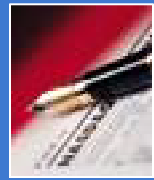
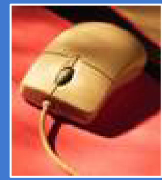




Topics in Knowledge Representation and Reasoning

# Optimizing Description Logic Subsumption

**Maryam Fazel-Zarandi**

**Department of Computer Science**
**University of Toronto**

# Outline

- Introduction
- Optimization Techniques
- Comparison with Other Systems
- Comparing Optimizations
- Discussion

# Introduction

- Realistic applications typically require:
  - expressive logics
  - acceptable performance from the reasoning services

- The usefulness of Description Logics (DLs) in applications has been hindered by the basic conflict between expressiveness and tractability.

- Early experiments with DLs indicated that performance was a serious problem, even for logics with relatively limited expressive powers.

# Introduction

- Terminological reasoning in a DL based Knowledge Representation System is based on determining subsumption relationships with respect to the axioms in a KB.

- Procedures for deciding subsumption (or equivalently satisfiability) in DLs have high worst-case complexities, normally exponential with respect to problem size.

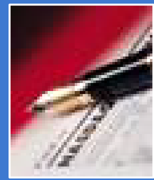
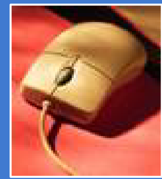
- Empirical analyses of real applications have shown that the kinds of construct which lead to worst case intractability rarely occur in practice.

# Introduction

- Syntax and Semantics:
  - DLs are formalisms that support the logical description of concepts and roles.
- Tableaux subsumption testing algorithm
  - “Using an Expressive Description Logic: FaCT or Fiction?”
  - <u>Problem</u>: The algorithm is too slow to form the basis of a useful DL system.
  - <u>Solution</u>: Employ optimization techniques.

# Outline

- ✓ Introduction
- Optimization Techniques
- Comparison with Other Systems
- Comparing Optimizations
- Discussion


# Optimization Techniques

# Different Optimization Techniques

- Preprocessing optimizations
  - Lexical Normalization and Simplification
  - Absorption
- Partial ordering optimizations
- Satisfiability optimizations
  - Semantic Branching Search
  - Local Simplification
  - Dependency Directed Backtracking
  - Heuristic Guided Search
  - Caching Satisfiability Status

# Lexical Normalization & Simplification

- Concepts in negation normal form.
  - An atomic concept and its negation in the same node label → clash!
  - Not good for concept expressions, the negation is in NNF
- Normalization:
  - Transform concept expressions into a lexically normalized form
  - Identify lexically equivalent expressions
- Simplification:
  - Eliminate redundancy
  - Identify obvious satisfiability and unsatisfiability

| Concept expression | Normal form |
|---|---|
| $\bot$ | $\neg\top$ |
| $C \sqcup D$ | $\neg(\neg C \sqcap \neg D)$ |
| $\exists R.C$ | $\neg(\forall R.\neg C)$ |
| $\neg\neg C$ | $C$ |
| $C \sqcap D$ | $\sqcap\{C, D\}$ |
| $\sqcap\{\sqcap\{C_1, \ldots, C_n\}, \ldots\}$ | $\sqcap\{C_1, \ldots, C_n, \ldots\}$ |
| $\sqcap\{C\}$ | $C$ |

**Table 3.** Normalisation rules for FaCT and DLP

| Concept expression | Simplification |
|---|---|
| $\forall R.\top$ | $\top$ |
| $\sqcap\{\top, C, \ldots\}$ | $\sqcap\{C, \ldots\}$ |
| $\sqcap\{\neg\top, \ldots\}$ | $\neg\top$ |
| $\sqcap\{C, \neg C, \ldots\}$ | $\neg\top$ |

**Table 4.** Lexical simplification rules for FaCT and DLP

# Lexical Normalization & Simplification

- Example: $\exists R.(C \sqcap D) \sqcap \forall R.\neg C,$
  $\sqcap\{\neg(\forall R.\neg\sqcap\{C,D\}),\forall R.\neg C\},$
- Advantages:
  - Easy to implement.
  - Subsumption/satisfiability problems can often be simplified, and sometimes even completely avoided.
  - The elimination of redundancies and the sharing of syntactically equivalent structures may lead to the KB being more compactly stored.
- Disadvantage:
  - For very unstructured KBs there may be no benefit, and it might even slightly increase size of KB.

# Absorption

- General axioms are costly to reason with due to the high degree of non-determinism that they introduce.
  - Eliminate general axioms from the KB whenever possible

- Absorption is a technique that tries to eliminate general inclusion axioms ($C \sqsubseteq D$) by absorbing them into primitive definition axioms.

- Example:

$$\text{geometric-figure} \sqcap \exists \text{angles.three} \sqsubseteq \exists \text{sides.three} \qquad \text{geometric-figure} \sqsubseteq \text{figure}$$

$$\text{geometric-figure} \sqsubseteq \exists \text{sides.three} \sqcup \neg \exists \text{angles.three}$$

$$\text{geometric-figure} \sqsubseteq \text{figure} \sqcap (\exists \text{sides.three} \sqcup \neg \exists \text{angles.three}).$$

# Absorption

- Advantages:
  - It can lead to a dramatic improvement in performance.
  - It is logic and algorithm independent.
- Disadvantage:
  - Overhead required for the pre-processing, although this is generally small compared to classification times.

# Different Optimization Techniques

- ✓ Preprocessing optimizations
  - ✓ Lexical Normalization and Simplification
  - ✓ Absorption
- Partial ordering optimizations
- Satisfiability optimizations
  - Semantic Branching Search
  - Local Simplification
  - Dependency Directed Backtracking
  - Heuristic Guided Search
  - Caching Satisfiability Status

# Optimizing Classification

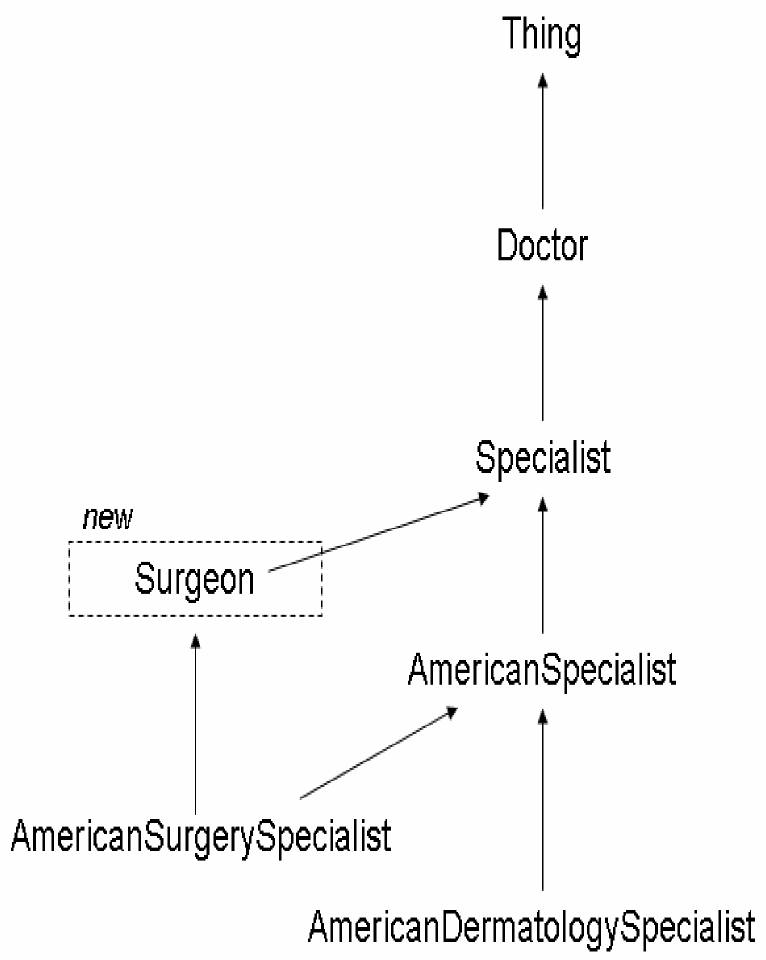
- DL systems are often used to classify a KB, that is to compute a partial ordering or *hierarchy* of named concepts in the KB based on the subsumption relationship.

- Must ensure that the classification process uses the smallest possible number of subsumption tests.

- Algorithms based on traversal of the concept hierarchy
  - Compute a concept's subsumers by searching down the hierarchy from the top node (the *top search* phase)
  - Compute a concept's subsumees by searching up the hierarchy from the bottom node (the *bottom search* phase).

# Optimizing Classification

- Advantages:
  - It can significantly reduce the number of subsumption tests required in order to classify a KB [Baader *et al.*, 1992a].
  - It is logic and algorithm independent.



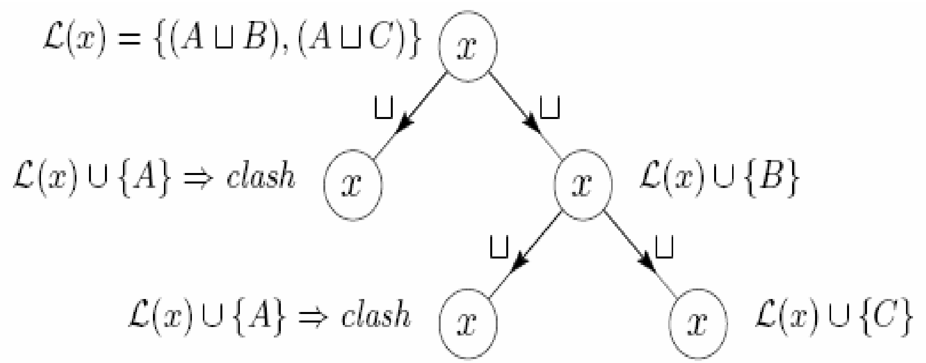

- It is used (in some form) in most implemented DL systems.

# Different Optimization Techniques

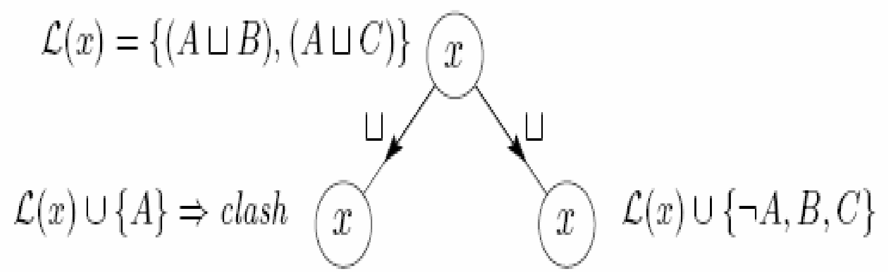
- ✓ Preprocessing optimizations
  - ✓ Lexical Normalization and Simplification
  - ✓ Absorption
- ✓ Partial ordering optimizations
- Satisfiability optimizations
  - Semantic Branching Search
  - Local Simplification
  - Dependency Directed Backtracking
  - Heuristic Guided Search
  - Caching Satisfiability Status

# Semantic Branching Search

- Syntactic branching:
  - Choose a disjunction $(C_1 \sqcup \ldots \sqcup C_n)$
  - Search the different models obtained by adding each of the disjuncts
- Alternative branches of the search tree are not disjoint $\rightarrow$ recurrence of an unsatisfiable disjunct in different branches.
- Semantic branching:
  - Choose a single disjunct *D*
  - Search the two possible search trees obtained by adding *D* or $\neg D$




Syntactic branching search.




Semantic branching search.

# Semantic Branching Search

- Advantages:
  - It is DPLL based. A great deal is known about the implementation and optimization of the this algorithm.
  - It can be highly effective with some problems, particularly randomly generated problems.

- Disadvantages:
  - It is possible that performance could be degraded by adding the negated disjunct in the second branch of the search tree:
    - Example: if the disjunct is a very large or complex concept.
  - Its effectiveness is problem dependent.

# Simplification

- A technique used to reduce the amount of branching in the expansion of node lables:
  - Deterministically expand disjunctions in $\mathcal{L}(x)$ that present only one expansion possibility.
  - Detect a clash when a disjunction in $\mathcal{L}(x)$ has no expansion possibilities.

- Also called boolean constraint propagation (BCP)
  - The inference rule $\frac{\neg C_1, \ldots, \neg C_n, C_1 \sqcup \ldots \sqcup C_n \sqcup D}{D}$ being used to simplify expressions.

# Simplification

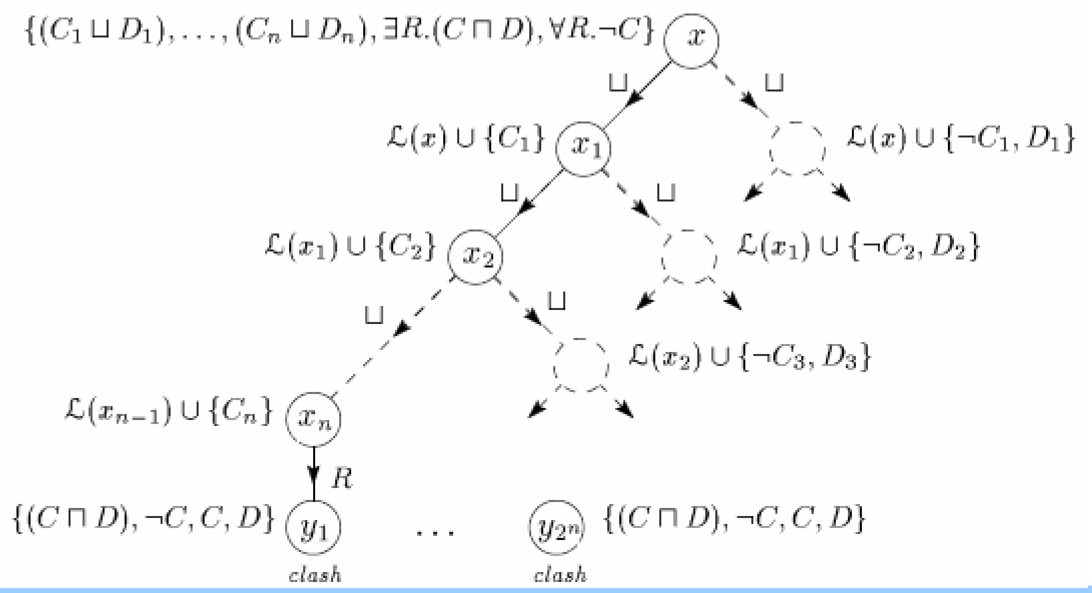
- Example:
  - $\{(C \sqcup (D_1 \sqcap D_2)), (\neg D_1 \sqcup \neg D_2), \neg C\} \subseteq \mathcal{L}(x)$
  - $\neg C \in \mathcal{L}(x) \rightarrow$ deterministically expand $(C \sqcup (D_1 \sqcap D_2)) \rightarrow$ add both $D_1$ and $D_2$ to $\mathcal{L}(x)$
  - Identify $(\neg D_1 \sqcup \neg D_2)$ as a clash
  - ➢ No branching

- Advantages:
  - It is applicable to a wide range of logics and algorithms.
  - It can never increase the size of the search space.

- Disadvantages:
  - It may be costly to perform without using complex data structures [Freeman, 1995].
  - Its effectiveness is relatively limited and problem dependant.
    - Most effective with randomly generated problems, particularly those that are over-constrained.

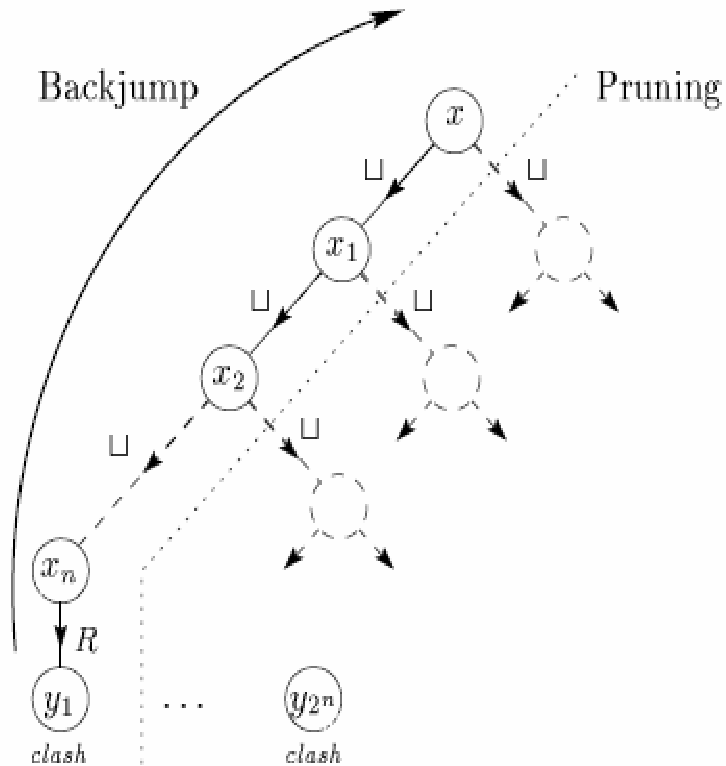
# Dependency Directed Backtracking

- Trashing:
  - Inherent unsatisfiability concealed in sub-problems can lead to large amounts of unproductive backtracking search.
- Example: $\mathcal{L}(x) = \{(C_1 \sqcup D_1), \ldots, (C_n \sqcup D_n), \exists R.(C \sqcap D), \forall R.\neg C\}$



# Dependency Directed Backtracking

- Allows rapid recovery from bad branching choices
- Most commonly used technique is backjumping
  - Tag concepts introduced at branch points
  - Expansion rules combine and propagate tags
  - On discovering a clash, identify most recently introduced concepts involved
  - Jump back to relevant branch points without exploring alternative branches
  - Effect is to prune away part of the search space
- Highly effective — essential for usable system
  - E.g., GALEN KB, 30s (with) $\rightarrow$ months++ (without)



# Dependency Directed Backtracking

- Advantages:
  - It can lead to a dramatic reduction in the size of the search tree and thus a huge performance improvement.
  - The size of the search space can never be increased.

- Disadvantage:
  - The overhead of propagating and storing the dependency sets.

# Heuristic Guided Search

- Guide the search → try to minimize the size of the tree.

- MOMS heuristic:
  - Branch on the disjunct that has the maximum number of occurrences in disjunctions of minimum size → maximizes the effectiveness of BCP

- JW heuristic: (a variant of MOMS)
  - Consider all occurrences of a disjunct, weight them according to the size of the disjunction in which they occur.
  - Select the disjunct with the highest overall weighting.

- Oldest-First heuristic:
  - Use dependency sets to guide the expansion → maximizes the effectiveness of backjumping.
  - Choose a disjunction whose dependency set does not include any recent branching points.

# Heuristic Guided Search

- Example:
  - $\{(C \sqcup D_1), \ldots, (C \sqcup D_n)\} \subseteq \mathcal{L}(x)$
  - When $C$ is added to $\mathcal{L}(x)$, all of the disjunctions are fully expanded
  - When $\neg C$ is added to $\mathcal{L}(x)$, BCP will expand all of the disjunctions
- Advantages:
  - They can be used to complement other optimizations.
  - They can be selected and tuned to take advantage of the kinds of problem that are to be solved (if this is known).
- Disadvantages:
  - They can add a significant overhead.
  - Heuristics can interact adversely with other optimizations.
  - Heuristics designed to work well with purely propositional reasoning may not be particularly effective with DLs, where much of the reasoning is modal.

# Caching

- During a satisfiability test there may be many successor nodes created.
  - These nodes tend to look very similar.
  - Considerable time can be spent re-performing the computations on nodes that end up having the same label.
    - The satisfiability algorithm only cares whether a node is satisfiable or not $\rightarrow$ this time is wasted.
- Successors are only created when other possibilities at a node are exhausted $\rightarrow$ The entire set of concept expressions that come into a node label can be generated at one time.
- The satisfiability status is determined by this set of concept expressions.
  - Other nodes with the same set of initial formulae will have the same satisfiability status $\rightarrow$ saves a considerable amount of processing.

# Caching

- Advantages:
  - It can be highly effective with some problems, particularly those with a repetitive structure.
  - It can be effective with both single satisfiability tests and across multiple tests (as in KB classification).

- Disadvantages:
  - Retaining node labels and their satisfiability status involves a storage overhead.
  - The adverse interaction with dependency directed backtracking
  - Its effectiveness is problem dependent.
    - Highly effective with some hand crafted problems,
    - Less effective with realistic classification problems,
    - Almost completely ineffective with randomly generated problems.

# Outline

- ✓ Introduction
- ✓ Optimization Techniques
- Comparison with Other Systems
- Comparing Optimizations
- Discussion

# Comparison with Other Systems

# Effectiveness of the Optimizations

- Schild has shown that determining subsumption in expressive DLs is equivalent to determining satisfiability of formulae in propositional modal or dynamic logics.

- Four systems were tested:
  - Optimized DL systems:
    - FaCT ✓
    - DPL ✓
  - Unoptimized DL system:
    - KRIS ✓
    - CRACK
  - Heavily-optimized reasoner for propositional modal logics:
    - KSAT ✓

- Neither KRIS nor KSAT can be used on all tests.
  - Neither handle transitive roles.
  - KSAT cannot handle a knowledge base.

# Test Suite 1 - Tableaux’98

- A propositional modal test suite.
- Consists of 9 classes of formulae, in both provable and non-provable forms, for each of **K**, **KT**, and **S4**.
- 21 examples of exponentially increasing difficulty for each class of formula
  - The increase in difficulty is achieved by increasing the modal depth.
- <u>Test methodology</u>: ascertain the number of the largest formula of each type that the system is able to solve within 100 seconds of CPU time.
- Results: FaCT and DLP outperformed the other systems, with DLP being a clear winner.

# Test Suite 1 - Tableaux'98

| | FaCT | | DLP | | KSAT | | Kris | |
|---|---|---|---|---|---|---|---|---|
| **K** | *p* | *n* | *p* | *n* | *p* | *n* | *p* | *n* |
| *branch* | 6 | 4 | 19 | 13 | 8 | 8 | 3 | 3 |
| *d4* | >20 | 8 | >20 | >20 | 8 | 5 | 8 | 6 |
| *dum* | >20 | >20 | >20 | >20 | 11 | >20 | 15 | >20 |
| *grz* | >20 | >20 | >20 | >20 | 17 | >20 | 13 | >20 |
| *lin* | >20 | >20 | >20 | >20 | >20 | 3 | 6 | 9 |
| *path* | 7 | 6 | >20 | >20 | 4 | 8 | 3 | 11 |
| *ph* | 6 | 7 | 7 | 9 | 5 | 5 | 4 | 5 |
| *poly* | >20 | >20 | >20 | >20 | 13 | 12 | 11 | >20 |
| *t4p* | >20 | >20 | >20 | >20 | 10 | 18 | 7 | 5 |
| **KT** | *p* | *n* | *p* | *n* | *p* | *n* | *p* | *n* |
| *45* | >20 | >20 | >20 | >20 | 5 | 5 | 4 | 3 |
| *branch* | 6 | 4 | 19 | 12 | 8 | 7 | 3 | 3 |
| *dum* | 11 | >20 | >20 | >20 | 7 | 12 | 3 | 14 |
| *grz* | >20 | >20 | >20 | >20 | 9 | >20 | 0 | 5 |
| *md* | 4 | 5 | 3 | >20 | 2 | 4 | 3 | 4 |
| *path* | 5 | 3 | 16 | 14 | 2 | 5 | 1 | 13 |
| *ph* | 6 | 7 | 7 | >20 | 4 | 5 | 3 | 3 |
| *poly* | >20 | 7 | >20 | 12 | 1 | 2 | 2 | 2 |
| *t4p* | 4 | 2 | >20 | >20 | 1 | 1 | 1 | 7 |

**Table 5.** Results for **K** and **KT**

| | FaCT | | DLP | |
|---|---|---|---|---|
| **S4** | *p* | *n* | *p* | *n* |
| *45* | >20 | >20 | >20 | >20 |
| *branch* | 4 | 4 | 18 | 12 |
| *grz* | 2 | >20 | >20 | >20 |
| *ipc* | 5 | 4 | 10 | >20 |
| *md* | 8 | 4 | 3 | >20 |
| *path* | 2 | 1 | 15 | 15 |
| *ph* | 5 | 4 | 7 | >20 |
| *s5* | >20 | 2 | >20 | >20 |
| *t4p* | 5 | 3 | >20 | >20 |

**Table 6.** Results for **S4**

- Neither KSAT nor KRIS can be used to perform **S4** satisfiability tests (They can't reason with transitive roles).

# Test Suite 2 – A Set of Random Formulae

- Another propositional modal test suite

- The method uses a random generator to produce formulae. Each formula is a conjunction of $L$ $K$-clauses
  - A $K$-clause is a disjunction of $K$ elements, each element being negated with a probability of 0.5.
  - An element is either:
    - a modal atom of the form $\forall R.C$, where $C$ is a $K$-clause
    - a propositional variable chosen from the $N$ propositional variables that appear in the formula, at the maximum modal depth $D$.

- 2 sets of formulae:
    - **PS12** with $N = 4$, $K = 3$, and $D = 1$
    - **PS13** with $N = 6$, $K = 3$, and $D = 1$

- The test sets are created by varying $L$ from $N$ to $30N$, and generating 100 formulae for each integer value of $L/N$.
    - For SAT problems, when the other parameters are fixed, the value of $L/N$ determines the "hardness" of formulae.

# Test Suite 2 – A Set of Random Formulae

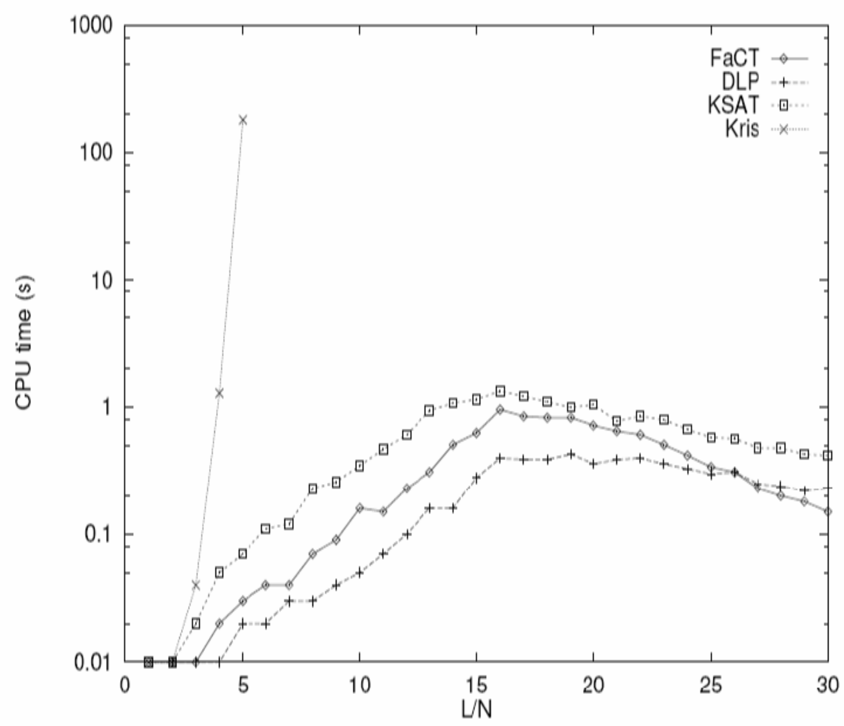



Fig. 5. Median solution times for PS12 formulae

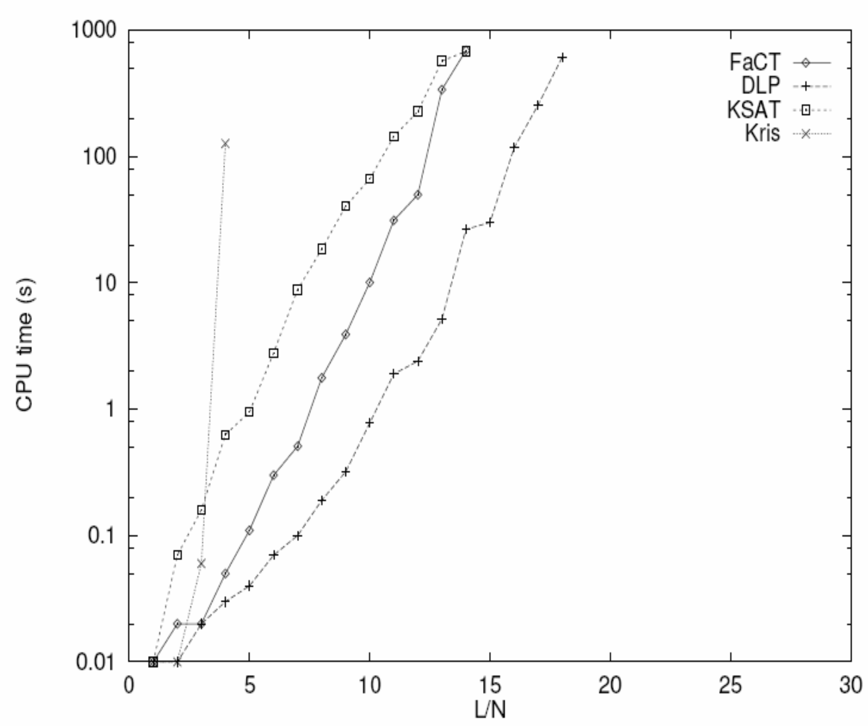



Fig. 6. Median solution times for PS13 formulae

- The performance differences between FaCT, DLP, and KSAT are much less marked
  - With such small number of literals, the purely propositional problems at depth 1 can almost always be solved deterministically.

# Test Suit 3 – Expressive KBs

- Take an expressive knowledge base and construct a version of it that is acceptable to FaCT, DLP, KRIS, and CRACK.

- GALEN knowledge base
  - High level ontology

- Test KB construction:
  - Translate the GRAIL syntax of the GALEN KB into the standard syntax
  - Eliminate concept inclusion axioms by using *absorption*
  - Discard all role axioms

- The resulting KB contains 2,719 named concepts and 413 roles.

# Test Suit 3 – Expressive KBs

- Results:
  - Neither KRIS nor CRACK was able to classify the KB
  - FaCT classified the KB in 211 seconds.
  - DLP did so in 70 seconds
- Testing other KBs:
  - They are too small or too simple

| | FaCT | DLP | KRIS | CRACK |
|---|---|---|---|---|
| Load | 6.03 | — | 135.90 | — |
| Pre-process | 0.85 | — | — | — |
| Classify | 204.03 | — | $\gg$400,000 | $\gg$10,000 |
| Total CPU time (s) | 210.91 | 69.56 | $\gg$400,000 | $\gg$10,000 |

**Table 7.** Classification times for GALEN knowledge base

| Knowledge base | Concepts | FaCT | DLP | KRIS | CRACK | NeoClassic |
|---|---|---|---|---|---|---|
| ckb-roles | 79 | 0.19 | 0.27 | 0.68 | 1.19 | 0.42 |
| datamont-roles | 120 | 0.42 | 0.36 | 0.89 | 1.18 | 0.65 |
| espr-roles | 142 | 0.33 | 0.13 | 0.58 | 0.00 | 0.63 |
| fss-roles | 132 | 0.66 | 0.64 | 1.16 | 0.37 | 0.78 |
| wines | 267 | 4.71 | 2.05 | 2.99 | 2.37 | 2.77 |
| wisber-roles | 140 | 0.48 | 0.78 | 1.03 | 1.63 | 1.03 |

**Table 8.** Classification times for other knowledge bases (CPU seconds)

# Outline

- ✓ Introduction
- ✓ Optimization Techniques
- ✓ Comparison with Other Systems
- ▪ Comparing Optimizations
- ▪ Discussion

# Comparing Optimizations

# Comparing Optimizations

- The comparison with other systems does not show which of the optimizations are most effective.
- Recent versions of DLP have compile-time configuration options.
- 22 configurations – Each was ran over the 3 test suites.
- Results:
  - Test Suite 1:
    Caching – Backjumping – Semantic Branching
  - Test Suite 2:
    - Normalization – Semantic Branching – Backjumping – BCP
  - Test Suite 3:
    - Backjumping – Caching – Semantic Branching
    - Without absorption, satisfiability could not be proved by either FaCT or DLP

# Outline

- ✓ Introduction
- ✓ Optimization Techniques
- ✓ Comparison with Other Systems
- ✓ Comparing Optimizations
- Discussion

# Discussion

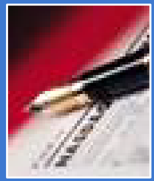
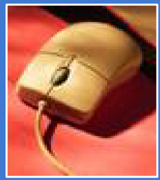
- To be useful in realistic applications, DL systems need both expressive logics and fast reasoners.
- Effective optimization techniques can make a dramatic difference in the performance of knowledge representation systems based on expressive DLs.
- These techniques can operate at every level of a DL system:
  - Simplify the KB,
  - Reduce the number of subsumption tests required to classify it,
  - Substitute tableaux subsumption tests with less costly tests,
  - Reduce the size of the search space resulting from non-deterministic tableaux expansion.

# Discussion

- The most effective of these optimizations are absorption and backjumping:
  - Impose a very small additional overhead,
  - Can dramatically improve typical case performance,
  - Hardly ever degrade performance (to any significant extent).
- Other widely applicable optimizations include normalization, semantic branching and local simplification.
- Various forms of caching can also be highly effective, but they do impose a significant additional overhead in terms of memory usage, and can sometimes degrade performance.
- Heuristic techniques, at least those currently available, are not particularly effective and can often degrade performance.

# Discussion

- Several exciting new application areas for very expressive DLs:
  - Reasoning about DataBase schemata and queries
  - Providing reasoning support for the Semantic Web.
  - ➢ Require logics even more expressive than those implemented in existing systems.
  - ➢ The challenge is to demonstrate that highly optimized reasoners can provide acceptable performance even for these logics.
- Given the immutability of theoretical complexity, no (complete) implementation can guarantee to provide good performance in all cases.
- The objective of optimized implementations is to provide acceptable performance in typical applications.

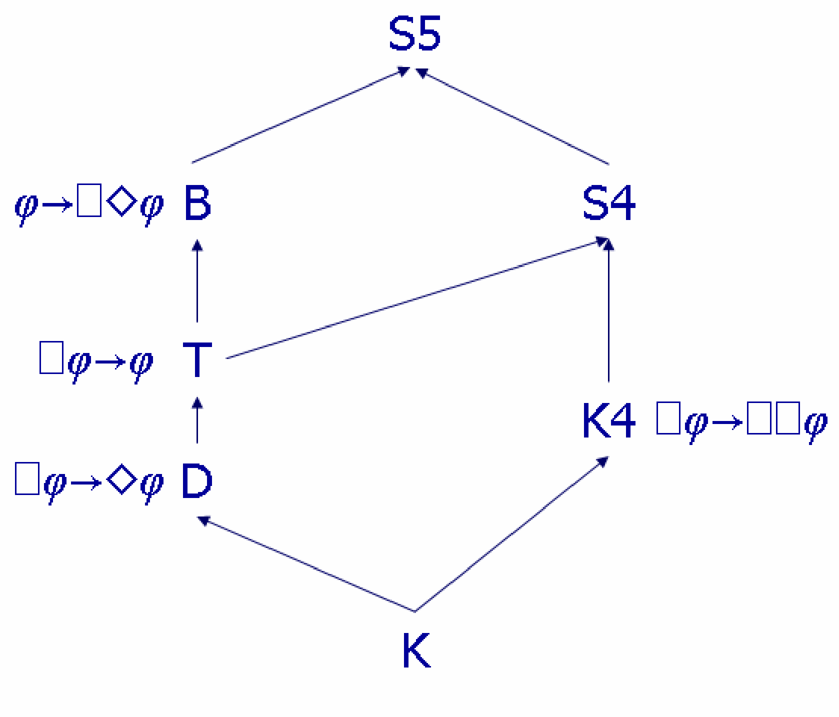




# THANK YOU!!!

# Any Questions ???

# Propositional Modal Logic

- Syntax
  - Propositional logic
  - Modal operators
    - - necessarily (box)
    - ◇ - possibly (diamond)




- K:
  - Necessitation Rule:

    If A is a theorem of K, then so is □A.
  - Distribution Axiom:

    □(A→B) → (□A→□B).

- http://plato.stanford.edu/entries/logic-modal/