Deductive Verification of C

Oleg Mürk

Chalmers University

June 15, 2007

Aim

- ANSI C
  - Portable type-safe subset
- Specializations
  - MISRA C
  - IA-32
  - ...
- Language variability of KeY architecture

C Vs Java: Type System

- Structures & substructures

```
struct S {
  int field;
  struct C child;
};
```

- Sized array types

```
struct S[10]
```

- Pointers

```
struct S[10]*
```

- Scalars

```
struct S[10]*@
```

C Vs Java: State

- Integers have multiple representations
- Objects can be deleted
- Dangling pointers
- Pointers to members, local variables
- Pointer to element == pointer to array

C Vs Java: Semantics

- LOTS of underspecification
- A bit of non-determinism

  ...

- More about this in the end of my talk

C Dynamic Logic: Model

- Integers types
  - Distinct from mathematical integers

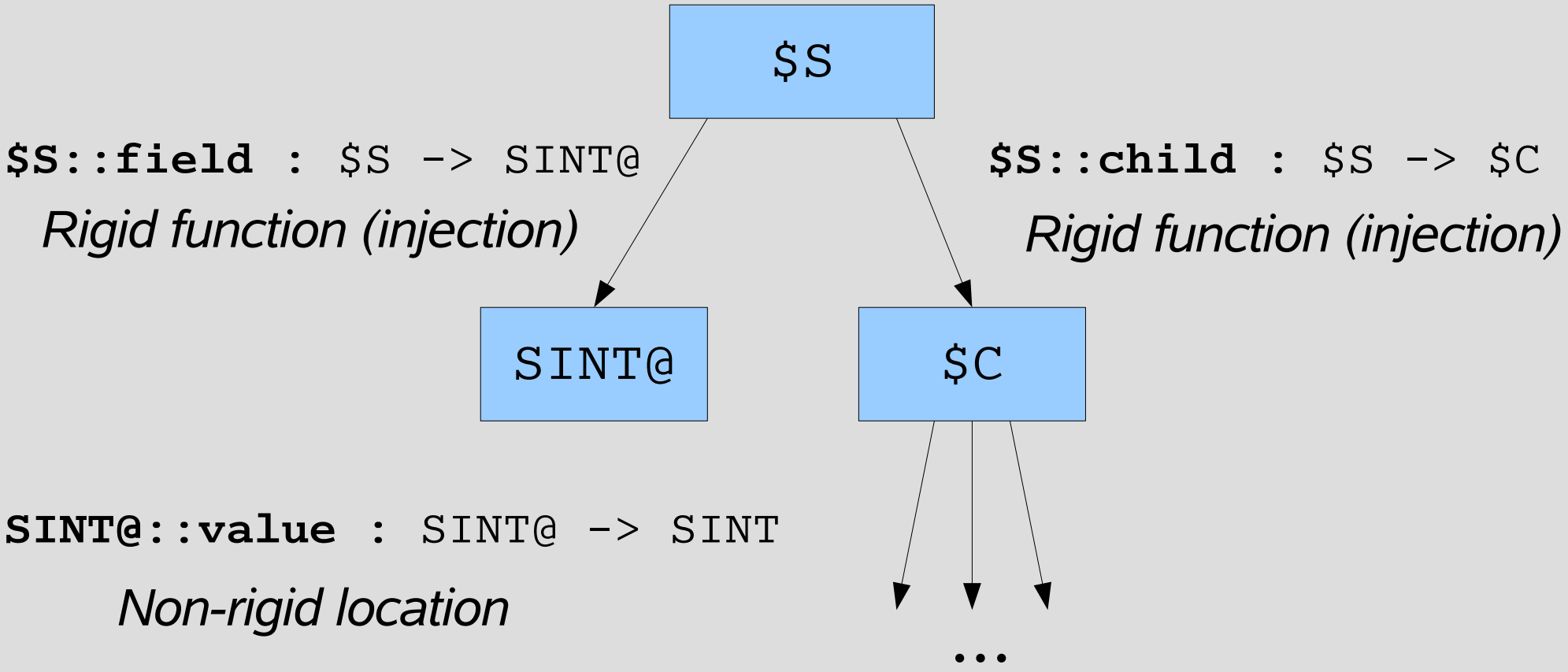
    `CHAR, ..., SINT, UINT, ...`

- Heap
  - Type-safe
  - Pointer = Reference
  - Trees of structure objects

C Dynamic Logic: Heap Model

`$S::<lookup> : int -> $S`

Rigid function (injection)



C Dynamic Logic: Modality

- WHILE language with
  - C declarations
  - C expressions
  - Memory allocation
- Block frame

C Dynamic Logic: Block Frame

```
<{
  int count = 0;
  ...
}>F
```

```
{ count := SINT::<lookup>(next); ... }
<block-frame(count) {
  count = 0;
  ...
}>F
```

Case Study

```
int count = 0;
struct Node* ptr = ifirst;
while (ptr != (struct Node*)0) {
      count++;
      ptr = ptr->next;
}
int* arr = (int*)0;
if (count > 0) {
      ...
}
*ocount = count;
*oarr = arr;
```

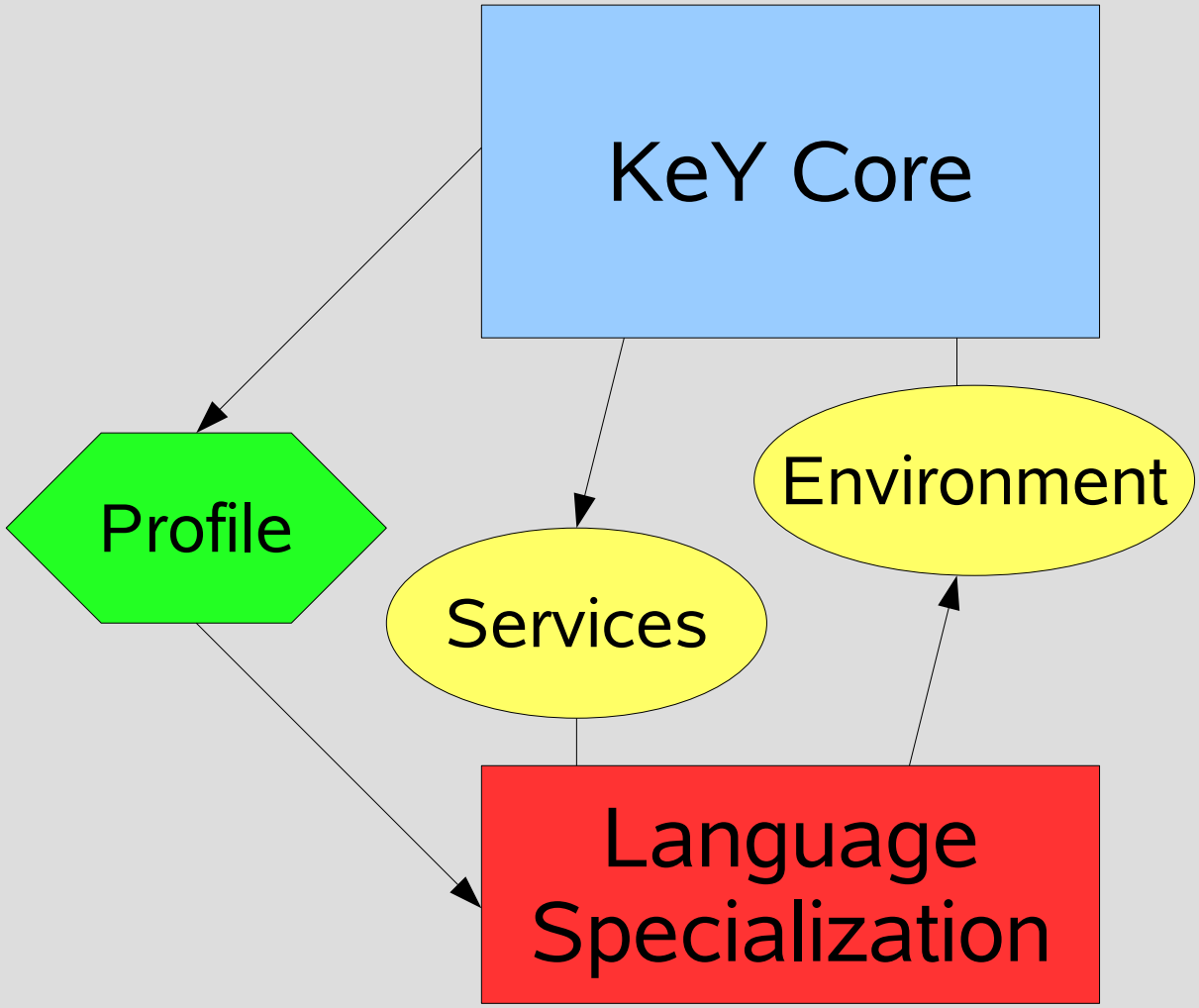
```
arr = (int*)calloc(count, sizeof(int));
if (arr != (int*)0) {
      struct Node* nptr = ifirst;
      int index = 0;
      while (index < count) {
            arr[index] = nptr->elem;
            struct Node* oldPtr = nptr;
            nptr = nptr->next;
            free(oldPtr);
            index++;
      }
}
```

Further Work

- Update Calculus
  - Deep Copy
  - Unions
  - Raw Memory Access
- Jump statements
- Modularity
- Static Analysis of Aliasing

KeY Language Variability

- Inversion of Control
- Plugins



KeY Language Variability

- Specialization
  - Type System
  - Sort System
  - Program AST
  - Parser
  - Pretty Printer
  - Schema Variables
  - Meta Operations
  - Rules
  - Strategy
  - Profile

Semantics of C

- Undefined Behavior
- Unspecified Values & Behavior
- Trap Values
- Indeterminate Values

Undefined Behavior, Indeterminate Values

```
<
   int i1, i2;
   i2 = i1;
>
i2 = i1
```

Integer Representation

```
isValidVal(i1) ->
<
  int i2 = i1 + 0;
>
i2 = i1
```

Overflows

```
<
  int i = (int)100000;
>
true
```

Sub-Expressions

```
<
    int i = 0;
    int r = a++ + a;
>
toInt(r) = 1
```

Array Tails

```
<
    int* pi1 = (new int[10])[10];
    int* pi2 = new int[5];
    int r = pi1 == pi2;
>
toInt(r) != 0
```

Type-Checking

```
<
  int i = 0;
  T temp = 100000 + i;
>
true
```

Thank You!

Handling Under-Specification

- Use abstractions
  - Mathematical integers instead of bit-reperesentations
- Condition for when the behavior is specified

```
!U noOverflow(i2) ==> U undef_x
  ==> U { i1 := convert(i2) }F
 ----------------------------
   ==> U < i1 = (int)i2; >F
```

Handling Under-Specification

- Use skolem symbols creatively

```
 toInt(temp)=toInt(i1)+toInt(i2)
                    ==>
              { r := temp }F
-----------------------------------
          < r = i1 + i2; > F

        \if (skolem = 0)
        \then (behavior1)
        \else (behavior2)
```

Handling Non-Determinism: Option 1

- Kripke structure:
  - Multiple transitions from one state
  - Possibly non-terminating transitions

- Modalities:
  - Exists transition: terminates & F `<P>F`
  - For all transitions: terminates -> F `[P]F`
  - For all transitions: terminates & F `<P>F`
  - Exists transition: terminates -> F `[P]F`

Handling Non-Determinism: Option 1

- Correctness:
  - Total correctness: `<P>F`
  - Partial correctness: `[P]F`

- No duality:

  `<P>F <-> ![P]!F`

Non-Determinism

$$\exists v; \langle i:=?; \rangle\ i = v$$

$$\langle i:=?;\ \text{if } (i==0)\ \text{bot}() \rangle\ i!=0$$

$$\Leftrightarrow$$

$$![i:=?;\ \text{if } (i==0)\ \text{bot}()]\ i=0$$

Handling Non-Determinism: Option 2

- Kripke structure: Deterministic
- Skolemize non-deterministic choice with `C`

```
    ==> { v:=C(N); N:=N+1 }F
  ---------------------------
    ==> < int v; >F
```

- Doesn't work if
  - Modality under `\exists` or `\not`
  - Two modalities

Under-Specification = Non-Determinism (If You really don't care)

- Calculus:

```
      ==> { v:=U indetVal_x }F
    ---------------------------
      ==> U < int v; >F
```

- Works both for
  - Under-Specification
  - Non-Determinism
- This is what we use in C Dynamic Logic!