

A Value Analysis for C Programs

Pascal Cuoq et al

CEA List

September 21, 2009

long n;
for (i = 0; i < n; i++)
 tmp2[i] = 0;
 of the

tmp2[0] = 1; /* (Nb1 - 1) else if (tmp1[0]) >= 1; /* (Nb1 - 1) tmp2[0] = (1 << (Nb1 - 1)) else tmp2[0] = tmp1[0]; /* Then the second part takes like the first one. tmp1[0] = 0; k = 5; k++) tmp1[0][k] += mc2[0][k] * tmp2[0][k]; /* The [i][j] coefficient of the matrix product MC2*TMP2, that is, *MC2*(TMP1) = MC2*(MC1*M1) = MC2*M1 *MC1. l = 1; tmp1[0][l] >= 1; /* Final rounding. tmp2[0][l] is now represented on 9 bits. *if (tmp1[0][l] < -255) tmp2[0][l] = -255; else if (tmp1[0][l] > 255) tmp2[0][l] = 255; else tmp2[0][l] = tmp1[0][l];



- ▶ A static analyzer is a plug-in

```

long ra
for (i = 0; i < N; i++)
  C[i] = i * i;
tmp2 =
  ...

```

```

tmp2[i][j] = 0; // Clear the matrix
for (k = 0; k < N; k++)
  tmp1[i][k] = C[i][k]; // Copy the first row of C into the first row of tmp1
for (k = 0; k < N; k++)
  tmp2[i][k] += tmp1[i][k] * C[k][j]; // The [i,j] coefficient of the matrix product MC2*TMP1, that is, *MC2*(TMP1) = MC2*(M1)*M1, is now represented on 9 bits.
Final rounding: tmp2[i][j] is now represented on 9 bits. *if (tmp1[i][j] < -256) tmp2[i][j] = -256; else if (tmp1[i][j] > 255) tmp2[i][j] = 255; else tmp2[i][j] = tmp1[i][j];

```



- ▶ A static analyzer is a plug-in
- ▶ Frama-C and plug-ins are written in OCaml
- ▶ Plug-ins can be distributed independently from Frama-C core
- ▶ No recompilation of core to integrate a new plug-in
- ▶ Plug-ins provide services to each other

```

long n;
for (i = 0; i < n; i++)
  tmp2 =
    ...

```

```

tmp2[i] = (i < (n-1) ? tmp1[i] : 0) + (i < (n-1) ? tmp1[i] : 0) + (i < (n-1) ? tmp1[i] : 0) + (i < (n-1) ? tmp1[i] : 0)
tmp1[i] = 0; k = 5; k = k + tmp1[i]; // The [i,j] coefficient of the matrix product MC2*TMP1, that is, *MC2*(TMP1) = MC2*(MC1*M1) = MC2*M1*MC1
i = i + tmp1[i] >> 1; // Final rounding: tmp2[i] is now represented on 9 bits: if (tmp1[i] < 256) tmp2[i] = 256; else if (tmp1[i] > 255) tmp2[i] = 255; else tmp2[i] = ...

```



- ▶ A static analyzer is a plug-in
- ▶ Frama-C and plug-ins are written in OCaml
- ▶ Plug-ins can be distributed independently from Frama-C core
- ▶ No recompilation of core to integrate a new plug-in
- ▶ Plug-ins provide services to each other

Think “The Gimp” or “Eclipse”

<http://frama-c.cea.fr/>



Initial focus of Frama-C: verification of critical embedded code.

Existing plug-ins are **correct**

long n
for 0 <=
C1; if (m
tmp2 =
of the

tmp2[j][i] = 0; if (i <= (n-1) && j <= (n-1)) tmp2[j][i] = (i <= (n-1) && j <= (n-1)) ? (tmp2[j][i] + tmp1[j][i]) : 0; Then the second part takes the first part and
tmp1[i][k] = 0; k = 0; k <= k-1; tmp1[i][k] += m2[i][k] * tmp2[k][j]; /* The [i,j] coefficient of the matrix product MC2*TMP2, that is, *MC2*(TMP1) = MC2*(M1)*M1 = MC2*M1*M1
i = 1; tmp1[i][i] >= 1; */ Final rounding: tmp2[i][i] is now represented on 9 bits: *if (tmp1[i][i] < -256) m2[i][i] = -256; else if (tmp1[i][i] > 255) m2[i][i] = 255; else m2[i][i] = tmp1[i][i];



Initial focus of Frama-C: verification of critical embedded code.

Existing plug-ins are **correct**

(each for its own definition of correct)

Abstract Interpretation, Weakest Precondition, Slicer

long no
for 0 <=
C1; if (a)
tmp2 =
of the

tmp2[0] = 1 << (n-1) / also if (tmp1[0]) >= 1 << (n-1) / if (tmp1[0]) = 1 << (n-1) / else tmp2[0] = tmp1[0]; /* Then the second part takes the first one
tmp1[0] = 0; k = 5; k--> tmp1[0] += mc2[0][k] * tmp2[k]; /* The [i][j] coefficient of the matrix product MC2*TMP2, that is, *MC2*(TMP1) = MC2*(MC1*M1) = MC2*M1 *MC1
i = 1; tmp1[0] >= 1; /* Final rounding: tmp2[0] is now represented on 9 bits: if (tmp1[0] < -256) tmp2[0] = -256; else if (tmp1[0] > 255) tmp2[0] = 255; else tmp2[0] = tmp1[0];



Initial focus of Frama-C: verification of critical embedded code.

Existing plug-ins are **correct**

(each for its own definition of correct)

Abstract Interpretation, Weakest Precondition, Slicer

Establish safety properties

long n
for 0 < k < n
c1[k] = m
tmp2 =
of the

tmp2[0] = 1; for (int i = 1; i < n; i++) tmp2[i] = 1 + tmp2[i-1]; // Then the second part takes the first part
tmp1[0] = 0; for (int k = 0; k < n; k++) tmp1[k] = m * tmp2[k]; // The [i,j] coefficient of the matrix product MC2 * TMP2, that is, *MC2*(TMP1) = MC2*(MC1 * M1) = MC2 * M1 * MC1
i = 1; tmp1[0] >= 1; // Final rounding: tmp2[0] is now represented on 3 bits: if (tmp1[0] < 256) m2[0] = 256; else m2[0] = 1;



Initial focus of Frama-C: verification of critical embedded code.

Existing plug-ins are **correct**
(each for its own definition of correct)

Abstract Interpretation, Weakest Precondition, Slicer

Establish safety properties
if codebase in the right C subset,...

```
long r;  
for (i = 0; i < N; i++)  
  r += i;  
tmp2 = r;  
return r;  
}
```

```
tmp2[i] += 1; // (tmp1 - i) also if (tmp1[i] >= 0) // (tmp1 - i) tmp2[i] += (tmp1[i] - i) // Then the second part equals the first one.  
tmp1[i][0] = 0; k = 5; k == tmp1[i][0] // m2[i][k] * tmp2[i][0] // The [i][j] coefficient of the matrix product MC2*TMP2, that is: *MC2*(TMP1) = MC2*(M1)*M1 = MC2*M1*M1  
l = 1; tmp1[i][0] >= 1; // Final rounding: tmp2[i][0] is now represented on 9 bits: if (tmp1[i][0] < -256) m2[i][0] = -256; else if (tmp1[i][0] > 255) m2[i][0] = 255; else m2[i][0] = tmp1[i][0];
```



Initial focus of Frama-C: verification of critical embedded code.

Existing plug-ins are **correct**

(each for its own definition of correct)

Abstract Interpretation, Weakest Precondition, Slicer

Establish safety properties

if codebase in the right C subset,...

Implementing a heuristic method as a Frama-C plug-in?



Initial focus of Frama-C: verification of critical embedded code.

Existing plug-ins are **correct**

(each for its own definition of correct)

Abstract Interpretation, Weakest Precondition, Slicer

Establish safety properties

if codebase in the right C subset,...

Implementing a heuristic method as a Frama-C plug-in?

Sure, go ahead...



Abstract Interpretation (comparable to Astrée or PolySpace)
 Emphasis on pointers, pointer casts,
 handling of statically allocated chained structures

long n;
 for (i = 0; i < n; i++)
 c[i] = 0;
 tmp2 =
 of the

tmp2[0] = 1; for (k = 1; k < n; k++) tmp1[k] += mc2[0][k] * tmp2[k]; /* The [i,j] coefficient of the matrix product MC2*TMP2, that is, *MC2*(TMP1) = MC2*(MC1*M1) = MC2*M1*MC1
 i = 1; tmp1[0] += 1; */ Final rounding: tmp2[0] is now represented on 9 bits: if (tmp1[0] < -255) tmp2[0] = -255; else if (tmp1[0] > 255) tmp2[0] = 255; else tmp2[0] = tmp1[0];



Abstract Interpretation (comparable to Astrée or PolySpace)
 Emphasis on pointers, pointer casts,
 handling of statically allocated chained structures

Results recorded for each statement for use by other plug-ins.
 100 000 lines of embedded code — 10GiB of memory (peak)

long n;
 for (i = 0; i < n; i++)
 c[i] = 0;
 tmp2 =
 of the

tmp2[i][j] = 0; for (k = 0; k < n; k++) tmp1[i][k] = m2[0][k] * tmp2[k][j]; /* The [i][j] coefficient of the matrix product MC2*TMP2, that is, *MC2*(TMP1) = MC2*(MC1*TMP1) = MC2*TMP1 */
 Final rounding: tmp2[i][j] is now represented on 3 bits: if (tmp1[i][j] < -255) m2[i][j] = -255; else if (tmp1[i][j] > 255) m2[i][j] = 255; else tmp2[i][j] =



?

long ra
for 0 ->
C1); if (m
tmp2 =
se of the

tmp2[0] = 1 << (N81 - 1); else if (tmp1[0]) >= 1 << (N81 - 1); tmp2[0] = (1 << (N81 - 1) - 1); else tmp2[0] = tmp1[0]; /* Then the second part looks like the first one: */
tmp1[0][k] = 0; k <-> k++) tmp1[0][k] += m2[0][k] * tmp2[0][k]; /* The [i][j] coefficient of the matrix product MC2*TMP2, that is, *MC2*(TMP1) = MC2*(MC1*M1) = MC2*M1*MC1
i <- 1; tmp1[0][i] >= -1; /* Final rounding: tmp2[0][i] is now represented on 9 bits: *if (tmp1[0][i] < -256/m2[0][i]) = -256; else if (tmp1[0][i] > 255/m2[0][i]) = 255; else tmp2[0][i] = tmp1[0][i];



Attention C compilers authors:
Nobody except you cares about your speed benchmarks.

if (long ra
for (i = 0
c1); if (m
tmp2 =
e of the

tmp2[i][j] = 0; if (i < (nbl - 1)) else if (tmp1[i][j] >= 0) if (i < (nbl - 1)) tmp2[i][j] = (i < (nbl - 1)) ? 0 : tmp1[i][j]; /* Then the second part takes like the first one. */
tmp1[i][j] = 0; k = 5; k <= 8; k <= 8; tmp1[i][j] += mc2[i][k] * tmp2[k][j]; /* The [i][j] coefficient of the matrix product MC2*TMP2, that is, *MC2*(TMP1) = MC2*(MC1*M1) = MC2*M1*MC1
i = 1; tmp1[i][j] >= 1; /* Final rounding: tmp2[i][j] is now represented on 9 bits. *if (tmp1[i][j] < -256) m2[i][j] = -256; else if (tmp1[i][j] > 255) m2[i][j] = 255; else m2[i][j] = tmp1[i][j];



Attention C compilers authors:

Nobody except you cares about your speed benchmarks.

Current trend in the C compilation world:

take advantage of any little bit of undefinedness in the standard.

- ▶ signed arithmetic overflows not giving 2-complement results
- ▶ `char*` assumed not to alias with an `int*`
- ▶ ...

```
long n;
for (i = 0; i < n; i++)
    tmp2[i] =
        of the
```

```
tmp2[0][i] = (i < (n1 - 1) ? a[i + tmp1[0]] : 0; i <= (n1 - 1) ? tmp2[0][i] : (i <= (n1 - 1) ? a[i + tmp2[0]] : tmp1[0]); // Then the second part takes the first part
tmp1[0][i] = 0; k = 5; k--); tmp1[0][i] = mc2[0][k] * tmp2[0][i]; // The [i][j] coefficient of the matrix product MC2*TMP2, that is, *MC2*(TMP1) = MC2*(MC1*M1) = MC2*(M1*M1)
i = 1; tmp1[0][i] >= 1; // Final operand tmp2[0][i] is now represented on 3 bits: if (tmp1[0][i] < 256) m2[0][i] = 255; else if (tmp1[0][i] > 255) m2[0][i] = 255; else m2[0][i] =
```





long n
for n
C1) if
tmp2
of the
tmp2[j] = 1 && (n&l-1) else if (tmp1[j]) >= 1 && (n&l-1) ? tmp2[j] = 1 : tmp1[j]; /* Then the second part looks like the first part: */
tmp1[j+k] = 0; k = k+1; tmp1[j+k] += mc2[k][k] * tmp2[k][j]; /* The [j] coefficient of the matrix product MC2*TMP2, that is: *MC2*(TMP1) = MC2*(MC1*M1) = MC2*M1 *MC1
l = 1; tmp1[j+l] >>= 1; /* Final rounding: tmp2[j+l] is now represented on 3 bits: if (tmp1[j+l] <= 255) m2[j+l] = 255; else if (tmp1[j+l] > 255) m2[j+l] = tmp1[j+l];