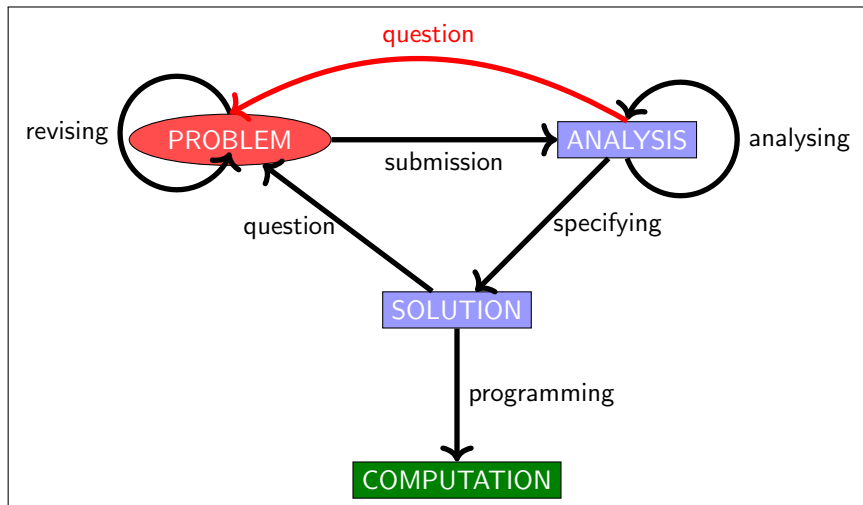


- 1 Tracking bugs in C codes
- 2 Introduction by Example
 - Detecting overflows in computations
 - Computing the velocity of an aircraft on the ground
- 3 Verification of program properties
- 4 Topics of course

Problem versus Solution



Listing 1 – 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;
}
```


Listing 3 – 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--and-y=%d\n", i, y);
    }

    return 0;
}
```

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--and-y=%d\n", i, y);
    }

    return 0;
}
```

**bug00.c prints ... Result : x= w1;
Result : i=100000 w2**

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 <= 200000; 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 -y=%d\n", i, y);
    }

    return 0;
}
```


Listing 6 – Bug bug000

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

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

    for (i = 0; i <= 200000; 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 -y=%d\n", i, y);
    }

    return 0;
}
```

Result: x= 70

Result: i=200000 2

Listing 8 – Bug bug1

```
#include <stdio.h>

int main() {
    int numbers[5] = {1, 2, 3, 4, 5};
    int sum = 0;

    // Attempt to calculate the sum of numbers in the array
    for (int i = 0; i <= 5; i++) {
        sum += numbers[i];
    }

    printf("Sum: %d\n", sum);
    sum = numbers[0];
    printf("Sum: %d\n", sum);

    return 0;
}
```

Sum: 16

Sum: 1

Listing 9 – Bug bug2

```
#include <stdio.h>

int main() {
    int x = 5;
    int y = 3;

    // Bug 1: Incorrect variable in the printf statement
    printf("The-value-of-x-is:-%d\n", y);

    // Bug 2: Infinite loop
    while (x > 0) {
        printf("x-is-greater-than-0\n");
    }

    return 0;
}
```

Listing 10 – Bug bug2

```
#include <stdio.h>

int main() {
    int x = 5;
    int y = 3;

    // Bug 1: Incorrect variable in the printf statement
    printf("The-value-of-x-is:-%d\n", y);

    // Bug 2: Infinite loop
    while (x > 0) {
        printf("x-is-greater-than-0\n");
    }

    return 0;
}
```

Infinite loop . . .

Listing 11 – Bug bug7

```
#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;
}
```

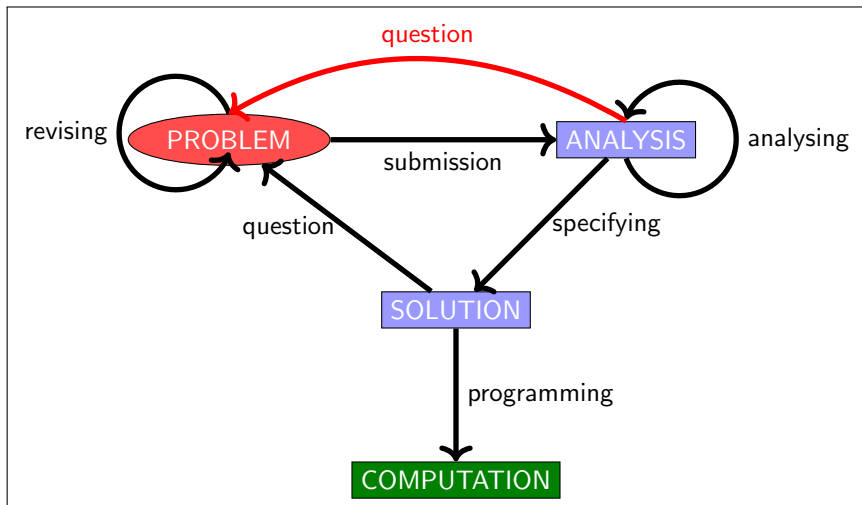
Listing 12 – Bug bug7

```
#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;
}
```

Average for 2147483647 and
2147483647 is -1

Problem versus Solution



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;
}
```

Listing 14 – 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);
}
```



- ▶ 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).

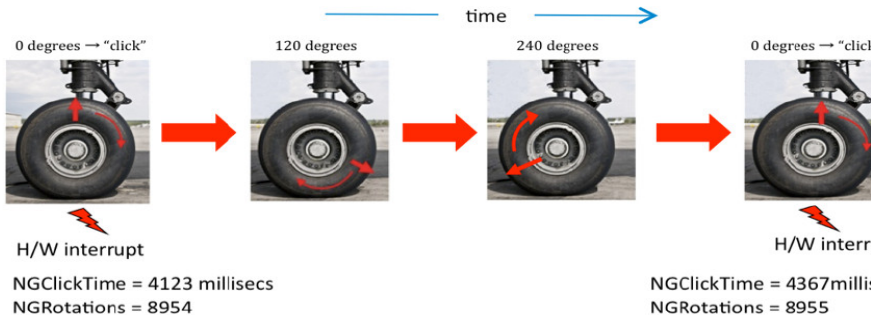
What are the main properties to consider for formalization ?

- ▶ 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 $estimateVelocity \in \text{MILES} \times \text{HOURS} \rightarrow \text{MPH}$
 - 2 $mphTokph \in \text{MPH} \rightarrow \text{KPH}$

Sample Velocity Estimation



WHEEL_DIAMETER = 22 inches
PI = 3.14

12 inches/foot
5280 feet/mile

estimatedGroundVelocity = distance travel/elapsed time
= $((3.14 * 22) / (12 * 5280)) / ((4367 - 4123) / (1000 * 3600))$
= 16 mph

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

- ▶ Validation : *Are we building the right product*
- ▶ Verification : *Are we building the process right?*

- ▶ Proving automatically $\text{REACHABLE}(M) \subseteq A$:

- ▶ Proving automatically $\text{REACHABLE}(M) \subseteq A$: undecidable ... no program is able to prove it automatically!
- ▶ Proving automatically $\text{REACHABLE}(M) \subseteq A$:

Decidability or Undecidability

- A problem $x \in P$ is generally stated by the function $\chi_{x \in P}$ where $\chi_{x \in P}(u) = 1$, if $P(u)$ is true and $\chi_{x \in P}(u) = 0$, if $P(u)$ is false :
- Problem 1 : $x \in 0..n$ where $n \in \mathbb{N}$
 - Problem 1 : $w \in \mathcal{L}(G)$ where G is a grammar over the finite set of alphabet symbols Σ and $\mathcal{L}(G) \subseteq \Sigma^*$.

Decidability or Undecidability

- ▶ A problem $x \in P$ is generally stated by the function $\chi_{x \in P}$ where $\chi_{x \in P}(u) = 1$, if $P(u)$ is true and $\chi_{x \in P}(u) = 0$, if $P(u)$ is false :
 - Problem 1 : $x \in 0..n$ where $n \in \mathbb{N}$
 - Problem 1 : $w \in \mathcal{L}(G)$ where G is a grammar over the finite set of alphabet symbols Σ and $\mathcal{L}(G) \subseteq \Sigma^*$.
- ▶ A problem $x \in P$ is decidable, when the function $\chi_{x \in P}$ is computable or more precisely the function can be computed by a program

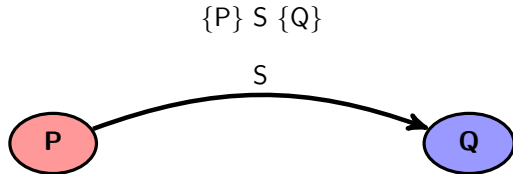
Implicite versus explicite

- Ecrire $101 = 5$ peut avoir une signification

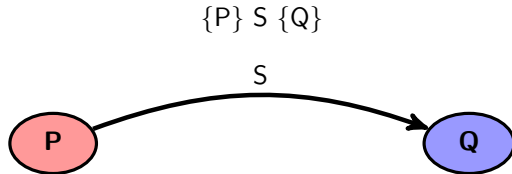
Example : description of static behaviour

- ▶ A train moving at absolute speed *spd1*
- ▶ A person walking in this train with relative speed *spd2*
 - One may compute the absolute speed of the person
- ▶ Modelling
 - Syntax. Classical expressions
 - ▶ Type *Speed* = *Float*
 - ▶ *spd1, spd2* : *Speed*
 - ▶ *AbsoluteSpeed* = *spd1* + *spd2*
 - Semantics
 - ▶ If *spd1* = 25.6 and *spd2* = 24.4 then *AbsoluteSpeed* = 50.0
 - ▶ If *spd1* = "val" and *spd2* = 24.4 then exception raised
 - Pragmatics
 - ▶ What if *spd1* is given in *mph* (miles per hour) and *spd2* in *km/s* (kilometers per second) ?
 - ▶ What if *spd1* is a relative speed ?

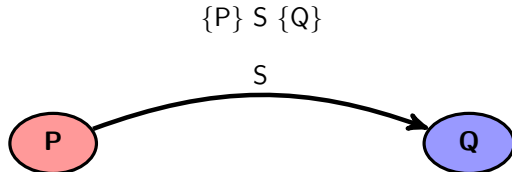
Asserted Program {P} S {Q}



Asserted Program $\{P\} S \{Q\}$

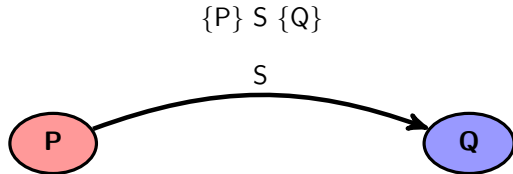


- ▶ $\{P\} S \{Q\}$: *asserted program*

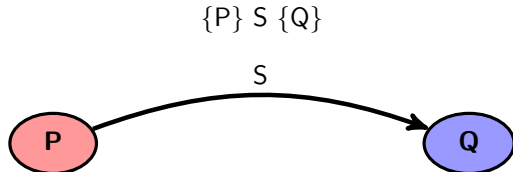


- ▶ $\{P\} S \{Q\}$: *asserted program*
- ▶ $P \Rightarrow WP(S)(Q)$: *logical formula*

Asserted Program $\{P\} S \{Q\}$



- ▶ $\{P\} S \{Q\}$: *asserted program*
- ▶ $P \Rightarrow WP(S)(Q)$: *logical formula*
- ▶ $C(S) \vdash P \Rightarrow WP(S)(Q)$: *logical formula*



- ▶ $\{P\} S \{Q\}$: *asserted program*
- ▶ $P \Rightarrow WP(S)(Q)$: *logical formula*
- ▶ $C(S) \vdash P \Rightarrow WP(S)(Q)$: *logical formula*
- ▶ $SP(S)(P) \Rightarrow Q$: *logical formula*
- ▶ $C(S) \vdash SP(S)(P) \Rightarrow Q$: *logical formula*

Predicate Transformer

$WP(S)(Q)$ is the Weakest-Precondition of S for Q and is a predicate transformer but $WP(S)(.)$ is not a computable function over the set of predicates.

- Découpage de l'unité : 10 cours de 2 h

Esquisse des cours, TDs et TPs

- ▶ Découpage de l'unité : 10 cours de 2 h
- ▶ Contenu :

- ▶ Découpage de l'unité : 10 cours de 2 h
- ▶ Contenu :
 - 😊 Principes de modélisation des systèmes informatiques : systèmes de transition

- ▶ Découpage de l'unité : 10 cours de 2 h
- ▶ Contenu :
 - 😊 Principes de modélisation des systèmes informatiques : systèmes de transition
 - 😊 Propriétés d'un système informatique : sûreté, vivacité, disponibilité, sécurité, dépendabilité

- ▶ Découpage de l'unité : 10 cours de 2 h
- ▶ Contenu :
 - 😊 Principes de modélisation des systèmes informatiques : systèmes de transition
 - 😊 Propriétés d'un système informatique : sûreté, vivacité, disponibilité, sécurité, dépendabilité
 - 😊 Modélisation de propriétés de systèmes
 - 😊 Analyse des programmes
 - 😊 Vérification de propriétés de systèmes avec un model checker TLC
 - 😊 Vérification de propriétés de systèmes avec un outil de preuve Rodin

- ▶ Découpage de l'unité : 10 cours de 2 h
- ▶ Contenu :
 - 😊 Principes de modélisation des systèmes informatiques : systèmes de transition
 - 😊 Propriétés d'un système informatique : sûreté, vivacité, disponibilité, sécurité, dépendabilité
 - 😊 Modélisation de propriétés de systèmes
 - 😊 Analyse des programmes
 - 😊 Vérification de propriétés de systèmes avec un model checker TLC
 - 😊 Vérification de propriétés de systèmes avec un outil de preuve Rodin
 - 😊 Vérification de propriétés de systèmes avec un analyseur Frama-c

