Fundamentals of Artificial Intelligence

Procedural Control of Reasoning

Shyamanta M Hazarika
Mechanical Engineering
Indian Institute of Technology Guwahati
s.m.hazarika@iitg.ac.in

http://www.iitg.ac.in/s.m.hazarika/

An Infinite Resolution Branch

Example

Suppose our KB consists of a single formula; showing R as a transitive relation. Could think of R(x,y): as x is the relative of y.

Rule: $\forall xyz\ [(R(x,y) \wedge R(y,z)) \rightarrow R(x,z)]$

C1. $(\neg R(x,y) \vee \neg R(y,z) \vee R(x,z))$

Given the query about existence of someone for everyone who is not a relative; the KB does not entail the query NOR its negation.

Goal: $\exists x \forall y\ \neg R(x,y)$

Negation: $\neg [\exists x \forall y\ \neg R(x,y)]$

$\forall x\ \exists y\ R(x,y)$

This should fail! Problem is if we pose it as a resolution, we end up generating an infinite sequence; we never get to the empty clause!

C2. $R(x,f(x))$

An Infinite Resolution Branch

(¬ R(x,y) ∨ ¬ R(y,z) ∨ R(x,z))

R(x,f(x))

f(x)/y

(¬ R(f(x),z) ∨ R(x,z))

y/x; f(f(y))/z

f(y)/x

$\boxed{\text{R(y,f(f(y)))}}$

f(f(x))/y

x/y

(¬ R(f(f(x)),z) ∨ R(x,z))

y/x; f(f(f(y)))/z

f(f(y))/x

$\boxed{\text{R(y,f(f(f(y))))}}$

This suggests that we cannot use a depth-first search procedure to look for the empty clause. We may get stuck on such an infinite branch.

Question is `Is there a way to detect when we are on such a branch?' so that we can give up and look for alternate paths to the empty clause. Unfortunately, the answer is NO,

Computational Intractability

- For FOL there is **no way to detect if a branch will continue indefinitely**
  - FOL Language is very powerful and can be used as a full programming language.
    - **Just as there is no way to detect when a program is looping; there is no way to detect if a branch will continue indefinitely**.
- Quite problematic from a KR perspective.
  - No procedure that, given a set of clauses, returns satisfiable when the clauses are satisfiable.
    - **Resolution is refutation complete**: returns an empty clause, if the set of clauses is unsatisfiable.
    - **When clauses are satisfiable, the search may or may not terminate**.

Resolution - not a panacea

- Resolution **does not provide a general effective solution** to the reasoning problem.
  - Decision about **which two clauses to resolve** and **which resolution to perform** are made by the control strategy.
    - **Determining the satisfiability of clauses may simply be too difficult computationally!**
- Need to consider **refinements to resolution** to help improve search.
  - One option is to explore a way to **search for derivations that eliminates unnecessary steps** as much as possible.
    - We shall **focus on strategies that can be used to improve the search** in this sense.

Most General Unifiers

- Most **important way of avoiding unnecessary search** in first-order derivation is to **keep search general**.
  - We are looking for substitutions that are NOT overly specific. **The substitution need to unify without making an arbitrary choice that may preclude a path to the empty clause**.
  - A substitution with above characteristics is a **most general unifier**.
  - We can **limit resolution to MGUs without loss of completeness**.

Most General Unifier

Definition When there exist multiple possible unifiers for an expression E, there is at least one, called the **most general unifier, MGU,** g of E, that has the property that if s is any unifier for E yielding Es, then there exist a substitution s’ such that Es = Egs’

Example: P(A, x,) and P(y,z);

g = {A/y,x/z} is an mgu

For s’ = {B/x}, we get

s = {A/y,B/x, B/z}

If we apply mgu, g and then apply the second substitution s’, we get s. Note that the reverse would not be possible.

Most General Unifier

- The **MGU preserves as much generality as possible** for a pair of formulas; by using the MGU we **leave maximum flexibility for the resolvent** to resolve with other clauses.

- The **most general unifier is not necessarily unique**.

  Example P(A, x,) and P(y,z);

  {A/y, z/x} is also an mgu.

Most General Unifier

- MGUs helps immensely in search as it **dramatically reduces the number of resolvents** that can be inferred from two input clauses.
- There exists **procedures including linear time algorithms for efficient computation of MGU** for a pair of literals.
  - MGUs greatly reduce the search and can be calculated efficiently; Consequently, **all Resolution-based systems implemented to date use them**.

Control Strategies

- Breadth-First Strategy
  - Breadth-first strategy is complete, but is grossly inefficient.
- Set-of-support Strategy
  - Have the flavour of a backward reasoning step.
- Unit Preference Strategy
  - Select a single literal clause (a *unit*) to be a parent; ordering strategy.
- Linear-input Form Strategy
  - At least one parent belong to the base set.
- Ancestry-filtered Form Strategy
  - Parent is either in the base set or is an ancestor of the other parent.
- Combination Strategy

An Illustrative Example

Example

1. Whoever can read is literate.
2. Dolphins are not literate.
3. Some dolphins are intelligent.

Prove: Some who are intelligent cannot read.

Predicates -	R(x) :	x can read.
	L(x) :	x is literate.
	D(x) :	x is a dolphin.
	I(x) :	x is intelligent.

Nils J. Nilsson, Principles of AI; Chapter 5, Pages 162-163.

An Illustrative Example

1. Whoever can read is literate.

$\forall x[\ R(x) \rightarrow L(x)]$ C1. $\neg R(x) \vee L(x)$

2. Dolphins are not literate.

$\forall x[D(x) \rightarrow \neg L(x)]$ C2. $\neg D(y) \vee \neg L(y)$

3. Some dolphins are intelligent.

$\exists x[D(x) \wedge I(x)]$ C3. $D(A)$

C4. $I(A)$

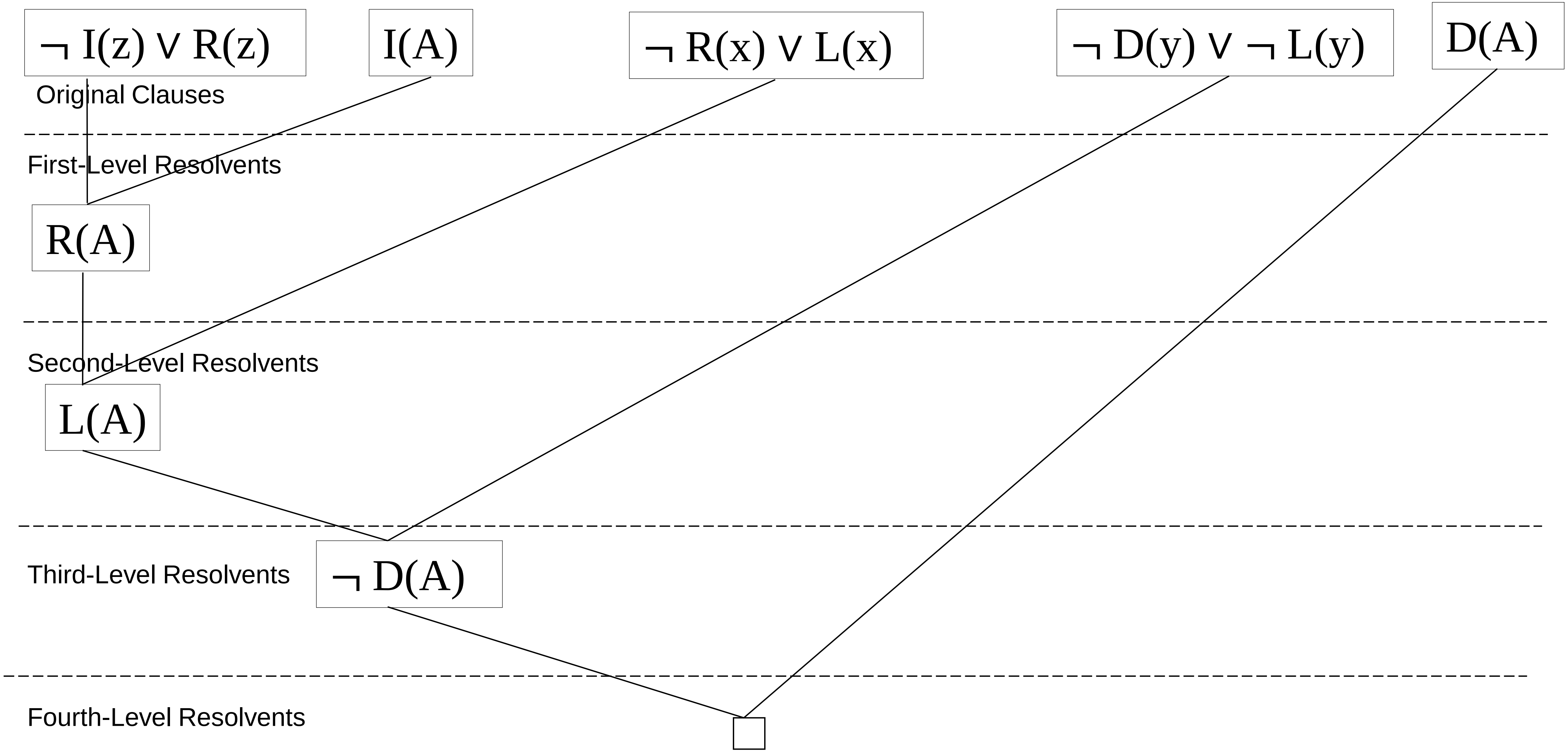
Prove Some who are intelligent cannot read.

$\exists x[I(x) \wedge \neg R(x)]$

Negation $\neg \exists x[I(x) \wedge \neg R(x)]$

$\forall x[\neg I(x) \vee R(x)]$ C5. $\neg I(z) \vee R(z)$

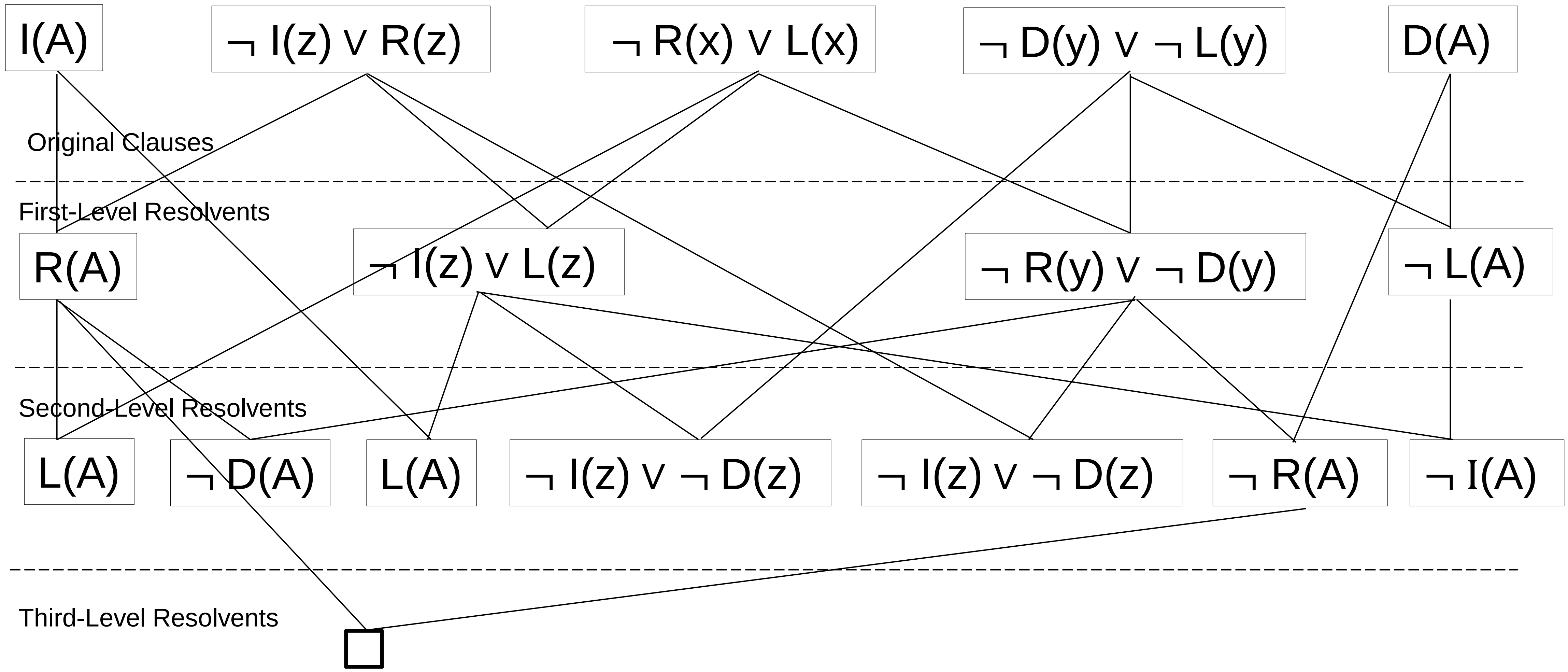
An Illustrative Example



Breadth-First Strategy

- In breadth-first strategy, **all of the first-level resolvents are computed first**, then the **second-level resolvents, and so on**.
  - A first-level resolvent is one between two clauses in the base set;
  - i-th level resolvent is one whose *deepest* parent is the an (i-1)-th level resolvent.
- The breadth-first strategy is complete, but is grossly inefficient.
  - A control strategy for a refutation system is said to be complete if its use results in a procedure that will find a contradiction whenever one exists.

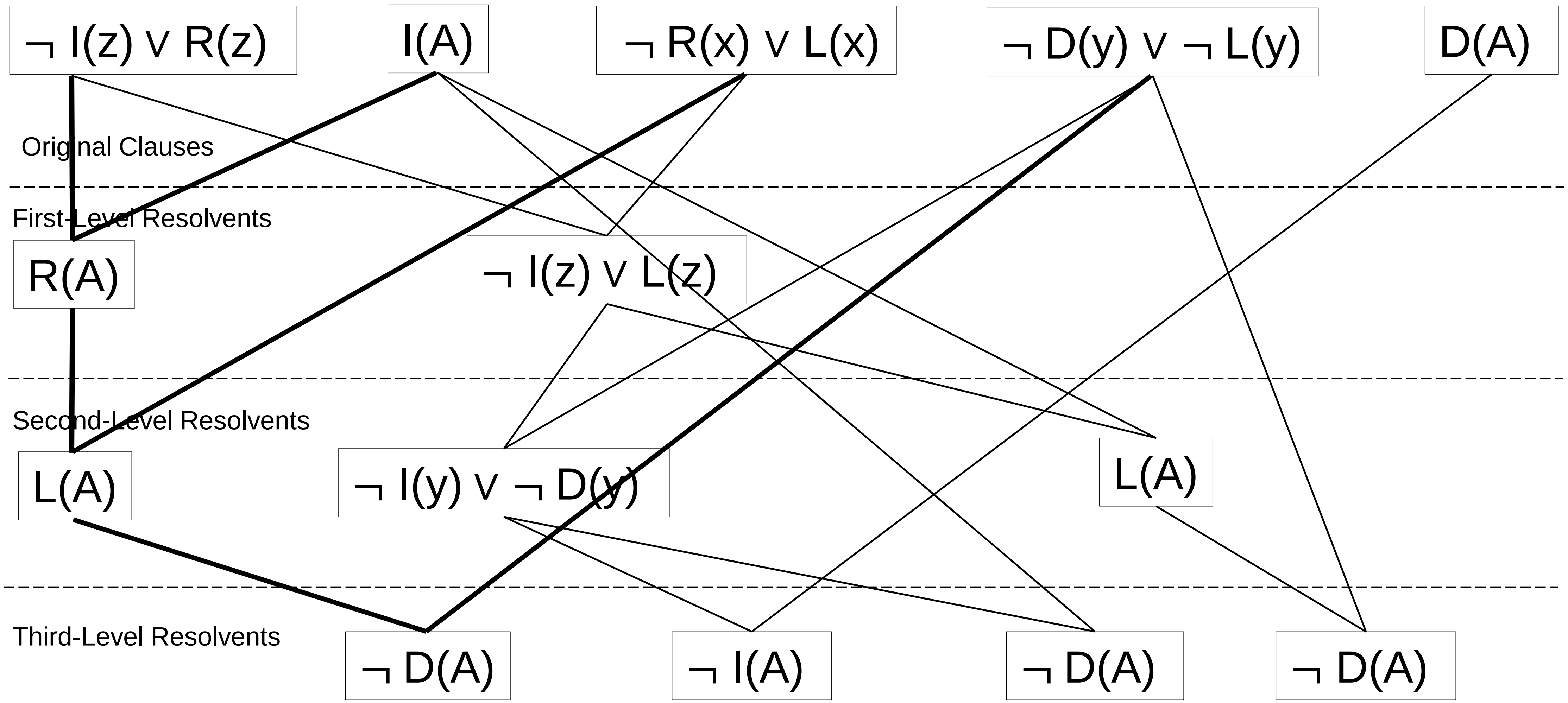
Breadth-first Strategy



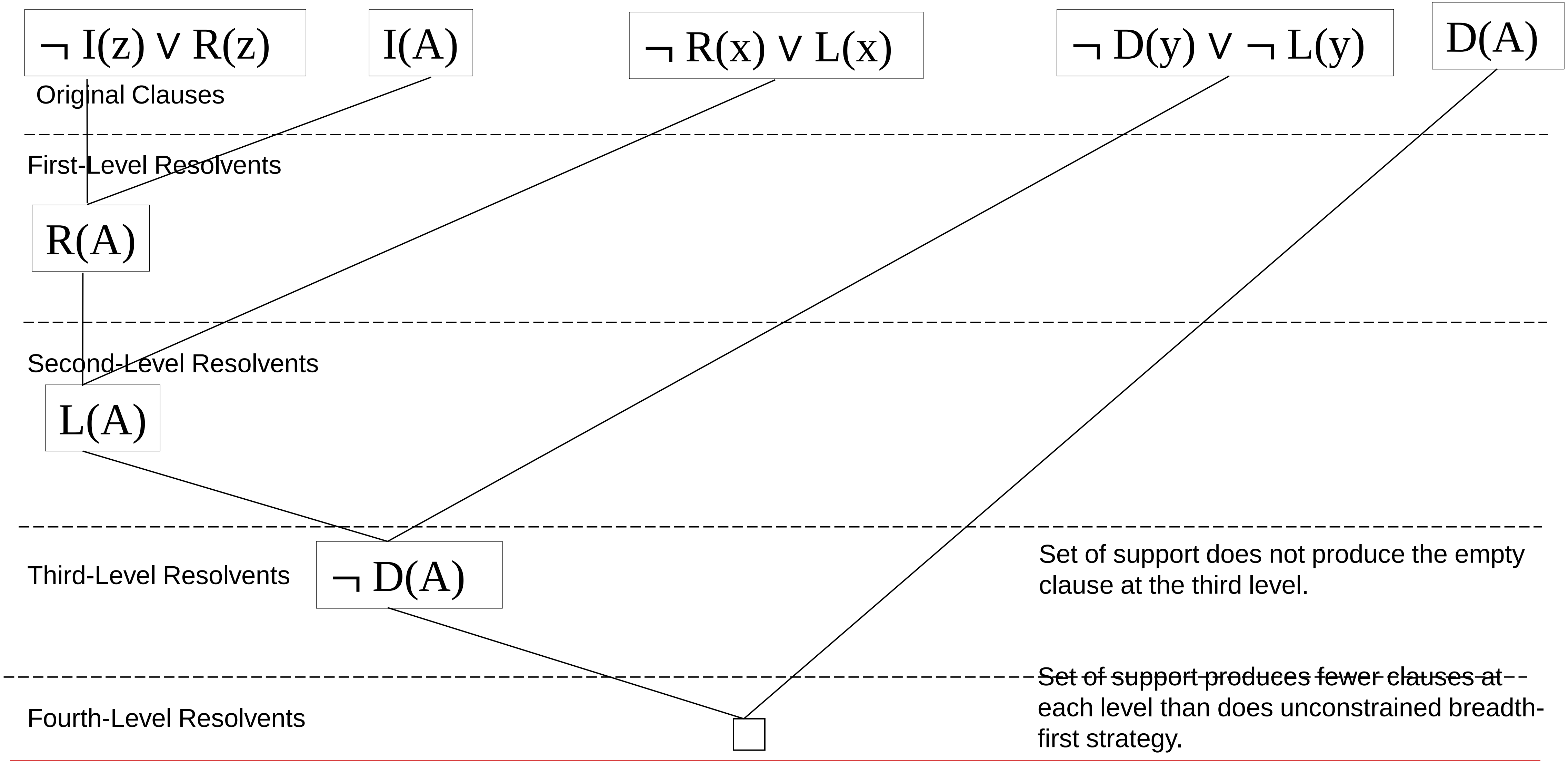
Set-of-support Strategy

- Set-of-support refutation is one in which **at least one parent of each resolvent** is selected from **among the clauses resulting from the negation of the goal wff or from their descendants.**
- In a set-of-support refutation, **each resolution has the flavour of a backward reasoning step**.
  - It uses a clause originating from the goal wff, or one of its descendants.
  - Each of the resolvents might correspond to a subgoal!

Set-of-Support Strategy



Set-of-support Strategy



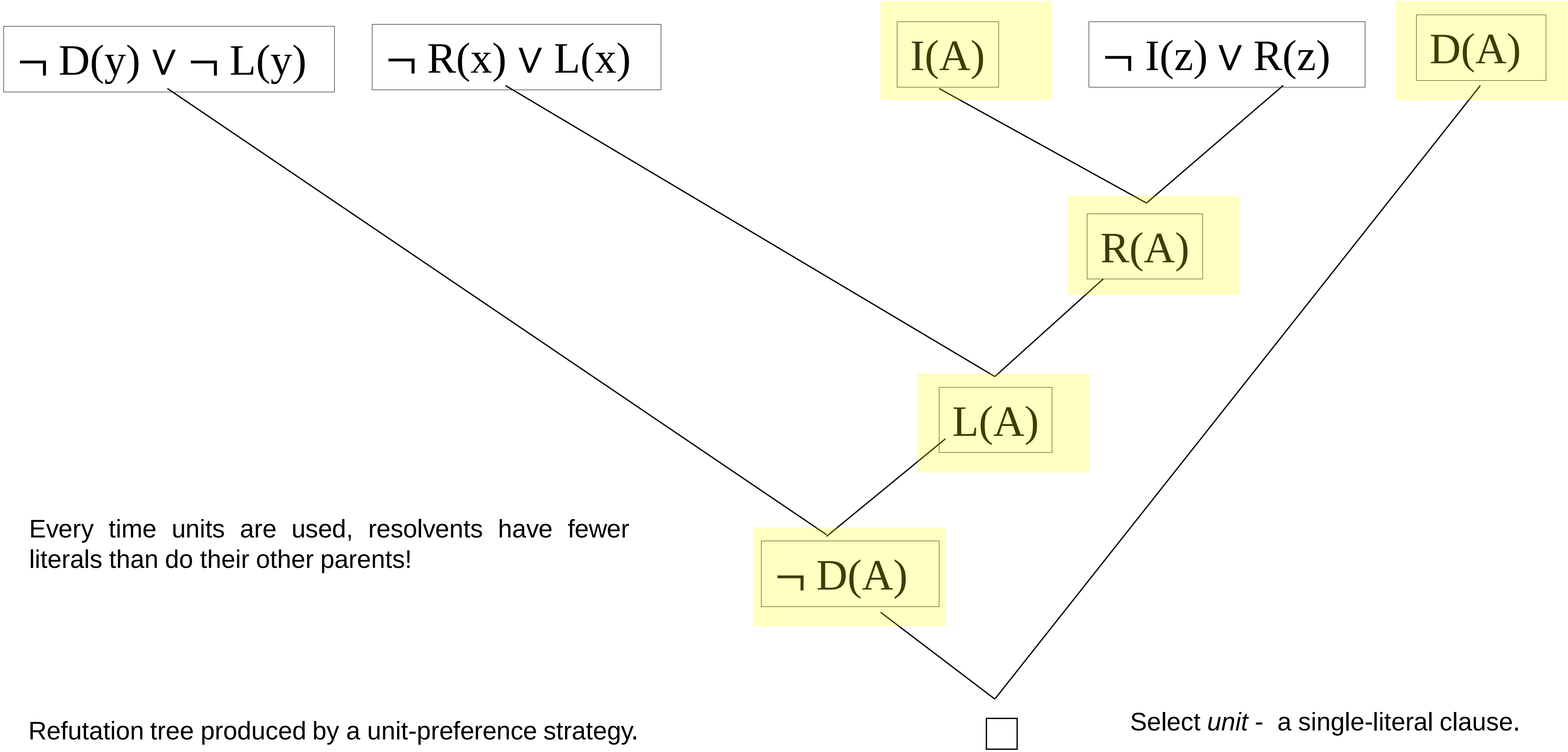
Set of support does not produce the empty clause at the third level.

Set of support produces fewer clauses at each level than does unconstrained breadth-first strategy.

Unit-preference Strategy

- Modification of the set-of-support strategy in which **instead of filling out each level in breadth-first fashion, try and select a single-literal clause (a unit) to be parent** in a resolution.
- Every time units are used, **resolvents have fewer literals than do their other parents**!
  - Using a **unit clause together with a clause of length k always produce a clause of length (k-1).**
  - Focus the search towards producing the empty clause.

Unit-preference Strategy



Every time units are used, resolvents have fewer literals than do their other parents!

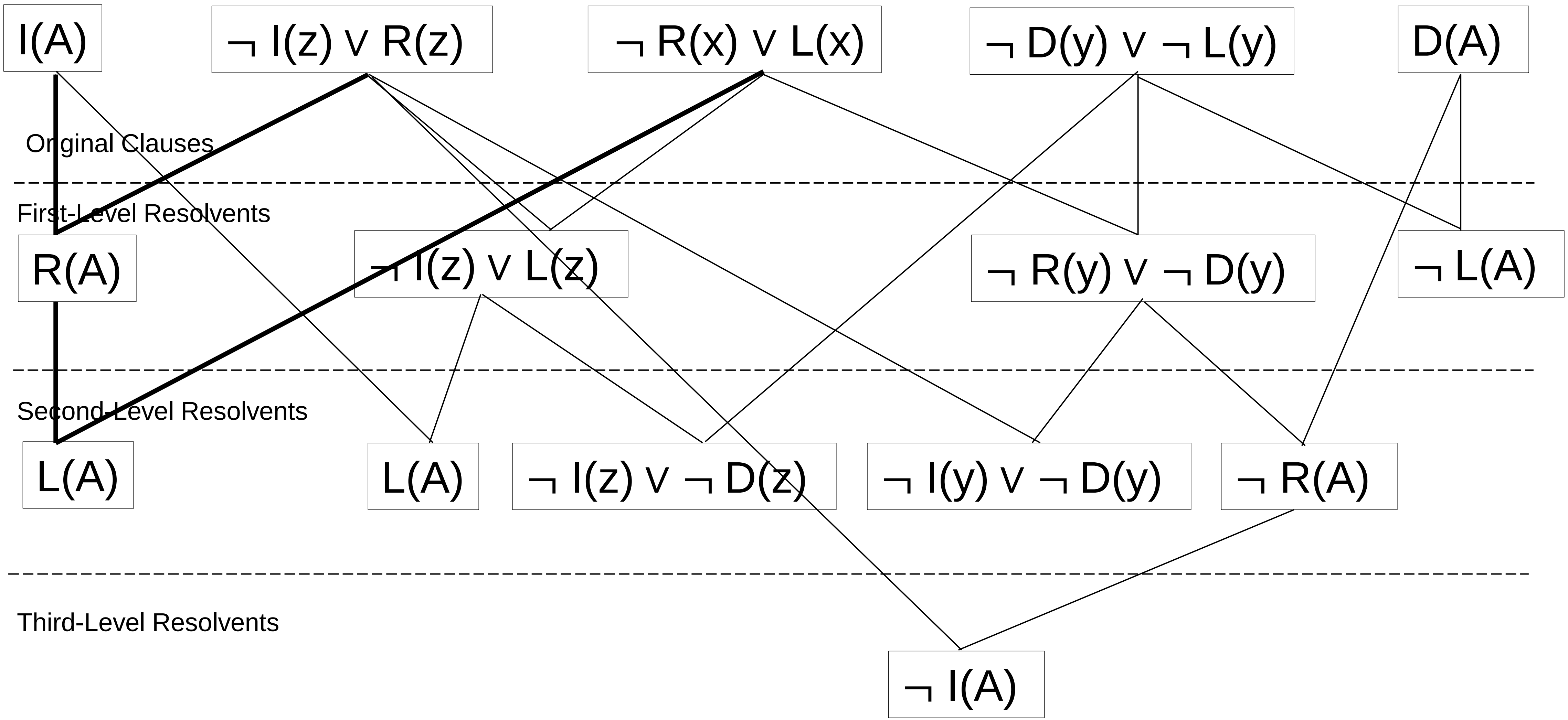
Refutation tree produced by a unit-preference strategy.

Select *unit* - a single-literal clause.

Linear-input Form Strategy

- A linear-input form strategy is one in which **each resolvent has at least one parent belonging to the base set**.
  - First level resolvents are same as a breadth-first search.
  - At subsequent levels, a **linear-input form strategy does reduce the number of clauses produced**.
  - Linear-input form strategies are not complete.

Linear-input Form Strategy



Linear-input Form Strategy

- There are cases in which a **refutation exists** but a **linear-input form refutation does not**; making **linear-input form strategy not complete**.

Example

C1. $Q(u) \vee P(A)$

C2. $\neg Q(w) \vee P(w)$

C3. $\neg Q(x) \vee \neg P(x)$

C4. $Q(y) \vee \neg P(y)$

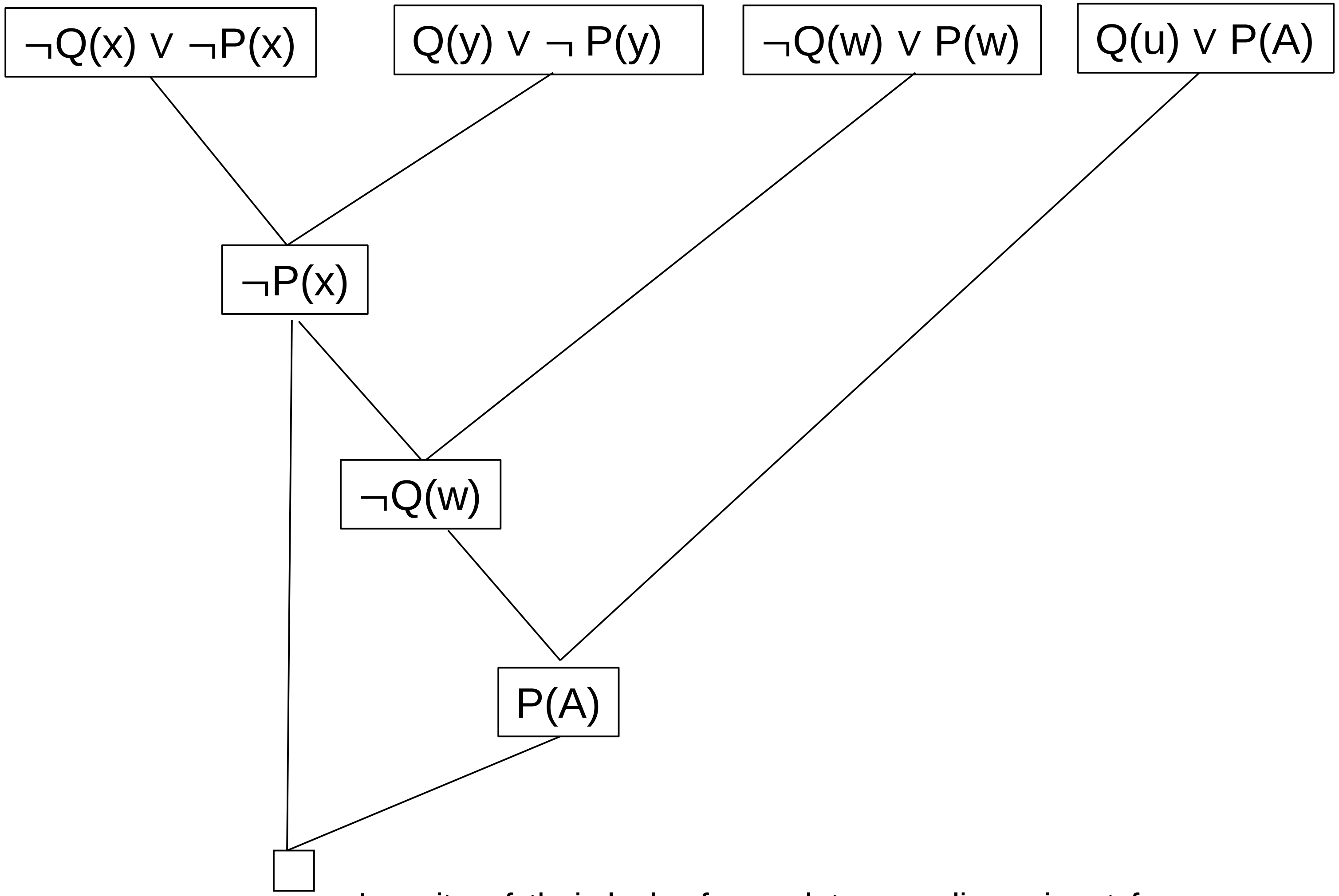
Linear-input Form Strategy

The **set of clauses is clearly unsatisfiable**; but no linear-input form resolution exist.

For a linear-input form refutation, one of the parents of the empty clause must be a member of the base set.

To produce the empty clause in this case, **one must either resolve two single literal clauses** or **two clauses that collapse to a single-literal**.

None of the base case members meet these criteria.



In spite of their lack of completeness, linear-input form strategy is used because of their simplicity and efficiency.

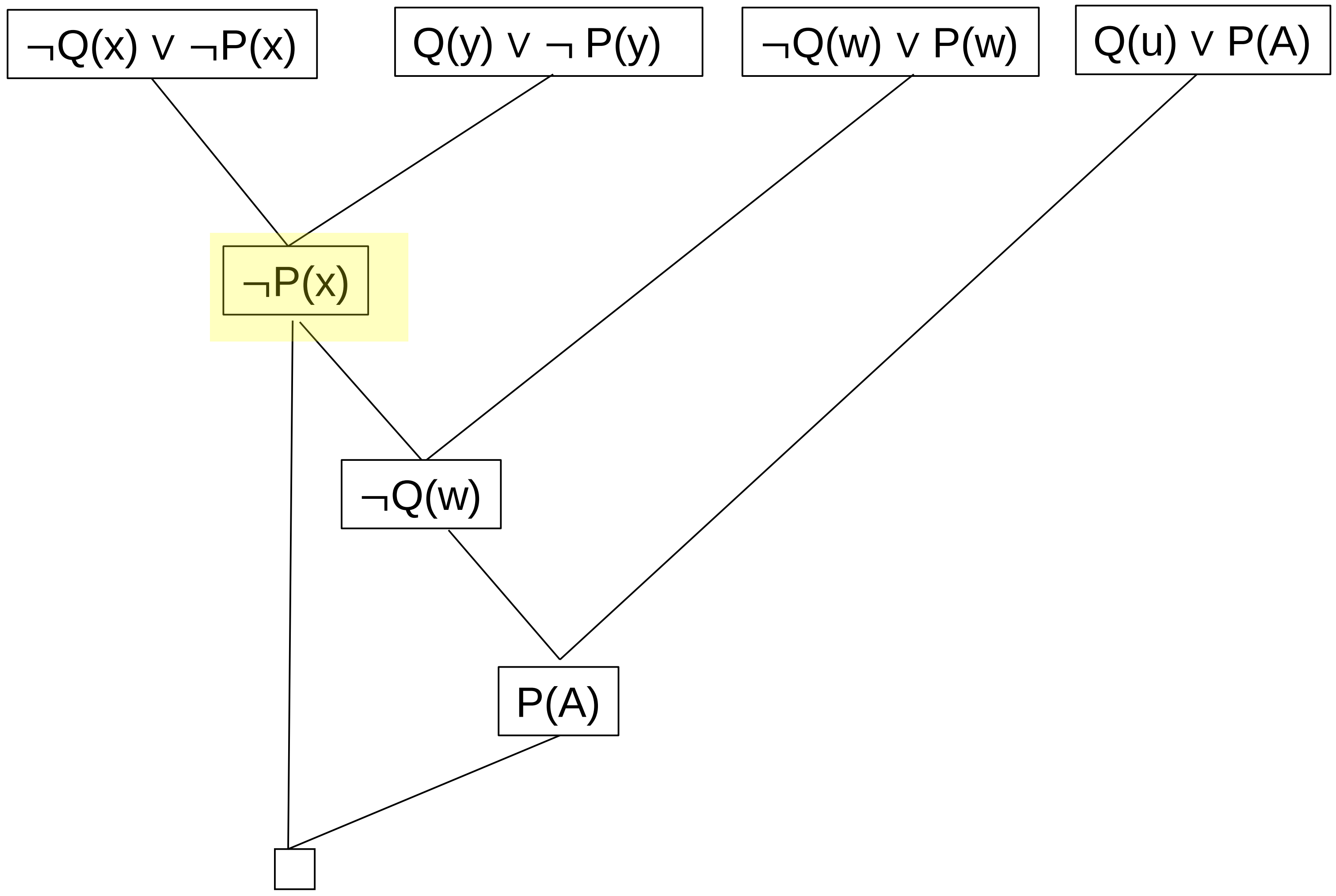
Ancestry-filtered Form Strategy

- In this form of refutation, **each resolvent has a parent that is either in the base set** or that is **an ancestor of the other parent**.
- Much like the linear-form strategy.
- Control strategy guaranteed to produce all ancestry-filtered form proofs is complete.
  - **Completeness is preserved if the ancestors that are used are limited to *merges.***
    - *Merge* is a resolvent that inherits a literal each from the parent such that this literal is collapsed to a singleton by the MGU.

Ancestry-filtered Form Strategy

The refutation tree on the right could have been produced by an ancestry-filtered form startegy.

Here the clause ¬ P(x) is used as an ancestor.



Combination Strategy

- Set-of-support with either linear-form or ancestry-filtered form is a common option.
  - Can be **viewed as a backward reasoning** from goal to sub-goal, to sub-subgoal and so on.
  - Occasionally, combinations can be lead to slower growth of the clause set than would either strategy alone.
- Ordering strategies such as unit-preference strategies can prevent the generation of large number of unneeded clauses.
  - **Order in which resolution is performed is crucial to the efficiency of the resolution system.**

Simplification Strategies

- **Clause Elimination**

  Idea is to keep the number of clauses generated as small as possible, without giving up completeness. Exploit the fact that **if there is a derivation to the empty clause, there is one that does not use certain types of clauses**.

  - Pure Clause
  - Tautologies
  - Subsumed Clauses

- **Procedural Attachment**

  Evaluate – interpret a literal by attached procedures.

Clause Elimination

- **Elimination of Tautologies**
  - Any clause containing a literal and its negation (i.e., a tautology) may be eliminated.
    - Any unsatisfiable set containing a tautology is still unsatisfiable after removing it, and conversely.
- **Elimination by Subsumption**
  - A clause $\{L_i\}$ subsumes a clause $\{M_i\}$, if there exists a substitution `s' such that $\{L_i\}$s is a subset of $\{M_{i.}\}$.
  - Examples:
    - P(x) subsumes P(y) v Q(z)
    - P(x) subsumes P(A)
    - P(x) subsumes P(A) v Q(z)
    - P(x) v Q(A) subsumes P(f(A)) v Q(A) v R(y)

Clause Elimination

Elimination of Tautologies

- Any clause containing a literal and its negation (i.e., a tautology) may be eliminated.
  - Any unsatisfiable set containing a tautology is still unsatisfiable after removing it, and conversely.

Elimination by Subsumption

- A clause $\{L_i\}$ subsumes a clause $\{M_i\}$, if there exists a substitution `s' such that $\{L_i\}s$ is a subset of $\{M_{i.}\}$.
- A clause in an unsatisfiable set that is subsumed by another clause in the set can be eliminated without affecting the unsatisfiability of the rest of the state.
  - Leads to **substantial reduction in the number of resolutions to find refutation**.

Procedural Attachment

- It is possible and more **convenient to evaluate the truth value of literals**; than to include these literals, or their negations in the base set.
- **`Evaluation' refers to interpretation of the expressions with reference to a model**.

<u>For example</u>

Equals(7,3) can be evaluated by *attaching a procedure* that computes / checks the equality of two numbers.

Given such a program for the above predicate,

Equals(7,3) evaluates to False.

Procedural Attachment

- It is also **possible to attach procedures to function symbols.**
- Establish connection or procedural attachment between executable computer code and predicate calculus expressions.
- **Clause set can be simplified by such evaluations.**
  - If a literal in a clause evaluates to True; entire clause can be removed.
  - If a literal evaluates to False; then the occurrence of just that literal in the clause can be eliminated.

For example

$[P(x) \vee Q(A) \vee \text{Equals}(7,3)]$ Can be replaced by $[P(x) \vee Q(A)]$ as Equals(7,3) evaluates to False.

Sorted Logic

- Sorted Logic involves **associating sorts with all terms**.

  Example

  Variable x might be a sort **Female.**

  Function mother may be of sort **Person** → **Female.**

- Keeping a **taxonomy of sorts** can help.

  Example

  Woman is a subsort of **Person**

- Refuse unification between P(t) and P(s) if s and t are from different sorts!
  - Only meaningful (with respect to sorts) unifications can lead to the empty clause.

Connection Graph

- Given a set of clauses, precompute a graph with edges between any two unifiable literals of opposite polarity and labelled with the MGU.
- Resolution procedure than involves selecting a link, computing a resolvent clause and inheriting links for the new clause from its input clauses.
  - No unification is done at run-time!
- Here, resolution can be seen as a state-space search problem – find a sequence of links that ultimately produce the empty clause.
  - Techniques for improving state-space search can be applied.

Knowledge Representation and Reasoning

- We have discussed Knowledge Representation and Reasoning in this Module of the course.
  - Argued why LOGIC is the first choice for knowledge representation and reasoning.
- Examined FOL as a knowledge representation formalism.
  - FOL is not the only choice.
  - FOL is simple and convenient one to begin with!
- Looked at Resolution and Resolution Refutation Proofs.
  - Resolution Derivations – symbol level operation leading to knowledge level logical interpretations.
  - Answer extraction.
  - Strategies and Simplifications leading to refinements in Resolution to help improve search.