UNIVERSITÉ
DE LORRAINE

TELECOM
nancy
Ingénieurs du numérique • Inspiring your digital future

Cours MALG & MOVEX

Overview of the course

Dominique Méry

Telecom Nancy, Université de Lorraine

Année universitaire 2025-2026 (16 février 2026 10:08 A.M.)

Plan

Sommaire

Problem versus Solution

Current Summary

1 Tracking bugs in C codes

2 Introduction by Example
    Detecting overflows in computations
    Computing the velocity of an aircraft on the ground
    Computing the date of Easter
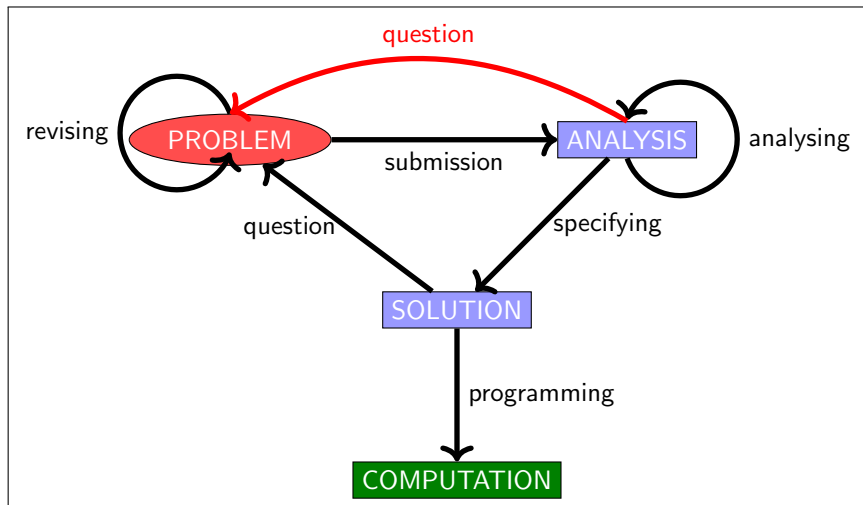   Computing how many days between two dates

3 Context and Objectives

4 Verification of program properties

5 stop movex 1

6 Topics of course

Listing 1 – Bug bug0

```c
#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;
}
```

bug0.c

Listing 2 – 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;
}
```

The statement $y = x / (100 - x)$; may lead to a messsage `Floating point exception: 8`. The Floating point exception : 8 message indicates that a program has triggered an arithmetic exception at system level (SIGFPE signal), causing it to stop abruptly.

bug00.c

Listing 3 – Bug bug00

```c
// Heisenbug
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
//
int main() {
    int x, y, i=0;
        for (i = 0; i <= 1000000; 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

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 <= 1000000; 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;
}
```

The code in bug0.c is repeated 1,000,000 times and does not detect any division by zero.

bug000.c

Listing 5 – Bug bug000

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


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

    for (i = 0; i <= INT_MAX; 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;
}
```

bug000.c

Listing 6 – Bug bug000

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


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

    for (i = 0; i <= INT_MAX; 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;
}
```

Iteration over the set of positive integers less than INT_MAX

Listing 7 – Bug bug1

```c
#include <stdio.h>
#define n 5
int main() {
    int numbers[n] = {1, 2, 3, 4, 5};
    int sum = 0;

    // Attempt to calculate the sum of numbers in the array
    for (int i = 0; i <= n; i++) {
        sum += numbers[i];
    };
    int val = n*(n+1)/2;
    printf("Iteratively-computed-sum=%d-and-direct-comutation-val=%d\n", sum, val);
    return 0;
}
```

bug1.c

Listing 8 – Bug bug1

```c
#include <stdio.h>
#define n 5
int main() {
    int numbers[n] = {1, 2, 3, 4, 5};
    int sum = 0;

    // Attempt to calculate the sum of numbers in the array
    for (int i = 0; i <= n; i++) {
        sum += numbers[i];
    };
    int val = n*(n+1)/2;
    printf("Iteratively-computed-sum=%d-and-direct-comutation-val=%d\n", sum, val);
    return 0;
}
```

Running bug1.c produces the output Iteratively computed
sum=17 and direct computation val=15 and the error is quite
simple. It is due to the definition of the size of the array and the
role of n.

bug2.c

Listing 9 – Bug bug2

```c
#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;
}
```

bug2.c

Listing 10 – Bug bug2

```c
#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;
}
```

The loop does not end because the condition never becomes false.
This example is fairly obvious, but loops may not end.

bbug5.c

Listing 11 – Bug bbug5

```c
#include <stdio.h>

int binarySearch(int arr[], int n, int target) {
    int left = 0;
    int right = n - 1;

    while (left <= right) {
        int mid = (left + right) / 2;

        //
        if (arr[mid] = target) {   //
            return mid;
        } else if (arr[mid] < target) {
            left = mid + 1;   //
        } else {
            //
        }
    }

    return -1;
}

int main() {
    int arr[] = {1, 2, 3, 4, 5, 5, 6, 7, 8, 9, 78};
    int n = sizeof(arr) / sizeof(arr[0]);

    int target = 10;

    int result = binarySearch(arr, n, target);

    if (result != -1) {
        printf("Element-%d-found-at-index-%d\n", target, result);
    } else {
        printf("Element-%d-not-found-in-the-array\n", target);
    }
    return 0;
}
```

The bbug5.c programme is compiled with the -w option and returns the following message : Element 10 found at index 5. Errors are accumulated and the -w option allows questions to be raised. et on tr

bug5.c

Listing 13 – Bug bug5

```c
#include <stdio.h>

int binarySearch(int arr[], int n, int target) {
    int left = 0;
    int right = n - 1;

    while (left <= right) {
        int mid = (left + right) / 2;

        // Bug 1: Incorrect comparison
        if (arr[mid] = target) {  // Should be '==' for comparison, not '='
            return mid;
        } else if (arr[mid] < target) {
            left = mid + 1;  // Bug 2: Update 'left' incorrectly
        } else {
            // Bug 3: Missing update for 'right' in the else case
        }
    }

    return -1;
}

int main() {
    int arr[] = {1, 2, 3, 4, 5, 5, 6, 7, 8, 9, 78};
    int n = sizeof(arr) / sizeof(arr[0]);

    int target = 10;

    int result = binarySearch(arr, n, target);

    if (result != -1) {
        printf("Element-%d-found-at-index-%d\n", target, result);
    } else {
        printf("Element-%d-not-found-in-the-array\n", target);
    }

    return 0;
```

The code is annotated with questions and answers.

Listing 15 – Bug bug6

```c
#include <stdio.h>
#include <limits.h>

int divide(int a, int b) {
    if (b != 0) {
        return a / b;   // Bug: Does not handle overflow
    } else {
        return 0;
    }
}

int main() {
    int x = INT_MIN;
    int y = -1;
    int result;
    result = 2 + x;
    printf("Result-for-result-=-2-+-x:-%d\n", result);
    result = divide(x, y);
    printf("Result-for-result-=-divide(x,-y):-%d\n", result);
    return 0;
}
```

bug6.c

Listing 16 – Bug bug6

```c
#include <stdio.h>
#include <limits.h>

int divide(int a, int b) {
    if (b != 0) {
        return a / b;  // Bug: Does not handle overflow
    } else {
        return 0;
    }
}

int main() {
    int x = INT_MIN;
    int y = -1;
    int result;
    result = 2 + x;
    printf("Result-for-result-=-2---x:-%d\n", result);
    result = divide(x, y);
    printf("Result-for-result-=-divide(x,-y):-%d\n", result);
    return 0;
}
```

Division by zero is considered and a condition `guard` the arithmetic operations, but division by -1 amounts to calculating the value —INT_MIN— = INT_MIN*(-1).

Listing 17 – Bug bug6bis

```c
#include <stdio.h>
#include <limits.h>
#include <assert.h>

int divide(int a, int b) {
    if (b != 0) {
        return a / b;  // Bug: Does not handle overflow
    } else {
        return 0;
    }
}

int main() {
    int x = INT_MIN;
    int y = -1;
    int result = divide(x,y);
    printf("Result - for - result -= - divide (x,-y): -%d\n", result);
    printf("Value - of -x=%d - and - value - of - y=%d\n",x,y);
    printf("Property - -x -= - - result *y -+ - remainder\n");
    int test = result*y;
    printf("Value - of - test -= - result *y -%d\n", test);
    assert( x == result*y);
    printf("Property - -x -= - - result *y -+ - remainder - is - tested .... -\n");

    return 0;
}
```

Problems are caused by memory overflows and lead to strange results.

Division by zero is considered and a condition `guard` the arithmetic operations, but division by -1 amounts to calculating the value INT_MIN = INT_MIN*(-1).

▶

bug6ter.c

Listing 19 – Bug bug6ter

```c
#include <limits.h>

int divide(int a, int b) {
    if (b != 0) {
        return a / b;
    } else {
        return 0;
    }
}
```

Listing 22 – Using frama-c bug6ter.c

```
[kernel] Parsing bug6ter.c (with preprocessing)
int divide(int a, int b)
{
  int __retres;
  if (b != 0) {
    /*@ assert rte: division_by_zero: b /= 0; */
    /*@ assert rte: signed_overflow: a / b <= 2147483647; */
    __retres = a / b;
    goto return_label;
  }
  else {
    __retres = 0;
    goto return_label;
  }
  return_label: return __retres;
}
```

▶

Examples of bugs in C codes

▶ bug0.c prints *Result : 1*

Examples of bugs in C codes

- bug0.c prints *Result : 1*
- bug00.c prints *... Result : x= 94 ; Result : i=100000 15*

Examples of bugs in C codes

▶ bug0.c prints *Result : 1*

▶ bug00.c prints . . . *Result : x= 94 ; Result : i=100000 15*

▶ bug000.c prints *Floating point exception : 8* and is a version modiified from bug00.c : srand(time(NULL)+i) ;

Examples of bugs in C codes

▶ bug0.c prints *Result : 1*

▶ bug00.c prints ... *Result : x= 94 ; Result : i=100000 15*

▶ bug000.c prints *Floating point exception : 8* and is a version modiified from bug00.c : srand(time(NULL)+i) ;

▶ *bug1.c* functional bug

Examples of bugs in C codes

- ▶ bug0.c prints *Result : 1*
- ▶ bug00.c prints ... *Result : x= 94 ; Result : i=100000 15*
- ▶ bug000.c prints *Floating point exception : 8* and is a version modiified from bug00.c : srand(time(NULL)+i) ;
- ▶ *bug1.c* functional bug
- ▶ *bug2.c* stupid bug of writing a code

Examples of bugs in C codes

- ▶ bug0.c prints *Result : 1*
- ▶ bug00.c prints *... Result : x= 94 ; Result : i=100000 15*
- ▶ bug000.c prints *Floating point exception : 8* and is a version modiified from bug00.c : srand(time(NULL)+i) ;
- ▶ *bug1.c* functional bug
- ▶ *bug2.c* stupid bug of writing a code
- ▶ bug5.c prints *Element 10 found at index 5*

Examples of bugs in C codes

- bug0.c prints *Result : 1*
- bug00.c prints ... *Result : x= 94 ; Result : i=100000 15*
- bug000.c prints *Floating point exception : 8* and is a version modiified from bug00.c : srand(time(NULL)+i);
- *bug1.c* functional bug
- *bug2.c* stupid bug of writing a code
- bug5.c prints *Element 10 found at index 5*
- bug6.c produces overflows ... and errors in arithmetic operations.

Current Summary

1 Tracking bugs in C codes

2 Introduction by Example
   Detecting overflows in computations
   Computing the velocity of an aircraft on the ground
   Computing the date of Easter
   Computing how many days between two dates

3 Context and Objectives

4 Verification of program properties

5 stop movex 1

6 Topics of course

Current Subsection Summary

1 Tracking bugs in C codes

2 Introduction by Example
   Detecting overflows in computations
   Computing the velocity of an aircraft on the ground
   Computing the date of Easter
   Computing how many days between two dates

3 Context and Objectives

4 Verification of program properties

5 stop movex 1

6 Topics of course

Detecting overflows of computations

Listing 23 – Function average

```c
#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 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 24 – Function average.....

```c
#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 Subsection Summary

1 Tracking bugs in C codes

2 Introduction by Example

3 Context and Objectives

4 Verification of program properties

5 stop movex 1

6 Topics of course

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

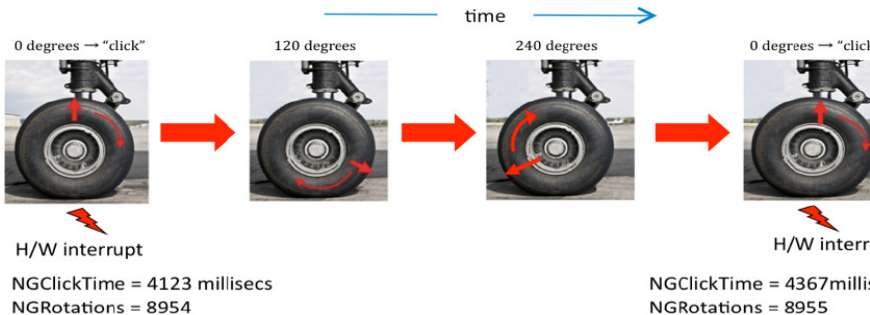- Note : expressed functional requirement is in *SI* unit system (km/hr).

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. $estimateVelocity \in \texttt{MILES} \times \texttt{HOURS} \rightarrow \texttt{MPH}$
  2. $mphTokph \in \texttt{MPH} \rightarrowtail \texttt{KPH}$

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 $VNGClick \in Int$

- $Length = \pi \cdot diameter \cdot VNGClick$ (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
  $3419 \leq 65535$

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 $Int \widehat{=} INT\_MIN..INT\_MAX$.

▶ 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 Subsection Summary

1 Tracking bugs in C codes

2 Introduction by Example

   Detecting overflows in computations

   Computing the velocity of an aircraft on the ground

   Computing the date of Easter

   Computing how many days between two dates

3 Context and Objectives

4 Verification of program properties

5 stop movex 1

6 Topics of course

Computing the date of easter

Easter is the Sunday following the first full moon of spring, that is, according to the definition established by the Council of Nicea in 325 :
"Easter is the Sunday following the 14th day of the Moon which reaches that age on or immediately after March 21."
According to this definition, Easter falls between March 22 and April 25 of each year.

▶ Easter date 2019 : 20 march / 21 avril
▶ Easter date 2020 : 7 april / 12 april
▶ Easter date 2021 : 28 march / 4 april
▶ Easter date 2022 : 16 april / 17 april 2022
▶ Easter date 2023 : 6 april / 9 april 2023
▶ Easter date 2024 : 25 march / 31 march 2024
▶ Easter date 2025 : 12 april / 20 april 2025
▶ Easter date 2026 : 2 april / 5 april 2026

Computing the date of easter

- The year is given as a number `year`
- The date of Easter is computed as a date `day` and `month`

General Protocol for Computing the Date of Easter

- Easter is a Sunday.
- Date of the spring full moon `full-moon-day` and `full-moon-month` from the 21st of March.
- `full-moon-month` is either 3 or 4.
- The earliest date if March 22nd
- Determine the First Sunday just after the spring full moon

Using the C function

Listing 25 – Using C function for date of Easter

```
*********************
* year * day * month *
*********************
* 2019 * 21 * April *
*********************
* 2020 * 12 * April *
*********************
* 2021 *  4 * April *
*********************
* 2022 * 17 * April *
*********************
* 2023 *  9 * April *
*********************
* 2024 * 31 * March *
*********************
* 2025 * 20 * April *
*********************
* 2026 *  5 * April *
*********************
*********************
```

Analyzing the protocol and its requirements

- The day of Spring namely March 21 : $dspring = (21,3)$ (**domain property**)
- The table of Full Moon of the current year : $tfullmoon[NFM]$ (**domain property**))
- Computing the date of the first full moon of Spring : $dfullmoon = tfullmoon[i]$ where $i \in 0..NFM-1$ (**algorithmic property**)
- Computing the date of the next Sunday starting from $nextday(dfullmoon)$ (**algorithmic property**)

Tryptich $\mathcal{D}, \mathcal{A}$ sat $\mathcal{R}$

- $\mathcal{D}$ (domain of problem) contains informations on Moon Phases.
- $\mathcal{R}$ (requirements or prescription) expresses the day of Easter following the rule.
- $\mathcal{A}$ (algorithm or program or protocol) contains the different algorithms used for computing necessary informations.

Verification and Validation

- ▶ Verification : techniques and tools applied to check the *internal* relationship between the three components $\mathcal{D}, \mathcal{A}, \mathcal{R}$
- ▶ Validation : techniques and tools applied to check the *external* relationship between the three components $\mathcal{D}, \mathcal{A}, \mathcal{R}$ and external entities.

Current Subsection Summary

1

Computing the number of days for waiting a given future date ?

You may want to gt the number of days between the current day and a future day :

► Number of days until Christmas

► Number of days until Retirement

► Notification of an event three days before a given date.

► . . .

What is the next day of agiven date ?

▶ Domain Properties related to dates and calendars :
  - A year is made up of 12 months
  - There are two kinds of months 30 days and 31 days but one month has 28 days and 29 days when the year is a leapyear.

▶ Mathematical definition of a leapyear

Listing 26 – pre/post specification for leapyear(int year)

```
#ifndef _LEAP_H

/*@ requires −5000 <= year;
  @ ensures year % 400 == 0  =⇒ \result == 1;
  @ ensures year % 400 != 0  && year % 100 == 0  && year % 4 == 0 =⇒ \result == 1;
  @ ensures year % 400 != 0  && year % 100 != 0  && year % 4 != 0 =⇒ \result == 0;
  @ ensures \result == 1 =⇒(year % 400 == 0 ) || ( year % 400 != 0 && year % 100 != 0 && year % 4
  @ ensures \result == 0 =⇒(year % 400 != 0) && ( year % 400 == 0 || year % 100 == 0 || year % 4
 */
int leapyear(int year);

#endif
```

Defining leapyear(int year)

Listing 27 – Using C function for date of Easter

```c
#include "leapyear.h"

  int leapyear(int year)
{
  int r=0;// = 0;
// leap year if perfectly divisible by 400
  if (year % 400 == 0) {
    r = 1; //printf("%d is a leap year.", year);
  }
  // not a leap year if divisible by 100
  // but not divisible by 400
  else if (year % 100 == 0) {
    r=0;// printf("%d is not a leap year.", year);
  }
  // leap year if not divisible by 100
  // but divisible by 4
  else if (year % 4 == 0) {
    r=1; // printf("%d is a leap year.", year);
  }
  // all other years are not leap years
  else {
    r=0; // printf("%d is not a leap year.", year);
  };

  return r;
}
```

Computing a sequence of dates

▶ Computing the number of days from `fromdate` till `todate` is following a sequence :

$$d_0 = fromdate \overset{nextday}{\longrightarrow} d_1 \overset{nextday}{\longrightarrow} d_2 \ldots \overset{nextday}{\longrightarrow} d_{n-1} \overset{nextday}{\longrightarrow} d_n = todate$$

▶ The value pf n is not known and is in fact the number we want to compute.

▶ Define a function `nextday` which is returning the next day of a given day.

▶ A date is a triple as (d,m,y) and is satisfying the function :

Listing 28 – checkdate(struct date now)

```
#include "structure.h"
#include "leapyear.h"
struct date;
#include "checkdate.h"

int checkdate(struct date now)
{
int r=0;
int day=now.day, month=now.month, year=now.year, aleapday;
int  aleapyear=leapyear(year);
    if ( month == 12 || month == 1 || month == 3 || month == 5 || month == 7 || month == 8 || month
    if (1 <= day && day <= 31) {r=1;};
    } else if (month == 4 || month == 6 || month == 9 || month == 11) {
    if (1 <= day && day <= 30) {r=1;};
    } else if ( aleapyear==1 && month == 2 ) {
    if (1 <= day && day <= 29) {r=1;};
    } else if ( aleapyear=0 && month == 2) {
    if (1 <= day && day <= 28) {r=1;}; };
    return r;
```

nextday(struct date now)

Listing 29 – nextday(struct date now)

```
#include "structure.h"
#include "leapyear.h"
#include "checkdate.h"
struct date;
#include "nextday.h"

struct date  nextday(struct date now)
{

  struct date r;
int  day=now.day, month=now.month, year=now.year, aleapday;
int  aleapyear=leapyear(year);
int  nday=day, nmonth=month, nyear=year;
      if ( month == 12 || month == 1 || month == 3 || month == 5 || month == 7 || month == 8 || month == 10
       if (day <31) {nday=day+1;} else if (day == 31 && month != 12) {nday=1;nmonth=month+1;} else {nday=1
      } else if (month == 4 || month == 6 || month == 9 || month == 11) {
       if (day <30) {nday=day+1;} else if (day == 30 ) {nday=1;nmonth=month+1;};
      } else if ( aleapyear==1 && month == 2 && day == 29 ) {nday=1; nmonth=3;}
      else if ( aleapyear==1 && month == 2 && day != 29 ) {nday=day+1;}
      else if ( aleapyear==0 && month == 2 && day == 28 ) {nday=1; nmonth=3;}
      else if ( aleapyear==0 && month == 2 && day != 28 ) {nday=day+1;}
      else   {nday=day + 1;}

      r.day = nday;
      r.month = nmonth;
      r.year = nyear;
      return r;
}
```

int daytilldate(struct date currentdate,struct date futuredate)

Listing 30 – int daytilldate(struct date currentdate,struct date futuredate)

```c
#include "structure.h"
#include "leapyear.h"
#include "checkdate.h"
#include "nextday.h"
struct date;
#include "daytilldate.h"

int daytilldate(struct date currentdate, struct date futuredate)
{
    int r=0;
    struct date d=currentdate;
    while (d.day != futuredate.day || d.month != futuredate.month || d.year != futuredate.year) {
        r++;
        d = nextday(d);
    };

    return r;
}
```

Current Summary

1 Tracking bugs in C codes

2 Introduction by Example

Detecting overflows in computations

Computing the velocity of an aircraft on the ground

Computing the date of Easter

Computing how many days between two dates

3 Context and Objectives

4 Verification of program properties

5 stop movex 1

6 Topics of course

Context and Objectives

- ▶ Software Systems assist our dayly lifes
- ▶ Questions on dependability and security assurance shoud be addressed
- ▶ Questions on certification with respect to norms and standards
- ▶ Improving the life-cycle development for addressing these questions

Problem Definition

Critical System

Critical systems are systems in which defects could have a dramatic impact on human life or the environment.

Critical System

Critical systems are systems in which defects could have a dramatic impact on human life or the environment.

System failure

Software failure or fault of complex systems is the major cause in the software crisis. For example,

▶ Therac-25 (1985-1987) : six people overexposed through radiation.

▶ Cardiac Pacemaker (1990-2002) : 8834 pacemakers were explanted.

▶ Insulin Infusion Pump (IIP) (2010) : 5000 adverse events.

▶ Safety-critical systems : A system whose failure may result in injury, loss of life or serious environmental damage. **An example of a safety-critical system is a control system for a chemical manufacturing plant.**

▶ Mission-critical systems : A system whose failure may result in the failure of some goal-directed activity. **An example of a mission-critical system is a navigational system for a spacecraft.**

▶ Business-critical systems : A system whose failure may result in very high costs for the business using that system. **An example of a business-critical system is the customer accounting system in a bank.**

Issue

The high costs of failure of critical systems means that trusted methods and techniques must be used for development.

Legacy systems

- Legacy systems are socio-technical computer-based systems that have been developed in the past, often using older or obsolete technology.
- Legacy systems include not only hardware and software but also legacy processes and procedures; old ways of doing things that are difficult to change because they rely on legacy software. Changes to one part of the system inevitably involve changes to other components.
- Legacy systems are often business-critical systems. They are maintained because it is too risky to replace them.
- For example, for most banks the customer accounting system was one of their earliest systems.

Software Process

A software process is a structured set of activities required to develop a software system : Spiral Model, Waterfall Model, V-Shaped Model, etc.

A software process involve the following steps :

- ▶ Specification *defining what the system should do ;*
- ▶ Design and implementation *defining the organization of the system and implementing the system ;*
- ▶ Validation *checking that it does what the customer wants ;*
- ▶ Evolution *changing the system in response to changing customer needs.*

software process model

A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective.

Software Process Models

- ▶ The waterfall model : Plan-driven model ; Separate and distinct phases of specification and development.
- ▶ Incremental development : Specification, development and validation are interleaved ; plan-driven or agile.
- ▶ Reuse-oriented software engineering : The system is assembled from existing components ; plan-driven or agile.

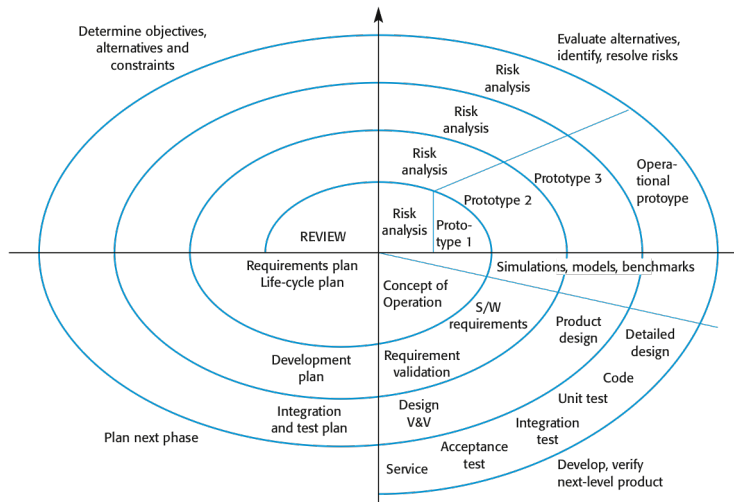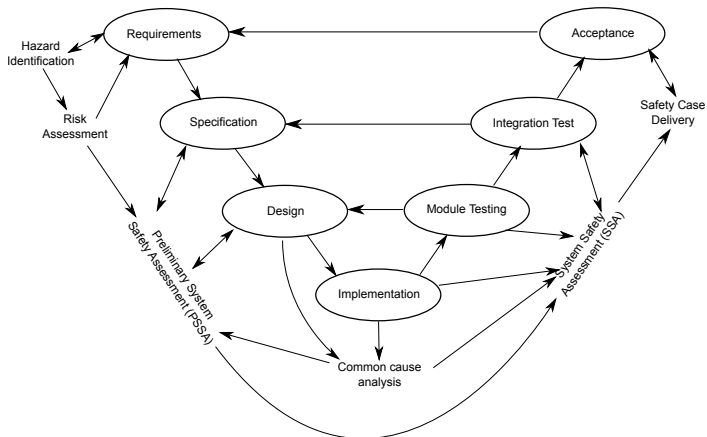**(from Software Engineering by Ian Sommerville)**
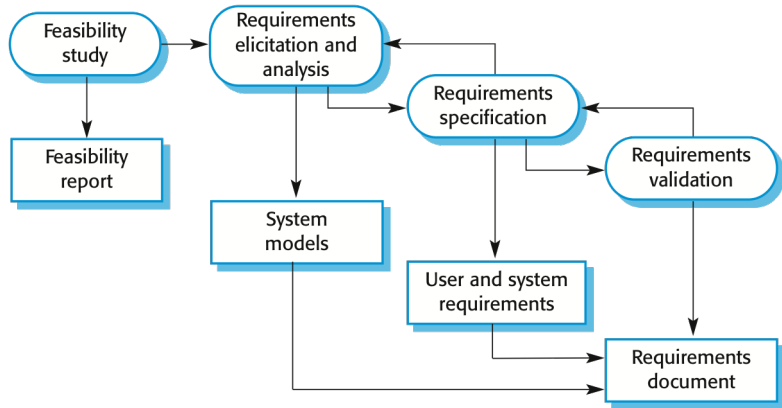
(from Software Engineering by Ian Sommerville)

Reuse-oriented software engineering

(from Software Engineering by Ian Sommerville)

Boehm's spiral model of the software process

Determine objectives, alternatives and constraints

Evaluate alternatives, identify, resolve risks

Risk analysis

Risk analysis

Risk analysis

Risk analysis

Proto-type 1

Prototype 2

Prototype 3

Opera-tional prototype

REVIEW

Requirements plan
Life-cycle plan

Concept of Operation

Simulations, models, benchmarks

S/W requirements

Product design

Detailed design

Development plan

Requirement validation

Code

Unit test

Integration and test plan

Design V&V

Integration test

Plan next phase

Acceptance test

Service

Develop, verify next-level product

(from Software Engineering by Ian Sommerville)

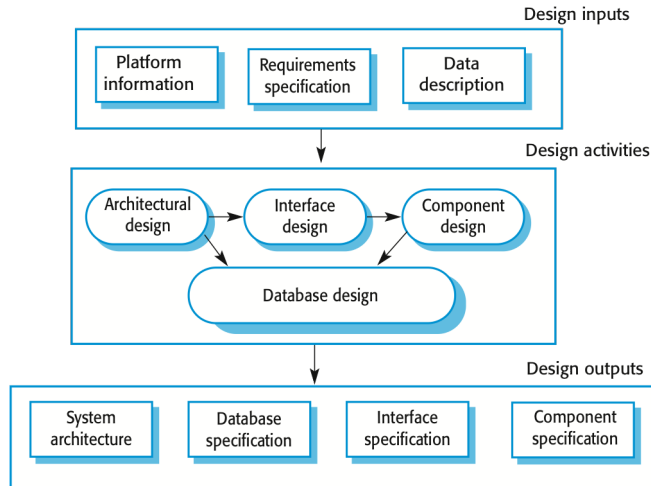**(from Software Engineering by Ian Sommerville)**

(from Software Engineering by Ian Sommerville)

Requirement Engineering

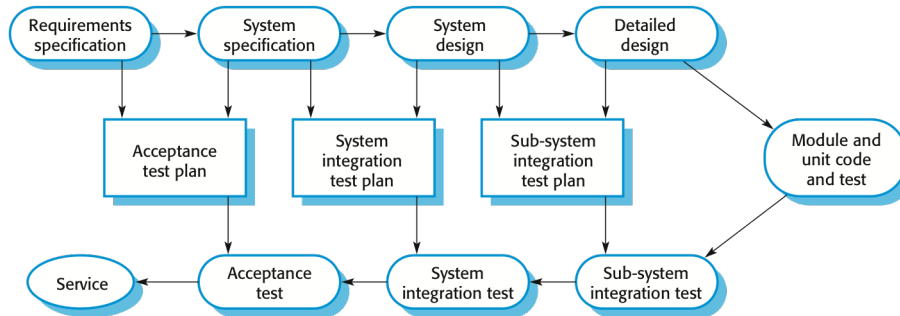The process of establishing what services are required and the constraints on the system's operation and development.

▶ Feasibility study : Is it technically and financially feasible to build the system ?

▶ Requirements elicitation and analysis : What do the system stakeholders require or expect from the system ?

▶ Requirements specification : Defining the requirements in detail

▶ Requirements validation : Checking the validity of the requirements

Design inputs

Design activities

Design outputs

(from Software Engineering by Ian Sommerville)

▶ Verification and validation (V & V) : intended to show that a system conforms to its specification and meets the requirements of the system customer ; involves checking and review processes and system testing.

▶ System testing : involves executing the system with test cases that are derived from the specification of the real data to be processed by the system.

▶ Testing is the most commonly used V & V activity


```
Component        System testing        Acceptance
testing                                 testing
```

(from Software Engineering by Ian Sommerville)

▶ Verification of software with respect to contract : **posthoc verification, semantic analysis, refinement-based methodology. . .**

Testing phases in a plan-driven software process

(from Software Engineering by Ian Sommerville)

Safety integrity level (SIL)

SIL

Safety integrity level (SIL) is defined as a relative level of risk-reduction provided by a safety function, or to specify a target level of risk reduction. In simple terms, SIL is a measurement of performance required for a safety instrumented function (SIF).

▶ The European functional safety standards based on the IEC 61508 standard defines four SILs (with SIL 4 the most dependable and SIL 1 the least).

▶ A SIL is determined based on a number of quantitative factors in combination with qualitative factors such as development process and safety life cycle management.

▶ Probability of Failure on Demand during 10 years.

▶ SILs :

- SIL4 : Conséquence très importante sur la communauté entraînant une réduction du danger de 10 000 à 100 000.
- SIL3 : Conséquence très importante sur la communauté et les employés entraînant une réduction du danger de 1 000 à 10 000.
- SIL2 : Protection importante de l'installation, de la production et des employés entraînant une réduction du danger de 100 à 1000.
- SIL1 : Faible protection de l'installation, de la production entraînant une réduction du danger de 10 à 100.

Grâce à une grande maîtrise en calcul formel, en sécurité de fonctionnement et à l'utilisation de la méthode B (beaucoup utilisée en milieu industriel pour réaliser des logiciels sécuritaires prouvés), ClearSy System Engineering est qualifiée pour mener à bien des projets nécessitant un contexte de certification de niveau SIL 2, SIL 3, ou SIL 4, selon la norme 61508.

Current Summary

1 Tracking bugs in C codes

2 Introduction by Example
   Detecting overflows in computations
   Computing the velocity of an aircraft on the ground
   Computing the date of Easter
   Computing how many days between two dates

3 Context and Objectives

4 Verification of program properties

5 stop movex 1

6 Topics of course

▶ Typing Properties using Typechecker (see for instance functional programming languages as ML, CAML, OCAML, . . .)

▶ Invariance and safety (A *nothing bad will happen !*) properties for a program P :

- Transformation of P into a relational model M simulating P
- Expression of safety properties :
  $\forall s, s' \in \Sigma.(s \in Init_S \wedge s \xrightarrow{\star} s') \Rightarrow (s' \in A).$
- Definition of the set of reachable states of P using M :
  $\text{REACHABLE}(M) = Init_S \cup \longrightarrow [\text{REACHABLE}(M)]$
- Main property of $\text{REACHABLE}(M)$: $\text{REACHABLE}(M) \subseteq A$
- Characterization of $\text{REACHABLE}(M)$:
  $\text{REACHABLE}(M) = \text{FP}(\text{REACHABLE}(M))$

▶ Proving automatically $\textsc{reachable}(M) \subseteq A$:

Problems

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

Problems

▶ Proving automatically REACHABLE(M) ⊆ A : undecidable ... no program is able to prove it automatically !

▶ Proving automatically REACHABLE(M) ⊆ A :

Problems

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

▶ Proving automatically $\text{REACHABLE}(M) \subseteq A$: possible when restrictions over the set of states is possible (finite set of states)

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

▶ Proving automatically $\text{REACHABLE}(M) \subseteq A$: possible when restrictions over the set of states is possible (finite set of states)

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

Problems

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

▶ Proving automatically $\mathrm{REACHABLE(M)} \subseteq A$: possible when restrictions over the set of states is possible (finite set of states)

▶ Proving automatically $\mathrm{REACHABLE(M)} \subseteq A$: possible for some classes of systems and with some tools.

Problems

- ▶ Proving automatically REACHABLE(M) ⊆ A : undecidable ... no program is able to prove it automatically !

- ▶ Proving automatically REACHABLE(M) ⊆ A : possible when restrictions over the set of states is possible (finite set of states)

- ▶ Proving automatically REACHABLE(M) ⊆ A : possible for some classes of systems and with some tools.

- ▶ Proving automatically REACHABLE(M) ⊆ A : changing the domain and solving in another domain as abstract interpretation if making possible

Problems

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

▶ Proving automatically $\text{REACHABLE}(M) \subseteq A$: possible when restrictions over the set of states is possible (finite set of states)

▶ Proving automatically $\text{REACHABLE}(M) \subseteq A$: possible for some classes of systems and with some tools.

▶ Proving automatically $\text{REACHABLE}(M) \subseteq A$: changing the domain and solving in another domain as abstract interpretation if making possible

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

Problems

- Proving automatically $\text{REACHABLE}(M) \subseteq A$: undecidable ... no program is able to prove it automatically!
- Proving automatically $\text{REACHABLE}(M) \subseteq A$: possible when restrictions over the set of states is possible (finite set of states)
- Proving automatically $\text{REACHABLE}(M) \subseteq A$: possible for some classes of systems and with some tools.
- Proving automatically $\text{REACHABLE}(M) \subseteq A$: changing the domain and solving in another domain as abstract interpretation if making possible
- Proving automatically $\text{REACHABLE}(M) \subseteq A$: approximating semantics of programs

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 $\mathcal{L}(G) \subseteq \Sigma^{\star}$.

▶ 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 $\mathcal{L}(G) \subseteq \Sigma^{\star}$.

▶ 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

▶ 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 $\mathcal{L}(G) \subseteq \Sigma^\star$.

▶ 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

▶ Problem of the correctness of a program :
- Assume that $\mathcal{F}$ is the set of unary function over natural numbers : $\mathcal{F} = \mathbb{N} \nrightarrow \mathbb{N}$.
- $\mathcal{C} \subseteq \mathcal{F}$: the set of computable (or programmable) functions is $\mathcal{C}$
- $f \in \mathcal{C} = \{\Phi_0, \Phi_1, \ldots, \Phi_n, \ldots\}$: the set of computable functions is denumerable.
- The problem $x \in dom(\Phi_y)$ is not decidable and it expresses the correctness of programs.

Some observations

Implicit versus explicit

▶ Writing $101 = 5$ can have a meaning

Implicit versus explicit

▶ Writing *101 = 5* can have a meaning

▶ The code for the number n is *101* to the left of the symbol $=$ and the code for the number n is its representation in base 10 to the right.

▶ $n_{10} = 5$ and $n_2 = 101$

▶ Verification : $base(2, 10, 101) = 1.2^2 + 0.2 + 1.2^0 = 5_{10}$

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 ?

Current Summary

▶ A program P *produces* results à from data according to its semantics :

- STATES is the set of all states of P : $STATES = X \to \mathbb{Z}$ where X denotes the variables of P.
- s_0 and s_f are two states of STATES : $\mathcal{D}(P)(s_0) = s_f$ means that P is executed from state s_0 and produces state s_f.
- For a current state s of P, we will denote $s(X) = x$ to distinguish the value of the variable X and its current value in s. :

▶ A program P *produces* results à from data according to its semantics :

- STATES is the set of all states of P : $STATES = X \to \mathbb{Z}$ where X denotes the variables of P.
- s_0 and s_f are two states of STATES : $\mathcal{D}(P)(s_0) = s_f$ means that P is executed from state s_0 and produces state s_f.
- For a current state s of P, we will denote $s(X) = x$ to distinguish the value of the variable X and its current value in s. :

$$s_0(X) = x_0, \; s_f(X) = x_f, \; s'(X) = x'$$

Intuition

▶ A program P *produces* results à from data according to its semantics :

- STATES is the set of all states of P : $STATES = X \to \mathbb{Z}$ where X denotes the variables of P.
- s_0 and s_f are two states of STATES : $\mathcal{D}(P)(s_0) = s_f$ means that P is executed from state s_0 and produces state s_f.
- For a current state s of P, we will denote $s(X) = x$ to distinguish the value of the variable X and its current value in s. :
$$s_0(X) = x_0,\ s_f(X) = x_f,\ s'(X) = x'$$
- $\mathcal{D}(P)(s_0) = s_f$ defines the following relation on the set of values :
$$x_0 \xrightarrow{\ P\ } x_f$$

▶ A program P *produces* results à from data according to its semantics :

- STATES is the set of all states of P : $STATES = X \to \mathbb{Z}$ where X denotes the variables of P.
- s_0 and s_f are two states of STATES : $\mathcal{D}(P)(s_0) = s_f$ means that P is executed from state s_0 and produces state s_f.
- For a current state s of P, we will denote $s(X) = x$ to distinguish the value of the variable X and its current value in s. :
$$s_0(X) = x_0, \ s_f(X) = x_f, \ s'(X) = x'$$
- $\mathcal{D}(P)(s_0) = s_f$ defines the following relation on the set of values :
$$x_0 \xrightarrow{\ \mathsf{P}\ } x_f$$

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

- P transforms a variable x from an initial value x_0 and producing a final value x_f : $x_0 \xrightarrow{\ \mathsf{P}\ } x_f$
- x_0 satisfies pre : $\mathsf{pre}(x_0)$
- x_f satisfies post : $\mathsf{post}(x_0, x_f)$
- $\mathsf{pre}(x_0) \wedge x_0 \xrightarrow{\ \mathsf{P}\ } x_f \Rightarrow \mathsf{post}(x_0, x_f)$

Checking Contracts

A programme P *fulfils* a contract (pre,post) :

- ▶ P transforms a variable x from an initial value x_0 and producing a final value $x_f : x_0 \xrightarrow{\text{P}} x_f$

- ▶ x_0 satisfies pre : $\text{pre}(x_0)$ and x_f satisfies post : $\text{post}(x_0, x_f)$

- ▶ $\text{pre}(x_0) \wedge x_0 \xrightarrow{\text{P}} x_f \Rightarrow \text{post}(x_0, x_f)$

```
requires pre(x_0)
ensures post(x_0, x_f)
variables X
    ┌ begin
    │ 0 : P_0(x_0, x)
    │ instruction_0
    │ ...
    │ i : P_i(x_0, x)
    │ ...
    │ instruction_{f-1}
    │ f : P_f(x_0, x)
    └ end
```

- ▶ $pre(x_0) \wedge x = x_0 \Rightarrow P_0(x_0, x)$
- ▶ $P_f(x_0, x) \Rightarrow post(x_0, x)$
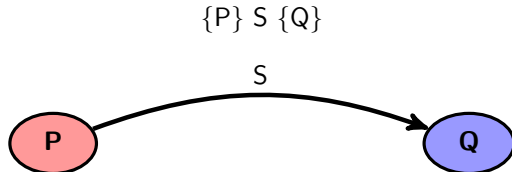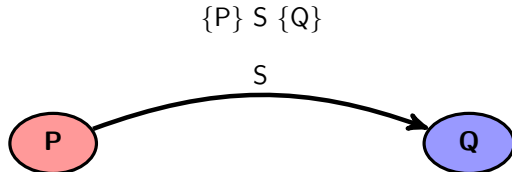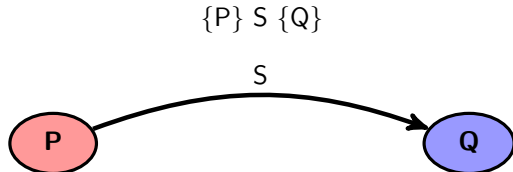- ▶ some conditions for verification related to pairs $\ell \longrightarrow \ell'$

{P} S {Q}

S

P　　　　　　　　　　　　**Q**

{P} S {Q}

S

P → **Q**

▶ {P} S {Q} : *asserted program*

{P} S {Q}

S

P → **Q**

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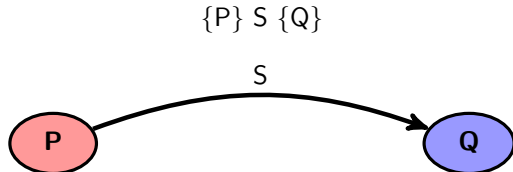- P $\Rightarrow WP(S)(Q)$: *logical formula*

$\{P\}$ S $\{Q\}$

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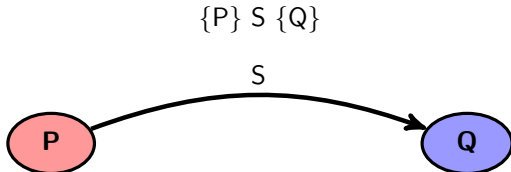- ▶ $C(S) \vdash P \Rightarrow WP(S)(Q)$: *logical formula*

$\{P\}$ S $\{Q\}$

S

P   **Q**

▶ {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*

$\{P\}$ S $\{Q\}$

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

{P} S {Q}

S

P   **Q**

- ▶ {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*

$\{P\}$ S $\{Q\}$

S

P → **Q**

- ▶ $\{P\}$ 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.

Current Summary

1 Tracking bugs in C codes

2 Introduction by Example
  Detecting overflows in computations
  Computing the velocity of an aircraft on the ground
  Computing the date of Easter
  Computing how many days between two dates

3 Context and Objectives

4 Verification of program properties

5 stop movex 1

6 Topics of course

Summary

Summary

Modelling, specification and verification (MALG1/MOVEX1)

Summary

Modelling, specification and verification (MALG1/MOVEX1)

▶ Set-theoretical notations using the Event-B modelling language.

▶ Relational modelling of a program or an algorithm.

▶ Program properties as safety, invariance, pre and post conditions.

▶ Design by contract.

▶ Method for proving invariance properties of programs and induction principles as Floyd's method, Hoar logics,

▶ Techniques for Model-Checking.

▶ Tools : the toolset TLAPS, the toolset Frama-c.

Summary

Summary

Logics (MALG1/MOVEX1)

Summary

Logics (MALG1/MOVEX1)

▶ Propositional Formulae and first order formulae
▶ Models
▶ Sequents Calculus
▶ Proofs and deduction
▶ Resolution
▶ Tools : toolsets TLAPS and Frama-c

Summary

Summary

Fixed-Point Theory (MALG2)

Fixed-Point Theory (MALG2)

▶ Complete Partiallly Ordered Sets (CPO) and Complete Lattices

▶ Fixed-Point Theorems : Kleene, Tarski, . . .

▶ Abstract Interpretation

▶ Galois Lattices

▶ Static analysis of Programs by abstract interpretation.

▶ Semantics of programs

Summary

Summary

Computability, Decidability, complexity, Undecidabiliy(MALG2)

Summary

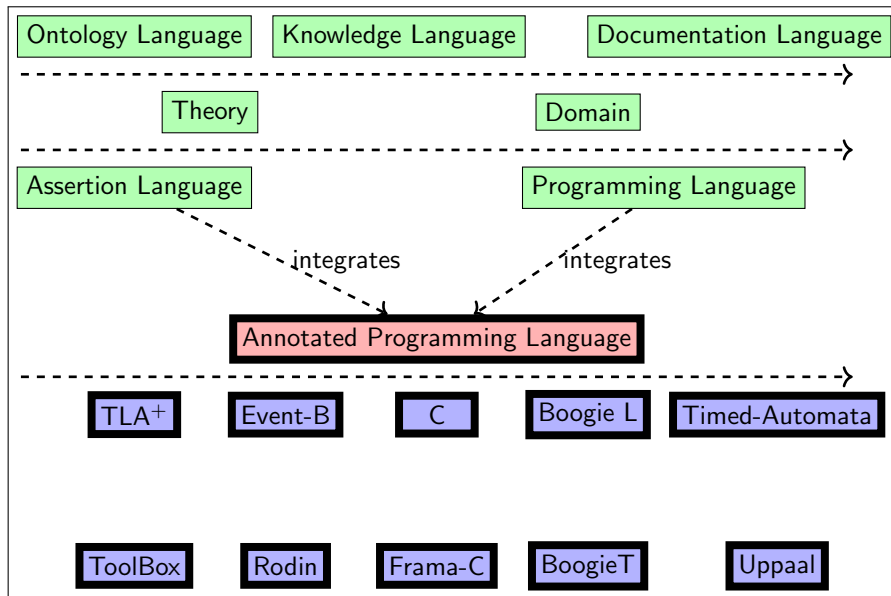## Computability, Decidability, complexity, Undecidabiliy(MALG2)

- ▶ Models of computing : Turing Machines, Partially Recursive Functions, URM, . . .
- ▶ Church's Thesis
- ▶ Decidability
- ▶ Undecidability
- ▶ Complexity

Summary

(MOVEX2)

- ▶ Fixed-point theory and its applications to semantics
- ▶ The LUSTRE language
- ▶ Kind2 synchronous modelling

Summry of concepts

Summary

Summary

Tools (MALG - MOVEX)

Summary

Tools (MALG - MOVEX)

- ► The TLA$^+$ ToolBox
- ► Frama-C
- ► Kind2

Informations on the two courses

- ▶ Documents on Arche MODELS AND ALGORITHMS with the password *mery2023* and at the following link : https://mery54.github.io/teaching/movex/
- ▶ Alternating lectures and tutorials with sessions on personal computers.
- ▶ Two tutorial groups
- ▶ Virtual machine and telecom machines with the software installed.