Introduction and practical arrangements

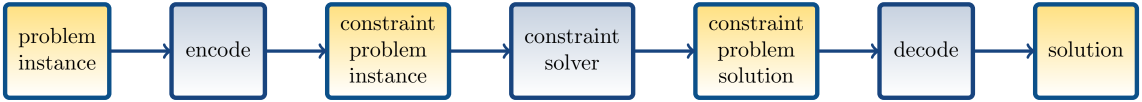
Tommi Junttila and Jussi Rintanen

Aalto University
School of Science
Department of Computer Science

CS-E3220 Declarative Programming
Autumn 2020

Declarative and constraint programming

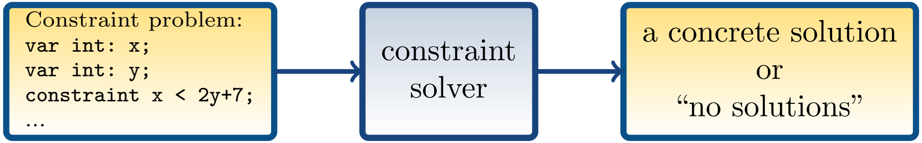
- Imperative programming: how something is done
- Declarative programming: declare what a program should accomplish, not how this is done
- Declarative programming is an "umbrella term" covering many paradigms
- We'll focus on constraint programming, where the problem at hand is described with **variables** and **constraints** so that any assignment to the variables that respects the constraint is a solution to the problem[1]
- The figure below shows a typical flow in constraint programming:
  - The problem instance is encoded to a constraint problem instance,
  - which is then solved by some highly-optimised constraint solver, and
  - a solution to the problem instance is decoded from the solver output



[1]The course could (and perhaps should) be called "constraint programming" but this term historically refers more strongly to one approach (CSPs, round 3) than to some others (SAT, SMT) that we also cover

Constraint solvers

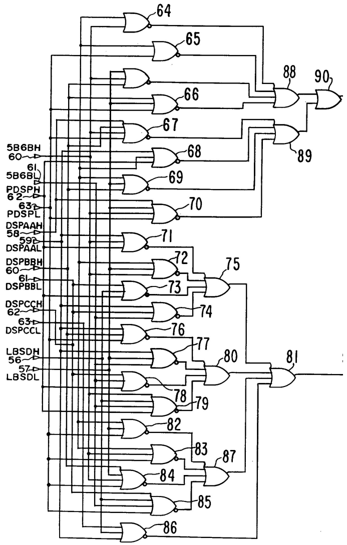
- Constraint programming is usually applied to *intractable*, NP-hard or harder, problems
- Such problems could be solved with backtracking search as, too ...
- but using **constraint solvers** makes the task easier as one only needs to declare the constraints (and it is the solver's responsibilitu to do the search)
- Basically, a constraint solver is a tool that
  - takes a problem instance as input,
  - finds whether the constraints have a solution, and
  - outputs such a solution if one exists or "no solutions" if the constraints cannot be satisfied



This course

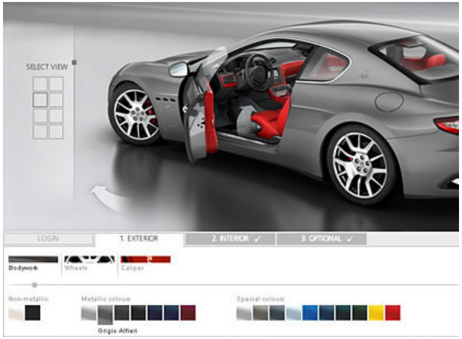
- Constraint problem types to be covered in this course
  - Propositional satisfiability (SAT)
  - Constraint satisfaction problems (CSP)
  - Satisfiability modulo theories (SMT)
- Practice: solving problems with these
- Theory: (a glimpse of) how the solvers for these formalisms work
- Applications: Where is all this applied

Application 1: Integrated circuits correctness, testing, diagnosis



- T. Larrabee: Test pattern generation using Boolean satisfiability, 1992.
- A. Biere et al.: Symbolic model checking without BDDs, 1999.
- E. I. Goldberg, M. R. Prasad and R.K. Brayton, Using SAT for combinational equivalence checking, 1997.
- J. R. Burch, E. H. Clarke, K. L. McMillan and D. L. Dill, Sequential circuit verification using symbolic model checking, 1991.
- A. Smith, A. Veneris, M. F. Ali, and A. Viglas, Fault diagnosis and logic debugging using Boolean satisfiability, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2005.

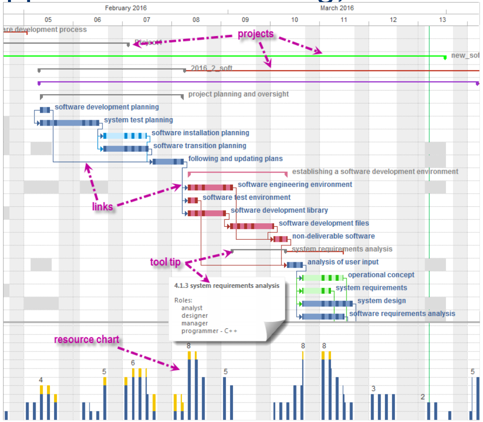
Application 2: Product/Software Configuration



- Configuration: Choose components based on requirements and inter-component dependencies (A requires B; C and D are incompatible)
- Product configuration: cars, all kinds of machinery, ...
- Software package configuration (operating systems)

- T. Soininen and I. Niemelä: Developing a Declarative Rule Language for Applications in Product Configuration, 1999.
- F. Mancinelli, J. Boender, R. Di Cosmo, J. Vouillon, B. Durak, X. Leroy, R. Treinen, Managing the Complexity of Large Free and Open Source Package-Based Software Distributions, 2006.
- P. Trezentos, I. Lynce, A. L. Oliveira, Apt-pbo: solving the software dependency problem using pseudo-boolean optimization, 2010.

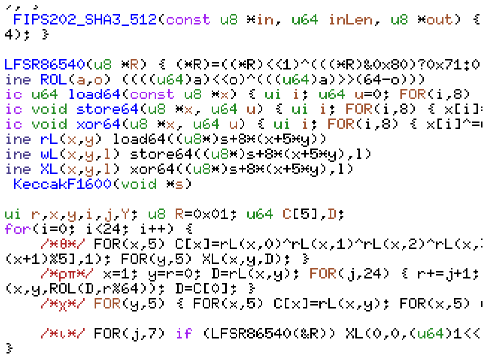
Application 3: Planning, Scheduling, Timetabling


February 2016
March 2016
05
06
07
08
09
10
11
12
13
are development process
projects
Project
new_sof
2016_2_soft
project planning and oversight
software development planning
system test planning
software installation planning
software transition planning
following and updating plans
establishing a software development environment
software engineering environment
software test environment
software development library
software development files
non-deliverable software
system requirements analysis
analysis of user input
operational concept
system requirements
system design
software requirements analysis
links
tool tip
4.1.3 system requirements analysis
Roles:
analyst
designer
manager
programmer - C++
resource chart

- Scheduling of courses/classes for schools, universities
- Project scheduling
- Production scheduling (manufacturing)
- Timetables/schedules for vehicles (trains, buses, airplanes)
- Staff/crew scheduling (airlines, trains, buses)

- P. Baptiste, Philippe, C. Le Pape and W. Nuijten, Wim. “*Constraint-based scheduling: applying constraint programming to scheduling problems*”, 2012.
- Companies and products: Quintiq, IBM ILOG CP Optimizer

Application 4: Software Model-Checking



```
 FIPS202_SHA3_512(const u8 *in, u64 inLen, u8 *out) {
4); }

LFSR86540(u8 *R) { (*R)=((*R)<<1)^(((*R)&0x80)?0x71:0
ine ROL(a,o) ((((u64)a)<<o)^(((u64)a)>>(64-o)))
ic u64 load64(const u8 *x) { ui i; u64 u=0; FOR(i,8)
ic void store64(u8 *x, u64 u) { ui i; FOR(i,8) { x[i]
ic void xor64(u8 *x, u64 u) { ui i; FOR(i,8) { x[i]^=
ine rL(x,y) load64((u8*)s+8*(x+5*y))
ine wL(x,y,l) store64((u8*)s+8*(x+5*y),l)
ine XL(x,y,l) xor64((u8*)s+8*(x+5*y),l)
 KeccakF1600(void *s)

ui r,x,y,i,j,Y; u8 R=0x01; u64 C[5],D;
for(i=0; i<24; i++) {
    /*θ*/ FOR(x,5) C[x]=rL(x,0)^rL(x,1)^rL(x,2)^rL(x,
(x+1)%5],1); FOR(y,5) XL(x,y,D); }
    /*ρπ*/ x=1; y=r=0; D=rL(x,y); FOR(j,24) { r+=j+1;
(x,y,ROL(D,r%64)); D=C[0]; }
    /*χ*/ FOR(y,5) { FOR(x,5) C[x]=rL(x,y); FOR(x,5)

    /*ι*/ FOR(j,7) if (LFSR86540(&R)) XL(0,0,(u64)1<<
}
```

- Test if program satisfies a given property
- Safety-critical applications
- Concurrency problems in multi-threaded programs

- R. Jhala, R. Majumdar, Software model checking, ACM Computing Surveys (CSUR), 2009.
- L. Cordeiro, B. Fischer, and J. Marques-Silva, SMT-Based Bounded Model Checking for Embedded ANSI-C Software, 2011.
- F. Merz, S. Falke, C. Sinz, LLBMC: Bounded Model Checking of C and C++ Programs Using a Compiler IR, 2012.

Application 5: Software Synthesis

- Project at Aalto U since 2016 (AISS research group)
- Synthesis of full-stack program code from declarative specifications of software functionalities
- Domain: web apps, information systems, and other UI and DB intensive SW
- Specification for a change in the state of an application
  - User inputs $x_1, x_2, \ldots, x_n$ (any data types)
  - Condition $\Phi(x_1, x_2, \ldots, x_n, y_1, \ldots, y_n)$ on the inputs and data $y_1, \ldots, y_n$ in DB
  - Program code to change DB according to the inputs
- Constraint satisfaction problem: $x_1, \ldots, x_n$ must satisfy $\Phi(x_1, x_2, \ldots, x_n, y_1, \ldots, y_n)$
- Automated synthesis of full stack code (DB, app logic, UI functionality) for whole application

Application 6: Solving mathematical problems

- Boolean Pythagorean Triples problem

  Is there an n such that in every partitioning of $\{1, 2, \ldots, n\}$ *into two parts, either part contains three numbers a, b, and c such that* $a^2 + b^2 = c^2$*?*

- In 2017, Heule and Kullmann showed that such *n* exists: 7825
- A 200 TB machine checkable proof of this was also produced
- Heule, Kullmann, and Marek: Solving and Verifying the Boolean Pythagorean Triples Problem via Cube-and-Conquer, 2016
- Also see Heule and Kullmann: The science of brute force, 2017

Practical arrangements

First things first...

- Exercises, changes in schedule, announcements etc are in MyCourses

  `https://mycourses.aalto.fi/course/view.php?id=28176`

- In order to see all these, please

  register to the course in Oodi

  immediately if you plan to take the course

Contents

Ten rounds (the order is still tentative):

1	Propositional logic	Tommi
2	Fundamentals of SAT solvers	Tommi
3	Constraint satisfaction problems (CSP)	Tommi
4	Binary decision diagrams	Jussi
5	Symbolic state space search with BDDs	Jussi
6	State-space search through satisfiability	Jussi
7	Specification languages, modal and temporal logics	Jussi
8	Model checking in verification	Jussi
9	Satisfiability modulo theories (SMT), part I	Tommi
10	Satisfiability modulo theories (SMT), part II	Tommi

Personnel

- Responsible teachers:
  - Professor Jussi Rintanen
  - Senior university lecturer Tommi Junttila
- Teaching assistant: Saurabh Fadnis
- Email: firstname dot lastname at aalto dot fi
  (No email consultations: please attend the exercise sessions)

Prerequisites

- This is a Master's level course in Computer Science
- The prerequisites are:
  - Programming skills in some procedural language such as Python, Java, Scala or C++
    We use Python in some exercises and you should be able to learn the syntax of Python quickly if you don't know it already
  - Fundamental data structures and algorithms, e.g., CS-A1140 Data Structures and Algorithms
  - Basics on discrete mathematics
  - Basics of propositional logic covered, e.g., CS-E4800 Artificial Intelligence
    A sufficient recap will be provided in Round 1

Passing and grading of the course

- To pass the course, one has to pass
  - obligatory online exercises (see MyCourses)
  - exam (one on Dec 14, 2020; another one on Feb 23, 2021)
- The total grade is obtained by the following scheme:

		exam grade					
		0	1	2	3	4	5
exercise points	0–199	0	0	0	0	0	0
	200–349	0	1	2	2	3	3
	350–499	0	2	2	3	3	4
	500–649	0	2	3	3	4	4
	650–799	0	3	3	4	4	5
	800–1000	0	3	4	4	5	5

- For instance, if one gets 600 points from the online exercises and grade 4 from the exam, the total grade will be 4
- As usual, total grade 0 means “not passed” or “failed”

Online exercises

- See MyCourses
- Both “theory” and “practice”
  - Theory: check that the topics, algorithms etc are understood. No programming.
  - Practice: solving problems with some actual state-of-the-art tools. Some programming required but emphasis is on modelling.
- Ten rounds, 100 points available on each $\Rightarrow$ max. 1000 points
  - roughly half of the points from no-programming theory exercises
  - roughly half of the points from programming practice exercises
  - some points for feedback
- Exercise sessions:
  - Online
  - See MyCourses for instructions and schedule
- Exercise points obtained in Autumn 2020 will be valid in all exams before the next course in Autumn 2021 (but *not* after that)
- Exercise points from Autumn 2019 and earlier are *not* valid anymore
- **The exercises are personal work!**

Questions?

Please post them in our discussion forum.