První kroky s Frama-C

Jan Tušil

16. 3. 2015

Obsah

Materiály k prezentaci

- `https://github.com/h0nzZik/Frama-C_Examples`

Jak získat?

- Balík frama-c v repozitářích Fedory i Debianu
- Alt-Ergo (theorem prover) tamtéž (hodí se pro WP plugin)

Představení

- Modulární platforma pro statickou analýzu C kódu

Představení

- Modulární platforma pro statickou analýzu C kódu
- Jádro

Představení

- Modulární platforma pro statickou analýzu C kódu
- Jádro

- Pluginy

Představení

- Modulární platforma pro statickou analýzu C kódu
- Jádro
  - Parsování, preprocessing, „linkování" a normalizace kódu
- Pluginy

Představení

- Modulární platforma pro statickou analýzu C kódu
- Jádro
  - Parsování, preprocessing, „linkování" a normalizace kódu
  - Ukládání/načítání session
- Pluginy

Představení

- Modulární platforma pro statickou analýzu C kódu
- Jádro
  - Parsování, preprocessing, „linkování" a normalizace kódu
  - Ukládání/načítání session
  - Uchovávání výsledků práce pluginů
- Pluginy

Představení

- Modulární platforma pro statickou analýzu C kódu
- Jádro
  - Parsování, preprocessing, „linkování" a normalizace kódu
  - Ukládání/načítání session
  - Uchovávání výsledků práce pluginů
  - Manipulace s AST
- Pluginy

Představení

- Modulární platforma pro statickou analýzu C kódu
- Jádro
  - Parsování, preprocessing, „linkování" a normalizace kódu
  - Ukládání/načítání session
  - Uchovávání výsledků práce pluginů
  - Manipulace s AST
- Pluginy
  - Analýza závislostí

Představení

- Modulární platforma pro statickou analýzu C kódu
- Jádro
  - Parsování, preprocessing, „linkování" a normalizace kódu
  - Ukládání/načítání session
  - Uchovávání výsledků práce pluginů
  - Manipulace s AST
- Pluginy
  - Analýza závislostí
  - Abstraktní interpretace

Představení

- Modulární platforma pro statickou analýzu C kódu
- Jádro
  - Parsování, preprocessing, „linkování" a normalizace kódu
  - Ukládání/načítání session
  - Uchovávání výsledků práce pluginů
  - Manipulace s AST
- Pluginy
  - Analýza závislostí
  - Abstraktní interpretace
  - Deduktivní verifikace

Představení

- Modulární platforma pro statickou analýzu C kódu
- Jádro
  - Parsování, preprocessing, „linkování" a normalizace kódu
  - Ukládání/načítání session
  - Uchovávání výsledků práce pluginů
  - Manipulace s AST
- Pluginy
  - Analýza závislostí
  - Abstraktní interpretace
  - Deduktivní verifikace
- Některé pluginy umožňují ověřování programu vůči specifikaci

Specifikace?

- Možné specifikace:

Specifikace?

- Možné specifikace:
  - Za běhu programu nenastane chyba

Specifikace?

- Možné specifikace:
  - Za běhu programu nenastane chyba (?)

Specifikace?

- Možné specifikace:
  - Za běhu programu nenastane chyba (?)
  - Chování programu je definované (dle standardu jazyka).

Specifikace?

- Možné specifikace:
  - Za běhu programu nenastane chyba (?)
  - Chování programu je definované (dle standardu jazyka).
  - Funkce nemodifikuje globální proměnné

Specifikace?

- Možné specifikace:
  - Za běhu programu nenastane chyba (?)
  - Chování programu je definované (dle standardu jazyka).
  - Funkce nemodifikuje globální proměnné
  - Funkce zachovává vlastnost používané datové struktury

Specifikace?

- Možné specifikace:
  - Za běhu programu nenastane chyba (?)
  - Chování programu je definované (dle standardu jazyka).
  - Funkce nemodifikuje globální proměnné
  - Funkce zachovává vlastnost používané datové struktury
- Potřeba jazyka pro zápis specifikace.

Jazyk ACSL

ACSL - Co je to za jazyk?

- ANSI / ISO C Specification language
- Vznikl pro potřeby Frama-C
- Vestavěný v komentářích C kódu.
- Asserty
- Invarianty
- Funkční kontrakty (DbC)
- http://frama-c.com/acsl.html

ACSL - vyjadřovací schopnosti

- C operátory a datové typy
- Matematické datové typy
- Prvořádová logika (s rozšířeními)

Asserty

```
int x = 17;
/*@ assert x > 5 */
```

- Základní specifikační jednotka
- Tvrzení o stavu programu v daném bodě.
- (ACSL specifikace zabudována v komentáři)

Kvantifikátory

```
int array[4] = {-15, 3, 17, 104};
/*@
assert \forall integer i, j;
0 <= i <= j < 4 ==> array[i] <= array[j];
*/
```

- Vázaná proměnná má daný typ
- Typ může být uživatelem definovaný (typedef, struct)
- `integer` označuje (matematické) celé číslo

Pointery

```
int array[4] = {-15, 3, 17, 104};
/*@ assert \valid(array + (0..3)); */
```

- Predikát \valid bere množinu termů
- 0..3 označuje množinu $\{0, 1, 2, 3\}$
- Význam: výrazy $\{array + 0, \ldots, array + 3\}$ jsou platné ukazatele
- Obvyklá pointer aritmetika

Uživatelské predikáty

```
/*@
predicate is_sorted ( int *array, integer len ) =
\forall integer i, j; 0 <= i <= j < len
==> array[i] <= array[j];
*/
```

- Predikát lze využít později

```
/*@ assert is_sorted ( array, 4 ); */
```

Invarianty smyček

```
int arr[7];
[ ... ]
/*@ loop invariant is_sorted(arr, i) */
for ( int i = 0; i < 7; i++ ) {
        [ ... ]
}
```

Invarianty smyček

```
int arr[7];
[ ... ]
/*@ loop invariant is_sorted(arr, i) */
for ( int i = 0; i < 7; i++ ) {
        [ ... ]
}
```

- Invariant musí být platný při (prvním) vstupu do smyčky

Invarianty smyček

```
int arr[7];
[ ... ]
/*@ loop invariant is_sorted(arr, i) */
for ( int i = 0; i < 7; i++ ) {
        [ ... ]
}
```

- Invariant musí být platný při (prvním) vstupu do smyčky
- Každý průchod smyčkou musí invariant zachovat

Funkční kontrakty

```
/*@
requires \valid ( array + (0 .. ( len-1 ) ) );
ensures is_sorted ( array, len );
assigns \nothing;
 */
void sort ( int *array, size_t len);
```

Funkční kontrakty

```
/*@
requires \valid ( array + (0 .. ( len-1 ) ) );
ensures is_sorted ( array, len );
assigns \nothing;
 */
void sort ( int *array, size_t len);
```

- Co funkce požaduje?

Funkční kontrakty

```
/*@
requires \valid ( array + (0 .. ( len-1 ) ) );
ensures is_sorted ( array, len );
assigns \nothing;
 */
void sort ( int *array, size_t len);
```

- Co funkce požaduje?
- Co funkce garantuje?

Funkční kontrakty

```
/*@
requires \valid ( array + (0 .. ( len-1 ) ) );
ensures is_sorted ( array, len );
assigns \nothing;
 */
void sort ( int *array, size_t len);
```

- Co funkce požaduje?
- Co funkce garantuje?
- Co funkce zachovává / modifikuje?

Funkční kontrakty

```
/*@
requires \valid ( array + (0 .. ( len-1 ) ) );
ensures is_sorted ( array, len );
assigns \nothing;
 */
void sort ( int *array, size_t len);
```

- Co funkce požaduje?
- Co funkce garantuje?
- Co funkce zachovává / modifikuje?
- Paradigma "Design by Contract"

Funkční kontrakty

```
/*@
requires \valid ( array + (0 .. ( len-1 ) ) );
ensures is_sorted ( array, len );
assigns \nothing;
 */
void sort ( int *array, size_t len);
```

- Co funkce požaduje?
- Co funkce garantuje?
- Co funkce zachovává / modifikuje?
- Paradigma "Design by Contract"
- Kde je problém?

Použití Frama-C

Jednoduché použití

- Soubor hello.c je v adresáři 01_hello
- `$ frama-c hello.c -val`

Jednoduché použití

- Soubor hello.c je v adresáři 01_hello
- `$ frama-c hello.c -val`
- Spustí Value plugin - analýzu možných hodnot proměnných.

Jednoduché použití

- Soubor hello.c je v adresáři 01_hello
- `$ frama-c hello.c -val`
- Spustí Value plugin - analýzu možných hodnot proměnných.
- Jak ověřit větší projekt z více souborů?

Jednoduché použití

- Soubor hello.c je v adresáři 01_hello
- `$ frama-c hello.c -val`
- Spustí Value plugin - analýzu možných hodnot proměnných.
- Jak ověřit větší projekt z více souborů?
- Všechny je vypíšeme na řádce.

Na co je preprocesor v C?

Na co je preprocesor v C?

- Makra a náhrady textu

Na co je preprocesor v C?

- Makra a náhrady textu
- Vkládání (hlavičkových) souborů

Preprocessing ve Frama-C

- Výchozí: `gcc -C -E -I`
- Možno předefinovat přepínačem `-cpp-command`

Preprocessing ve Frama-C

- Výchozí: `gcc -C -E -I`
- Možno předefinovat přepínačem `-cpp-command`
- Frama-C umí předzpracovat i anotace (s GCC)
- Rozpracovaný projekt je možné uložit a znovu načíst
- Viz soubor build.mk

Drobnosti

- RTE plugin generuje anotace pro obvyklé runtime chyby
- Kombinace RTE + Value může prokázat absenci runtime chyb
- Uložený projekt je možné načíst do programu frama-c-gui

Value plugin

Value plugin - Principy

- Abstraktní interpretace
- Počítá variační domény proměnných
- Overaproximace - dokazuje korektnost

Variační domény

- Množina možných hodnot, které může obsahovat daná proměnná.
- Různé způsoby zápisu
  - Výčtem $\{2, 12, 22, 32, 42\}$
  - Intervalem $[2 \ldots 42], 2\%10$

Ukázka (01_hello/hello.c) - 1

```
int main(int argc, char **argv)
{
  int __retres;
  int array[25];
  int idx;
  /*@ assert rte: index_bound: 0 ≤ argc; */
  /*@ assert rte: index_bound: argc < 25; */
  array[argc] = 7;
  idx = Frama_C_interval(-1,26);
  idx /= 2;
  /*@ assert rte: index_bound: 0 ≤ idx; */
  /*@ assert rte: index_bound: idx < 25; */
  array[idx] = 0x1234;
  /*@ assert rte: signed_overflow: (int)(idx*2)+3 ≤ 2147483647; */
  /*@ assert rte: signed_overflow: -2147483648 ≤ idx*2; */
  /*@ assert rte: signed_overflow: idx*2 ≤ 2147483647; */
  idx = idx * 2 + 3;
  /*@ assert rte: index_bound: 0 ≤ idx; */
  /*@ assert rte: index_bound: idx < 25; */
  array[idx] = 0x4567;
  /*@ assert rte: signed_overflow: idx+28 ≤ 2147483647; */
  idx += 28;
  /*@ assert rte: index_bound: 0 ≤ idx; */
  /*@ assert rte: index_bound: idx < 25; */
  array[idx] = 15;
  __retres = 0;
  return __retres;
}
```

Ukázka (01_hello/hello.c) - 2

```
$ git clone https://github.com/h0nzZik/Frama-C_Examples.git
$ cd Frama-C_Examples/01_hello
$ make
$ frama-c-gui -load project_after_analysis
```

- Informace vypsané během analýzy jsou k dispozici i v gui
- Zajímavé jsou řádky začínající `hello.c:123:[value]`
- Gui zobrazuje variační domény hodnoty proměnných na kartě "Information"

Ukázka (01_hello/hello.c) - 3

```
int main ( int argc, char *argv[] ) {
/*@ assert rte: index_bound: 0 <= argc; */
/*@ assert rte: index_bound: argc < 25; */
array[argc] = 7;
```

```
hello.c:17:[value] Assertion 'rte,index_bound' got
status unknown.
```

- Nelze ověřit, že zápis do pole proběhne v pořádku.
- Mimochodem, tyto asserty v původním zdrojovém kódu nejsou, byly vygenerovány pluginem RTE. Proto jsou označeny identifikátorem `rte`.

Ukázka (01_hello/hello.c) - 4

```
idx = Frama_C_interval(-1,26);
idx /= 2;
/*@ assert rte: index_bound: 0 < idx; */
/*@ assert rte: index_bound: idx < 25; */
array[idx] = 0x1234;
```

```
hello.c:20:[value] Assertion 'rte,index_bound' got
status valid.
```

- Funkce `Frama_C_interval` vrací nedeterministickou hodnotu ze zadaného intervalu. Je deklarována (i s anotacemi) v souboru `FRAMA_C_SHARE/builtin.h`.
- Value plugin spočítá, že po dělení bude proměnná idx ležet v intervalu $[0 \ldots 13]$

Ukázka (01_hello/hello.c) - 5

```
/*@ assert rte: signed_overflow: (int)(idx*2)+3 <= 21474836
/*@ assert rte: signed_overflow: -2147483648 <= idx*2; */
/*@ assert rte: signed_overflow: idx*2 <= 2147483647; */
idx = idx * 2 + 3;
```

- Mnoho lidí neví, že výsledek znaménkového přetečení v jazyce C není definován. RTE plugin to ale ví.
- Value určil `idx` $\in [3..29], 1\%2$

Ukázka (01_hello/hello.c) - 6

```
/*@ assert rte: index_bound: 0 <= idx; */
/*@ assert rte: index_bound: idx < 25; */
array[idx] = 0x4567;
```

- Value plugin ví, že jistě idx $\in [3..29], 1\%2$, tedy první assert projde.
- Ale protože Value pracuje s overaproximací, nedokáže rozhodnout, zda dojde k narušení druhého assertu.
- Pokud dojde k runtime chybě, nemá smysl pokračovat v běhu programu. Z toho důvodu po provedení uvedeného příkazu Value plugin uvažuje pouze hodnoy z intervalu $[3..23], 1\%2$.

Ukázka (01_hello/hello.c) - 7

```
idx += 28;
/*@ assert rte: index_bound: 0 <= idx; */
/*@ assert rte: index_bound: idx < 25; */
array[idx] = 15;
```

```
hello.c:26:[value] Assertion 'rte,index_bound' got
status invalid (stopping propagation).
```

- Po přičtení bude hodnota proměnné `idx` větší než 25. Assert tedy neprošel.

Ukázka (01_hello/hello.c) - 7

```
idx += 28;
/*@ assert rte: index_bound: 0 <= idx; */
/*@ assert rte: index_bound: idx < 25; */
array[idx] = 15;
```

```
hello.c:26:[value] Assertion 'rte,index_bound' got
status invalid (stopping propagation).
```

- Po přičtení bude hodnota proměnné `idx` větší než 25. Assert tedy neprošel.
- Kontrola: jaký je zde rozdíl oproti předchozímu případu?

Ukázka (01_hello/hello.c) - 7

```
idx += 28;
/*@ assert rte: index_bound: 0 <= idx; */
/*@ assert rte: index_bound: idx < 25; */
array[idx] = 15;
```

```
hello.c:26:[value] Assertion 'rte,index_bound' got
status invalid (stopping propagation).
```

- Po přičtení bude hodnota proměnné `idx` větší než 25. Assert tedy neprošel.
- Kontrola: jaký je zde rozdíl oproti předchozímu případu?
- Následující kód je označen za nedosažitelný. Co to znamená v praxi?

WP plugin

Motivace

- Value plugin umí dokázat některé vlastnosti funkcí

Motivace

- Value plugin umí dokázat některé vlastnosti funkcí
- Některé ale ne.
- Vyhodnocování složitých smyček je buď nepřesné (odhad invariantu), nebo pomalé (rozbalování).
- WP plugin umožňuje dokazovat vlastnosti programů podobně, jako jsme zvyklí dokazovat vlastnosti algoritmů - deduktivně.
- Silná podpora Design by Contract

Ukázkový kód

```
$ cd Frama-C_Examples/02
$ make
$ frama-c -load frama_project -wp-fct try_initialize_array

[wp] 7 goals scheduled
[wp] [Qed] Goal some_assert_1 : Valid
[wp] [Alt-Ergo] Goal some_assert_2 : Unknown (Qed:2ms)
```

- Soubor `02/array_initialization.c`
- (Vyplatí se prozkoumat dodané Makefily)
- Třeba mít nainstalován theorem prover Alt-Ergo
- Možné načíst pomoci `frama-c-gui`

```
int array[5];
for (int i = 0; i < 5; i++) {
        array[i] = 1 - i;
}
if ( array[2] == -2 )
        array[2]++;
//@ assert array[2] != -2;
//@ assert array[3] == -2;
```

- Toto již jsou ručně psané anotace.
- První assert plyne s předchozího IFu.
- WP o stavu programu po skončení cyklu nebo volání funkce nepředpokládá vůbec nic. Proto se nepodaří dokázat druhý assert.

```
int array[5];
for (int i = 0; i < 5; i++) {
        array[i] = 1 - i;
}
if ( array[2] == -2 )
        array[2]++;
//@ assert array[2] != -2;
//@ assert array[3] == -2;
```

- Toto již jsou ručně psané anotace.
- První assert plyne s předchozího IFu.
- WP o stavu programu po skončení cyklu nebo volání funkce nepředpokládá vůbec nic. Proto se nepodaří dokázat druhý assert.
- Value to zvládne, pokud mu povolíme trávit více času rozbalováním smyček.

S invariantem

```
int array[5];
/*@ loop invariant: \forall integer j;
    0 <= j < i ==> array[j] == 1 - j; */
for (int i = 0; i < 5; i++) {
/*@ assert rte: index_bound: 0 <= i; */
        array[i] = 1 - i;
}
if ( array[2] == -2 )
        array[2]++;
//@ assert array[2] != -2;
//@ assert array[3] == -2;
```

- Z invariantu lze snadno dokázat požadované vlastnosti.
- Invariant ovšem platí pouze za podmínky, že nenastane runtime chyba.
- Pokud nelze dokázat RTE assert, invariant nemusí platít.

Podmíněná platnost

```
int array[5];
/*@ loop invariant: \forall integer j;
    0 <= j < i ==> array[j] == 1 - j; */
for (int i = 0; i < 5; i++) {
        i = -1;
/*@ assert rte: index_bound: 0 <= i; */
        array[i] = 1 - i;
}
```

- Dojde k runtime chybě (možno zjistit s pomocí Value pluginu)
- WP dokáže totéž, co v předchozím případě.

Terminace

```
int array[5];
/*@ loop invariant: \forall integer j;
    0 <= j < i ==> array[j] == 1 - j; */
for (int i = 0; i < 5; i++) {
        i = 0;
/*@ assert rte: index_bound: 0 <= i; */
        array[i] = 1 - i;
}
```

- Invariant bezpodmínečně platí (WP ho dokáže)
- Tedy asserty také platí

Terminace

```
int array[5];
/*@ loop invariant: \forall integer j;
    0 <= j < i ==> array[j] == 1 - j; */
for (int i = 0; i < 5; i++) {
        i = 0;
/*@ assert rte: index_bound: 0 <= i; */
        array[i] = 1 - i;
}
```

- Invariant bezpodmínečně platí (WP ho dokáže)
- Tedy asserty také platí
- Naneštěstí cyklus neterminuje

Správné řešení

```
int array[5];
/*@ loop invariant: 0 <= i <= 5 &&
(   \forall integer j;
0 <= j < i ==> array[j] == 1 - j);
          loop variant 5 - i */
for (int i = 0; i < 5; i++) {
/*@ assert rte: index_bound: 0 <= i; */
          array[i] = 1 - i;
}
```

- Omezení na proměnnou `i` vede k platnosti RTE assertu.
- Hodnota variantu se snižuje v každém kroku cyklu až k nule.
- Program je korektní a terminuje. U rozumných smyček ale není třeba terminaci ověřovat (a psát varianty)

Výhody

- Široká dokumentace
- Snadná instalace

Omezení

Chybějící vlastnosti

- Pouze C (nepodporuje C++)
- Líbilo by se mi:
  - Třídy, používání jmenných prostorů (šetří psaní)
- Implementována pouze část ACSL
- Ošklivé chybové hlášky

Bugy

- Při výpisu chyb nesedí čísla řádků
- Občasný pád grafického rozhraní