universität
innsbruck

Specialisation Seminar

Abstraction Interpretations

Georg Moser

cbr.uibk.ac.at

Organisation

Schedule

week 1	March 7	administration	week 8	May 9
week 2	March 14		week 9	May 16
week 3	March 21		week 10	May 23
week 4	March 28		week 11	June 6
week 5	April 4		week 12	June 13
week 6	April 11		week 13	June 27
week 7	May 2			

Schedule

week 1	March 7	administration	week 8	May 9
week 2	March 14		week 9	May 16
week 3			week 10	May 23
week 4			week 11	June 6
week 5			week 12	June 13
week 6			week 13	
week 7				

Schedule

week 1	March 7	administration	week 8	May 9
week 2	March 14		week 9	May 16
week 3			week 10	May 23
week 4			week 11	June 6
week 5			week 12	June 13
week 6			week 13	
week 7				

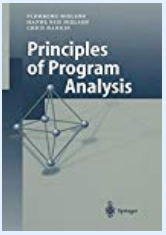
Time and Place

Seminar Thursday, 11:15–12:00, 3W03

Literature

Principles of Programming Analysis

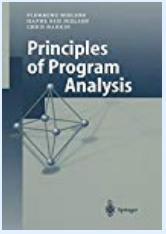
Flemming Nielson, Hanne Riis Nielson, Chris Hankin



Literature

Principles of Programming Analysis

Flemming Nielson, Hanne Riis Nielson, Chris Hankin



- We study the 4th Chapter of the book on Abstract Interpretations in detail
- Each team of students (1-2) will prepare one of the subsections and present them at the end of the term
- The seminar will be extended to 60min on presentations days
- Each team has to write one short report (5 pages)
- Today and next week I'll give a teaser on static program analysis

Program Analysis

Static Analysis

Aims to prove properties about the runtime behavior of a program without actually running it.

> *The algorithmic discovery of properties of a program by inspection of the source text.* — *Manna and Pnueli, "Algorithmic Verification"*

Static Analysis

Aims to prove properties about the runtime behavior of a program without actually running it.

> *The algorithmic discovery of properties of a program by inspection of the source text.* — *Manna and Pnueli, "Algorithmic Verification"*

Example

- is the program free of runtime errors, eg. overflows, division by zero, null-pointer dereference?
- does the program terminate?
- when does the program terminate, how much memory, energy, etc. is required?
- are two structures on the heap disjoint?

Zune Bug

```
year = ORIGINYEAR; /* = 1980 */
while (days > 365) {
        if (IsLeapYear(year)) {
                if (days > 366) {
                        days -= 366;
                        year += 1; } }
        else {
                days -= 365;
                year += 1; } }
```

Zune Bug

```
year = ORIGINYEAR; /* = 1980 */
while (days > 365) {
        if (IsLeapYear(year)) {
                if (days > 366) {
                        days -= 366;
                        year += 1; } }
        else {
                days -= 365;
                year += 1; } }
```

“Solution”

wait one day and reboot

Ed Felten, Deputy US Chief Technology Officer (since 2015)

What lessons can we learn from this? First, even seemingly simple computations can be hard to get right. Microsoft's quality control process, which is pretty good by industry standards, failed to catch the problem in this simple code. How many more errors like this are lurking in popular software products? Second, errors in seemingly harmless parts of a program can have serious consequences. Here, a problem computing dates caused the entire system to be unusable for a day.

This story might help to illustrate why experienced engineers assume that any large software program will contain errors, and why they distrust anyone who claims otherwise. Getting a big program to run at all is an impressive feat of engineering. Making it error-free is too much to hope for. For the foreseeable future, software errors will be a fact of life.