



Planning & Scheduling

Roman Barták

Department of Theoretical Computer Science and Mathematical Logic

Course questions

What is the content?

- planning and scheduling
- but what is planning and scheduling?

Why could it be interesting to me?

- is it used somewhere?
- any applications?

What is the course about?

- problem formalisation and modelling
- solving approaches

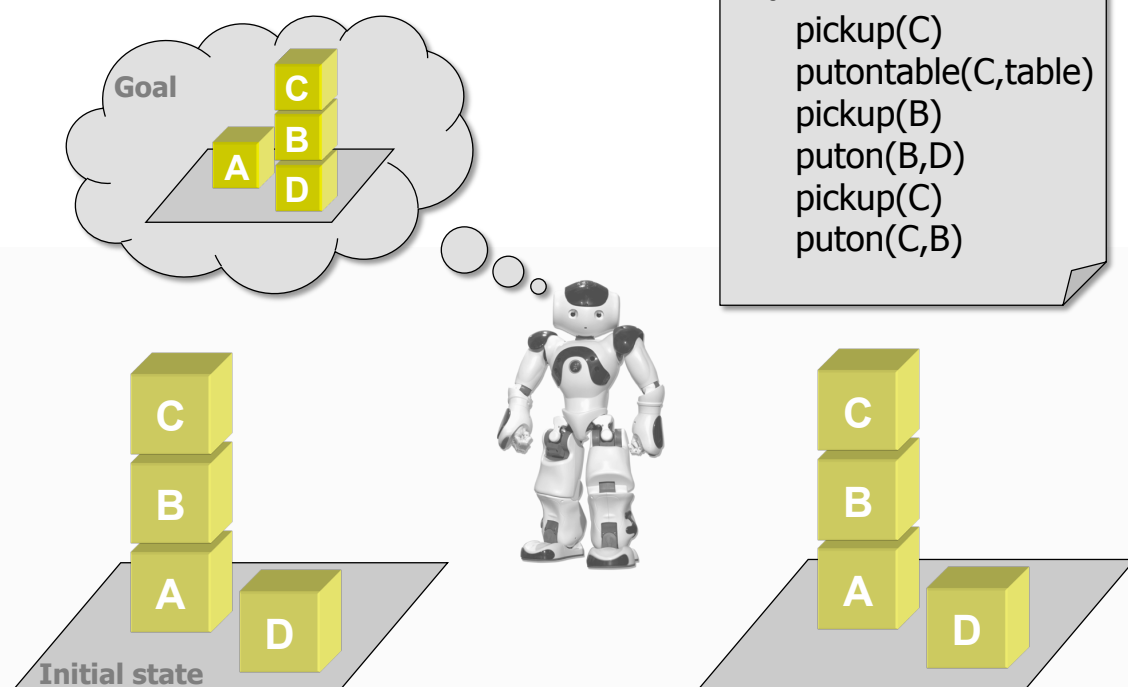


What?

What is planning and scheduling?

What is a difference between them?

What is planning?



Input:

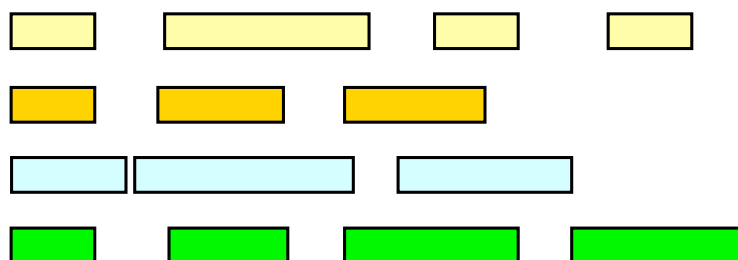
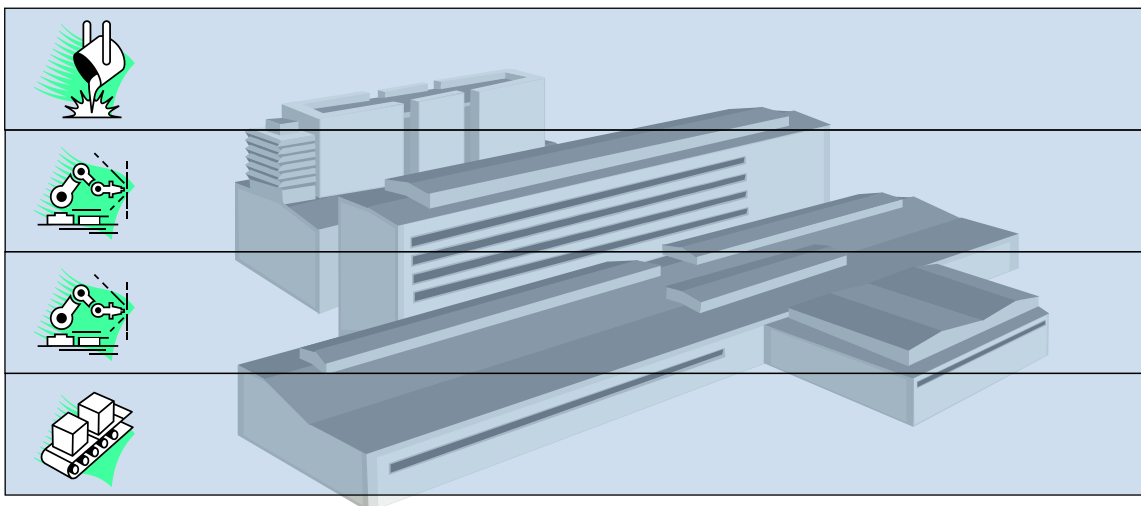
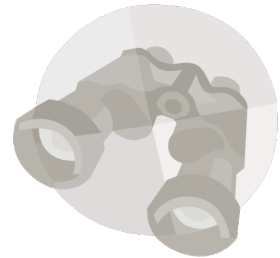
- initial (current) state of the world
- description of actions that can change the world
- desired state of the world

Output:

- a sequence of actions (a plan)

Properties:

- actions in the plan are unknown
- time and resources are not assumed



Input:

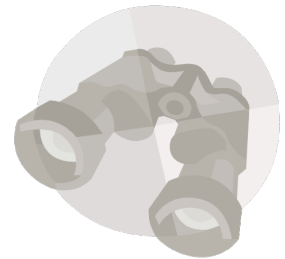
- a set of partially ordered activities
- available resources (machines, people, ...)

Output:

- allocation of activities to time and resources (schedule)

• **Properties:**

- activities are known in advance
- limited time and resources

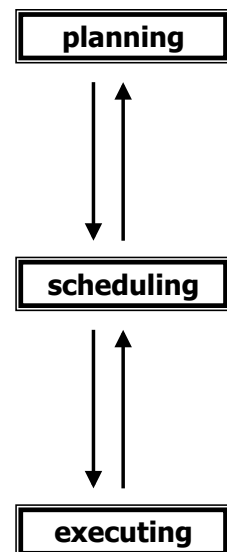


Planning

- deciding which actions are necessary to achieve the goals
- topic of artificial intelligence
- complexity is usually worse than NP-c (in general, undecidable)

Scheduling

- deciding how to process the actions using given restricted resources and time
- topic of operations research
- complexity is typically NP-c



Why?

Is this technology practically useful?
Any applications?

Aircraft assembly

570 tasks, 17 resources

A traditional approach:

- ARTEMIS
- 20 hours to produce a schedule

Intelligent Planning and Scheduling:

- ARTEMIS substituted by a CSP
- 30 minutes to generate an optimal schedule
- 10 - 15% shorter makespan

Savings:

- 4 to 6 days shorter scheduled
- **\$200k – \$1m per day**



7000 tasks per boat and approx. 125 resource classes

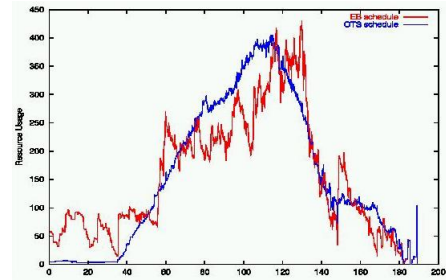
A traditional approach:

- ARTEMIS
- 6 weeks to generate a schedule
- very non-uniform resource profile



Intelligent Planning and Scheduling:

- ARTEMIS substituted by a CSP
- 2 days per schedule
- uniform resource profile



Savings:

- **30% less overtime and sub-contracts**

Contribution of On Time Systems

Gulf war 1991:

A traditional approach:

- hundreds of human planners
- months to generate a plan



Intelligent Planning and Scheduling:

- System O-PLAN2

Savings:

- faster background creation
- less flight missions
- Financial backflow >> **all research AI supported by US government:**



- since 1956
- not only IP&S, but **but all AI research!**

Contribution of On Time Systems



Launch: October 24, 1998

Target: Comet Borrelly

testing a payload of 12 advanced, high risk technologies

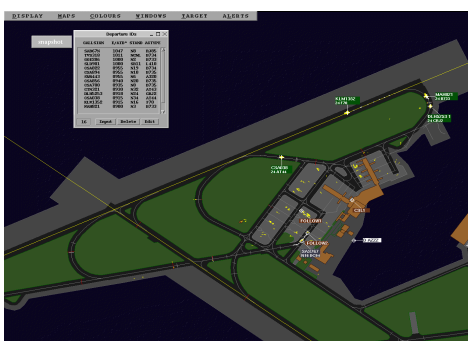
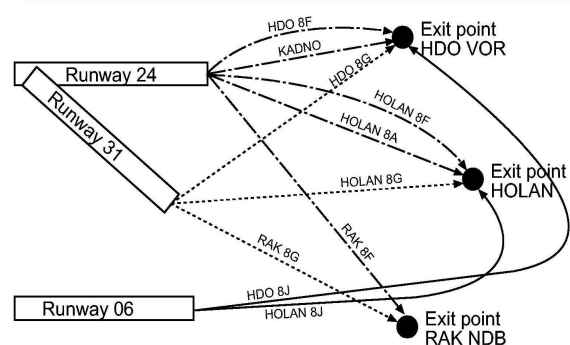
– autonomous remote agent

- planning, execution, and monitoring spacecraft activities based on general commands from operators
- three testing scenarios
 - 12 hours of low autonomy (execution and monitoring)
 - 6 days of high autonomy (operating camera, simulation of faults)
 - 2 days of high autonomy (keep direction)
 - » beware of backtracking!
 - » beware of deadlock in plans!



Departure management

- pre-flight control
 - exit assignment and clearance
 - coordinates with Brussels
- ground control
 - taxiing
- control tower
 - runway assignment
 - separation



MANTEA

(MANagement of surface Traffic in European Airports)

- implemented in **ILOG Scheduler**
- tested in **Prague** (27.5. – 7.6. 2002)

About what?

What does this course bring?

Which topics are covered?

Course outline

Preliminaries

- search algorithms, constraint satisfaction and SAT

Planning

- classical planning (STRIPS)
- neo-classical planning (Graphplan)
- hierarchical planning
- planning with time and resources

Scheduling

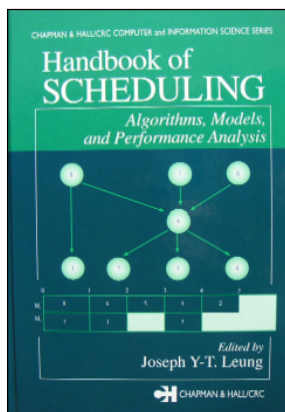
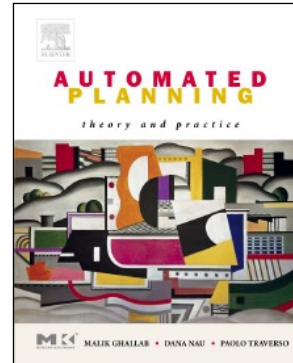
- classical scheduling
- constraint-based scheduling

Applications



Automated Planning: Theory and Practice

- M. Ghallab, D. Nau, P. Traverso
- <http://www.laas.fr/planning/>
- Morgan Kaufmann

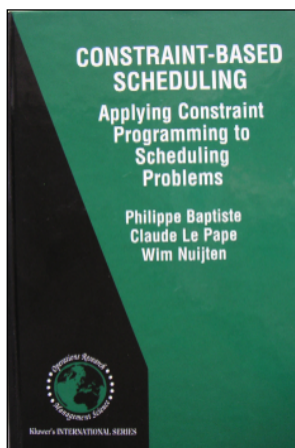
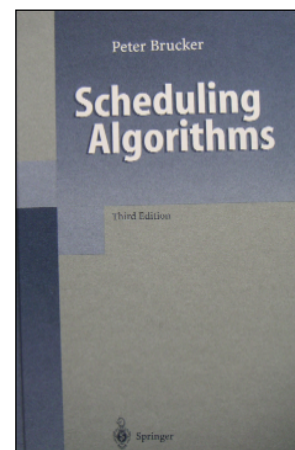


Handbook of Scheduling

- J. Leung
- Chapman&Hall/CRC

Scheduling Algorithms

- P. Brucker
- Springer



Constraint-based Scheduling

- P. Baptiste, C. Le Pape, W. Nuijten
- Kluwer



Planování a rozvrhování - Windows Internet Explorer
 http://kt.mff.cuni.cz/~bartak/planovani/index.html

Plánování a rozvrhování

NAIL071, 2/0 Zk, letní semestr
 Roman Barták, KTIML

[Zdroje](#) | [Přednáška](#) | [Zkouška](#) | [Kontakt](#)

Plánování je rozumovou složkou konání. Jeho cílem je vybrat a uspořádat akce tak, aby se co nejlépe dosáhlo vytyčeného cíle. **Rozvrhování** se potom stará o optimální realizaci plánu v prostředí s omezenými zdroji a časem.

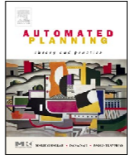
Zdroje: [nahoru](#)

Přednáška je připravena převážně podle knihy **M. Ghallab, D. Nau, P. Traverso: Automated Planning: Theory and Practice, Morgan Kaufmann, 2004**. Materiály ke knize jsou dostupné na [webu](#).

Některé pasáže jsou podrobněji zpracovány v anglických tutoriálech:

- **Constraint Satisfaction for Planning and Scheduling** [[WWW](#)], ICAPS 2004
- **Filtering Techniques in Planning and Scheduling** [[slajdy](#)], ICAPS 2006

Další informace lze čerpat ze stránek sítě excelence [PLANET](#), konferencí [ICAPS](#) a [MISTA](#).



Přednáška (LS 2008/2009): [nahoru](#)
 úterý 15:40 - 17:10, posluchárna S4 (Malá Strana, 3. patro)

Tento rozvrh je předběžný a je možné, že bude v průběhu semestru modifikován.

24.02.2009	Úvod, plánovací vs. rozvrhovací problém, ukázky aplikací. Obecné prohledávací algoritmy, omezující podmínky a SAT.	
03.03.2009	Formalizace plánovacího problému. Množinová a klasická reprezentace.	
10.03.2009	Plánování se stavovým prostorem (dopředné, zpětné, STRIPS).	
17.03.2009	Plánování s prostorem plánů.	
24.03.2009	Neoklasické plánování. Plánovací graf, Graphplan.	
31.03.2009	<i>pravděpodobně odpadne</i>	
07.04.2009	Plánování jako SAT. Plánování jako CSP.	
14.04.2009	Modely času (algebra okamžiků, algebra intervalů, temporální sítě)	
21.04.2009	Plánování s časem a se zdroji	
28.04.2009	Plánovací heuristiky a řídicí prostředí, hierarchická plánování	

Preliminaries

What am I supposed to know?

- search techniques
- basics of constraint satisfaction
- logic and SAT

Search techniques are the core solving approach used in AI (and beyond AI).

Classes of search techniques:

- **State-space search**
 - find a state (path to a state) with some properties
- **Problem-reduction search**
 - find a reduction of task to primitive tasks



Properties of algorithms

soundness

- The output of the algorithm is a problem solution.

completeness

- If there is any solution then the algorithm finds it.

admissibility

- The algorithm guarantees finding an optimal solution.
- There must be some measure of optimality!
- It also means soundness and completeness.

State space S is a set of nodes (states) and the task is to find a state satisfying some goal condition g .

Formally, the **problem specification** is a triple (s_0, g, O) :

- s_0 is the **initial state**
- g is a **goal condition** (the goal state satisfies $g(s)$)
- O is a set of **operators** defining the next state
 - State space is defined recursively as:
 - $s_0 \in S$; if $s \in S$, $o \in O$ and $o(s)$ is defined then $o(s) \in S$
 - $o(s)$ is a child of node s

Breadth-First Search

explores tree levels

– q is a **queue**

- **sound and complete**
- **Complexity** to find a goal node at depth d with the branching factor b :
 - time complexity $O(b^d)$
 - space complexity $O(b^d)$

```

bfs( $s_0, g, O$ )
   $q \leftarrow \{s_0\}$ 
  while non-empty( $q$ ) do
     $s \leftarrow \text{first}(q)$ 
    if  $g(s)$  then return  $s$ 
     $q \leftarrow \text{delete\_first}(q)$ 
     $q \leftarrow q + \{s' \mid \exists o \in O, s' = o(s)\}$ 
  end while
  return failure

```

Depth-First Search**(backtracking)**

go in one direction

backtrack upon failure

– q is a **stack**

```
dfs(s0,g,O)
  q ← {s0}
  while non-empty(q) do
    s ← first(q)
    if g(s) then return s
    q ← delete_first(q)
    q ← {s' | ∃o∈O, s'=o(s)} + q
  end while
  return failure
```

- **Sound and complete**, if there are no infinite branches or can be detected.
- **Complexity** to find a goal node at depth d:
 - Time complexity depends on the selected direction (can explore a complete search space but can also go directly to the goal)
 - space complexity $O(d)$

Sometimes we are looking for a goal state while minimizing an objective function $f(s)$.

Best-First Search

Go to the best next state

– q is a **priority queue**

```
bestfs(s0,g,O,f)
  q ← {s0}
  while non-empty(q) do
    s ← best(q,f)
    if g(s) then return s
    q ← delete_best(q,f)
    q ← q ⊕ {s' | ∃o∈O, s'=o(s)}
  end while
  return failure
```

- If f is not decreasing ($s'=o(s) \Rightarrow f(s) \leq f(s')$), then the found solution is optimal. If the search space is finite then the algorithm is admissible.
- If there is some $\delta > 0$ s.t. $s'=o(s) \Rightarrow f(s) + \delta \leq f(s')$, then the algorithm is admissible even for infinite search space.

Another algorithm optimizing objective f .

Depth-First Branch-and-Bound Search

Explore “all” branches
and remember the best
– q is a **stack**

- If f is not decreasing and a state space is finite and with no loops, then the algorithm is.

```
dfbbs( $s_0, g, O, f$ )
 $s^* \leftarrow$  dummy %  $f(\text{dummy}) = \infty$ 
 $q \leftarrow \{s_0\}$ 
while non-empty( $q$ ) do
   $s \leftarrow$  first( $q$ )
   $q \leftarrow$  delete_first( $q$ )
  if  $g(s) \ \& \ f(s) < f(s^*)$  then
     $s^* \leftarrow s$ 
  else
     $q \leftarrow \{s' \mid \exists o \in O, s' = o(s)\} + q$ 
  end if
end while
return  $s^*$ 
```

Greedy Search

Like DFS
but no backtracks

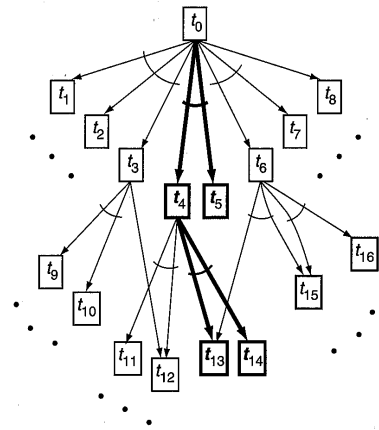
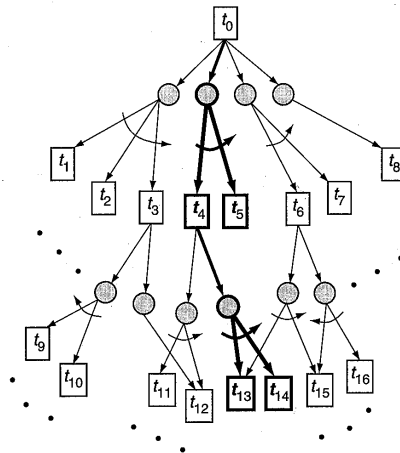
```
gs( $s_0, g, O, f$ )
 $s \leftarrow s_0$ 
while not  $g(s)$  do
   $s \leftarrow$  best( $\{s' \mid \exists o \in O, s' = o(s)\}, f$ )
end while
return  $s$ 
```

- No guarantee of optimality
- Sometimes saves a lot of time necessary to prove optimum.
- Frequently used to find the first solution.

Sometimes the operator o gives a set of children, **sub-problems**, and solution of them represents a portion of the solution of the parent.

This gives an **AND-OR graph**.

↪ **Problem-reduction search**



Problem-reduction search

Problem Reduction Search

Decompose the problem
and find solutions of sub-problems

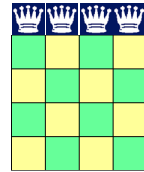
- non-deterministic
- naive
 - Repeatedly solves common sub-problems

```

preds(s,g,O)
  if g(s) then return s
  applicable ← {o∈O | o(s)↓}
  if applicable=∅ then return failure
  o ← choose_nondet(applicable)
  {s1,...,sn} ← o(s)
  for every si∈{s1,...,sn} do
    vi ← preds(si,g,O)
    if vi=failure then return failure
  end for
  return {v1,...,vn}
    
```

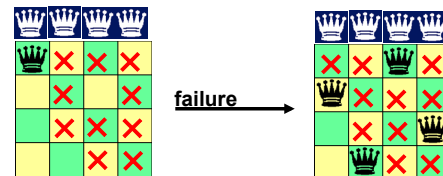
Modeling (problem formulation)

- N queens problem
- **decision variables** for positions of queens in rows
 $r(i)$ in $\{1, \dots, N\}$
- **constraints** describing (non-)conflicts
 $\forall i \neq j \quad r(i) \neq r(j) \ \& \ |i-j| \neq |r(i)-r(j)|$



Search and inference (propagation)

- **backtracking** (assign values and return upon failure)
- infer consequences of decisions
 via maintaining **consistency**
 of constraints



based on **declarative problem description** via:

- **variables with domains** (sets of possible values)
 describe **decision points** of the problem with possible
options for the decisions
 e.g. the start time of activity with time windows
- **constraints** restricting combinations of values,
 describe arbitrary **relations** over the set of variables
 e.g. $\text{end}(A) < \text{start}(B)$

A **feasible solution** to a constraint satisfaction problem
 is a complete assignment of variables satisfying all the
 constraints.

An **optimal solution** to a CSP is a feasible solution
 minimizing/maximizing a given objective function.

Search is combined with filtering techniques that prune the search space.

Maintaining Arc Consistency During Search

procedure labeling(V, D, C)

if all variables from V are assigned **then** return V

 select not-yet assigned variable x from V

for each value v from D_x **do**

(TestOK, D') \leftarrow consistent($V, D, C \cup \{x=v\}$)


if TestOK=true **then** $R \leftarrow$ labeling(V, D', C)

if $R \neq$ fail **then** return R

end for

 return fail

end labeling



The screenshot shows a web browser window with the URL <http://kti.mff.cuni.cz/~bartak/podminky/>. The page title is "Programování s omezuujícími podmínkami" (Programming with constraints) for course NOPT042, 2/1 Zk, zimní semestr, taught by Roman Barták, KTIML. The page includes navigation links for "Zdroje", "Přednáška", "Cvičení", "Zkouška", and "Kontakt". A paragraph describes the course as an introduction to constraint programming, citing Eugene C. Freuder's "Constraints 1997". Under "Zdroje:", it lists books by R. Dechter and F. Rossi et al. A "Přednáška (ZS 2010/2011):" section lists a lecture on Monday 10:40-12:10 in room S9. A table at the bottom lists lecture dates and topics.

Datum	Obsah přednášky
04.10.2010	Úvod, historické souvislosti, ukázky aplikací, vlastnosti omezuujících podmínek. Definice CSP. Binarizace podmínek.
11.10.2010	odpadá
18.10.2010	Přehled algoritmů pro řešení podmínek prohledáváním. Metoda generuj a testuj. Algoritmy lokálního prohledávání (HC, MC, RW, TS)

A formal system consisting of three constituents:

- **language**
(a set of possible statements called formulas)
e.g. $p \rightarrow q$
- **semantics**
(assigns a meaning to each statement)
e.g. if both p and q are true then $p \rightarrow q$ is true
- **proof theory**
(rules to transform statements and derive new statements)
e.g. the modus ponens rule ($p, p \rightarrow q \vdash q$)

The language is a **set P of propositions** – defined inductively starting from an enumerable set of atomic propositions (propositional variables) P_0 :

- if $p \in P_0$ then $p \in P$,
- if $p \in P$ then $\neg p \in P$,
- If $p \in P$ and $q \in P$ then $p \wedge q \in P$,
- Nothing else is a propositional formula.
- We can also define
 - $p \vee q$ as abbreviation for $\neg(\neg p \wedge \neg q)$
 - $p \rightarrow q$ as abbreviation for $\neg p \vee q$
- **Conjunctive Normal Form (CNF):**
 - **formula** is a conjunction of clauses
 - **clause** is a disjunction of literals (clause with a single literal is call a **unit clause**)
 - **literal** is a propositional variable (positive literal) or its negation (negative literal)

A model of propositional formula is an assignment of truth values to the propositional variables (interpretation) for which the formula evaluates to true:

- $\neg p$ is true if and only if p is not true
- $p \wedge q$ is true if and only if both p and q are true

A satisfiability problem (SAT) is the problem of determining whether a formula has a model.

- The SAT problem (given as a CNF) can be solved using **depth-first search** with **unit propagation**.
- **Unit propagation** determines the truth values of literals in unit clauses as follows:
 - the variable in a positive literal is assigned to true,
 - the variable in a negative literal is assigned to falseThe assigned value is propagated to other clauses as follows.
If D is assigned to true then:
 - the clause containing D (e.g. $A \vee \neg B \vee D$) can be discarded
 - the clauses containing $\neg D$ (e.g. $C \vee \neg D \vee E$) can be simplified by removing $\neg D$ ($C \vee E$)

Symmetrically for the case when D is assigned to false.

```

procedure DP(A, Assignment)
  A: is a CNF formula (represented as a set of clauses)
  A and Assignment are local within DP
  if  $\emptyset \in A$  then return
  if  $A = \emptyset$  then exit with Assignment
  Unit-Propagate(A, Assignment)
  select a variable P such that P or  $\neg P$  occurs in A
  DP( $A \cup \{P\}$ , Assignment)
  DP( $A \cup \{\neg P\}$ , Assignment)
end DP

```

```

procedure Unit-Propagate(A, Assignment)
  A and Assignment are global within Unit-Propagate
  while there is a unit clause  $\{l\}$  in A do
    Assignment  $\leftarrow$  Assignment  $\cup \{l\}$ 
    for every clause  $C \in A$  do
      if  $l \in C$  then  $A \leftarrow A - \{C\}$ 
      else if  $\neg l \in C$  then  $A \leftarrow A - \{C\} \cup (C - \{\neg l\})$ 
  end Unit-Propagate

```



© 2015 Roman Barták

Department of Theoretical Computer Science and Mathematical Logic
 bartak@ktiml.mff.cuni.cz