Checking contracts in Event-B Reporting the introduction and the use of automated tools for verifying software-based systems in higher education

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

FMTea in Milan, Italy, September 10, 2024

Summary

Current Summary

Objectives

modelling, verifying, validating

Lectures on *modelling, designing, verifying and validating software-based systems* taught in the MsC *Computer Science* at Faculty of Science of the University of Lorraine and in the Computer Engineering Master of the School *Telecom Nancy* of the University of Lorraine.

- The epistemological concepts were given using the classical blackboard and chalk and progressively we have moved to integrate automated verification techniques and tools.
- Group of 50 students in +4 for regular students from a highly competive selection ...
- Group of 20 students in +4 for apprentice students (two months at school and two months at company)

Objectives

modelling, verifying, validating

Lectures on *modelling, designing, verifying and validating software-based systems* taught in the MsC *Computer Science* at Faculty of Science of the University of Lorraine and in the Computer Engineering Master of the School *Telecom Nancy* of the University of Lorraine.

- The epistemological concepts were given using the classical blackboard and chalk and progressively we have moved to integrate automated verification techniques and tools.
- Group of 50 students in $+4$ for regular students from a highly competive selection ...
- Group of 20 students in $+4$ for apprentice students (two months at school and two months at company)

Idea

To introduce progressively the concepts of verification using the Floyd-Hoare principle and to show how students can develop a tool for their pet programming language.

Ingredients for lectures

Ingredients for lectures

- **Our main reference is the work of Patrick and Radhia Cousot on induction principles for program properties.**

Ingredients for lectures

- **Our main reference is the work of Patrick and Radhia Cousot on induction principles for program properties.**
- **Tools are available for teaching as Frama-c, Rodin, TLA Toolbox, Z3, . . . but have a learning time.**
- Main concepts to be acquired:

Ingredients for lectures

- **Our main reference is the work of Patrick and Radhia Cousot on induction principles for program properties.**
- **Tools are available for teaching as Frama-c, Rodin, TLA Toolbox, Z3, . . . but have a learning time.**
- Main concepts to be acquired:
  - Abstraction

Ingredients for lectures

- **Our main reference is the work of Patrick and Radhia Cousot on induction principles for program properties.**
- **Tools are available for teaching as Frama-c, Rodin, TLA Toolbox, Z3, . . . but have a learning time.**
- Main concepts to be acquired:
  - Abstraction
  - Induction

Ingredients for lectures

- **Our main reference is the work of Patrick and Radhia Cousot on induction principles for program properties.**
- **Tools are available for teaching as Frama-c, Rodin, TLA Toolbox, Z3, . . . but have a learning time.**
- Main concepts to be acquired:
  - Abstraction
  - Induction
  - Fixed-point Theory

Ingredients for lectures

- **Our main reference is the work of Patrick and Radhia Cousot on induction principles for program properties.**
- **Tools are available for teaching as Frama-c, Rodin, TLA Toolbox, Z3, . . . but have a learning time.**
- Main concepts to be acquired:
  - Abstraction
  - Induction
  - Fixed-point Theory
  - Mathemarical notations for set theory

Ingredients for lectures

- **Our main reference is the work of Patrick and Radhia Cousot on induction principles for program properties.**
- **Tools are available for teaching as Frama-c, Rodin, TLA Toolbox, Z3, . . . but have a learning time.**
- Main concepts to be acquired:
  - Abstraction
  - Induction
  - Fixed-point Theory
  - Mathemarical notations for set theory
  - Logics and mechanized reasoning

Ingredients for lectures

- **Our main reference is the work of Patrick and Radhia Cousot on induction principles for program properties.**
- **Tools are available for teaching as Frama-c, Rodin, TLA Toolbox, Z3, . . . but have a learning time.**
- Main concepts to be acquired:
  - Abstraction
  - Induction
  - Fixed-point Theory
  - Mathemarical notations for set theory
  - Logics and mechanized reasoning
  - Computability

Ingredients for lectures

- **Our main reference is the work of Patrick and Radhia Cousot on induction principles for program properties.**
- **Tools are available for teaching as Frama-c, Rodin, TLA Toolbox, Z3, . . . but have a learning time.**
- Main concepts to be acquired:
  - Abstraction
  - Induction
  - Fixed-point Theory
  - Mathemarical notations for set theory
  - Logics and mechanized reasoning
  - Computability
  - Transition systems

Ingredients for lectures

- **Our main reference is the work of Patrick and Radhia Cousot on induction principles for program properties.**
- **Tools are available for teaching as Frama-c, Rodin, TLA Toolbox, Z3, . . . but have a learning time.**
- Main concepts to be acquired:
  - Abstraction
  - Induction
  - Fixed-point Theory
  - Mathemarical notations for set theory
  - Logics and mechanized reasoning
  - Computability
  - Transition systems

Ingredients for lectures

- **Our main reference is the work of Patrick and Radhia Cousot on induction principles for program properties.**
- **Tools are available for teaching as Frama-c, Rodin, TLA Toolbox, Z3, . . . but have a learning time.**
- Main concepts to be acquired:
  - Abstraction
  - Induction
  - Fixed-point Theory
  - Mathemarical notations for set theory
  - Logics and mechanized reasoning
  - Computability
  - Transition systems

First steps in FMTea...academic year 1983/1984

- Course Semantics and Logics for Programs: lectures 26 h (13 sessions 2 h) and tutorial 32,5 h (13 sessions 2h30)

First steps in FMTea. . . academic year 1983/1984

- Course Semantics and Logics for Programs: lectures 26 h (13 sessions 2 h) and tutorial 32,5 h (13 sessions 2h30)
- Course Logics for concepts of logics: lectures 26 h (13 sessions 2 h) and tutorial 32,5 h (13 sessions 2h30)

First steps in FMTea...academic year 1983/1984

- Course Semantics and Logics for Programs: lectures 26 h (13 sessions 2 h) and tutorial 32,5 h (13 sessions 2h30)
- Course Logics for concepts of logics: lectures 26 h (13 sessions 2 h) and tutorial 32,5 h (13 sessions 2h30)
- Derivation of proofs using the PAP tool

First steps in FMTea. . . academic year 1983/1984

- Course Semantics and Logics for Programs: lectures 26 h (13 sessions 2 h) and tutorial 32,5 h (13 sessions 2h30)
- Course Logics for concepts of logics: lectures 26 h (13 sessions 2 h) and tutorial 32,5 h (13 sessions 2h30)
- Derivation of proofs using the PAP tool **Pen And Paper**

First steps in FMTea... academic year 1983/1984

- Course Semantics and Logics for Programs: lectures 26 h (13 sessions 2 h) and tutorial 32,5 h (13 sessions 2h30)
- Course Logics for concepts of logics: lectures 26 h (13 sessions 2 h) and tutorial 32,5 h (13 sessions 2h30)
- Derivation of proofs using the PAP tool **Pen And Paper**
- Main tools are pen and paper !

Teaching Verification Techniques in 1983/1984

Third Year University Degree (List of courses)

RAPPEL DE L'ORGANISATION DES ENSEIGNEMENTS DE LA LICENCE SMI.

Matière	U.V.	Cours		T.D.	
Programmation PASCAL (semaine bloquée)	0	15	Perrin	25	de Bary
Architecture des Ordinateurs et langage d'assemblage	1/2	26	Perrin	34,5	Perrin
Algorithmique des Tables Arbres et Graphes	1/2	26	Cousot	34,5	Kaced
Informatique de Gestion	1/2	24	de Bary	32	de Bary
Bases de Données	1/2	24	de Bary	32	de Bary
2c Système d'Exploitation des Ordinateurs	1/2	24	Cousot	32	Mery
Analyse Numérique	1/2	19,5	Schmitt	39	Ginie d'Arnaud
Analyse	1	37,5	Rhin	75	Idt

Total 196 + 304 = 500 heures annu

Teaching Verification Techniques in 1983/1984

Fourth Year University Degree (List of courses)

ANNEXE 3. : ORGANISATION DES ENSEIGNEMENTS DE LA MAITRISE SMI. - 2/

Matière	U.V.	Cours		T.D.	
Logique Mathématique	1/2	26	Chauvin	34,5	Tyvaert
Sémantique et Logique des programmes	1/2	26	Cousot	34,5	Mery
Langages de Program-mation et Compilation	1/2	26	Cousot	39	Kaced
Mathématiques du contrôle	1/2	24	Sec	37,5	Sallet
Contrôle de Processus en temps réel	1/2	24	Perrin	32	Perrin
Téléinformatique	1/2	24	Cousot	32	Mery
Probabilités et Statis-tiques.	1	38	Rhin et Nasi	76	Sertour
		188	+	285,5	473,5 heures annuelles plus stage industrie

Teaching Verification Techniques in 1983/1984

Course Semantics and Logics for Programs

SEMANTIQUE ET LOGIQUE DES PROGRAMMES

- Sémantique opérationnelle des langages de programmation,
- Méthode de Floyd de preuve de propriétés d'invariance (correction partielle, absence d'erreur à l'exécution) des programmes impératifs séquentiels,
- Méthode axiomatique de Hoare,
- Preuves de terminaison utilisant un ordre bien fondé,
- Méthode des assertions intermittentes de Burstall pour démontrer la correction totale des programmes impératifs séquentiels,
- Correction et complétude des méthodes de preuve relatives à une sémantique opérationnelle,
- Démonstration de la correction de programmes applicatifs par induction

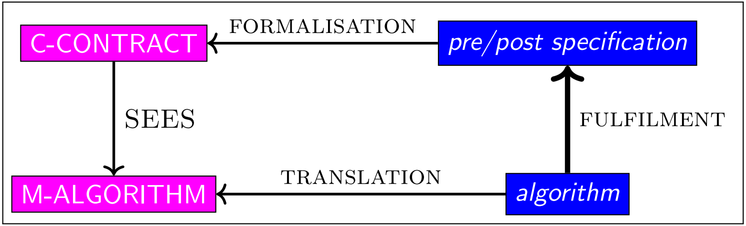
From PAP (Pen And Paper) to FIDE (Formal IDE)

- (1983-1993) Fruitful period for proof tools as Coq, Isabelle, ...
- The temporal framework and model checking techniques
- The B proposal and the use of proof assistants
- Experiment with techniques and tools
  - TLA tools
  - Rodin, Atelier-B
  - PAT
  - PRISM
  - Z3, CVC, ...
  - Frama-c
  - DAFNY

Sketch

The main steps of our method:

- FORMALISATION Expression of the contract as assertions defined in an Event-Bcontext.
- TRANSLATION Translation of annotations as elements of the invariant and of the basic computation steps between two successive labels as events.



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 $\mathtt{km/hr}$ 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?

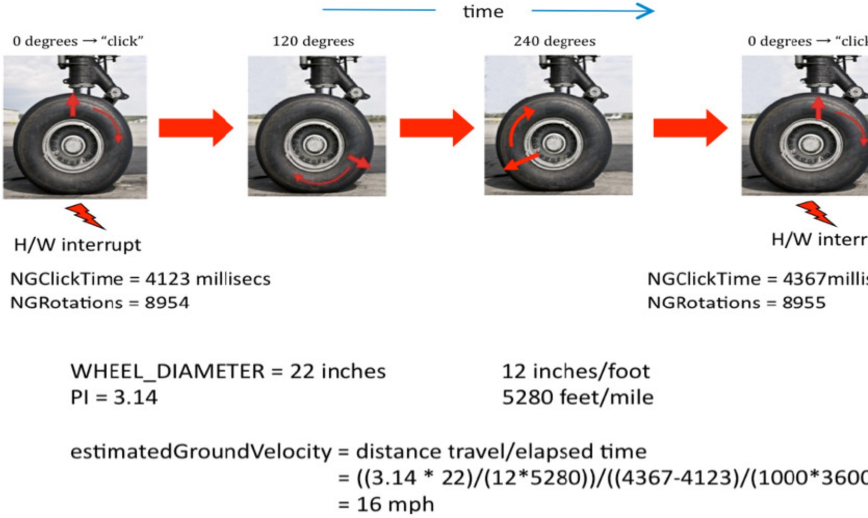
- Two different types of data:
  - counters with modulo semantics
  - non-negative values for time, distance, and velocity
- Two dimensions: *distance* and *time*
- Many units: distance (`inches`, `kilometers`, `miles`), time (`milliseconds`, `hours`), velocity (`kph`, `mph`)
- And interaction among 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 $3419 \leq= 65535$.

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 $VNGClick \in Int$

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

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

Current Summary

bug0.c

Listing 3: Bug bug0

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
    int x, y;
    // Seed the random number generator with the current time
    srand(time(NULL));
    // Generate a random number between 1 and 100
    x = rand() % 100 + 1;
    // Perform some calculations
    y = x / (100 - x);
    printf("Result: %d\n", y);
    return 0;
}
```

bug00.c

Listing 4: Bug bug00

```
// Heisenbug
#include <stdio.h>
#include <stdlib.h>
#include <time.h>



int main() {
  int x, y,i=0;

    for (i = 0; i <= 100000; i++) {
    // Seed the random number generator with the current time
    srand(time(NULL));

    // Generate a random number between 1 and 100
    x = rand() % 100 + 1;
        printf("Result: x= %d\n",x);
    // Perform some calculations
    y = x / (100 - x);

    printf("Result: i=%d  %d\n",i, y);
    }

    return 0;
}
```

bug000.c

Listing 5: Bug bug000

```
// Heisenbug
#include <stdio.h>
#include <stdlib.h>
#include <time.h>



int main() {
  int x, y,i=0;

    for (i = 0; i <= 100; i++) {
    // Seed the random number generator with the current time
    srand(time(NULL)+i);

    // Generate a random number between 1 and 100
    x = rand() % 100 + 1;
         printf("Result: x= %d\n",x);
    // Perform some calculations
    y = x / (100 - x);

    printf("Result: i=%d  %d\n",i, y);
    }

    return 0;
}
```

Current Summary

Verifying program correctness

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: $\mathsf{pre}(v_0)$ and v_f satisfies post : $\mathsf{post}(v_0, v_f)$
- $\mathsf{pre}(v_0) \wedge v_0 \xrightarrow{\mathsf{P}} v_f \Rightarrow \mathsf{post}(v_0, v_f)$
- $\mathbb{D}$ est le domaine RTE de V

Verifying program correctness

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: $v_0 \xrightarrow{\text{P}} v_f$
- v_0 satisfies pre: pre(v_0) and v_f satisfies post : post(v_0, v_f)
- pre$(v_0) \wedge v_0 \xrightarrow{\text{P}} v_f \Rightarrow$ 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

begin
$0 : P_0(v_0, v)$
$\mathtt{instruction}_0$
$\ldots$
$i : P_i(v_0, v)$
$\ldots$
$\mathtt{instruction}_{f-1}$
$f : P_f(v_0, v)$
end

- $pre(v_0) \wedge v = v_0 \Rightarrow P_0(v_0, v)$
- $pre(v_0) \wedge P_f(v_0, v) \Rightarrow post(v_0, v)$
- For any pair of labelsℓ, ℓ' such that $\ell \longrightarrow \ell'$, one verifies that, pour any values $v, v' \in \textsc{Memory}$

$$\left(\begin{array}{l} \left(\begin{array}{l} pre(v_0) \wedge P_\ell(v_0, v)) \\ \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \end{array} \right) \\ \Rightarrow P_{\ell'}(v_0, v') \end{array} \right),$$

Contracts - Verification Conditions

contract P
variables v
requires $pre(v_0)$
ensures $post(v_0, v_f)$

begin
$0 : P_0(v_0, v)$
$\mathtt{S}_0$
$\ldots$
$i : P_i(v_0, v)$
$\ldots$
$\mathtt{S}_{f-1}$
$f : P_f(v_0, v)$
end

Contracts - Verification Conditions

```
contract P
variables v
requires pre(v0)
ensures post(v0, vf)
    begin
    0 : P0(v0, v)
    S0
    ...
    i : Pi(v0, v)
    ...
    Sf-1
    f : Pf(v0, v)
    end
```

Verification conditions are listed as follows:

- (initialisation)
  $pre(v_0) \wedge v = v_0 \Rightarrow P_0(v_0, v)$
- (finalisation)
  $pre(v_0) \wedge P_f(v_0, v) \Rightarrow post(v_0, v)$
- (induction)
  For each labels pair ℓ, ℓ'
  such that $\ell \longrightarrow \ell'$, one checks that,
  for any value $v, v' \in \textsc{Memory}$
  $$\left(\begin{array}{l} \left(\begin{array}{l} pre(v_0) \wedge P_\ell(v_0, v)) \\ \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \end{array} \right) \\ \Rightarrow P_{\ell'}(v_0, v') \end{array} \right),$$

Three kinds of verification conditions should be checked and we justify the method in the full version..

From PAP to Rodin . . .

From PAP to Rodin . . .

MACHINE M
SEES $C0$
VARIABLES
v, pc
INVARIANTS
typing : $v \in D$
control : $pc \in L$
$\ldots$
atℓ : $pc = \ell \Rightarrow P_{\ell}(v0, v)$
$\ldots$
$th1 : pre(v_0) \wedge v = v_0 \Rightarrow P_0(v_0, v)$
$th2 : pre(v_0) \wedge P_f(v_0, v)$
$\Rightarrow post(v_0, v)$
$\ldots$
END
$\ldots$
END

From PAP to Rodin ...

MACHINE M
SEES $C0$
VARIABLES
v, pc
INVARIANTS
typing : $v \in D$
control : $pc \in L$
$\ldots$
atℓ : $pc = \ell \Rightarrow P_\ell(v0, v)$
$\ldots$
$th1 : pre(v_0) \wedge v = v_0 \Rightarrow P_0(v_0, v)$
$th2 : pre(v_0) \wedge P_f(v_0, v) \Rightarrow post(v_0, v)$
$\ldots$
END
$\ldots$
END

MACHINE M
EVENTS
INITIALISATION
BEGIN
$(pc, v) :| \left(\begin{array}{l} pc' = l0 \wedge v' = v0 \\ \wedge pre(v0) \end{array} \right)$
END
$\ldots$
$e(\ell, \ell')$
WHEN
$pc = \ell$
$cond_{\ell,\ell'}(v)$
THEN
$pc := \ell'$
$v := f_{\ell,\ell'}(v)$
END
$\ldots$
END

Technical problems for students

(Induction Principle (I))

A property $S(z0, z)$ is a safety for an annotated program P if, and only if, there exists a property $I(z0, z)$ satisfying:

1. $\forall z0, z \in \mathsf{L} \times \mathsf{D}.init(z0) \wedge z = z0 \Rightarrow I(z0, z)$
2. $\forall z0, z, z' \in \mathsf{L} \times \mathsf{D}.init(z0) \wedge I(z0, z) \wedge (z \underset{P}{\longrightarrow} z') \Rightarrow I(z0, z')$
3. $\forall z0, z \in \mathsf{L} \times \mathsf{D}.init(z0) \wedge I(z0, z) \Rightarrow S(z0, z)$

(Induction Principle (II))

A property $S(\ell 0, x0, \ell, x)$ is a safety property for an annotated program P if, and only if, there exists a property $I(\ell 0, x0, \ell, x)$ satisfying:

1. $\forall \ell 0, \in \mathsf{L}, x0 \in \mathsf{D}.\ell 0 \in \mathsf{L0} \wedge pre(x0) \wedge x = x0 \wedge pc = \ell 0 \Rightarrow J(\ell 0, x0, \ell, x)$
2. $\forall \ell, \ell' \in \mathsf{L}, x, x0 \in \mathsf{D}.\ell 0 \in \mathsf{L0} \wedge pre(x0) \wedge J(\ell 0, x0, \ell, x) \wedge$ $BA(e(\ell, \ell'),)(\ell, x, \ell', x') \Rightarrow J(\ell 0, x0, \ell', x')$
3. $\forall \ell 0, \ell \in \mathsf{L}, x0, x \in \mathsf{D}.pre(x0) \wedge \ell 0 \in$ $\mathsf{L0} \wedge J(\ell 0, x0, \ell, x) \Rightarrow S(\ell 0, x0, \ell, x)$

Technical problems for students

(Induction Principle (II))

A property $S(\ell 0, x0, \ell, x)$ is a safety property for an annotated program P if, and only if, there exists a property $I(\ell 0, x0, \ell, x)$ satisfying:

1. $\forall \ell 0, \in \mathsf{L}, x0 \in \mathsf{D}. \ell 0 \in \mathsf{L0} \wedge pre(x0) \wedge x = x0 \wedge pc = \ell 0 \Rightarrow J(\ell 0, x0, \ell, x)$
2. $\forall \ell, \ell' \in \mathsf{L}, x, x0 \in \mathsf{D}. \ell 0 \in \mathsf{L0} \wedge pre(x0) \wedge J(\ell 0, x0, \ell, x) \wedge BA(e(\ell, \ell'),)(\ell, x, \ell', x') \Rightarrow J(\ell 0, x0, \ell', x')$
3. $\forall \ell 0, \ell \in \mathsf{L}, x0, x \in \mathsf{D}. pre(x0) \wedge \ell 0 \in \mathsf{L0} \wedge J(\ell 0, x0, \ell, x) \Rightarrow S(\ell 0, x0, \ell, x)$

(Induction Principle (III))

A property $S(x0, \ell, x)$ is a safety for an annotated program P with one entry point if, and only if, there exists a property $I(x0, \ell, x)$ satisfying:

1. $\forall x0 \in \mathsf{D}. pre(x0) \wedge x = x0 \wedge \ell = \ell 0 \Rightarrow J(x0, \ell, x)$
2. $\forall \ell, \ell' \in \mathsf{L}, x, x0 \in \mathsf{D}. pre(x0) \wedge J(x0, \ell, x) \wedge BA(e(\ell, \ell'),)(\ell, x, \ell', x') \Rightarrow J(x0, \ell', x')$
3. $\forall \ell \in \mathsf{L}, x0, x \in \mathsf{D}. pre(x0) \wedge J(x0, \ell, x) \Rightarrow S(x0, \ell, x)$

Soundness of the translation

(Soundness of the method)

If the initialisation init, the generalisation gen and the step induction are proved to be correct by the Rodin platform, the property $S(x0, \ell, x)$ is a correct safety property for the program P. In particular, one can handle the partial correctness and the run time error safety properties.

Soundness of the translation

(Soundness of the method)

If the initialisation init, the generalisation gen and the step induction are proved to be correct by the Rodin platform, the property $S(x0, \ell, x)$ is a correct safety property for the program P. In particular, one can handle the partial correctness and the run time error safety properties.

- Contract and verification conditions are translated into Event-B and are discharged by Rodin and its provers.
- Verification conditions are derived from Floyd's method.
- Annotation as assertion

A short example

contract SIMPLE
variables x
requires $x_0 \in \mathbb{N}$
ensures $x_f = 0$
begin
$\ell_0 : \{0 \leq x \leq x_0 \wedge x_0 \in \mathbb{N}\}$
while $0 <$ x $\mathbf{do}$
  $\ell_1 : \{0 < x \wedge x \leq x_0 \wedge x_0 \in \mathbb{N}\}$
  x := x $- 1$;
od
$\ell_2 : \{x = 0\}$end

A short example

contract SIMPLE
variables x
requires $x_0 \in \mathbb{N}$
ensures $x_f = 0$
begin
$\ell_0 : \{0 \leq x \leq x_0 \wedge x_0 \in \mathbb{N}\}$
while $0 <$ x **do**
  $\ell_1 : \{0 < x \wedge x \leq x_0 \wedge x_0 \in \mathbb{N}\}$
  x := x − 1;
od
$\ell_2 : \{x = 0\}$end

INVARIANTS
$inv1 : x \in \mathbb{N}$
$inv2 : l \in L$
$inv3 : l = l0 \Rightarrow$
$0 \leq x \wedge x \leq x0 \wedge x0 \in \mathbb{N}$
$inv4 : l = l1 \Rightarrow$
$0 < x \wedge x \leq x0 \wedge x0 \in \mathbb{N}$
$inv5 : l = l2 \Rightarrow x = 0$
$requires : x0 \in \mathbb{N} \wedge x = x0$
$\Rightarrow x = x0 \wedge x0 \in \mathbb{N}$
$ensures : x = 0 \wedge x = x0$
$\Rightarrow x = 0$

Event $Init$
then
  $act1 : x := x0$
  $act2 : l := l0$

Event $el0l1$
when
  $grd1 : l = l0$
  $grd2 : 0 < x$
then
  $act1 : l := l1$

Event $el0l2$
when
  $grd1 : l = l0$
  $grd2 : \neg(0 < x)$
then
  $act1 : l := l2$

Event $el1l0$
when
  $grd1 : l = l1$
then
  $act1 : l := l0$
  $act2 : x := x - 1$

Current Summary

From Floyd to Hoare

From Floyd to Hoare

- $\forall x_f, x_0.\mathsf{pre}(x_0) \wedge x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$

From Floyd to Hoare

- $\forall x_f, x_0.\mathsf{pre}(x_0) \wedge x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_f, x_0.\mathsf{pre}(x_0) \Rightarrow x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$

From Floyd to Hoare

- $\forall x_f, x_0.\mathsf{pre}(x_0) \wedge x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_f, x_0.\mathsf{pre}(x_0) \Rightarrow x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_f, x_0.\mathsf{pre}(x_0) \Rightarrow x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$

From Floyd to Hoare

- $\forall x_f, x_0 . \mathsf{pre}(x_0) \wedge x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_f, x_0 . \mathsf{pre}(x_0) \Rightarrow x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_f, x_0 . \mathsf{pre}(x_0) \Rightarrow x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_0 . \mathsf{pre}(x_0) \Rightarrow \forall x_f . x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$

From Floyd to Hoare

- $\forall x_f, x_0.\mathsf{pre}(x_0) \wedge x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_f, x_0.\mathsf{pre}(x_0) \Rightarrow x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_f, x_0.\mathsf{pre}(x_0) \Rightarrow x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_0.\mathsf{pre}(x_0) \Rightarrow \forall x_f.x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_0.\mathsf{pre}(x_0) \Rightarrow [P]\mathsf{post}(x_0, x_f)$
- wlp calculus is introduced

From Floyd to Hoare

- $\forall x_f, x_0.\mathsf{pre}(x_0) \wedge x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_f, x_0.\mathsf{pre}(x_0) \Rightarrow x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_f, x_0.\mathsf{pre}(x_0) \Rightarrow x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_0.\mathsf{pre}(x_0) \Rightarrow \forall x_f.x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_0.\mathsf{pre}(x_0) \Rightarrow [P]\mathsf{post}(x_0, x_f)$
- wlp calculus is introduced
- $[x := e]P(x) = P[x \mapsto e]$

From Floyd to Hoare

- $\forall x_f, x_0.\mathsf{pre}(x_0) \wedge x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_f, x_0.\mathsf{pre}(x_0) \Rightarrow x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_f, x_0.\mathsf{pre}(x_0) \Rightarrow x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_0.\mathsf{pre}(x_0) \Rightarrow \forall x_f.x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_0.\mathsf{pre}(x_0) \Rightarrow [P]\mathsf{post}(x_0, x_f)$
- wlp calculus is introduced
- $[x := e]P(x) = P[x \mapsto e]$
- $[\text{if } b(x) \text{ then } S1 \text{ else } S2\ ]P(x) = b(x) \wedge [S1]P(x) \vee \text{ not } b(x)\ [S2]P(x)$

From Floyd to Hoare

- $\forall x_f, x_0.\text{pre}(x_0) \wedge x_0 \xrightarrow{\text{P}} x_f \Rightarrow \text{post}(x_0, x_f)$
- $\forall x_f, x_0.\text{pre}(x_0) \Rightarrow x_0 \xrightarrow{\text{P}} x_f \Rightarrow \text{post}(x_0, x_f)$
- $\forall x_f, x_0.\text{pre}(x_0) \Rightarrow x_0 \xrightarrow{\text{P}} x_f \Rightarrow \text{post}(x_0, x_f)$
- $\forall x_0.\text{pre}(x_0) \Rightarrow \forall x_f.x_0 \xrightarrow{\text{P}} x_f \Rightarrow \text{post}(x_0, x_f)$
- $\forall x_0.\text{pre}(x_0) \Rightarrow [P]\text{post}(x_0, x_f)$
- wlp calculus is introduced
- $[x := e]P(x) = P[x \mapsto e]$
- [if $b(x)$ then $S1$ else $S2$]$P(x) = b(x) \wedge [S1]P(x) \vee$ not $b(x)$ $[S2]P(x)$
- Frama-c uses the HOARE logic for defining the verification conditions as R. Leino in DAFNY.

From Floyd to Hoare

- $\forall x_f, x_0.\mathsf{pre}(x_0) \wedge x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_f, x_0.\mathsf{pre}(x_0) \Rightarrow x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_f, x_0.\mathsf{pre}(x_0) \Rightarrow x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_0.\mathsf{pre}(x_0) \Rightarrow \forall x_f.x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$
- $\forall x_0.\mathsf{pre}(x_0) \Rightarrow [P]\mathsf{post}(x_0, x_f)$
- wlp calculus is introduced
- $[x := e]P(x) = P[x \mapsto e]$
- $[\text{if } b(x) \text{ then } S1 \text{ else } S2\ ]P(x) = b(x) \wedge [S1]P(x) \vee \text{ not } b(x)\ [S2]P(x)$
- Frama-c uses the HOARE logic for defining the verification conditions as R. Leino in DAFNY.
- Questions of termination require the wp calculus . . .

Current Summary

Concluding Remarks

- Automatic proof of verification conditions of an Event-B model written by annotation.
- Learn Event-B language and set notation for the next year.
- Deepening of notations related to algorithms (endurant variables and perdurant variables) for preparing specific features of Frama-c.
- Next step: learning Frama-c

Concluding Remarks

- Automatic proof of verification conditions of an Event-B model written by annotation.
- Learn Event-B language and set notation for the next year.
- Deepening of notations related to algorithms (endurant variables and perdurant variables) for preparing specific features of Frama-c.
- Next step: learning Frama-c
- Feedbacks from students:

Concluding Remarks

- Automatic proof of verification conditions of an Event-B model written by annotation.
- Learn Event-B language and set notation for the next year.
- Deepening of notations related to algorithms (endurant variables and perdurant variables) for preparing specific features of Frama-c.
- Next step: learning Frama-c
- Feedbacks from students:
  - 10 % of studets are "trying to install *Rodin*, *TLAToolBox* and *Frama-c* the day before the practcal exam ... they complain on the fact that they have problems.

Concluding Remarks

- Automatic proof of verification conditions of an Event-B model written by annotation.
- Learn Event-B language and set notation for the next year.
- Deepening of notations related to algorithms (endurant variables and perdurant variables) for preparing specific features of Frama-c.
- Next step: learning Frama-c
- Feedbacks from students:
  - 10 % of studets are "trying to install *Rodin*, *TLAToolBox* and *Frama-c* the day before the practcal exam ... they complain on the fact that they have problems.
  - Effective problems with installing *Frama-c* on MacOS ...

Concluding Remarks

- Automatic proof of verification conditions of an Event-B model written by annotation.
- Learn Event-B language and set notation for the next year.
- Deepening of notations related to algorithms (endurant variables and perdurant variables) for preparing specific features of Frama-c.
- Next step: learning Frama-c
- Feedbacks from students:
  - 10 % of studets are "trying to install *Rodin*, *TLAToolBox* and *Frama-c* the day before the practcal exam . . . they complain on the fact that they have problems.
  - Effective problems with installing *Frama-c* on MacOS . . . making a virtual disk with required software . . .

Concluding Remarks

- Automatic proof of verification conditions of an Event-B model written by annotation.
- Learn Event-B language and set notation for the next year.
- Deepening of notations related to algorithms (endurant variables and perdurant variables) for preparing specific features of Frama-c.
- Next step: learning Frama-c
- Feedbacks from students:
  - 10 % of studets are "trying to install *Rodin*, *TLAToolBox* and *Frama-c* the day before the practcal exam ... they complain on the fact that they have problems.
  - Effective problems with installing *Frama-c* on MacOS ... making a virtual disk with required software ... but problems with weak student machines or with new mac on M ...

Concluding Remarks

- Automatic proof of verification conditions of an Event-B model written by annotation.
- Learn Event-B language and set notation for the next year.
- Deepening of notations related to algorithms (endurant variables and perdurant variables) for preparing specific features of Frama-c.
- Next step: learning Frama-c
- Feedbacks from students:
  - 10 % of studets are "trying to install *Rodin*, *TLAToolBox* and *Frama-c* the day before the practcal exam ... they complain on the fact that they have problems.
  - Effective problems with installing *Frama-c* on MacOS ... making a virtual disk with required software ... but problems with weak student machines or with new mac on M ...

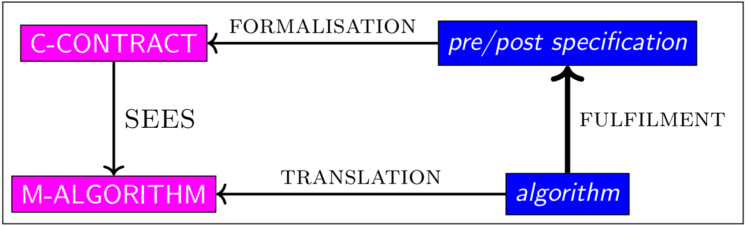
Concluding Remarks

- Automatic proof of verification conditions of an Event-B model written by annotation.
- Learn Event-B language and set notation for the next year.
- Deepening of notations related to algorithms (endurant variables and perdurant variables) for preparing specific features of Frama-c.
- Next step: learning Frama-c
- Feedbacks from students:
  - 10 % of studets are "trying to install *Rodin*, *TLAToolBox* and *Frama-c* the day before the practcal exam ... they complain on the fact that they have problems.
  - Effective problems with installing *Frama-c* on MacOS ... making a virtual disk with required software ... but problems with weak student machines or with new mac on M ...
- One teacher or more teachers

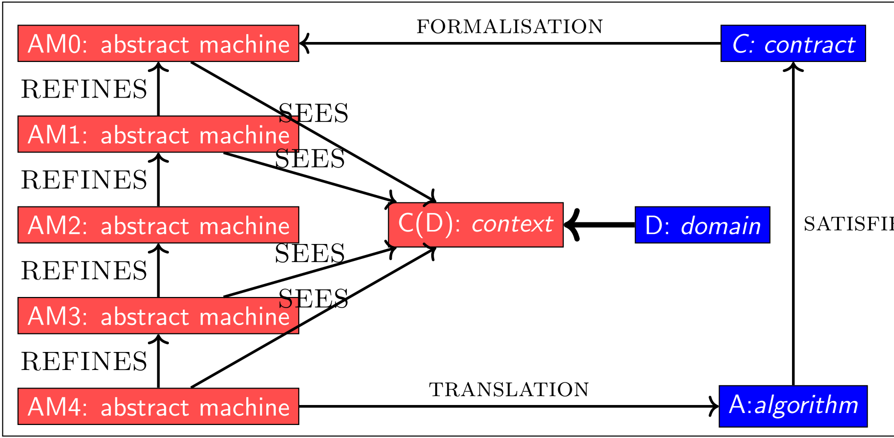
Sketch

The main steps of our method:

- FORMALISATION Expression of the contract as assertions defined in an Event-Bcontext.
- TRANSLATION Translation of annotations as elements of the invariant and of the basic computation steps between two successive labels as events.



Next year . . .



FORMALISATION
AM0: abstract machine
C: contract
REFINES
SEES
AM1: abstract machine
SEES
REFINES
AM2: abstract machine
C(D): context
D: domain
SATISFI
REFINES
SEES
AM3: abstract machine
SEES
REFINES
TRANSLATION
AM4: abstract machine
A:algorithm

Summary of concepts