Declarative Programming PROLOG (+ Bayesian Nets)

- Motivation
  - ⋆ Warm Fuzzies
  - ⋆ What is Logic? . . . Logic Programming?
- Mechanics of Prolog
  - ⋆ Terms, Substitution, Unification, Horn Clauses
  - ⋆ Proof (procedure)
  - ⋆ Example: List Processing
- Theoretical Foundations
  - ⋆ Semantics
  - ⋆ Logic / Theorem Proving ... Resolution
- Issues
  - ⋆ Search Strategies
  - ⋆ Declarative/Procedural, . . .
- Other parts of Prolog
  - ⋆ "Impure" Operators — `NOT`, `!`
  - ⋆ Utilities
- Constraint Programming
- Bayesian Belief Nets

What is Logic?

Logic is *formal system for reasoning*

Reasoning is *inferring new facts from old*

Eg: *Given:* $\left\{ \begin{array}{l} \text{All men are mortal.} \\ \text{Socrate is a man.} \end{array} \right\}$,

infer (conclude, reason that, ...)

`Socrates is mortal.`

What is role of Logic within CS?

1. Foundation of discrete mathematics
2. Automatic theorem proving
3. Hardware design/debugging
4. Artificial intelligence (Cmput366)

Components: Syntax (What does it look like?)

Semantics (What does it mean?)

Reasoning/ProofTheory (New facts from old)

Logic Programming

- Program $\equiv$ Logic Formula
- Execution of Program $\equiv$ theorem proving

User: 1. Specifies WHAT is true
2. Asks if something else follows

`Prolog` answers question.

- By comparison,
  using Procedural Programming (C, Pascal, ...):
  User must
  – decide on data-structure
  – explicitly write procedure
  search, match, substitute
  – write diff programs for
  `father(X, tom)` vs `father(tom, Y)`

Logic in general

Logics are formal languages
for representing information
such that conclusions can be drawn

Syntax defines the sentences in the language
. . . what does it look like?

Semantics define "meaning" of sentences;
i.e., define truth of a sentence in a world
How is it linked to the world?

Proof Theory "new facts from old"
find implicit information. . . "pushing symbols"

Eg, wrt arithmetic

$\boxed{x + 2 \geq y}$ is sentence; $\boxed{x2 + y >}$ is not

$x + 2 \geq y$ is true iff
the number $x + 2$ is no less than the number y

$x + 2 \geq y$ is true in a world where $x = 7$, $y = 1$
$x + 2 \geq y$ is false in a world where $x = 0$, $y = 6$

What are Parts of a Logic?

- Syntax: Set of Expressions

  ? Well-formed or not?

  SEQUENCE of Symbols $\mapsto \begin{cases} \text{Accept} \\ \text{Reject} \end{cases}$

  Accept: `The boys are at home.`
  `at(X, home) :- boy(X).`

  Reject: `boys. home the angrily democracy`
  `X(at), x Boy(1X,( ) :-`

- Proof Process:

  Given *Believed* statements,

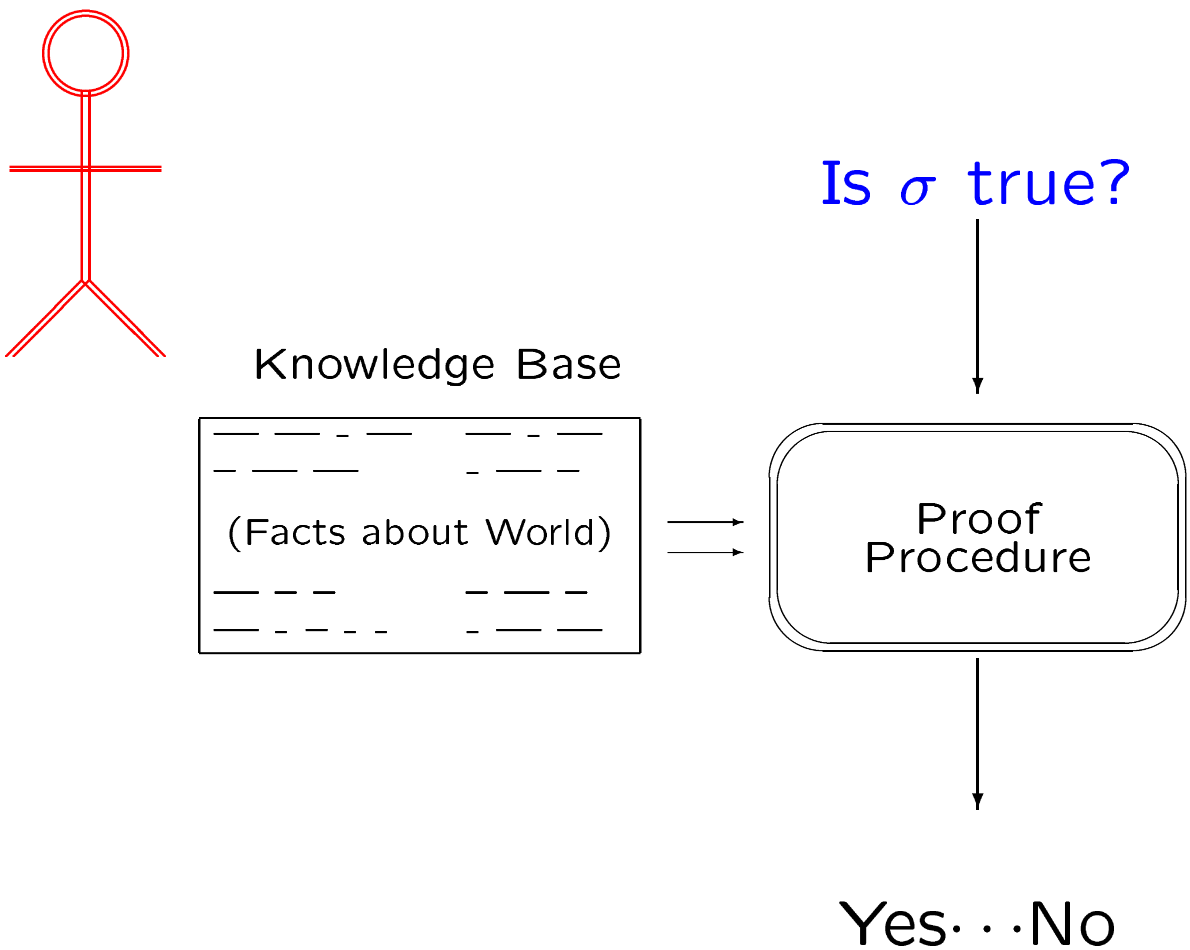
  Determine other *Believed* statements.

$$\{ s_1, \ldots, s_n \} \quad \vdash_P \quad s$$

(Semantics: Which expressions are *Believed*?)

`John's mother is (the individual) Mary.` $\mapsto \mathcal{T}$
`John's mother is (the individual) Fred.` $\mapsto \mathcal{F}$
`Colorless green ideas sleep furiously.` $\mapsto \mathcal{F}$

“Logic Programming” Framework



Concept of PROLOG

PROgramming in **LOG**ic
≈ *Sound Reasoning* Engine

1. User asserts true statements.

   User asserts { `All men are mortal.` `Socrates is a man.` }

2. User poses query.
   A. User asks "Is Socrates mortal?"
   B. User asks "Who/what is mortal?"

3. *Prolog* provides answer (Y/N, binding).
   A. *Prolog* answers "`Yes`".
   B. *Prolog* answers "`Socrates`".

Tying Prolog to Logic

- Syntax: Horn Clauses
  (aka Rules, Facts; Axioms)
  - Terms

- Proof Process: Resolution
  - Substitution
  - Unification

(Semantics
  - Only in that Resolution is Sound)

Proof Process: Backward Chaining

- To prove `X`,
  find FACT `X` in database

  To prove `X`,
  find RULE `Y ⇒ X` in database,
  then prove `Y`.

- Actually...

  To prove `X`,
  find FACT `X'` in database
  (where X'≈X)

  To prove `X`,
  find RULE `Y ⇒ X'` in database,
  (where X'≈X)
  then prove `Y`.

- Need to define. . .
  What `X` is? "Term"
  When `X'≈X` ? "Unification"

Terms

- BNF:

$$
\begin{array}{lcl}
\langle\texttt{term}\rangle & ::= & \langle\texttt{constant}\rangle \mid \langle\texttt{variable}\rangle \\
 & & \mid \langle\texttt{functor}\rangle \\
\langle\texttt{constant}\rangle & ::= & \langle\texttt{atom starting w/lower case}\rangle \\
\langle\texttt{variable}\rangle & ::= & \langle\texttt{atom starting w/upper case}\rangle \\
\langle\texttt{functor}\rangle & ::= & \langle\texttt{constant}\rangle(\langle\texttt{tlist}\rangle) \\
\langle\texttt{tlist}\rangle & ::= & \text{``''} \mid \langle\texttt{term}\rangle \; \{,\langle\texttt{tlist}\rangle\}
\end{array}
$$

- Examples of ⟨term⟩:

a1 b fred	⟨constant⟩
X Yc3 Fred	⟨variable⟩
married(fred) g(a, f(Yc3), b)	⟨functor⟩

- Ground Term ≡ term with *no* variables

`f(q)` `g( f(w), w1(b,c) )` are ground,

`f(A)` `g( f(w), w1(B,c) )` are not.

Substitution

A *Substitution* is a set $\{ \mathtt{v}_1/\mathtt{t}_1\ \mathtt{v}_2/\mathtt{t}_2 \cdots \mathtt{v}_n/\mathtt{t}_n \}$
where v_i are distinct variables
t_i are terms that do not use any of the v_js.

Examples:

+	{ X/a }
	{ X/a Y/b Z/f(a,W) }
	{ X/W Y/f(W) Z/W }
−	{ f(X)/a }
	{ X/a X/b }
	{ X/f(X) }
	{ X/f(Y) Y/g(q) }

Applying a Substitution

- Given $\begin{cases} t & \texttt{— a term} \\ \sigma & \texttt{— a substitution} \end{cases}$

  "$t\sigma$" is the term resulting from applying substitution σ to term t.

- Small Examples:

  `X{X/a} = a`

  `f(X){X/a} = f(a)`

- **Example:** Using $t =$ `f( a, h(Y,b), X )`

  t`{ X/b }` = `f( a, h(Y,b), b )`

  t`{ X/b Y/f(Z) }` = `f( a, h(f(Z),b), b )`

  t`{ X/Z Y/f(Z,a) }` = `f( a, h(f(Z,a),b), Z )`

  t`{ W/Z }` = `f( a, h(Y,b), X )`

- σ need not include all variables in t; σ can include variables not in t.

Composition of Substitutions

- Composition:
  $\sigma \circ \theta$ is *composition* of substitutions σ , θ.
  For any term t, $t[\sigma \circ \theta] = (t\sigma)\theta$.

- Example:

$$\begin{aligned} \texttt{f(X)[}\{\texttt{X/Z}\} \circ \{\texttt{Z/a}\}\texttt{]} &= (\texttt{f(X)}\{\texttt{X/Z}\})\{\texttt{Z/a}\} \\ &= \texttt{f(Z)}\{\texttt{Z/a}\} \\ &= \texttt{f(a)} \end{aligned}$$

- $\sigma \circ \theta$ is a *substitution* (usually)

- Eg:
  $[\{\texttt{X/a}\} \circ \{\texttt{Y/b}\}] = \{\texttt{X/a}, \ \texttt{Y/b}\}$
  $[\{\texttt{X/Z}\} \circ \{\texttt{Z/a}\}] = \{\texttt{X/a}, \ \texttt{Z/a}\}$

Unifiers

- t_1 and t_2 are *unified by* σ iff
  $t_1\sigma = t_2\sigma$.
  Then σ is called a *unifer*
  t_1 and t_2 are *unifiable*

- Examples:

t_1	t_2	unifer	term
f(b,c)	f(b,c)	{}	f(b,c)
f(X,b)	f(a,Y)	{ X/a Y/b }	f(a,b)
f(a,b)	f(c,d)	*	
f(a,b)	f(X,X)	*	
f(X,a)	f(Y,Y)	{ X/a Y/a }	f(a,a)
f(g(U),d)	f(X,U)	{ U/d X/g(d) }	f(g(d),d)
f(X)	f(g(X))	*	
f(X,g(X))	f(Y,Y)	*	
f(X)	f(Y)	{ X/Y }	f(Y)

- NB t_1 and t_2 are symmetrical!
  (Both can have variables.)

Multiple Unifiers

- Unifier for $\mathtt{t}_1 = \mathtt{f(X)}$ and $\mathtt{t}_2 = \mathtt{f(Y)}$

θ	$\mathtt{t}_1\theta = \mathtt{t}_2\theta =$
{ X/Y }	f(Y)
{ Y/X }	f(X)
{ Y/a X/a }	f(a)
{ Y/g(b,Z) X/g(b,Z) }	f(g(b(Z)))
{ X/Y W/f(q,Z) }	f(Y)

- {Y/X} and {X/Y} make sense, but
  { Y/a X/a } has irrelevant constant
  { X/Y W/g } has irrelevant binding (W)

- Adding irrelevant bindings: ∞ unifiers!

? Is there a best one ?

Quest for Best Unifier

- Wish list:
  - No irrelevant constants
    So {Y/X} prefered over { Y/a, X/a }
  - No irrelevant bindings
    So {Y/X} prefered over { Y/X, W/f(4,Z) }

- Spse λ_1 has constant where λ_2 has variable
  (Eg, λ_1 = { X/a, Y/a }, λ_2 = {X/Y})
  Then $\exists$ subsitution μ s.t. $\lambda_2 \circ \mu = \lambda_1$
  (Eg, μ = {Y/a}: {X/Y}$\circ${Y/a} = { X/a, Y/a })

- Spse λ_1 has extra binding over λ_2
  (Eg, λ_1 = {X/a, Y/b}, λ_2 = {X/a})
  Then $\exists$ subsitution μ s.t. $\lambda_2 \circ \mu = \lambda_1$
  (Eg, μ = {Y/b}: {X/a}$\circ${Y/b} = {X/a, Y/b})

- INFERIOR unifier = composition of
  Good Unifer + another substitution

Most General Unifier

- σ is a *mgu* for t_1 and t_2 iff
  - σ unifies t_1 and t_2, and
  - $\forall\ \mu$: unifier of t_1 and t_2,
    $\exists$ subsitution, θ, s.t. $\sigma \circ \theta = \mu$.

  (Ie, for all terms t, $t\mu = (t\sigma)\theta$.)

- Example: $\sigma = \{\, \texttt{X/Y} \,\}$ is mgu for $\texttt{f(X)}$ and $\texttt{f(Y)}$.
  Consider unifier $\mu = \{\, \texttt{X/a} \quad \texttt{Y/a} \,\}$.
  Use substitution $\theta = \{\, \texttt{Y/a} \,\}$:

$$\begin{aligned}
\texttt{f(X)}\mu &= \texttt{f(X)}\{\, \texttt{X/a Y/a} \,\} \\
&= \texttt{f(a)} \\
\texttt{f(X)}[\sigma \circ \theta] &= (\texttt{f(X)}\sigma) \quad \theta \\
&= (\, \texttt{f(X)}\{\, \texttt{X/Y} \,\}\,)\theta \\
&= \texttt{f(Y)}\{\, \texttt{Y/a} \,\} \\
&= \texttt{f(a)}
\end{aligned}$$

  Similarly, $\texttt{f(Y)}\mu = \texttt{f(a)} = \texttt{f(Y)}[\sigma \circ \theta]$

  (μ is NOT a mgu, as $\neg\exists\theta'$ s.t. $\mu \circ \theta' = \sigma$!)

MGU — Example#2

A mgu for

`f(W,g(Z),Z)` & `f(X,Y,h(X))`

is { `X/W` `Y/g(h(W))` `Z/h(W)` }

MGU (con't)

- Notes:
  - If t_1 and t_2 are unifiable, then $\exists$ a mgu.
  - Can be more than 1 mgu

    but they differ only in variable names.
  - Not every unifier is a mgu.
  - A mgu uses constants only as necessary.

- Implementation:

  $\exists$ fast algorithm that computes
  a mgu of t_1 and t_2, if one exists;
  or reports failure.

(Slow part is verifying legal substitution:
none of v_i appear in any t_j.
Avoid by resetting *Prolog*'s `occurscheck` parameter.)

MGU Procedure

```
Recursive Procedure MGU (x,y)
  If x=y then Return ()
  If Variable(x) then Return( MguVar(x,y) )
  If Variable(y) then Return( MguVar(y,x) )
  If Constant(x) or Constant(y) then Return( False )
  If Not(Length(x) = Length(y)) then Return( False )
  g ← []
  For i = 0 ..  Length(x)
      s ← MGU( Part(x,i), Part(y,i) )
      g ← Compose(g,s)
      x ← Substitute(x,g)
      y ← Substitute(y,g)
  Return( g )
End

Procedure MguVar (v,e)
  If Includes(v,e) then Return( False )
    Return( [v/e] )
End
```

Backward Chaining

- Recall

> To prove `X`,
> find FACT `X'` in database
> (where $X' \approx X$)
>
> To prove `X`,
> find RULE `Y ⇒ X'` in database,
> (where $X' \approx X$)
> then prove `Y`.

- Prolog writes `Y ⇒ X'` as `X' :- Y`

  so always unifies `X` against "first part" . . .
  `X'` `X' :- Y`

- Issue: What if rule is `Y1 & Y2 ⇒ X'` ?

Prolog's Syntax

- BNF:

$$\langle \texttt{Horn} \rangle ::= \langle \texttt{literal} \rangle\texttt{.} \quad | \quad \langle \texttt{literal} \rangle \texttt{ :-} \langle \texttt{llist} \rangle\texttt{.}$$

$$\langle \texttt{llist} \rangle ::= \langle \texttt{literal} \rangle \ \{\texttt{,}\ \langle \texttt{llist} \rangle\ \}$$

$$\langle \texttt{literal} \rangle ::= \langle \texttt{term} \rangle$$

- Examples:

```
father(john, sue).
father(odin, X).
parent(X, Y) :- father(X, Y).
gparent(X, Z) :- parent(X, Y), parent(Y, Z).
```

- How to read as predicate calculus?

$\texttt{father(john, sue)}$
$\forall \texttt{X}. \texttt{father(odin, X)}.$
$\forall \texttt{X}, \texttt{Y}. \texttt{father(X,Y)} \Rightarrow \texttt{parent(X,Y)}.$
$\forall \texttt{X}, \texttt{Y}, \texttt{Z}. \texttt{parent(X,Y)} \,\&\, \texttt{parent(Y,Z)} \Rightarrow \texttt{gparent(X,Z)}$

Relation to Predicate Calculus

- In general:

  `t.`

  $\mapsto \forall x_1 \ldots \forall x_m . \mathtt{t}$

  [called "atomic formula"]

  `t :- t`$_1$`, t`$_2$`, ..., t`$_n$`.`

  $\mapsto \forall x_1, \ldots, x_m . \mathtt{t}_1 \,\&\, \mathtt{t}_2 \ldots \,\&\, \mathtt{t}_n \Rightarrow \mathtt{t}$

  [called "(production) rule"]

- Set of Predicate Calculus Expressions = Knowledge Base ≡
  Conjunctive Normal Form:
  $(A_1 \vee \neg A_2 \vee \neg A_7) \;\&\; (\neg A_1 \vee A_3 \vee A_4) \;\&\; \cdots \;\&\; (\neg A_2 \vee \neg A_4)$

- **Horn** clause is disjunction with ONE Positive Literal

- ⟨`Horn`⟩ Form is CNF, where every clause is Horn
  ... has ONE Positive Literal

So ⟨`Horn`⟩ ⊂ CNF.
∃ Predicate Calculus expressions
which canNOT be written as Horn Clauses.

(Eg: $A \vee B$)

Prolog's Proof Process

- User provides
  - KB: Knowledge Base
    (List of Horn Clauses — axioms)
  - γ: Query (aka Goal, Theorem)
    (Literal) 1. Who is mortal? `mortal(X).`
    2. Is Socrates mortal? `mortal(soc).`
- *Prolog* finds
  - a *Proof* of γ, from KB , if one exists
    & substitution for γ's variables: σ

$$
\begin{array}{lll}
KB & \vdash_P & \gamma\{\sigma\} \\
KB_1 & \vdash_P & \texttt{mortal(X)}\{\,\texttt{X/soc}\,\}
\end{array}
$$

  - Failure (otherwise)

- Returns bindings
  Finds "Top-Down" (refutation) Proof
  Actually returns LIST of σ_is [one for each proof]

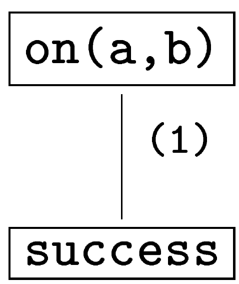
  { `X/soc` } { `X/plato` } { `X/freddy` } ...

Examples of Proofs: I

- Using Knowledge Base, KB_1 =

$$\begin{cases} \text{on(a, b).} & \textit{(1)} \\ \text{on(b, c).} & \textit{(2)} \\ \text{above(X, Y) :- on(X,Y).} & \textit{(3)} \end{cases}$$

- Query γ_1: `on(a,b)`



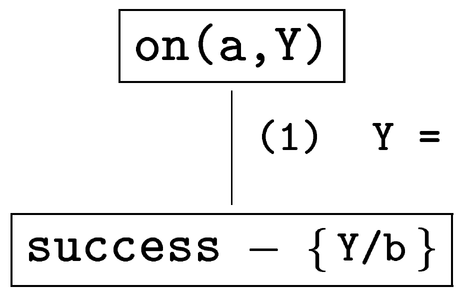
- Hence, $KB_1 \vdash_P$ `on(a,b)`$\{\}$.
  $\nwarrow$ empty substitution

(Like Data Base retrieval)

Examples of Proof: II (variables)

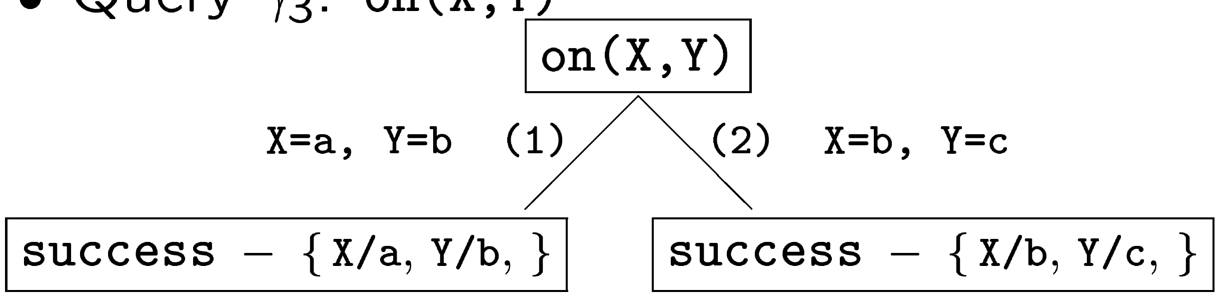
- Using Knowledge Base, KB_1

- Query γ_2: `on(a,Y)`



(Say $KB_1 \vdash_P$ `on(a,Y)`$\{$`Y/b`$\}$)

- Query γ_3: `on(X,Y)`



$$\begin{pmatrix} KB_1 \vdash_P \texttt{on(X,Y)}\{\texttt{X/a, Y/b}\} & \rightarrow & KB_1 \vdash_P \texttt{on(a,b)} \\ KB_1 \vdash_P \texttt{on(X,Y)}\{\texttt{X/b, Y/c}\} & \rightarrow & KB_1 \vdash_P \texttt{on(b,65)} \end{pmatrix}$$

Examples of Proof: III (failures)

(Using Knowledge Base, KB_1)

- Query γ_4: `on(a,b10)`

`on(a,b10)`
$\times$

(Hence, $KB_1 \not\vdash_P$ `on(a,b10)`)

- Query γ_5: `on(X,b10)`

`on(X,b10)`
$\times$

(Hence, $KB_1 \not\vdash_P$ `on(X,b10)`, for any value of `X`.)

Examples of Proof: IV (rules)

(Using KB_1)

- Query γ_6: `above(b,c)`

```
above(b,c)
    |  (3)  X = b, Y=c
on(b,c)
    |  (2)
success
```

(Hence, $KB_1 \vdash_P$ `above(b,c)`)

- Query γ_7: `above(b,W)`

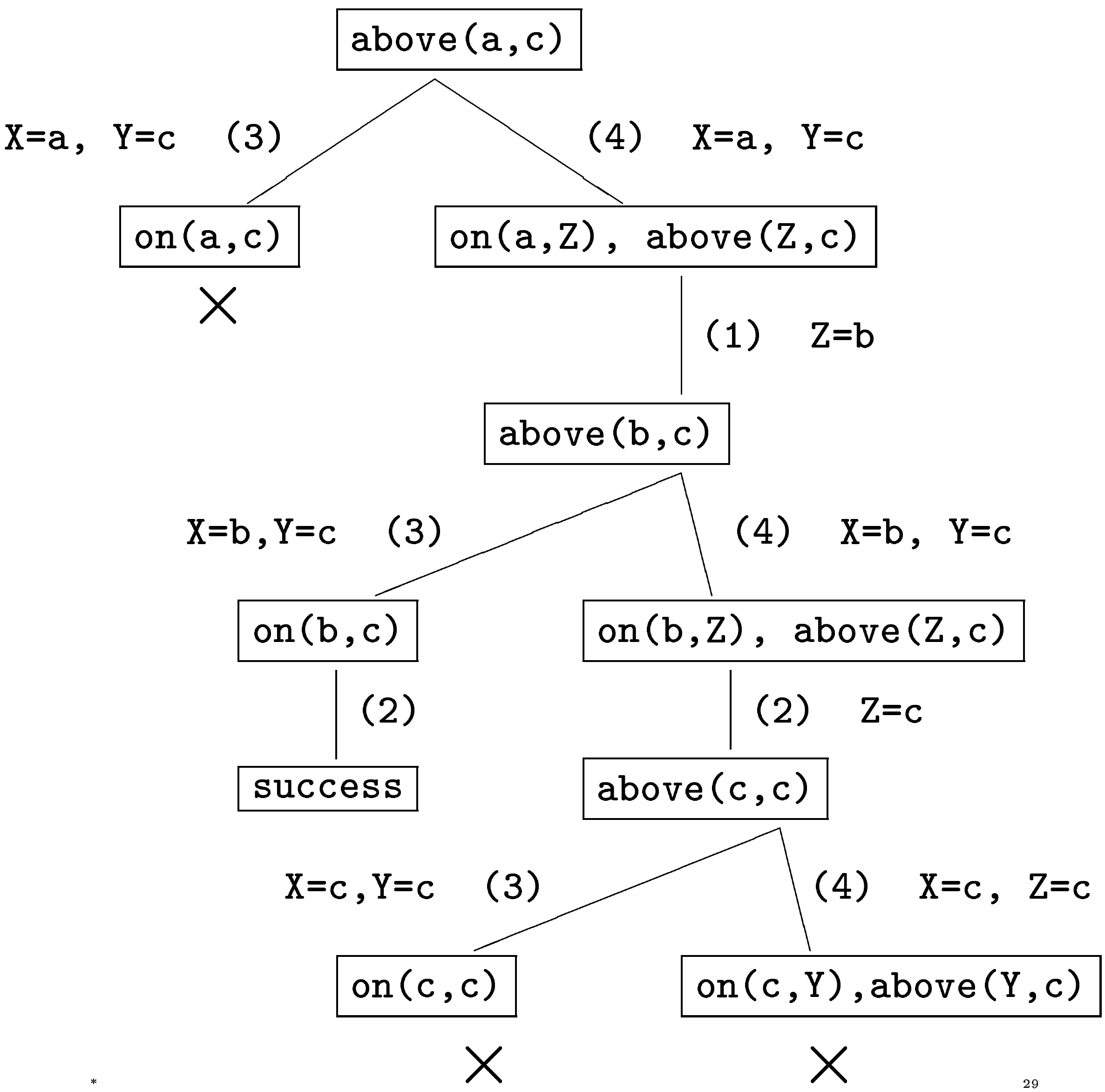
```
above(b,W)
    |  (3)  X=b, Y=W
on(b,W)
    |  (2)  W=c
success - {W/c}
```

(Hence, $KB_1 \vdash_P$ `above(b,W)`$\{$`W/c`$\}$

$\rightarrow$ $KB_1 \vdash_P$ `above(b,c)`)

Examples of Proof: V (big)

$$KB_2 = \begin{cases} \text{on(a, b).} & (1) \\ \text{on(b, c).} & (2) \\ \text{above(X, Y) :- on(X,Y).} & (3) \\ \text{above(X, Y) :- on(X,Z), above(Z,Y).} & (4) \end{cases}$$



Examples of Proof: VI (many answers)

- Using KB_3 =

$$\begin{cases} \text{on(a, b).} & (1) \\ \text{on(b, c).} & (2) \\ \text{above(X, Y) :- on(X,Y).} & (3) \\ \text{above(X, Y) :- on(X,Z), above(Z,Y).} & (4) \\ \text{above(c1, c2).} & (5) \end{cases}$$

  Query γ_9: `above(X,Y)`

- Answers:

```
– [X=a,Y=b]    above(a , b )  (3), (1)
– [X=b,Y=c]    above(b , c )  (3), (2)
– [X=a,Y=c]    above(a , c )  (4), (1), (3), (2)
– [X=c1,Y=c2]  above(c1, c2)  (5)
```

Prolog's Proof Process

- A *goal* is either
  - a sequence of literals (conjunction),
  - the special goal "`success`"

(eg, `on(X,Y)` `p(X,5), q(X)` `success` ...)

- The sequence of goals

$$\langle \mathtt{G}_1,\ \mathtt{G}_2,\ \ldots,\ \mathtt{G}_n \rangle$$

is a *top-down proof* of $\mathtt{G}_1$
(from the knowledge base, KB) *iff*

1. $\mathtt{G}_n = \mathtt{success}$, and

2. $\mathtt{G}_i$ is a *SUBGOAL* (in KB) of $\mathtt{G}_{i-1}$,
$i = 2, 3, \ldots n$

Subgoals

Subgoals of $\mathtt{G} = \{\, \mathtt{g}_1, \ldots \mathtt{g}_r \,\}$ in KB:

Rule 1 If atomic axiom "t" in KB
where t and $\mathtt{g}_i$ have mgu σ,
then
$\{\, \mathtt{g}_1\sigma, \ldots, \mathtt{g}_{i-1}\sigma, \mathtt{g}_{i+1}\sigma, \ldots, \mathtt{g}_r\sigma \,\}$
is a subgoal of G.

(If $r = 1$, then "success" is subgoal of G.)

Rule 2 If axiom "t :- $\mathtt{t}_1$, ..., $\mathtt{t}_k$" in KB
where t and $\mathtt{g}_i$ have mgu σ,
then

$\{\, \mathtt{t}_1\sigma, \ldots, \mathtt{t}_k\sigma, \mathtt{g}_1\sigma, \ldots, \mathtt{g}_{i-1}\sigma, \mathtt{g}_{i+1}\sigma, \ldots \mathtt{g}_r\sigma \,\}$

is a subgoal of G.

Example of Subgoals – I

$$KB_3 = \begin{cases} (1) & \text{on(a, b).} \\ (2) & \text{on(b, c).} \\ (3) & \text{above(X, Y) :- on(X,Y).} \\ (4) & \text{above(X, Y) :- on(X,Z), above(Z,Y).} \\ (5) & \text{above(c1, c2).} \end{cases}$$

Subgoals of . . .

- `above(A,B)` are
  - `on(A,B)`: $\sigma = \{$ X/A, Y/B $\}$
    using Rule 2, *(3)*
  - `on(A,Z), above(Z,B)`: $\sigma = \{$ X/A, Y/B $\}$
    using Rule 2, *(4)*
  - `success`: $\sigma = \{$ A/c1, B/c2 $\}$
    using Rule 1, *(5)*

Example of Subgoals – II

$$KB_3 = \begin{cases} \textit{(1)} & \text{on(a, b).} \\ \textit{(2)} & \text{on(b, c).} \\ \textit{(3)} & \text{above(X, Y) :- on(X,Y).} \\ \textit{(4)} & \text{above(X, Y) :- on(X,Z), above(Z,Y).} \\ \textit{(5)} & \text{above(c1, c2).} \end{cases}$$

Subgoals of . . .

- $\{$ `on(A,Z1), above(Z1,B)` $\}$ are
  - `above(b,B)`: $\sigma = \{$ `A/a, Z1/b` $\}$
    using Rule 1, *(1)* [1st literal]
  - `above(c,B)`: $\sigma = \{$ `A/b, Z1/c` $\}$
    using Rule 1, *(2)* [1st literal]
  - `on(A,c1)`: $\sigma = \{$ `Z1/c1, B/c2` $\}$
    using Rule 1, *(5)* [2nd literal]
  - `on(Z1,B), on(A,Z1)`: $\sigma = \{$ `X/Z1, Y/B` $\}$
    using Rule 2, *(3)* [2nd literal]
  - `on(Z1,Z), above(Z,B), on(A,Z1)`: $\sigma = \{$ `X/Z1, Y/B` $\}$
    using Rule 2, *(4)* [2nd literal]

Comments wrt Prolog's Proof Procedure

- $\left\{ \begin{array}{c} \text{Variable bindings} \\ \text{Unifier} \end{array} \right\}$ found during proof

- *Prolog* returns these overall mgu's **1-by-1**

- Which "strategy"?
  - Within "frontier" of subgoal-sets,
    which to expand?
  - Given specific subgoal-set,
    which literal?
  - Given specific literal (within subgoal-set),
    which rule/assertion?

  (*Prolog* uses "SLD Resolution" strategy)

- Does *Prolog* work correctly?

- Does *Prolog* run efficiently?

What User Really Types

```
> sicstus
SICStus 3.11.2 (x86-linux-glibc2.3):  Wed Jun 2 11:44:50 CEST
Licensed to cs.ualberta.ca
| ?- [user].          % For user to enter ``Assert-fact'' mode.
| on(a,b).
| on(b,c).
| above(X,Y) :- on(X,Y).
| ^D user con...      % Typing ``^D exits ``assert'' mode.

yes                   % Prolog's answer to most operations.

| ?- on(a,b).         %  User asks a question.
yes                   %  Prolog's answer.

| ?- on(a,Y).         %  User's second question.
Y = b_                %  Prolog's answer:  a binding list.
                      %  User types CR.
yes                   %  Prolog's statement that there was answe

| ?- on(X,Y).         %  User's third question.

X = a
Y = b;                %  Prolog's binding list
                      %  User asks for ANOTHER answer
                      %   by typing ``;''.
X = b
Y = c;                %  Prolog supplies another binding list
                      %  Still not satisfied, user asks for
                      %  yet ANOTHER answer by typing ``;''.
no                    %  Prolog's no means ¬∃ other answers

| ?-
```

What User Really Types – II

```
> sicstus
SICStus 3.11.2 (x86-linux-glibc2.3):  Wed Jun 2 11:44:50 CEST
Licensed to cs.ualberta.ca
| ?- [file1].        % File ``file1'' contains propositions
file1 consulted 120 bytes 0.0333333 sec.

yes                  % Prolog's answer to this operation.

| ?- on(a,b10).      %  User asks a question.
no                   %  Prolog's answer:  not derivable.

| ?- on(X,b10).      %  User's next question.
no                   %  Again, no answer.

| ?- above(b,c).

yes                  %  Prolog can find a proof

                     %  Notice:  needs more than simple lookup.

| ?- above(b,W).

W = c;               %  Prolog find an answer.

no                   %  ... but only one answer.

| ?-
```