

Towards combining static and dynamic analyses within Frama-C

Julien Signoles

CEA LIST

Software Safety Labs

Hi-Lite Meeting

May 2012, the 29th



HI-LITE

Simplifying the use of formal methods



1. what's new in E-ACSL
2. property statuses for combining analyses

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

tmp2[i] = 0; if (i < (N1 - 1)) else if (tmp1[i] >= 0) tmp2[i] = (i < (N1 - 1)) ? abs(tmp1[i]) + tmp1[i] : 1; /* Then the second part takes like the first one. */
tmp1[i][k] = 0; k = 0; k => tmp1[i][k] += mc2[i][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[i][i] >= 1; /* Final rounding: tmp2[i] is now represented on 9 bits. */ if (tmp1[i][i] < -256) m2[i] = -256; else if (tmp1[i][i] > 255) m2[i] = 255; else m2[i] = tmp1[i][i];



Executable Ansi/ISO C Specification Language

<http://frama-c.com/eacsl>

What is it?

- ▶ executable subset of ACSL
- ▶ preserve ACSL semantics as much as possible
- ▶ compatible with ALFA as much as possible

Which goals?

- ▶ runtime assertion checking
- ▶ usable by dynamic analyses tools
- ▶ usable by static verification tools like Frama-C plug-ins
- ▶ verification of mixed ADA/C programs



- ▶ takes an annotated C program as input
- ▶ checks that each annotation belongs to E-ACSL
- ▶ returns a new C program
- ▶ equivalent to the input
- ▶ each annotation is converted into new C statements
- ▶ including (at least) one guard
- ▶ which fails at runtime if the annotation is wrong



- ▶ new release of E-ACSL Reference Manual (version 1.5-4)
- ▶ first release of the E-ACSL plug-in (version 0.1)
 - ▶ compatible with Frama-C Nitrogen
 - ▶ compatible with E-ACSL 1.5-4
 - ▶ open source (LGPL v2.1)
 - ▶ webpage: <http://frama-c.com/eacsl>
 - ▶ what's currently implemented detailed in frama-c.com/download/e-acsl/e-acsl-implementation.pdf
- ▶ submitted publication (short paper):
M. Delahaye, N. Kosmatov and J. Signoles. Towards a Common Specification Language for Static and Dynamic Analysis of C Programs.



implemented

- ▶ C types
- ▶ integer
- ▶ boolean
- ▶ implicit coercions

not yet implemented

- ▶ real

No change since the last meeting



implemented

- ▶ integer constants
- ▶ C left values
- ▶ arithmetic operators
- ▶ casts
- ▶ address &
- ▶ sizeof
- ▶ alignof
- ▶ `\null`
- ▶ `\at`
- ▶ `\result`
- ▶ boolean operators
- ▶ conditionals `_?:_`

under implementation

- ▶ `\base_addr`
- ▶ `\offset`
- ▶ `\block_length`

not yet implemented

- ▶ `\true` and `\false`
- ▶ bitwise operators
- ▶ let binding
- ▶ `typeof`
- ▶ t-sets



implemented

- ▶ `\true` and `\false`
- ▶ relations (`==`, `<=`, ...)
- ▶ lazy conjunction `&&`
- ▶ lazy disjunction `||`
- ▶ lazy implication `==>`
- ▶ negation `!`
- ▶ equivalence `<==>`
- ▶ quantifications over integral types
- ▶ `\at`
- ▶ conditionals `_?_:_`

under implementation

- ▶ `\valid et al.`
- ▶ `\initialized`

not yet implemented

- ▶ other quantifications
- ▶ exclusive or `^^`
- ▶ let bindings



implemented

- ▶ assertions
- ▶ function contracts
- ▶ statement contracts
- ▶ loop invariants

not yet implemented

- ▶ behavior-specific annotations
- ▶ loop variants
- ▶ loop assigns
- ▶ global annotations



implemented

- ▶ assumes
- ▶ requires
- ▶ ensures

not yet implemented

- ▶ assigns
- ▶ decreases
- ▶ abrupt termination
- ▶ complete behaviors
- ▶ disjoint behaviors

No change since the last meeting

```

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

```

```

tmp2[0] = 1; // (n-1) else if (tmp1[0]) >= 1; // (n-1) // else tmp2[0] = tmp1[0]; // Then the second part looks like 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];

```



- ▶ E-ACSL 0.1 uses **GMP integers** to **correctly** translate integers
- ▶ **heavy encoding**

```

/*@ assert y != 0; */
mpz_t __e_acsl_y; // declare additional variables
mpz_t __e_acsl;
int __e_acsl_ne;
mpz_init_set_si(__e_acsl_y, y); // allocate GMP's
mpz_init_set_si(__e_acsl, 0);
// do the comparison
__e_acsl_ne = mpz_cmp(__e_acsl_y, __e_acsl);
e_acsl_assert(__e_acsl_ne != 0,
              "Assertion", "(y != 0)", 2);

// deallocate GMP's
mpz_clear(__e_acsl_y);
mpz_clear(__e_acsl);
  
```



- ▶ for each integer n , a type system infers an interval $[a; b]$ containing n as small as possible
- ▶ if there are C types containing $[a; b]$, use the smallest one.
- ▶ otherwise use GMP's as before
- ▶ the previous example becomes simple

```
/*@ assert y != 0; */
e_acsl_assert(y != 0, "Assertion", "(y != 0)", 2);
```

- ▶ in practice, no more GMP's on the E-ACSL's test suite
- ▶ will be part of E-ACSL 0.2 (July? September?)



- ▶ how to ensure the **safety of an annotated program**
- ▶ by using several **customizable analyzers**
- ▶ based on **different techniques?**
- ▶ a “**consolidation algorithm**” merges all the results coming from the different analyzers with their different configurations

Demo!



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

tmp2[i] = 1 << (n-i) / 2; else tmp2[i] = 0; // The [i] coefficient of the matrix product MC2*TMP2, that is *MC2*(TMP1) = MC2*(MC1*M1) = MC2*M1 * MC1
i = 1; tmp2[i] >>= 1; // Final rounding, tmp2[i] is now represented on 9 bits. if (tmp2[i] < -256) tmp2[i] = -256; else if (tmp2[i] > 255) tmp2[i] = 255; else tmp2[i] = tmp2[i];

► the consolidation algorithm is correct












if each analyzer is correct, then the algorithm returns “Valid” (resp. “Invalid”) for a valid (resp. invalid) property. It returns “Inconsistent” if there are both a proof of validity and invalidity.

► the consolidation algorithm is complete

if each analyzer is correct and indicates the right hypotheses, and if one analyzer does not indicate “Dont know” under recursively valid hypotheses, then the computed status is either “Valid” or “Invalid”.



long n
t for 0
ct; if
tmp2
of the
tmp2[0]
= (n-1) * tmp1[0] + tmp1[1];
tmp2[1] = (n-1) * tmp1[1] + tmp1[2];
tmp2[2] = (n-1) * tmp1[2] + tmp1[3];
tmp2[3] = (n-1) * tmp1[3] + tmp1[4];
tmp2[4] = (n-1) * tmp1[4] + tmp1[5];
tmp2[5] = (n-1) * tmp1[5] + tmp1[6];
tmp2[6] = (n-1) * tmp1[6] + tmp1[7];
tmp2[7] = (n-1) * tmp1[7] + tmp1[8];
tmp2[8] = (n-1) * tmp1[8] + tmp1[9];
tmp2[9] = (n-1) * tmp1[9] + tmp1[10];
tmp2[10] = (n-1) * tmp1[10] + tmp1[11];
tmp2[11] = (n-1) * tmp1[11] + tmp1[12];
tmp2[12] = (n-1) * tmp1[12] + tmp1[13];
tmp2[13] = (n-1) * tmp1[13] + tmp1[14];
tmp2[14] = (n-1) * tmp1[14] + tmp1[15];
tmp2[15] = (n-1) * tmp1[15] + tmp1[16];
tmp2[16] = (n-1) * tmp1[16] + tmp1[17];
tmp2[17] = (n-1) * tmp1[17] + tmp1[18];
tmp2[18] = (n-1) * tmp1[18] + tmp1[19];
tmp2[19] = (n-1) * tmp1[19] + tmp1[20];
tmp2[20] = (n-1) * tmp1[20] + tmp1[21];
tmp2[21] = (n-1) * tmp1[21] + tmp1[22];
tmp2[22] = (n-1) * tmp1[22] + tmp1[23];
tmp2[23] = (n-1) * tmp1[23] + tmp1[24];
tmp2[24] = (n-1) * tmp1[24] + tmp1[25];
tmp2[25] = (n-1) * tmp1[25] + tmp1[26];
tmp2[26] = (n-1) * tmp1[26] + tmp1[27];
tmp2[27] = (n-1) * tmp1[27] + tmp1[28];
tmp2[28] = (n-1) * tmp1[28] + tmp1[29];
tmp2[29] = (n-1) * tmp1[29] + tmp1[30];
tmp2[30] = (n-1) * tmp1[30] + tmp1[31];
tmp2[31] = (n-1) * tmp1[31] + tmp1[32];
tmp2[32] = (n-1) * tmp1[32] + tmp1[33];
tmp2[33] = (n-1) * tmp1[33] + tmp1[34];
tmp2[34] = (n-1) * tmp1[34] + tmp1[35];
tmp2[35] = (n-1) * tmp1[35] + tmp1[36];
tmp2[36] = (n-1) * tmp1[36] + tmp1[37];
tmp2[37] = (n-1) * tmp1[37] + tmp1[38];
tmp2[38] = (n-1) * tmp1[38] + tmp1[39];
tmp2[39] = (n-1) * tmp1[39] + tmp1[40];
tmp2[40] = (n-1) * tmp1[40] + tmp1[41];
tmp2[41] = (n-1) * tmp1[41] + tmp1[42];
tmp2[42] = (n-1) * tmp1[42] + tmp1[43];
tmp2[43] = (n-1) * tmp1[43] + tmp1[44];
tmp2[44] = (n-1) * tmp1[44] + tmp1[45];
tmp2[45] = (n-1) * tmp1[45] + tmp1[46];
tmp2[46] = (n-1) * tmp1[46] + tmp1[47];
tmp2[47] = (n-1) * tmp1[47] + tmp1[48];
tmp2[48] = (n-1) * tmp1[48] + tmp1[49];
tmp2[49] = (n-1) * tmp1[49] + tmp1[50];
tmp2[50] = (n-1) * tmp1[50] + tmp1[51];
tmp2[51] = (n-1) * tmp1[51] + tmp1[52];
tmp2[52] = (n-1) * tmp1[52] + tmp1[53];
tmp2[53] = (n-1) * tmp1[53] + tmp1[54];
tmp2[54] = (n-1) * tmp1[54] + tmp1[55];
tmp2[55] = (n-1) * tmp1[55] + tmp1[56];
tmp2[56] = (n-1) * tmp1[56] + tmp1[57];
tmp2[57] = (n-1) * tmp1[57] + tmp1[58];
tmp2[58] = (n-1) * tmp1[58] + tmp1[59];
tmp2[59] = (n-1) * tmp1[59] + tmp1[60];
tmp2[60] = (n-1) * tmp1[60] + tmp1[61];
tmp2[61] = (n-1) * tmp1[61] + tmp1[62];
tmp2[62] = (n-1) * tmp1[62] + tmp1[63];
tmp2[63] = (n-1) * tmp1[63] + tmp1[64];
tmp2[64] = (n-1) * tmp1[64] + tmp1[65];
tmp2[65] = (n-1) * tmp1[65] + tmp1[66];
tmp2[66] = (n-1) * tmp1[66] + tmp1[67];
tmp2[67] = (n-1) * tmp1[67] + tmp1[68];
tmp2[68] = (n-1) * tmp1[68] + tmp1[69];
tmp2[69] = (n-1) * tmp1[69] + tmp1[70];
tmp2[70] = (n-1) * tmp1[70] + tmp1[71];
tmp2[71] = (n-1) * tmp1[71] + tmp1[72];
tmp2[72] = (n-1) * tmp1[72] + tmp1[73];
tmp2[73] = (n-1) * tmp1[73] + tmp1[74];
tmp2[74] = (n-1) * tmp1[74] + tmp1[75];
tmp2[75] = (n-1) * tmp1[75] + tmp1[76];
tmp2[76] = (n-1) * tmp1[76] + tmp1[77];
tmp2[77] = (n-1) * tmp1[77] + tmp1[78];
tmp2[78] = (n-1) * tmp1[78] + tmp1[79];
tmp2[79] = (n-1) * tmp1[79] + tmp1[80];
tmp2[80] = (n-1) * tmp1[80] + tmp1[81];
tmp2[81] = (n-1) * tmp1[81] + tmp1[82];
tmp2[82] = (n-1) * tmp1[82] + tmp1[83];
tmp2[83] = (n-1) * tmp1[83] + tmp1[84];
tmp2[84] = (n-1) * tmp1[84] + tmp1[85];
tmp2[85] = (n-1) * tmp1[85] + tmp1[86];
tmp2[86] = (n-1) * tmp1[86] + tmp1[87];
tmp2[87] = (n-1) * tmp1[87] + tmp1[88];
tmp2[88] = (n-1) * tmp1[88] + tmp1[89];
tmp2[89] = (n-1) * tmp1[89] + tmp1[90];
tmp2[90] = (n-1) * tmp1[90] + tmp1[91];
tmp2[91] = (n-1) * tmp1[91] + tmp1[92];
tmp2[92] = (n-1) * tmp1[92] + tmp1[93];
tmp2[93] = (n-1) * tmp1[93] + tmp1[94];
tmp2[94] = (n-1) * tmp1[94] + tmp1[95];
tmp2[95] = (n-1) * tmp1[95] + tmp1[96];
tmp2[96] = (n-1) * tmp1[96] + tmp1[97];
tmp2[97] = (n-1) * tmp1[97] + tmp1[98];
tmp2[98] = (n-1) * tmp1[98] + tmp1[99];
tmp2[99] = (n-1) * tmp1[99] + tmp1[100];

-  unknown
-  valid
-  invalid
-  never tried
-  considered valid (e.g. precondition of main)
-  valid under hypotheses
-  invalid under hypotheses
-  unknown but dead
-  valid but dead
-  invalid but dead
-  inconsistent



► 2 submitted articles:

- L. Correnson and J. Signoles. Combining Analyses for C Program Verification.
- P. Cuoq, F. Kirchner, N. Kosmatov, V. Prevosto, J. Signoles and B. Yakobowski. Frama-C: A Software Analysis Perspective.



- ▶ release of **Frama-C Oxygen** in July
- ▶ release of **E-ACSL v0.2** in July or September.
 - ▶ will include the **new type system**
- ▶ integrating and testing the **executable memory model**
- ▶ handling **runtime errors** in annotations
- ▶ going further into **collaboration** between static and dynamic analyses
- ▶ writing a **full paper** on E-ACSL
- ▶ what about **mix C/ADA programs**?



Any questions?

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

tmp2[ji] = 0; if (i < (Nbr - 1)) else if (tmp1[ji] >= 0) tmp2[ji] = (1 << (Nbr - 1)) - 1; else tmp2[ji] = tmp1[ji]; /* Then the second part takes like the first one: */
tmp1[0][k] = 0; k = 0; k <= 5; 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*(M1*M1) = MC2*M1*M1
i = 1; tmp1[0][i] >= -1; */ Final rounding: tmp2[0][i] is now represented on 9 bits: *if (tmp1[0][i] < -256) tmp2[0][i] = -256; else if (tmp1[0][i] > 255) tmp2[0][i] = 255; else tmp2[0][i] = tmp1[0][i];

