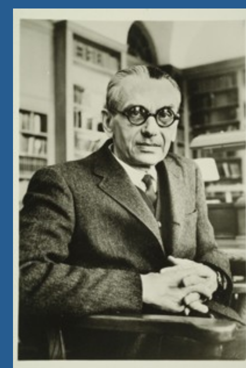
Leibnitz



Godel

FOL Inference

EECS 492
February 17th, 2011

The story so far…

- Propositional Logic
  - Model Checking
  - Forward Chaining
  - Backwards Chaining
  - Resolution
- First-Order Logic
  - Richer language with objects, relations
  - We practiced writing sentences in FOL
  - Reduction of FOL to PL for inference
    - Why was this problematic?

Today

- FOL Inference without reducing to PL
  - Unification – an important building block
  - Forward Chaining
  - Backwards Chaining
  - Resolution

Administrative

- PS3 extension: until Thursday Feb 24
- Midterm Feedback
  - Thanks for your comments!
  - On challenge problems…
  - Have more feedback? I want to hear it!

Midterm 1 Results

Count										1	1	4	3	6	6	15	12	12	11	4
Score	2.0	6.0	10.0	14.0	18.0	22.0	26.0	30.0	34.0	38.0	42.0	46.0	50.0	54.0	58.0	62.0	66.0	70.0	74.0	78.0

Class median: 64.0
Class average: 63.3
Class stddev: 9.2
Num scores: 76

Reasoning within FOL

- In order to reason *within* FOL, we need to be able to deal with Universal Instantiation without expanding for every object.
- Consider:
  - S1. IsKing(x) ^ IsGreedy(x) => IsEvil(x)
  - S2. IsKing(John)
  - S3. IsGreedy(John)
- Do IsKing(John) and IsGreedy(John) match the implication?
  - Can we determine this without plugging in every possible object into S1?

Unification

- Unification: The process of making two FOL sentences equivalent by substituting values for variables.
  - S1. IsKing(x) ^ IsGreedy(x) => IsEvil(x)
  - S2. IsKing(John)
- The first term of S1 and S2 can be unified with:
  - { x/ John}

Unification

8

NextTo(r1,r2) NextTo(A,B)

{ r1 / A, r2 / B }

Unification Examples

9

NT(r1,C) — NT(B,r3)

{ r1 / B, r3 / C }

NT(RoomA,n) — NT(r,Hallway(DoorOf(r)))

{ r / RoomA,
n / Hallway(DoorOf(RoomA)) }

NT(RoomA,r1) — NT(RoomB,r2)

FAIL

Most general unifier

10

NT(r1,B) — NT(r2,r3)

{ r3 / B, r1 / A, r2 / A }

{ r3 / B, r1 / r2 }

Most general unifier

Unification Algorithm (sketch)

- θ =Unify(x, y, θ)
  - Return substitutions that cause x and y to match, given already known substitutions.
- Recursively examine two sentences (x,y)
  - Base case:
    - If both sentences are ground terms, ensure that terms are equal or fail.
  - If one sentence is a variable, unify it with the other expression.
  - If sentences have multiple parts, recursively unify each of the parts.

Unification Algorithm (sketch)

- Unify(King(x), King(John) , { })

  Functions have two parts: function symbol and argument list

- Unify (Unify (King, King), Unify (x, John), { })

  { } { x / John }

- Result: { x / John }

Unify (in detail)

function UNIFY(x, y, θ) **returns** a substitution to make x and y identical
inputs: x, a variable, constant, list, or compound
y, a variable, constant, list, or compound
θ, the substitution built up so far

if θ = failure **then return** failure
else if $x = y$ **then return** θ
else if VARIABLE?(x) **then return** UNIFY-VAR(x, y, θ)
else if VARIABLE?(y) **then return** UNIFY-VAR(y, x, θ)
else if COMPOUND?(x) **and** COMPOUND?(y) **then**
return UNIFY(ARGS[x], ARGS[y], UNIFY(OP[x], OP[y], θ))
else if LIST?(x) **and** LIST?(y) **then**
return UNIFY(REST[x], REST[y], UNIFY(FIRST[x], FIRST[y], θ))
else return failure

function UNIFY-VAR(var, x, θ) **returns** a substitution
inputs: var, a variable
x, any expression
θ, the substitution built up so far

if $\{var/val\} \in \theta$ **then return** UNIFY(val, x, θ)
else if $\{x/val\} \in \theta$ **then return** UNIFY(var, val, θ)
else if OCCUR-CHECK?(var, x) **then return** failure
else return add $\{var/x\}$ to θ

Adapting Modus Ponens to FOL

- PL Modus Ponens

$$\frac{\alpha, \alpha \Rightarrow \beta}{\beta}$$

- FOL Modus Ponens example

King(John), King(x) => Ruler(x)

Ruler(John)

- Given that Subst(θ, a’) = Subst(θ, a)

$$\frac{\alpha', \alpha \Rightarrow \beta}{Subst(\theta, \beta)}$$

Generalized Modus Ponens

- Given that Subst(θ, a') = Subst(θ, a)

$$\frac{\alpha', \alpha \Rightarrow \beta}{Subst(\theta, \beta)}$$

- Given that Subst(θ, a_i') = Subst(θ, a_i) for all i

$$\frac{\alpha_1', \alpha_2', \ldots, \alpha_m', \alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_m \Rightarrow \beta}{Subst(\theta, \beta)}$$

Your turn: Self-Shaver?

- There is a barber who shaves every man in town who does not shave himself, and nobody else.
  - Is this a paradox?
  - Step 1. Translate English to FOL
  - Step 2. Convert FOL to CNF
  - Step 3. Perform unit resolution using unification.
    - If paradox is unresolvable, then you will arrive at a contradiction!

Practical FOL Inference

- Classic FOL language subsets
  - Datalog
  - Prolog
- Inference methods
  - Forward Chaining
  - Backwards Chaining
  - Resolution

Datalog

- First-order definite clauses
  - disjunction of literals, exactly one positive term
  - Why are these definite clauses?
    - King(x) ^ Greedy(x) => Evil(x)
    - King(John)
    - Greedy(y)
- No functions allowed
- These are like _____ clauses in PL
- Inference method?
  - Forward chaining

Forward Chaining

- Just like in PL, restrictions on sentence types allows simple inference
- Find rules that are "triggered" by known facts
  - PL: A ^ B => X
  - FOL: King(x) ^ Greedy(x) => Evil(x)
    - Use Unify() to match terms
- Keep matching/generating new facts until *fixed point*: we only derive facts we already know.

Example knowledge base

- The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.
- Prove: Col. West is a criminal

Forward Chaining

function FOL-FC-ASK(KB, α) **returns** a substitution or *false*

    repeat until *new* is empty
        $new \leftarrow \{\}$
        for each sentence r **in** KB **do**
            $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-APART(r)
            for each θ such that $(p_1 \wedge \ldots \wedge p_n)\theta = (p'_1 \wedge \ldots \wedge p'_n)\theta$
                    for some $p'_1, \ldots, p'_n$ in KB
                $q' \leftarrow$ SUBST(θ, q)
                if q' is not a renaming of a sentence already in KB or *new* **then do**
                    add q' to *new*
                    $\phi \leftarrow$ UNIFY(q', α)
                    if ϕ is not *fail* **then return** ϕ
        add *new* to KB
    return *false*

Example knowledge base contd.

... it is a crime for an American to sell weapons to hostile nations:
S1: American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)

Nono … has some missiles
∃x Owns(Nono,x) ∧ Missile(x):
S2: Owns(Nono,M_1) ^ Missile(M_1)

… all of its missiles were sold to it by Colonel West
S3: Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)

Missiles are weapons:
S4: Missile(x) ⇒ Weapon(x)

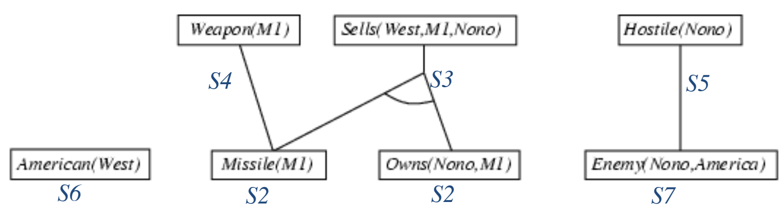
An enemy of America counts as "hostile“:
S5: Enemy(x,America) ⇒ Hostile(x)

West, who is American …
S6: American(West)

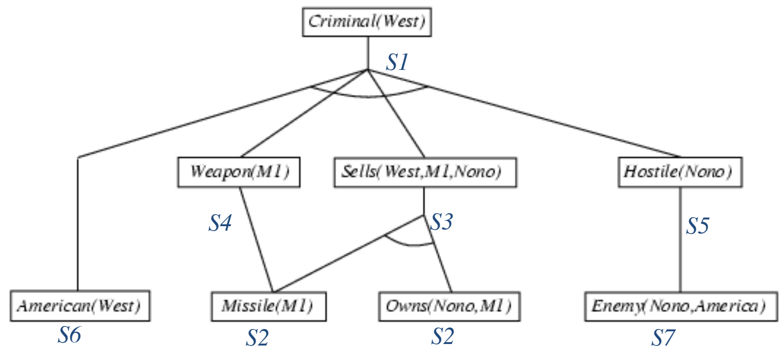
The country Nono, an enemy of America …
S7: Enemy(Nono,America)

Forward chaining proof

Forward chaining proof



Forward chaining proof



Forward Chaining

- Performance:
  - Worst case: only learn one thing per iteration
  - p = # of predicates (“King”)
  - n = # of ground symbols (“John”)
  - k = maximum arity (# of arguments of predicate)

  think: How many facts could we learn from Siblings(x, y)

  - pn^k
- Infinite domains (i.e., if KB includes Peano axioms)?
  - Herbrand’s theorem to the rescue again.

Forward Chaining: Practical Issues

- Described approach spends lots of time trying to match premises of implications.
- Incremental forward chaining: no need to match a rule on iteration *k* if a premise wasn't added on iteration *k-1*
  ⇒ match each rule whose premise contains a newly added positive literal
- Database indexing allows O(1) retrieval of known facts
  - e.g., query *Missile(x)* retrieves $Missile(M_1)$
  - What data structure do we use for the index?
- Suppose we learn Sells(West, M1, Nono). How should we index it?

Backwards Chaining

- Most widely used form of automated reasoning
- Similar to PL version
  - Depth first search
    - What kind of problems do you anticipate?

Backward chaining algorithm

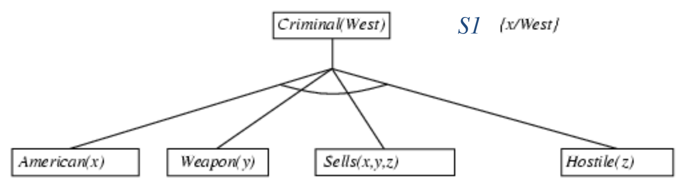
```
function FOL-BC-ASK(KB, goals, θ) returns a set of substitutions
   inputs: KB, a knowledge base
           goals, a list of conjuncts forming a query
           θ, the current substitution, initially the empty substitution { }
   local variables: ans, a set of substitutions, initially empty

   if goals is empty then return {θ}
   q' ← SUBST(θ, FIRST(goals))
   for each r in KB where STANDARDIZE-APART(r) = ( p1 ∧ ... ∧ pn ⇒ q)
           and θ' ← UNIFY(q, q') succeeds
     ans ← FOL-BC-ASK(KB, [p1, ..., pn|REST(goals)], COMPOSE(θ, θ')) ∪ ans
   return ans
```

SUBST(COMPOSE(θ_1, θ_2), p) = SUBST(θ_2, SUBST(θ_1, p))

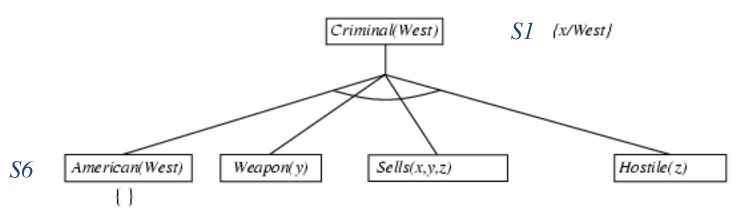
Backward chaining example

Criminal(West)

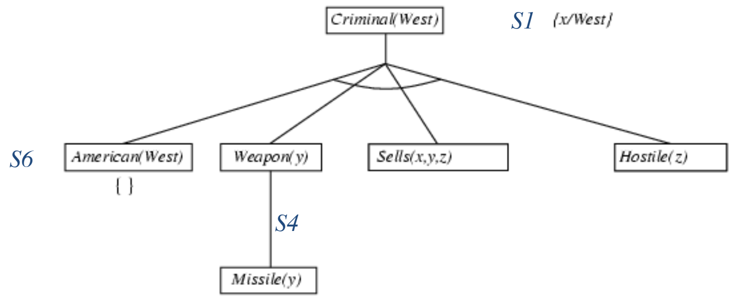
Backward chaining example



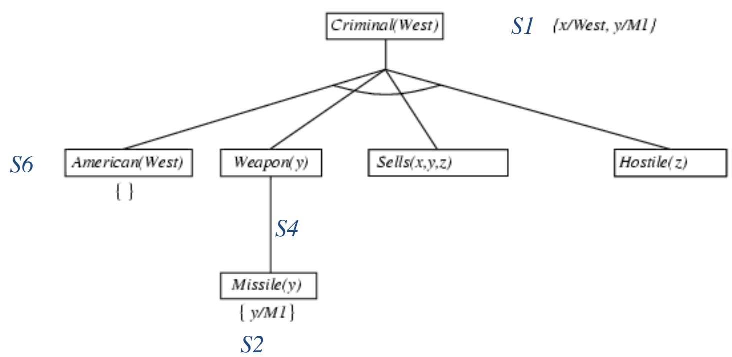
Backward chaining example



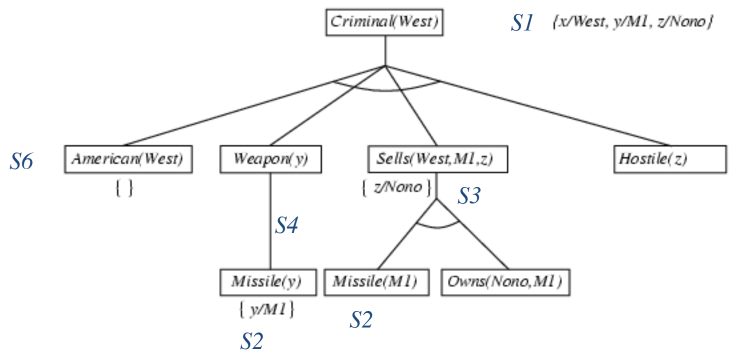
Backward chaining example



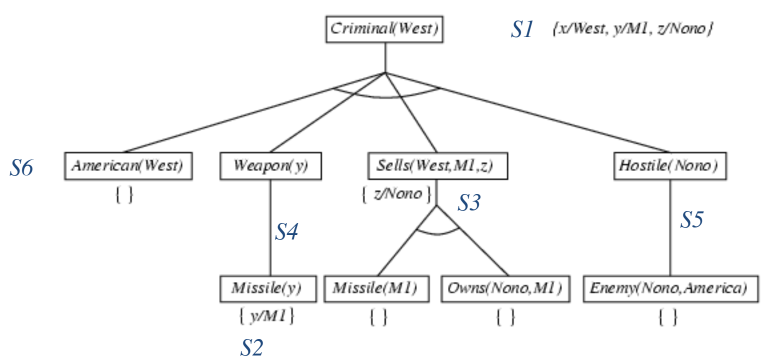
Backward chaining example



Backward chaining example



Backward chaining example



Properties of backward chaining

- Depth-first recursive proof search: space is linear in size of proof
- Incomplete due to infinite loops
  - ⇒ fix by checking current goal against every goal on stack
- Inefficient due to repeated subgoals (both success and failure)
  - ⇒ fix using caching of previous results (extra space)
  - “memoization”
- Widely used for logic programming

Prolog

- Datalog + functions
- Specific, widely-used syntax
  - Variables are UPPERCASE
  - Literals are lowercase.

```
criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).
```

- Depth-first, left-to-right backwards chaining

Prolog: Failure Modes

path(X,Z) :- link(X,Z).
path(X,Z) :- path(X,Y), link(Y,Z)

Consider a path through three nodes:

a --- b --- c

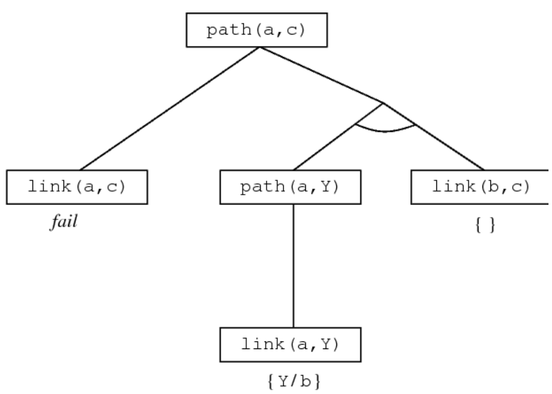
Link(a,b)
Link(b,c)

Query: path(a,c)?

Prolog: Failure Mode

path(X,Z) :- link(X,Z).
path(X,Z) :- path(X,Y), link(Y,Z)

a --- b --- c

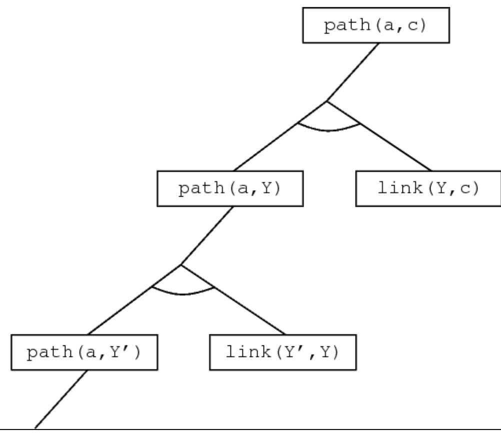


Prolog: Failure Mode

path(X,Z) :- path(X,Y), link(Y,Z) (Order Switched)

path(X,Z) :- link(X,Z).

a --- b --- c



Resolution

- As with PL, we can use Resolution in FOL
- No restrictions on clauses (don't need to be definite clauses)
- Sound
- Complete? (We'll come back to this)

Standardize Variables

43

- Everything is great or something is wrong.

∀x. Great(x) ∨ ∃x. Wrong(x)

We saw that, with unification, all occurrences of a variable need the same binding. And in substitution, we must make the same substitution for all occurrences of a variable.

∀x. Great(x) ∨ ∃y. Wrong(y)

Standardize Variables Example

44

1. A father of someone isn't a woman.
∀x,y. ¬Father(x,y) ∨ ¬Woman(x)
2. A mother of someone is a woman.
∀x,y. ¬Mother(x,y) ∨ Woman(x)
3. Mary is the mother of Chris.
Mother(Mary,Chris)

Resolve 1&2, {x / x}
4. ∀x,y. ¬Father(x,y) ∨ ¬Mother(x,y)
Resolve 4&3, {x / Mary, y / Chris}
5. ¬Father(Mary,Chris)

OK, but why didn't we conclude that Mary wasn't <u>anyone's</u> father?

Standardize Variables Example

45

1. A father of someone isn't a woman.
$\forall$x1,y1. ¬Father(x1,y1) ∨ ¬Woman(x1)
2. A mother of someone is a woman.
$\forall$x2,y2. ¬Mother(x2,y2) ∨ Woman(x2)
3. Mary is the mother of Chris.
Mother(Mary,Chris)

Resolve 1&2, {x1 / x2}
4. $\forall$x2,y1,y2. ¬Father(x2,y1) ∨ ¬Mother(x2,y2)
Resolve 4&3, {x2 / Mary, y2 / Chris}
5. $\forall$y1. ¬Father(Mary,y1)

Better. But to ensure standardized variables, we'd even want:
4. $\forall$x4,y4,z4. ¬Father(x4,y4) ∨ ¬Mother(x4,z4)
5. $\forall$y5. ¬Father(Mary,y5)

Conversion to Conjunctive Normal Form

46

- Translate bidirectionals to implications
- Translate implications to disjunctions.
- Move negations inward
  - Use De Morgan's laws
  - until only atoms are negated
- Standardize variables.
- Eliminate existentials via *skolemization*.
- Drop universal quantifiers.
- Distribute and associate into CNF.

FOL steps

Skolemization (revisited)

47

∃c. IsMother(Jen,c) —*skolem*→ IsMother(Jen,JensKid)

∀c ∃p. IsParent(p,c) —*skolem*→ ∀c IsParent(Pof(c),c)

∀g,c. IsGP(g,c) ⇒ ∃p. IsP(g,p) ∧ IsP(p,c) —*skolem*→

∀g,c. IsGP(g,c) ⇒ IsP(g,Sk3(g,c)) ∧ IsP(Sk3(g,c),c)

Why, in the last example, was the same Skolem function used?
Why does it take both g and c as arguments?

Resolution Example

- Everyone who loves all animals is loved by someone
  ∀ x [∀ y Animal(y) => Loves(x,y)] => ∃ y. Loves(y,x)
- Anyone who kills an animal is loved by no one.
  ∀x. [∃ y. Animal(y) ^ Kills(x,y)] => [∀z. –Loves(z,x)
- Jack loves all animals.
  ∀x. Animal(x) => Loves(Jack, x)
- Either Jack or Curiosity killed the cat, who is named Tuna.
  Kills(Jack, Tuna) v Kills(Curiosity, Tuna)
  Cat(Tuna)
  ∀x Cat(x) => Animal(x) (background knowledge)
- Did Curiosity kill the cat?
  -Kills(Curiosity, Tuna)

Conversion to CNF

- Everyone who loves all animals is loved by someone

  $\forall$ x [$\forall$ y Animal(y) => Loves(x,y)] => $\exists$ y. Loves(y,x)

Resolution Example

A1. Animal(F(x)) v Loves(G(x),x)

A2. –Loves(x,F(x)) v Loves(G(x), x)

B. –Animal(y) v –Kills(x,y) v –Loves(z,x)

C. –Animal(x) v Loves(Jack,x)

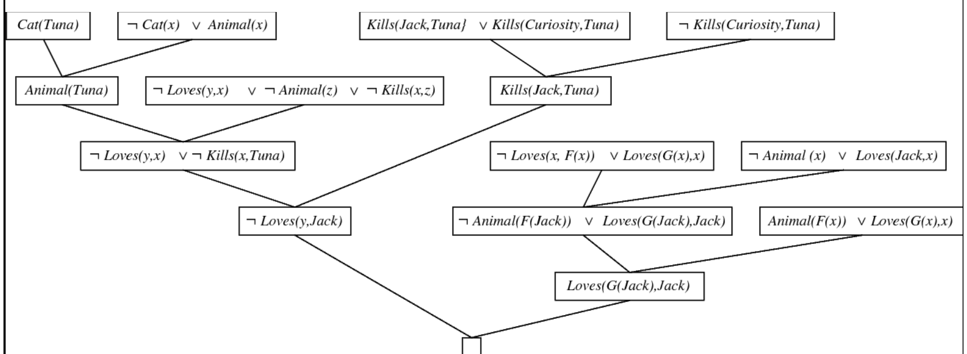
D. Kills(Jack, Tuna) v Kills(Curiosity, Tuna)

E. Cat(Tuna)

F. –Cat(x) v Animal(x)

-G. –Kills(Curiosity, Tuna)

Resolution



Non-constructive Proofs

- Suppose we asked "*Who* killed the cat?"
  - There exists a 'w' such that Kills(w, Tuna)
  - Query: -Kills(w, Tuna)

- -Kills(w,Tuna) , Kills (Jack, Tuna) v Kills(Curiosity, Tuna)
  => Kills(Jack, Tuna) *{ w / Curiosity}*
- Kills(Jack, Tuna), -Kills(w, Tuna)
  => {} *{ w / Jack }*

- Resolution tells us our query is true: There *does* exist a w that killed Tuna. (But what was *w*?)

- Note that the proof assigned multiple values to w: we can detect this and reject those proofs!

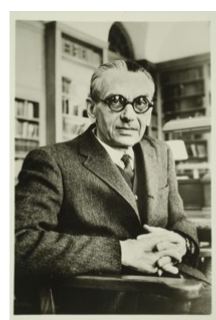
Gottfried Wilhelm Leibniz (1646-1716)… again!

… if we could find characters or signs appropriate for expressing all our thoughts as definitely and as exactly as arithmetic expresses numbers…, we could in all subjects in so far as they are amenable to reasoning accomplish what is done in Arithmetic… For all inquiries… would be performed by the transposition of characters and by a kind of calculus, which would immediately facilitate the discovery of beautiful results…

—*Dissertio de Arte Combinatoria, 1666*

Incompleteness Theorem

- Godel:
  - In any consistent KB involving an inductive schema, there are true sentences that cannot be proved.
  - Arithmetic defined in terms of inductive schema
    - S(0), S(S(0)), S(S(S(0))), …
    - Godel's theorem applies
- Bad news for Leibniz!
  - Can't resolve every argument via inference.
- Practical limitation?
  - Hasn't stopped theorem provers from proving many open problems!



Godel

Next Time

- Done with Logic…
- Planning!