Knowledge Compilation: A Sightseeing Tour

Pierre Marquis

Université Lille-Nord de France, Artois
CRIL UMR CNRS 8188
France

Tutorial at ECAI’08, Patras, July 21th, 2008
Part I

Part I: Introduction
What is "Knowledge Compilation"?

- A family of approaches for addressing the **intractability of a number of AI problems**
- Is concerned with **pre-processing** pieces of available information for **improving some tasks from a computational point of view**
- Amounts to a **translation** issue:
  - **Off-line phase**: Turn some pieces of information $\Sigma$ into a compiled form $\text{comp}(\Sigma)$
  - **On-line phase**: Exploit the compiled form $\text{comp}(\Sigma)$ (and the remaining pieces of information $\alpha$) to achieve the task(s) under consideration
Knowledge Compilation: A Recent Research Topic

- Identified as a **research topic in AI** in the recent past
- The name “knowledge compilation” dates back to the late 80’s/beginning of the 90’s (the purpose was to improve propositional reasoning)
- **Many developments** from there
  - From the theoretical side (concepts, algorithms, etc.)
  - From the practical side (benchmarks, pieces of software, applications, etc.)
Pre-processing pieces of information for improving computations is an old idea!

Improving computations means (typically) “saving computation time”

Many applications in Computer Science (even before the modern computer era)
This is not a Book!
A Table of Logarithms!

- A “compiled form” useful for many computations (for more than three centuries)
- \[ \Sigma \subseteq [1, 10) \]
- \( \text{comp}(\Sigma) = \text{pairs } \langle x, \log_{10}(x) \rangle \text{ for each } x \in \Sigma \)
- \( \alpha \) is a description of what is to be computed; for instance \( \sqrt[5]{1234} \)

\[
\sqrt[5]{1234} = (1.234 \times 10^3)\frac{1}{5}
\]

\[
\log_{10}(\sqrt[5]{1234}) = \log_{10}((1.234 \times 10^3)\frac{1}{5})
\]

\[
= \frac{\log_{10}(1.234)+3}{5}
\]

- Look up \( \log_{10}(1.234) \) in the table

\[
\langle 1.234, 0.09131516 \rangle \in \text{comp}(\Sigma)
\]

- Compute \( \frac{\log_{10}(1.234)+3}{5} = \frac{0.09131516+3}{5} = 0.618263032 \)

- Look up the antecedent by \( \log_{10} \) of the resulting value in the table (or use an antilog table)

\[
\langle 4.152054371, 0.618263032 \rangle \in \text{comp}(\Sigma)
\]
Knowledge Compilation: In the More Recent Past

- **In many CS areas**
  - Compiling computer languages
  - Database indexes
  - Lookup tables (e.g. for computer graphics)
  - ...

- **Even in AI**
  - Compiling rule-based systems (Rete algorithm [Forgy, AIJ 1982])
What is "Knowledge"?

- Taken in a rather broad sense (and not necessarily as “true belief”)
- Pieces of information and **ways to exploit** them
- Same meaning as “knowledge” in “knowledge representation”
- Pieces of information are typically encoded as formulas $\Sigma, \alpha, ...$ in a **logic-based language**

$\langle L, \vdash \rangle$
Assumptions

For time reasons I assume **basic knowledge** about

- **Propositional logic:** syntax, semantics, consistency, validity, $|=,$ $\equiv$

- **Computational complexity:** decision and function problems, Turing machine (nondeterministic, with oracle), polynomial reduction, completeness, classes of the polynomial hierarchy PH: P, NP, coNP, etc.
What is ”Exploiting Knowledge”?

- What are the tasks to be computationally improved via knowledge compilation?
- A domain-dependent issue in general
- Typically combinations of basic queries and transformations
Basic Queries and Transformations

- **Queries**
  - **Inference**: Does $\Sigma \vdash \alpha$ hold?
  - **Consistency**: Does there exist $\alpha$ such that $\Sigma \not\vdash \alpha$ holds?
  - ...

- **Transformations**
  - **Conditioning**: Make some elementary propositions true (or false) in $\Sigma$
  - **Closures under connectives**: Compute a representation in $L$ of $\alpha \oplus \beta$ from $\alpha \in L$ and $\beta \in L$
  - **Forgetting**: When defined, compute a representation of the most general consequence w.r.t. $\vdash$ of $\Sigma \in L$ not containing some given elementary propositions
  - ...

Example: Consistency-Based Diagnostic
Example: Consistency-Based Diagnostic

- \( S = (SD, OBS) \) gathers a **system description** \( SD \) and some **observations** \( OBS \)
- \( SD \) describes the behaviour of the system components and how they are connected
  \[
  \neg ab - inv_1 \Rightarrow (out - inv_1 \Leftrightarrow \neg in - inv_1)
  \]
  \[
  \neg ab - inv_2 \Rightarrow (out - inv_2 \Leftrightarrow \neg in - inv_2)
  \]
  \[
  out - inv_1 \Leftrightarrow in - inv_2
  \]
- \( OBS \) describes the inputs and outputs of the system
  \[
  in - inv_1 \land \neg out - inv_2
  \]
- \( \Delta \) is a **consistency-based diagnosis** for \( S \) iff it is a conjunction of \( ab \)-literals such that \( \Delta \land SD \land OBS \) is consistent
  \[
  ab - inv_1, ab - inv_2, ab - inv_1 \land ab - inv_2
  \]
Example: Consistency-Based Diagnostic

- Generating the **consistency-based diagnoses** of a system \( S = (SD, OBS) \)

\[
\text{mod}(\exists (PS \setminus AB).(SD \mid OBS))
\]

\[
\exists (PS \setminus AB).(SD \mid OBS) \equiv ab - inv_1 \lor ab - inv_2
\]

- The task can be viewed as a combination of **conditioning**, **forgetting** and **model enumeration**
When is Knowledge Compilation Useful?

Two conditions are necessary:

- Some pieces of information are more subject to change than others.
- The archetypal inference problem: a set of pairs \( \{ \langle \Sigma, \alpha \rangle \} \)
  - A “knowledge” base \( \Sigma \) (the fixed part)
  - Queries \( \alpha \) about it: \( \alpha_1, \ldots, \alpha_n \) (the varying part)
- Some queries/transformations of interest become “less intractable”, provided that the computational effort spent during the off-line phase is “reasonable”
Evaluating KC: The Problem Level

The theoretical side

- "Less intractable" queries/transformations: removing some sources of complexity (results in decision problems at a lower level of the polynomial hierarchy)
- "Reasonable" pre-processing: the size of the compiled form $\text{comp}(\Sigma)$ is polynomial in the size of $\Sigma$ (remember that the complexity of any algorithm is a function of its input size)
- This size requirement is crucial
Example: Plan Existence in Classical Planning

Propositional STRIPS planning

- \( F = \{p_1, \ldots, p_n\} \) fluents
- \( A = \{act_1, \ldots, act_k\} \) a set of STRIPS(-like) actions
- \( \Sigma = \langle F, A \rangle \)
- \( \alpha = \langle s_0, G \rangle \) (initial state and goal formula over the fluents)
- \( \text{comp}(\Sigma) = \) transition model (state graph) associated to \( \Sigma \)
- PLAN EXISTENCE is \text{PSPACE-complete} when the input is \( \langle \Sigma, \alpha \rangle \)
- PLAN EXISTENCE can be decided \text{in linear time} when the input is \( \langle \text{comp}(\Sigma), \alpha \rangle \)
- We cannot conclude that making the transition model explicit is the right way to solve PLAN EXISTENCE!
- For quite small values of \( n \), \( \text{comp}(\Sigma) \) cannot be generated!
Evaluating KC: The Instance Level

The experimental side

- Refers to a specific compilation function $\text{comp}$
- Consider a set of $n$ instances of the problem, sharing the same fixed part $\Sigma$
- Determine the time/space needed to solve the $n$ instances using a “compiled approach”
- Determine the time/space needed to solve the $n$ instances using a “direct, uncompiled approach”
- Compare the computational resources spent for the two approaches

Computes some statistics summarizing the comparisons for several fixed parts $\Sigma$
The Problem Level vs. The Instance Level

- Two complementary approaches with their own drawbacks
- The problem level: an application corresponds to a/some specific instance(s) of a problem (not always the worse ones!)
- KC can prove useful for some instances of a problem, even if the problem itself is “not compilable”
- The instance level: refers to a specific compilation function \( \text{comp} \) and is more informationally demanding (set of instances, baseline algorithm)
- Can be hard to get a consensus on the set of instances and the baseline algorithm to be chosen
Many works about KC concern **propositional reasoning**

- A “minimal” KR framework in AI is **(classical) propositional logic**
- At the heart of **many other formalisms in AI**
- Sufficiently expressive for a number of AI problems (e.g. diagnostic as above)
- But propositional reasoning typically is **intractable**
  - Inference (entailment) is coNP-complete (even in the clausal case)
  - Consistency is NP-complete (the famous SAT problem)
  - Forgetting is NP-hard
  - ...
- As such propositional reasoning is a **good candidate** for KC
Several Key Issues

- How to **compile** propositional formulas in order to improve clausal entailment?
- How to **evaluate** from a theoretical point of view whether KC can prove useful?
- How to **choose** a target language for the KC purpose?
- ...
Outline

- Part II: Some Propositional Languages
- Part III: The Clausal Entailment Problem
- Part IV: The Compilability Issue
- Part V: The Knowledge Compilation Map
- Part VI: Conclusion
Part II

Part II: Some Propositional Languages
A DAG Language: **NNF**

- Let PS be a denumerable set of propositional variables (atoms)

- A **formula in NNF** is a **rooted, directed acyclic graph (DAG)** where:
  - each leaf node is labeled with *true, false, x* or \( \neg x \), \( x \in PS \)
  - each internal node is labeled with \( \land \) or \( \lor \) and can have arbitrarily many children

- **NNF** formulas which do not have a tree-like structure can be viewed as **compact representations** of the corresponding formulas having a tree shape, obtained by sharing subformulas

- Two families of subsets of **NNF** (fragments): **flat** ones \( f\text{-NNF} \) (height at most 2) and **nested** fragments
An NNF Formula
Imposing some Properties leads to Interesting Flat Fragments

- **Simple-disjunction**: The children of each or-node are leaves that share no variables (the node is a clause)
- **Simple-conjunction**: The children of each and-node are leaves that share no variables (the node is a term)
Well-Known Flat Fragments

- The language $\text{CNF}$ is the subset of $\text{f-NNF}$ satisfying simple–disjunction.
- The language $\text{DNF}$ is the subset of $\text{f-NNF}$ satisfying simple–conjunction.
Other Flat Fragments

- The language $\mathbf{PI}$ is the subset of $\mathbf{CNF}$ in which each clause entailed by the formula is entailed by a clause that appears in the formula; and no clause in the formula is entailed by another.

- The language $\mathbf{IP}$ is the subset of $\mathbf{DNF}$ in which each term entailing the formula entails some term which appears in the formula; and no term in the formula is entailed by another term.
Example: Flat Fragments

\[ (\neg a \lor b \lor c) \land (\neg b \lor d) \land (\neg c \lor d) \] is a \textcolor{red}{\text{CNF}} formula

\[ (\neg a \land \neg b \land \neg c) \lor (b \land d) \lor (c \land d) \] is a \textcolor{red}{\text{DNF}} formula

\[ (\neg a \lor b \lor c) \land (\neg b \lor d) \land (\neg c \lor d) \land (\neg a \lor d) \] is a \textcolor{red}{\text{PI}} formula

\[ (\neg a \land \neg b \land \neg c) \lor (b \land d) \lor (c \land d) \lor (\neg a \land d) \] is a \textcolor{red}{\text{IP}} formula
Another Flat Fragment: HORN–CNF

- The language HORN–CNF is the subset of CNF formulas consisting of conjunctions of clauses containing at most one positive literal
- \((\neg a \lor b) \land (\neg b \lor d) \land (\neg c \lor d)\) is a HORN–CNF formula
- Unlike CNF, DNF, PI, IP, it does not allow for the representation of every formula (e.g. \(a \lor b\))
Let $L$ be a subset of $\text{NNF}$.

$L$ satisfies $\text{CO}$ if and only if there exists a polynomial time algorithm that maps every formula $\Sigma$ from $L$:

- to 1 if $\Sigma$ is consistent,
- and to 0 otherwise.
Let $L$ be a subset of $\text{NNF}$.

$L$ satisfies $\text{CD}$ if and only if there exists a polynomial time algorithm that maps every formula $\Sigma$ from $L$ and every consistent term $\gamma$ to a formula from $L$ that is logically equivalent to $\Sigma | \gamma$.

$\Sigma | \gamma$ is the formula obtained by replacing in $\Sigma$ every occurrence of a variable $x \in \text{Var}(\gamma)$ by $true$ if $x$ is a positive literal of $\gamma$ and by $false$ if $\neg x$ is a negative literal of $\gamma$. 
Let $L$ be a subset of $\text{NNF}$.

$L$ satisfies CE if and only if there exists a polynomial time algorithm that maps every formula $\Sigma$ from $L$ and every clause $\gamma$ from $\text{NNF}$:

- to 1 if $\Sigma \models \gamma$ holds,
- and to 0 otherwise.
Flat Fragments and \textbf{CO, CD, CE}

- Like \texttt{DNF, PI, IP, HORN–CNF} satisfies the \textbf{CO} query and the \textbf{CD} transformation, hence the \textbf{CE} query
- \textbf{Unit resolution} proves enough for \textbf{CO} in the \texttt{HORN–CNF} fragment
Imposing some Properties leads to Interesting Nested Fragments

- Decomposability
- Determinism
- Decision
- Ordering
Decomposability and Determinism

- **Decomposability**: An and-node $C$ is decomposable if and only if the conjuncts of $C$ do not share variables. That is, if $C_1, \ldots, C_n$ are the children of and-node $C$, then $\text{Var}(C_i) \cap \text{Var}(C_j) = \emptyset$ for $i \neq j$. An NNF formula satisfies the decomposability property if and only if every and-node in it is decomposable.

- **Determinism**: An or-node $C$ is deterministic if and only if each pair of disjuncts of $C$ is logically contradictory. That is, if $C_1, \ldots, C_n$ are the children of or-node $C$, then $C_i \land C_j \models \text{false}$ for $i \neq j$. An NNF formula satisfies the determinism property if and only if every or-node in it is deterministic.
Figure: A formula in NNF. The marked node is decomposable.
Determinism

Figure: A formula in NNF. The marked node is deterministic.
Decision and Ordering

**Decision**: A decision node $N$ in an $\text{NNF}$ formula is one which is labeled with $\text{true}$, $\text{false}$, or is an or-node having the form $(x \land \alpha) \lor (\neg x \land \beta)$, where $x$ is a variable, $\alpha$ and $\beta$ are decision nodes. In the latter case, $d\text{Var}(N)$ denotes the variable $x$. An $\text{NNF}$ formula satisfies the decision property when its root is a decision node.

**Ordering**: Let $<$ be a total, strict ordering over the variables from $PS$. An $\text{NNF}$ formula satisfying the decision property satisfies the ordering property w.r.t. $<$ if and only if the following condition is satisfied: if $N$ and $M$ are or-nodes, and if $N$ is an ancestor of node $M$, then $d\text{Var}(N) < d\text{Var}(M)$. 
Some Nested Fragments

- The language $\text{DNNF}$ is the subset of $\text{NNF}$ of formulas satisfying decomposability
- The language $\text{d-DNNF}$ is the subset of $\text{NNF}$ of formulas satisfying decomposability and determinism
- The language $\text{OBDD}_<$ is the subset of $\text{NNF}$ of formulas satisfying decomposability, decision and ordering
Figure: On the left part, a formula in the $\text{OBDD}_<$ language. On the right part, a more standard notation for it.
Nested Fragments and CO, CD, CE

DNNF, d-DNNF, OBDD< satisfies:

- the CO query
- the CD transformation
- the CE query
Part III

Part III: The Clausal Entailment Problem
Knowledge Compilation for the Clausal Entailment Problem

How to decide classical entailment $\Sigma \models \alpha$ in a more efficient way assuming that $\Sigma$ is compiled during an off-line phase and that queries $\alpha$ are CNF formulas?

- Restricting queries is necessary since deciding $\models \alpha$ is already coNP-complete for unrestricted queries
- The CNF format for queries is a reasonable assumption (complete and conjunctive)
Exact vs. Approximate KC

- **Exact KC**: any clause $\gamma$ is taken into account as a possible query
- **Approximate KC**: a proper subset $S$ of all clauses $\gamma$ is taken into account
  - $S$ is specified a priori: all clauses over $V \subset PS$, all clauses of length at most $k$, etc.
  - $S$ is not specified a priori (defined by the approximation)
Let $\text{comp}(\Sigma) = \langle \Sigma_l, \Sigma_u \rangle$ s.t. $\Sigma_l \models \Sigma \models \Sigma_u$ and $\Sigma_l, \Sigma_u$ belong to a fragment $L$ satisfying $CE$:

- If $\Sigma_u \models \gamma$ then $\Sigma \models \gamma$
- If $\Sigma_l \not\models \gamma$ then $\Sigma \not\models \gamma$
- $S$ is not specified a priori
Approximate KC: Horn Approximations

Selman and Kautz [AAAI’91, JACM96]

- $L = \text{HORN-CNF}$
- A Horn LB $\Sigma_l$ of $\Sigma$ is a HORN-CNF formula s.t. $\Sigma_l \models \Sigma$
- A Horn GLB $\Sigma_{glb}$ of $\Sigma$ is a Horn LB of $\Sigma$ that is maximal w.r.t. $\models$
- A Horn UB $\Sigma_u$ is a HORN-CNF formula s.t. $\Sigma \models \Sigma_u$
- A Horn LUB $\Sigma_{lub}$ of $\Sigma$ is a Horn UB of $\Sigma$ that is minimal w.r.t. $\models$
- Horn LUBs and Horn GLBs must be preferentially used to improve query-answering (maximise $S$):

$$\text{comp}(\Sigma) = \langle \Sigma_{glb}, \Sigma_{lub} \rangle$$
Horn LUBs and GLBs

- The Horn LUB of $\Sigma$ is unique up to logical equivalence but it has no representation of size polynomial in the size of $\Sigma$ in the worst case.
- Every Horn consequence $\gamma$ of $\Sigma$ is a consequence of the Horn LUB.
- There can be exponentially many Horn GLBs but Horn GLBs have representations of size linear in the size of $\Sigma$. 

Computing Horn GLBs

\[ \Sigma \text{ CNF} = \gamma_1 \land \ldots \land \gamma_n \]

- **A Horn strengthening** \( \gamma_s \) of a clause \( \gamma \) is a Horn clause \( \gamma \) that is maximal w.r.t. \( \models \) among the Horn clauses which imply \( \gamma \)

- **A Horn strengthening** \( \gamma_{s1} \land \ldots \land \gamma_{sn} \) of a CNF formula \( \Sigma = \gamma_1 \land \ldots \land \gamma_n \) is a HORN–CNF formula s.t. for \( i \in 1 \ldots n \), \( \gamma_{si} \) is a Horn strengthening of \( \gamma_i \)

One can prove that every Horn GLB of \( \Sigma \) is logically equivalent to a Horn strengthening of it
Computing Horn GLBs

**Input:** a CNF $\Sigma$

**Output:** a Horn GLB of $\Sigma$

$L \leftarrow$ the lexicographically first Horn-strengthening of $\Sigma$

while a lexicographically next Horn-strengthening $L'$ of $\Sigma$ exists do
  if $L \models L'$ then $L \leftarrow L'$

endwhile

remove subsumed clauses from $L$

Return$(L)$
Computing Horn LUBs

Input: a CNF $\Sigma = \Sigma_H$ (Horn subset) $\land \Sigma_{\overline{H}}$ (non-Horn)
Output: the Horn LUB of $\Sigma$

while a resolvent $\gamma$ of a clause of $\Sigma_H$ and a clause of $\Sigma_{\overline{H}}$ that is not subsumed by any clause from $\Sigma_H \cup \Sigma_{\overline{H}}$ exists do
    remove from $\Sigma_H \cup \Sigma_{\overline{H}}$ every clause subsumed by $\gamma$
    if $\gamma$ is Horn then
        add $\gamma$ to $\Sigma_H$
    else
        add $\gamma$ to $\Sigma_{\overline{H}}$
    endif
endwhile
Return($\Sigma_H$)
Example: Horn Approximations

- \( \Sigma = (\neg a \lor b \lor c) \land (\neg b \lor d) \land (\neg c \lor d) \) is a \( \text{CNF} \) formula
- \( \Sigma_{glb} = (\neg a \lor b) \land (\neg b \lor d) \land (\neg c \lor d) \) is a Horn GLB of \( \Sigma \)
- \( \Sigma_{lub} = (\neg b \lor d) \land (\neg c \lor d) \land (\neg a \lor d) \) is the Horn LUB of \( \Sigma \)
- \( \Sigma \not\models \neg a \lor c \) since \( \Sigma_{glb} \not\models \neg a \lor c \)
- \( \Sigma \models \neg a \lor b \lor d \) since \( \Sigma_{lub} \models \neg a \lor b \lor d \)
- \( \neg a \lor b \lor c \not\in S \)
Further Readings

Other fragments are targeted (propositional and FOL):

- del Val [IJCAI’95, AAAI’96, AAAI’99, AIJ05]
- Boufkhad [AAAI’98]
- Simon and del Val [IJCAI’01]
Many \textit{comp} functions and the corresponding targeted (complete) fragments satisfying \textbf{CE} \\
\textit{DNNF, d–DNNF, DNF, OBDD}_<, \textit{PI, IP, etc.}

Focus on simple fragments: \textit{PI} and \textit{DNF}

Separability properties

Theory reasoning for KC: \\
\begin{itemize}
  \item Theory prime implicates
  \item Tractable covers
\end{itemize}
Query-Answering from $\mathcal{PI}$ Formulas

Reiter and de Kleer [AAAI’87]

$\text{comp}(\Sigma) = \mathcal{PI}(\Sigma)$

- A linear-time query answering algorithm exists for querying $\mathcal{PI}$ formulas with clausal queries
- $\Sigma \models \gamma$ iff there exists $\delta \in \mathcal{PI}(\Sigma)$ s.t. $\delta \models \gamma$
- $\Sigma$ is equivalent to the conjunction of its prime implicates
- The prime implicates form of $\Sigma$ is unique (when clauses are considered up to logical equivalence)
- $|\mathcal{PI}(\Sigma)|$ can be exponential in $|\Sigma|$ for some $\Sigma$, like $\bigvee_{i=0}^{n-1} (x_{2i} \land x_{2i+1})$
Computing $PI$ Formulas

- Dozen algorithms (the first ones date back to the 50’s)
- Consider various inputs (any formula, a $CNF$ one, a $DNF$ one, etc.)
- Incremental vs. non-incremental algorithms
- **Resolution-based algorithms**: $\delta \in PI(\Sigma)$ iff $\delta$ is one of the logically strongest clauses which can be derived from a $CNF$ formula $\Sigma$ using a resolution strategy complete in consequence-finding
- **Distribution-based algorithms**:

\[
PI(\Sigma_1 \lor \ldots \lor \Sigma_k) = \min(\{\delta_1 \lor \ldots \lor \delta_k \mid \delta_i \in PI(\Sigma_i)\}, \models)
\]
Computing Prime Implicates: Tison’s Algorithm

Input: a CNF formula $\Sigma$
Output: $PI(\Sigma)$

$PI \leftarrow \Sigma$

foreach $x \in Var(\Sigma)$ do

    foreach pair of clauses $x \lor \gamma_1, \lnot x \lor \gamma_2$ of $PI$ do

        if $\gamma_1 \lor \gamma_2$ is not entailed by any clause of $PI$ then

            remove from $PI$ every clause entailed by $\gamma_1 \lor \gamma_2$

            $PI \leftarrow PI \land (\gamma_1 \lor \gamma_2)$

        endif

    endforeach

endforeach

Return $PI$
Query-Answering from \( \text{DNF} \) Formulas

Castell [ICTAI’96], Schrag [AAAI’96]

\[
\text{comp}(\Sigma) = \text{a DNF formula equivalent to } \Sigma
\]

- A **linear-time query answering** algorithm exists for querying \( \text{DNF} \) formulas with clausal queries

\[
\delta_1 \lor \ldots \lor \delta_k \models \gamma \iff \gamma \text{ is a valid clause or for every } i \in 1 \ldots k, \delta_i \text{ contains a literal occurring in } \gamma
\]

- Many different \( \text{DNF} \) formulas equivalent to \( \Sigma \) may exist

- But **none of size polynomial** in \(|\Sigma|\) for some \( \Sigma \), like

\[
\bigwedge_{i=0}^{n-1} (\neg x_{2i} \lor x_{2i+1})
\]

- The smallest ones contains only **prime implicants** of \( \Sigma \), i.e., terms maximal w.r.t. \( \models \) among those implying \( \Sigma \)

- **Adaptation of DPLL algorithms** can be used for computing \( \text{DNF} \) formulas
A DPLL-like Algorithm for Computing \textbf{DNF} Formulas

Essentially DPLL but \textbf{search the whole DPLL tree} and \textbf{store the implicants found}; use only equivalence-preserving filtering rules (no pure literal rule)

\textbf{Input: } a \textbf{CNF} formula $\Sigma$

\textbf{Output: } a \textbf{DNF} formula equivalent to $\Sigma$

\text{DNF} & $\leftarrow$ $\emptyset$

DPLL$^*$($\Sigma$, $\emptyset$)

\textbf{Return}(\textbf{DNF})
**DPLL*\**

**Input:** a CNF formula $\Sigma$, a partial assignment (term) $\tau$

**Output:** false

if $\Sigma$ is the empty set of clauses then

$$\text{DNF} \leftarrow \text{DNF} \cup \{\tau\}$$

endif

Return($false$)

if $\Sigma$ contains $\square$ then Return($false$)

UNIT-PROPAGATE($\Sigma$, $\tau$)

select a branching literal $l$ using a branching rule

DPLL*(simp($\Sigma$, $\tau \cup \{l\}$))

DPLL*(simp($\Sigma$, $\tau \cup \{\neg l\}$))
DPLL* can be improved to compute prime implicants covers, i.e., DNF formulas $\text{comp}(\Sigma)$ in which each term is a prime implicant of $\Sigma$

Interesting since DNF of minimal size are prime implicants covers

Before adding a term $\tau$ to DNF, extract a prime implicant of $\Sigma$ from it by removing literals in a greedy fashion

Use such prime implicants to prune the search space
The Conjunctive Separability Property

- **Conjunctive separability:**
  A clause $\gamma$ is a logical consequence of a conjunctive formula $\alpha_1 \land \ldots \land \alpha_n$ if and only if there exists $i \in 1 \ldots n$ s.t.
  $\gamma$ is a logical consequence of $\alpha_i$

- Satisfied by $\Pi$ formulas and by $\text{DNNF}$ formulas
The Disjunctive Separability Property

- **Disjunctive separability:**
  
  A formula $\alpha$ is a logical consequence of a disjunctive formula $\alpha_1 \lor \ldots \lor \alpha_n$ if and only if for every $i \in 1 \ldots n$ s.t. $\alpha$ is a logical consequence of $\alpha_i$.

- Satisfied by every NNF formula
The Power of Separability Properties

- The two separability properties **underly many target fragments** for KC satisfying CE, especially DNNF and its subsets d-DNNF, OBDD<, DNF
- Formulas from some of those fragments can be characterized by considering **Shannon decompositions** of formulas:

\[(\neg x \land (\Sigma \mid \neg x)) \lor (x \land (\Sigma \mid x))\]

Such a formula is equivalent to \(\Sigma\) and **exhibits several separable subformulas**
- Shannon decompositions can be computed **using DPLL-like algorithms**
An **old idea** in automated reasoning: theory resolution, unification modulo A/C, CLP(R), SMT, etc.

Often the theory is exogeneous (and not explicitly represented)

Not here!

**Key idea:** Consider any \( \Phi \) s.t. \( \Sigma \models \Phi \) and \( \Phi \) belongs to a fragment satisfying **CE**
Marquis [IJCAI’95]
Let $\Sigma$, $\Phi$ be two propositional formulas

- $\Sigma \models_{\Phi} \gamma$ iff $\Sigma \land \Phi \models \gamma$
- A theory implicate of $\Sigma$ w.r.t. $\Phi$ is a clause $\gamma$ s.t. $\Sigma \models_{\Phi} \gamma$
- A theory prime implicate (tpi) of $\Sigma$ w.r.t. $\Phi$ is a theory implicate $\delta$ of $\Sigma$ w.r.t. $\Phi$ that is minimal w.r.t. $\models_{\Phi}$
- $TPI(\Sigma, \Phi)$ is the set of all tpis of $\Sigma$ w.r.t. $\Phi$ (assuming that one representative per equivalence class is kept, only)
Properties of Theory Prime Implicates

- The theory prime implicates form of $\Sigma$ is **unique** (when clauses are considered up to $\Phi$-equivalence).
- The prime implicates of $\Sigma$ are its theory prime implicates w.r.t. $true$.
- Accordingly **exponentially many tpis** may exist.
- A **covering property** holds: $\Sigma \models_\Phi \gamma$ holds iff there exists a tpi $\delta$ of $\Sigma$ w.r.t. $\Phi$ s.t. $\delta \models_\Phi \gamma$.
- $\Sigma$ is $\Phi$-equivalent to the conjunction of all its tpis w.r.t. $\Phi$.
- Strengthening $\Phi$ may only **decrease** the number of tpis of $\Sigma$ w.r.t. $\Phi$. 

Query-Answering from Theory Prime Implicates

\[ \text{comp}(\Sigma) = \langle TPI(\Sigma, \Phi), \Phi \rangle \]

- \( \Sigma \models_\Phi \gamma \) iff \( \exists \delta \in TPI(\Sigma, \Phi), \delta \models_\Phi \gamma \)
- \( \delta \models_\Phi \gamma \) iff \( \forall l \in \delta, \Phi \models \neg l \lor \gamma \)
- If \( \Sigma \models \Phi \) and \( \Phi \) belongs to a fragment satisfying CE, then determining whether \( \Sigma \models \gamma \) holds can be \textbf{achieved in polynomial time} from \text{comp}(\Sigma)
- Many \( \Phi \) can be selected or computed (select the subset of Horn clauses from \( \Sigma \), compile a subset of \( \Sigma \), etc.)
Example: Theory Prime Implicates

- $\Sigma = (a \lor b \lor d) \land (a \lor e) \land (b \lor e) \land (b \lor d \lor f) \land \Phi$
- $\Phi = (\neg a \lor b) \land (\neg b \lor c) \land (\neg e \lor f)$
- $a \lor e$ and $b \lor d$ are the tpis of $\Sigma$ w.r.t. $\Phi$ (up to $\Phi$-equivalence)
- $\Sigma \models b \lor f$ since $a \lor e \models_{\Phi} b \lor f$
Computing Theory Prime Implicates

- We have \( TPI(\Sigma, \Phi) = \min(PI(\Sigma \land \Phi), \models \Phi) \)
- **Algorithms for computing pis** (like directional resolution, Tison’s consensus algorithm) can be used to generate tpis
- Mainly it suffices to replace entailment tests by entailment tests w.r.t. \( \models \Phi \)
- **More sophisticated algorithms** that are based on a distribution property or on Shannon’s decomposition and do not require the generation of pis also exist (Marquis and Sadaoui [AAAI’96])
Use several classes of theories instead of a preset theory

Let $\tau$ be a partial assignment (term) s.t. $\text{simp}(\Sigma, \tau)$ belongs to a fragment $L$ satisfying $\text{CE}$

$\Sigma$ and $\text{simp}(\Sigma, \tau)$ are $\tau$-equivalent

For a clause $\gamma$, determining whether $\tau \land \text{simp}(\Sigma, \tau) \models \gamma$ can be done in polynomial time:

- **conjunctive separability** $\tau \land \text{simp}(\Sigma, \tau) \models \gamma$ iff $\tau \models \gamma$ or $\text{simp}(\Sigma, \tau) \models \gamma$

- **theory reasoning**: each $\text{simp}(\Sigma, \tau) \in L$ is a tractable theory ($L$ satisfies $\text{CE}$)
Tractable Cover Compilations

- **disjunctive separability**: search for a valid DNF \( \tau_1 \lor \ldots \lor \tau_k \) s.t. each \( \text{simp}(\Sigma, \tau_i) \) \((i \in 1 \ldots k)\) belongs to a fragment satisfying CE in order to derive a tractable cover \( \bigvee_{i=1}^{k} (\tau_i \land \text{simp}(\Sigma, \tau_i)) \) of \( \Sigma \)

- \( \Sigma \equiv (\tau_1 \lor \ldots \lor \tau_k) \land \Sigma \equiv \bigvee_{i=1}^{k} (\tau_i \land \Sigma) \equiv \bigvee_{i=1}^{k} (\tau_i \land \text{simp}(\Sigma, \tau_i)) \)

- \( \text{comp}(\Sigma) = \langle \tau_1 \lor \ldots \lor \tau_k, \Sigma \rangle \): storing \( \tau_1, \ldots, \tau_k \) and \( \Sigma \) is sufficient since \( \text{simp} \) is a linear time function

- Nevertheless \( k \) can be exponentially large in the worst case

- DNF compilations correspond to the specific case when the only fragment \( L \) under consideration is the singleton consisting of the empty conjunction of clauses

- Tractable covers can be computed using DPLL-like algorithms
Example: Tractable Cover Compilations

- Let \( \Sigma = (a \lor b \lor c) \land (\neg a \lor \neg b \lor \neg c) \)

- \( \text{comp}(\Sigma) = \langle a \lor (\neg a \land b) \lor (\neg a \land \neg b), \Sigma \rangle \) represents a tractable cover of \( \Sigma \)

- The \text{HORN-CNF} fragment is targeted
  
  - \( \text{simp}(\Sigma, a) = (\neg b \lor \neg c) \)
  
  - \( \text{simp}(\Sigma, \neg a \land b) = \text{true} \)
  
  - \( \text{simp}(\Sigma, \neg a \land \neg b) = c \)

- \([a \land (\neg b \lor \neg c)] \lor [\neg a \land b] \lor [\neg a \land \neg b \land c] \) is a \text{HORN-CNF}[\lor] formula equivalent to \( \Sigma \)
The Size of the Compiled Form

- For all target fragments for KC considered before, there exist $\Sigma$ such that $|\text{comp}(\Sigma)|$ is exponential in $|\Sigma|$
- It is unlikely that one can find better fragments with this respect
- Hence there is no guarantee that KC can prove useful for every formula
- This does not prevent it from being useful for some formulas
- Having several target fragments satisfying CE is interesting (some of them are incomparable one another w.r.t. succinctness)
Further Readings

- del Val [KR’94]
- Mathieu and Delahaye [JELIA’90, TCS94]
- Roussel and Mathieu [Inf.Comp.00]
- Hai and Jigui [JAR04]
- ...


Part IV

Part IV: The Compilability Issue
The Compilability Issue

Cadoli et al. [AIJ96, AI Comm.98, AIJ99, JAIR00, Inf.Comp.02]
Liberatore [Ph.D.98, JACM01]

▸ Evaluating KC at the problem level

▸ Intuition: A (decision) problem is compilable to a complexity class $C$ if it is in $C$ once the fixed part $\Sigma$ of any instance has been pre-processed, i.e., turned off-line into a data structure of size polynomial in $|\Sigma|$

▸ Several compilability classes organized into hierarchies (which echo PH)

▸ Enable to classify problems as compilable to $C$, or as non-compilable to $C$ (usually under standard assumptions of complexity theory)
Decision Problems = Languages $L$ of Pairs

- $\langle \Sigma, \alpha \rangle \in L$
- $\Sigma$: The fixed part
- $\alpha$: The varying part
- **Example:**

  \[
  \text{CLAUSE ENTAILMENT} = \{ \langle \Sigma, \alpha \rangle \mid \Sigma \text{ a NNF formula and } \alpha \text{ a clause s.t. } \Sigma \models \alpha \}\]
C = a complexity class closed under polynomial reductions and admitting complete problems for such reductions

A language of pairs \( L \) belongs to \( \text{compC} \) if and only if there exists a polysize function \( \text{comp} \) and a language of pairs \( L' \in C \) such that for every pair \( \langle \Sigma, \alpha \rangle, \langle \Sigma, \alpha \rangle \in L \) iff \( \langle \text{comp}(\Sigma), \alpha \rangle \in L' \)

For every admissible complexity class \( C \), we have the inclusion \( C \subseteq \text{compC} \)
compC

- Membership to compC: Follow the definition!
- Non-membership to compC: A more complex issue in general
- Classes C/poly are useful
Advice-Taking Turing Machines

- An **advice-taking Turing machine** is a Turing machine that has associated with it a special “advice oracle” $A$, which can be any function (not necessarily a recursive one).
- On input $s$, a special “advice tape” is automatically loaded with $A(|s|)$ and from then the computation proceeds as normal, based on the two inputs, $s$ and $A(|s|)$.
C/poly

- An advice-taking Turing machine uses **polynomial advice** if its advice oracle $A$ is **polysize**
- If $C$ is a class of languages defined in terms of resource-bounded Turing machines, then $C/poly$ is the class of languages defined by Turing machines with the same resource bounds but augmented by polynomial advice
- $C/poly$ contains all languages $L$ for which there exists a polysize function $A$ from $N$ to the set of strings s.t. the language $\{\langle A(|s|), s \rangle \mid s \in L\}$ belongs to $C$
P/poly vs. PH

Karp and Lipton [ACM STOC’98], Yap [TCS83]

- If $\text{NP} \subseteq \text{P/poly}$ then $\Pi_2^p = \Sigma_2^p$ (hence $\text{PH}$ collapses at the second level)
- If $\text{NP} \subseteq \text{coNP/poly}$ then $\Pi_3^p = \Sigma_3^p$ (hence $\text{PH}$ collapses at the third level)
Kautz and Selman [AAAI’92]

- Let $n$ be any non-negative integer
- Let $\Sigma_{n}^{\text{max}}$ be the CNF formula
  \[
  \bigwedge_{\gamma_i \in 3 - C_n} \neg \text{holds}_i \lor \gamma_i
  \]

- $3 - C_n$ is the set of all 3-literal clauses that can be generated from $\{x_1, \ldots, x_n\}$ and the $\text{holds}_i$ are new variables, not among $\{x_1, \ldots, x_n\}$
- $|\Sigma_{n}^{\text{max}}| \in \mathcal{O}(n^3)$
CLAUSE ENTAILMENT $\not\in$ compP

- Each 3-CNF formula $\alpha_n$ built up from the set of variables $\{x_1, \ldots, x_n\}$ is in bijection with the subset $S_{\alpha_n}$ of the variables $holds_i$ s.t. $\gamma_i$ is a clause of $\alpha_n$ if and only if $holds_i \in S_{\alpha_n}$

- $\alpha_n$ is unsatisfiable iff

$$\Sigma_n^{max} \models \gamma_{\alpha_n} = \bigvee_{holds_i \in S_{\alpha_n}} \neg holds_i$$
CLAUSE ENTAILMENT $\not\in$ compP

- Assume that we have a polysize compilation function $comp$ such that determining whether $comp(\Sigma) \models \gamma \in P$

- Then $3$-SAT $\in$ P/poly:
  - Let $\alpha$ be a $3$–CNF formula
  - If $|\text{Var}(\alpha)| = n$, then the machine loads

  \[ A(n) = comp(\Sigma_n^{\text{max}}) \]

  - Finally it determines in polynomial time whether

  $comp(\Sigma_n^{\text{max}}) \models \gamma_\alpha$

- Since $3$-SAT is complete for NP, this would imply NP $\subseteq$ P/poly

- Works also if $comp$ is not exact but computes a representation of the Horn LUB of its input
The Impact of the Language of Varying Parts

- If for each fixed part $\Sigma$, there are only \textit{polynomially many} possible varying parts $\alpha$, then the corresponding language of pairs $L$ belongs to $\text{compP}$

- During the off-line phase, consider successively every $\alpha$ and store it in a lookup-table $\text{comp}(\Sigma)$ whenever $\langle \Sigma, \alpha \rangle$ belongs to $L$

- For every $\Sigma$, $|\text{comp}(\Sigma)|$ is polynomially bounded in the size of $\Sigma$ and determining on-line whether $\langle \Sigma, \alpha \rangle \in L$ amounts to a lookup operation
Example: LITERAL ENTAILMENT $\in \text{compP}$

- $\{\langle \Sigma, \alpha \rangle \mid \Sigma \in \text{NNF}, \alpha \in L_{\text{Var}}(\Sigma), \Sigma \models \alpha \} \in \text{compP}$
- LITERAL ENTAILMENT is coNP-complete: intractable when viewed “all-at-once”, tractable as a language of pairs
A Sufficient, yet Non-Necessary Restriction

- The fact that only polynomially many varying parts $\alpha$ are possible is not a necessary condition for the membership to $\text{compP}$
- $\text{TERM ENTAILMENT} \in \text{compP}$
- A separability property at the query level (the dual of the disjunctive one)

\[ \Sigma \models \alpha_1 \land \ldots \land \alpha_n \text{ iff } \forall i \in 1 \ldots n, \Sigma \models \alpha_i \]
Many non-compilability results from the literature cannot be rephrased as compC-completeness results.

E.g. it is unlikely that CLAUSE ENTAILMENT is compcoNP-complete (it would make P = NP).

There is a need for more general compilability classes: nu-compC.
Proving Non-Compilability

- In order to show that a problem is not in \( \text{compC} \), it is enough to prove that it is \textit{nu-}\( \text{compC}'\)-hard, where \( \text{C}' \) is located higher than \( \text{C} \) in the polynomial hierarchy.
- \textbf{Complete problems} for any nu-\( \text{compC} \) class can be easily derived from complete problems for \( \text{C} \).
- Hence nu-\( \text{compC} \)-complete problems appear as a very interesting tool for proving non-\( \text{compilability} \) results.
Further Readings

Compilability of a number of AI problems: diagnosis, planning, abduction, belief revision, closed-world reasoning, paraconsistent inference from belief bases, etc.

- Liberatore [PhD98, KR’98, ACM TCL00, IJIS05]
- Cadoli et al. [AIJ99, Inf.Comp.02]
- Liberatore and Schaerf [ACM TCL07]
- Nebel [JAIR00]
- Coste and Marquis [AMAI02]
- Darwiche and Marquis [AIJ04]
- Chen [IJCAI’05]
- ...
Part V

Part V: The Knowledge Compilation Map
Darwiche and Marquis [IJCAI’01, JAIR02]
A multi-criteria evaluation of dozen target languages for KC for addressing the **choice problem**

- **Queries**: operations returning information from a compiled form without changing it
- **Transformations**: operations modifying the compiled form
- **Succinctness**: the ability of a language to represent information using little space
- ...
Queries

Decision or function problems / properties of fragments

- CO (consistency)
- CE (clause entailment: implicates)
- VA (validity)
- EQ (equivalence)
- SE (sentential entailment)
- IM (implicants)
- CT (model counting)
- ME (model enumeration)
- ...

Transformations

Function problems / properties of fragments

- **CD** conditioning
- $\land C (\land BC)$ (closure under $\land$)
- $\lor C (\lor BC)$ (closure under $\lor$)
- $\neg C$ (closure under $\neg$)
- **FO** ($SFO$) (forgetting)
- ...
Forgetting

Lin and Reiter [AAAI Symp. 94]
Lang, Liberatore and Marquis [JAIR03]

Let \( \Sigma \) be a propositional formula and let \( X \) be a subset of variables from PS

The \textit{forgetting} of \( X \) from \( \Sigma \), denoted \( \exists X.\Sigma \), is the \textbf{most general consequence} of \( \Sigma \) that is \textbf{independent} of \( X \)

Example: \( \exists \{ b \}.\((\neg a \lor b) \land (\neg b \lor c)) \equiv \neg a \lor c \)
Forgetting via Conditioning

An **inductive** characterization:

1. $\exists \emptyset. \Sigma \equiv \Sigma$
2. $\exists \{x\}. \Sigma \equiv (\Sigma \mid \neg x) \lor (\Sigma \mid x)$
3. $\exists (X \cup \{x\}). \Sigma \equiv \exists X. (\exists \{x\}. \Sigma)$
The Importance of Forgetting

Forgetting is an important transformation, with numerous applications:

- diagnosis,
- planning,
- reasoning under inconsistency,
- ...


Queries and Transformations are not Independent

Let $L$ be a subset of $\text{NNF}$

- If a language $L$ satisfies $\text{SE}$, then it satisfies $\text{CE}$ and $\text{EQ}$
- If $L$ satisfies $\text{ME}$, then it satisfies $\text{CO}$
- If $L$ satisfies $\text{CO}$ and $\text{CD}$, then it satisfies $\text{CE}$
- If $L$ satisfies $\text{CT}$, then it satisfies $\text{CO}$ and $\text{VA}$
- If $L$ satisfies $\text{CO}$, $\land C$ and $\neg C$, then it satisfies $\text{SE}$
- If $L$ satisfies $\text{VA}$, $\lor C$ and $\neg C$, then it satisfies $\text{SE}$
- If $L$ contains $L_{PS}$ and satisfies $\land C$ and $\lor BC$, then it does not satisfy $\text{CO}$ unless $P = \text{NP}$
- If $L$ satisfies $\text{FO}$, then it satisfies $\text{CO}$
- ...
Succinctness captures the ability of a language to represent information using little space

- $L_1$ is at least as succinct as $L_2$, denoted $L_1 \leq_s L_2$, iff there exists a polynomial $p$ such that for every formula $\alpha \in L_2$, there exists an equivalent formula $\beta \in L_1$ where $|\beta| \leq p(|\alpha|)$
- A pre-order $\leq_s$ over the subsets of NNF
“Positive” results

- **DNNF** satisfies **CO, CE, ME, CD, FO, ∨C**
- **d-DNNF** satisfies **CO, VA, CE, IM, CT, ME, CD**
- **OBDD<** satisfies **CO, VA, CE, IM, EQ, CT, ME, CD, SFO, ∧BC, ∨BC, ¬C**
- **DNF** satisfies **CO, CE, ME, CD, FO, ∧BC, ∨C**
- **PI** satisfies **CO, VA, CE, IM, EQ, SE, ME, CD, FO, ∨BC**
- **IP** satisfies **CO, VA, CE, IM, EQ, SE, ME, CD, ∧BC**
DNNF satisfies FO

An **inductive** characterization:

- $\exists X. \text{false} \equiv \text{false}$
- $\exists X. \text{true} \equiv \text{true}$
- $\exists X. l \equiv \text{true} \text{ if } \text{var}(l) \in X, \equiv l \text{ otherwise}$
- $\exists X. (\alpha_1 \lor \ldots \lor \alpha_n) \equiv (\exists X. \alpha_1) \lor \ldots \lor (\exists X. \alpha_n)$
- $\exists X. (\alpha_1 \land \ldots \land \alpha_n) \equiv (\exists X. \alpha_1) \land \ldots \land (\exists X. \alpha_n)$ since $\alpha_1 \land \ldots \land \alpha_n$ is decomposable
“Negative” results

- **DNNF** does not satisfy any of **VA**, **IM**, **EQ**, **SE**, **CT**, $\land BC$, $\neg C$ unless $P = NP$

- **VA**: the validity problem for **DNF** formulas is coNP-complete

- ...
The Succinctness of Propositional Fragments

- $\text{DNNF} <_s \text{d-DNNF} <_s \text{OBDD}$
- $\text{CNF} \not<_s \text{DNF}$
- $\text{DNNF} \not<_s \text{CNF}$
- ...
Succinctness vs. Non-Succinctness Results

Different kinds of proof

- **DNNF $\leq_s$ DNF**: Easy since DNNF $\supset$ DNF
- **DNF $\not\leq_s$ DNNF**: Combinatorial arguments

$$\bigwedge_{i=0}^{n-1} \left( \neg x_{2i} \lor x_{2i+1} \right) \in \text{DNNF}$$

- **DNNF $\not\leq_s$ CNF**: Exploit non-compilability results!
  - DNNF satisfies CE
  - CLAUSE ENTAILMENT from CNF formulas $\Sigma$ is not in compP unless PH collapses
Taking Advantage of the KC Map

- **Identify** the queries and transformations required by the application
- **Select** the fragments satisfying them
- **Choose** one of the most succinct fragments among the selected ones
Example: Consistency-Based Diagnostic

Generating the **consistency-based diagnoses** of a system

- **ME, FO, CD** are required
- **DNNF, DNF, PI** satisfy them
- **DNNF and PI** are the most succinct ones
- **Explain the success of DNNF and PI** for model-based diagnostic?
Further Readings

- Waechter and Haenni [KR’06] (PDAG)
- Fargier and Marquis [AAAI’06] (DDG)
- Subbarayan et al. [AAAI’07] (tree-of-BDDs)
- Pipatsrisawat and Darwiche [AAAI’08] (DNNF$_T$)
- Fargier and Marquis [AAAI’08] (Krom, Horn, Affine, etc.)
- Fargier and Marquis [ECAI’08] (closure principles)
Part VI

Part VI: Conclusions
Conclusion

- An overview of KC
- The propositional case
- Key concepts: KC, compilability, KC map, etc.
Further Readings

See the tutorial notes for references and further materials

- KC for reasoning under inconsistency
- KC for closed-world reasoning and default reasoning
- KC languages based on other formal settings, like CSPs, Bayesian networks, valued CSPs, description logics, etc.
- Applications of KC to diagnosis
- Applications of KC to planning
- ...

...
Issues for Further Work

- The conception of a **refined compilability framework**, obtained by imposing some computational restrictions on *comp* (in the current framework, *comp* can even be a non-recursive function)
- The **study of the benefits** which can be obtained by taking advantage of KC within many other AI problems (e.g., merging and other forms of paraconsistent inference)
- At the **problem level and at the instance level**
- **Completing the KC map** with additional languages, queries and transformations
Issues for Further Work

- The **decomposition of other AI problems** into such queries and transformations.
- The **development of KC maps** for more expressive settings than propositional logic.
- The **design of additional compilers** and their evaluation on benchmarks.
- The **successful exploitation of KC for other applications**.
Acknowledgements

- Thanks to the ECAI’08 committees for their invitation!
- Thanks to all my KC co-authors!
- Thanks for your attention