Correct by Construction Algorithms by Refinement

Dominique Méry
Telecom Nancy,Université de Lorraine
dominique.mery@loria.fr

Visit of the Dishui Lake International Software Engineering Institute at East China Normal University (ECNU) from. 27th October 2025 to 31st October 2025.

Summary

Current Summary

Current Summary

Detecting overflows of computations

Listing 1: Function average

```
#include <stdio.h>
#include <limits.h>
int average(int a,int b)
{
  return((a+b)/2);
}

int main()
{
  int x,y;
  x=INT_MAX;y=INT_MAX;
  printf("Average for %d and %d is %d\n",x,y,
         average(x,y));
  return 0;
}
```

Execution

Execution produces a result

```
Average  for 2147483647 and 2147483647 is -1
```

Execution

Execution produces a result

```
Average  for 2147483647 and 2147483647 is -1
```

Using frama-c produces a required annotation

```
int average(int a, int b)
{
  int __retres;
  /*@ assert rte: signed_overflow: -2147483648 <= a + b; */
  /*@ assert rte: signed_overflow: a + b <= 2147483647; */
  __retres = (a + b) / 2;
  return __retres;
}
```

Annotated Example 1

Listing 2: Function average.....

```
#include <stdio.h>
#include <limits.h>
/*@  requires  0 <= a;
     requires a <= INT_MAX ;
     requires  0 <= b;
     requires b <= INT_MAX ;
     requires  0 <= a+b;
     requires a+b <= INT_MAX ;
     ensures \result <= INT_MAX;
*/
int average(int a,int b)
{
  return((a+b)/2);
}

int main()
{
  int x,y;
  x=INT_MAX / 2;y=INT_MAX / 2;
  // printf("Average for %d and %d is %d\n",x,y,
  //     );
  return average(x,y);
}
```

Current Summary

Nose Gear Velocity

- Estimated ground velocity of the aircraft should be available only if it is within 3 km/hr of the true velocity at some moment within past 3 seconds

Characterization of a System (I)

- NG velocity system:
  - **Hardware**:
    - *Electro-mechanical sensor*: detects rotations
    - *Two 16-bit counters*: Rotation counter, Milliseconds counter
    - *Interrupt service routine*: updates rotation counter and stores current time.
  - **Software**:
    - *Real-time operating system*: invokes update function every `500 ms`
    - *16-bit global variable*: for recording rotation counter update time
    - *An update function*: estimates ground velocity of the aircraft.
- Input data available to the system:
  - *time*: in `milliseconds`
  - *distance*: in `inches`
  - *rotation angle*: in `degrees`
- Specified system performs velocity estimations in *imperial* unit system
- Note: expressed functional requirement is in *SI* unit system (km/hr).

Characterization of a System (II) cont.

What are the main properties to consider for formalization?

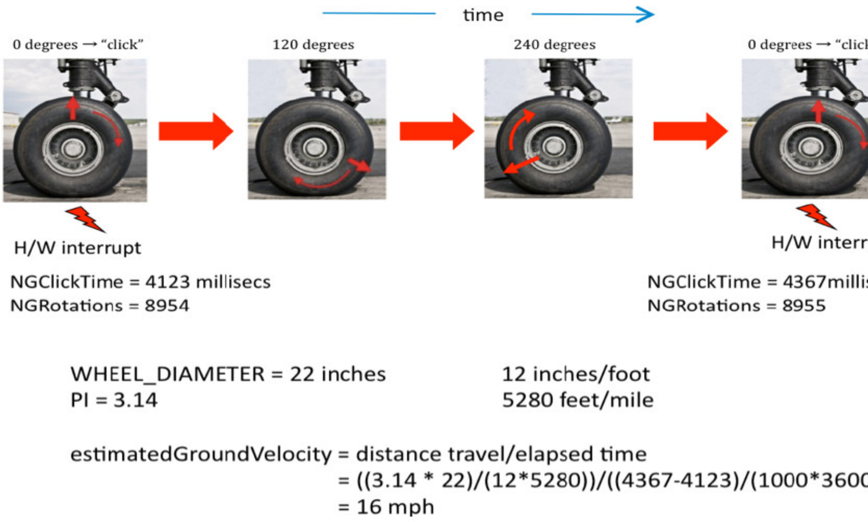
- Two different types of data:
  - counters with modulo semantics
  - non-negative values for time, distance, and velocity
- Two dimensions: *distance* and *time*
- Many units: distance (`inches`, `kilometers`, `miles`), time (`milliseconds`, `hours`), velocity (`kph`, `mph`)
- And interaction among components

How should we model?

- Designer needs to consider units and conversions between them to manipulate the model
- One approach: Model units as *sets*, and conversions as constructed types – *projections*.
- Example:

1 $estimateVelocity \in \mathtt{MILES} \times \mathtt{HOURS} \rightarrow \mathtt{MPH}$
2 $mphTokph \in \mathtt{MPH} \rightarrowtail\!\!\!\!\twoheadrightarrow \mathtt{KPH}$

Sample Velocity Estimation



Safety Property Run Time Error (RTE)

Safety Property

Safety Property Run Time Error (RTE)

Safety Property

- Storing the number of `NGClick` in a n-bit variable VNGClick
- Integers are denoted by the set *Int* and is simply defined by the interval *Int*$\hat{=} INT\_MIN..INT\_MAX$.
- Safety requirement:
  The value of `VNGClick` is always in the range of implementation Int
  or equivalently $VNGClick \in Int$

Safety Property Run Time Error (RTE)

Safety Property

- Storing the number of `NGClick` in a n-bit variable VNGClick
- Integers are denoted by the set *Int* and is simply defined by the interval *Int*$\hat{=}INT\_MIN..INT\_MAX$.
- Safety requirement:
  The value of ***`VNGClick`*** *is always in the range of implementation Int* or equivalently $VNGClick \in Int$

- $Length = \pi * diameter * VNGClick$ (**mathematical property**)
- $Length \leq 6000$ (**domain property**)
- $\pi * diameter * VNGClick \leq 6000$
- $VNGClick \leq 6000/(\pi * diameter)$
  - if n=8, then $2^7 - 1 = 127$ and $6000/(\pi * [22, inch]) = 6000/(\pi * 55, 88) = 6000/(3, 24 * [55, 88, cm]) = 6000/(3, 24 * 0.5588) \approx 3419$ and the condition of safety can not be satisfied in any situation.
  - if n=16, then $2^{15} - 1 = 65535$ and $6000/(\pi * [22, inch]) \approx 3419$ and the condition of safety can be satisfied in any situation since $3419 <= 65535$.

Safety Property Run Time Error (RTE)

Safety Property

- Storing the number of `NGClick` in a n-bit variable VNGClick
- Integers are denoted by the set *Int* and is simply defined by the interval *Int*$\hat{=} INT\_MIN..INT\_MAX$.
- Safety requirement:
  The value of `VNGClick` is always in the range of implementation Int
  or equivalently $VNGClick \in Int$

$$RTE\_VNGClick : 0 \leq vNGClick \leq INT\_MAX \quad (1)$$

- The current value of VNGClick is always bounded by the two values 0 and INT_MAX.

Current Summary

Verifying program correctness (Run Time Errors, . . .)

A program P *satisfies* a (pre,post) contract:

- P transforms a variable v from initial values v_0 and produces a final value v_f: $v_0 \xrightarrow{P} v_f$
- v_0 satisfies pre: $\text{pre}(v_0)$ and v_f satisfies post : $\text{post}(v_0, v_f)$
- $\text{pre}(v_0) \wedge v_0 \xrightarrow{P} v_f \Rightarrow \text{post}(v_0, v_f)$
- $\mathbb{D}$ est le domaine RTE de V

Verifying program correctness (Run Time Errors, . . .)

A program P *satisfies* a (pre,post) contract:

- P transforms a variable v from initial values v_0 and produces a final value v_f: $v_0 \xrightarrow{\mathsf{P}} v_f$
- v_0 satisfies pre: pre(v_0) and v_f satisfies post : post(v_0, v_f)
- pre$(v_0) \wedge v_0 \xrightarrow{\mathsf{P}} v_f \Rightarrow$ post(v_0, v_f)
- $\mathbb{D}$ est le domaine RTE de V

requires $pre(v_0)$
ensures $post(v_0, v_f)$
variables X

$$\left[\begin{array}{l}
\text{begin} \\
0 : P_0(v_0, v) \\
\texttt{instruction}_0 \\
\ldots \\
i : P_i(v_0, v) \\
\ldots \\
\texttt{instruction}_{f-1} \\
f : P_f(v_0, v) \\
\text{end}
\end{array}\right.$$

- $pre(v_0) \wedge v = v_0 \Rightarrow P_0(v_0, v)$
- $pre(v_0) \wedge P_f(v_0, v) \Rightarrow post(v_0, v)$
- For any pair of labelsℓ, ℓ' such that $\ell \longrightarrow \ell'$, one verifies that, pour any values $v, v' \in \textsc{Memory}$
$$\left(\begin{array}{l} \left(\begin{array}{l} pre(v_0) \wedge P_\ell(v_0, v)) \\ \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \end{array} \right) \\ \Rightarrow P_{\ell'}(v_0, v') \end{array} \right),$$
- For any pair of labels m, n such taht $m \longrightarrow n$, one verifies that, $\forall v, v' \in \textsc{Memory}$: $pre(v_0) \wedge P_m(v_0, v) \Rightarrow \textbf{DOM}(m, n)(v)$

Example of an annotation

Variables X
Requires $\ldots$
Ensures $\ldots$
While $0 < X$ **DO**
$X := X - 1;$
Od;

Example of an annotation

Variables X
Requires $\ldots$
Ensures $\ldots$
While $0 < X$ **DO**
$X := X - 1;$
Od;

$\longrightarrow$

Example of an annotation

Variables X
Requires $\ldots$
Ensures $\ldots$
While $0 < X$ **DO**
$X := X - 1;$
Od;

$\longrightarrow \longrightarrow$

Example of an annotation

Variables X
Requires $\ldots$
Ensures $\ldots$
While $0 < X$ **DO**
$X := X - 1;$
Od;

$\longrightarrow \longrightarrow \longrightarrow$

Example of an annotation

Variables X
Requires $\ldots$
Ensures $\ldots$
While $0 < X$ **DO**
$X := X - 1;$
Od;

$\longrightarrow \longrightarrow \longrightarrow \longrightarrow$

Example of an annotation

Variables X
Requires $\ldots$
Ensures $\ldots$
While $0 < X$ **DO**
$X := X - 1;$
Od;

$\longrightarrow \longrightarrow \longrightarrow \longrightarrow$

Contract EX
Variables $X(int)$
Requires $x_0 \in \mathbb{N}$
Ensures $x_f = 0$
$\ell_0 : \{ x = x_0 \wedge x_0 \in \mathbb{N}\}$
While $0 < X$ **DO**
$\ell_1 : \{0 < x \leq x_0 \wedge x_0 \in \mathbb{N}\}$
$X := X - 1;$
$\ell_2 : \{0 \leq x \leq x_0 \wedge x_0 \in \mathbb{N}\}$
Od;
$\ell_3 : \{x = 0\}$

A Simple C Function

Listing 3: Simple contract

```
/*@ requires  \false ;
   @ ensures \false ; */
int f1(int x)
{ if (f1(x) <= 0)
     { return(1);
     }
  else
     { return(0);
     }
}
```

A Simple C Function

Listing 4: Simple contract

```
#include <stdio.h>
#include <math.h>


/*@ requires  \false ;
   @ ensures \false ; */
int f1(int x)
{ if (f1(x) <= 0)
     { return(1);
     }
  else
     { return(0);
     }
}
```

A Simple C Function

Listing 5: Simple contract

```
#include <stdio.h>
#include <math.h>


/*@ requires  \false ;
   @ ensures \false ; */
int f1(int x)
{ if (f1(x) <= 0)
    { return(1);
    }
  else
    { return(0);
    }
}
```

Correct by Construction

- Finding or computing annotations is difficult and even undecidable!

Correct by Construction

- Finding or computing annotations is difficult and even undecidable!
- Given

Contract EX Variables $X(int)$ Requires $pre(x_0)$ Ensures $post(x_0, x_f)$

Correct by Construction

- Finding or computing annotations is difficult and even undecidable!
- Given

  Contract EX
  Variables $X(int)$
  Requires $pre(x_0)$
  Ensures $post(x_0, x_f)$

- Design of the algorithm ALG fulfiling the contract

Correct by Construction

- Finding or computing annotations is difficult and even undecidable!
- Given

Contract EX Variables $X(int)$ Requires $pre(x_0)$ Ensures $post(x_0, x_f)$

- Design of the algorithm ALG fulfiling the contract
- HOARE triple:

$$\{\mathbf{pre(x_0)} \wedge \mathbf{x} = \mathbf{x_0}\}\text{ALG}\{\mathbf{post(x_0, x)}\}$$

Correct by Construction

- Finding or computing annotations is difficult and even undecidable!
- Given

Contract EX
Variables $X(int)$
Requires $pre(x_0)$
Ensures $post(x_0, x_f)$

- Design of the algorithm ALG fulfiling the contract
- HOARE triple:

$$\{\mathbf{pre}(\mathbf{x_0}) \wedge \mathbf{x} = \mathbf{x_0}\}\text{ALG}\{\mathbf{post}(\mathbf{x_0}, \mathbf{x})\}$$

Idea?

From Contract EX, using a step by step approach to find an algorithm ALG satisfying it using refinement and Event-B models.
Applying the CLEANROOM model!

Current Summary

Short Summary on Event-B (context)

- Context: static properties of Event-B models
  - Sets: user-defined types
  - Constants: static object in development
  - Axioms: presumed properties about sets and constants
  - Theorems: derived properties about sets and constants

SETS
  A
CONSTANTS
  B, C, f
AXIOMS
  $ax1 : B \subseteq A$
  $ax2 : C \subseteq A$
  $ax3 : g \in B \twoheadrightarrow C$
  $ax4 : \forall A.A \subseteq \mathbb{N} \wedge 0 \in A \wedge suc[A] \subseteq A \Rightarrow \mathbb{N} \subseteq A$
...

Short Summary on Event-B (discrete)

- Machine: behavioral properties of Event-B models
  - Variables: states
  - Invariants: properties of variables that always need to hold
  - Theorems: derived properties about variables
  - Events: possible state changes

```
EVENT e
  ANY t
  WHERE
    G(c,s,t,x)
  THEN
    x :|(P(c,s,t,x,x'))
  END
```

- c et s are constantes and visible sets by e
- x is a state variable or a list of variabless
- $G(c,s,t,x)$ is the condition for observing e.
- $P(c,s,t,x,x')$ is the assertion for the relation over x and x'.
- $BA(e)(c,s,x,x')$ is the *before-after* relationship for e and is defined by $\exists t.G(c,s,t,x) \ \wedge \ P(c,s,t,x,x')$.

Short Summary on Event-B (refinement)

- Given an abstract and a corresponding concrete event

EVENT ae $\widehat{=}$	EVENT ce $\widehat{=}$
any v **where**	**any** w **where**
$G(x,v)$	$H(y,w)$
then	**then**
$x := E(x,v)$	$y := F(y,w)$
end	**end**

$$
\begin{array}{l}
I(x) \quad \wedge \quad J(x,y) \quad \wedge \quad H(y,w) \\
\Longrightarrow \\
\exists v \cdot (\, G(x,v) \quad \wedge \quad J(E(x,v), F(y,w))\,)
\end{array}
$$

- $BA(ae)(x,x') \;\widehat{=}\; \exists v.G(x,v) \wedge x' = E(x)$
- $BA(ce)(y,y') \;\widehat{=}\; \exists w.H(y,w) \wedge y' = F(y)$

Modelling systems in Event-B

MACHINE
m
SEES
c
VARIABLES
x
INVARIANT
$I(x)$
THEOREMS
$Q(x)$
INITIALISATION
$Init(x)$
EVENTS
$\ldots e$
END

Modelling systems in Event-B

MACHINE
m
SEES
c
VARIABLES
x
INVARIANT
$I(x)$
THEOREMS
$Q(x)$
INITIALISATION
$Init(x)$
EVENTS
$\ldots e$
END

c defines the static environment for the proofs related to m: sets, constants, axioms, theorems $\Gamma(m)$.

Modelling systems in Event-B

MACHINE
m
SEES
c
VARIABLES
x
INVARIANT
$I(x)$
THEOREMS
$Q(x)$
INITIALISATION
$Init(x)$
EVENTS
$\ldots e$
END

c defines the static environment for the proofs related to m: sets, constants, axioms, theorems $\Gamma(m)$.

$\Gamma(m) \vdash \forall x \in Values : \textsc{Init}(x) \Rightarrow \mathrm{I}(x)$

Modelling systems in Event-B

MACHINE
m
SEES
c
VARIABLES
x
INVARIANT
$I(x)$
THEOREMS
$Q(x)$
INITIALISATION
$Init(x)$
EVENTS
$\ldots e$
END

c defines the static environment for the proofs related to m: sets, constants, axioms, theorems $\Gamma(m)$.

$\Gamma(m) \vdash \forall x \in Values : \text{INIT}(x) \Rightarrow \mathrm{I}(x)$

$\forall e :$

$\Gamma(m) \vdash \forall x, x', u \in Values : \mathrm{I}(x) \wedge R(u, x, x') \Rightarrow \mathrm{I}(x')$

Modelling systems in Event-B

MACHINE
m
SEES
c
VARIABLES
x
INVARIANT
$I(x)$
THEOREMS
$Q(x)$
INITIALISATION
$Init(x)$
EVENTS
$\ldots e$
END

c defines the static environment for the proofs related to m: sets, constants, axioms, theorems $\Gamma(m)$.

$\Gamma(m) \vdash \forall x \in Values : \textsc{Init}(x) \Rightarrow \mathrm{I}(x)$

$\forall e :$

$\Gamma(m) \vdash \forall x, x', u \in Values : \mathrm{I}(x) \wedge R(u, x, x') \Rightarrow \mathrm{I}(x')$

$\Gamma(m) \vdash \forall x \in Values : \mathrm{I}(x) \Rightarrow \mathrm{Q}(x)$

MACHINE
  m
SEES
  c
VARIABLES
  x
INVARIANT
  $I(x)$
THEOREMS
  $Q(x)$
INITIALISATION
  $Init(x)$
EVENTS
  $\ldots e$
END

c defines the static environment for the proofs related to m: sets, constants, axioms, theorems $\Gamma(m)$.

$\Gamma(m) \vdash \forall x \in Values : \textsc{Init}(x) \Rightarrow \mathrm{I}(x)$

$\forall e :$

$\Gamma(m) \vdash \forall x, x', u \in Values : \mathrm{I}(x) \wedge R(u, x, x') \Rightarrow \mathrm{I}(x')$

$\Gamma(m) \vdash \forall x \in Values : \mathrm{I}(x) \Rightarrow \mathrm{Q}(x)$

e
ANY
  u
WHERE
  $G(x, u)$
THEN
  $x : |(R(u, x, x'))$
END

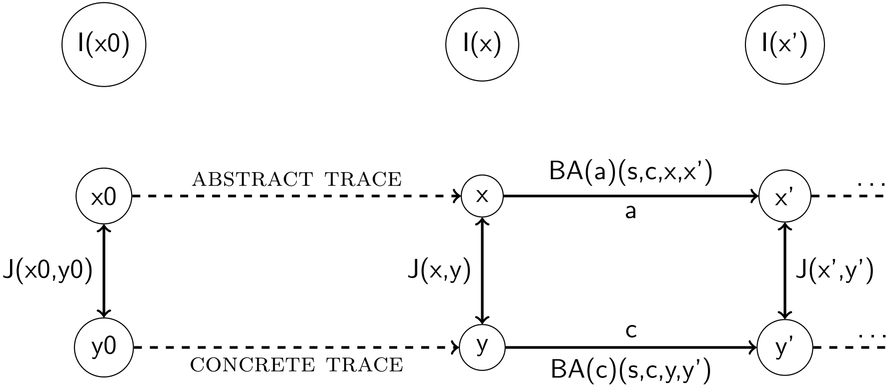
or e is **observed** $x \xrightarrow{e} x'$

Event B Structure and Proofs

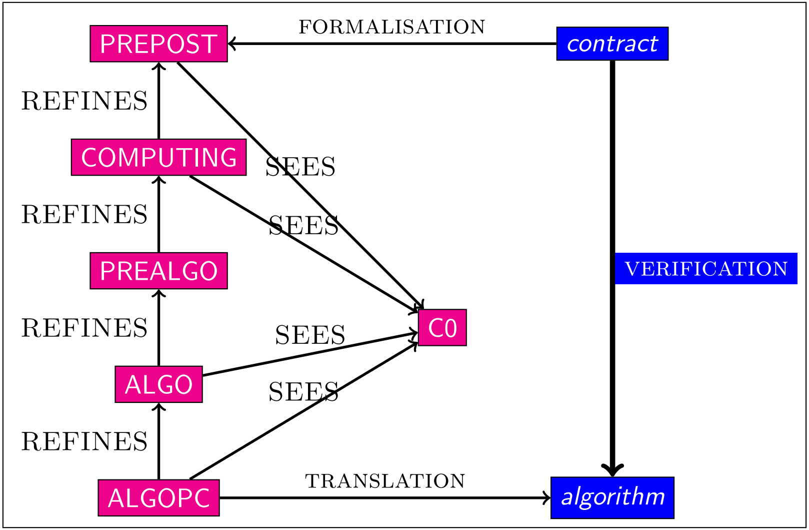
CONTEXT *ctxt_id_2* **EXTENDS** *ctxt_id_1* **SETS** s **CONSTANTS** c **AXIOMS** $A(s,c)$ **THEOREMS** $T_c(s,c)$ **END**	**MACHINE** *machine_id_2* **REFINES** *machine_id_1* **SEES** *ctxt_id_2* **VARIABLES** v **INVARIANTS** $I(s,c,v)$ **THEOREMS** $T_m(s,c,v)$ **VARIANT** $V(s,c,v)$ **EVENTS** **EVENT** e **any** x **where** $G(s,c,v,x)$ **then** $v :\mid BA(s,c,v,x,v')$ **end** **END**

Invariant preservation	$A(s,c) \wedge I(s,c,v)$ $\wedge G(s,c,v,x)$ $\wedge BA(s,c,v,x,v')$ $\Rightarrow I(s,c,v')$
Event feasibility	$A(s,c) \wedge I(s,c,v)$ $\wedge G(s,c,v,x)$ $\Rightarrow \exists v'.BA(s,c,v,x,v')$
Variant modelling progress	$A(s,c) \wedge I(s,c,v)$ $\wedge G(s,c,v,x)$ $\wedge BA(s,c,v,x,v')$ $\Rightarrow V(s,c,v') < V(s,c,v)$
Theorems	$A(s,c) \Rightarrow T_c(s,c)$ $A(s,c) \wedge I(s,c,v)$ $\Rightarrow T_m(s,c,v)$

Refinement between two machines



The Iterative Pattern



Current Summary

Current Summary

Computing two sums

First n numbers and first odd/even numbers

- First the **pre/post specification** ...
- Possibility to use a programming language with contracts too

Computing two sums

First n numbers and first odd/even numbers

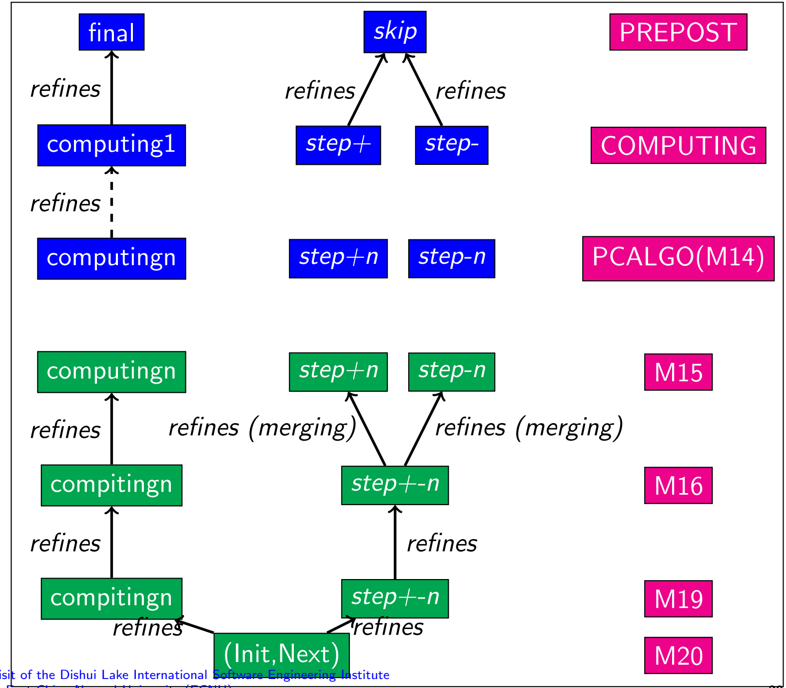
- First the **pre/post specification** ...
- Possibility to use a programming language with contracts too

requires $input_0 \geq 0 \wedge rs_0, re_0 \in \mathbb{Z}$

ensures $\begin{cases} rs_f = s(input_0) \\ re_f = es(input_0) \\ input_f = input_0 \end{cases}$

variables input, re, rs

- Find two sequences s and es computing the sum of first n natural numbers and first even numbers smaller than n
- Prove that:
  - $\forall n.n \in \mathbb{N} \Rightarrow s(n) = \sum_{i=0}^{i=n} i.$
  - $\forall n.n \in \mathbb{N} \Rightarrow es(n) = \sum_{i=0}^{i=n/2} 2 * i.$


final
skip
PREPOST
refines
refines
refines
computing1
step+
step-
COMPUTING
refines
computingn
step+n
step-n
PCALGO(M14)
computingn
step+n
step-n
M15
refines
refines (merging)
refines (merging)
compitingn
step+-n
M16
refines
refines
compitingn
step+-n
M19
refines
refines
(Init,Next)
M20

First n numbers and first odd/even numbers

$axm1 : n \in \mathbb{N}$
$axm2 : s \in \mathbb{N} \to \mathbb{N} \land os \in \mathbb{N} \to \mathbb{N} \land es \in \mathbb{N} \to \mathbb{N}$
$axm3 : es(0) = 0 \land os(0) = 0 \land s(0) = 0$
$axm4 : \forall i, l \cdot i \in \mathbb{N} \land l \in \mathbb{N} \land i = 2 * l \Rightarrow s(i+1) = s(i) + i + 1 \land es(i+1) =$
$axm5 : \forall i, l \cdot i \in \mathbb{N} \land l \in \mathbb{N} \land i = 2 * l + 1 \Rightarrow s(i+1) = s(i) + i + 1 \land es(i +$
$axm6 : suc \in \mathbb{N} \to \mathbb{N} \land (\forall i \cdot i \in \mathbb{N} \Rightarrow suc(i) = i + 1)$
$axm7 : \forall A \cdot A \subseteq \mathbb{N} \land 0 \in A \land suc[A] \subseteq A \Rightarrow \mathbb{N} \subseteq A$
$th1 : \forall i \cdot i \in \mathbb{N} \Rightarrow s(i+1) = s(i) + i + 1$
$th2 : \forall u, v \cdot u \in \mathbb{N} \land v \in \mathbb{N} \land 2 * u = v \Rightarrow u = v/2$
$th3 : \forall k \cdot k \in \mathbb{N} \Rightarrow 2 * s(k) = k * k + k$
$th4 : \forall k \cdot k \in \mathbb{N} \Rightarrow s(k) = (k * k + k)/2$
$th5 : \forall k \cdot k \in \mathbb{N} \Rightarrow es(2 * k) = 2 * s(k)$
$th6 : \forall k \cdot k \in \mathbb{N} \Rightarrow es(2 * k + 1) = 2 * s(k)$
$th7 : \forall k \cdot k \in \mathbb{N} \land k \neq 0 \Rightarrow os(2 * k) = k * k$
$th8 : \forall k \cdot k \in \mathbb{N} \Rightarrow os(2 * k + 1) = (k + 1) * (k + 1)$

Stating the pre/post specification in Event-B

INVARIANTS

$inv1 : input \in \mathbb{Z}$
$inv6 : input = n$
$inv2 : rs \in \mathbb{Z} \land re \in \mathbb{Z}$
$inv3 : ok \in BOOL$
$inv4 : ok = TRUE \Rightarrow rs = s(input) \land re = es(input)$

END

Stating the pre/post specification in Event-B

then

$act1 : ok := FALSE$
$act2 : rs :\in \mathbb{Z}$
$act3 : re :\in \mathbb{Z}$
$act4 : input := n$

EVENT computing
when

$grd1 : ok = FALSE$

then

$act1 : rs := s(input)$
$act2 : ok := TRUE$
$act3 : re := es(input)$

END

Refining for computing (1)

INVARIANTS

$inv1 : cur \in 0 \,..\, n$

$inv2 : ee \in 0 \,..\, n \nrightarrow \mathbb{N}$

$inv3 : ss \in 0 \,..\, n \nrightarrow \mathbb{N}$

$inv5 : dom(ss) = 0 \,..\, cur \land dom(ee) = dom(ss) \land dom(ss) \subseteq \mathbb{N}$

$inv6 : \forall i \cdot i \in 0 \,..\, cur \Rightarrow ee(i) = es(i) \land ss(i) = s(i)$

Variant

then

$act1 : ok := FALSE$

$act2 : rs :\in \mathbb{Z}$

$act3 : re :\in \mathbb{Z}$

$act4 : input := n$

$act5 : ee := \{0 \mapsto 0\}$

$act6 : ss := \{0 \mapsto 0\}$

$act7 : cur := 0$

END

Refining for computing (2)

EVENT computing11
when

$grd1 : ok = FALSE$
$grd2 : cur = n$

then

$act1 : rs := ss(input)$
$act2 : ok := TRUE$
$act3 : re := ee(input)$

END

Refining for computing (3)

EVENT step11+
Any
$k,$
when
$grd1 : ok = FALSE$
$grd2 : k \in \mathbb{N} \land cur = 2 * k + 1$
$grd3 : cur < n$
$grd4 : cur < n$

then
$act1 : ss(cur + 1) := ss(cur) + cur + 1$
$act2 : ee(cur + 1) := ee(cur) + cur + 1$
$act3 : cur := cur + 1$

END

Refining for computing (4)

EVENT step11=

Any

$k,$

when

$grd1 : ok = FALSE$

$grd2 : k \in \mathbb{N} \land cur = 2 * k$

$grd3 : cur < n$

then

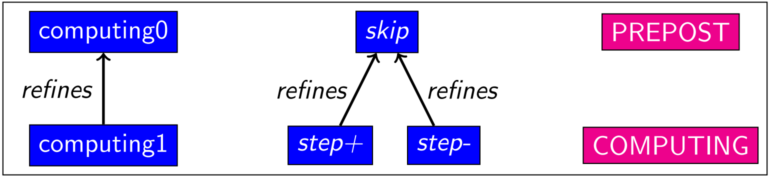
$act1 : ss(cur + 1) := ss(cur) + cur + 1$

$act2 : ee(cur + 1) := ee(cur)$

$act3 : cur := cur + 1$

END

Diagram of events



Completing the analysis

MACHINE $M1$ **SEES** $C0$

VARIABLES

$input, rs, re, ok,$

INVARIANTS

$inv1 : input \in \mathbb{Z}$

$inv6 : input = n$

$inv2 : rs \in \mathbb{Z} \wedge re \in \mathbb{Z}$

$inv3 : ok \in BOOL$

$inv4 : ok = TRUE \Rightarrow rs = s(input) \wedge re = es(input)$

Completing the analysis

MACHINE $M11$ **SEES** $C0$

VARIABLES

$input, rs, re, ok, ee, ss, cur,$

INVARIANTS

$inv1 : cur \in 0 \,..\, n$

$inv2 : ee \in 0 \,..\, n \rightarrow\!\!\!\!\!\!+ \mathbb{N}$

$inv3 : ss \in 0 \,..\, n \rightarrow\!\!\!\!\!\!+ \mathbb{N}$

$inv5 : dom(ss) = 0 \,..\, cur \wedge dom(ee) = dom(ss) \wedge dom(ss) \subseteq \mathbb{N}$

$inv6 : \forall i \cdot i \in 0 \,..\, cur \Rightarrow ee(i) = es(i) \wedge ss(i) = s(i)$

Completing the analysis

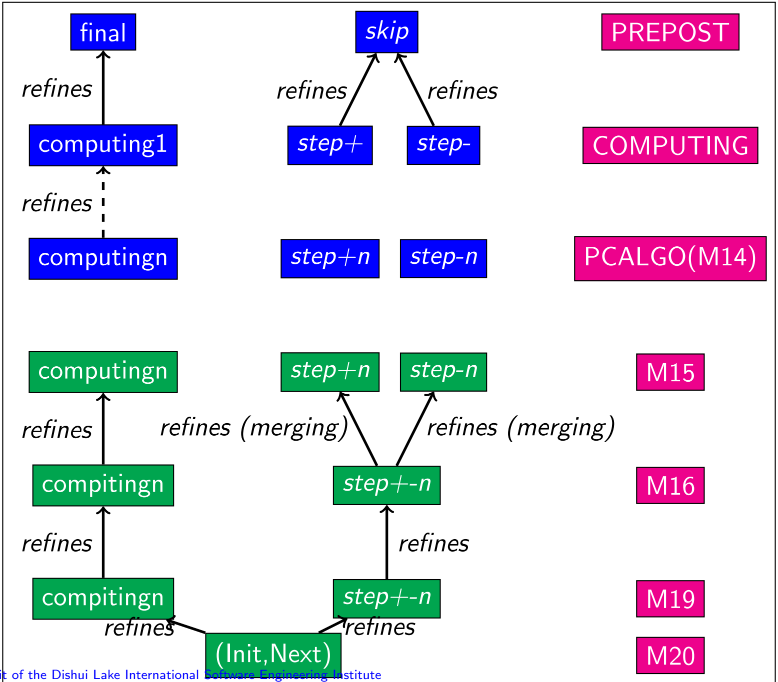
MACHINE $M12$ **SEES** $C0$
VARIABLES
$input, rs, re, ok, ee, ss, cur, cs, ce,$
INVARIANTS
$inv1 : cs = ss(cur)$
$inv2 : ce = ee(cur)$

Completing the analysis

MACHINE $M13$ **SEES** $C0$
VARIABLES
$input, rs, re, ok, ee, ss, cur, cs, ce,$
INVARIANTS
$inv1 : cs = ss(cur)$
$inv2 : ce = ee(cur)$
$inv4 : ce = es(cur)$
$inv3 : cs = s(cur)$

Completing the analysis

MACHINE $M14$ **SEES** $C0, C1$
VARIABLES
$input, rs, re, cur, cs, ce, l,$
INVARIANTS
$inv1 : cs = s(cur)$
$inv5 : l = start \Leftrightarrow ok = FALSE$
$inv6 : l = end \Leftrightarrow ok = TRUE$
$inv2 : cs = s(cur)$
$inv3 : l \in L$
$inv4 : l = end \Rightarrow re = es(n) \wedge rs = s(n)$


final
skip
PREPOST
refines
refines
refines
computing1
step+
step-
COMPUTING
refines
computingn
step+n
step-n
PCALGO(M14)
computingn
step+n
step-n
M15
refines
refines (merging)
refines (merging)
compitingn
step+-n
M16
refines
refines
compitingn
step+-n
M19
refines
refines
(Init,Next)
M20

Generation of an algorithm from M14

Translation from Evenrts

```
struct sums  codesum(int n)
{int  k,ce,cs; struct sums r;
  r.s=0;r.se=0;k=0;ce=0;cs=0;
  while (k<n)
    {
      if ( k % 2 != 0)
        { ce = ce + k + 1;cs = cs +k+1;       k = k +1;}
      else
        { ce = ce ;cs = cs +k+1;       k = k +1;}
     }
  r.s=cs;r.se=ce;
  return(r);
}
```

Generation of an algorithm from M14

Improving and checking with Frama-c

```
#include "structure.h"
#include "trans-even.h"
struct sums  codesum(int n)
{int  k,ce,cs; struct sums r;
  r.s=0;r.se=0;k=0;ce=0;cs=0;
  /*@ loop invariant k >= 0 && k <= n && mathsum(k) == cs;
    @ loop invariant  ( (k % 2 == 0) ==> (mathse(k) == ce));
    @ loop invariant  ( (k % 2 != 0) ==> (mathse(k) == ce));
    loop assigns k, ce,cs;
    loop variant n-k;
   */
  while (k<n)
    {
      if ( k % 2 != 0)
        { ce = ce + k + 1;cs = cs +k+1;       k = k +1;}
      else
        { ce = ce ;cs = cs +k+1;        k = k +1;}
      }
  r.s=cs;r.se=ce;
  return(r);
```

Generation of an algorithm from M14

Improving and checking with Frama-c

```c
#include "structure.h"
#include "trans-even.h"
struct sums codesum(int n)
{int k,ce,cs; struct sums r;
  r.s=0;r.se=0;k=0;ce=0;cs=0;
  /*@ loop invariant k >= 0 && k <= n && mathsum(k) == cs;
    @ loop invariant ( (k % 2 == 0) ==> (mathse(k) == ce));
    @ loop invariant ( (k % 2 != 0) ==> (mathse(k) == ce));
    loop assigns k, ce,cs;
    loop variant n-k;
   */
  while (k<n)
    {
      if ( k % 2 != 0)
        { ce = ce + k + 1;}
      cs = cs +k+1;       k = k +1;
 cp trabs-e*          }
  r.s=cs;r.se=ce;
  return(r);
}
```

Using the Plugin EB2Algo

```
rs :∈ Z ||
re :∈ Z ||
input = n ||
cur = 0 ||
ce = 0 ||
cs = 0

while cur≠n do
     if cur mod 2≠0 then
          cur = cur+1 ||
          cs = cs+cur+1 ||
          ce = ce+cur+1
     else
          cur = cur+1 ||
          cs = cs+cur+1 ||
          ce = ce
```

Current Summary

Merging events

EVENT e1
any
x
where
$G1(s,c,x,u)$
then
$u :|(R(s,c,x,u,u'))$
end

EVENT e2
any
x
where
$G2(s,c,x,u)$
then
$u :|(R(s,c,x,u,u'))$
end

Merging events

EVENT e1
any
x
where
$G1(s, c, x, u)$
then
$u :| (R(s, c, x, u, u'))$
end

EVENT e2
any
x
where
$G2(s, c, x, u)$
then
$u :| (R(s, c, x, u, u'))$
end

EVENT e $refines\ e1, e2$
any
x
where
$H(s, c, x, u)$
then
$u :| (R(s, c, x, u, u'))$
end

Merging events

EVENT e1
any
x
where
$G1(s,c,x,u)$
then
$u :|(R(s,c,x,u,u'))$
end

EVENT e2
any
x
where
$G2(s,c,x,u)$
then
$u :|(R(s,c,x,u,u'))$
end

EVENT e $refines\ e1, e2$
any
x
where
$H(s,c,x,u)$
then
$u :|(R(s,c,x,u,u'))$
end

Checking the Proof Obligation:

$$Ax(s,c), I(s,c,u), H(s,c,x,u) \vdash G1(s,c,x,u) \lor G2(s,c,x,u)$$

Synthesis Phase

- Preparing merging by transforming actions.

EVENT e
any
  x
where
  $G(s, c, x, u)$
then
  $u :| (R(s, c, x, u, u'))$
end

EVENT f refines e
any
  x
where
  $G(s, c, x, u)$
then
  $u :| (G(s, c, x, u) \Rightarrow R(s, c, x, u, u'))$
end

- Internalizing the action as a before after relation
- $G(s, c, x, u) \wedge (G(s, c, x, u) \Rightarrow R(s, c, x, u, u')) \Leftrightarrow$
  $G(s, c, x, u) \wedge R(s, c, x, u, u')$

Synthesis Phase

- Preparing merging by transforming actions.

EVENT e
any
  x
where
  $G(s,c,x,u)$
then
  $u :| (R(s,c,x,u,u'))$
end

EVENT f refines e
any
  x
where
  $G(s,c,x,u)$
then
  $u :| (G(s,c,x,u) \Rightarrow R(s,c,x,u,u'))$
end

- $G(s,c,x,u) \wedge (G(s,c,x,u) \Rightarrow R(s,c,x,u,u')) \Leftrightarrow G(s,c,x,u) \wedge R(s,c,x,u,u')$
- $A(s,c,u) \wedge I(s,c,u) \wedge BA(f)(s,c,u,u') \Rightarrow I(s,c,u') \wedge BA(e)(s,c,u,u')$

Synthesis Phase

- Preparing merging by transforming actions.

EVENT e
any
  x
where
  $G(s, c, x, u)$
then
  $u :| (R(s, c, x, u, u'))$
end

EVENT f refines e
any
  x
where
  $G(s, c, x, u)$
then
  $u :| (G(s, c, x, u) \Rightarrow R(s, c, x, u, u'))$
end

- $G(s, c, x, u) \wedge (G(s, c, x, u) \Rightarrow R(s, c, x, u, u')) \Leftrightarrow G(s, c, x, u) \wedge R(s, c, x, u, u')$
- $A(s, c, u) \wedge I(s, c, u) \wedge BA(f)(s, c, u, u') \Rightarrow I(s, c, u') \wedge BA(e)(s, c, u, u')$
- $A(s, c, u) \wedge I(s, c, u) \wedge G(s, c, x, u) \wedge (G(s, c, x, u) \Rightarrow R(s, c, x, u, u')) \Rightarrow I(s, c, u') \wedge G(s, c, x, u) \wedge R(s, c, x, u, u')$

Synthesis Phase: review of machines from M15-M20

MACHINE $M15$ **SEES** $C0, C1$
VARIABLES
$input, rs, re, cur, cs, ce, l,$

Synthesis Phase: review of machines from M15-M20

MACHINE $M16$ **SEES** $C0, C1$
VARIABLES
$input, rs, re, cur, cs, ce, l,$

Synthesis Phase: review of machines from M15-M20

MACHINE $M17$ **SEES** $C0, C1$
VARIABLES
$input, rs, re, cur, cs, ce, l,$

Synthesis Phase: review of machines from M15-M20

MACHINE $M18$ **SEES** $C0, C1$
VARIABLES
$input, rs, re, cur, cs, ce, l,$

Synthesis Phase: review of machines from M15-M20

MACHINE $M19$ **SEES** $C0, C1$
VARIABLES
$input, rs, re, cur, cs, ce, l,$

Synthesis Phase: review of machines from M15-M20

MACHINE $M20$ **SEES** $C0, C1$
VARIABLES
$input, rs, re, cur, cs, ce, l,$

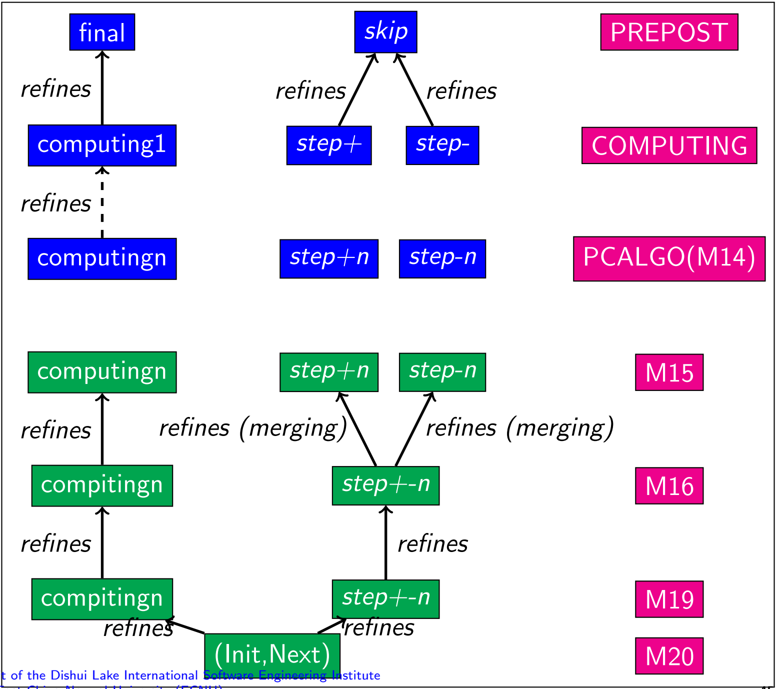
Generating a TLA specification

```
-----------------------------
next1 ==
      (l="start" /\   cur < n  /\ ( cur % 2 = 0))
      /\   (cur'=cur+1 /\  ce'=ce /\  l'=l
      /\  cs'=cs+cur+1 /\  re'=re /\  rs'=rs)
o
delsnext2 ==
      (l="start" /\   cur < n  /\ ( cur % 2 # 0))
      /\ (cur'=cur+1 /\  ce'=ce+cur+1 /\  l'=l
      /\  cs'=cs+cur+1 /\  re'=re /\ rs'=rs)

next3 ==
  (l="start" /\  cur=n) /\   (rs'=cs /\  re'=ce
  /\  l'="end" /\  cur'=cur /\ cs'=cs/\ ce'=ce)

Next ==
      (next1 \/ next2 \/ next3)

Init == l="start" /\   cur=0 /\ rs=0 /\ cs=0 /\ re=0 /\ ce =0
=============================================================
```



final
skip
PREPOST
refines
refines
refines
computing1
step+
step-
COMPUTING
refines
computingn
step+n
step-n
PCALGO(M14)
computingn
step+n
step-n
M15
refines
refines (merging)
refines (merging)
compitingn
step+-n
M16
refines
refines
compitingn
step+-n
M19
refines
refines
(Init,Next)
M20

Current Summary

Conclusion

- Paradigm for planning refinements.
- Teaching why and how sequential algorithms are working.
- Relating Event-B to TLA
- Application to controller synthesis: events versus operations.

Conclusion

- Paradigm for planning refinements.
- Teaching why and how sequential algorithms are working.
- Relating Event-B to TLA
- Application to controller synthesis: events versus operations.

Next

- Atlas of *correct-by-construction* sequential algorithms
- Definition of link between events and codes
- Integration of certifiction techniques for proofs.

Case studies

Case studies

- odd and even summation (gc-oddevensummation)
- power functions: x^n (bb-power3)
- primes
- binary search
- gcd
- fibonacci-like functions


Questions
???