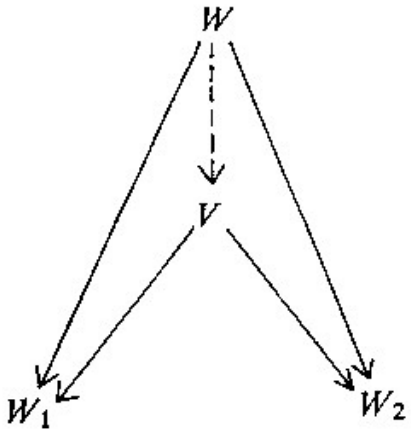
Learning in Logic



Tamás Horváth

University of Bonn &
Fraunhofer IAIS, Sankt Augustin, Germany
tamas.horvath@iais.fraunhofer.de

universitätbonn
RHEINISCHE FRIEDRICH-WILHELMS-UNIVERSITÄT

Fraunhofer
IAIS

Learning in Logic - Inductive Generalization of Clauses

observations:

A: The result of heating **this** bit of iron to 419 °C was that it melted.

B: The result of heating **that** bit of iron to 419 °C was that it melted.

induction: The result of heating **any** bit of iron to 419 °C was that it melted.

observations expressed in first-order logic:

A: $\text{Bitofiron}(\text{bit}_1) \wedge \text{Heated}(\text{bit}_1,419) \rightarrow \text{Melted}(\text{bit}_1)$

B: $\text{Bitofiron}(\text{bit}_2) \wedge \text{Heated}(\text{bit}_2,419) \rightarrow \text{Melted}(\text{bit}_2)$

induction: $\forall x(\text{Bitofiron}(x) \wedge \text{Heated}(x,419) \rightarrow \text{Melted}(x))$

as clause: $\forall x(\neg\text{Bitofiron}(x) \vee \neg\text{Heated}(x,419) \vee \text{Melted}(x))$

Learning in Logic

- **inductive generalization of first-order clauses**
  1. **preliminaries**
  2. **generalization of words (terms and literals)**
  3. **generalization of clauses**

First-Order Logic – Syntax (Alphabet)

- a set of **constants** $a, b, \ldots$
- a set of **variables** $u, v, w, x, y, \ldots$
- a set of **function symbols** $f, g, \ldots$
  - each function symbol is associated with a natural number (its arity)
  - constants can be considered as function symbols of arity 0
- a non-empty set of **predicate symbols** $P, Q, \ldots$
  - each predicate symbol is associated with a natural number (its arity)
- **connectives**: $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$
- **quantifiers**: $\forall$, $\exists$
- **punctuation symbols**: '(', ')', and '.'

First-Order Logic – Syntax (Terms)

- set of **terms** over an alphabet is the smallest set satisfying
  1. any constant is a term
  2. any variable is a term
  3. if f is an n-ary function symbol and $t_1, \ldots, t_n$ are terms then $f(t_1, \ldots, t_n)$ is a term

- **examples:**
  - a
  - x
  - $g(x_2, x_1, f(f(f(a))))$

First-Order Logic – Syntax (Formulas)

- set of **formulas** over an alphabet is the smallest set satisfying

  1. if P is an n-ary predicate symbol and $t_1, \ldots, t_n$ are terms then $P(t_1, \ldots, t_n)$ is a formula, called **atom**
  2. if ϕ is a formula then $\neg\phi$ is a formula
  3. if ϕ and ψ are formulas then $(\phi \wedge \psi)$, $(\phi \vee \psi)$, $(\phi \rightarrow \psi)$, $(\phi \leftrightarrow \psi)$ are formulas
  4. if ϕ is a formula and x is a variable then $\exists x\phi$ and $\forall x\phi$ are formulas

- **examples:**

  - $Q(a)$
  - S (S is a 0-ary predicate symbol)
  - $(\forall x_1 \exists x_2 R(x_2, x_1) \wedge \forall y Q(y))$

First-Order Logic – Syntax (First-Order Language)

- The **first-order language** given by an alphabet is the set of all formulas which can be constructed from the symbols of the alphabet.
- **further notions:**
  - the **scope** of $\forall x$ (respectively $\exists x$) in $\forall x\phi$ (respectively $\exists x\phi$) is ϕ
  - the **bound occurrence** of a variable x in a formula is an occurrence of x immediately following a quantifier, or an occurrence of x within the scope of a quantifier that is immediately followed by x
  - an occurrence of a variable is **free** if it is not bound
  - a **closed formula** is a formula which does not contain any free occurrences of variables
  - a **ground term** (respectively a **ground formula**) is a term (resp. a formula) which does not contain any variables

First-Order Logic – Semantics (Pre-Interpretation)

- **pre-interpretation** J of a first-order language L consists of the following:
  - A non-empty set D, called the **domain** of the pre-interpretation.
  - Each constant in L is assigned an element of D.
  - Each n-ary function f in L is assigned a function $J_f : D^n \rightarrow D$.
- Let J be a pre-interpretation with domain D of a first-order language L. A **variable assignment** V with respect to L is a mapping from the set of variables in L to the domain D of J.
  - **notation:** $V(x/d)$ denotes the variable assignment which maps the variable x to $d \in D$, and maps the other variables according to V

First-Order Logic – Semantics (Term Assignment)

- Let J be a pre-interpretation with domain D of a first-order language L, and let V be a variable assignment with respect to L. The **term assignment** with respect to J and V of the terms in L is the following mapping from the set of terms in L to the domain D:
  - Each constant is mapped to an element in D by J.
  - Each variable is mapped to an element in D by V.
  - If $d_1, \ldots, d_n$ are the elements of the domain to which the terms $t_1, \ldots, t_n$ are mapped, respectively, then the term $f(t_1, \ldots, t_n)$ is mapped to $J_f(d_1, \ldots, d_n)$, where J_f is the function assigned to f by J.

First-Order Logic – Semantics (Interpretation, Truth Value)

- An **interpretation** I of a first-order language L consists of the following:
  - A pre-interpretation J, with some domain D, of L. I is said to be based on J.
  - Each n-ary predicate symbol P in L is assigned a function I_P mapping D^n to $\{T, F\}$.
- Let I be an interpretation, based on the pre-interpretation J with domain D, of the first-order language L, and let V be a variable assignment with respect to L. Let Z be the term assignment with respect to J and V. Then a formula ϕ in L has a **truth value under I and V** as follows: (*next slide ...*)

First-Order Logic – Semantics (Interpretation, Truth Value)

(i) If ϕ is the atom $P(t_1, \ldots, t_n)$, and $d_i \in D$ is assigned to t_i by Z $(i = 1, \ldots, n)$, then the truth value of ϕ under I and V is $I_P(d_1, \ldots, d_n)$.

(ii) If ϕ is the formula of the form $\neg\psi$, $(\psi \wedge \chi)$, $(\psi \vee \chi)$, $(\psi \rightarrow \chi)$, $(\psi \leftrightarrow \chi)$ then the truth value of ϕ under I and V is determined by the truth value of the corresponding propositional formula obtained by taking the truth values of ψ and χ.

(iii) If ϕ is the formula of the form $\exists x\psi$, then ϕ has the truth value T under I and V if there exists an element $d \in D$ for which ψ has truth value T under I and $V(x/d)$; otherwise it has the truth value F.

(iv) If ϕ is the formula of the form $\forall x\psi$, then ϕ has the truth value T under I and V if for all $d \in D$, ψ has truth value T under I and $V(x/d)$; otherwise it has the truth value F.

First-Order Logic – Semantics (Model, Implication)

- Let ϕ be a *closed* formula in a first-order language L, and I an interpretation of L. Then ϕ is said to be **true under** I if its truth value under I is T. I is then said to satisfy ϕ. Similarly, ϕ is said to be **false under** I if its truth value under I is F. I is then said to falsify ϕ.

- Let ϕ be a closed formula in a first-order language L, and I an interpretation of L. Then I is a **model** of ϕ if I satisfies ϕ.

- Let Σ be a set of closed formulas in a first-order language L, and I an interpretation of L. Then I is a **model** of Σ if I is a model of all formulas in Σ.

- Let Σ be a set of closed formulas and ϕ a closed formula in a first-order language L. Then ϕ is a **logical consequence** of Σ, or Σ **logically implies** ϕ, denoted $\Sigma \models \phi$, if every model of Σ is also a model of ϕ.

Clauses

- a **literal** is an atom or the negation of an atom; a **positive literal** is an atom and a **negative literal** is the negation of an atom
- a **clause** is a *closed* formula of the form

$$\forall x_1 \ldots \forall x_m (A_1 \vee \ldots \vee A_n \vee \neg B_1 \vee \ldots \vee \neg B_p) \ ,$$

where the A_i's and B_j's are positive literals.

  - all variables are quantified
  - usually will be considered as the set $\{A_1, \ldots, A_n, \neg B_1, \ldots, \neg B_p\}$

Thm (Schmidt-Schauss, 1988): The problem whether $C \models D$, where C and D are arbitrary clauses, is undecidable.

Substitutions

- a **substitution** θ is a finite set of the form

$$\{x_1/t_1, \ldots, x_n/t_n\}$$

for some $n \geq 0$, where the x_i's are distinct variables and the t_i's are terms.

  - θ is a **ground substitution** if all the t_i's are ground
  - θ is the **identity substitution** if $\theta = \emptyset$

- an **expression** is either a term, a literal, or a conjunction or disjunction of literals; a **word** is a term or a literal

Substitutions

- Let $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$ be a substitution and E an expression. Then $E\theta$, the **instance** of E by θ, is the expression obtained from E by simultaneously replacing each occurrence of x_i by t_i $(1 \leq i \leq n)$.
  - if $E\theta$ is ground, then $E\theta$ is called a **ground instance**

example:

$$\begin{aligned} E &= P(y, f(x)) \\ \theta &= \{x/a, y/g(g(x))\} \\ E\theta &= P(g(g(x)), f(a)) \end{aligned}$$

Substitutions

- Let $\theta = \{x_1/s_1, \ldots, x_m/s_m\}$ and $\sigma = \{y_1/t_1, \ldots, y_n/t_n\}$ be substitutions. The **composition** of θ and σ, denoted $\theta\sigma$, is the substitution obtained from

$$\{x_1/(s_1\sigma), \ldots, x_m/(s_m\sigma), y_1/t_1, \ldots, y_n/t_n\}$$

by deleting all $x_i/(s_i\sigma)$ for which $x_i = s_i\sigma$, and any y_j/t_j for which $y_j \in \{x_1, \ldots, x_m\}$.

example:

$$\begin{aligned} \theta &= \{x/f(y), z/u, y/u\} \\ \sigma &= \{y/b, u/z)\} \\ \theta\sigma &= \{x/f(b), \underline{z/z}, y/z, \underline{y/b}, u/z\} \setminus \{z/z, y/b\} = \{x/f(b), y/z, u/z\} \end{aligned}$$

Substitutions

- Let E be an expression and $\theta = \{x_1/y_1, \ldots, x_n/y_n\}$ be a substitution. Then θ is a **renaming substitution** for E if each x_i occurs in E and $y_1, \ldots, y_n$ are pairwise distinct variables such that each y_i is either equal to some x_j in θ, or y_i does not occur in E.

  example: for $E = f(a, x, y, z)$ and $\theta = \{x/x_1, z/x\}$, we have $E\theta = f(a, x_1, y, x)$

- Let E and F be expressions. Then E and F are **alphabetical variants** if there are renaming substitutions θ and σ such that $E = F\theta$ and $F = E\sigma$.

  example: $E = P(x) \vee Q(x, y)$ and $F = P(y) \vee Q(y, z)$ are alphabetical variants because $E\{x/y, y/z\} = F$ and $F\{y/x, z/y\} = E$

  Prop. Let E and F be expressions and θ and σ substitutions such that $E\theta = F$ and $F\sigma = E$. Then E and F are alphabetical variants.

  proof: *exercise*

Learning in Logic

- **inductive generalization of first-order clauses**
  1. **preliminaries**
  2. **generalization of words (terms and literals)**
  3. **generalization of clauses**

Generalization of Words: Notions

- **word:** term or literal
- W_1 is **more general than** W_2 ($W_1 \leq W_2$) if there exists a substitution θ such that $W_1\theta = W_2$

example: $P(x, x, f(g(y))) \leq P(l(3), l(3), f(g(x)))$

because $P(x, x, f(g(y)))\theta = P(l(3), l(3), f(g(x)))$ for $\theta = \{x/l(3), y/x\}$

Generalization of Words: Notions

- $\leq$ is a **quasi-order** (or **preorder**) on the set of words,
  - i.e., $\leq$ is reflexive and transitive
- $W_1 \backsim W_2$ if $W_1 \leq W_2$ and $W_2 \leq W_1$

Prop.: $W_1 \backsim W_2$ iff W_1 and W_2 are alphabetical variants

proof: *exercise*

Inductive Generalization of Words

- **generalization of words:** a word W is a generalization of a set $\mathcal{K}$ of words iff $W \leq V$ for all $V \in \mathcal{K}$

- **least general generalization (LGG) of words:** a word W is a least general generalization of a set $\mathcal{K}$ of words iff
  - W is a generalization of $\mathcal{K}$ and
  - $W' \leq W$ for any generalization W' of $\mathcal{K}$

$\Rightarrow$ if W_1 and W_2 are any two LGGs of $\mathcal{K}$ then $W_1 \backsim W_2$

Inductive Generalization of Words

example:

$$\mathcal{K} = \{P(f(a, g(y)), x, g(y)), P(h(a, g(x)), x, g(x))\}$$

- generalization of $\mathcal{K}$: $P(x, y, z)$
- LGG of $\mathcal{K}$: $P(x, y, g(z))$

Inductive Generalization of Words

- two words are **compatible** iff they are both terms or have the same predicate symbol and sign

Theorem (Plotkin, 1970): Every non-empty, finite set $\mathcal{K}$ of words has a least general generalization if and only if any two words in $\mathcal{K}$ are compatible.

proof: The compatibility condition is necessary. For the other direction, we give an algorithm (next slide) and show that this algorithm computes the LGG for any two compatible words.

Inductive Generalization of Words

Def. We say that a term t is in the I-th **place** in a word W iff:

- $t = W$ and $I =<>$ or
- $W = \varphi(t_1, \ldots, t_m)$, I is a sequence $< i_1, \ldots, i_n >$ of positive integers, $i_1 \leq m$, and t is in the $< i_2, \ldots, i_n >$-th place in t_{i_1}.

example: x is in the $< 3, 2 >$-th place in $P(a, b, g(y, x))$

The Algorithm

input: two compatible words W_1 and W_2

output: LGG of W_1 and W_2

1. $V_i := W_i$ and $\epsilon_i = \emptyset$ $(i = 1, 2)$
2. Try to find terms t_1, t_2 such that they have the same place in V_1, V_2, respectively, $t_1 \neq t_2$, and either they begin with different function symbols or at least one of them is a variable.
3. If there are no such t_1, t_2 then return V_1. In this case, V_1 is an LGG of $\{W_1, W_2\}$, $V_1 = V_2$, and $V_i\epsilon_i = W_i$ $(i = 1, 2)$.
4. Choose a variable x distinct from any in V_1 or V_2 and wherever t_1 and t_2 occur in the same place in V_1 and V_2, respectively, replace each by x.
5. $\epsilon_i := \{x/t_i\}\epsilon_i$ $(i = 1, 2)$
6. go to 2

Example

example: $W_1 = P(f(a, g(y)), x, g(y)), W_2 = P(h(a, g(x)), x, g(x))$

- initially: $V_1 = P(f(a, g(y)), x, g(y)), V_2 = P(h(a, g(x)), x, g(x)), \epsilon_1 = \epsilon_2 = \emptyset$
- we take $t_1 = y$, $t_2 = x$ in place $< 1, 2, 1 >$ (step 3) and z as the new variable (step 4) and get after step 4:

$$V_1 = P(f(a, g(z)), x, g(z)), V_2 = P(h(a, g(z)), x, g(z))$$

and after step 5: $\epsilon_1 = \{z/y\}$, $\epsilon_2 = \{z/x\}$

(cont'd on the next slide ...)

Example (Cont'd)

- next we take $t_1 = f(a, g(z))$, $t_2 = h(a, g(z))$ in place $< 1 >$ (step 3) and u as the new variable (step 4) and get after step 4:

$$V_1 = P(u, x, g(z)) = V_2$$

and after step 5:

$$\epsilon_1 = \{u/f(a, g(z))\}\epsilon_1 = \{u/f(a, g(y), z/y\}$$
$$\epsilon_2 = \{u/h(a, g(z))\}\epsilon_2 = \{u/h(a, g(x), z/x\}$$

- the algorithm halts and returns $P(u, x, g(z))$ as the LGG

Proof of Plotkin's Theorem

Let $\mathcal{K} = \{W_1, \ldots, W_n\}$ be a finite set of compatible words.

- If $n = 1$ then the theorem is trivial.
- For $n > 1$ the following proposition holds:

Prop. 1 $LGG(\mathcal{K}) = LGG(\{W_1, LGG(\{W_2, \ldots, LGG(\{W_{n-1}, W_n\})\ldots\})\})$

proof: *exercise*

$\Rightarrow$ to prove the theorem, it is sufficient to show that the previous algorithm works

Proof of Plotkin's Theorem – Definitions and Notions

Def.: two terms t_1 and t_2 are **replacable** in V_1 and V_2, respectively, if they fulfill the conditions of step 2 of the algorithm, i.e.,

- $t_1 \neq t_2$
- either they begin with different function symbols or at least one of them is a variable

Def.: the **difference** of two words V_1 and V_2 are defined by

$$\text{diff}(V_1, V_2) = |\{I : \text{ the terms in } V_1 \text{ and } V_2 \text{ in place } I \text{ are replacable}\}|$$

Notation: V_1' and V_2' denote the result of replacing t_1 and t_2 in V_1 and V_2, respectively, by a new variable in the way described in step 4 of the algorithm

Main Steps of the Proof of Plotkin's Theorem

(i) every time a pair of replaceable terms is replaced the difference drops (by **Lemma 3**)

(ii) the difference will eventually become zero and when it does we must have $V_1 = V_2$ and the algorithm will then halt (by (i) and **Lemma 3**)

(iii) $V_i' < V_i$ because $V_i'\{x/t_i\} = V_i$ $(i = 1, 2)$ (by **Lemma 4**)

$\Rightarrow$ when the algorithm terminates, we must have $V_i'\epsilon_i = W_i$ $(i = 1, 2)$

$\Rightarrow$ the output is a **generalization** of $\{W_1, W_2\}$

(iv) every generalization of $\{W_1, W_2\}$ is more general than the output (by **Lemma 5**)

Lemma 2

Lemma 2: If V_1 and V_2 are distinct compatible words, then there are t_1, t_2 in V_1, V_2, respectively, which are replaceable in them.

proof: by induction on words on V_1

- trivially holds, if one of V_1, V_2 is a constant or a variable, or if they begin with different function symbols
- if $V_1 = \varphi(t_1^1, \ldots, t_n^1)$ and $V_2 = \varphi(t_1^2, \ldots, t_n^2)$, then there exists an i with $t_i^1 \neq t_i^2$

  $\Rightarrow$ applying the induction hypothesis to t_i^1, t_i^2, there are u_1, u_2 which are replaceable in t_i^1, t_i^2, respectively

  $\Rightarrow$ as t_i^1, t_i^2 are in the same place in V_1, V_2 (i.e., $< i >$), respectively, u_1, u_2 are replaceable in V_1, V_2

Lemma 3

Lemma 3: if V_1 and V_2 are distinct compatible words, then

$$\text{diff}(V_1', V_2') < \text{diff}(V_1, V_2) \ .$$

proof: by induction on words on V_1

case 1: if one of V_1, V_2 is a constant or a variable, then $t_1 = V_1$, $t_2 = V_2$, and $V_1' = V_2' = x$, so

$$0 = \text{diff}(V_1', V_2') < \text{diff}(V_1, V_2) = 1$$

case 2: *next slide ...*

Proof of Lemma 3 (cont'd)

case 2: if $V_1 = f(v_1, \ldots, v_n)$ and $V_2 = g(u_1, \ldots, u_m)$ with $f \neq g$ then

case 2.1: if $t_i = V_i$ $(i = 1, 2)$, then $0 = \text{diff}(V_1', V_2') < \text{diff}(V_1, V_2)$ by Lemma 2;

case 2.2: otherwise

$$\begin{aligned} \text{diff}(V_1', V_2') &= 1 + \sum_{i=1}^{\min(m,n)} \text{diff}(v_i', u_i') \\ &< 1 + \sum_{i=1}^{\min(m,n)} \text{diff}(v_i, u_i) \quad \text{// by the induction hyp.} \\ &= \text{diff}(V_1, V_2) \end{aligned}$$

case 3: if $V_1 = f(v_1, \ldots, v_n)$ and $V_2 = f(u_1, \ldots, u_n)$ then the proof is similar ...

Lemma 4

Lemma 4: $V_i'\{x/t_i\} = V_i \ (i = 1, 2)$

proof: *exercise*

Lemma 5

Lemma 5: if V_1 and V_2 are distinct compatible words and $V\sigma_i = V_i$ $(i = 1, 2)$, then there are σ'_1, σ'_2 such that $V\sigma'_i = V'_i$ $(i = 1, 2)$

proof: denote by $f_i(u_1, u_2, t_1, t_2)$ the result of applying the operation in step 4 of the algorithm to u_1, u_2 on u_i $(i = 1, 2)$, and let

$$\begin{aligned} \sigma_i &= \{y_1/u_i^1, \ldots, y_m/u_i^m\} \ (i = 1, 2) \\ v_i^j &= f_i(u_1^j, u_2^j, t_1, t_2) \ (i = 1, 2; j = 1, \ldots, m) \\ \sigma'_i &= \{y_1/v_i^1, \ldots, y_m/v_i^m\} \ (i = 1, 2) \end{aligned}$$

- by Lemma 4: $\sigma_i = \sigma'_i\{x/t_i\}$ $(i = 1, 2)$.

Proof of Lemma 5 (cont'd)

we show by induction on V that:

If V, V_1, V_2 are such that $V\sigma_i = V_i$ $(i = 1, 2)$ then $V\sigma'_i = V'_i$ $(i = 1, 2)$.

case 1: If V is a constant, then $V = V_1 = V_2$, and the result is trivial.

case 2: If V is a variable y then:

case 2.1: If $y \neq y_i$ for $i = 1, \ldots, m$, then $y = V = V_1 = V_2$, and the claim is trivial.

case 2.2: If $y = y_j$ for some j, then $V\sigma'_i = f_i(V_1, V_2, t_1, t_2) = V'_i$.

case 3: Suppose $V = \varphi(u_1, \ldots, u_n)$ then if $V'_i = \varphi(w^i_1, \ldots, w^i_n)$,

$$\begin{aligned} V\sigma'_i &= \varphi(u_1\sigma'_i, \ldots, u_n\sigma'_i) \\ &= \varphi(w^i_1, \ldots, w^i_n) \qquad \text{(induction hypothesis)} \\ &= V'_i \end{aligned}$$

Learning in Logic

- **inductive generalization of first-order clauses**
  1. generalization of words (terms and literals)
  2. **generalization of clauses**
  3. **example**

Generalization of Clauses

- C, D: first-order clauses
- C is **more general than** D ($C \leq D$) if there exists a substitution θ such that $C\theta \subseteq D$

example:

$$\{P(x_1, x_2), P(x_2, x_3), P(x_3, x_4), P(x_4, x_1)\} \leq \{P(a_1, a_2), P(a_2, a_1)\}$$

because

$$\{P(x_1, x_2), P(x_2, x_3), P(x_3, x_4), P(x_4, x_1)\}\theta \subseteq \{P(a_1, a_2), P(a_2, a_1)\}$$

for

$$\theta = \{x_1/a_1, x_2/a_2, x_3/a_1, x_4/a_2\}$$

Generalization of Clauses

properties:

- $\leq$ is a **quasi-order** (or **preorder**) on the set of clauses
  - i.e., $\leq$ is reflexive and transitive
- $C \backsim D$ if $C \leq D$ and $D \leq C$

BUT! $C \backsim D$ does not imply that they are alphabetical variants

proof: *exercise*

Subsumption vs. Implication

let C, D be clauses

- $\leq$ is **sound**, i.e., $C \leq D$ implies $C \models D$
  - Why? (*exercise*)
- $\leq$ is **incomplete**, i.e., $C \models D$ does **not** imply $C \leq D$

  example: for $C = \{P(f(x)), \neg P(x)\}$ and $D = \{P(f(f(x))), \neg P(x)\}$ we have that $C \models D$, but $C \not\leq D$
- the problem of deciding if $C \models D$ is **undecidable**
- the problem of deciding if $C \leq D$ is **NP-complete**

  proof: *exercise*

Subsumption vs. Implication

Thm. (Gottlob, 1987): For any first-order clauses C and D, if C is not self-resolving and D is not tautological then $C \models D$ implies (and is thus equivalent to) $C \leq D$.

Proof: *omitted*

Least General Generalization of Clauses

- **generalization of clause:** a clause C is a generalization of a set $\mathcal{K}$ of clauses iff $C \leq D$ for all $D \in \mathcal{K}$
- **least general generalization (LGG) of clauses:** a clause C is a least general generalization of a set $\mathcal{K}$ of clauses iff
  - C is a generalization of $\mathcal{K}$ and
  - $C' \leq C$ for any generalization C' of $\mathcal{K}$

$\Rightarrow$ if C_1 and C_2 are any two LGGs of $\mathcal{K}$ then $C_1 \backsim C_2$

Least General Generalization of Clauses

example:

- $C_1 = \{\neg\text{Bitofiron}(\text{bit}_1), \neg\text{Heated}(\text{bit}_1,419), \text{Melted}(\text{bit}_1)\}$
- $C_2 = \{\neg\text{Bitofiron}(\text{bit}_2), \neg\text{Heated}(\text{bit}_2,419), \text{Melted}(\text{bit}_2)\}$
- generalization of $\mathcal{K} = \{C_1, C_2\}$: $\{\neg\text{Bitofiron(x)}, \neg\text{Heated(y,419)}, \text{Melted(z)}\}$
- LGG of $\mathcal{K}$: $\{\neg\text{Bitofiron(x)}, \neg\text{Heated(x,419)}, \text{Melted(x)}\}$

Least General Generalization of Clauses

Def. Let $S = \{C_1, \ldots, C_n\}$ be a set of clauses. A set $K = \{L_1, \ldots, L_n\}$ of *compatible* literals is a **selection** from S if and only if $L_i \in C_i$ for every $i = 1, \ldots, n$.

Thm. (Plotkin, 1970)

(i) Every finite set S of clauses has a non-empty LGG if and only if S has a selection.

(ii) If C_1 and C_2 are two clauses with at least one selection, then the following algorithm computes an LGG of C_1 and C_2.

proof:

(i) *exercise*

(ii) we first give an algorithm and then show that it is correct

Proof of (ii): The Algorithm

input: $S = \{C_1, C_2\}$ with selections $\{L_l^1, L_l^2\}$ $(l = 1, \ldots, n)$, where $L_l^j \in C_j$ $(j = 1, 2)$. Suppose that $L_l^j = (\pm)P_l(t_{1,l}^j, \ldots, t_{k_l,l}^j)$, where $(\pm)P_l$ is either P_l or $\neg P_l$.

(i) Let f_l be a function symbol with arity k_l for $l = 1, \ldots, n$ and let P be a predicate symbol with arity n. Let M_j be the literal

$$P(f_1(t_{1,1}^j, \ldots, t_{k_1,1}^j), \ldots, f_n(t_{1,n}^j, \ldots, t_{k_n,n}^j)) \quad j = 1, 2$$

(ii) Compute the LGG M of the **words** $\{M_1, M_2\}$.

(iii) Suppose M is of the form $P(f_1(u_{1,1}, \ldots, u_{k_1,1}), \ldots, f_n(u_{1,n}, \ldots, u_{k_n,n}))$; then **return** the clause

$$C = \{(\pm)P_1(u_{1,1}, \ldots, u_{k_1,1}), \ldots, (\pm)P_n(u_{1,n}, \ldots, u_{k_n,1})\}$$

Example

$$C_1 = \{grandfather(abraham, jacob), \neg father(abraham, isaac), \neg parent(isaac, jacob)\}$$

$$C_2 = \{grandfather(kohath, miriam), \neg father(kohath, amram), \neg parent(amram, miriam)\},$$

selections:

$$(grandfather(abraham, jacob), grandfather(kohath, miriam)),$$
$$(\neg father(abraham, isaac), \neg father(kohath, amram)),$$
$$(\neg parent(isaac, jacob), \neg parent(amram, miriam)$$

step (i) words:

$$M_1 = P(f_1(abraham, jacob), f_2(abraham, isaac), f_3(isaac, jacob)),$$
$$M_2 = P(f_1(kohath, miriam), f_2(kohath, amram), f_3(amram, miriam))$$

step (ii) LGG of M_1 and M_2:

$$M = P(f_1(x, y), f_2(x, z), f_3(z, y))$$

step (iii) LGG of C_1 and C_2:

$$\{grandfather(x, y), \neg father(x, z), \neg parent(z, y)\}$$

Lemma A

Lemma A Let $\mathcal{K} = \{W_1, \ldots, W_n\}$ be a set of words with LGG W and with substitutions $W\mu_i = W_i$ for $i = 1, \ldots, n$.

1. If t is a term in W then t is an LGG of $\{t\mu_1, \ldots, t\mu_n\}$.
2. If x, y are variables in W and $x\mu_i = y\mu_i$ for $i = 1, \ldots, n$ then $x = y$.

proof: *omitted* (trivial)

Proof of (ii) on Slide 44

proof of (ii):

- by Plotkin's theorem for words: there are ν_1 and ν_2 such that

$$M\nu_1 = M_1 \text{ and } M\nu_2 = M_2$$

$$\Rightarrow C\nu_1 = \bigcup_{l=1}^{n} L_l^1 \subseteq C_1 \text{ and } C\nu_2 = \bigcup_{l=1}^{n} L_l^2 \subseteq C_2$$

$\Rightarrow C$ is a **generalization** of C_1 and C_2

Proof of (ii) on Slide 46

- we show that C is an LGG, i.e., $E \leq C$ for all generalizations E of C_1 and C_2
  - by **construction** and by **Lemma A**: $L_i = (\pm)P_i(u_{1,i}, \ldots, u_{k_i,i}) \in C$ is an **LGG** of $\{L_i^1, L_i^2\}$ for all $i = 1, \ldots, n$
- let $E = \{M_1, \ldots, M_m\}$ be a **generalization** of C_1 and C_2, and let α_1, α_2 be substitutions such that $E\alpha_1 \subseteq C_1$ and $E\alpha_2 \subseteq C_2$

  $\Rightarrow (M_p\alpha_1, M_p\alpha_2) = (L_p^1, L_p^2)$ is a selection for every $p = 1, \ldots, m$

  $\Rightarrow M_p \leq L_{p'}$, where $L_{p'} \in C$ is the corresponding LGG of $\{L_p^1, L_p^2\}$

  $\Rightarrow$ there is a substitution β_p such that $M_p\beta_p = L_{p'}$

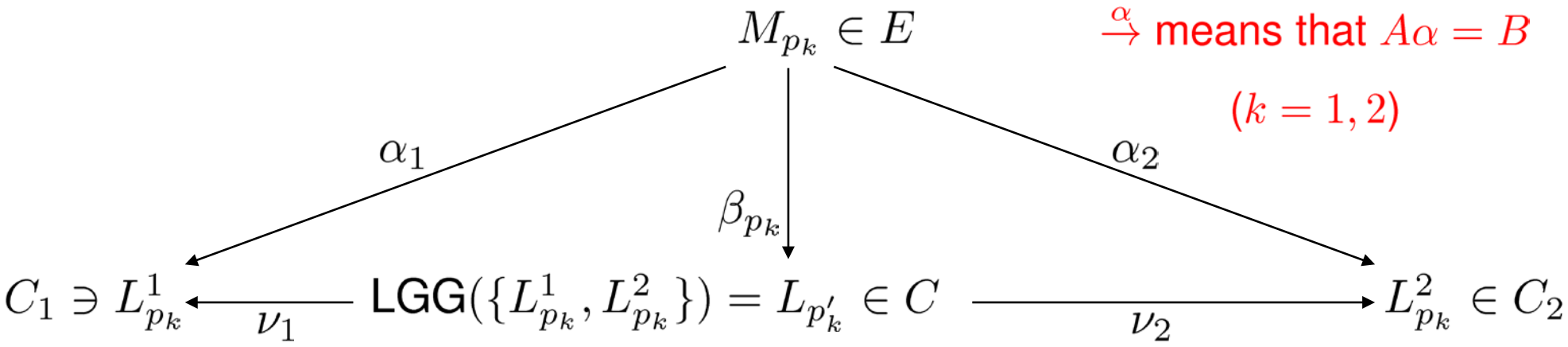
$\Rightarrow$ we need to show that $\bigcup_{p=1}^{m} \beta_p$ is a **substitution**

$\Rightarrow$ implies that $E\left(\bigcup_{p=1}^{m} \beta_p\right) \subseteq C$, i.e., $E \leq C$ as desired

Proof of (ii) on Slide 46

$\bigcup_{p=1}^{m} \beta_p$ exists if and only if **for all** variable x and **for all** $M_{p_1}, M_{p_2} \in E$

with $p_1 \neq p_2$ it holds that $x\beta_{p_1} = x\beta_{p_2}$



- $x\beta_{p_k}$ is a term in $L_{p'_k}$ $(k = 1, 2)$

$\Rightarrow$ $x\beta_{p_k}$ is an LGG of $\{x\beta_{p_k}\nu_1, x\beta_{p_k}\nu_2\} = \{x\alpha_1, x\alpha_2\}$ $(k = 1, 2)$ // by Lemma A

$\Rightarrow$ $x\beta_{p_1}$ and $x\beta_{p_2}$ are **alphabetical variants**

Proof of (ii) on Slide 46

- let x_1, x_2 be variables having the same place in $x\beta_{p_1}$, $x\beta_{p_2}$, respectively
- we know that $x\beta_{p_1}\nu_1 = x\beta_{p_2}\nu_1 = x\alpha_1$ and $x\beta_{p_1}\nu_2 = x\beta_{p_2}\nu_2 = x\alpha_2$

  $\Rightarrow x_1\nu_1 = x_2\nu_1$ and $x_1\nu_2 = x_2\nu_2$

  $\Rightarrow x_1 = x_2$ // by Lemma A.2

  $\Rightarrow x\beta_{p_1} = x\beta_{p_2}$

$\Rightarrow \bigcup_{p=1}^{m} \beta_p$ exists, i.e., it is a **substitution**

- this is what we wanted to prove (see Slide 49) Q.E.D.

Reduced Clauses

- if two clauses are equivalent then they are not necessarily alphabetical variants

example: $C_1 = \{P(x, y)\}, C_2 = \{P(u, v), P(u, z)\}$

- both $C_1 \leq C_2$ and $C_2 \leq C_1$ hold

$\Rightarrow$ any two LGGs of a set of clauses are equivalent to each other, but need not be alphabetical variants

Def.: a literal $L \in C$ is **redundant** iff $C \leq C \setminus \{L\}$

- $P(u, v)$ in C_2 above is redundant because $C_2 \leq \{P(u, z)\}$

Def.: a clause is **reduced** iff it is not equivalent to any proper subset of itself

- C_1 above is a reduced clause

Clause Reduction

Thm (Plotkin, 1970)

(i) If $C \backsim D$, and C and D are reduced, then they are alphabetical variants.

(ii) For a clause C, the algorithm on the next slide outputs a **reduced** clause E such that $C \backsim E$.

Clause Reduction Algorithm

algorithm:

input: clause C

output: reduced clause E such that $C \rightsquigarrow E$

1. $E := C$
2. Find a literal $L \in E$ and a substitution σ such that $E\sigma \subseteq E \setminus \{L\}$.
   If there is no such L, **return** E.
3. $E := E\sigma$ and go to 2

Example

let $C = \{P(u, v), P(u, z)\}$

1. $E := C = \{P(u, v), P(u, z)\}$
2. $L = P(u, v)$ and $\sigma = \{v/z\}$ satisfy

$$\begin{aligned} E\sigma &= \{P(u, v), P(u, z)\}\{v/z\} \\ &= \{P(u, z)\} \\ &\subseteq E \setminus \{L\} \end{aligned}$$

3. $E := E\sigma = \{P(u, v), P(u, z)\}\{v/z\} = \{P(v, z)\}$

2. no new literal can be selected, **return** $\{P(v, z)\}$

Proof of the Theorem on Slide 53

proof of (i)

- as $C \rightsquigarrow D$, there are μ and ν such that $C\mu \subseteq D$ and $D\nu \subseteq C$

  $\Rightarrow C\mu\nu \subseteq C$ and $D\nu\mu \subseteq D$

  $\Rightarrow$ since C are D reduced, $C\mu\nu = C$ and $D\nu\mu = D$

  $\Rightarrow$ by Lemma B (next slide), $\mu\nu$ and $\nu\mu$ are renaming substitutions

  $\Rightarrow \mu$ and ν are renaming substitutions

  $\Rightarrow C$ and D are alphabetical variants $\square$

Lemma B

Lemma B For any clause C, if $C\mu = C$ then C and $C\mu$ are alphabetic variants.

proof: we regard C as a set of literals ordered by $\leq$

- $C = C\mu^n$ for every $n \geq 0$

  $\Rightarrow \forall L \in C$, there is an $0 \leq k_L < |C|$ so that $L\mu^{k_L} \backsim L\mu^{k_L+i}$ for all $i \geq 0$

  $\Rightarrow$ there is an $0 \leq N < |C|$ such that $\forall L \in C$, $L\mu^N \backsim L\mu^{N+i}$ for all $i \geq 0$

  $\Rightarrow$ as $C\mu = C$, for all $L \in C$, there is an $M \in C$ such that $M\mu^N = L$

  $\Rightarrow L = M\mu^N \backsim M\mu^{N+1} = L\mu$

  $\Rightarrow \mu$ maps variables to variables

  - C and $C\mu$ have the same number of variables because $C = C\mu$

  $\Rightarrow \mu$ maps distinct variables of C to distinct variables of $C\mu$

  $\Rightarrow C$ and $C\mu$ are alphabetic variants $\square$

Proof of (ii) in the Theorem on Slide 53

proof of (ii):

- the algorithm terminates after at most $|C|$ iterations
- there is always a μ so that $C\mu \subseteq E$:
  - stage 1: $\mu = \emptyset$
  - if such a μ exists before stage 2: $\mu\sigma$ will be one after stage 2

$\Rightarrow$ when the algorithm halts: $C\mu \subseteq E$ and $E \subseteq C$, i.e., $C \backsim E$

Proof of (ii) in the Theorem on Slide 53 (cont'd)

proof of (ii): (previous slide: C and the output E are equivalent)

- suppose E is not reduced at termination

$\Rightarrow \exists E' \subsetneq E$ so that $E' \backsim E$

$\Rightarrow \exists \sigma$ such that $E\sigma \subseteq E'$

$\Rightarrow$ pick $L \in E \setminus E'$; we have $E\sigma \subseteq E' \subseteq E \setminus \{L\}$

$\Rightarrow$ contradicts that the algorithm has terminated $\square$

Putting All Together

- Every finite set S of clauses has a non-empty reduced LGG which is unique up to variable renaming if and only if S has a selection.

- For a set $S = \{C_1, \ldots, C_k\}$ of clauses, the LGG of S can be computed by

$$LGG(S) = LGG(\{C_1, LGG(\{C_2, \ldots, LGG(\{C_{k-1}, C_k\}) \ldots\})\})$$

Learning in Logic

- notions and notations
- **inductive generalization of first-order clauses**
  1. generalization of words (terms and literals)
  2. generalization of clauses
  3. **example**

Application Example: Relation Extraction from Texts

Problem: Automatic extraction of semantic relations between entities from natural language texts.

Example:

Training Data: Fraunhofer was a German optician.

Schrödinger was an Austrian - Irish physicist.

Planck was a German physicist.

Heisenberg was a celebrated German physicist and Nobel laureate.

Learning: definition of the ***unknown*** *target predicate* Is_Physicist(***X*** , ***Y***)

Prediction: Einstein was a German theoretical physicist.

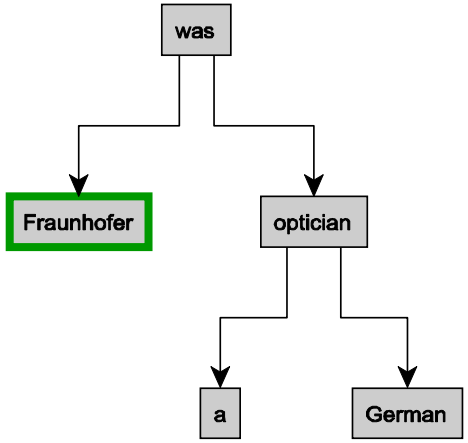
Data Preprocessing

sentences → **dependency trees**

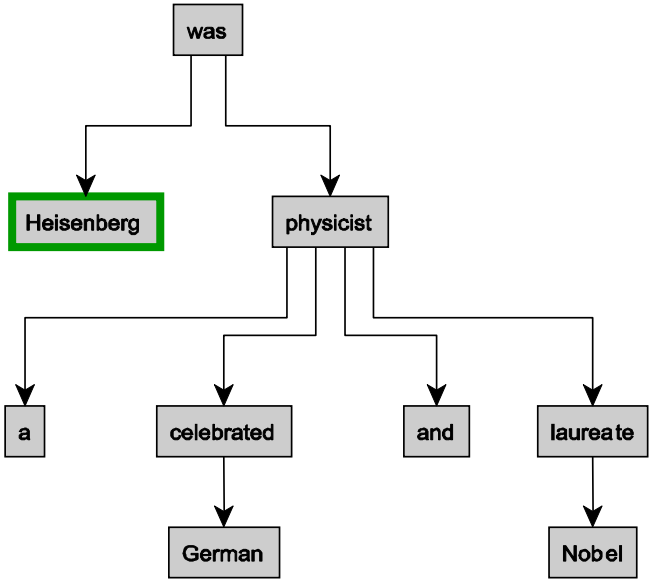
- labeled rooted directed trees representing grammatical dependencies among the words in a sentence
- capture a *low-level* syntactic structure of sentences
- *bijective map* between words in a sentence and nodes in the tree
- generated by the *Stanford Parser*
- nodes defining the same entity are *merged* into a single node
  - e.g., Ludwig Wittgenstein → Ludwig_Wittgenstein

Example: Unary Target Relation

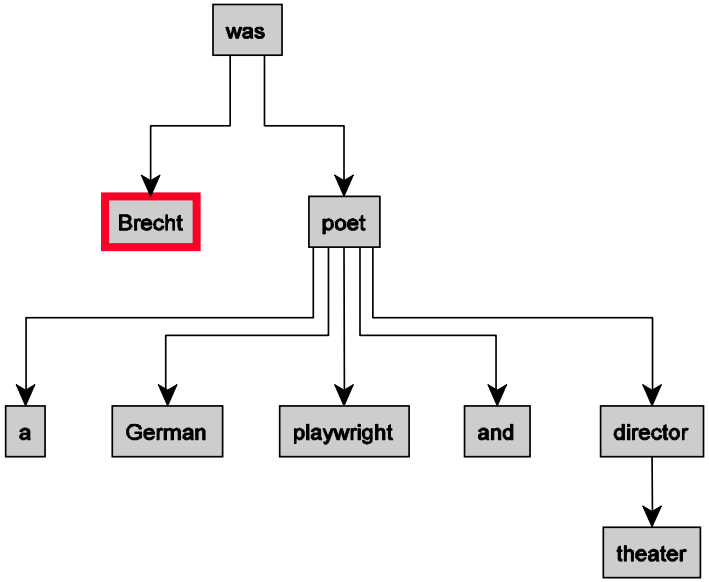
Fraunhofer was a German optician.



Heisenberg was a celebrated German physicist and Nobel laureate.



Brecht was a German poet, playwright, and theatre director.



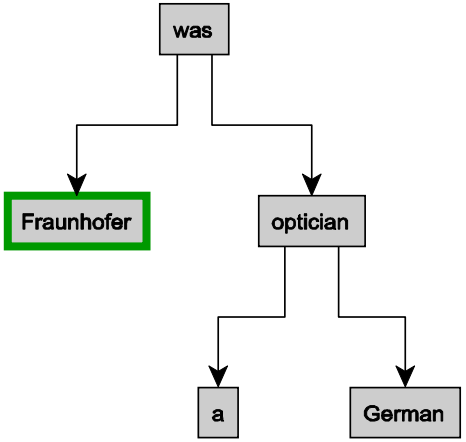
unknown target relation: **Is_Physicist** (unary)

POS: { Is_Physicist(Fraunhofer), Is_Physicist(Heisenberg) }

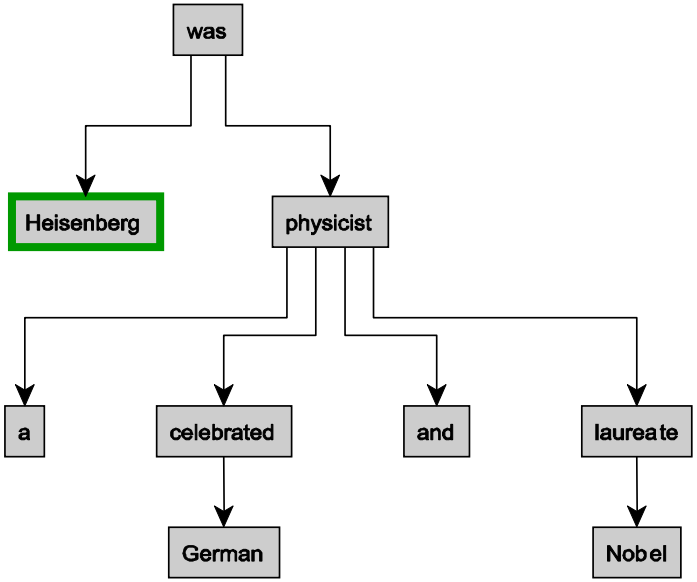
NEG: { Is_Physicist(Brecht) }

Example – Generalization of Dependency Trees

Fraunhofer was a German optician.



Heisenberg was a celebrated German physicist and Nobel laureate.



POS: { Is_Physicist(Fraunhofer), Is_Physicist(Heisenberg)}

We want to generalize these two structures!

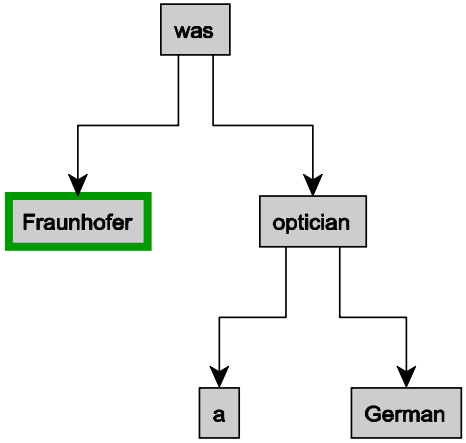
Consider them as ground clauses and use Plotkin's LGG algorithm!

Dependency Trees as Relational Structures

- labeled trees are considered as *relational structures*
  - unique *constant* for each vertex
  - *unary* and *binary* predicates only
  - ground
- **training examples**: *m-tuples* of vertices of the dependency trees
  - *P*: *m*-ary *target relation* to be learned
    - POS: set of instances (*m*-tuples) of the target relation *P*
    - NEG: set of non-instances (*m*-tuples) of the target relation *P*
  - ground atoms of the target predicate *P*

Example - Dependency Trees as Ground Clauses

Fraunhofer was a German optician.



comes from semantic hierarchy (e.g., Wordnet)

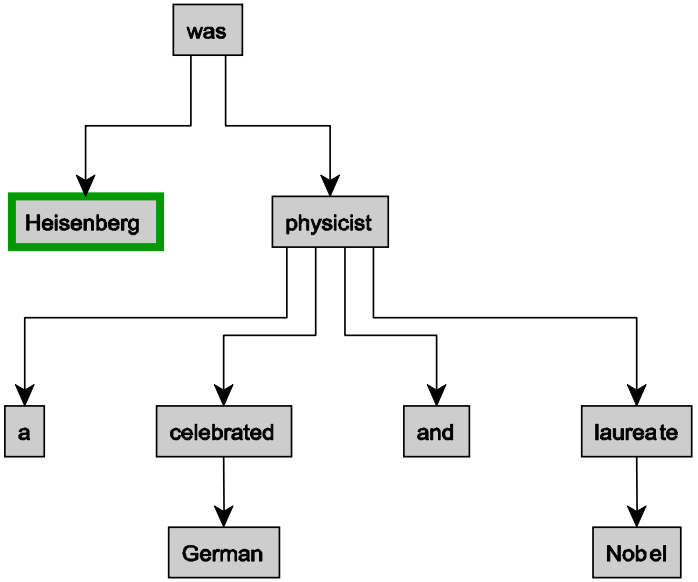
$$\{\neg \text{was}(a_{1,1}), \neg \text{Fraunhofer}(a_{1,2}), \neg \text{optician}(a_{1,3}), \neg \text{physicist}(a_{1,3}), \neg \text{a}(a_{1,4}), \neg \text{German}(a_{1,5}),$$
$$\neg R(a_{1,1}, a_{1,2}), \neg R(a_{1,1}, a_{1,3}), \neg R(a_{1,3}, a_{1,4}), \neg R(a_{1,3}, a_{1,5}), \text{is\_physicist}(a_{1,2})\}$$

or **equivalently**

$$\text{was}(a_{1,1}) \wedge \text{Fraunhofer}(a_{1,2}) \wedge \text{optician}(a_{1,3}) \wedge \text{physicist}(a_{1,3}) \wedge \text{a}(a_{1,4}) \wedge \text{German}(a_{1,5}) \wedge$$
$$\wedge R(a_{1,1}, a_{1,2}) \wedge R(a_{1,1}, a_{1,3}) \wedge R(a_{1,3}, a_{1,4}) \wedge R(a_{1,3}, a_{1,5}) \rightarrow \text{is\_physicist}(a_{1,2})$$

Example - Dependency Trees as Ground Clauses

Heisenberg was a celebrated German physicist and Nobel laureate.



$$\{\neg\text{was}(a_{2,1}), \neg\text{Heisenberg}(a_{2,2}), \neg\text{physicist}(a_{2,3}), \neg\text{a}(a_{2,4}), \neg\text{celebrated}(a_{2,5}),$$
$$\neg\text{German}(a_{2,6}), \neg\text{and}(a_{2,7}), \neg\text{laureate}(a_{2,8}), \neg\text{Nobel}(a_{2,9}),$$
$$\neg R(a_{2,1}, a_{2,2}), \neg R(a_{2,1}, a_{2,3}), \neg R(a_{2,3}, a_{2,4}), \neg R(a_{2,3}, a_{2,5}), \neg R(a_{2,3}, a_{2,7}), \neg R(a_{2,3}, a_{2,8}),$$
$$\neg R(a_{2,5}, a_{2,6}), \neg R(a_{2,8}, a_{2,9}), \text{is\_physicist}(a_{2,2})\}$$

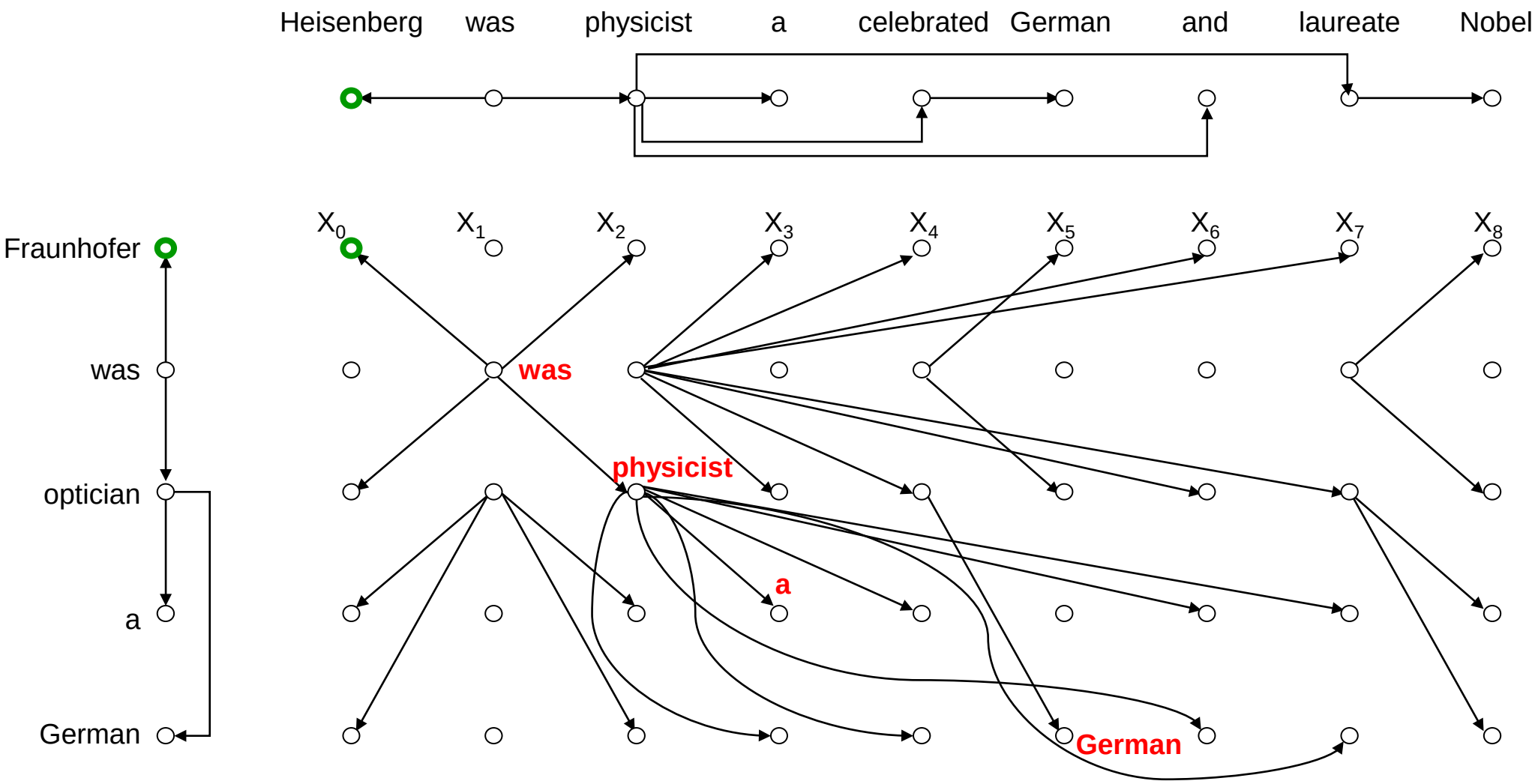
Example – LGG of the Two Ground Clauses

the two structures correspond to **labeled directed trees**

⇨ LGG: **direct product** of labeled trees

- introduce a **product vertex** for each pair *(u,v)* of constants
  - each such product vertex corresponds to a new variable $x_{(u,v)}$ in the LGG
- add an **edge** from (u_1,v_1) to (u_2,v_2) in the product iff there is an edge from u_1 to u_2 and there is an edge from v_1 to v_2
  - add the literal $\neg R(x_{(u_1,v_1)}, x_{(u_2,v_2)})$ to the LGG
- „color“ the product vertex *(u,v)* by the unary predicate *Q* iff *u* and *v* are both colored by *Q*
  - add the literal $\neg Q(x_{(u,v)})$ to the LGG

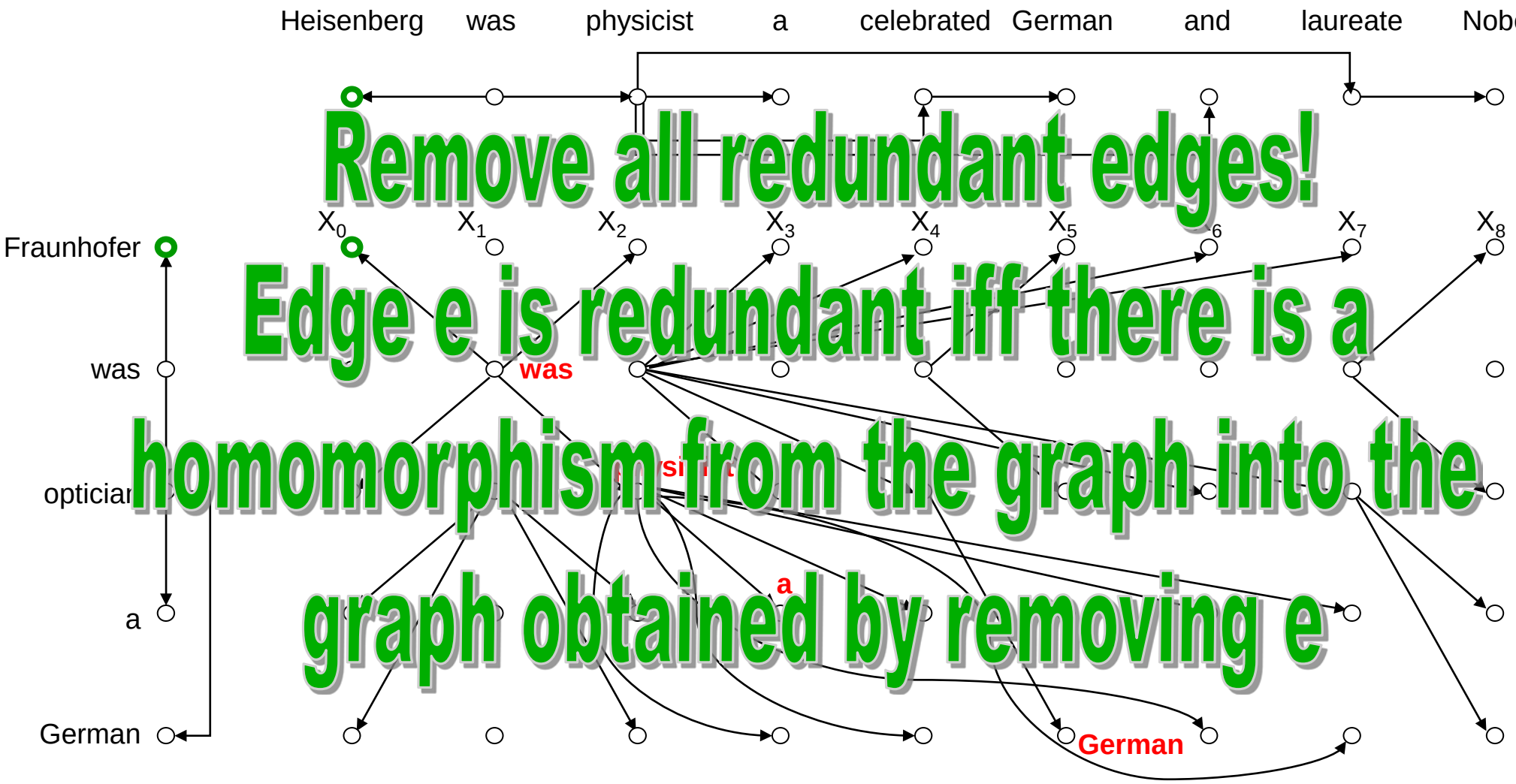
Example – LGG of the Two Ground Clauses



universitätbonn
RHEINISCHE FRIEDRICH-WILHELMS-UNIVERSITÄT

Fraunhofer IAIS

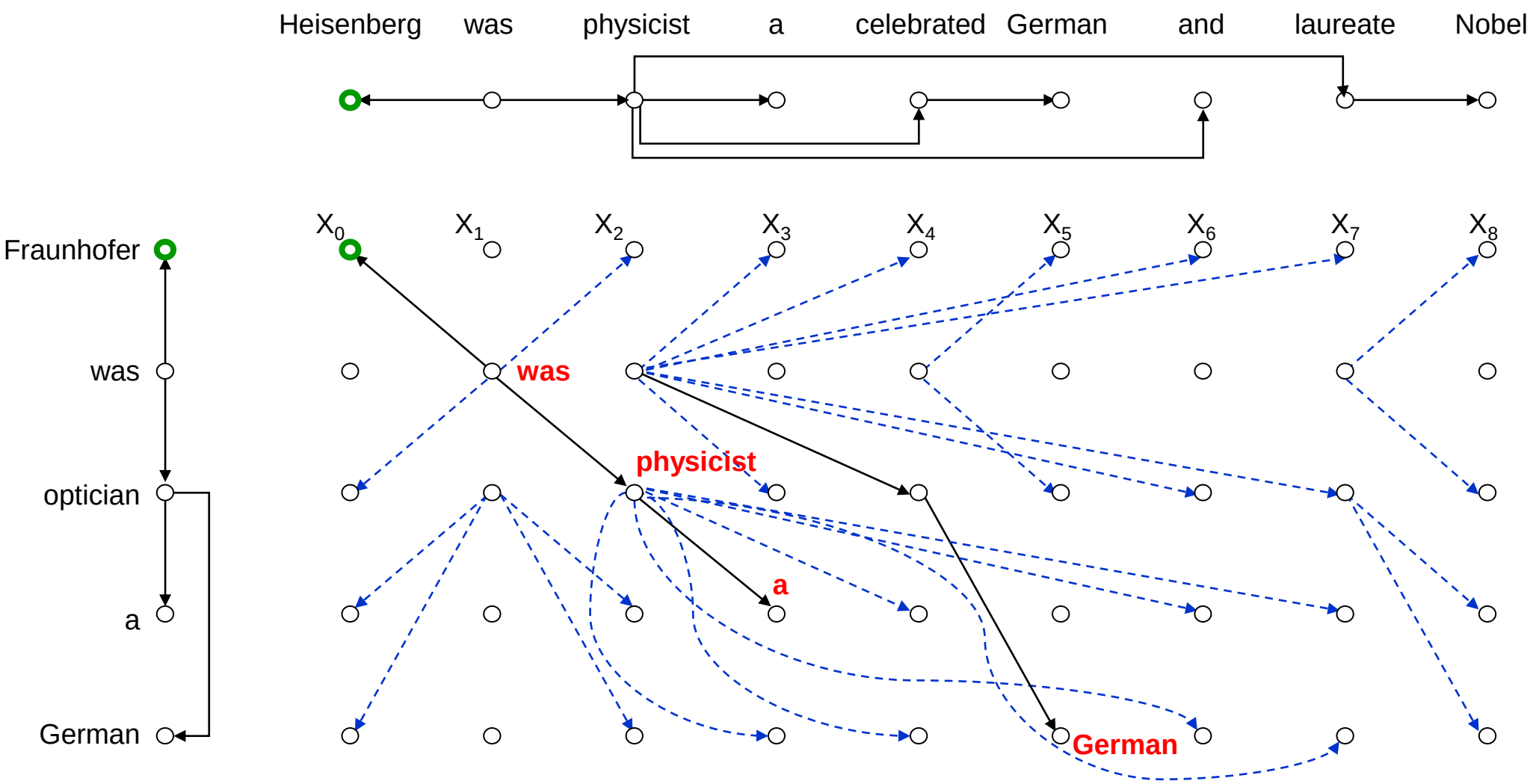
Example – LGG Reduction



universitätbonn
RHEINISCHE FRIEDRICH-WILHELMS-UNIVERSITÄT

Fraunhofer IAIS

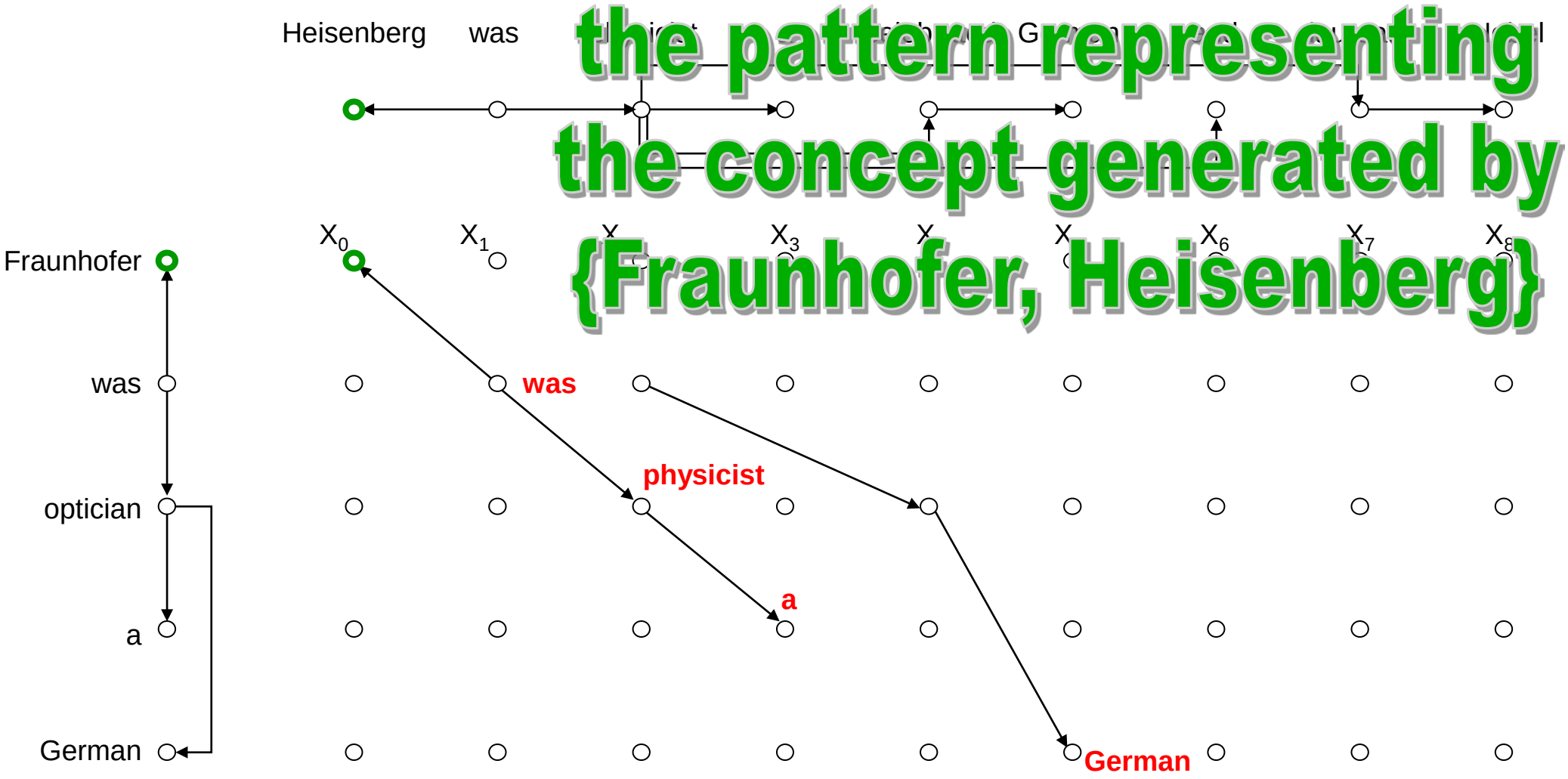
Example – LGG Reduction



universitätbonn
RHEINISCHE FRIEDRICH-WILHELMS-UNIVERSITÄT

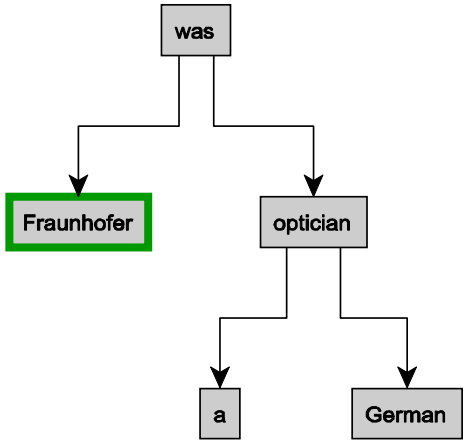
Fraunhofer IAIS

Example – LGG Reduction

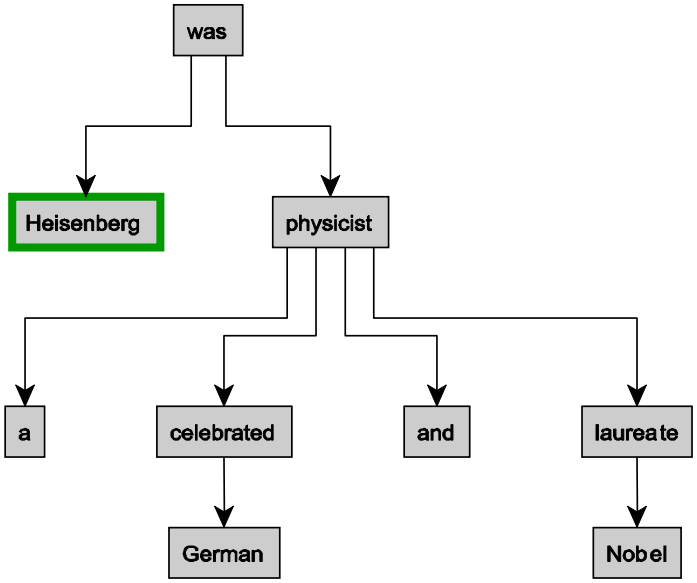


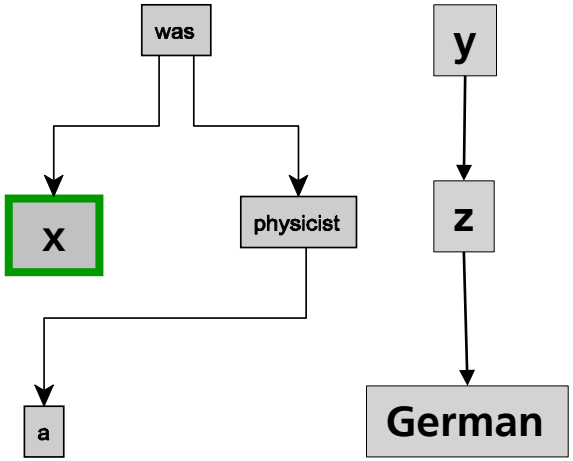
Example – The Reduced LGG as Graph Pattern

Fraunhofer was a German optician.



Heisenberg was a celebrated German physicist and Nobel laureate.





the pattern representing the concept generated by POS

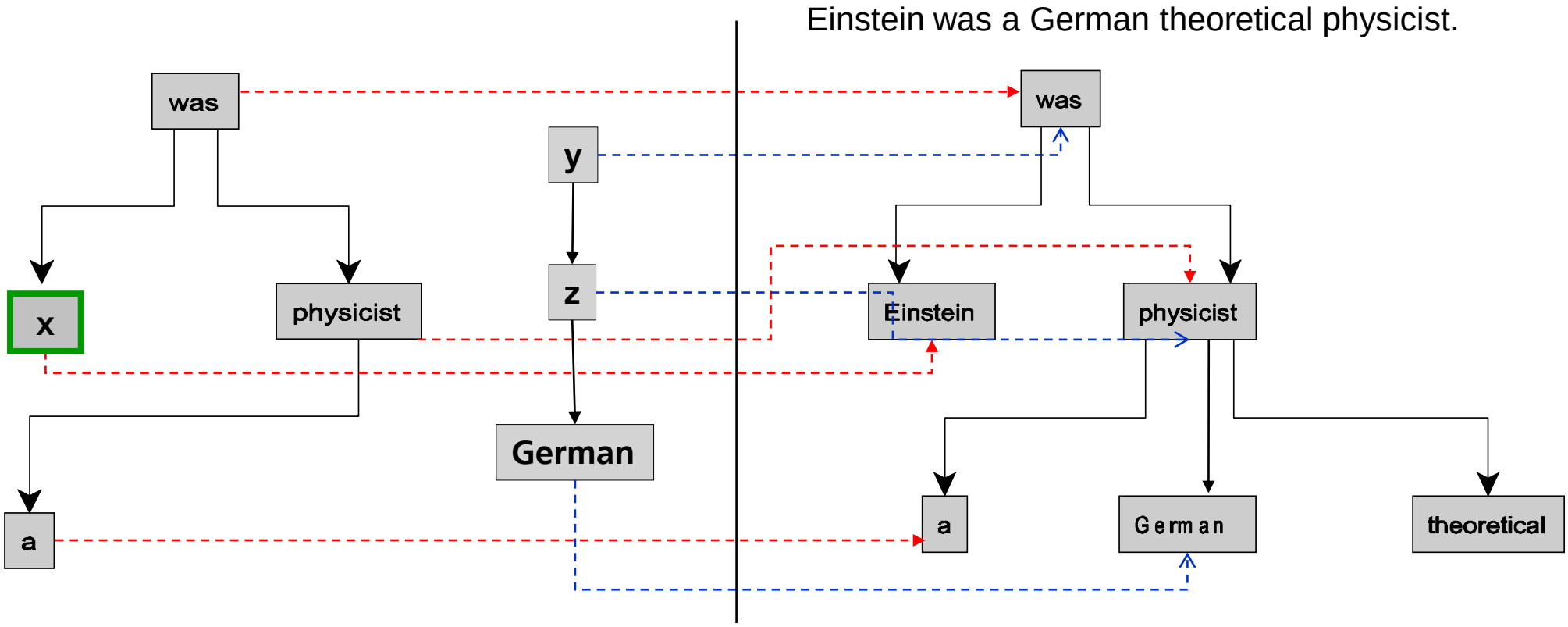
target relation: **Is_Physicist** (unary)

POS: { Is_Physicist(Fraunhofer), Is_Physicist(Heisenberg) }

Example – The Reduced Pattern as First-Order Clause

$$\forall X_0 \forall X_{10} \forall X_{11} \forall X_{20} \forall X_{22} \forall X_{30} \forall X_{41} (\text{Is\_Physicist}(X_0) \wedge$$
$$\neg \text{was}(X_{10}) \wedge \neg \text{physicist}(X_{20}) \wedge \neg \text{a}(X_{30}) \wedge$$
$$\wedge \neg R(X_0, X_{10}) \wedge \neg R(X_{10}, X_{20}) \wedge \neg R(X_{20}, X_{30}) \wedge$$
$$\wedge \neg \text{German}(X_{41}) \wedge \neg R(X_{11}, X_{22}) \wedge \neg R(X_{22}, X_{41}))$$

Example (cont'd)

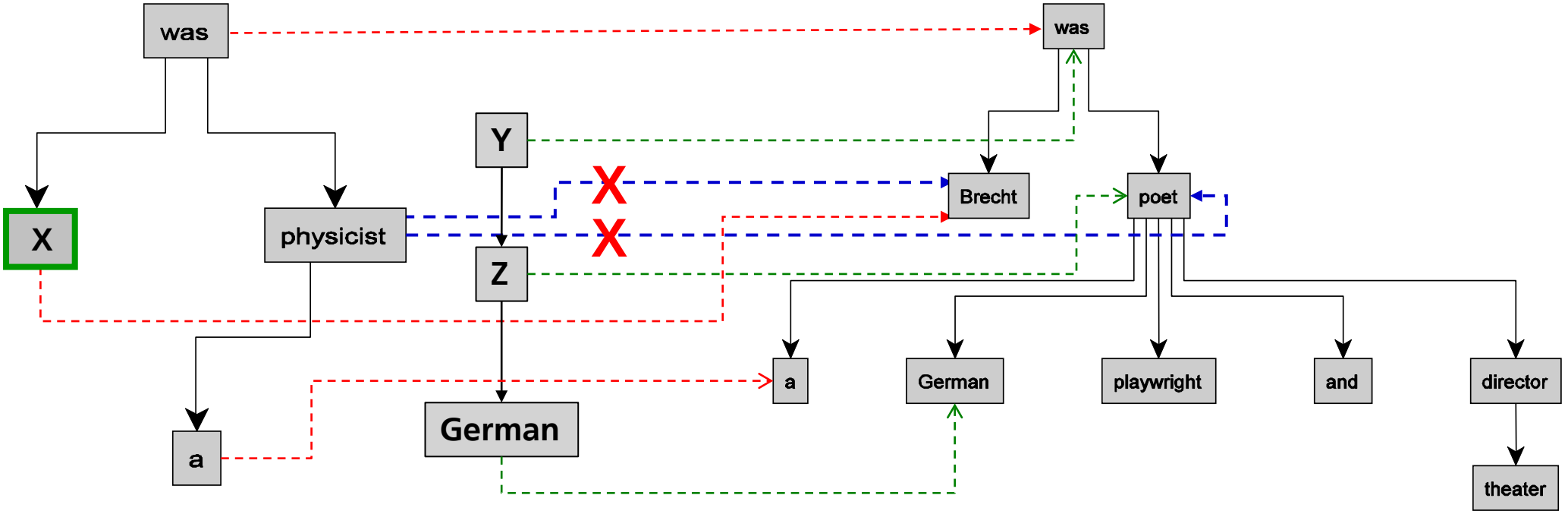


the pattern representing the concept generated by
{Is_Physicist(Fraunhofer), Is_Physicist(Heisenberg)**}**

Is_Physicist(Einstein)

Example (cont'd)

Brecht was a German poet, playwright, and theatre director.



the pattern representing the concept generated by
{Is_Physicist(Fraunhofer), Is_Physicist(Heisenberg)**}**

NOT Is_Physicist(Brecht)

Summary

- LGG is a natural notion for the generalization of first-order clauses with respect to subsumption
- computing the LGG of clauses is reduced to computing the LGG of words
- a reduced non-empty LGG, if it exists, is unique up to variable renaming
- **problems** with the LGG
  - the size of the reduced LGG can grow exponentially with the number of clauses
  - as subsumption is NP-complete, deciding whether the LGG generalizes (i.e., subsumes) a clause is NP-complete

Outline

- complexity of learning function-free definite Horn clauses
- bottom-up induction of clauses
  - the relative least general generalization (RLGG)
  - a generic algorithm
  - on the length of the reduced RLGG
- top-down induction of clauses
  - the FOIL algorithm

Horn Clauses

Def. A **Horn clause** is a clause containing at most one positive literal;
a **definite** Horn clause is a clause containing precisely one positive literal.

- the Horn clause $\{\neg L_1, \ldots, \neg L_k\}$ is denoted by

$$\leftarrow L_1, \ldots, L_k$$

- the definite Horn clause $\{L_0, \neg L_1, \ldots, \neg L_k\}$ is denoted by

$$L_0 \leftarrow L_1, \ldots, L_k$$

database theory: function-free Horn clauses, i.e., terms are either variables or constants

- referred to as **conjunctive queries**
- having no positive literal: **Boolean conjunctive queries**

Finding a Consistent Clause

the consistent hypothesis finding problem:

Given two sets $\{C_1, \ldots, C_k\}$ and $\{D_1, \ldots, D_l\}$ of function-free definite Horn clauses, decide whether there exists a Horn clause H such that

- $H \leq C_i$ for all $i = 1, \ldots, k$ and
- $H \not\leq D_j$ for all $j = 1, \ldots, l$

observations:

- such an H exists *iff* LGG$(\{C_1, \ldots, C_k\}) \not\leq D_j$ for all $j = 1, \ldots, l$
- if H exists and the positive literals in the C_i's are pairwise compatible, then H must be definite and function-free

Finding a Consistent Clause

Theorem (Kietz, 1993)

The consistent hypothesis finding problem is NP-hard.

Proof: *see Appendix of the lecture slides*

remarks

- the problem is a decision problem
  - i.e., we do not want to output H if it exists
- the negative result does not imply that the problem is in NP
  - it is an open problem, whether it belongs to PSPACE

Learning Function-Free Definite Horn Clauses

- complexity of learning function-free definite Horn clauses
- **bottom-up induction of clauses**
  - **the relative least general generalization (RLGG)**
  - **a generic algorithm**
  - **on the length of the reduced RLGG**
- **top-down induction of clauses**
  - **the FOIL algorithm**

Learning Non-Recursive Definite Horn Clauses

vocabulary:

- *target* (or *intensional*) predicate P
- *background* (or *extensional*) predicates $R_1, \ldots, R_l$ with $P \notin \{R_1, \ldots, R_l\}$
- constants $a_1, \ldots, a_n$

non-recursive definite Horn clause (*conjunctive query*):

$$\underbrace{P(\cdots)}_{\textbf{head}} \leftarrow \quad \underbrace{R_{i_1}(\cdots), \ldots, R_{i_r}(\cdots)}_{\textbf{body (background atoms)}}$$

- *function-free* (i.e., arguments are variables or constants)

background knowledge $\mathcal{B}$:
set of *ground* atoms of the background predicates (*extensional database*)

Learning Non-Recursive Definite Horn Clauses

- let A be a ground P-atom and C be a non-recursive definite Horn clause

Def.: C *implies* A *wrt.* $\mathcal{B}$ if there is a substitution θ such that

$$\text{head}(C)\theta = A \text{ and } B\theta \in \mathcal{B} \text{ for every literal } B \in \text{body}(C)$$

- $C \cup \{\mathcal{B}\}$ is a **logic program**
- this case: subsumption $\equiv$ implication (Gottlob, 87)

Example

- $C = P(X, Y) \leftarrow R(X, Z), R(Z, Y)$
- $A = P(a, c)$
- $\mathcal{B} = \{R(a, b), R(b, c), R(c, a)\}$
- C implies A with respect to $\mathcal{B}$
  - for the substitution $\theta = \{X/a, Y/c, Z/b\}$

Finding Consistent Clauses wrt. Background Knowledge

Given background knowledge $\mathcal{B}$ and disjoint sets E^+ and E^- of ground P-atoms,

find a non-recursive definite Horn clause C such that

(i) C implies all positive examples with respect to $\mathcal{B}$

(ii) C does not imply any of the negative examples with respect to $\mathcal{B}$,

if such a clause exists, and **print 'no'** otherwise.

- clauses satisfying (i) and (ii) are **consitent** with E^+ and E^- with respect to $\mathcal{B}$

Relative Least General Generalization

Prop.: For background knowledge $\mathcal{B}$ and disjoint sets E^+ and E^- of ground P-atoms, there exists a non-recursive definite Horn clause C *consistent* with E^+ and E^- with respect to $\mathcal{B}$ if and only if

$$\mathrm{LGG}(\{A \leftarrow \mathcal{B} : A \in E^+\})$$

does not imply any of the negative examples with respect to $\mathcal{B}$.

Proof: *exercise*

Def.: $\mathrm{LGG}(\{A \leftarrow \mathcal{B} : A \in E^+\})$ is the *relative least general generalization* of E^+ wrt. $\mathcal{B}$

– **notation:** $\mathrm{RLGG}_{\mathcal{B}}(E^+) = \mathrm{LGG}(\{A \leftarrow \mathcal{B} : A \in E^+\})$

Problem Reformulation: Notions

instance space X:

set of all ground P-atoms

concept: $C_{\mathcal{B}}$ represented by a non-recursive definite Horn clause C wrt. $\mathcal{B}$

$C_{\mathcal{B}} = \{A \in X : \ C \text{ implies } A \text{ wrt. } \mathcal{B}\}$

$\Rightarrow C_{\mathcal{B}} \subseteq X$

concept class $\mathcal{C}_{\mathcal{B}}$:

set of all such concepts

$\Rightarrow \mathcal{C}_{\mathcal{B}} \subseteq 2^X$

Reformulation of the Problem on Slide 87

Given background knowledge $\mathcal{B}$ and disjoint sets E^+ and E^- of ground P-atoms, (i.e., $E^+, E^- \subseteq X$),

find a non-recursive definite Horn clause C such that

(i) $E^+ \subseteq C_{\mathcal{B}}$ and

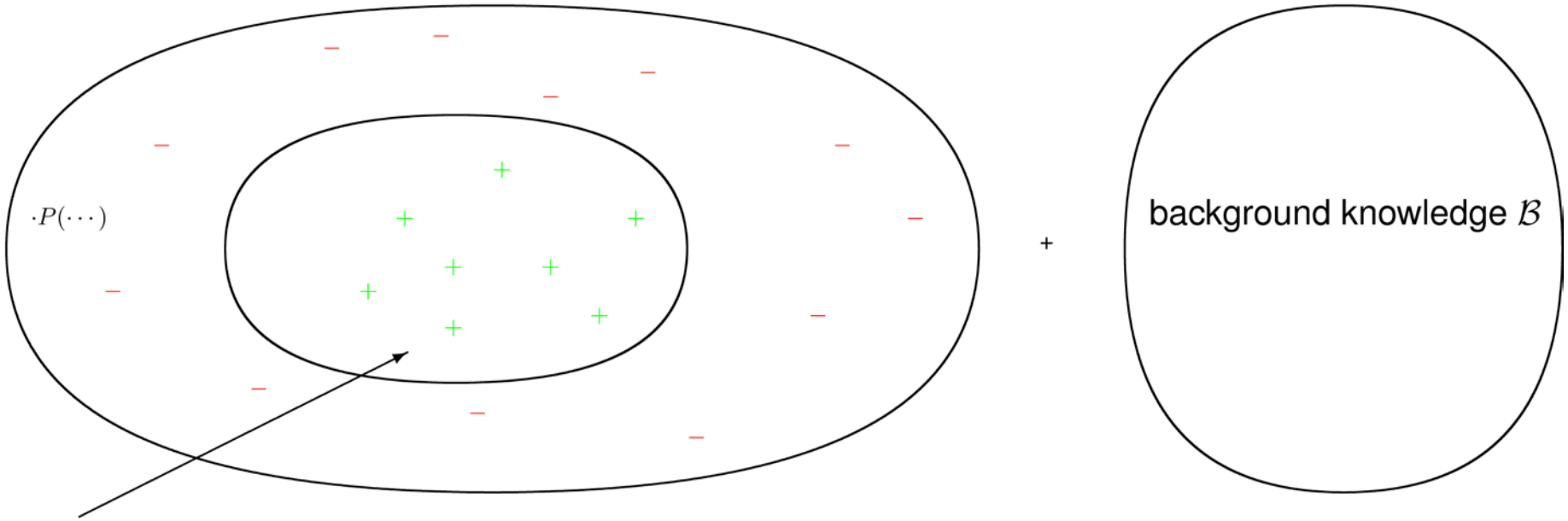
(ii) $E^- \cap C_{\mathcal{B}} = \emptyset$,

if such a clause exists, and **print 'no'** otherwise.

– one of the basic problems of **Inductive Logic Programming** (Muggleton and De Raedt, 94)

Reformulation of the Problem on Slide 87

instance space X $\qquad$ $(+ \in E^+, - \in E^-)$



consistent concept c (it has a non-recursive definite Horn clause representation)

- **goal**: PRINT such a clause representing c if it exists wrt. $\mathcal{B}$

Reformulation of the Problem on Slide 87

Thm.: $\mathcal{C}_{\mathcal{B}}$ is *closed* under nonempty intersection

Proof: *omitted*

$\Rightarrow$ if $c_1, \ldots, c_k \in \mathcal{C}_{\mathcal{B}}$ and $c_1 \cap \ldots \cap c_k \neq \emptyset$ then there is a non-recursive definite Horn clause C such that $c_1 \cap \ldots \cap c_k = C_{\mathcal{B}}$

$\Rightarrow$ the *intersection* of all concepts containing E^+ (**the concept generated** by E^+ wrt. $\mathcal{B}$) is also a concept in $\mathcal{C}_{\mathcal{B}}$

$\Rightarrow$ a non-recursive definite Horn clause consistent with E^+ and E^- wrt. $\mathcal{B}$ exists if and only if E^- and the concept generated by E^+ wrt. $\mathcal{B}$ are disjoint

Reformulation of the Problem on Slide 87

Thm.: The concept generated by E^+ wrt. $\mathcal{B}$ is equal to the set of ground P-atoms implied by $\text{RLGG}_{\mathcal{B}}(E^+)$ wrt. $\mathcal{B}$

Proof: *omitted*

Example

vocabulary: target predicate $P/2$, background predicate $R/2$, constants a, b, c

background knowledge: $\mathcal{B} = \{R(a,b), R(b,a), R(c,b)\}$

positive examples: $E^+ = \{P(a,b), P(b,c)\}$

negative examples: $E^- = \{P(b,a)\}$

$\Rightarrow$ *previous theorem:* a consistent clause exists if and only if E^- is disjoint with the set of ground P-atoms implied by $\text{RLGG}_{\mathcal{B}}(E^+)$ wrt. $\mathcal{B}$

Example

$$\text{RLGG}_{\mathcal{B}}(E^+)$$

$$= \text{LGG}(P(a,b) \leftarrow R(a,b), R(b,a), R(c,b), P(b,c) \leftarrow R(a,b), R(b,a), R(c,b))$$

$$= P(X,Y) \leftarrow R(Y,X), R(X,Z), R(Z,X), \underbrace{R(a,b), R(b,a), R(c,b)}_{\text{background knowledge } \mathcal{B}}$$

- $\mathcal{B}$ can be removed, as the clause is evaluated wrt. $\mathcal{B}$

$$\Rightarrow C : P(X,Y) \leftarrow R(Y,X), R(X,Z), R(Z,X)$$

Example

- it follows from the theory that

$$C : P(X, Y) \leftarrow R(Y, X), R(X, Z), R(Z, X)$$

  implies both positive examples $P(a, b)$ and $P(b, c)$ wrt.

$$\mathcal{B} = \{R(a, b), R(b, a), R(c, b)\}$$

- C also implies the negative example $P(b, a)$ wrt. $\mathcal{B}$

$\Rightarrow$ there is no consistent non-recursive definite Horn clause

Bottom-Up Induction of First-Order Clauses

Input: $\mathcal{B}$, E^+, E^-
Output: a set H of non-recursive definite Horn clauses such that all positive examples is implied by at least one clause in H wrt. $\mathcal{B}$ and no negative example is implied by one of the clauses in H wrt. $\mathcal{B}$

1: $H = \emptyset$
2: **while** $E^+ \neq \emptyset$ **do**
3: find a maximal subset $E \subseteq E^+$ such that $\text{RLLG}_{\mathcal{B}}(E)$ implies no negative example wrt. $\mathcal{B}$
4: add $\text{RLLG}_{\mathcal{B}}(E)$ to H
5: $E^+ = E^+ \setminus E$
6: **end while**
7: **return** H

Bottom-Up Induction of First-Order Clauses

Prop.: the previous algorithm is correct and always terminates if $E^+ \cap E^- = \emptyset$

Proof: *exercise*

Problem 1: it is NP-complete to decide the condition in Step 3

Problem 2: the algorithm does not necessarily find the smallest number of consistent clauses (i.e., no guarantee for the minimality of $|H|$)

Problem 3: the size (number of literals) of the RLGG computed in step 3 is exponential in $|E|$

Problem 3: The Length of the Reduced RLGG

Thm.: There exists a background knowledge $\mathcal{B}$ with n constants, and a set S of ground unary P-atoms such that the size of any consistent clause is exponential in n.

Proof (sketch):

- let $\mathcal{B}$ be the background knowledge over a single binary background predicate R with n constants, consisting of directed cycles $K_1, \ldots, K_l$ of different prime lengths $2, 3, \ldots, p_l$, where l is as large as possible
  - the remaining vertices are isolated
- select constants $a_1, \ldots, a_l$, one from each cycle, and let
  - $E^+ = \{P(a_1), \ldots, P(a_{l-1})\}$ and
  - $E^- = \{P(a_l)\}$

Problem 3: The Length of the Reduced RLGG

Thm.: There exists a background knowledge $\mathcal{B}$ with n constants, and a set S of ground unary P-atoms such that the size of any consistent clause is exponential in n.

Proof (sketch):

- let $\mathcal{B}$ be the background knowledge over a single binary background predicate R with n constants, consisting of directed cycles $K_1, \ldots, K_l$ of different prime lengths $2, 3, \ldots, p_l$, where l is as large as possible
  - the remaining vertices are isolated
- select constants $a_1, \ldots, a_l$, one from each cycle, and let
  - $E^+ = \{P(a_1), \ldots, P(a_{l-1})\}$ and
  - $E^- = \{P(a_l)\}$

Problem 3: The Length of the Reduced RLGG

Proof (sketch) cont'd:

- the body of the **reduced** RLGG of E^+ wrt. $\mathcal{B}$ is a directed cycle of length

$$2 \cdot \ldots \cdot p_{l-1}$$

where p_l is the largest prime number with $\sum_{i=1}^{l} p_i \leq n$

- no strictly smaller clause equivalent to the RLGG exists (*why?*)
- using some basic results from number theory, we have that

$$2 \cdot \ldots \cdot p_{l-1} = 2^{\Theta(\sqrt{n \log n})}$$

□

Summary

- **consistent hypothesis finding problem:** computationally intractable
- **bottom-up induction**: using the relative LGG, it iteratively generalizes the current clauses as long as it is consistent with the negative examples
  - system based on this approach: **Golem** [Muggleton and Feng, 1993]
  - **problems** with this approach:
    (1) the size of the reduced LGG can grow exponentially with the number of positive examples
    (2) as subsumption is NP-complete, deciding whether the LGG implies an example wrt. to the background knowledge is NP-complete

Outline

- complexity of learning function-free definite Horn clauses
- bottom-up induction of clauses
  - the relative least general generalization (RLGG)
  - a generic algorithm
  - on the length of the reduced RLGG
- top-down induction of clauses
  - the FOIL algorithm

FOIL: First-Order Inductive Learner

- Quinlan (1990-1993)
- combines the **divide-and-conquer method** designed for propositional TDIDT (top-down induction of decision trees) systems with the **covering method** developed for disjunctive logical expressions
  - **information-based heuristics** in the divide-and-conquer method
- hypothesis space is **searched top-down** in a heuristic fashion, looking for **maximally general rules** consistent with the negative examples

☺ usually **fast** running times, no parameters, easy to use

☹ **may miss** good solutions

- **implementations:**
  FOIL 6 (Quinlan; publicly available), mFoil (Dzeroski), Grendel (Cohen)

1. The Divide-and-Conquer Method

- Hunt et al. (1966), Quinlan (1979,1986), Breiman et al. (1984) Cestnik et al. (1987)
- the method below yields a decision tree

1. **if** all training objects belong to a single class, the tree is a leaf labelled with that class
2. **otherwise**
    1. select a test based on one attribute
    2. divide the training set into subsets, each corresponding to one of the possible (mutually exclusive) outcomes of the test, and
    3. apply the same procedure to each subset

2. The Covering Method

- Michalski (1989), Michalski et al. (1986)
- target class is represented by a **disjunctive logical expression**

1. find a **conjunction** of conditions that is satisfied by some objects in the target class, but no objects from another class
2. append this conjunction as one **disjunct** of the logical expression being developed
3. **remove** all objects that satisfy this conjunction and, if there are still some remaining objects of the target class, **repeat** this procedure

FOIL: The Outer Loop

Input : disjoint sets E^+ and E^- of ground P-atoms (P is the target predicate) and background knowledge $\mathcal{B}$ (set of ground background atoms)
Output: set of non-recursive definite Horn clauses consistent with E^+ and E^- wrt. $\mathcal{B}$

1: $H = \emptyset$
2: **while** $E^+ \neq \emptyset$ **do**
3: find a maximally general Horn clause C covering part of E^+ and no element of E^- wrt. $\mathcal{B}$
4: $E^+ = E^+ \setminus \{A : A \text{ is implied by } C \text{ wrt. } \mathcal{B}\}$
5: add C to H
6: **return** H

FOIL: How to Perform Step 3? (The Inner Loop of FOIL)

Input : disjoint sets E^+ and E^- of ground P-atoms (P is the target predicate) and background knowledge $\mathcal{B}$ (set of ground background atoms)
Output: maximally general non-recursive definite Horn clause C consistent with E^- wrt. $\mathcal{B}$

1: $C = P(X_1, \ldots, X_k) \leftarrow$ // P is the target predicate of arity k
2: $T_1^{\oplus} = E^+, T_1^{\ominus} = E^-$
3: $i = 1$
4: **while** $T_i^{\ominus} \neq \emptyset$ **do**
5: find a literal L_i to add the right-hand side of C // specialization of C
6: produce new training set $T_{i+1}^{\oplus}$ (resp. $T_{i+1}^{\ominus}$) based on those tuples in $T_i^{\oplus}$ (resp. $T_i^{\ominus}$) that satisfy L_i; if L_i introduces new variables, each such tuple may give rise to several tuples in $T_{i+1}^{\oplus}$ (resp. $T_{i+1}^{\ominus}$) // see Slide 110
7: $i = i + 1$
8: **return** C

Inner Loop of FOIL: Which Literals Are Considered in Step 5?

- let $C = P(X_1, \ldots, X_k) \leftarrow L_1, \ldots, L_{i-1}$
- then L_i is a literal satisfying
  - L_i must contain at least one variable appearing in C
  - L_i has either a background predicate or is of the form $X_j = X_k$ or $X_j \neq X_k$, where X_j and X_k are both existing variables in C

remarks:

- FOIL: *negated* literals can also be added to the body
- FOIL allows recursive clauses as well (i.e., literals of the target predicate P can also be added to the clause)
- we omit the discussion of these more general features; for more details, see Quinlan's original paper

J.R. Quinlan: Learning Logical Definitions from Relations. *Machine Learning*, 5, 239-266, 1990.

Inner Loop of FOIL: Which Literal Is Selected in Steps 5-6?

suppose $C = L_0 \leftarrow L_1, \ldots, L_{i-1}$ has k_i variables $X_1, \ldots, X_{k_i}$ and L_i has l_i new variables $Y_1, \ldots, Y_{l_i}$ (i.e., which do not appear in C)

property: $T_i^{\oplus}$ and $T_i^{\ominus}$ consist of k_i-tuples of constants

- a tuple $(c_1, \ldots, c_{k_i})$ is in $T_i^{\oplus}$ (resp. in $T_i^{\ominus}$) iff there is a substitution $\theta = \{X_1/c_1, \ldots, X_{k_i}/c_{k_i}\}$ satisfying $L_0\theta \in E^+$ (resp. $L_0\theta \in E^-$) and $\{L_1, \ldots, L_{i-1}\}\theta \subseteq \mathcal{B}$

calculation: $T_{i+1}^{\oplus}$ and $T_{i+1}^{\ominus}$ consist of $(k_i + l_i)$-tuples of constants

- a tuple $(c_1, \ldots, c_{k_i}, c_{k_i+1}, \ldots, c_{k_i+l_i})$ is in $T_{i+1}^{\oplus}$ (resp. in $T_{i+1}^{\ominus}$) iff $(c_1, \ldots, c_{k_i})$ is in $T_i^{\oplus}$ (resp. in $T_i^{\ominus}$) and

$$L_i\{X_1/c_1, \ldots, X_{k_i}/c_{k_i}, Y_1/c_{k_i+1}, \ldots, Y_{k_i+l_i}/c_{k_i+l_i}\} \in \mathcal{B}$$

Inner Loop of FOIL: Which Literal Is Selected in Step 5-6?

notations:

- $n_i^{\oplus} = |T_i^{\oplus}|$
- $n_i^{\ominus} = |T_i^{\ominus}|$
- $n_{i+1}^{\oplus} = |T_{i+1}^{\oplus}|$
- $n_{i+1}^{\ominus} = |T_{i+1}^{\ominus}|$
- $n_i^{\oplus\oplus} = |\{(c_1, \ldots, c_{k_i}) \in T_i^{\oplus} : \exists (c_1, \ldots, c_{k_i}, c_{k_i+1}, \ldots, c_{k_i+l_i}) \in T_{i+1}^{\oplus}\}|$

remarks:

- if $l_i = 0$ (i.e., L_i has no new variable) then $n_{i+1}^{\oplus} \leq n_i^{\oplus}$ and $n_{i+1}^{\ominus} \leq n_i^{\ominus}$
- if $l_i > 0$ it can happen that $n_{i+1}^{\oplus} > n_i^{\oplus}$ and/or $n_{i+1}^{\ominus} > n_i^{\ominus}$

Inner Loop of FOIL: Which Literal Is Selected in Step 5-6?

- given $T_i^{\oplus} \cup T_i^{\ominus}$, we need $I(C_i) = -\log_2\left(\frac{n_i^{\oplus}}{n_i^{\oplus}+n_i^{\ominus}}\right)$ bits to signal that a tuple is positive

- if we add L_i to C_i, for $T_{i+1}^{\oplus} \cup T_{i+1}^{\ominus}$, we need $I(C_{i+1}) = -\log_2\left(\frac{n_{i+1}^{\oplus}}{n_{i+1}^{\oplus}+n_{i+1}^{\ominus}}\right)$ bits to signal that a tuple is positive

- **weighted information gain:**

$$\mathrm{WIG}(C_{i+1}, C_i) = n_i^{\oplus\oplus} \times (I(C_i) - I(C_{i+1}))$$

- **Step 5:** Select the literal with the highest weighted information gain!

FOIL: Example

vocabulary:

target predicate: $d/2$ (for daughter)

background predicates: $p/2$ (for parent), $f/1$ (for female)

constants: m (for mary), a (for ann), e (for eve), t (for tom), i (for ian)

background knowledge: $\mathcal{B} = \{p(a,m), p(a,t), p(t,e), p(t,i), f(a), f(m), f(e)\}$

positive examples: $E^+ = \{d(m,a), d(e,t)\}$

negative examples: $E^- = \{d(t,a), d(e,a)\}$

Learn a set of rules consistent with the examples wrt. $\mathcal{B}$!

FOIL: Example

- $C = C_1\colon\ d(X_1, X_2) \leftarrow$

$\Rightarrow\ T_1^{\oplus} = E^{+} = \{d(m, a), d(e, t)\}$

$\Rightarrow\ n_1^{\oplus} = 2$

$\Rightarrow\ T_1^{\ominus} = E^{-} = \{d(t, a), d(e, a)\}$

$\Rightarrow\ n_1^{\ominus} = 2$

$\Rightarrow\ I(C_1) = -\log_2(2/4) = 1$

$\Rightarrow$ literals for L_1: $p(X_1, X_1), p(X_2, X_2), p(X_1, X_2), p(X_2, X_1), p(X_1, Y_1),$
$p(Y_1, X_1), p(X_2, Y_1), p(Y_1, X_2), f(X_1), f(X_2), X_1 = X_2, X_1 \neq X_2$

FOIL: Example

- consider e.g. $L_1 = p(X_2, Y_1)$, i.e., $C_2 : d(X_1, X_2) \leftarrow p(X_2, Y_1)$

$\Rightarrow T_2^{\oplus} = \{(m, a, m), (m, a, t), (e, t, e), (e, t, i)\}$

$\Rightarrow n_2^{\oplus} = 4$

$\Rightarrow T_2^{\ominus} = \{(t, a, m), (t, a, t), (e, a, m), (e, a, t)\}$

$\Rightarrow n_2^{\ominus} = 4$

$\Rightarrow I(C_2) = -\log_2(4/8) = 1$

$\Rightarrow n_1^{\oplus\oplus} = 2$

$\Rightarrow \boxed{\mathsf{WIG}(C_2, C_1) = 2 \cdot (1 - 1) = 0}$

FOIL: Example

- literal maximizing WIG: $L_1 = f(X_1)$, i.e., $C_2 : d(X_1, X_2) \leftarrow f(X_1)$

$\Rightarrow\ T_2^{\oplus} = \{(m, a), (e, t)\}$

$\Rightarrow\ n_2^{\oplus} = 2$

$\Rightarrow\ T_2^{\ominus} = \{(e, a)\}$

$\Rightarrow\ n_2^{\ominus} = 1$

$\Rightarrow\ I(C_2) = -\log_2(2/3) = 0.58$

$\Rightarrow\ n_1^{\oplus\oplus} = 2$

$\Rightarrow\ \boxed{\mathrm{WIG}(C_2, C_1) = 2 \cdot (1 - 0.58) = 0.84}$

FOIL: Next Iteration of the Inner Loop

- $C_2 : d(X_1, X_2) \leftarrow f(X_1)$
- literal maximizing WIG: $L_2 = p(X_2, X_1)$, i.e., $C_3 : d(X_1, X_2) \leftarrow f(X_1), p(X_2, X_1)$

$\Rightarrow T_3^{\oplus} = \{(m, a), (e, t)\}$

$\Rightarrow n_3^{\oplus} = 2$

$\Rightarrow T_3^{\ominus} = \emptyset$

$\Rightarrow n_2^{\ominus} = 0$

$\Rightarrow I(C_3) = -\log_2(2/2) = 0$

$\Rightarrow n_1^{\oplus\oplus} = 2$

$\Rightarrow \boxed{\text{WIG}(C_3, C_2) = 2 \cdot (0.58 - 0) = 1.16}$

FOIL: Next Iteration of the Inner Loop

- since $T_3^{\ominus} = \emptyset$, we finish the inner loop and **return**

$$C : d(X_1, X_2) \leftarrow f(X_1), p(X_2, X_1)$$

- since this clause covers all positve examples, we finish the outer loop as well and **return** the set of clauses:

$$\{d(X_1, X_2) \leftarrow f(X_1), p(X_2, X_1)\}$$

Summary

- **Bottom-up induction**: using the relative LGG, it iteratively generalizes the current clauses as long as it is consistent with the negative examples
  - system based on this approach: **Golem** [Muggleton and Feng, 1993]
  - **problems** with this approach:
    (1) the size of the reduced LGG can grow exponentially with the number of positive examples
    (2) as subsumption is NP-complete, deciding whether the LGG implies an example wrt. to the background knowledge is NP-complete
- **Top-down induction**: iteratively specializes the current clauses by extending it with a literal as long as it is consistent with the negative examples
  - system based on this approach: **FOIL** [Quinlan, 1990] and its variants
  - **problems** with this approach: same as (2) above
  - the approach can be extended to learning recursive Horn clauses, as well as to allowing negated literals in the clause‘ body

Appendix: Proof of Kietz's Theorem

proof: reduction from the conjunctive normal form satisfiability (CNF-SAT) problem

- $V = \{v_1, \ldots, v_n\}$ is a set of Boolean variables
  - a CNF over V is a conjunction of (propositional) clauses
  - a (propositional) clause is a disjunction of (propositional) literals
  - a (propositional) literal is either a variable or the negation of a variable from V
- **CNF-SAT problem:** given a CNF F over V, decide whether there exists a truth assignment of V that satisfies all clauses in F
  - NP-complete

Reduction Lemma

CNF-SAT instance: pair (V, F), where $V = \{v_1, \ldots, v_n\}$ is a set of Boolean variables and $F = c_1 \wedge \ldots \wedge c_m$ is a CNF over V

Lemma The CNF-SAT instance (V, F) is satisfiable if and only if there exists a Horn clause consistent with the sets E^+ and $E^- = \{D\}$ of function-free definite Horn clauses defined as follows:

- let $P_1, \ldots, P_{2n}$ be unary (i.e., arity 1) predicate symbols
- let $\psi : \{v_1, \ldots, v_n, \neg v_1, \ldots, \neg v_n\} \to \{P_1, \ldots, P_{2n}\}$ be a bijective function
  - $\psi(v_i) = P_i$ and $\psi(\neg v_i) = P_{n+i}$ for all $i = 1, \ldots, n$

Reduction Lemma (cont'd)

- $E^+ = \{C_i : 1 \leq i \leq n\}$ where

$$C_i: \quad T(a_i) \leftarrow R(a_i, b_i), P_1(b_i), \ldots, P_{i-1}(b_i), P_{i+1}(b_i), \ldots, P_{2n}(b_i),$$
$$R(a_i, c_i), P_1(c_i), \ldots, P_{n+i-1}(c_i), P_{n+i+1}(c_i), \ldots, P_{2n}(c_i)$$

- $D = \{T(d)\} \cup \{\neg R(d, e_j), \neg P_i(e_j) : \psi^{-1}(P_i) \notin c_j, 1 \leq i \leq 2n, 1 \leq j \leq m\}$

remarks:

- the size of E^+ is bounded by a polynomial of n (i.e., size of V)
- the size of $E^- = \{D\}$ is bounded by a polynomial of the size of F

$\Rightarrow$ polynomial reduction

Reduction Lemma: Example

- $V = \{v_1, v_2\}$
- $F = v_1 \wedge \neg v_2$
- $E^+ = \{C_1, C_2\}$ with

$$C_1 : T(a_1) \leftarrow R(a_1, b_1), P_2(b_1), P_3(b_1), P_4(b_1), R(a_1, c_1), P_1(c_1), P_2(c_1), P_4(c_1)$$
$$C_2 : T(a_2) \leftarrow R(a_2, b_2), P_1(b_2), P_3(b_2), P_4(b_2), R(a_2, c_2), P_1(c_2), P_2(c_2), P_3(c_2)$$

- $E^- = \{D\}$ with

$$D : T(d) \leftarrow R(d, e_1), P_2(e_1), P_3(e_1), P_4(e_1), R(d, e_2), P_1(e_2), P_2(e_2), P_3(e_2)$$

Proof of the Reduction Lemma: „IF“ Direction

- suppose there exists a Horn clause consistent with the examples

$\Rightarrow$ the LGG H of the positive examples must also be consistent

$\Rightarrow$ H must be of the form

$$\begin{aligned} T(X) \quad \leftarrow \quad & R(X,Y_1), P_1(Y_1), \ldots, P_{2n}(Y_1), \\ & R(X,Y_2), P_1(Y_2), \ldots, P_{2n}(Y_2), \\ & \vdots \\ & R(X,Y_{2^n}), P_1(Y_{2^n}), \ldots, P_{2n}(Y_{2^n}), \end{aligned}$$

where for every $k = 1, \ldots, 2^n$, exactly one of $P_i(Y_k)$ and $P_{n+i}(Y_k)$ is only present in H for all $i = 1, \ldots, n$ and for every $1 \leq k_1 < k_2 \leq 2^n$,

$$(\leftarrow R(X,Y_{k_1}), P_1(Y_{k_1}), \ldots, P_{2n}(Y_{k_1})) \not\succ (\leftarrow R(X,Y_{k_2}), P_1(Y_{k_2}), \ldots, P_{2n}(Y_{k_2}))$$

Proof of the Reduction Lemma: „IF“ Direction

- as H does not subsumes the negative example D, there is a k, $1 \leq k \leq 2^n$, such that the subclause

$$H' = T(X) \leftarrow R(X, Y_k), P_1(Y_k), \ldots, P_{2n}(Y_k)$$

of H does not subsume D

$\Rightarrow$ for every $j = 1, \ldots, m$, there is no substitution $\theta = \{X/d, Y_k/e_j\}$ satisfying $H'\theta \subseteq D$

$\Rightarrow$ for every $j = 1, \ldots, m$, H' contains at least one literal $\neg P_i(Y_k)$ such that $\psi(P_i) \in c_j$

$\Rightarrow$ $\{\psi(P_i) : \neg P_i(Y_k) \in H'\}$ is a partial truth assignment for V satisfying all clauses in F

Proof of the Reduction Lemma: „ONLY IF" Direction

- suppose (V, F) is satisfiable; let $x \in \{0,1\}^n$ be a satisfying assignment of F and L be the set of literals true in x

$\Rightarrow$ either v_i or $\neg v_i$ is in L, but not both

- define the clause H by

$$H = \{T(X)\} \cup \{\neg R(X,Y), \neg P_i(Y) : \psi(P_i) \in L, 1 \leq i \leq 2n\}$$

$\Rightarrow$ $H \leq C$ for every $C \in E^+$, i.e., H is a generalization of the set of positive examples

- we need to show that $H \not\leq D$

Proof of the Reduction Lemma : „ONLY IF“ Direction

- suppose for contradiction that $H \leq D$

$\Rightarrow$ there exists $\theta = \{X/d, Y/e_j\}$ such that

$$H\theta \subseteq \{T(d)\} \cup \{\neg R(d, e_j), \neg P_i(e_j) : \psi(P_i) \notin c_j, 1 \leq i \leq 2n\}$$

$\Rightarrow$ H contains only $\neg P_i(Y)$ such that $\psi(P_i) \in L$ but $\psi(P_i) \notin c_j$

$\Rightarrow$ contradicts the assumption that L satisfies all clauses in F