Resolution Refutation Proof

10/02/2025

Koustav Rudra

Procedure for Resolution

- Convert the set of rules and facts into clause form (conjunction of clauses)
- Insert the negation of the goal as another clause
- Use resolution to deduce a refutation

- If the refutation is obtained then the goal can be deduced from the set of facts and rules

- φ : $F1 \wedge F2 \wedge \ldots \wedge Fn \rightarrow G$
- φ : $\sim(F1 \wedge F2 \wedge \ldots \wedge Fn) \vee G$ → Valid
- $\sim\varphi$: $F1 \wedge F2 \wedge \ldots \wedge Fn \wedge \sim G$ → Unsatisfiable

Resolution

- If Unify(z_j, ~q_k) = θ
- $z_1 \lor \ldots z_m, q_1 \lor \ldots, q_n$
- SUBST(θ, $z_1 \lor \ldots z_{i-1} \lor z_{i-1} \ldots z_m \lor q_1 \lor \ldots q_{k-1} \lor q_{k+1} \ldots \lor q_n$)

Example

- Akash, Amit, and Arun are students of a school
- Every student is either wicked or is a good Cricket player, or both
- No Cricket player likes rain and all wicked students like potions
- Arun dislikes whatever Akash likes and likes whatever Akash dislikes
- Arun likes rain and potions

- Is there anyone who is good in Cricket but not in potions?

Representation in Predicate Logic

- Akash, Amit, and Arun are students of a school
  - C1: Student(Akash)
  - C2: Student(Amit)
  - C3: Student(Arun)

- Every student is either wicked or is a good Cricket player, or both
  - $\forall_x$Student(x)→Wicked(x) ∨ Cricket(x)
  - C4: ~Student(x) ∨ Wicked(x) ∨ Cricket(x)

Representation in Predicate Logic

- No Cricket player likes rain and all wicked students like potions
  - $\forall_x$ Cricket(x)→~Likes(x, Rain)
  - $\forall_x$ Wicked(x)→Likes(x, Potions)
  - C5: $\forall_x$ ~Cricket(x) ∨ ~Likes(x, Rain)
  - C6: $\forall_x$ ~Wicked(x) ∨ Likes(x, Potions)

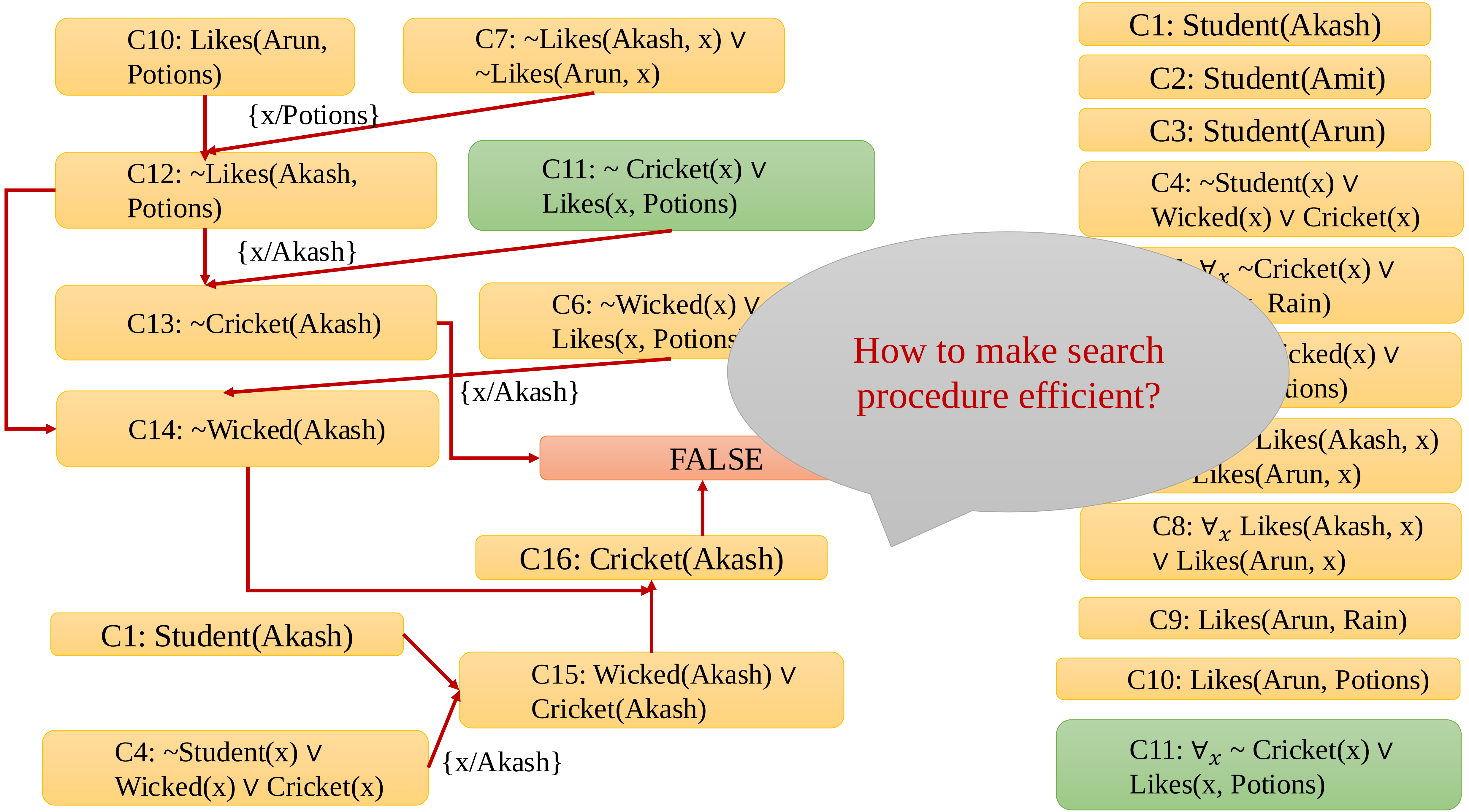
Representation in Predicate Logic

- Arun dislikes whatever Akash likes and likes whatever Akash dislikes
  - $\forall_x$ Likes(Akash, x) $\Leftrightarrow$ ~Likes(Arun, x)
  - $\forall_x$ [Likes(Akash, x) $\rightarrow$~Likes(Arun, x)] $\wedge$ [~Likes(Arun, x) $\rightarrow$Likes(Akash, x)]
  - C7: $\forall_x$ ~Likes(Akash, x) $\vee$ ~Likes(Arun, x)
  - C8: $\forall_x$ Likes(Akash, x) $\vee$ Likes(Arun, x)

- Arun likes rain and potions
  - C9: Likes(Arun, Rain)
  - C10: Likes(Arun, Potions)

Representation in Predicate Logic

- Is there anyone who is good in Cricket but not in potions?
  - G: $\exists_x$ Cricket(x) ∧ ~Likes(x, Potions)
  - ~G: $\forall_x$ ~ Cricket(x) ∨ Likes(x, Potions)
  - C11: $\forall_x$ ~ Cricket(x) ∨ Likes(x, Potions)



C10: Likes(Arun, Potions)
C7: ~Likes(Akash, x) ∨ ~Likes(Arun, x)
{x/Potions}
C12: ~Likes(Akash, Potions)
C11: ~ Cricket(x) ∨ Likes(x, Potions)
{x/Akash}
C13: ~Cricket(Akash)
C6: ~Wicked(x) ∨ Likes(x, Potions
{x/Akash}
C14: ~Wicked(Akash)
FALSE
How to make search procedure efficient?
C16: Cricket(Akash)
C1: Student(Akash)
C15: Wicked(Akash) ∨ Cricket(Akash)
C4: ~Student(x) ∨ Wicked(x) ∨ Cricket(x)
{x/Akash}
C1: Student(Akash)
C2: Student(Amit)
C3: Student(Arun)
C4: ~Student(x) ∨ Wicked(x) ∨ Cricket(x)
$\forall_x$ ~Cricket(x) ∨
Rain)
cked(x) ∨
tions)
Likes(Akash, x)
Likes(Arun, x)
C8: $\forall_x$ Likes(Akash, x) ∨ Likes(Arun, x)
C9: Likes(Arun, Rain)
C10: Likes(Arun, Potions)
C11: $\forall_x$ ~ Cricket(x) ∨ Likes(x, Potions)

Resolution Refutation Strategies

Resolution Strategies

- Unit Resolution
  - Every resolution step must involve a unit clause

  - Unit clause
    - A clause that does not have any OR
    - It just has one predicate or its negation

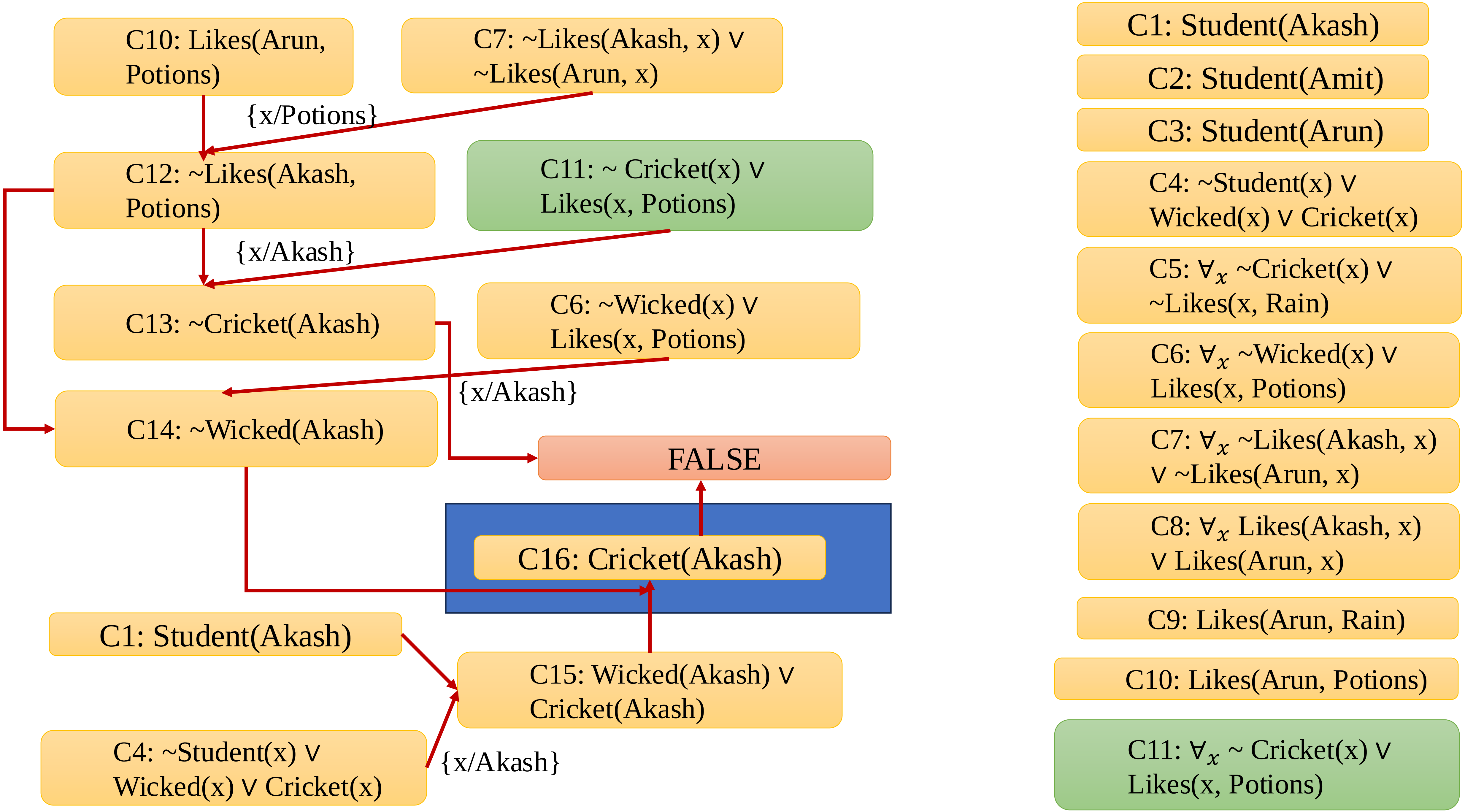
  - Leads to a good speedup

  - Incomplete in general
    - There might be cases that can’t be deduced using unit resolution but can be deduced using other resolution methods

  - Complete for Horn knowledge bases

Resolution Strategies

- Input Resolution
  - Every resolution step must involve an input sentence (from the query or KB)
    - All can’t be derived clauses
  - In Horn knowledge bases, Modus Ponens is a kind of input resolution strategy
  - Incomplete in general
  - Complete for Horn knowledge bases

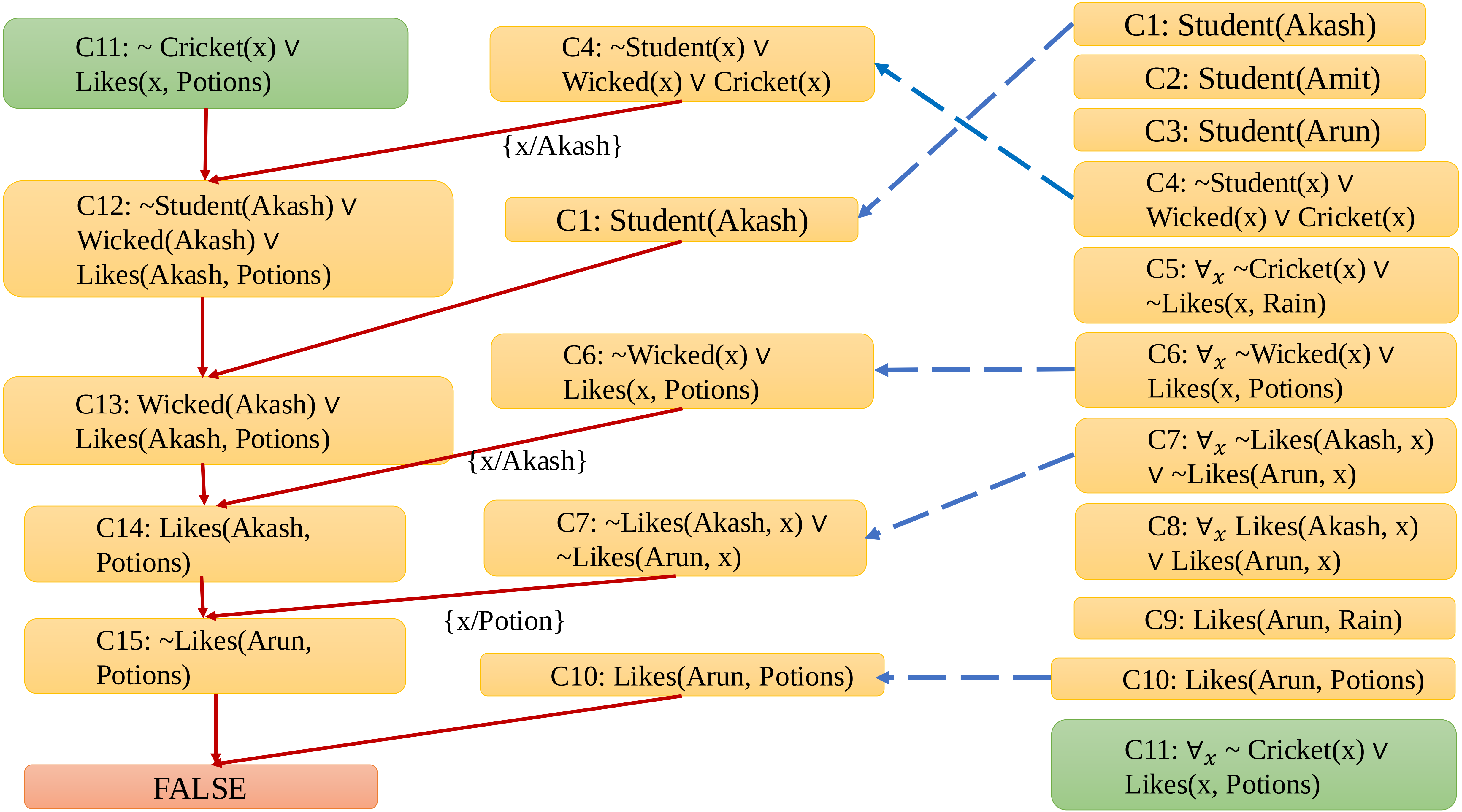

C10: Likes(Arun, Potions)
C7: ~Likes(Akash, x) ∨ ~Likes(Arun, x)
{x/Potions}
C12: ~Likes(Akash, Potions)
C11: ~ Cricket(x) ∨ Likes(x, Potions)
{x/Akash}
C13: ~Cricket(Akash)
C6: ~Wicked(x) ∨ Likes(x, Potions)
{x/Akash}
C14: ~Wicked(Akash)
FALSE
C16: Cricket(Akash)
C1: Student(Akash)
C15: Wicked(Akash) ∨ Cricket(Akash)
C4: ~Student(x) ∨ Wicked(x) ∨ Cricket(x)
{x/Akash}
C1: Student(Akash)
C2: Student(Amit)
C3: Student(Arun)
C4: ~Student(x) ∨ Wicked(x) ∨ Cricket(x)
C5: $\forall_x$ ~Cricket(x) ∨ ~Likes(x, Rain)
C6: $\forall_x$ ~Wicked(x) ∨ Likes(x, Potions)
C7: $\forall_x$ ~Likes(Akash, x) ∨ ~Likes(Arun, x)
C8: $\forall_x$ Likes(Akash, x) ∨ Likes(Arun, x)
C9: Likes(Arun, Rain)
C10: Likes(Arun, Potions)
C11: $\forall_x$ ~ Cricket(x) ∨ Likes(x, Potions)

Resolution Strategies

- Linear Resolution
  - Slight generalization of input resolution

  - Allows P and Q to be resolved together either
    - if P is in the original KB, or
    - if P is an ancestor of Q in the proof tree

  - Linear resolution is complete


C11: ~ Cricket(x) ∨ Likes(x, Potions)
C4: ~Student(x) ∨ Wicked(x) ∨ Cricket(x)
{x/Akash}
C12: ~Student(Akash) ∨ Wicked(Akash) ∨ Likes(Akash, Potions)
C1: Student(Akash)
C6: ~Wicked(x) ∨ Likes(x, Potions)
C13: Wicked(Akash) ∨ Likes(Akash, Potions)
{x/Akash}
C14: Likes(Akash, Potions)
C7: ~Likes(Akash, x) ∨ ~Likes(Arun, x)
{x/Potion}
C15: ~Likes(Arun, Potions)
C10: Likes(Arun, Potions)
FALSE
C1: Student(Akash)
C2: Student(Amit)
C3: Student(Arun)
C4: ~Student(x) ∨ Wicked(x) ∨ Cricket(x)
C5: ∀x ~Cricket(x) ∨ ~Likes(x, Rain)
C6: ∀x ~Wicked(x) ∨ Likes(x, Potions)
C7: ∀x ~Likes(Akash, x) ∨ ~Likes(Arun, x)
C8: ∀x Likes(Akash, x) ∨ Likes(Arun, x)
C9: Likes(Arun, Rain)
C10: Likes(Arun, Potions)
C11: ∀x ~ Cricket(x) ∨ Likes(x, Potions)

What is Deduction?

- Deduction is also a kind of search
- We have to search within the rules, facts, and clauses
- Find them in appropriate order in which to apply them to deduce clauses and goals

Logic Programming: Prolog

10/02/2025

Koustav Rudra

Objective

- How to write programs using Prolog?

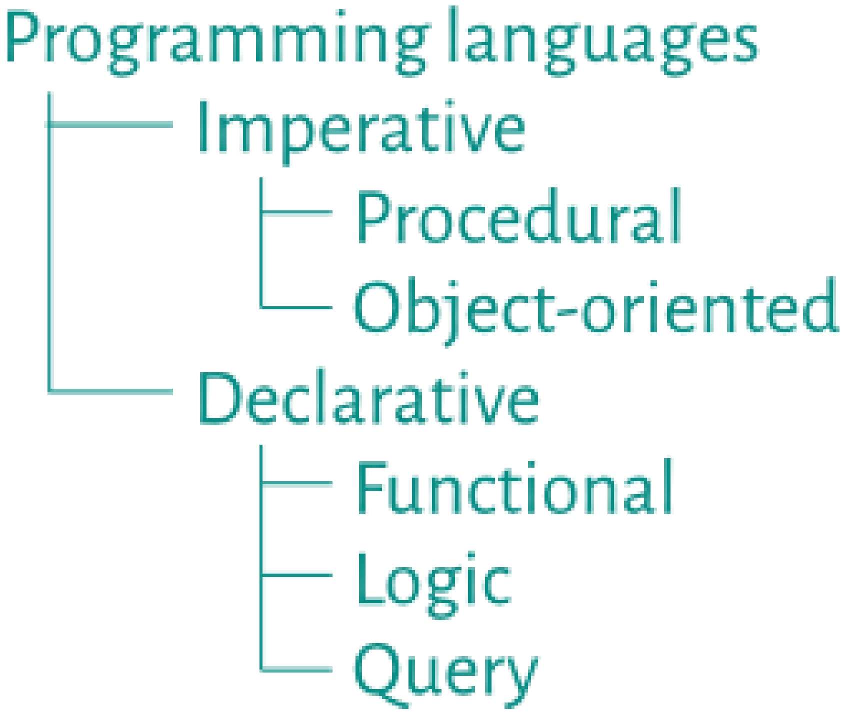
- Tools:
  - GNU Prolog
  - SWI Prolog

Language Choice

- “Known” languages like FORTRAN, C/C++, Java, python
  - Imperative: How-type language

- Goal Oriented Languages (Declarative)
  - Declarative: What-type language
  - LISP
  - ProLog: Truly what-type language

Imperative vs Declarative



- Imperative: Comprises a sequence of commands

- Declarative: Declare what result we want and leave the language to come up with the procedure to produce them

Prolog and FOL

- Prolog Language Syntax
  - Horn clause
  - F1 ∧ F2…Fn→G
  - child(x)∧male(x)→boy(x)

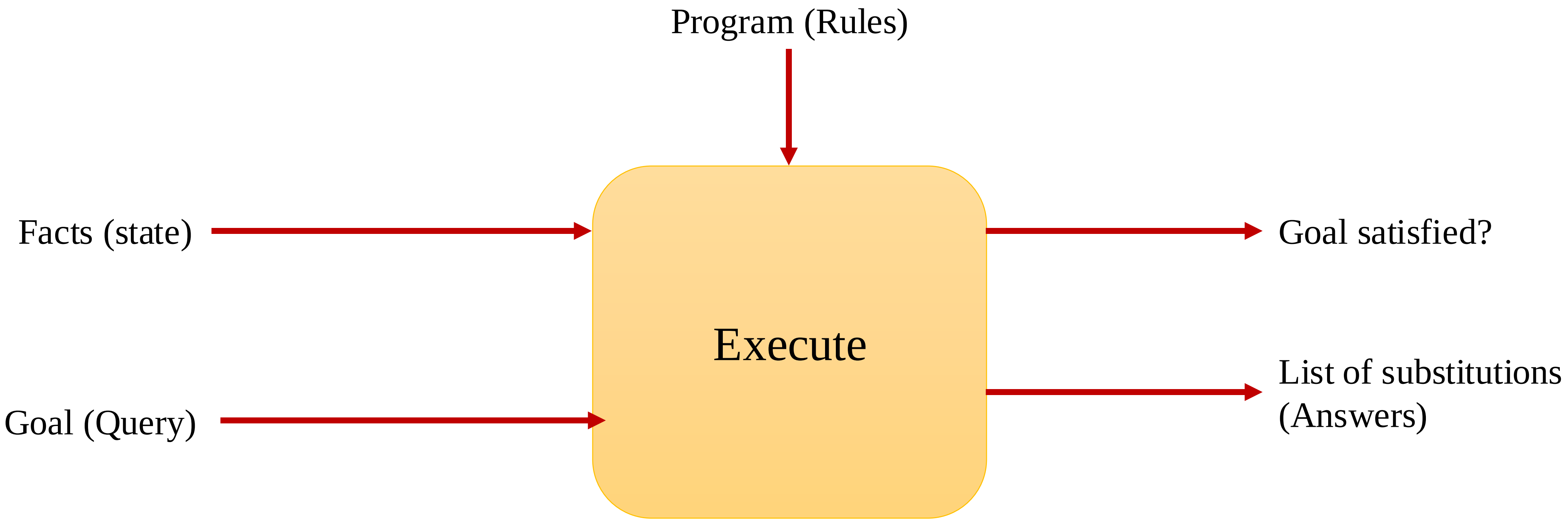
- Prolog proof procedure
  - Resolution Principle

- Prolog goal matching
  - Unification and substitution

How to specify rule?

- $child(x) \land male(x) \rightarrow boy(x)$
- $boy(x) :- child(x) \land male(x)$

Prolog Computation Model



Prolog

- Declarative language
  - Don’t have to specify how a program should execute
  - Just declare what you want to do

Basics

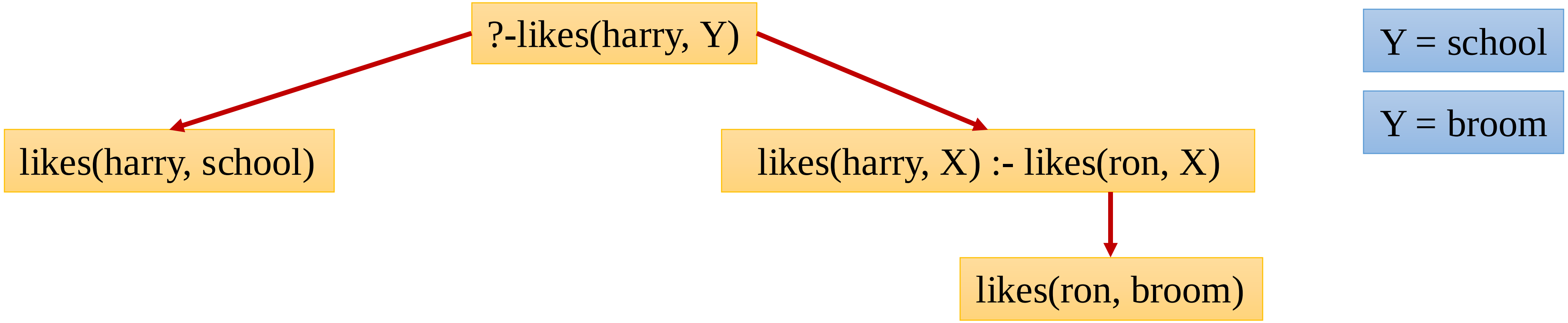
- The notion of instantiation
  - likes(harry, school).
  - likes(ron, broom).
  - likes(harry, X) :- likes(ron, X). [likes(ron, X) → likes(harry, X)]
    - In order to deduce what harry likes we have to deduce first what ron likes

- Consider following goals:
  - ?-likes(harry,broom)

Solution

- ?-likes(harry, broom)
  - likes(harry, X) :- likes(ron, X)
  - likes(ron, broom)

- ?-likes(harry, Y)
  - Prolog will identify all possible instantiations of Y that satisfies likes(harry, Y)

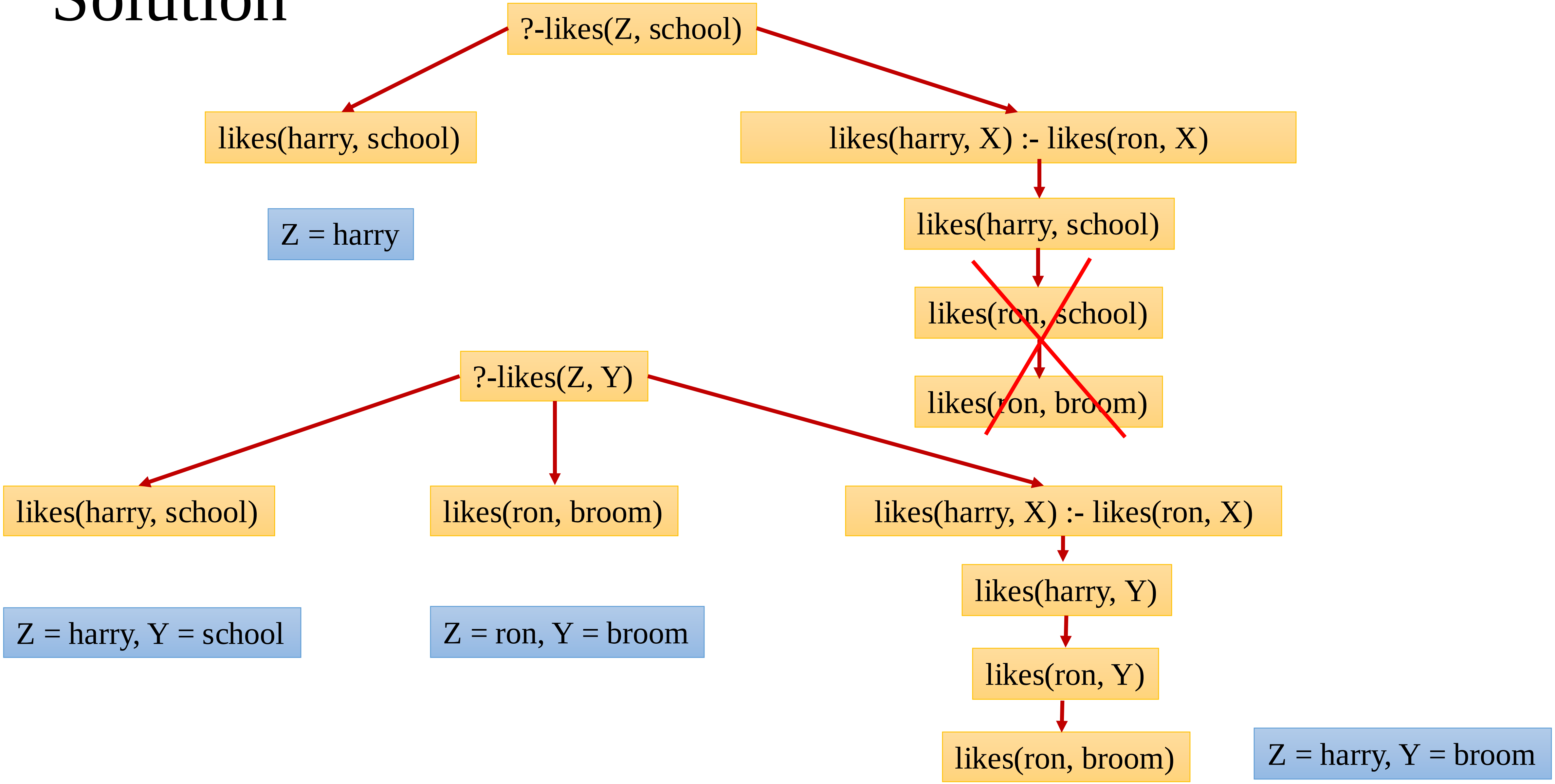


Processing Sequence

- Prolog processes the facts and rules in sequential order

- Put the base condition always has to be specified high up in the order, so that it first tries the base condition, then recursion

Solution



?-likes(Z, school)
likes(harry, school)
Z = harry
likes(harry, X) :- likes(ron, X)
likes(harry, school)
likes(ron, school)
likes(ron, broom)
?-likes(Z, Y)
likes(harry, school)
likes(ron, broom)
likes(harry, X) :- likes(ron, X)
Z = harry, Y = school
Z = ron, Y = broom
likes(harry, Y)
likes(ron, Y)
likes(ron, broom)
Z = harry, Y = broom

Prolog Rules

$$\underbrace{\text{grand\_advisor(X,Z)}}_{\substack{\text{head} \\ \text{(consequent)}}} \text{ :- } \underbrace{\text{advisor(X,Y), advisor(Y,Z)}}_{\substack{\text{body} \\ \text{(antecedent)}}}$$

- $\forall_{xz}\exists_{y}$ advisor(X,Y) $\wedge$ advisor(Y,Z) $\rightarrow$ grand_advisor(X,Z)
- IF there is a Y such that X is advisor of Y AND Y is advisor of Z THEN X is a grand advisor of Z
- Prolog rules are Horn Clauses:
  - $(P_{11} \vee P_{12} \vee \ldots \vee P_{1m}) \wedge \ldots \wedge (P_{n1} \vee P_{n2} \vee \ldots \vee P_{nr}) \rightarrow Q$
  - $Q\text{:-}\ P_{11};P_{12};\ldots;\ P_{1m},\ldots\ \ldots,\ P_{n1};\ P_{n2};\ldots;\ P_{nr}$

Prolog Rules: Recursion

- ancestor(X, Z) :- advisor(X, Z)
- ancestor(X, Z) :- advisor(X, Y), advisor(Y, Z)
- ancestor(X, Z) :- advisor(X, Y1), advisor(Y1, Y2), advisor(Y2, Z)

- ancestor(X, Z) :- advisor(X, Z)
- ancestor(X, Z) :- advisor(X, Y), ancestor(Y, Z)
- X is an ancestor of Z if X is an advisor of Y AND Y is an ancestor of Z

Thank You