Knowledge-Based Systems

David Suendermann

http://suendermann.com

Baden-Wuerttemberg Cooperative State University

Stuttgart, Germany

General remarks

- **The most up-to-date version of this document as well as auxiliary material can be found online at**

  `http://suendermann.com`

- **Scripts and other materials by my colleague Dirk Reichardt covering some of the topics discussed in this lecture:**

  `http://wwwlehre.dhbw-stuttgart.de/~reichard/index.php?site=wbs`

Outline

- **logic and computer-assisted proof**
- **intelligent search and problem solving strategies**
- **expert systems and dialog systems**
- **Prolog**

Outline

- **logic and computer-assisted proof**
- **intelligent search and problem solving strategies**
- **expert systems and dialog systems**
- **Prolog**

Propositional logic: tautology

- **Ludwig Wittgenstein (1921): A tautology is a formula which is true in every possible interpretation.**
- **examples:**
  - $A \leftrightarrow A$
  - $A \vee \neg A$ **(excluded middle) [$\sqrt{2}$]**
  - $A \rightarrow B \leftrightarrow \neg B \rightarrow \neg A$ **(contraposition) [outlet]**
  - $A \rightarrow B \leftrightarrow \neg A \vee B$
  - $(A \rightarrow B) \wedge (A \rightarrow \neg B) \rightarrow \neg A$ **(reductio ad absurdum)**
  - $\neg(A \wedge B) \leftrightarrow \neg A \vee \neg B$ **(de Morgan's law)**
  - $(A \rightarrow B) \wedge (B \rightarrow C) \rightarrow (A \rightarrow C)$ **(syllogism)**
  - $(A \vee B) \wedge (A \rightarrow C) \wedge (B \rightarrow C) \rightarrow C$ **(proof by cases)**

Precedence and associativity of logical connectives

- **Outermost parenthesis can be dropped:**

$$(p \wedge q) \quad \Leftrightarrow \quad p \wedge q \tag{1}$$

- **Consider the following precedences:**

operator	precedence
$\neg$	1 (strongest)
$\wedge$	2
$\vee$	3
$\rightarrow$	4
$\leftrightarrow$	5 (weakest)

- **E.g., we have:**

$$(p \wedge q) \rightarrow (q \vee r) \quad \Leftrightarrow \quad p \wedge q \rightarrow q \vee r \tag{2}$$

- **Assume operators of the same precedence to be left-associative:**

$$(p \rightarrow q) \rightarrow r \quad \Leftrightarrow \quad p \rightarrow q \rightarrow r \tag{3}$$

Tautology: exercise

- **Prove that the following formula is a tautology**

$$A \wedge B \rightarrow C \leftrightarrow A \rightarrow (B \rightarrow C) \tag{4}$$

using

a) known equivalences,

b) a truth table.

Clauses

- **Multiple applications require sentences in propositional or 1st-order logic to be given as a (conjunctive) set of clauses.**
- **A clause is a disjunction of literals.**
- **A literal is an atomic formula or its negation.**
- **These conditions are fulfilled by the conjunctive normal form (CNF):**

$$\bigwedge_i \bigvee_j [\neg] P_{ij}. \tag{5}$$

- **example for a propositional formula in CNF:**

$$(\neg A \vee B \vee C) \wedge (A \vee B \vee \neg C) \tag{6}$$

- **Among other ways, we can convert a given propositional formula into CNF by**
  - **a) applying equivalences or**
  - **b) establishing a truth table.**
- **Example: We want to transform the formula**

$$A \rightarrow (B \leftrightarrow C) \tag{7}$$

into CNF.

a)

$$\begin{aligned} A \rightarrow (B \leftrightarrow C) &\Leftrightarrow A \rightarrow ((\neg B \vee C) \wedge (B \vee \neg C)) \\ &\Leftrightarrow \neg A \vee (\neg B \vee C) \wedge (B \vee \neg C) \\ &\Leftrightarrow (\neg A \vee \neg B \vee C) \wedge (\neg A \vee B \vee \neg C) \end{aligned} \tag{8}$$

Another way to put this CNF is the set notation:

$$\{\{\neg A, B, \neg C\}, \{\neg A, \neg B, C\}\}. \tag{9}$$

b) The conjunctive combination of all those clauses producing the result 0 in the truth table is the CNF.

A	B	C	$B \leftrightarrow C$	$A \rightarrow (B \leftrightarrow C)$	**clause**
0	0	0	1	1	$\neg A \vee \neg B \vee \neg C$
0	0	1	0	1	$\neg A \vee \neg B \vee C$
0	1	0	0	1	$\neg A \vee B \vee \neg C$
0	1	1	1	1	$\neg A \vee B \vee C$
1	0	0	1	1	$A \vee \neg B \vee \neg C$
1	0	1	0	0	$\neg A \vee B \vee \neg C$
1	1	0	0	0	$\neg A \vee \neg B \vee C$
1	1	1	1	1	$A \vee B \vee C$

- **So, we are getting the same CNF here, too:**

$$(\neg A \vee B \vee \neg C) \wedge (\neg A \vee \neg B \vee C) \qquad (10)$$

- **Inspector Watson is called to a jewelry store that has been subject to a robbery where three subjects, Austin, Brian, and Colin, were arrested.**
- **After evaluation of all facts, this is known:**

1. **At least one of the subjects is guilty:**
$$f_1 := A \vee B \vee C. \tag{11}$$
2. **If Austin is guilty he had exactly one accomplice:**
$$f_2 := A \rightarrow B \wedge \neg C \vee \neg B \wedge C. \tag{12}$$
3. **If Brian is innocent, so is Colin:**
$$f_3 := \neg B \rightarrow \neg C. \tag{13}$$
4. **If exactly two subjects are guilty, Colin is one of them. Hence, out of three possible pairs of subjects, there is only one impossible:**
$$f_4 := \neg(A \wedge B \wedge \neg C). \tag{14}$$
5. **If Colin is innocent then Austin is guilty:**
$$f_5 := \neg C \rightarrow A. \tag{15}$$

- **Question: Who are the culprits?**

- **A handy first step to approach this question is to turn all the involved formulas into CNF:**

$$\begin{aligned}
f_1 &\Leftrightarrow \{\{A, B, C\}\}\\
f_2 &\Leftrightarrow A \rightarrow B \wedge \neg C \vee \neg B \wedge C\\
&\Leftrightarrow \neg A \vee B \wedge \neg C \vee \neg B \wedge C\\
&\Leftrightarrow (\neg A \vee B \vee \neg B) \wedge (\neg A \vee B \vee C)\\
&\quad \wedge(\neg A \vee \neg C \vee \neg B) \wedge (\neg A \vee \neg C \vee C)\\
&\Leftrightarrow \{\{\neg A, B, C\}, \{\neg A, \neg C, \neg B\}\}\\
f_3 &\Leftrightarrow \{\{B, \neg C\}\}\\
f_4 &\Leftrightarrow \{\{\neg A, \neg B, C\}\}\\
f_5 &\Leftrightarrow \{\{C, A\}\} \qquad (16)
\end{aligned}$$

- **Now, to answer our question, there are again several possibilities:**

  a) resolution,

  b) truth table.

- **Resolution (introduced 1965 by John Robinson) is a method to test the validity of a formula or to find a solution to a set of assumptions.**
- **Resolution is defined in form of an algorithm and can, thus, be performed by a computer program.**
- **We are given two clauses of a propositional formula in CNF: C_1 and C_2.**
- **We assume there is a literal L which exists in C_1 and whose complement $\neg L$ exists in C_2.**
- **Then, we can derive a resolvent R by merging the original clauses eliminating the complimentary literals L and $\neg L$:**

$$\frac{\begin{array}{l} C_1 := A_1 \vee \ldots \vee A_n \vee L \\ C_2 := B_1 \vee \ldots \vee B_m \vee \neg L \end{array}}{\therefore R := A_1 \vee \ldots \vee A_n \vee B_1 \vee \ldots \vee B_m}$$

- **Exercise: Prove the resolution rule.**

Resolution algorithm

1. All sentences in the knowledge base (and the negation of a sentence we may want to prove, the so-called conjecture) are conjunctively connected.
2. The resulting sentence is transformed into CNF represented by the set S in set notation.
3. The resolution rule is applied to all possible pairs of clauses containing complimentary literals producing the resolvent R.
4. Repeated literals are removed from R.
5. If R contains complimentary literals, it is discarded. Otherwise, R is added to S, if it is not yet an element.
6. If the empty clause can be derived after an application of the resolution rule, we have proven contradiction. This can either mean that the knowledge base is inconsistent or that the negation of the sentence we tried to prove is unsatisfiable, i.e., the conjecture follows from the knowledge base.

- **In our example, we want to find a solution to the facts in our knowledge base:**

$$K := \{\underbrace{\{A,B,C\}}_{a}, \underbrace{\{\neg A,B,C\}}_{b}, \underbrace{\{\neg A,\neg C,\neg B\}}_{c}, \underbrace{\{B,\neg C\}}_{d}, \underbrace{\{\neg A,\neg B,C\}}_{e}, \underbrace{\{A,C\}}_{f}\}$$

$$\begin{array}{lllllll} \{a,b\} & \rightarrow & \{B,C\} =: g & \quad & \{c,e\} & \rightarrow & \{\neg A,\neg B\} =: l \\ \{a,d\} & \rightarrow & \{A,B\} =: h & & \{d,g\} & \rightarrow & \{B\} =: m \\ \{b,d\} & \rightarrow & \{\neg A,B\} =: i & & \{e,f\} & \rightarrow & \{\neg B,C\} =: n \\ \{b,e\} & \rightarrow & \{\neg A,C\} =: j & & \{f,j\} & \rightarrow & \{C\} =: o \\ \{c,d\} & \rightarrow & \{\neg A,\neg C\} =: k & & \{i,l\} & \rightarrow & \{\neg A\} =: p \end{array}$$

- **In conclusion, we find that Brian and Colin are guilty, Austin is not.**
- **We have to systematically try all combinations of clauses when searching for a solution since if any of them had resulted in an empty clause, we would have found that the knowledge base has no solution, i.e., it is inconsistent in itself.**

- **We can derive the same solution by means of a truth table:**

A	B	C	a	b	c	d	e	f	K
0	0	0	0	1	1	1	1	0	0
0	0	1	1	1	1	0	1	1	0
0	1	0	1	1	1	1	1	0	0
0	1	1	1	1	1	1	1	1	1
1	0	0	1	0	1	1	1	1	0
1	0	1	1	1	1	0	1	1	0
1	1	0	1	1	1	1	0	1	0
1	1	1	1	1	0	1	1	1	0

Resolution: example (cont.)

- **Let us now try to prove whether Brad or Colin are culprits, so, we now have to take the CNF of all the facts from our knowledge base**

$$\{\underbrace{\{A,B,C\}}_{a},\underbrace{\{\neg A,B,C\}}_{b},\underbrace{\{\neg A,\neg C,\neg B\}}_{c},\underbrace{\{B,\neg C\}}_{d},\underbrace{\{\neg A,\neg B,C\}}_{e},\underbrace{\{A,C\}}_{f}\}$$

and the CNF of the negated conjecture $J := B \vee C$, i.e.,

$$\neg J \Leftrightarrow \neg(B \vee C) \Leftrightarrow \{\underbrace{\{\neg B\}}_{g},\underbrace{\{\neg C\}}_{h}\} \qquad (17)$$

and try to derive an empty clause:

$$\begin{array}{lcl} \{b,g\} & \rightarrow & \{\neg A, C\} =: i \\ \{f,i\} & \rightarrow & \{C\} =: j \\ \{h,j\} & \rightarrow & \{\} \end{array}$$

- **In conclusion, we were able to prove that Brad or Colin are culprits.**

- **Again, the same result can be found when consulting the truth table:**

A	B	C	a	b	c	d	e	f	g	h	$K \wedge \neg J$
0	0	0	0	1	1	1	1	0	1	1	0
0	0	1	1	1	1	0	1	1	1	0	0
0	1	0	1	1	1	1	1	0	0	1	0
0	1	1	1	1	1	1	1	1	0	0	0
1	0	0	1	0	1	1	1	1	1	1	0
1	0	1	1	1	1	0	1	1	1	0	0
1	1	0	1	1	1	1	0	1	0	1	0
1	1	1	1	1	0	1	1	1	0	0	0

- **1st-order logic (aka as predicate logic) is an extemsion to propositional logic.**
- **Main difference is its additional use of predicates, functions and quantifiers.**
- **A predicate returns boolean, a function non-boolean values.**
- **A quantifier is an operator defining the scope of variables:**
  - $\forall$ **is the universal quantifier,**
  - $\exists$ **is the existential quantifier.**

- **example functions:**
  - $+$ **as in** $x + y$
  - **any constant such as the numeral** 1
  - **age**(x) **returning the age of the object** x
- **example predicates:**
  - $>$ **as in** $x > y$
  - **propositional variables**
  - $\top$ **and** $\bot$
  - **isStudent**(x) **returning** $\top$ **iff** x **is a student**
- **terms:**
  1. **Any variable is a term.**
  2. **Any expression** $f(t_1, \ldots, t_n)$**, with the** n**-ary function symbol** f **and the terms** $t_1, \ldots, t_n$**, is a term.**

1st-order logic: examples on how to translate natural language into formulas

- **All students are smart:**

$$\forall x(\textbf{isStudent}(x) \rightarrow \textbf{isSmart}(x)). \quad (18)$$

- **There is a smart student:**

$$\exists x(\textbf{isStudent}(x) \wedge \textbf{isSmart}(x)). \quad (19)$$

- **Every student who takes Knowledge-Based Systems repeats the fundamentals of Logic.**

$$\forall x(\textbf{isStudent}(x) \wedge \textbf{takes}(x, \textbf{kbs}) \rightarrow \textbf{repeats}(x, \textbf{logic})). \quad (20)$$

- **Every student loves some student:**

$$\forall x(\textbf{isStudent}(x) \rightarrow \exists y(\textbf{isStudent}(y) \wedge \textbf{loves}(x, y))). \quad (21)$$

- **Billy has one brother:**

$$\exists x(\textbf{isBrotherOf}(x, \textbf{billy}) \wedge \forall y(\textbf{isBrotherOf}(y, \textbf{billy}) \rightarrow x = y)). \quad (22)$$

- **Modal logic extends the standards of formal logic with elements of modality:**
  - **possibility (Kripke 1959: "possible worlds"; operator $\Diamond$),**
  - **necessity (operator $\Box$).**
- **Each of them can be represented by the other with negation:**

$$\Diamond\varphi \leftrightarrow \neg\Box\neg\varphi, \tag{23}$$

$$\Box\varphi \leftrightarrow \neg\Diamond\neg\varphi. \tag{24}$$

- **modal operators and quantifiers—the Barcan formulæ:**

$$\exists x\Diamond\varphi \rightarrow \Diamond\exists x\varphi \tag{25}$$

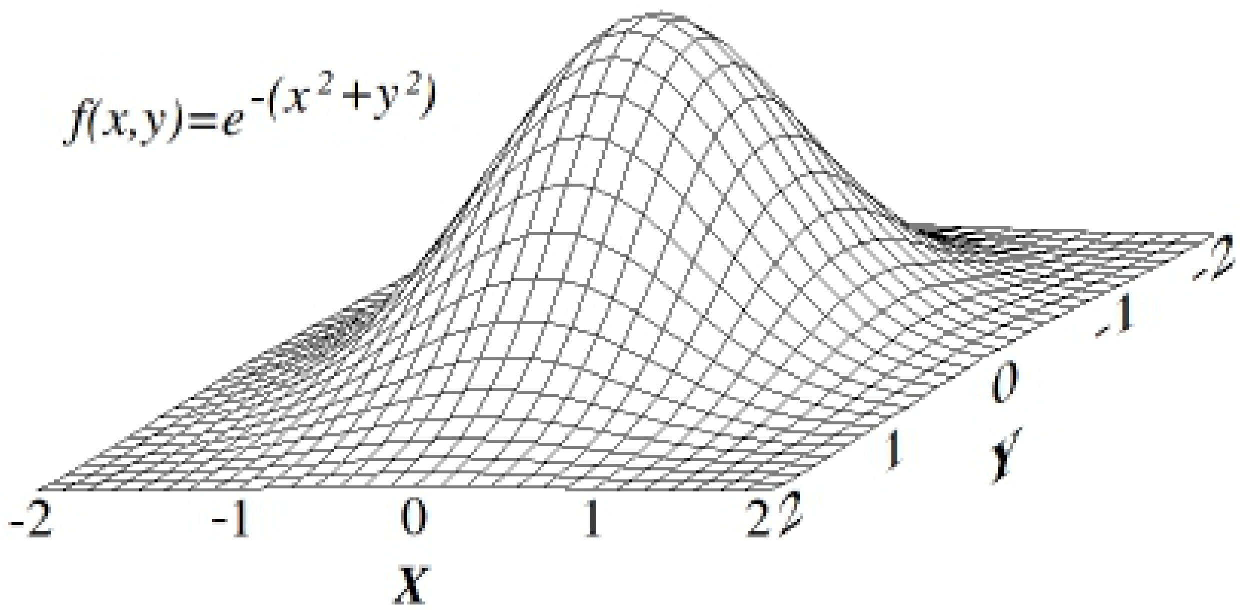
$$\Diamond\exists x\varphi \stackrel{?}{\rightarrow} \exists x\Diamond\varphi \quad \textbf{(Wittgenstein's son)} \tag{26}$$

- **logic and computer-assisted proof**
- **intelligent search and problem solving strategies**
- **expert systems and dialog systems**
- **Prolog**

- Problem solving is the search for a solution in a given scenario.
- Questions raised by search algorithms at runtime include
  - How good am I at the moment?
  - How do I estimate what is still missing?
- Popular search families are
  - local search (e.g. hill climbing)
  - graph and tree traversal
    - depth-first search (DFS)
    - breadth-first search (BFS)
    - A*

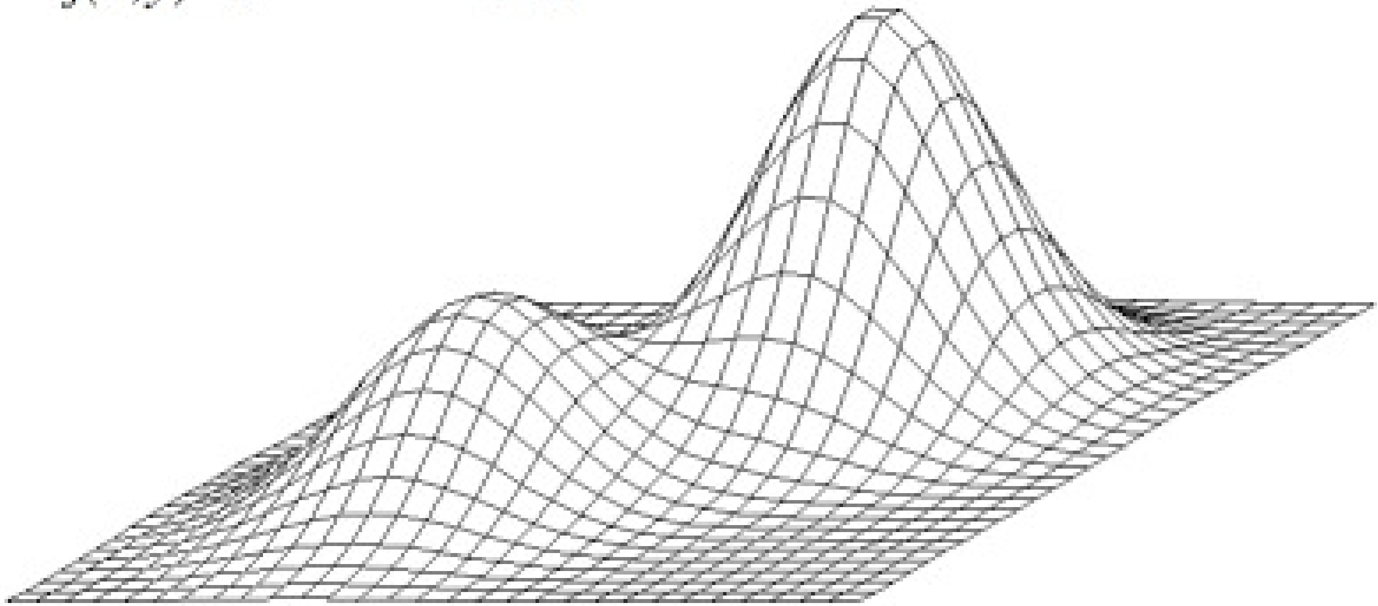
- **Hill climbing** is an **iterative** algorithm that
  1. starts with an **arbitrary solution** $x = x_0$ to the problem with a performance $f(x)$,
  2. incrementally changes a single element of x resulting in x',
  3. if the change improved the solution (i.e., $f(x') > f(x)$), then the solution is updated ($x := x'$), and the algorithm continues at Step 2.
  4. If no further improvement can be produced, the algorithm stops.

- Hill-climbers are well suited for convex surfaces.
- They will converge to the global optimum.



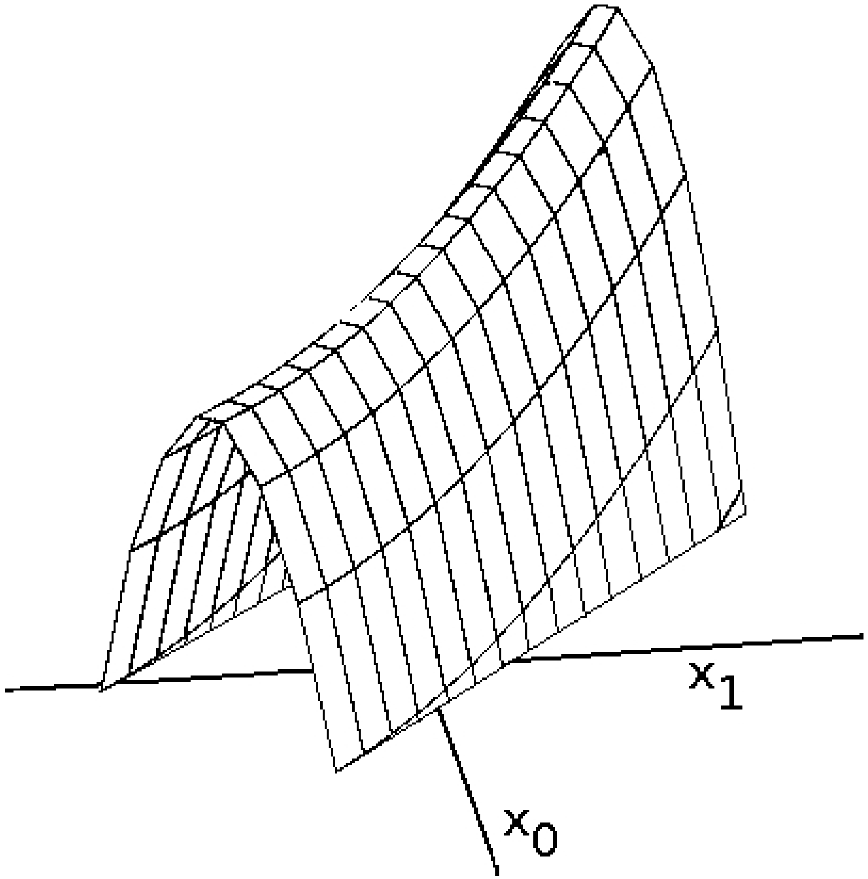
- **Hill climbing will find only local maxima.**
- **Hence, if $f(x)$ is not convex, it may not find the global optimum.**
- **E.g., if the algorithm starts at a poor location in the following example, it may not converge to the global maximum:**

$$f(x,y)=e^{-(x^2+y^2)} + 2e^{-((x-1.7)^2+(y-1.7)^2)}$$



- **Stochastic hill climbing, random walks, or simulated annealing try to overcome this problem.**

- **Hill climbers adjust one vector element at a time.**
- **So, each step will move in an axis-aligned direction.**
- **If $f(x)$ features a narrow ridge ascending in a non-axis direction, the climber has to zig-zag.**
- **If the ridge's sides are very steep, the climber has to take tiny steps and, therefore, may take an unreasonable time to ascend.**
- **Gradient descend methods can overcome this effect when $f(x)$ is differentiable.**
- **Another problem is when the search space is flat around the current search position (plateau).**



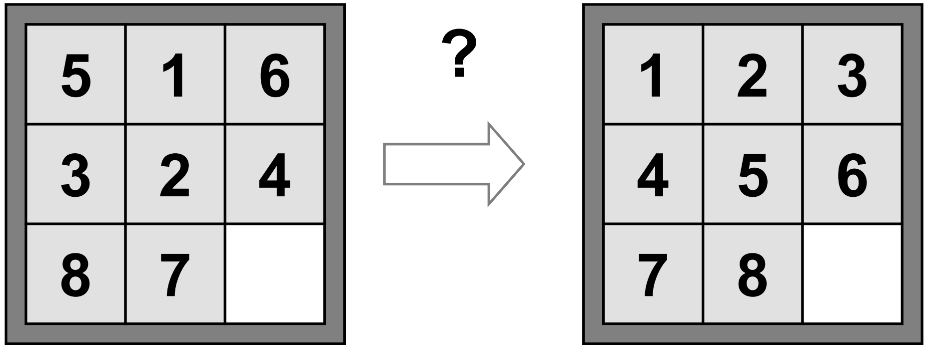
Simulated annealing

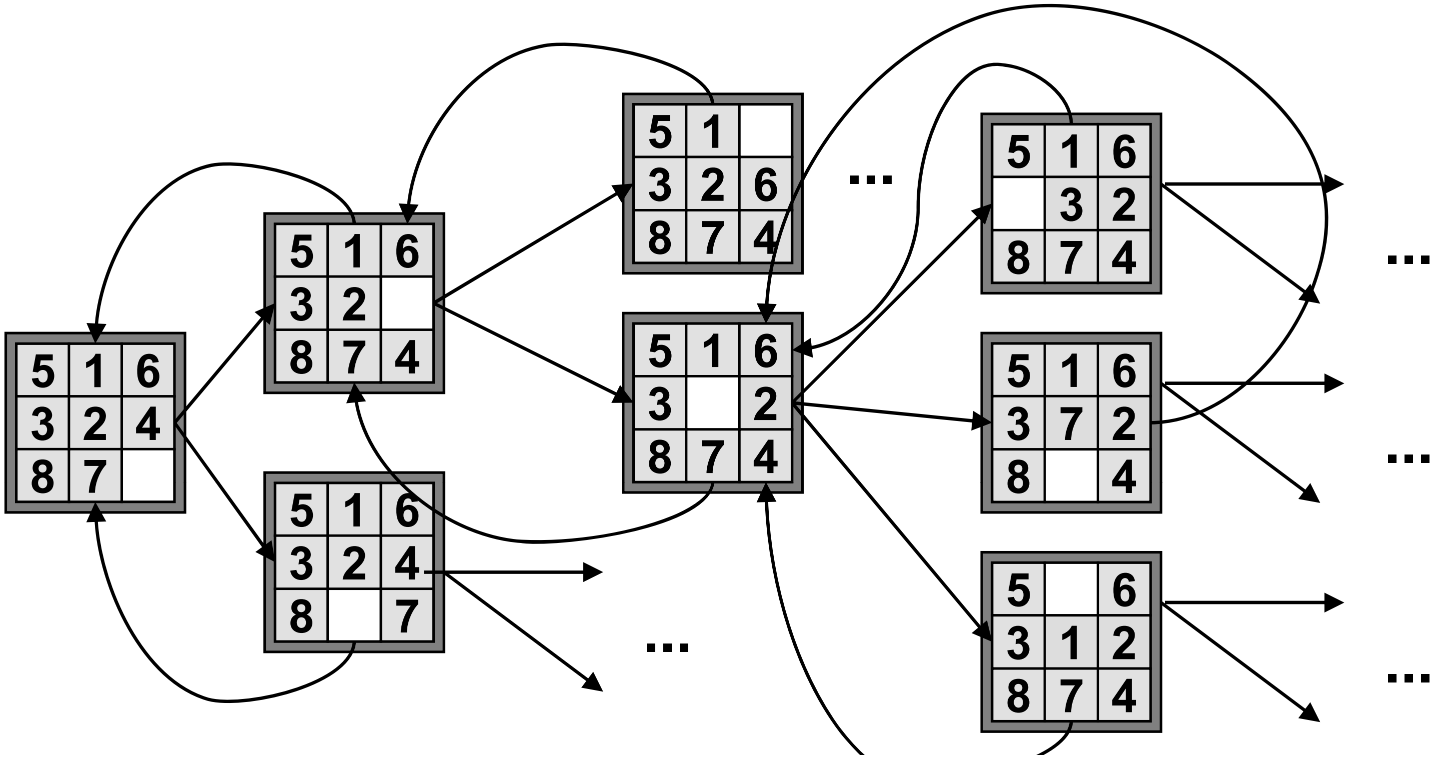
- Simulated annealing (SA) is a probabilistic heuristic to find the global optimum of $f(x)$.
- Name and inspiration come from the annealing in metallurgy.
- Each step of the SA algorithm replaces x with a random nearby x'.
- The randomization is based on a probability that depends on
  - $f(x) - f(x')$ and
  - the temperature T, a parameter gradually decreased during the process.
- Due to the randomness of picking x', the method can escape local optima.
- SA does not guarantee to reach the global optimum but increases chances to do so.

Graph traversal

- **Graph traversal refers to a search algorithm visiting the nodes in a graph in a particular manner.**
- **Starting at a root node S, all children are generated and added to an open list.**
- **If all children of S are generated, S gets removed from the open list and added to a closed list.**
- **Generation and expansion are performed until a goal node or leaf is found.**

Graph traversal: 8-puzzle

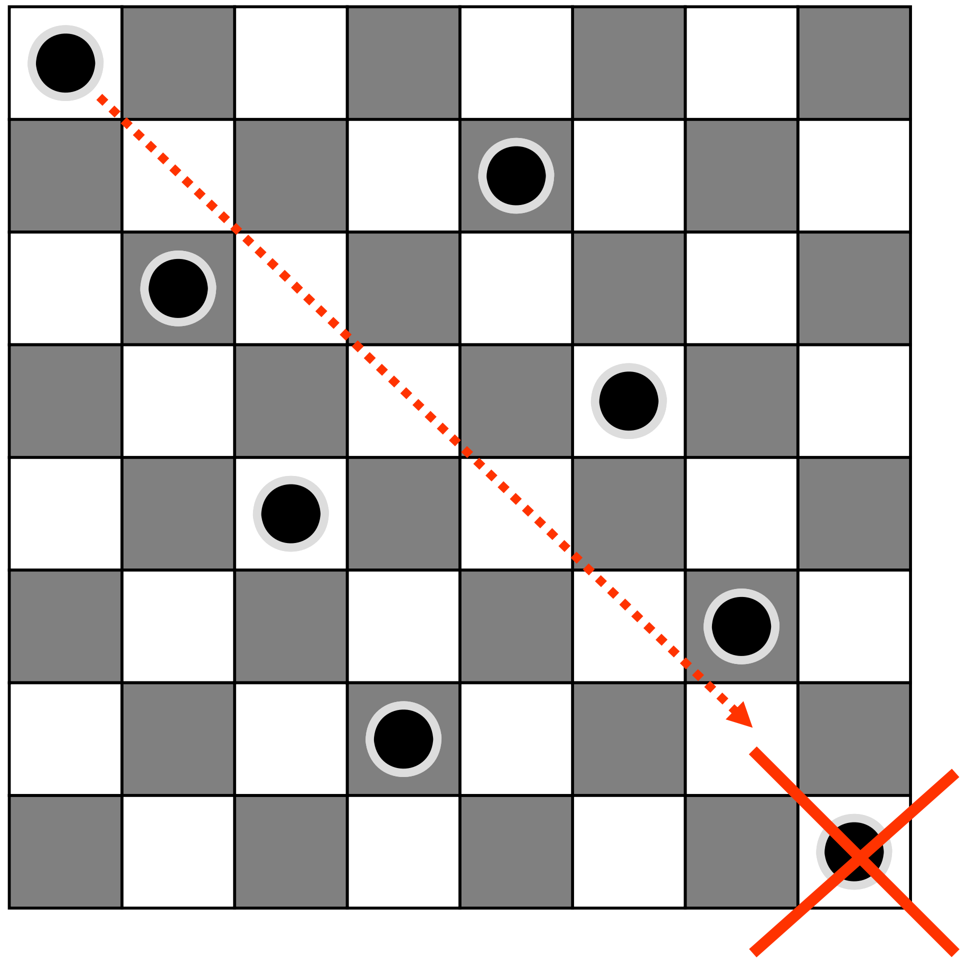




- **Place 8 chess queens on a chessboard that no two queens attack each other.**
- **There are**

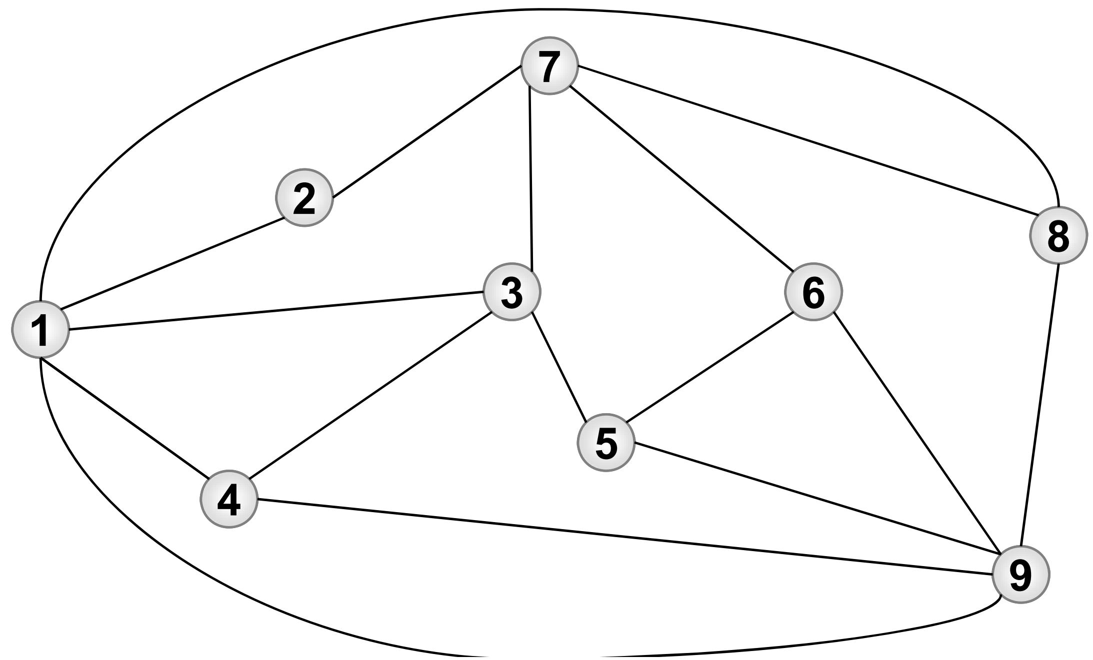
$$\binom{64}{8} = 4,426,165,368$$

possible arrangments.
- **But only 92 solutions.**



Graph traversal: travelling salesman problem (TSP)

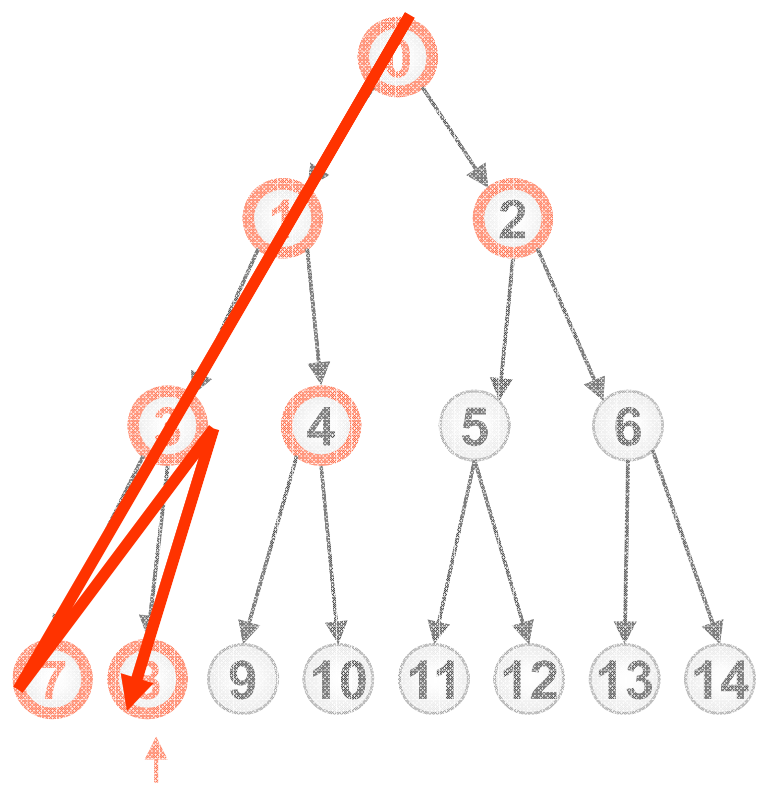
- **Given a list of cities and their pairwise distances, find the shortest possible tour visiting each city exactly once.**
- **TSP is an NP-hard problem and belongs to the most intensively studied ones in optimization.**

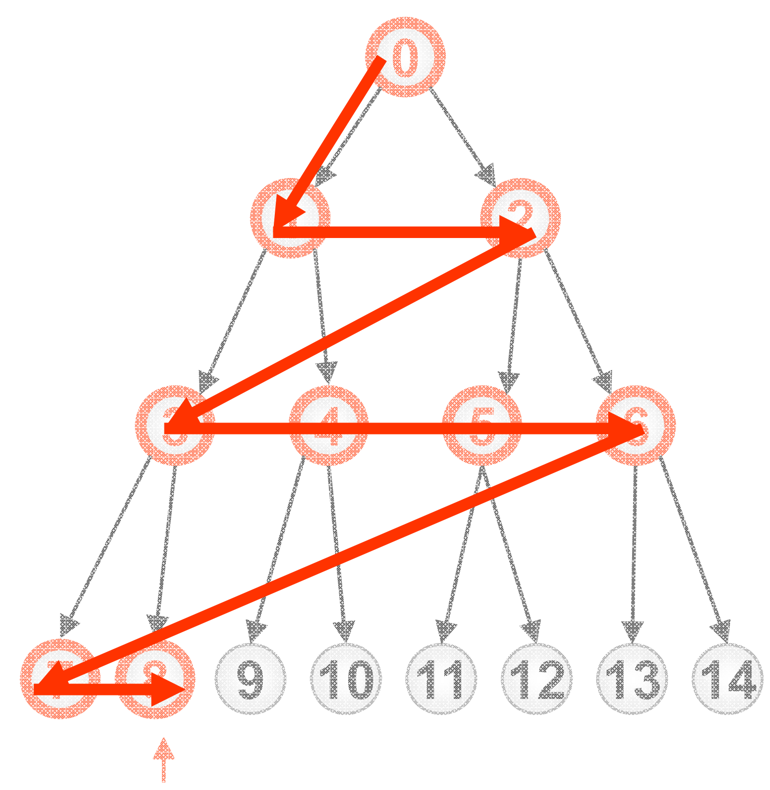


Graph search algorithms

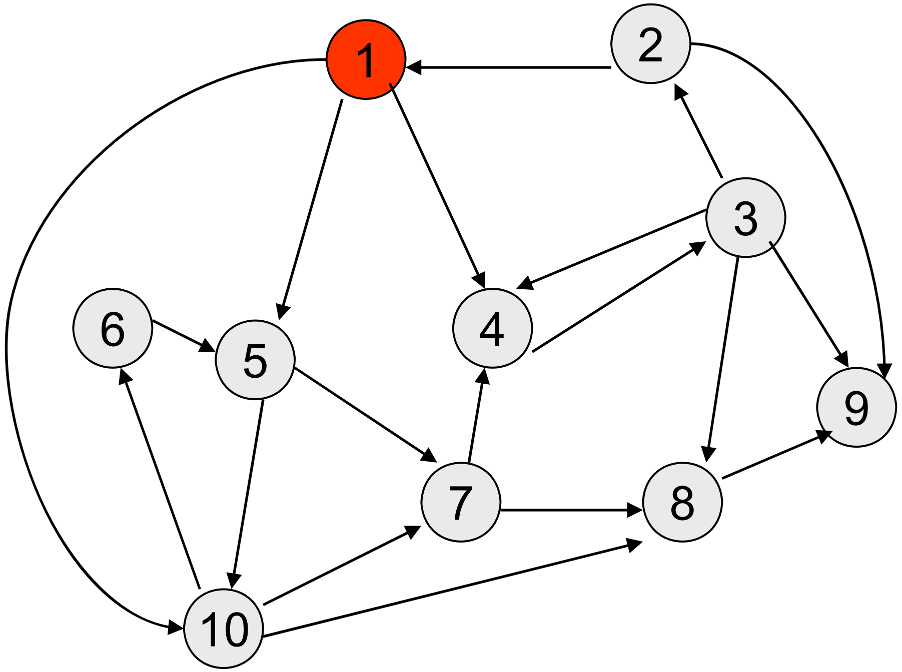
- **exhaustive search algorithms**
  - **depth-first search (DFS)**
  - **breadth-first search (BFS)**
  - **backtracking**
- **heuristic and statistical search algorithms**
  - **best-first search**
  - **A***
  - **minimax algorithm**

Exhaustive search algorithms: DFS vs. BFS

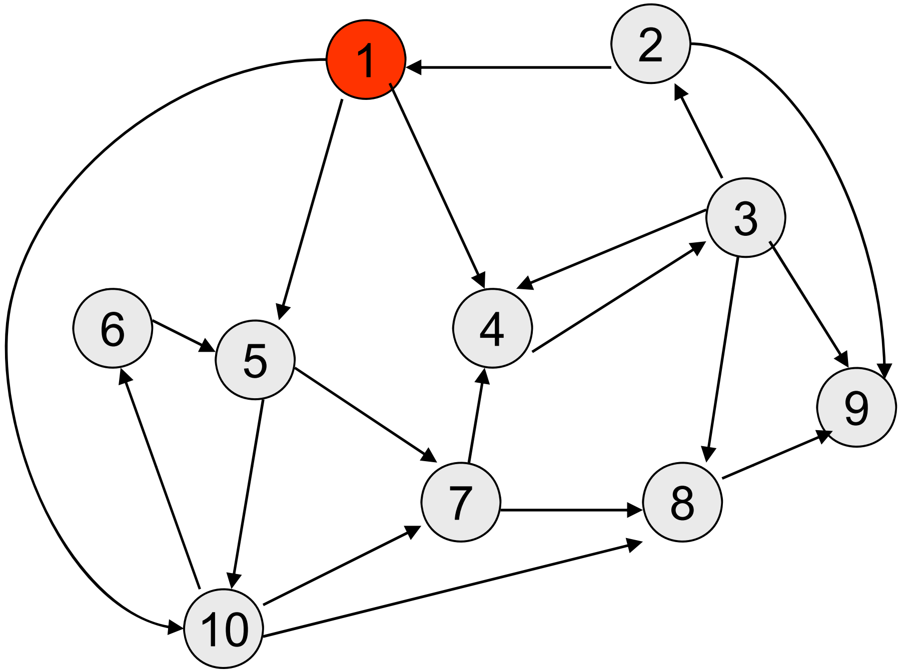




DFS: example



OPEN (stack)	CLOSED
1	
4, 5, 10	1
3, 5, 10	1, 4
2, 8, 9, 5, 10	1, 4, 3
9, 8, 9, 5, 10	1, 4, 3, 2
8, 9, 5, 10	1, 4, 3, 2, 9
5, 10	1, 4, 3, 2, 9, 8
7, 10, 10	1, 4, 3, 2, 9, 8, 5
10, 10	1, 4, 3, 2, 9, 8, 5, 7
6, 10	1, 4, 3, 2, 9, 8, 5, 7, 10
	1, 4, 3, 2, 9, 8, 5, 7, 10, 6



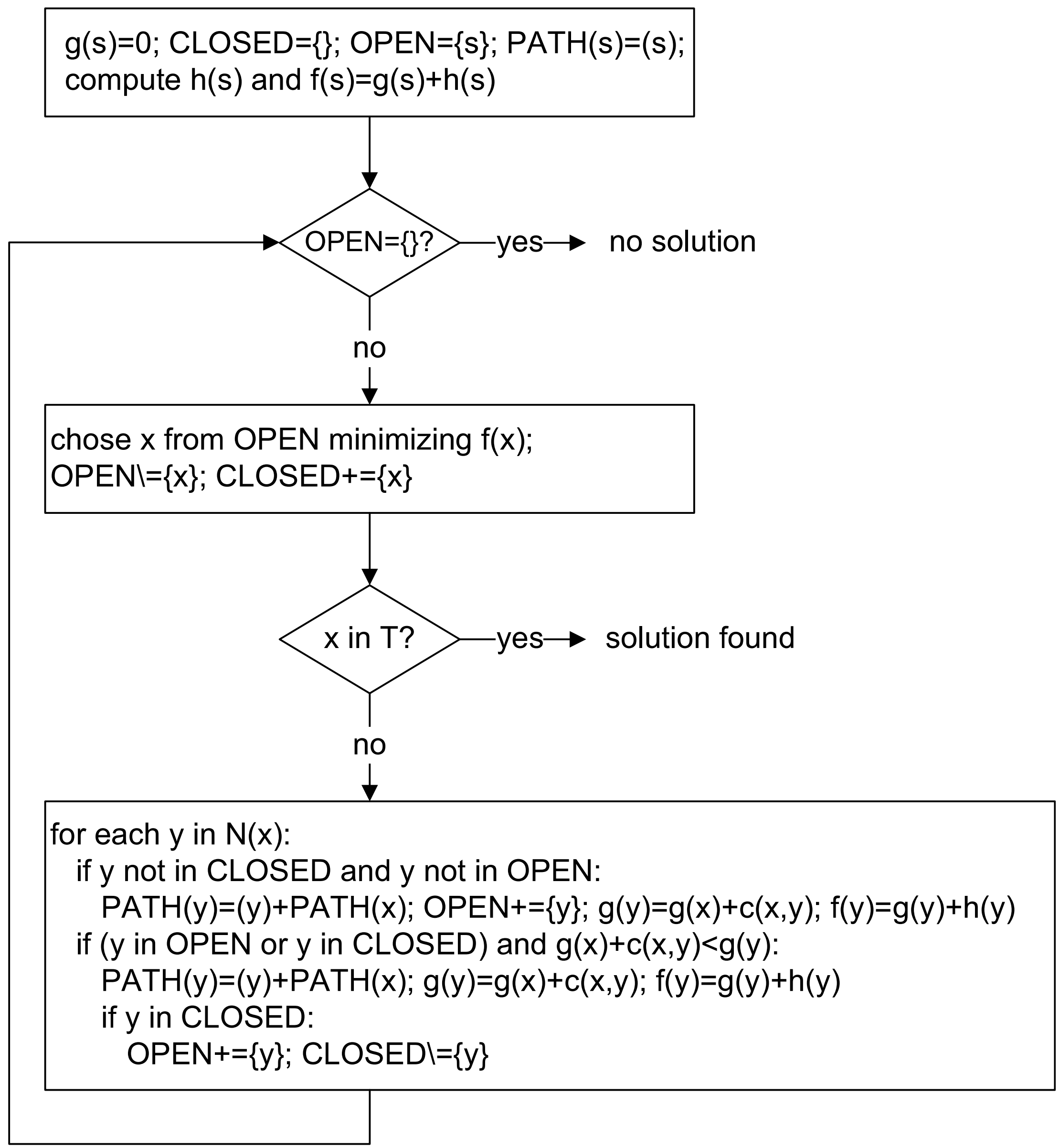
OPEN (queue)	CLOSED
1	
4, 5, 10	1
5, 10, 3	1, 4
10, 3, 7	1, 4, 5
3, 7, 6, 8	1, 4, 5, 10
7, 6, 8, 2, 9	1, 4, 5, 10, 3
6, 8, 2, 9	1, 4, 5, 10, 3, 7
8, 2, 9	1, 4, 5, 10, 3, 7, 6
2, 9	1, 4, 5, 10, 3, 7, 6, 8
9	1, 4, 5, 10, 3, 7, 6, 8, 2
	1, 4, 5, 10, 3, 7, 6, 8, 2, 9

Exhaustive search algorithms: modifications

- **depth-limited search**
  - works exactly like DFS, but imposes a maximum limit on the depth of the search.
- **iterative deepening**
  - runs a depth-limited search repeatedly.
  - In doing so, it increases the depth limit with each iteration until reaching d, the maximum depth.
- **bidirectional search**
  - runs two instances of BFS, one from the initial node, one from the goal node.
  - A solution is found when both instancs hit an identical node.
  - Compared to pure BFS, the complexity of this algorithm can be significantly lower, e.g. $O(b^{d/2})$ rather than $O(b^d)$ with the **branching factor** b.

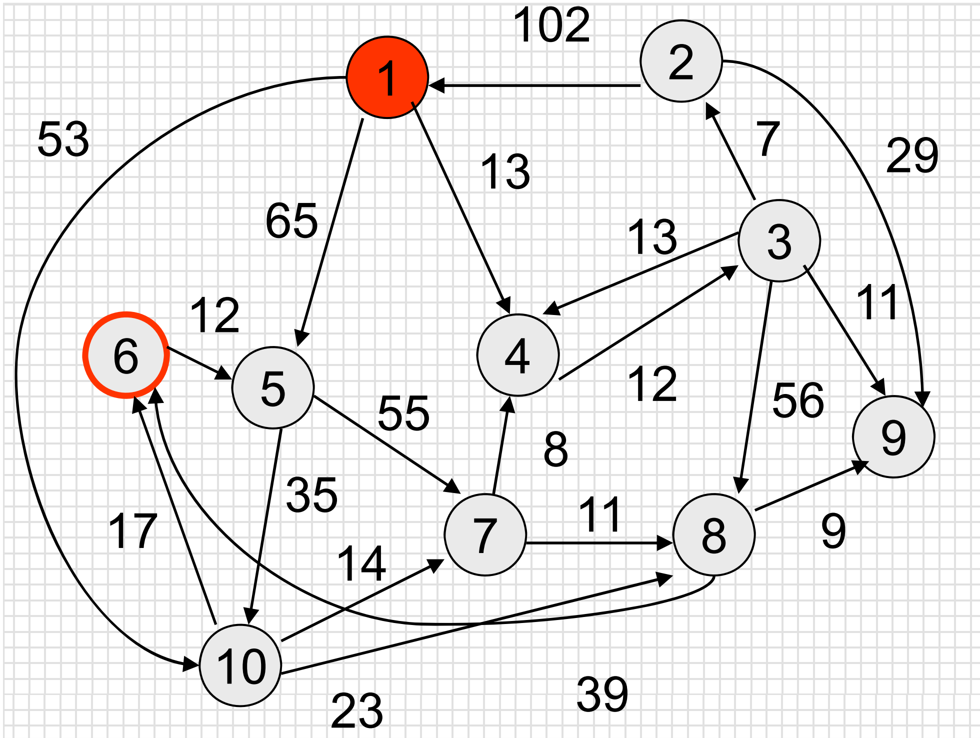
- **involved terms:**
  - s: **starting node**
  - T: **the set of goal nodes**
  - $c(x, y)$: **cost from x to y**
  - $g(x)$: **cost from the starting node to the current node x**
  - $h(x)$: **a heuristic estimate of the distance to the goal**
  - $f(x)$: **distance-plus-cost heuristic to determine the order in which to visit nodes in the tree**
  - $N(x)$: **set of neighbor nodes of x**
  - **CLOSED: closed set**
  - **OPEN: open set**
  - **PATH**(x): **path to node x**

A*: algorithm



g(s)=0; CLOSED={}; OPEN={s}; PATH(s)=(s);
compute h(s) and f(s)=g(s)+h(s)
OPEN={}?
yes
no solution
no
chose x from OPEN minimizing f(x);
OPEN\={x}; CLOSED+={x}
x in T?
yes
solution found
no
for each y in N(x):
if y not in CLOSED and y not in OPEN:
PATH(y)=(y)+PATH(x); OPEN+={y}; g(y)=g(x)+c(x,y); f(y)=g(y)+h(y)
if (y in OPEN or y in CLOSED) and g(x)+c(x,y)<g(y):
PATH(y)=(y)+PATH(x); g(y)=g(x)+c(x,y); f(y)=g(y)+h(y)
if y in CLOSED:
OPEN+={y}; CLOSED\={y}

A*: example



x	1	2	3	4	5	6	7	8	9	10
$h(x)$	22	94	88	34	88	4	8	5	1	0

A*: example (cont.)

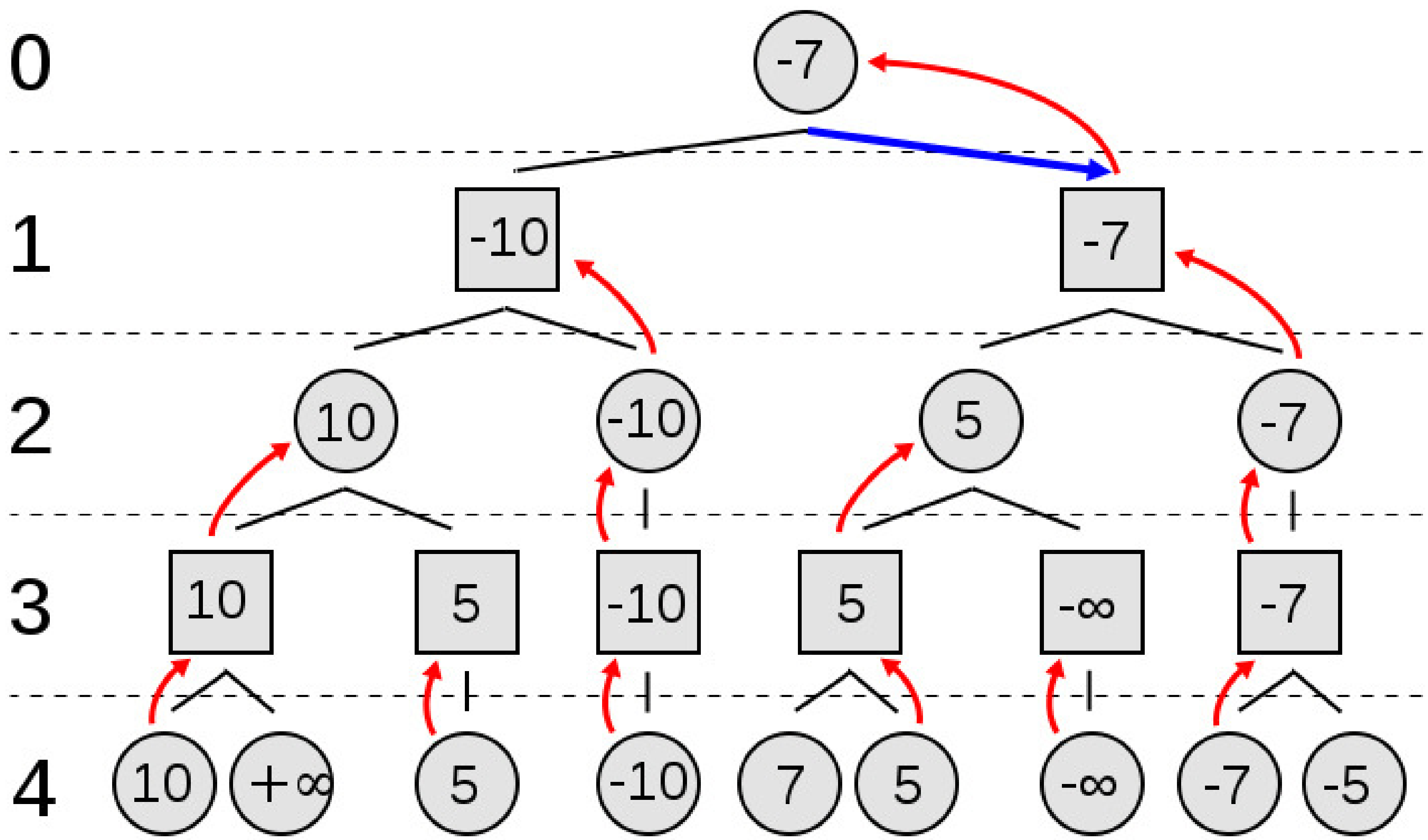
step	x	$N(x)$	PATH(x)	OPEN	CLOSED	$g(x)$	$f(x)$
init	1	{4,5,10}	(1)	{1}	{}	0	22
pick min	1	{4,5,10}	(1)	{}	{1}	0	22
iterate	4	{3}	(4,1)	{4}	{1}	13	47
	5	{7,10}	(5,1)	{4,5}	{1}	65	153
	10	{6,7,8}	(10,1)	{4,5,10}	{1}	53	53
pick min	4	{3}	(4,1)	{5,10}	{1,4}	13	47
iterate	3	{2,4,8,9}	(3,4,1)	{3,5,10}	{1,4}	25	113
pick min	10	{6,7,8}	(10,1)	{3,5}	{1,4,10}	53	53
iterate	6	{5}	(6,10,1)	{3,5,6}	{1,4,10}	70	74
	7	{4,8}	(7,10,1)	{3,5,6,7}	{1,4,10}	67	75
	8	{6,9}	(8,10,1)	{3,5,6,7,8}	{1,4,10}	76	81
pick min	6	{5}	(6,10,1)	{3,5,7,8}	{1,4,6,10}	70	74

Minimax search

- **Originally formulated for two-player game theory.**
- **Each game situation is a state, i.e. a node in a graph.**
- **Assumption: The opponent always chooses the best-possible move.**
- **The minimax principle:**
  - **One's move maximizes one's winning probability.**
  - **The opponent's move minimizes one's winning probability.**

- **Can minimax be applied to chess?**
- **Not without further assumptions since the state space is too large.**
- **Possible solutions:**
  - **limited search depth,**
  - **heuristic cost/reward functions.**
- **Heuristic cost/reward functions do not reward 1/0 for winning/losing but try to find a reasonable approximation.**

Minimax algorithm: example



- **pic:**
  - **source:** `http://en.wikipedia.org/wiki/Image:Minimax.svg`
  - **author: Nuno Nogueira**
  - **license: Creative Commons Attribution-Share Alike 2.5 Generic**

Outline

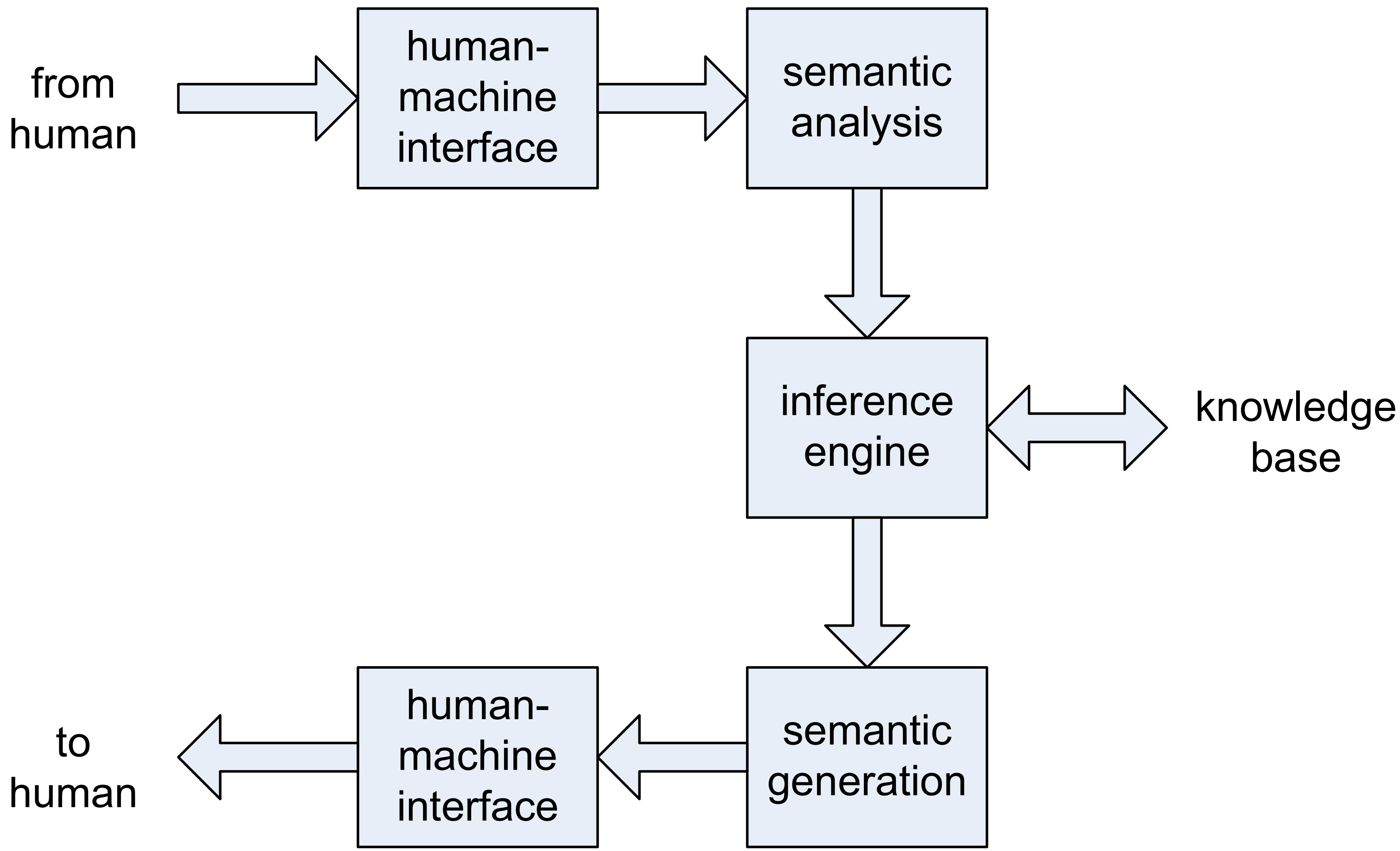
- **logic and computer-assisted proof**
- **intelligent search and problem solving strategies**
- **expert systems and dialog systems**
- **Prolog**

Expert systems: introduction

- An expert system (XPS) is a computer program emulating the decision making of human experts.
- XPSs are one of the most popular applications of artificial intelligence.
- In contrast to conventional software, an XPS is designed to solve complex problems by reasoning about knowledge.
- Accordingly, the two main components of an XPS are
  - the inference engine
  - the knowledge base

  At runtime, an XPS has to communicate with a human user, so it also requires
  - human-machine interfaces for in- and output.

Expert systems: architecture



Mycin

- **XPS designed to identify bacteria causing severe infections (e.g. meningitis).**
- **Mycin also recommended medication (antibiotics) adjusted to the patient's characteristics.**
- **based on the PhD thesis of a student at Stanford University in the early 1970s**
- **The knowledge base consisted of about 600 rules established with the help of medical experts.**
- **A performance test resulted in 69% good recommendations outperforming infectious desease experts from Stanford's medical school.**
- **Mycin was not released to the real world.**
- **One of the reasons is that of the reliability of medical decisions made which is particularly crucial in the U.S.**

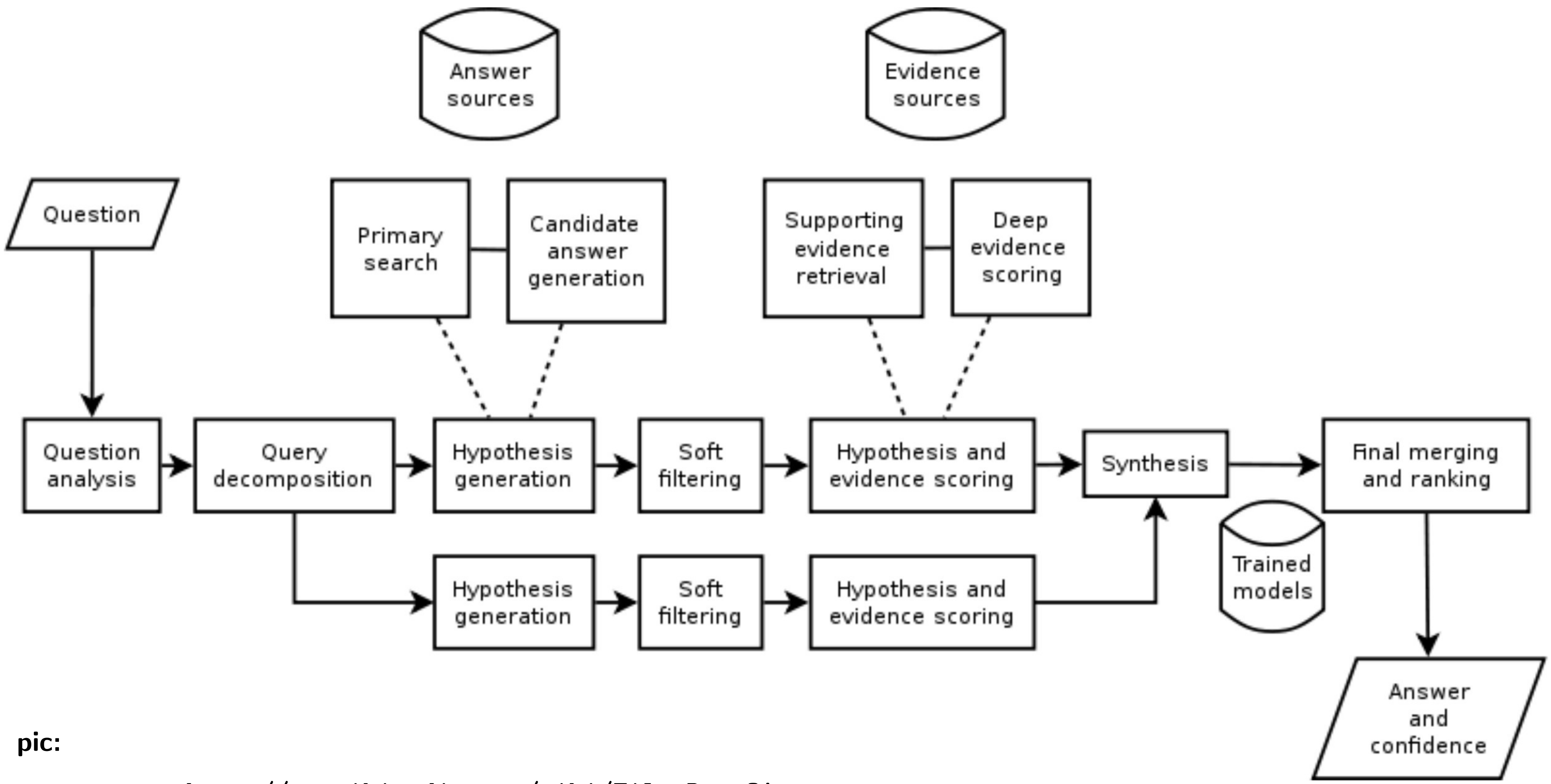
Deep Blue

- chess-playing computer by IBM
- On May 11, 1997, Deep Blue won a six-game match against Garry Kasparov.
- based on brute-force computing power (30 nodes with 480 VLSI chess chips)
- written in C under AIX
- The evaluation function contained multiple parameters tuned on 700,000 grandmaster games.
- pic:
  - source: `http://flickr.com/photos/22453761@N00/592436598/`
  - author: James the photographer
  - license: Creative Commons Attribution 2.0 Generic

Watson

- Watson is an AI computer system from IBM for question answering.
- It combines applications of
  - machine learning,
  - NLP,
  - information retrieval,
  - knowledge representation,
  - reasoning.
- To showcase its abilities, in February 2011, Watson competed on the show Jeopardy! against the human champions and won.
- During the quiz, Watson had no access to the Internet.
- It had access to 200M pages of structured and unstructured data (including a copy of the entire Wikipedia), amounting to 4TB.
- Hardware consisted of
  - 90 IBM Power 750 servers with 2880 processors and 16TB of RAM.

Watson: architecture



- **pic:**
  - **source:** `http://en.wikipedia.org/wiki/File:DeepQA.svg`
  - **author: Pgr94**
  - **license: Creative Commons CC0 1.0 Universal Public Domain Dedication**

- **Traditionally, the knowledge base stores knowledge in a computer-readable manner (e.g. using SQL, logical formulas).**
- **In the case of an XPS, the knowledge base can be composed of heterogenious sources such as**
  - **expert knowledge encoded by knowledge engineers (e.g. by interviewing physicians, chess masters, or call center agents etc.—depending on the XPS's domain)**
  - **structured data derived from encyclopedias, directories, cataloges, Wordnet, etc.**
  - **unstructured data (as provided by FAQs, scientific articles, Wikipedia, or the WWW)**
  - **probabilistical models (automatically) learned from structured and unctructured data (e.g., statistics of survival rates given patients' symptoms and medication, winning probabilities given a game scenario, caller behavior and state etc.)**

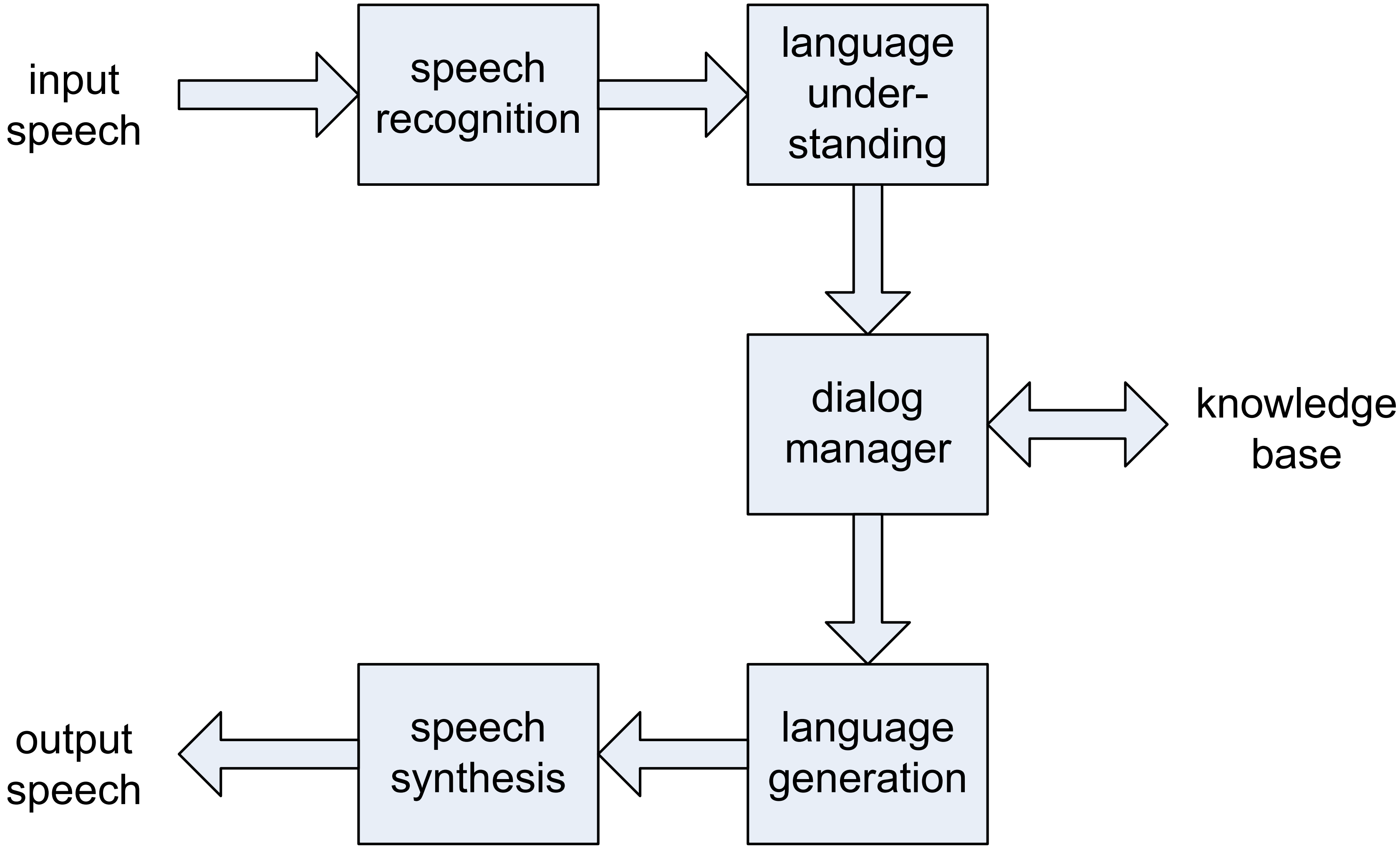
Knowledge base: examples

- *If something is living then it is mortal.* **(turn into 1st-order logic)**
- *If somebody's age is known then his birth date is today's date minus his age.* **(turn into SQL)**
- *IF the identity of the germ is not known with certainty AND the germ is gram-positive AND the morphology of the organism is "rod" AND the germ is aerobic THEN there is a strong probability (0.8) that the germ is of type enterobacteriacae.* **(Mycin rule)**

Expert systems: inference engine

- **The inference engine evaluates rules and/or statistics provided by the knowledge base to produce a reasoning.**
- **It can be based on (a combination of)**
  - **propositional logic (0th-order XPS)**
  - **other types of logic (predicate, modal, temporal, fuzzy)**
  - **classification (e.g. decision trees)**
  - **regression**
- **In general, an inference engine can run in two modes:**
  - **batch (all input variables for a query are given at once)**
  - **conversational (input variables are provided one after the other, this way, non-salient variables can be skipped) (dialog systems)**

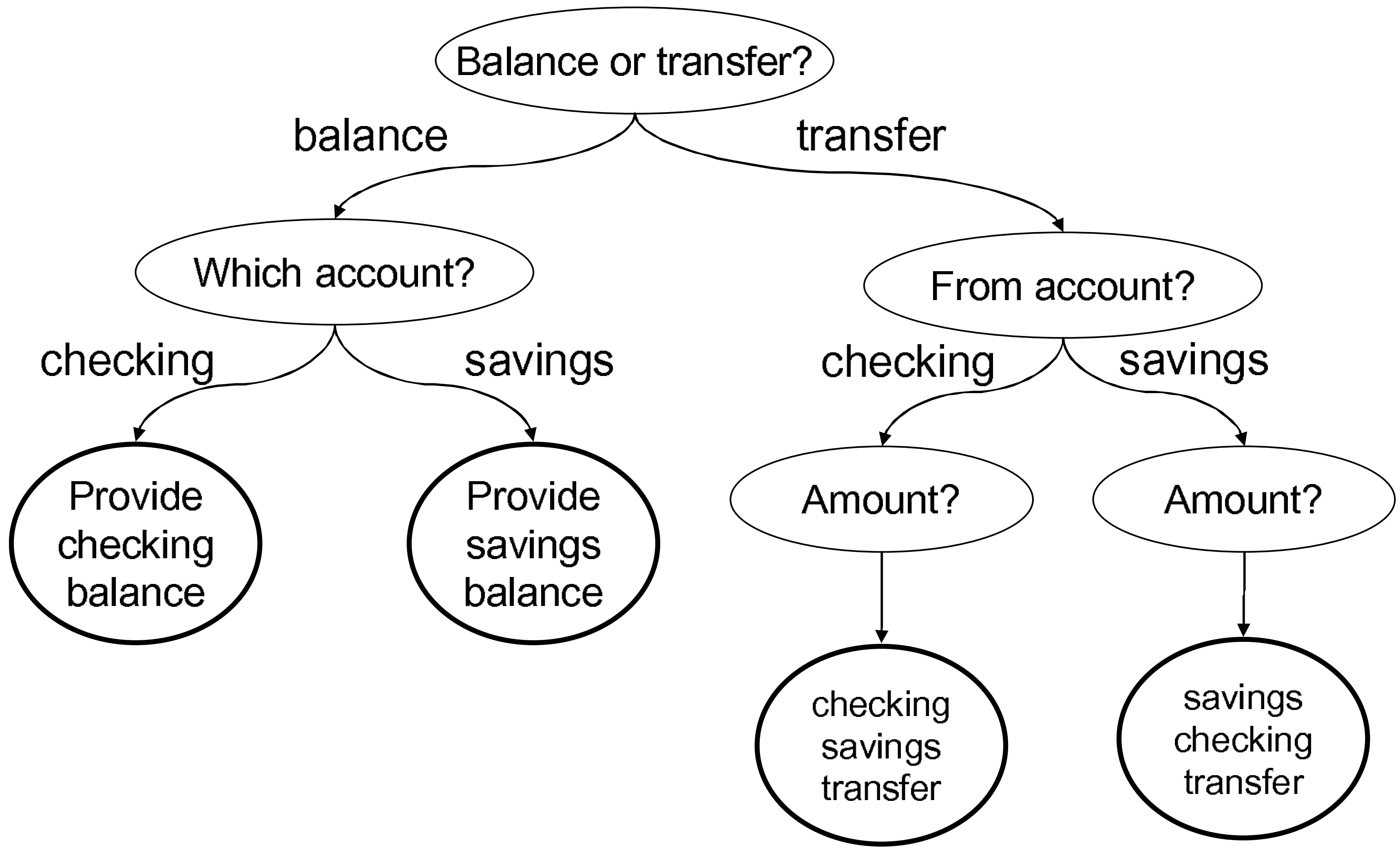
Spoken dialog systems: architecture



Sample call

(play sample call)

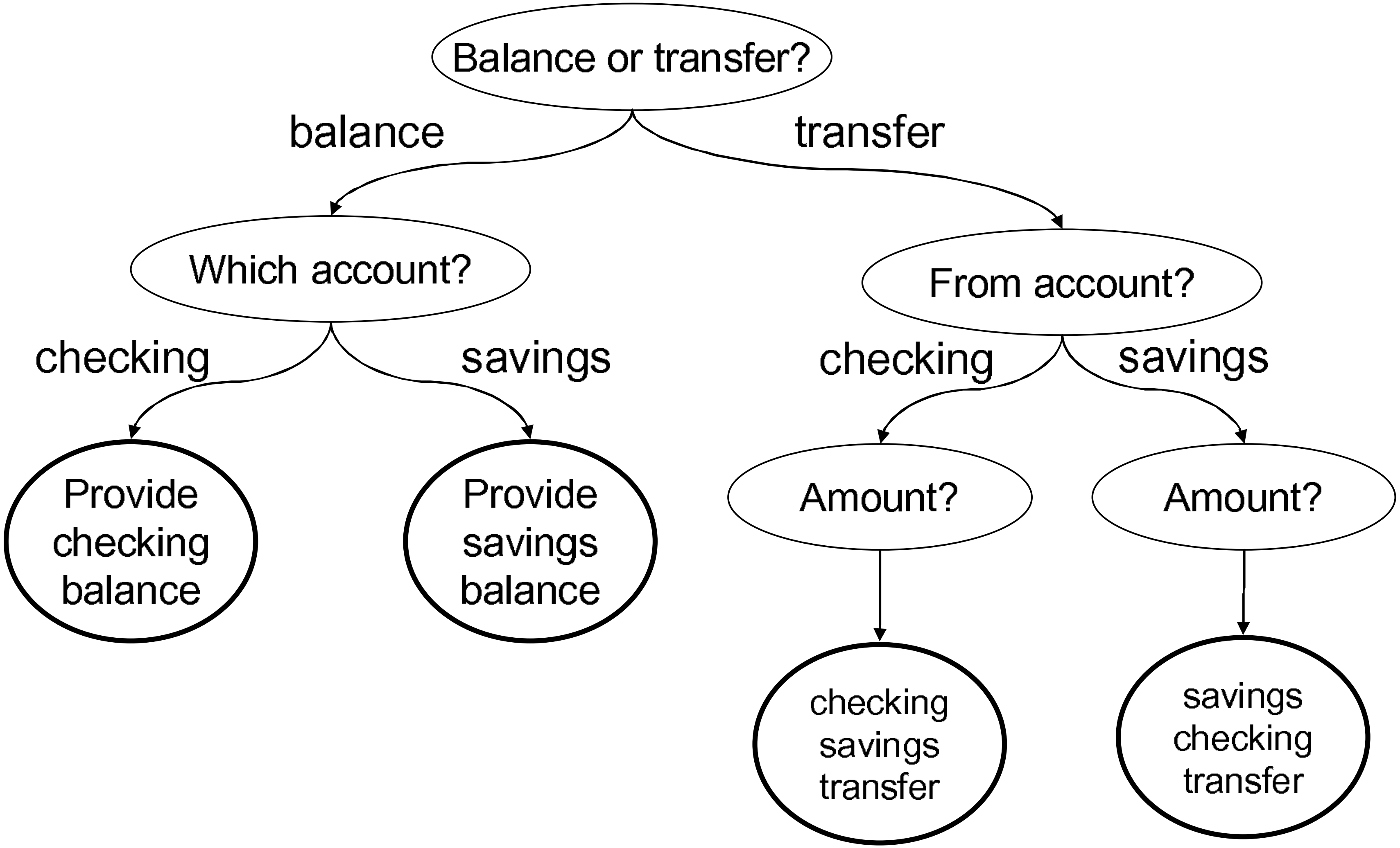
an example call flow...



Manual vs. automatic design

- **In commercial spoken dialog systems, call flows are built by call flow designers.**
- **They implement a predefined business logic.**
- **It may appear obvious how to implement this logic, i.e.,**
  - **which questions have to be asked,**
  - **which backend requests have to be performed,**
  - **in which sequence.**
- **However, there are strong arguments for automatic call flow generation:**
  - **manual generation is time-consuming,**
  - **manual generation is suboptimal and error-prone,**
  - **automatic generation can react on dynamically changing business logic or external factors such as the distribution of callers and call reasons.**

an example call flow...



Call flow and business logic (cont.)

...and the underlying business logic table:

service type?	account type?	amount?	destination
balance	checking		give checking balance
balance	savings		give savings balance
transfer	checking	$x$$	check.-sav. transfer $x$$
transfer	savings	$x$$	sav.-check. transfer $x$$

Call flow and business logic (cont.)

- **Real-world call flows can be very complex (1000s of nodes and transitions).**
- **Complex call flows can be broken down in sub-call flows each of which can be represented by individual business logic tables.**
- **Without loss of generality, we consider a (sub-)call flow to be of a question-answer-destination type.**
  ⟶ Columns of the business logic table represent questions and routing destination (or final call flow action).
  ⟶ Rows contain individual answers and destinations.
  ⟶ An additional column contains the probability with which every row is visited.

Call flow and business logic (cont.)

a general business logic table:

Q_1	Q_2	$\dots$	Q_M	D	P
A_1^1	A_2^1	$\dots$	A_M^1	D^1	P^1
A_1^2	A_2^2	$\dots$	A_M^2	D^2	P^2
$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$
A_1^N	A_2^N	$\dots$	A_M^N	D^N	P^N

- **Q_m is the mth question,**
- **A_m^n is an answer to Q_m,**
- **D^n is the nth routing destination,**
- **P^n is the prior probability of a call ending at D^n.**

Why does the order of questions matter?

- **In an example, we want to determine which of these modem types a caller has:**
  1. **black Ambit**
  2. **white Ambit**
  3. **black Arris**
- **The system designer considers the two questions**
  - **A** **Is your modem black or white?**
  - **B** **Do you have an Ambit or an Arris modem?**

- Since A=white $\rightarrow$ 2 and B=Arris $\rightarrow$ 3, the optimal order (A,B vs. B,A) depends on the priors $p(1), p(2), p(3)$ we can estimate from log data.
- Here, optimality means that the expected number of questions asked is minimal.
- One way to estimate this number is to use the definition of the expected value

$$E = \sum_i x_i p(x_i). \tag{27}$$

- Our random variable x is the number of asked questions.
- In our example, x has two possible values ($x_1 = 1$ and $x_2 = 2$; i.e., either we ask only one or both questions).

Why does the order of questions matter? (cont.)

- **In case of the order A,B, we ask one question with the probability $p(2)$ and two questions with $1 - p(2)$, i.e.,**

$$E(A, B) = 1 \cdot p(2) + 2 \cdot (1 - p(2)) = 2 - p(2). \tag{28}$$

- **Similarly, for the order B,A, we get**

$$E(B, A) = 1 \cdot p(3) + 2 \cdot (1 - p(3)) = 2 - p(3). \tag{29}$$

- **Depending on the specific values of $p(2)$ and $p(3)$ either $E(A, B)$ or $E(B, A)$ is minimal, thereby determining the optimal order.**
- **As an example, assume**

$$p(1) = 0.3; \quad p(2) = 0.4; \quad p(3) = 0.3. \tag{30}$$

- **This results in**

$$E(A, B) = 1.6; \quad E(B, A) = 1.7. \tag{31}$$

- **Consequently, the optimal order in this example is A,B.**

- **A call flow resembles a decision tree.**
  $\longrightarrow$ We can use well-established machine learning techniques.

- **To find the most relevant questions, i.e. the ones providing maximum information, let's use the information gain measure:**

$$I(D; A_m) = H(A_m) + H(D) - H(A_m, D) \ . \tag{32}$$

H **is Shannon's entropy defined as, e.g.**

$$H(D) = -\sum_{\delta=1}^{\Delta} P(\delta) \log_2 P(\delta) \tag{33}$$

where $\delta \in \{1, \ldots, \Delta\}$ are the distinct destinations in the currently processed business logic table.

- **At every node in the call flow, we determine which question leads to the maximum information gain:**

$$Q_{\hat{m}} \quad \text{with} \quad \hat{m} = \arg\max_{m=1,\ldots,M} I(D; A_m) \ . \tag{34}$$

- We took the business logic table from a mature call routing application processing about 4M calls per month.
- Based on call logs of an entire month, the probabilities P^n were estimated.
- Experiment parameters:

number of calls	$3,868,014$
number of questions	$M = 4$
number of rows	$N = 31$
number of distinct destinations	$\Delta = 20$

- **The original app asked $M = 4$ questions:**
  - **service type (orders, billing, technical support, etc.)**
  - **product (Internet, cable TV, telephone)**
  - **actions (cancel, schedule, make a payment, etc.)**
  - **modifiers (credit card, pay-per-view, digital TV conversion, etc.)**

- **Automatic call flow generation with maximum information gain strategy resulted in $\hat{M} = 2.87$.**
  $\longrightarrow$ **30% reduction of average number of asked questions**
  $\longrightarrow$ **possible savings of five- to six-figure US$ per month**

A reward function

- The main argument for using commercial spoken dialog systems is to replace the human agent to save costs.
- Can we quantify the savings?
- And if so, what can we do to optimize them?

Time is money: From S to R

- Principally, an application's performance is determined by the fraction of calls completed without agent intervention (the automation rate A).
- Consider an average cost W_A associated with a call successfully handled by a human agent.
- On the other hand, automated calls produce costs (hosting, licensing or telephony fees) that depend on the call duration T.
- The per-time-unit cost is W_T.
- Consequently, the overall savings/reward is

$$S = W_A A - W_T T \ [\$] \quad \longrightarrow \quad R = T_A A - T \ [s] \qquad (35)$$

with the trade-off param $T_A = \frac{W_A}{W_T}$.

- **When automation rate falls below $\frac{T}{T_A}$, savings turn negative (!)**

- **To avoid this situation, there are several techniques to adapt and optimize spoken dialog systems.**

- **This is to continually increase performance over time (or keep it at a high saturation point).**

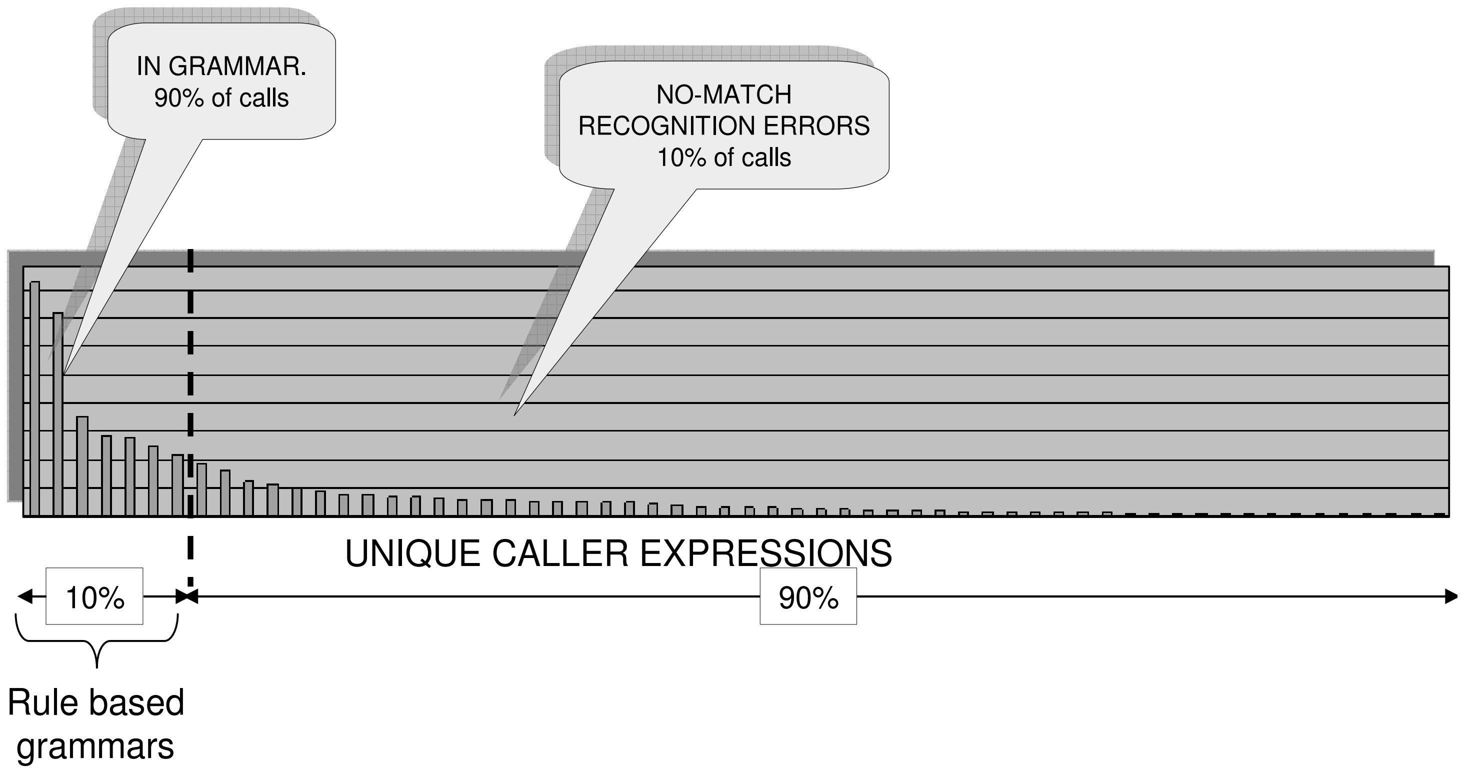
- **This talk is about two components of a spoken dialog system subject to adaptation and optimization:**
  - **speech recognition and understanding**
  - **dialog management**

- **The major criticism on spoken dialog systems is their tendency to misunderstand human speech.**
- **Speech recognition and understanding problems cause**
  - **escalations to a human upon reaching a max number of “speech errors”,**
  - **going down the wrong path leading to a dead end,**
  - **poor user experience.**

- **Directed dialog: the prompt suggests what the caller should say**
  - **Do you have more than one TV at home?**
  - **What brand of modem do you have?**
- **Open prompt: callers are invited to use their own expressions**
  - **Please tell me the reason you are calling about today.**
- **Commercial applications use rule-based grammars for directed dialog:**
  - **data for statistical grammars initially unavailable**
  - **lack of knowledge and tools to build statistical grammars**
  - **practitioners and developers feel statistical grammars are out of their control**

The curse of the unexpected

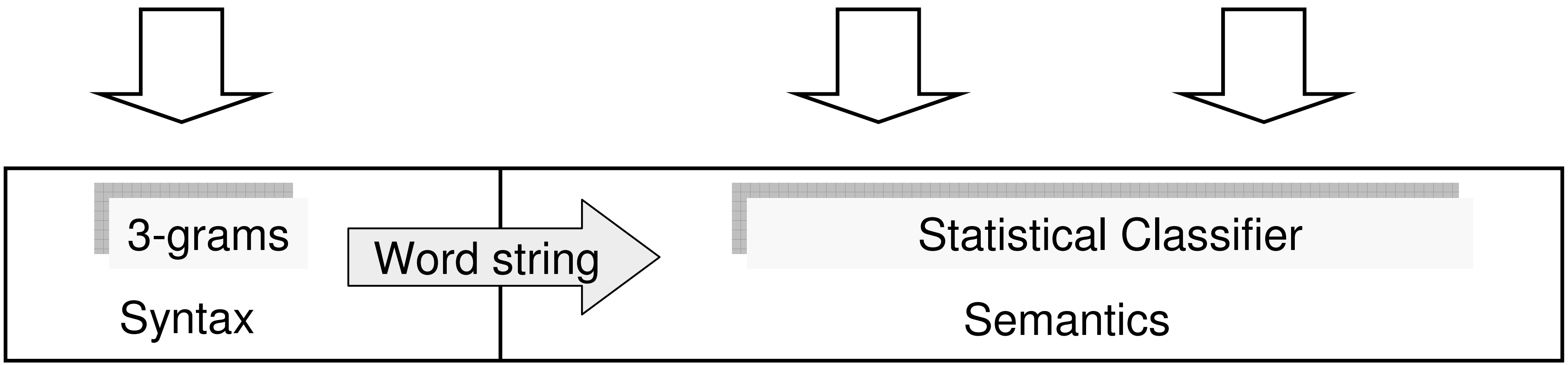
- **Grammars are designed to match prompts**
  - **Do you have more than one TV at home? (***yes* | *yeah* | *yup* | *no* | *nope***)**
  - **What brand of modem do you have? (***Motorola* | *Toshiba* | *Sony* | *...***)**
- **Unfortunately, a significant portion of users speak out-of-grammar**
  - **Do you have more than one TV at home?** *I have three TVs.*
  - **What brand of modem do you have?** *Can't read the brand, it is blue.*

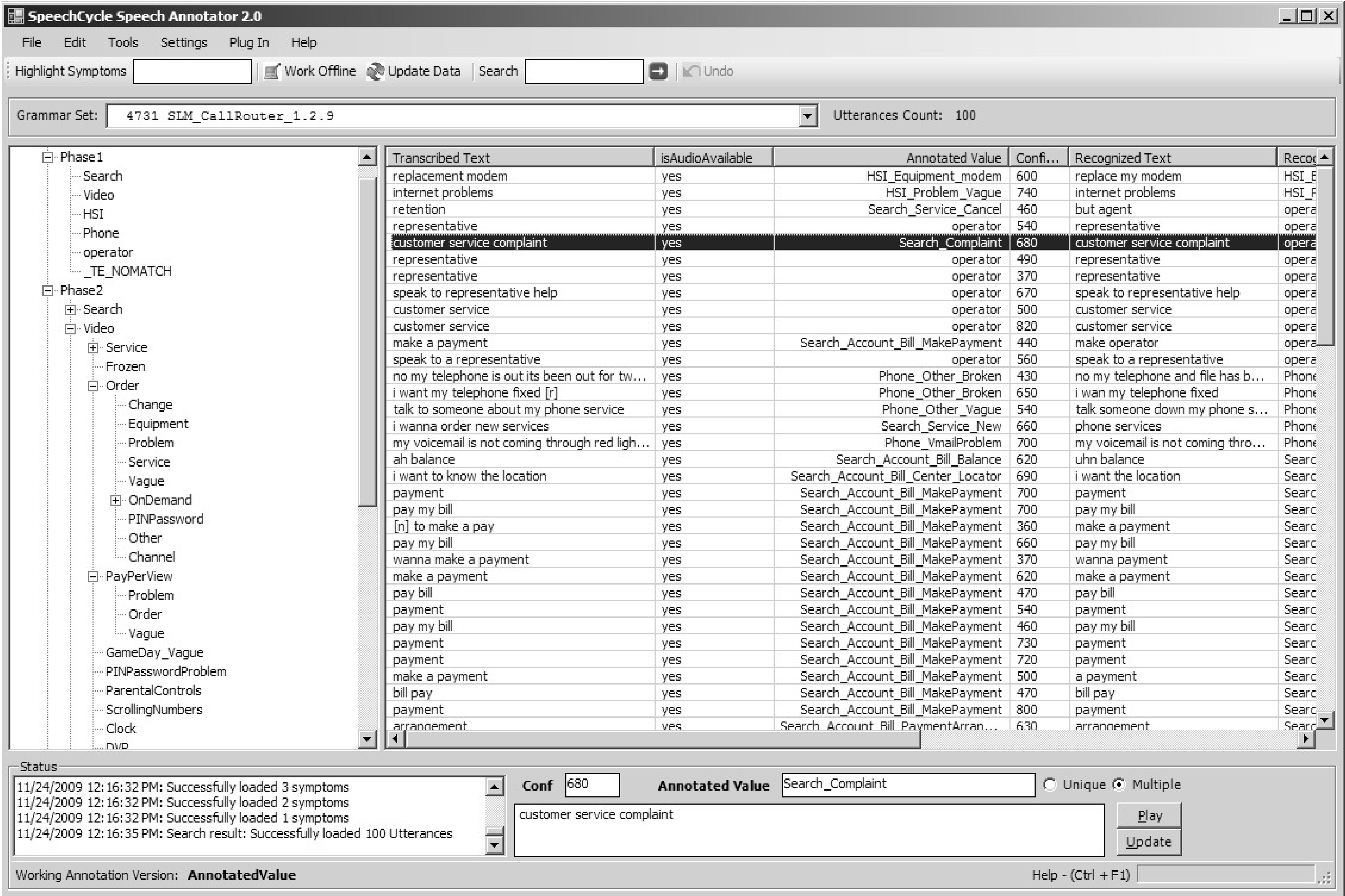


Grammar tuning

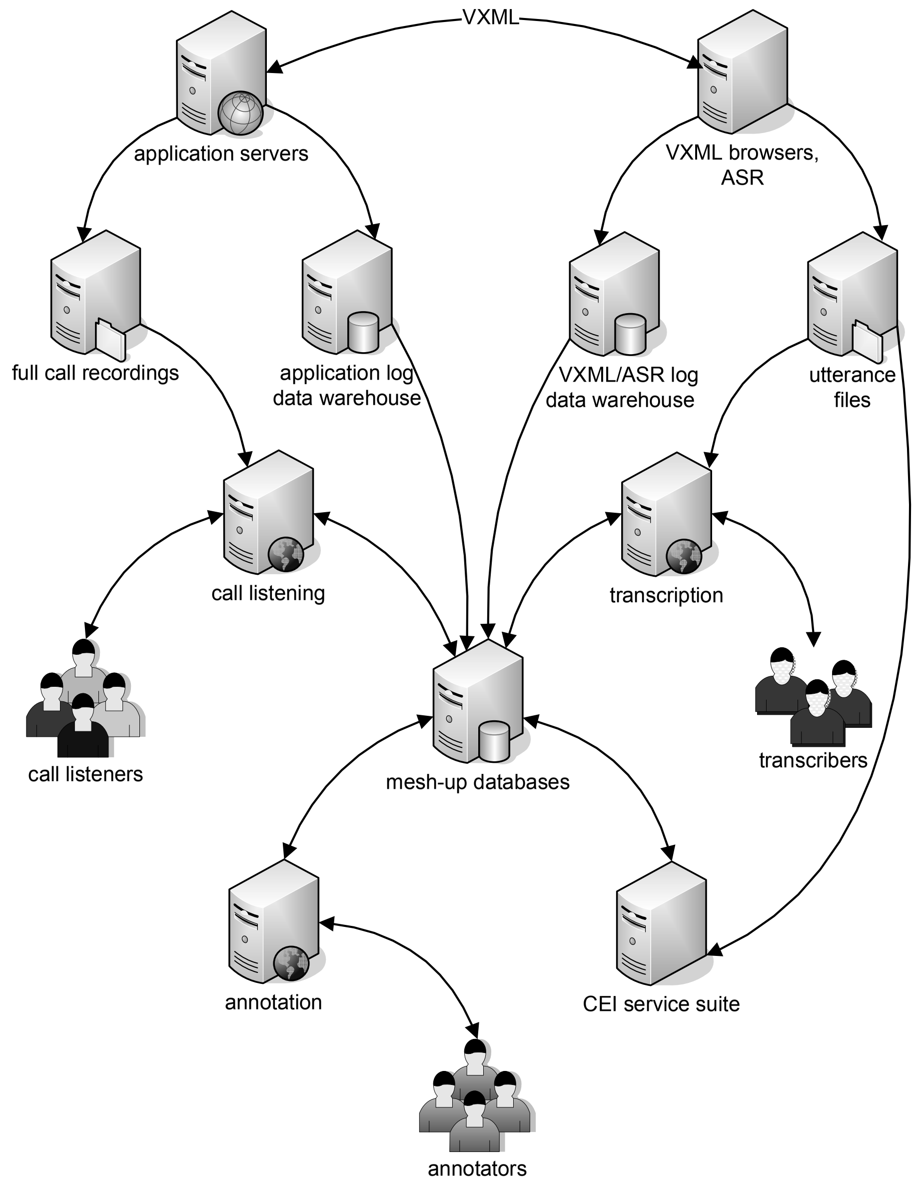
- Conventionally, commercial tuning is performed (sporadically!) by
  - "speech scientists" looking at small samples of transcribed calls
  - changing, adding, removing rules to match observed data
- Manual tuning is expensive and does not scale
  - cannot take advantage of large amounts of data
  - cannot systematically tune all grammars in large applications (1000s of nodes)
- Manually tuned grammars cannot ever perform as well as statistical ones (certain conditions apply)

TRANSCRIPTIONS	ANNOTATIONS
want to cancel the account	**CANCEL_ACCOUNT**
cancel service	**CANCEL_ACCOUNT**
I cant send a particular message to a certain group of people	**CANNOT_SEND_RECEIVE_EMAIL**
cancellation of the service	**CANCEL_ACCOUNT**
I need to setup my email	**EMAIL_SETUP**
they registered my modem in from my internet and I need to get my email address	**EMAIL_SETUP**
my emails are not been received at the address I sent it to	**CANNOT_SEND_RECEIVE_EMAIL**
…	

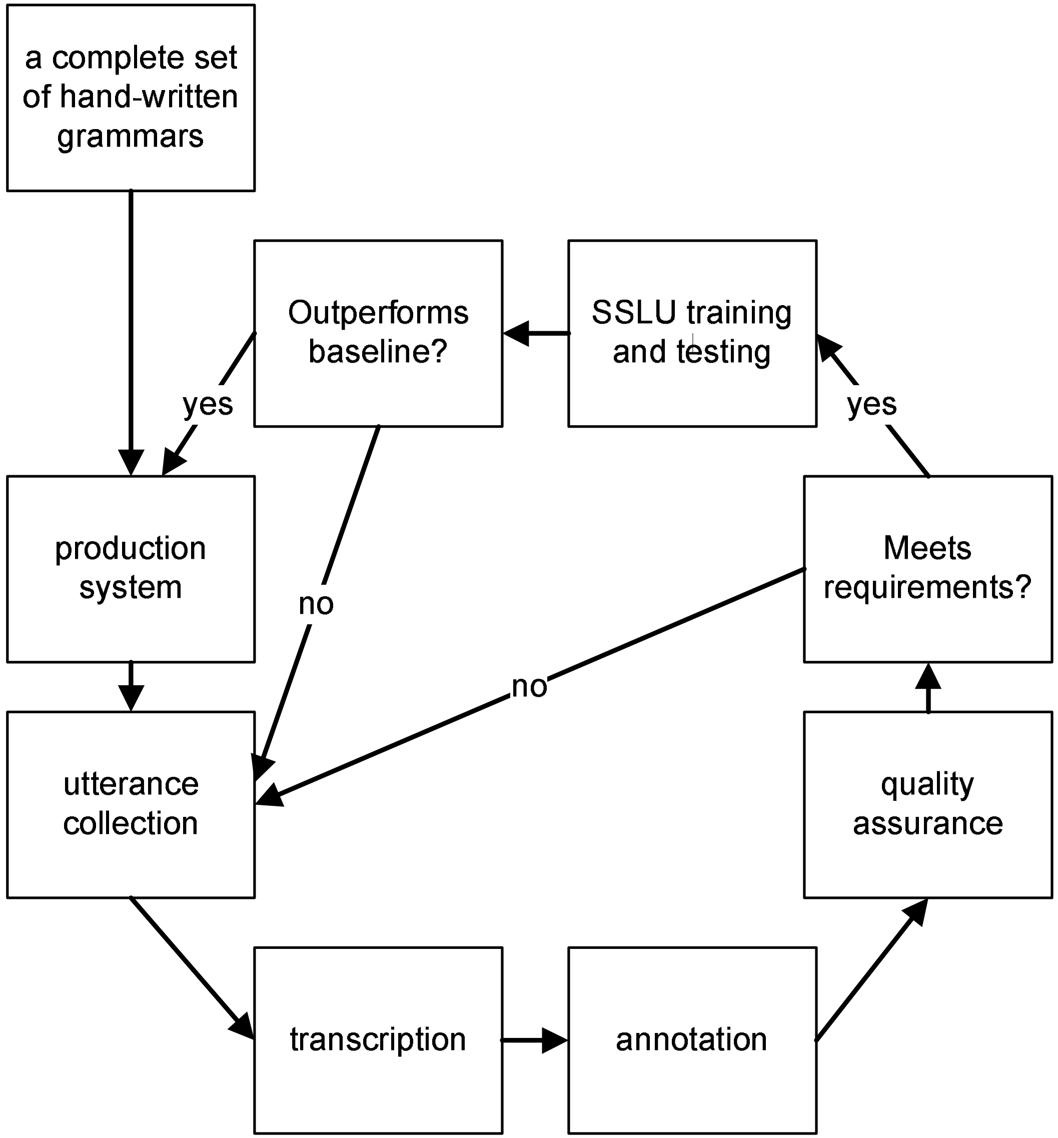



SpeechCycle Speech Annotator 2.0
File Edit Tools Settings Plug In Help
Highlight Symptoms
Work Offline
Update Data
Search
Undo
Grammar Set: 4731 SLM_CallRouter_1.2.9
Utterances Count: 100
Phase1
Search
Video
HSI
Phone
operator
_TE_NOMATCH
Phase2
Search
Video
Service
Frozen
Order
Change
Equipment
Problem
Service
Vague
OnDemand
PINPassword
Other
Channel
PayPerView
Problem
Order
Vague
GameDay_Vague
PINPasswordProblem
ParentalControls
ScrollingNumbers
Clock
Transcribed Text
isAudioAvailable
Annotated Value
Confi...
Recognized Text
replacement modem yes HSI_Equipment_modem 600 replace my modem
internet problems yes HSI_Problem_Vague 740 internet problems
retention yes Search_Service_Cancel 460 but agent
representative yes operator 540 representative
customer service complaint yes Search_Complaint 680 customer service complaint
representative yes operator 490 representative
representative yes operator 370 representative
speak to representative help yes operator 670 speak to representative help
customer service yes operator 500 customer service
customer service yes operator 820 customer service
make a payment yes Search_Account_Bill_MakePayment 440 make operator
speak to a representative yes operator 560 speak to a representative
no my telephone is out its been out for tw... yes Phone_Other_Broken 430 no my telephone and file has b...
i want my telephone fixed [r] yes Phone_Other_Broken 650 i wan my telephone fixed
talk to someone about my phone service yes Phone_Other_Vague 540 talk someone down my phone s...
i wanna order new services yes Search_Service_New 660 phone services
my voicemail is not coming through red ligh... yes Phone_VmailProblem 700 my voicemail is not coming thro...
ah balance yes Search_Account_Bill_Balance 620 uhn balance
i want to know the location yes Search_Account_Bill_Center_Locator 690 i want the location
payment yes Search_Account_Bill_MakePayment 700 payment
pay my bill yes Search_Account_Bill_MakePayment 700 pay my bill
[n] to make a pay yes Search_Account_Bill_MakePayment 360 make a payment
pay my bill yes Search_Account_Bill_MakePayment 660 pay my bill
wanna make a payment yes Search_Account_Bill_MakePayment 370 wanna payment
make a payment yes Search_Account_Bill_MakePayment 620 make a payment
pay bill yes Search_Account_Bill_MakePayment 470 pay bill
payment yes Search_Account_Bill_MakePayment 540 payment
pay my bill yes Search_Account_Bill_MakePayment 460 pay my bill
payment yes Search_Account_Bill_MakePayment 730 payment
payment yes Search_Account_Bill_MakePayment 720 payment
make a payment yes Search_Account_Bill_MakePayment 500 a payment
bill pay yes Search_Account_Bill_MakePayment 470 bill pay
payment yes Search_Account_Bill_MakePayment 800 payment
Status
11/24/2009 12:16:32 PM: Successfully loaded 3 symptoms
11/24/2009 12:16:32 PM: Successfully loaded 2 symptoms
11/24/2009 12:16:32 PM: Successfully loaded 1 symptoms
11/24/2009 12:16:35 PM: Search result: Successfully loaded 100 Utterances
Conf 680
Annotated Value Search_Complaint
Unique Multiple
customer service complaint
Play
Update
Working Annotation Version: AnnotatedValue
Help - (Ctrl + F1)

Infrastructure



Continuous improvement cycle



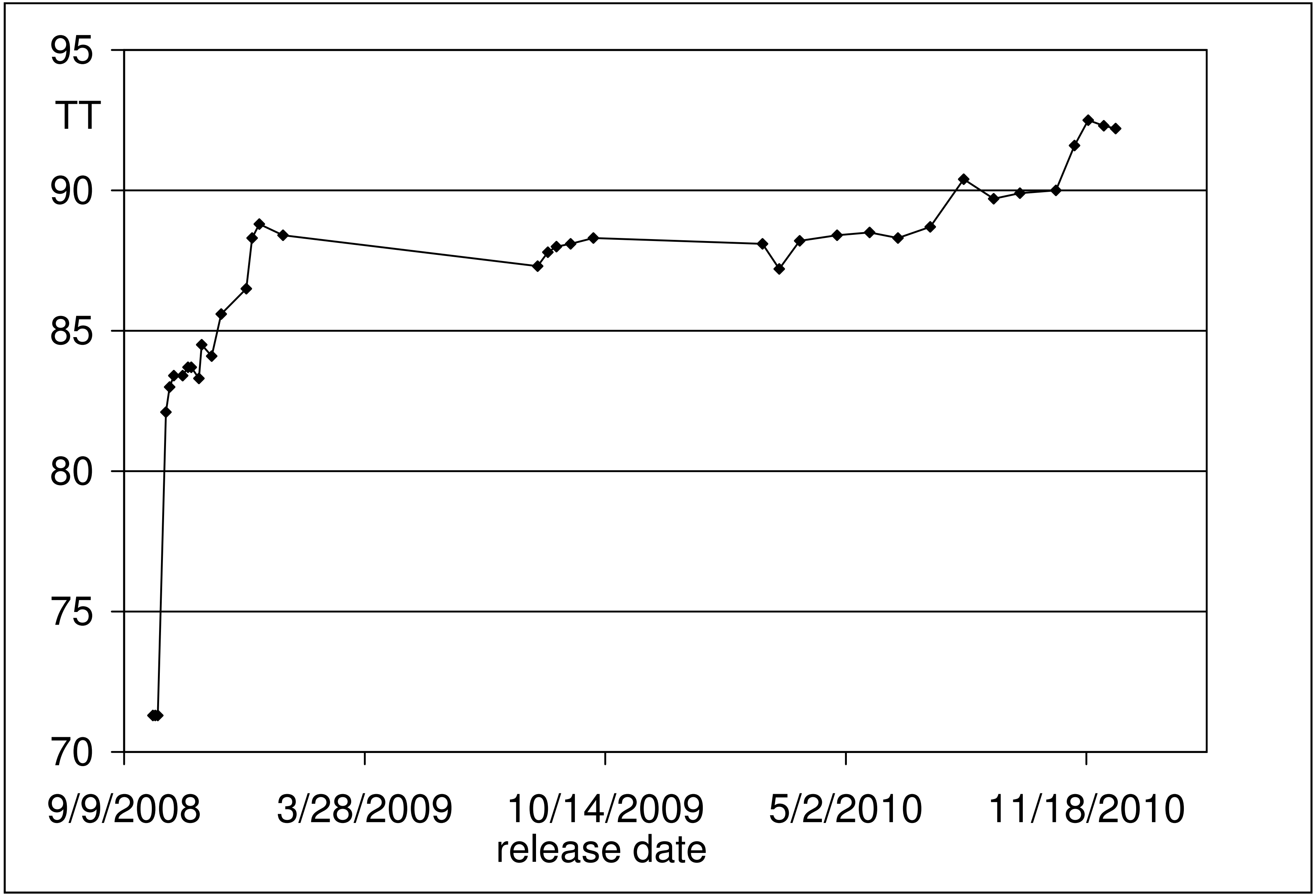
C^7 data conditioning process

- **Completeness**
  - **Transcribed/annotated data to match the distribution**
- **Correlation**
  - **Correlation analysis guarantee that utterances are annotated consistently across multiple annotators**
- **Consistency**
  - **Similar utterances need to be annotated consistently.**
- **Confusion**
  - **Reduce confusion across classifier categories**

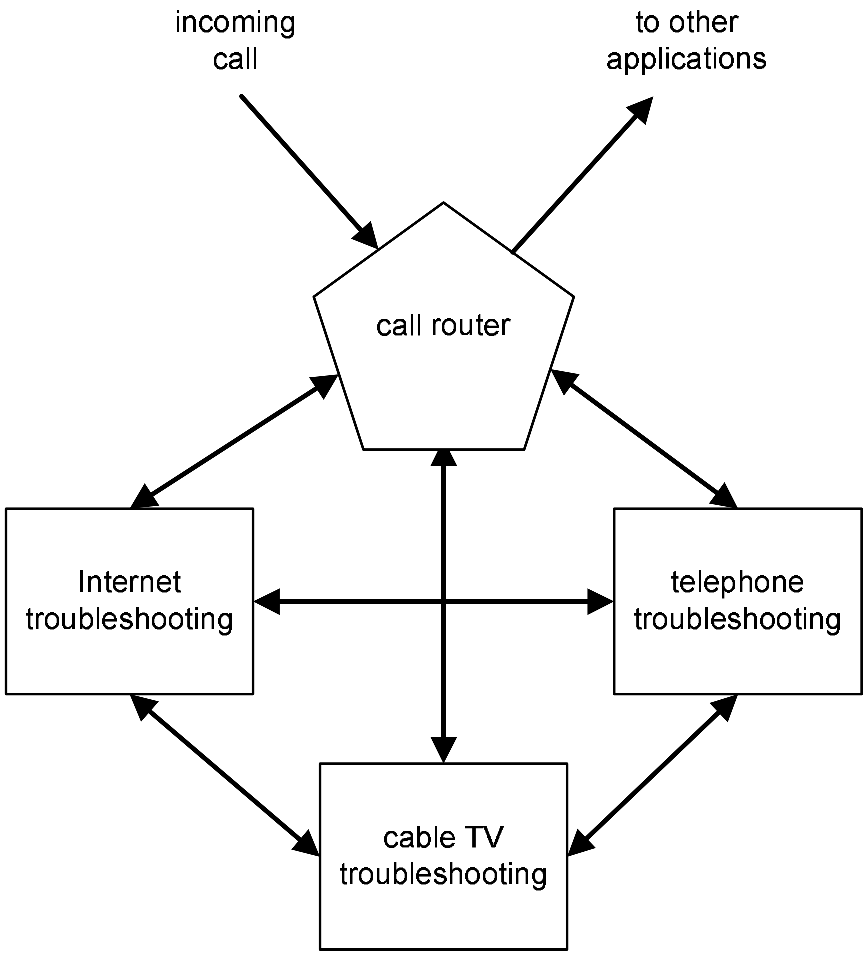
C^7 data conditioning process (cont.)

- **Congruence**
  - Rule based grammars, if available, need to be congruent with annotation
- **Coverage**
  - minimize out-of-grammar utterance by adding new semantic symptoms
- **Corpus size**
  - minimum training, development, and test corpus sizes are required to ensure statistical significance

Continuous improvement of semantic accuracy



Continuous improvement of semantic accuracy (cont.)



#utterances	**2,184,203**
#calls	**533,343**
#nodes	**2,021**
True Total (Jun. 2008)	**78.0%**
True Total (Sep. 2008)	**90.5%**

- **Equation 35 ($R = T_A A - T$) suggests that R can by increased by decreasing call duration T.**
- **This holds true for automated and non-automated calls.**
- **However, non-automated calls can be shortened aggressively by escalating to an agent as early as possible.**
- **We call an algorithm that deliberately escalates calls based on its opinion about the call outcome Escalator.**
- **Escalators can be based on**

  (1) manual rules (unsolicited agent requests, speech recognition problems, situations the system does not know how to handle, etc.),

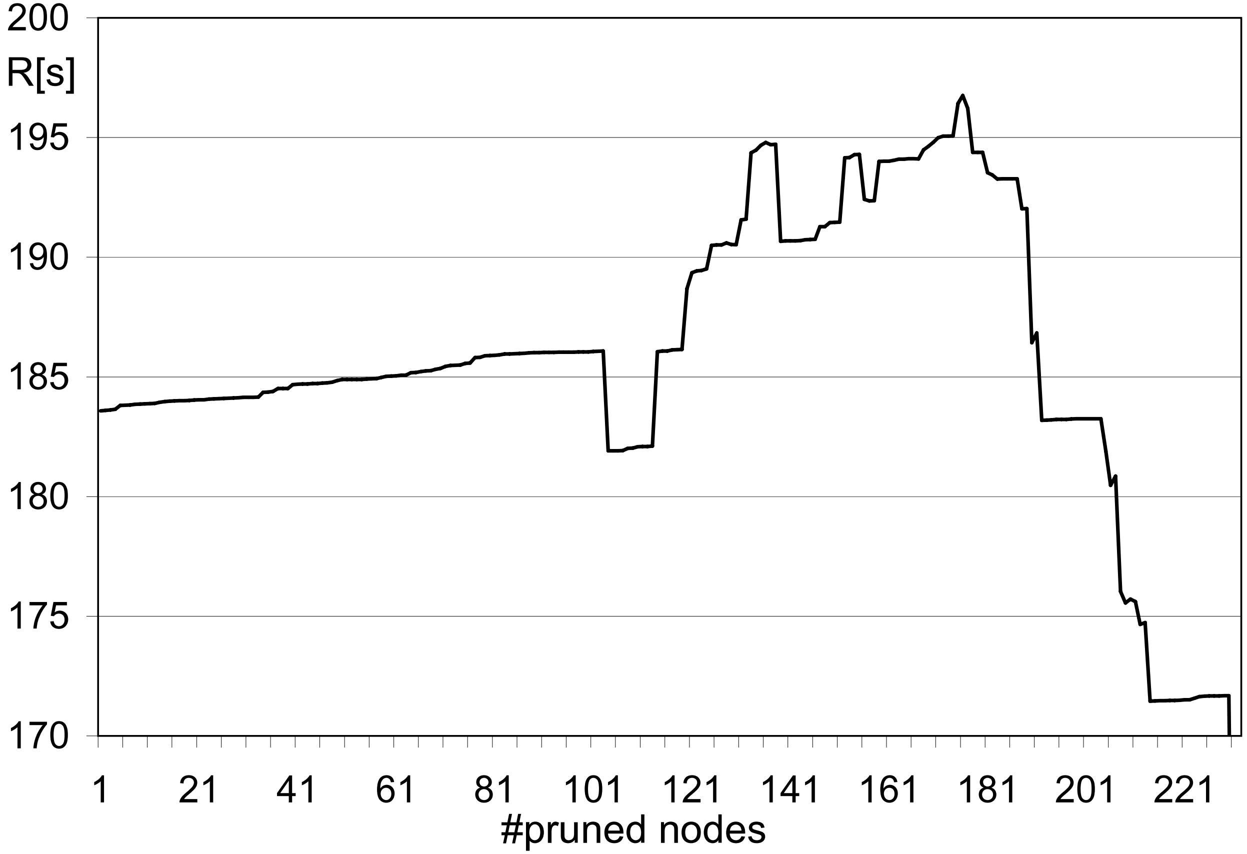
  (2) the probability of the call ending unsuccessfully,

  (3) ...

Escalator (cont.)

- **Features used to estimate Escalator probabilities can be based on the dialog history including**
  - **transitions taken**
  - **textual and acoustic speech input**
  - **acoustic and semantic confidence scores**
  - **number of no-match, no-input, or dis-confirmation**
  - **etc.**
- **An example implementation is described in [Levin and Pieraccini, 2006].**
- **Another example is based on pruning the call flow:**
  - **compute the average reward per node by exploiting log data,**
  - **establish a ranking list**
  - **compute the app's overall reward incrementally eliminating nodes.**

Escalator Example

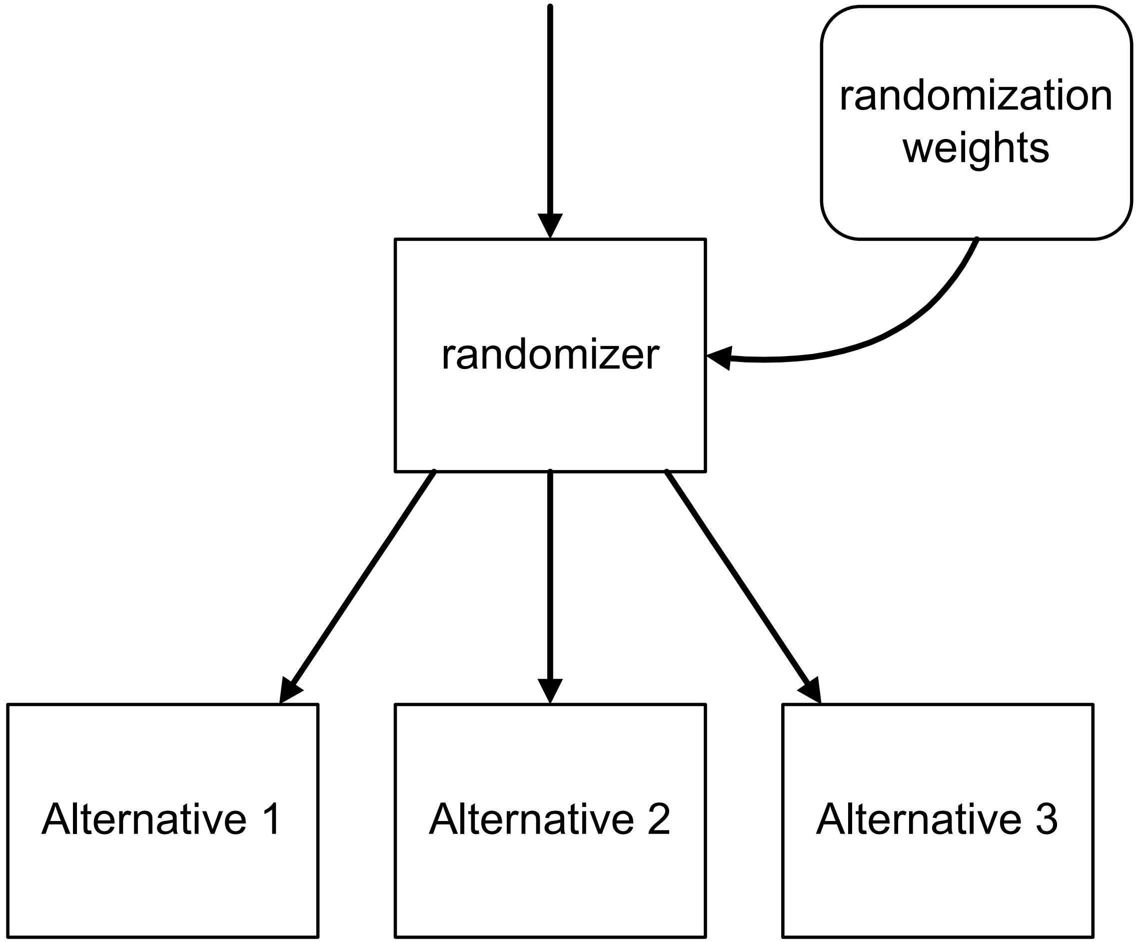


#calls (tokens)	**45,631**
#nodes (types)	**847**
#nodes pruned	**176**
T_A	**5,000s**
R **w/o pruning**	**183.5s**
R **w/ pruning**	**196.8s**
ΔR	**13.3s**

Contender

- **Escalator focuses on reducing handling time.**
- **What can we do to boost automation rate?**
- **There can be 1000 things impacting automation, e.g.**
  - **Is directed dialog or open prompt better in this context?**
  - **Or a y/n question followed by an open prompt with examples?**
  - **How much time should I wait until I offer a backup menu?**
  - **What is the ideal voice activity detection sensitivity?**
  - **When do I time out?**
  - **What is the best recovering strategy after a no-match?**
  - **...**
- **To find out which strategy is best, we can implement all of the above.**
- **Then we route certain portions of traffic to each of the Contender paths.**

Contender (cont.)



Contender: example

- As shown in [Suendermann et al., 2010], the amount of traffic hitting path A should be the winning probability $p(A)$.
- This approach maximizes the accumulated reward.
- In case of a 2-way split, $p(A)$ can be estimated based on statistical tests such as t or z tests (p value).
- In case of an n-way split, the numerical solution of an n-dimensional integral over Contender probability distributions is required.

#calls (tokens)	38,004
T_A	5,000s
R baseline	253.4s
R after contending	282.9s
ΔR	29.4s

Outline

- **logic and computer-assisted proof**
- **intelligent search and problem solving strategies**
- **expert systems and dialog systems**
- **Prolog**

- **Prolog (programming in logic) is programming language associated with artificial intelligence as well as computer linguistics.**
- **In accordance with the architecture of XPSs, the main components of logical programming are**
  1. **a knowledge base (facts and rules),**
  2. **an inference engine.**
- **Advantage of logical programming is that one does not have to develop an algorithm to solve the problem since this job is done by the inference engine.**
- **Instead, we describe the problem by means of logical formulas.**
- **The open-source SWI-Prolog is available as part of the major Linux distributions as well as Cygwin (`http://cygwin.com`) or can be obtained from**

  `http://www.swi-prolog.org`

- **Facts are atomic formulas with the Prolog syntax**

$$p(t_1, \ldots, t_n) \tag{36}$$

featuring the predicate p and the terms $t_1, \ldots, t_n$.

- **All the variables in facts are universally bound, i.e., Eq. 36 represents the logical formula**

$$\forall x_1, \ldots, x_m(p(t_1, \ldots, t_n)). \tag{37}$$

- **Rules are conditional propositions with the Prolog syntax**

$$A : -B_1, \ldots, B_n. \tag{38}$$

featuring the atomic formulas $A, B_1, \ldots, B_n$.

- **Again, all the variables in rules are universally bound, so, Eq. 38 represents the formula**

$$\forall x_1, \ldots, x_m(B_1 \wedge \ldots \wedge B_n \rightarrow A). \tag{39}$$

- **This generally requires formulas to be given as Horn clauses.**

Some conventions

- The first character of **variables** is a capital letter or an underscore.
- The first character of **predicates** or **functions** is a lower-case letter.
- The predicate `true` represents validity.
- The symbols `+`, `-`, `*`, `/`, `.` are function symbols you can use in **infix** notation.
- The symbols <, >, =, =<, >=, \=, ==, \== are predicate symbols you can use in **infix** notation. Note that

  == tests for equality,

  \== tests for inequality, and

  = is the **unification** operator.
- The symbol \+ (or, alternatively, `not()`) is the **negation** operator.
- The symbol % is used for comments.
- The symbols `,` and `;` is used for conjunction and disjunction, respectively.

- **The following derivation shows that disjunctions in Prolog rules are effectively no additional feature:**

$$\begin{aligned} A :- B_1; \ldots ; B_n. \quad &\Leftrightarrow \quad B_1 \vee \ldots \vee B_n \rightarrow A. \\ &\Leftrightarrow \quad \neg(B_1 \vee \ldots \vee B_n) \vee A \\ &\Leftrightarrow \quad \neg B_1 \wedge \ldots \wedge \neg B_n \vee A \\ &\Leftrightarrow \quad (\neg B_1 \vee A) \wedge \ldots \wedge (\neg B_n \vee A) \\ &\Leftrightarrow \quad A :- B_1. \\ &\qquad \ldots \\ &\qquad A :- B_n. \end{aligned} \tag{40}$$

An example

- **Let us now consider a realistic example:**
  - **All students are smart.**
  - **Whoever is smart is powerful.**
  - **Whoever is computer scientist and professor is powerful.**
  - **Computer scientists are crazy.**
  - **Alan is a student.**
  - **Brad is a student.**
  - **Colin is a computer scientist.**
  - **Colin is a professor.**

- **This is the respective Prolog code (see** `student.pl` **in the auxiliary package** `kbs_*.zip`**):**

```
smart(X):-student(X).
powerful(X):-smart(X).
powerful(X):-cs(X),prof(X).
crazy(X):-cs(X).
student(alan).
student(brad).
cs(colin).
prof(colin).
```

An example (cont.)

- **We want to find out whether**

  there is a powerful and crazy individual.

- **The respective logical formula is**

$$\exists x(\texttt{powerful}(x) \wedge \texttt{crazy}(x)). \tag{41}$$

- **In order to find out, we first launch Prolog with the command**

```
pl
```

  and get the command prompt

```
?-
```

- **To load our knowledge base, we type**

```
consult(student).
```

- **Now, we can use the Prolog syntax of Eq. 41 to check the validity of our conjecture:**

```
powerful(X),crazy(X).
```

An example (cont.)

- **We obtain the response**

  ```
  X = colin
  ```

  telling us that Colin is a powerful and crazy individual.

- **In order to identify other potential candidates, we type**

  ```
  ;
  ```

  resulting in the response

  ```
  No
  ```

  which indicates that there are no more solutions to the problem.

- **We are given the Prolog program P consisting of a number of rules of the form**

$$R := A : -B_1, \ldots, B_m \tag{42}$$

and a query of the form

$$G = Q_1, \ldots, Q_n. \tag{43}$$

- **Here, facts are expanded to rules by**

$$A \leftrightarrow A : -\texttt{true}. \tag{44}$$

- **The inference algorithm works as follows:**

  1. **Search (in order of appearance) all the rules A in P, for which there exists a unifier**

$$\mu = \begin{cases} [] & \text{if } Q_1 = \texttt{true} \\ \text{mgu}(Q_1, A) & \text{otherwise} \end{cases} \tag{45}$$

2. In case there are multiple such rules,
   a) select the first rule (in order of appearance),
   b) set a choice point (CP) to perform a different selection at this point in case it becomes necessary at a later moment.
3. Here, two cases are distinguished:
   a) $m + n = 1$: This means success, and Prolog returns the last non-empty μ.
   b) Otherwise, we recursively continue with the query
   $$G := B_1\mu, \ldots, B_m\mu, Q_2\mu, \ldots, Q_n\mu. \tag{46}$$
   If we do not find a solution, we return to the last choice point reversing the replacements $G := G\mu$ accordingly.

- Negation is implemented in Prolog as negation as failure.
- I.e., if Q_1 in 1 is of the syntax $\mathtt{not}(Q'_1)$ the algorithm tries to prove Q'_1.
- If it succeeds, we know that Q_1 is false, otherwise, we assume it is true.

- **Let us sketch a proof of Prolog's inference rule.**
- **For the sake of simplicity, we limit ourselves to propositional logic and assume a Prolog rule**

$$A :- B. \tag{47}$$

and a query

$$Q, R. \tag{48}$$

- **Prolog's inference rule as used in the above algorithm is hence**

$$\begin{array}{l} A :- B. \\ A \leftrightarrow Q \\ B, R. \\ \hline \therefore Q, R. \end{array}$$

- **Let us prove that this inference rule is a tautology:**

$$
\begin{aligned}
V &:= (B \rightarrow A) \land (A \leftrightarrow Q) \land (B \land R) \rightarrow (Q \land R) \\
&\Leftrightarrow (\neg B \lor A) \land (\neg A \lor Q) \land (A \lor \neg Q) \land B \land R \rightarrow Q \land R \\
&\Leftrightarrow \neg(\neg B \lor A) \lor \neg(\neg A \lor Q) \lor \neg(A \lor \neg Q) \lor \neg B \lor \neg R \lor Q \land R \\
&\Leftrightarrow \underbrace{\overbrace{B \land \neg A \lor \underbrace{A \land \neg Q \lor \neg A \land Q}_{S}}^{T} \lor \neg B \lor \neg R}_{U} \lor Q \land R \\
S &\Leftrightarrow (A \lor Q) \land (\neg Q \lor \neg A) \\
T &\Leftrightarrow (B \lor A \lor Q) \land (B \lor \neg Q \lor \neg A) \land (\neg A \lor \neg Q) \\
U &\Leftrightarrow \neg A \lor \neg Q \lor \neg B \lor \neg R \\
V &\Leftrightarrow \top
\end{aligned}
\tag{49}
$$

Prolog's inference algorithm: example

ID	CP	G	R	μ
1	**1**	powerful(X), crazy(X)	powerful(X) :− smart(X)	[]
2	**1**	smart(X), crazy(X)	smart(X) :− student(X)	[]
3	**3**	student(X), crazy(X)	student(alan) :− true	[$X \mapsto$ alan]
4	**3**	true, crazy(alan)		[]
5	**3**	crazy(alan)	crazy(X) :− cs(X)	[$X \mapsto$ alan]
6	**3**	cs(alan)	cs(colin) :− true	Ω
7	**1**	student(X), crazy(X)	student(brad) :− true	[$X \mapsto$ brad]
8	**3**	true, crazy(brad)		[]
9	**1**	crazy(brad)	crazy(X) :− cs(X)	[$X \mapsto$ brad]
10	**1**	cs(brad)	cs(colin) :− true	Ω

Prolog's inference algorithm: example (cont.)

ID	CP	G	R	μ
11		powerful(X), crazy(X)	powerful(X) :- cs(X), prof(X)	[]
12		cs(X), prof(X), crazy(X)	cs(colin) :- true	[$X \mapsto$ colin]
13		true, prof(colin), crazy(colin)		[]
14		prof(colin), crazy(colin)	prof(colin) :- true	[]
15		true, crazy(colin)		[]
16		crazy(colin)	crazy(X) :- cs(X)	[$X \mapsto$ colin]
17		cs(colin)	cs(colin) :- true	[]
18		true		[]

Prolog's response is hence: [$X \mapsto$ colin].

Drawbacks of Prolog's inference algorithm

- **Consider the Prolog program:**

```
1  a:-not(true).
```

- **Let us query whether** `a`**:**

ID	CP	G	R	μ
1		a	a : −not(true)	[]
2∗		true		[]

- **The query infers false by negation as failure (indicated by ∗ which means that a result derived from this step needs to be inverted).**
- **This, however, does not coincide with our understanding of the semantics of the implication:** $\bot \rightarrow$ a **is true independent of whether** a **or not.**
- **The reason is Prolog's closed-world assumption: It assumed the database is complete; I.e., if the answer cannot be deduced, it is false.**
- **Even worse, the response to the query** not(a) **is true due to two applications of inversion.**

Drawbacks of Prolog's inference algorithm (cont.)

- **Consider the Prolog program:**

```
a:-b.
b:-a.
```

- **Let us query whether** a, b**:**

ID	CP	G	R	μ
1		a, b	a : −b	[]
2		b, b	b : −a	[]
3		a, b	a : −b	[]
4		b, b	b : −a	[]

...

- **The program enters an infinite loop even though the query could be proven true in a few steps:**

$$(b \to a) \land (a \to b) \to a \land b \Leftrightarrow \top. \quad (50)$$

- **The nature of Prolog being based on Horn logic and its negation and loop handling show a considerable weakness of its inference algorithm.**

- **Apart from Prolog's inference engine, a predominant feature is its list handling.**
- **Lists can be written in three ways:**
  1. $.(s, t)$ **defines a list with the element** s **and the tail** t**;**
  2. $[s|t]$ **does the same;**
  3. $[s_1, \ldots, s_n]$ **defines a list with the elements** $s_1, \ldots, s_n$
- **Accordingly, these are equivalent lists:**

$$.(1, .(2, .(3, []))) \tag{51}$$

$$[1|[2|[3|[]]]] \tag{52}$$

$$[1, 2, 3] \tag{53}$$

- **We want to design a function `cat` that concatenates two lists `L1` and `L2` resulting in the list `L2`.**
- **In the world of logical programming, this could be conceived as the 3-ary function `cat(L1,L2,L3)` which becomes true iff `L3` is the concatenation of `L2` and `L2`.**
- **A respective Prolog program is:**

```
1 cat([X|L1],L2,[X|L3]):-cat(L1,L2,L3).
2 cat([],L,L).
```

- **This program reads**

  An empty list concatenated with a list L results in the same list L (Fact 2). Furthermore, if the concatenation of the lists L_1 and L_2 results in L_3, then L_1 with an preceding element X concatenated with L_2 must result in L_3 with the same preceding element X (Rule 1).

- **In the following, we run an example to understand the program's functionality.**

Lists: example function (cont.)

ID	CP	G	R	μ
1		$\texttt{cat}([1,2],[3,4],Y)$	$\texttt{cat}([X\|L_1],L_2,[X\|L_3]):-$	$[X\mapsto[1],L_1\mapsto[2],$
			$\texttt{cat}(L_1,L_2,L_3)$	$L_2\mapsto[3,4],Y\mapsto[1\|L_3]]$
2		$\texttt{cat}([2],[3,4],L_3)$	$\texttt{cat}([X'\|L_1'],L_2',[X'\|L_3']):-$	$[X'\mapsto[2],L_1'\mapsto[],$
			$\texttt{cat}(L_1',L_2',L_3')$	$L_2'\mapsto[3,4],L_3\mapsto[2\|L_3']]$
3		$\texttt{cat}([],[3,4],L_3')$	$\texttt{cat}([X''\|L_1''],L_2'',[X''\|L_3'']):-$	Ω
			$\texttt{cat}(L_1'',L_2'',L_3'')$	
4		$\texttt{cat}([],[3,4],L_3')$	$\texttt{cat}([],L,L):-$	$[L\mapsto[3,4],L_3'\mapsto[3,4]]$

Prolog's response is hence:

$$\begin{aligned} Y &\mapsto [1|L_3] \\ &\mapsto [1|[2|L_3']] \\ &\mapsto [1|[2|[3,4]]] \\ &= [1,2,3,4] \end{aligned} \tag{54}$$

- **You may perceive some flavor of Prolog's elegance if you consider which use cases the above example function features:**
  - **Concatenate two lists:**

$$\texttt{cat}([1,2],[3,4],\boldsymbol{Y}). \tag{55}$$

  - **Check whether a list resulted from another list by way of concatenation:**

$$\texttt{cat}([1,2],\boldsymbol{Y},[1,2,3,4]). \tag{56}$$

  - **Find all possible splits of a list into two lists:**

$$\texttt{cat}(\boldsymbol{X},\boldsymbol{Y},[1,2,3,4]). \tag{57}$$

- **Consider the following program:**

```
1 a.
2 a:-b.
3 b:-a.
```

- **We get the query result**

$$\texttt{a.} \quad \rightarrow \quad \texttt{Yes.} \tag{58}$$

- **Now, we reorder the rules:**

```
1 a:-b.
2 a.
3 b:-a.
```

- **This, time, the query result is**

$$\texttt{a.} \quad \rightarrow \quad \texttt{ERROR : Out of local stack.} \tag{59}$$

- **The inference algorithm keeps accessing Rule 1 over and over again.**
- **Other than the example on Page 105, this time, we do not get an infinite loop but a stack overflow.**
- **This is because Prolog has to create a choice point for every recursion due to the presence of the alternative Rule 3.**

- **Consider the following program:**

```
1  s([X],[X]).
2  s([A,B],[A,D]):-s([B],[D]),A<D.
```

- **We get the query result**

$$\texttt{s([1,2],X).} \quad \rightarrow \quad \texttt{X} = [1,2]. \tag{60}$$

- **Now, we switch the elements in Rule 2's body:**

```
1  s([X],[X]).
2  s([A,B],[A,D]):-A<D,s([B],[D]).
```

- **This time, we get**

s([1,2],X). → ERROR : Arguments are not sufficiently instantiated.

- **The inference algorithm tries to evaluate** A < D **first, before** D **had been determined by way of evaluating** s([B],[D])**.**

- **Consider the following program:**

```
p(X,Y):-Y==X+1.
q(X,Y):-Y>X+0.9999999,Y<X+1.0000001.
r(X,Y):-Y is X+1.
s(X,Y):-Y=X+1.
```

- **We get the following query results:**

```
p(1,2).  →  No.
q(1,2).  →  Yes.
r(1,2).  →  Yes.
s(1,2).  →  No.
p(1,Y).  →  No.
q(1,Y).  →  ERROR : Arguments are not sufficiently instantiated.
r(1,Y).  →  Y = 2.
s(1,Y).  →  Y = 1 + 1.
```

Symbolic vs. numerical computation (cont.)

- **The check for equality (==) fails due to issues with Prolog's numerical precision.**
- **Rather than for equality, `q` checks for a small range around the expected value and thereby succeeds. When queried with the free parameter `Y`, however, Prolog is not able to limit the (real-valued) search space and complains about insufficient instantiation.**
- **Prolog's keyword `is` assigns the exact value of $X + 1$ to `Y` and therefore succeeds. Accordingly, the free parameter `Y` gets assigned the sum of 1 and 1.**
- **The unification operator = tries to solve the syntactical equation $2 = 1 + 1$ which is not possible since different function symbols cannot be unified. Hence, it fails. When queried with a free parameter, however, the syntactical equation is $Y = 1 + 1$ whose solution is the result set.**

Prolog: exercises

- **Write programs to**
  1) **determine the maximum of two numbers (2 lines)**
  2) **calculate the factorial (2 lines)**
  3) **uniq a list (3 lines)**
  4) **find identical elements in two lists (3 lines)**
  5) **sort a list (4 lines)**

- **Deadlines:**

group	introduction	proposal due	code due	presentation
AIA	March 26	March 31	April 18	April 23 (?)
AIB	March 22	March 27	April 14	April 19
AIC	March 22	March 27	April 14	April 25 (?)
AID	March 27	April 1	April 19	April 24

- **Proposals have to be submitted to all of the following e-mail addresses:**

  `david@suendermann.com`

  `suender@icsi.berkeley.edu`

  `suendermann@dhbw-stuttgart.de`

- **The subject line of the e-mail has to contain**

  a) **the word "proposal",**

  b) **your first and last name(s),**

  c) **your matriculation number(s), and**

  d) **your group ID(s) including the year (e.g. AIA10).**

Notes for the Prolog programming project (cont.)

- **Up to two students can collaborate on each project.**
- **Proposals have to contain a brief (no more than 200 words) but clear description of what your program is supposed to achieve and how typical rules and queries are expected to look like.**
- **The Prolog code of your project needs to be well-documented according to common coding standards. For a Prolog coding style reference, see:**

  `http://www.ai.uga.edu/mc/plcoding.pdf`
- **Make sure the code clearly shows example queries to run the code.**
- **There is no need for any documentation outside of the code itself.**
- **Submit your program to the above listed e-mail addresses with the above described subject line, replacing "proposal" by "code".**
- **Do not dare to copy anybody else's code. Every identified attempt will cause failure of the project.**
- **So does missing a deadline without prior permission or medical certificate.**

Notes for the Prolog programming project (cont.)

- **Presentations will be held in the class room during the time of the regular lecture. An exact schedule will be compiled shortly before.**
- **Presentations are 15 minutes in duration. During this time, you need to convince me to respond positively to the questions in the table below.**
- **To derive the coding project's final score, I will consider answers to the following questions:**

Does the proposal address a sufficiently challenging KBS task?	20%
Does the program fulfill the proposed task?	40%
Was the code well documented?	20%
Was the project well presented?	20%

- **In doing so, I will generally apply a weighting scheme according to the percentages of the table (exceptions are possible, e.g., if somebody proposes a very challenging task and does not submit anything useful, there is no right to claim the 20% associated with difficulty, either).**

Examples of Previous Prolog programming projects

- **Four in a Row**
- **Freecell**
- **Davis and Putnam algorithm for propositional logic**
- **Hidden-Markov Models for Emotion Recognition**
- **Huffman Code**
- **Levenshtein Distance**
- **Lisp**
- **Minesweeper**
- **Pacman**
- **Peg Solitaire**
- **Rubik's Cube**
- **Towers of Hanoi**
- **Zork**