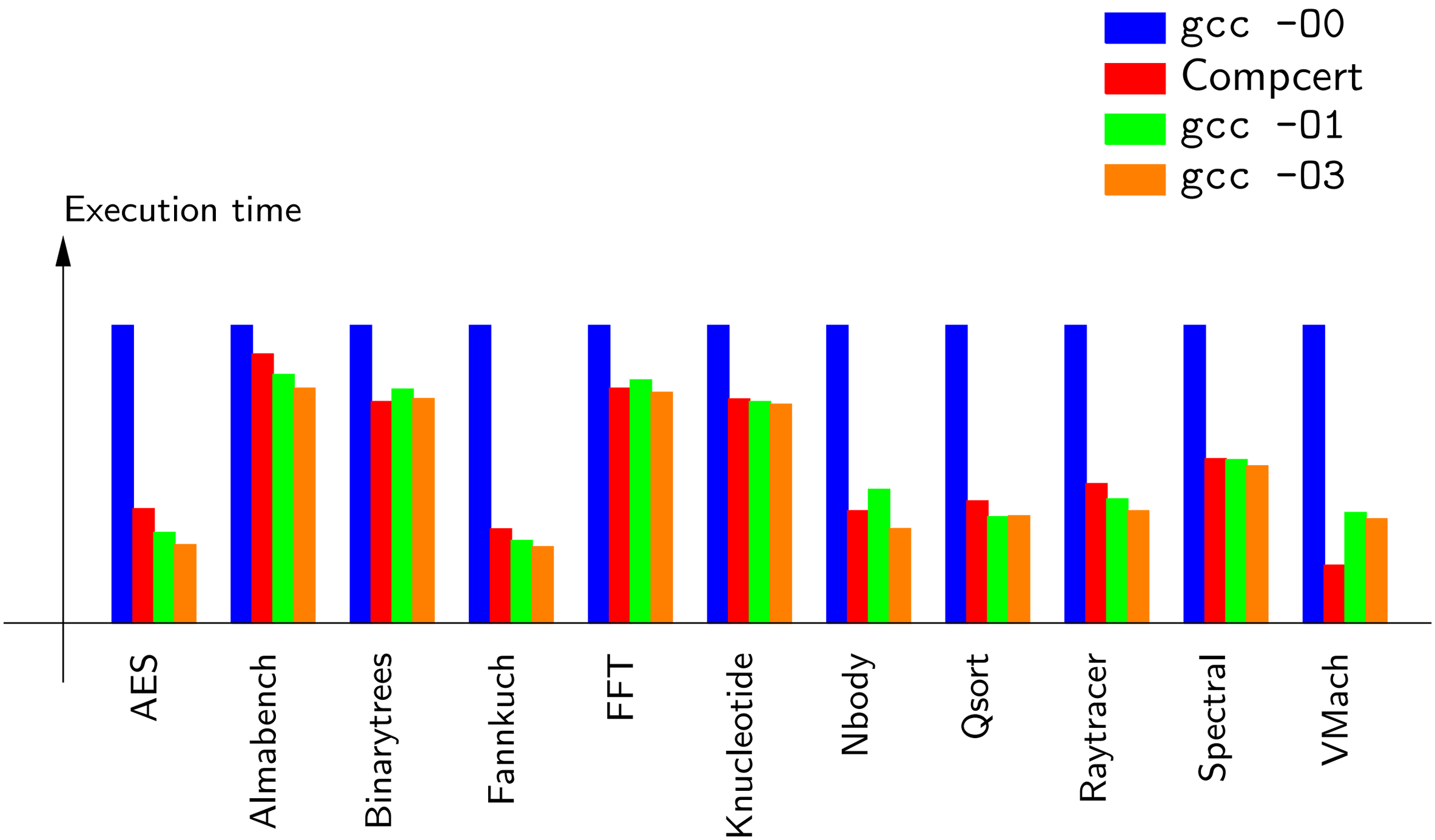


Part of the TCB

Not part of the TCB

Performance of generated code

(On a PowerPC G5 processor)



Conclusions

La vérification formelle d’un compilateur réaliste est *faisable.*

Encore de nombreux défis

- Réduire la base de confiance
  (e.g. analyse lexicale/syntaxique, assemblage, édition de liens).
- Plus d’optimisations
  (par exemple CompCert SSA).
- Cible pour des langages sources de plus haut niveau (Caml, Java)
- Parallélisme des langages sources/cibles
- Connexion avec vérificateur source *vérifié*
  (projet ANR Verasco sur un analyseur statique C vérifié)

Une partie de cette présentation est inspirée des ressources suivantes

Pour aller plus loin

Compilation

- Du langage à l'action: compilation et typage, Xavier Leroy, Collège de France 2008
  http://pauillac.inria.fr/~xleroy/talks/compilation_typage_College_de_France.pdf

Découvrir Coq

- Preuves de programme en Coq, Yves Bertot, Fuscia.info
  http://fuscia.inrialpes.fr/cours/coq/