Semantic and Declarative Technologies

Péter Szeredi, László Kabódi, Péter Tóth

szeredi@cs.bme.hu
kabodil@gmail.com
peter@toth.dev

Aquincum Institute of Technology

Budapest University of Technology and Economics
Department of Computer Science and Information Theory

2023 Spring Semester

Course information

- Course layout
  - Introduction to Logic — Weeks 1–2
  - Declarative Programming
    - Prolog – Programming in Logic — Weeks 3–7
    - Constraint Programming — Weeks 8–12
  - Semantic Technologies
    - Logics for the Semantic Web — Weeks 13–14
- Requirements
  - 2 assignments (150 points each) — 300 points
  - 2 tests (mid-term and final, 200 points each) — 400 points total
  - many small exercises + class activity — 300 points total
- Course webpage: `http://cs.bme.hu/~szeredi/ait`
- Course rules: `http://cs.bme.hu/~szeredi/ait/course-rules.pdf`

Part I

Introduction to Logic

Foundations of logic – overview

- Main theme of the course:
  - How to use mathematical logic in
    - programming
    - intelligent web search
- We start with a brief introduction to Logic
  - Propositional Logic:
    - Syntax and semantics
    - The notion of consequence
    - The resolution inference algorithm
    - Bonus: solving various logic puzzles
  - First Order Logic (FOL)
    - Syntax and Model oriented semantics
    - The notion of consequence for FOL
    - The resolution inference algorithm for FOL

Contents

Atomic and compound propositions

- Consider the sentence: *It is raining and I'm staying at home*
- How many propositions (statements) are there in this sentence?
- There are three:
  - two atomic propositions: $A =$ "*It is raining*", $B =$ "*I'm staying at home*"
  - and the whole sentence is a compound proposition $C = A \wedge B$
  - read the symbol $\wedge$ as "and"
  - C is called a conjunction, A and B are conjuncts
- An atomic proposition is the basic building block of general propositions:
  - it can be assigned a truth value
  - it cannot be broken down to simpler propositions
- Truth values: true and false, often represented by integers 1 and 0
- The term propositional formula (or proposition for short) refers to both atomic and compound propositions

Conjunction

- Knowing the truth values of A and B, can you tell the truth value of $A \wedge B$? Think of $A =$ *"It is raining"*, $B =$ *"I'm staying at home"*

A	B	$A \wedge B$
false	false	false
false	true	false
true	false	false
true	true	true

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

  In brief: $A \wedge B$ is true if and only if (iff) ... both A and B are true

- Is the $\wedge$ operator commutative? I.e. $A \wedge B \stackrel{?}{=} B \wedge A$. Why? Because $0 \wedge 1 = 1 \wedge 0$
- Is $\wedge$ associative? I.e. $(A_1 \wedge A_2) \wedge A_3 \stackrel{?}{=} A_1 \wedge (A_2 \wedge A_3)$. Why? Because both sides are 1 iff each of A_1, A_2, A_3 is 1.
- n-fold conjunction: $C_n = A_1 \wedge \cdots \wedge A_n$. When is $C_n = 1$? If *all* A_is are 1.
- What value should be assigned to an empty conjunction C_0 (C_n for $n = 0$)? Hint: Describe the relationship between C_{n-1} and C_n, use this for $n = 1$ $C_n = C_{n-1} \wedge A_n$, $C_1 = A_1$, hence $A_1 = C_0 \wedge A_1$. This is true iff $C_0 = 1$.

Disjunction and negation

- Another example: *It is not raining or (else) I'm staying at home*
- The two atomic propositions are the same as earlier:
  $A =$ *"It is raining"*, $B =$ *"I'm staying at home"*
- *"It is not raining"* converts to $\neg A$, where $\neg$ denotes negation, read as "it is not the case that ..."
- The whole sentence can be formalised as $\neg A \vee B$
- Read the symbol $\vee$ as "or"; $A \vee B$ is called a disjunction, A and B are disjuncts
- The truth tables for disjunction and negation (with $0-1$ values only):

A	B	$A \vee B$
0	0	0
0	1	1
1	0	1
1	1	1

A	$\neg A$
0	1
1	0

Implication

- Example: *If it is raining, then drive slower than 100 km/h*
- I obey this sign provided that *If it is raining, then I drive slowly...*
- This is an implication, formally written as $A \rightarrow B$ (A implies B)
  the premise: $A =$ "*It is raining*", *conclusion:* $B =$ "*I drive slowly ...* "
- When it is not raining, does it matter whether I drive slowly?
- The truth table for implication:

A	B	$A \rightarrow B$
0	0	1
0	1	1
1	0	0
1	1	1

- Express implication using disjunction and negation: $A \rightarrow B = \neg A \vee B$
- $A \rightarrow B$ evaluates to 0 iff $A = 1, B = 0$

Equivalence and exclusive or

- Example 1: *I use an umbrella if and only if it is raining*
- This is an equivalence, formally written as $A \leftrightarrow B$ or $A \equiv B$, $A =$"*I use an umbrella*", $B =$"*It is raining*",
- Example 2: *We either go to movies or have dinner (but not both)*
- This is an exclusive or (XOR), formally written as A *xor* B or $A \oplus B$, $A =$"*we go to movies*", $B =$"*we have a dinner*"
- The truth tables for equivalence and exclusive or:

A	B	$A \equiv B$
0	0	1
0	1	0
1	0	0
1	1	1

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

- Express equivalence using exclusive or, and the other way round: $(A \equiv B) = \neg(A \oplus B)$, $(A \oplus B) = \neg(A \equiv B)$

Normal forms

- A proposition has lots of equivalent formulations:
  $A \to B \equiv \neg A \vee B \equiv \neg(A \wedge \neg B)$
- To design an efficient reasoning algorithm, it makes sense to use one of normal forms (NF), such as:
  - DNF (Disjunctive Normal Form) or CNF (Conjunctive NF)
- Both allow only three operations: $\wedge$, $\vee$, and $\neg$
- In both NFs '$\neg$' can only be used in front of atomic propositions.
- A formula is called a literal if it is either A or $\neg A$, where A is atomic.
- A DNF takes the form $C_1 \vee \ldots \vee C_n$, $n \geq 0$, where each C_i is a conjunction of literals $L_{i1} \wedge \ldots \wedge L_{im_i}$
- A CNF takes the form $D_1 \wedge \ldots \wedge D_n$, $n \geq 0$, where each D_i is a disjunction of literals $L_{i1} \vee \ldots \vee L_{im_i}$
- Transform $A \oplus B$ (exclusive or) to both CNF and DNF formats
- Notice that the DNF can be easily derived from a truth table

Models and tautologies

- Recall two kinds of algebraic formulas from high school:

  $x^2 - 3x + 2 = 0$ equation – true for *some* values of x

  $x^2 - y^2 = (x - y)(x + y)$ identity – true for *all* values of x (*)

- Consider a propositional formula with n atomic propositions, e.g.

$$((A \wedge B) \rightarrow C) \quad \equiv \quad (A \rightarrow (B \rightarrow C))$$

- Here $n = 3$, so there are $2^n = 8$ *valuations* for atomic propositions: (A, B, C) can be $(0, 0, 0); (0, 0, 1); (0, 1, 0); \ldots; (1, 1, 0); (1, 1, 1)$
- Each such valuation is called a *model* or a *universe*
- A model satisfies a propositional formula, if the formula is true when the atomic propositions take the 0–1 values specified by the model.
  E.g. the model $(0, 0, 0)$ satisfies the above equivalence
- A formula is called a tautology if all models satisfy the formula
  (cf. the algebraic identity (*) being true for all possible values of x)

Some important tautologies

- Show that this formula is a tautology:

$$((A \wedge B) \rightarrow C) \quad \equiv \quad (A \rightarrow (B \rightarrow C)) \tag{1}$$

  - Let us find all the models in which the left hand side evaluates to 0: There is only one such model $(A, B, C) = (1, 1, 0)$
  - Let us find all the models in which the right hand side evaluates to 0: There is only one such model $(A, B, C) = (1, 1, 0)$
  - Hence the above formula is a tautology
- Show that the following formulas are tautologies:

$$\neg\neg U \quad \equiv \quad U$$
$$\neg(U \wedge V) \quad \equiv \quad \neg U \vee \neg V \tag{2}$$
$$\neg(U \vee V) \quad \equiv \quad \neg U \wedge \neg V \tag{3}$$

  (2) and (3) are called De Morgan's laws.
- Hint: use case-based reasoning for proving formulas (2) and (3):
  1. Select an arbitrary atomic proposition in the formula, say U
  2. Show that the formula to be proven holds for both $U = 0$ and $U = 1$

Contents

An automated inference system: resolution

- The *first order resolution* inference algorithm was devised by Alan Robinson around 1964
- We now introduce resolution for propositional logic
- Resolution uses CNF, *conjunctive normal form* (recall):
  - a CNF is a conjunction of *clauses*: $Cl_1 \land \ldots \land Cl_n$
  - a clause is a disjunction of *literals*: $L_1 \lor \ldots \lor L_k$
  - a literal is either A or $\neg A$, where A is an atomic proposition

Translating propositions to clausal form

- Steps needed to transform an arbitrary formula to CNF:
  1. replace all connectives by equivalents using only $\neg, \wedge, \vee$
  2. move negations inside using De Morgan Laws
  3. apply distributivity (repeatedly, if needed) to eliminate $\wedge$ s inside $\vee$ s: transform $U \vee (V \wedge W)$ to $(U \vee V) \wedge (U \vee W)$
  4. transform $\wedge$ and $\vee$ operators to sets, elminating duplicates

  The result is thus a set of sets, e.g. $\{\{A, B\}, \{B, C\}\} \equiv (A \vee B) \wedge (B \vee C)$ ("Outer" set elements are conjuncts, "inner" set elements are disjuncts)
- Simplified notation (used in first Prolog versions)
  - a literal is written as a signed atomic proposition, e.g. `-A`, `+B` (for $\neg$`A`, `B`)
  - a clause is written as a sequence of literals followed by a full stop, e.g. $\neg A \vee \neg B \vee D$ written as `-A -B +D.`
- Example: transform $((A \wedge B) \rightarrow D) \wedge (C \rightarrow (A \wedge B))$ to to clausal form

  The CNF form: $(\neg A \vee \neg B \vee D) \wedge (\neg C \vee A) \wedge (\neg C \vee B)$

  The CNF in set notation: $\{\{\neg A, \neg B, D\}, \{\neg C, A\}, \{\neg C, B\}\}$

  The CNF in simplified notation: `-A -B +D. -C +A. -C +B.`

The resolution inference rule – introduction

- Consider these two clauses: `+A -B -C.` (1)
  `+A +D +B.` (2)
- Literal # 2 in clause (1) is `-B`, while literal # 3 in clause (2) is `+B`. These literals are *opposite*, i.e. one is the negation of the other.
- Given two clauses containing opposite literals, the resolution rule infers a new clause, called the resolvent, containing the union of all literals of the two clauses, except the two opposite literals.
- In the example the resolvent clause is `+A -C +D.` (3)
  Note that there is only one `+A` as `A` $\vee$ `A` = `A`.
- Resolution is sound, i.e. (3) is implied by (1) and (2). This is due to the *resolution principle*:

$$\underbrace{(\neg U \vee V)}_{\text{(i)}} \wedge \underbrace{(U \vee W)}_{\text{(ii)}} \rightarrow (V \vee W) \tag{4}$$

- Proof: Assume the LHS is true, so both (i) and (ii) are true.
  - If U is true V has to be true, for disjunction (i) to be true.
  - If U is false W has to be true, for disjunction (ii) to be true.

  In either case the RHS is true.

The resolution inference rule – full definition (ADVANCED)

- Input: two clauses $C = \mathtt{L}_1\ \ \mathtt{L}_2\ \ \ldots\ \ \mathtt{L}_n\,.$
  $D = \mathtt{M}_1\ \ \mathtt{M}_2\ \ \ldots\ \ \mathtt{M}_k\,.$

  where $\mathtt{L}_i\ =\ +\mathtt{X}$ and $\mathtt{M}_j\ =\ -\mathtt{X}$, or $\mathtt{L}_i\ =\ -\mathtt{X}$ and $\mathtt{M}_j\ =\ +\mathtt{X}$.
- Let $C' = C \setminus \{\mathtt{L}_i\}$, $D' = D \setminus \{\mathtt{M}_j\}$, where $\setminus$ denotes set difference.

  (The set difference $S_1 \setminus S_2$ is obtained by removing all elements of S_2 – if present – from S_1)

  Thus $C' = \mathtt{L}_1\ \ \ldots\ \ \mathtt{L}_{i-1}\ \ \mathtt{L}_{i+1}\ \ \ldots\ \ \mathtt{L}_n\,.$
  $D' = \mathtt{M}_1\ \ \ldots\ \ \mathtt{M}_{j-1}\ \ \mathtt{M}_{j+1}\ \ \ldots\ \ \mathtt{M}_k\,.$
- Resolution of C and D yields the clause $E = C' \cup D'$ (meaning $C' \vee D'$), called the $\mathit{resolvent}_{ij}(C, D)$, or simply $\mathit{resolvent}(C, D)$;

  $E = \mathtt{L}_1\ \ \ldots\ \ \mathtt{L}_{i-1}\ \ \mathtt{L}_{i+1}\ \ \ldots\ \ \mathtt{L}_n\ \ \mathtt{M}_1\ \ \ldots\ \ \mathtt{M}_{j-1}\ \ \mathtt{M}_{j+1}\ \ \ldots\ \ \mathtt{M}_k\,.$
  (with duplicates removed)
- Note that only a single pair of opposite literals is removed by the resolution step!

The resolution rule – remarks

- Informally: the resolution rule can be interpreted as viewing the clauses as arithmetic formulas, to be summed up and removing *exactly one* pair of “summands” `+X -X`
  - Example: *resolvent*(`+A-B-C`, `+B+D`) = `+A-C+D`
  - Remark: this analogy does not work, if there is a literal which occurs in both clauses,
    e.g. *resolvent*(`+A-B-C`, `+B+D+A`) = `+A-C+D` (only one `+A` is kept)
- The case of having two or more “summands” with opposite signs also breaks the analogy
  - Here only one pair of such summands is removed
  - Example: $\textit{resolvent}_{21}$(`+A-B-C`, `+B+D+C`) = `+A-C+D+C` = 1 (true), or
    $\textit{resolvent}_{33}$(`+A-B-C`, `+B+D+C`) = `+A-B+B+D` = 1
  - Thus resolution does not produce a meaningful clause in this case

Example: solving an inspector Craig puzzle using resolution

- The puzzle below is cited from “What Is The Name Of This Book?” by Raymond M. Smullyan, chapter “From the cases of Inspector Craig”
- Puzzles in this chapter involve suspects of a crime, named A, B, etc. Some of them are guilty, some innocent.
- Example:
  An enormous amount of loot had been stolen from a store. The criminal (or criminals) took the heist away in a car. Three well-known criminals A, B, C were brought to Scotland Yard for questioning. The following facts were ascertained:
  1. No one other than A, B, C was involved in the robbery.
  2. C never works without A (and possibly others) as an accomplice.
  3. B does not know how to drive.

  Is A innocent or guilty?

Inspector Craig puzzle – solution

- Let's recall the facts
  1. No one other than A, B, C was involved in the robbery.
  2. C never works without A (and possibly others) as an accomplice.
  3. B does not know how to drive.
- Transform each statement into a formula involving the letters A, B, C as atomic propositions. Proposition A stands for "A is guilty", etc.
  1. A is guilty or B is guilty or C is guilty: $A \vee B \vee C$
  2. If C is guilty then A is guilty: $C \rightarrow A$
  3. It cannot be the case that only B is guilty: $B \rightarrow (A \vee C)$
- Transform each propositional formula into conjunctive normal form (CNF), then show the clauses in simplified form:

	Original formula	CNF	Simplified clausal form
1	$A \vee B \vee C$	$A \vee B \vee C$	`+A +B +C.`
2	$C \rightarrow A$	$\neg C \vee A$	`-C +A.`
3	$B \rightarrow (A \vee C)$	$\neg B \vee A \vee C$	`-B +A +C.`

(Note that in general a single formula can give rise to multiple clauses.)

Inspector Craig puzzle – resolution proof

- Collect the clauses, give each a reference number and perform a resolution proof:

(1)	`+A +B +C.`	Only *A*, *B*, *C* was involved in the robbery.
(2)	`-C +A.`	*C* never works without *A* as an accomplice.
(3)	`-B +A +C.`	*B* does not know how to drive.
		resolve `(1)` lit 2 with `(3)` lit 1 resulting in `(4)`
(4)	`+A +C.`	resolve `(4)` lit 2 with `(2)` lit 1 resulting in `(5)`
(5)	`+A.`	

- We deduced that *A* is true, so the solution of the puzzle is: *A* is guilty
- Notice that `+A` occurs in each of the above clauses, hence each of `(1)`–`(4)` follows from `(5)`
- This, together with the fact that `(5)` follows from the input clauses `(1)`–`(3)`, means that `(5)` is equivalent to the set of input clauses
- Hence the statements of the puzzle impose no restrictions on propositions *B* and *C*
  (either can be guilty or innocent – all 4 combinations allowed)

Removing trivial consequences

Consider this set of clauses: $CS = \{$ `-B+C+D`, `+A+C`, `-A-B`, `+A-B+C` $\}$

- Find a clause in CS that is a consequence of another clause in CS.
- Hint: of these formulas, which implies which other? $U \vee V$, U, V?
  (If we know $U \vee V$ is true, can U be false?) Yes, it can.
  (If we know U is true, can $U \vee V$ be false?) No
- Hence U implies $U \vee V$, and similarly V implies $U \vee V$
- Viewing clauses as sets, if $C \subseteq D$, then $C \rightarrow D$ (“subset” $\rightarrow$ “whole set”)
- `+A+C` $\rightarrow$ `+A-B+C`, so `+A-B+C` is a trivial consequence of `+A+C`

Trivial consequences

- A clause $C \vee D$ ($D \neq$ empty) is said to be a trivial consequence of C
- Is it of interest to obtain the set of all consequences of CS?
- No, we get marred by trivial consequences, e.g. `-A-B-C`, `-A-B+C`, . . .
- It makes more sense to construct a maximal set of *non-trivial* consequences, i.e. a set MCS which contains all consequences of CS, except those that are a trivial consequence of a clause already in MCS
- Removing a trivial consequence is valid because $(C \wedge (C \vee D)) \equiv C$

Maximal set of non-trivial consequences (ADVANCED)

For the mathematically minded, here is a precise definition of the *maximal set of non-trivial consequences*

- For a set of clauses CS, its maximal set of consequences is MCS iff:
  - *each clause in MCS is a consequence of CS:*
    for each $C \in MCS$, $CS \rightarrow C$
  - *there are no trivial consequences in MCS:*
    for each $C_1, C_2 \in MCS$, C_2 is not a trivial consequence of C_1
  - *MCS contains all non-trivial consequences:*
    for each clause C such that $CS \rightarrow C$ holds, either $C \in MCS$ holds, or else C is a trivial consequence of a $C' \in MCS$.

Constructing *MCS* – continuing the example

- The set of input clauses:

```
(1) -B+C+D
(2) +A+C
(3) -A-B
(4) +A-B+C
```

- Remove `(4)`, as it is implied by `(2)`
- Resolve `(2)` with `(3)` producing a new clause:

```
(5) -B+C
```

- Remove `(1)`, as it is implied by `(5)`
- As no removal or resolution step can be applied, exit with the following maximal set of (non-trivial) consequences:

```
(2) +A+C
(3) -A-B
(5) -B+C
```

A saturation algorithm for obtaining *MCS* (ADVANCED)

Given a set of clauses CS_0, you can obtain its maximal set of consequences by performing the following algorithm:

1. set CS to CS_0
2. (exit if inconsistency is detected)
   if CS contains an empty clause, then exit reporting CS_0 is inconsistent
3. (remove a trivial consequence)
   if there are $C_1, C_2 \in CS$ such that C_2 is a trivial consequence of C_1, then remove C_2 from CS, and repeat step 3
4. (perform a meaningful resolution step)
   if there are $C_1, C_2 \in CS$ such that C_1 resolved with C_2 yields C_3 where $C_3 \neq true$ and $C_3 \notin CS$, then add C_3 to CS, and continue at step 3
5. (exit when saturated)
   as the conditions of both steps 3 and 4 failed, exit with $MCS = CS$

Finding a single consequence using an indirect proof

- For large sets of formulas finding all consequences is not viable
- Recall the Inspector Craig puzzle discussed earlier:

(1)	`+A +B +C.`	Only A, B, C was involved in the robbery.
(2)	`-C +A.`	C never works without A as an accomplice.
(3)	`-B +A +C.`	B does not know how to drive.

- To prove indirectly that `(1)`–`(3)` implies `A`, add $\neg$ `A` to the set of clauses:

(4)	`-A.`	... and perform resolutions, adding resolvents to the set
		`(4)/1 rw (1)/1` cl. `(4)` lit. `1` **r**esolved **w**ith cl. `(1)` lit. `1` $\Rightarrow$ `(5)`
(5)	`+B +C.`	`(5)/1 rw (3)/1` $\Rightarrow$ `(6)`
(6)	`+A +C.`	`(6)/1 rw (4)/1` $\Rightarrow$ `(7)`
(7)	`+C.`	`(7)/1 rw (2)/1` $\Rightarrow$ `(8)`
(8)	`+A.`	`(8)/1 rw (4)/1` $\Rightarrow$ `(9)`
(9)	$\square$	This denotes an empty disjunction $\equiv$ false

- Adding $\neg$`A` to `(1)`–`(3)` leads to contradiction, so {`(1)`,`(2)`,`(3)`} implies `A`
- This indirect proof is focused on proving the given statement

  Notice that the above proof is quite mechanical:
  - the first input clause is the result of the previous resolution step
  - we always resolve on the first literal of the first input clause

Inspector Craig puzzle – further proof attempts (ADVANCED)

```
(1)    +A +B +C.
(2)    -C +A.
(3)    -B +A +C.
```

- We now try to prove indirectly that ¬C follows from (1)–(3), by adding C:

(4)	+C.		implied clauses removed: (1), (3)
(5)	+A.	(4)/1 rw (2)/1	implied clauses removed: (2)

- The set {(4),(5)} is saturated, hence {(1)–(3)} does not imply ¬ C
- Let's now try to prove that C follows from (1)-(3), by adding ¬ C:

(6)	-C.		implied clauses removed: (2)
(7)	+A +B.	(6)/1 rw (1)/3	implied clauses removed: (1)
(8)	-B +A.	(6)/1 rw (3)/3	implied clauses removed: (3)
(9)	+A.	(7)/2 rw (8)/1	implied clauses removed: (7), (8)

- The set {(6),(9)} is saturated, hence {(1)–(3)} does not imply C
- We conclude that neither C nor its negation can be deduced from (1)–(3)
- (However, the two unsuccessful proofs put together show that no matter whether C is true or not, A has to be true. :-)