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 Σ into a **compiled form** $comp(\Sigma)$
  - **On-line phase:** Exploit the compiled form $comp(\Sigma)$ (and the remaining pieces of information α) to achieve the task(s) under consideration

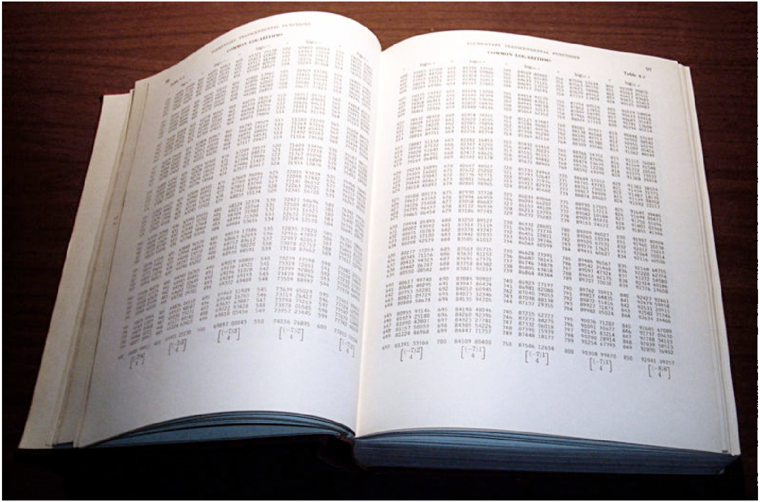
Knowledge Compilation: A Recent Research Topic

- 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.)

Knowledge Compilation: An Old Idea

- 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)
- $\Sigma \subseteq [1, 10)$
- $comp(\Sigma) =$ pairs $\langle x, log_{10}(x)\rangle$ for each $x \in \Sigma$
- α 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

$$\langle 1.234, 0.09131516\rangle \in comp(\Sigma)$$

  - 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)

$$\langle 4.152054371, 0.618263032\rangle \in comp(\Sigma)$$

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 Σ, α, ... in a **logic-based language**

$$\langle L, \vdash \rangle$$

Assumptions

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

- **Propositional logic:** syntax, semantics, consistency, validity, $\models$, $\equiv$
- **Computational complexity:** decision and function problems, Turing machine (nondeterministic, with oracle), polynomial reduction, completeness, classes of the polynomial hierarchy PH: P, NP, coNP, etc.

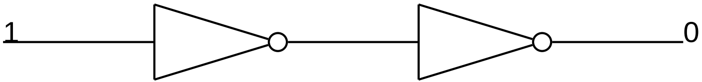
What is ”Exploiting Knowledge”?

- 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 α such that $\Sigma \not\vdash \alpha$ holds?
  - ...
- **Transformations**
  - **Conditioning:** Make some elementary propositions true (or false) in Σ
  - **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

$$in - inv_1 \wedge \neg out - inv_2$$

- Δ is a **consistency-based diagnosis** for S iff it is a conjunction of *ab*-literals such that $\Delta \wedge SD \wedge OBS$ is consistent

$$ab - inv_1, ab - inv_2, ab - inv_1 \wedge ab - inv_2$$

Example: Consistency-Based Diagnostic

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

$$mod(\exists(PS \setminus AB).(SD \mid OBS))$$

$$\exists(PS \setminus AB).(SD \mid OBS) \equiv ab - inv_1 \vee ab - inv_2$$

- 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 Σ (the fixed part)
  - Queries α 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 $comp(\Sigma)$ is **polynomial** in the size of Σ (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)
- $comp(\Sigma) =$ transition model (state graph) associated to Σ
- PLAN EXISTENCE is PSPACE**-complete** when the input is $\langle \Sigma, \alpha \rangle$
- PLAN EXISTENCE can be decided **in linear time** when the input is $\langle comp(\Sigma), \alpha \rangle$
- We cannot conclude that making the transition model explicit is the right way to solve PLAN EXISTENCE!
- For quite small values of n, $comp(\Sigma)$ cannot be generated!

Evaluating KC: The Instance Level

The experimental side

- Refers to **a specific compilation function** *comp*
- Consider **a set of** n **instances** of the problem, sharing the same fixed part Σ
- 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 Σ

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 *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

Knowledge Compilation and Propositional Reasoning

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

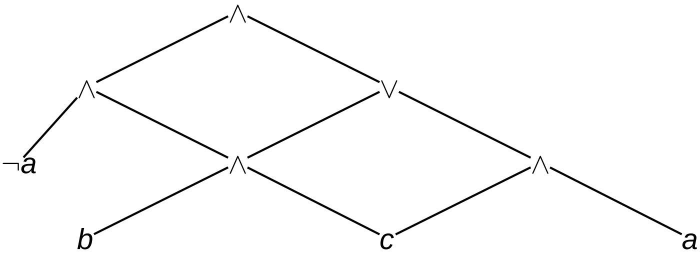
Part II

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 $\wedge$ or $\vee$ 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-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 `CNF` is the subset of `f-NNF` satisfying simple–disjunction
- The language `DNF` is the subset of `f-NNF` satisfying simple–conjunction

Other Flat Fragments

- The language `PI` is the subset of `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 `IP` is the subset of `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

- $(\neg a \vee b \vee c) \wedge (\neg b \vee d) \wedge (\neg c \vee d)$ is a `CNF` formula
- $(\neg a \wedge \neg b \wedge \neg c) \vee (b \wedge d) \vee (c \wedge d)$ is a `DNF` formula
- $(\neg a \vee b \vee c) \wedge (\neg b \vee d) \wedge (\neg c \vee d) \wedge (\neg a \vee d)$ is a `PI` formula
- $(\neg a \wedge \neg b \wedge \neg c) \vee (b \wedge d) \vee (c \wedge d) \vee (\neg a \wedge d)$ is a `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
- $(\neg a \vee b) \wedge (\neg b \vee d) \wedge (\neg c \vee d)$ is a `HORN-CNF` formula
- Unlike `CNF`, `DNF`, `PI`, `IP`, it does not allow for the representation of every formula (e.g. $a \vee b$)

CO

- Let **L** be a subset of $\mathtt{NNF}$
- **L** satisfies **CO** if and only if there exists a polynomial time algorithm that maps every formula Σ from **L**
  - to 1 if Σ is consistent,
  - and to 0 otherwise

CD

- Let **L** be a subset of $\texttt{NNF}$
- **L** satisfies **CD** if and only if there exists a polynomial time algorithm that maps every formula Σ from **L** and every consistent term γ to a formula from **L** that is logically equivalent to $\Sigma \mid \gamma$
- $\Sigma \mid \gamma$ is the formula obtained by replacing in Σ every occurrence of a variable $x \in Var(\gamma)$ by *true* if x is a positive literal of γ and by *false* if $\neg x$ is a negative literal of γ

CE

- Let **L** be a subset of $\mathtt{NNF}$
- **L** satisfies **CE** if and only if there exists a polynomial time algorithm that maps every formula Σ from **L** and every clause γ from $\mathtt{NNF}$
  - to 1 if $\Sigma \models \gamma$ holds,
  - and to 0 otherwise

Flat Fragments and **CO**, **CD**, **CE**

- Like `DNF`, `PI`, `IP`, `HORN-CNF` satisfies the **CO** query and the **CD** transformation, hence the **CE** query
- **Unit resolution** proves enough for **CO** in the `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 $Var(C_i) \cap Var(C_j) = \emptyset$ 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 $C_i \wedge C_j \models \mathit{false}$ for $i \neq j$. An NNF formula satisfies the determinism property if and only if every or-node in it is deterministic.

Decomposability

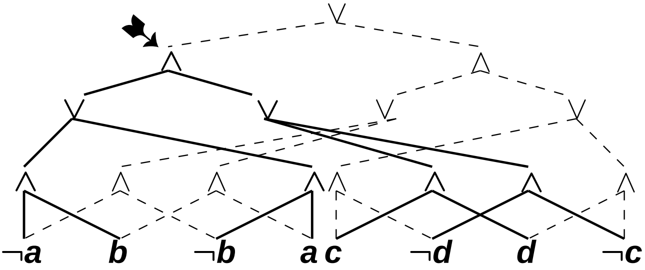


Figure: A formula in NNF. The marked node is decomposable.

Determinism

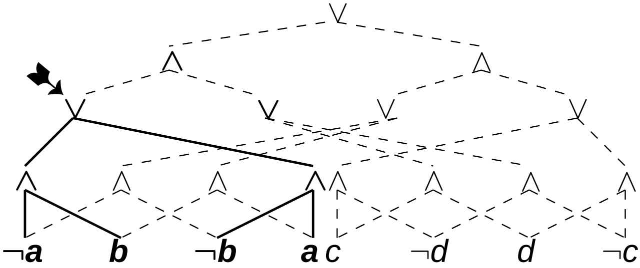


Figure: A formula in NNF. The marked node is deterministic.

Decision and Ordering

- **Decision**: A decision node N in an `NNF` formula is one which is labeled with *true*, *false*, or is an or-node having the form $(x \wedge \alpha) \vee (\neg x \wedge \beta)$, where x is a variable, α and β are decision nodes. In the latter case, $dVar(N)$ denotes the variable x. An `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 `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 $dVar(N) < dVar(M)$.

Some Nested Fragments

- The language DNNF is the subset of NNF of formulas satisfying decomposability
- The language d-DNNF is the subset of NNF of formulas satisfying decomposability and determinism
- The language $\texttt{OBDD}_<$ is the subset of NNF of formulas satisfying decomposability, decision and ordering

$\texttt{OBDD}_<$

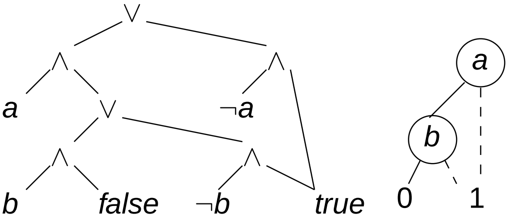


Figure: On the left part, a formula in the $\texttt{OBDD}_<$ language. On the right part, a more standard notation for it.

Nested Fragments and **CO**, **CD**, **CE**

DNNF, d-DNNF, $\text{OBDD}_<$ 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 Σ is compiled during an off-line phase and that queries α are $\mathtt{CNF}$ formulas?

- Restricting queries is necessary since deciding $\models \alpha$ is already coNP-complete for unrestricted queries
- The $\mathtt{CNF}$ format for queries is a reasonable assumption (complete and conjunctive)

Exact vs. Approximate KC

- **Exact KC:** any clause γ is taken into account as a possible query
- **Approximate KC:** a proper subset S of all clauses γ 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)

Query-Answering from Approximations

Let $comp(\Sigma) = \langle \Sigma_l, \Sigma_u \rangle$ s.t. $\Sigma_l \models \Sigma \models \Sigma_u$ and Σ_l, Σ_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** = `HORN-CNF`
- A Horn LB Σ_l of Σ is a `HORN-CNF` formula s.t. $\Sigma_l \models \Sigma$
- A Horn GLB Σ_{glb} of Σ is a Horn LB of Σ that is maximal w.r.t. $\models$
- A Horn UB Σ_u is a `HORN-CNF` formula s.t. $\Sigma \models \Sigma_u$
- A Horn LUB Σ_{lub} of Σ is a Horn UB of Σ that is minimal w.r.t. $\models$
- Horn LUBs and Horn GLBs must be **preferentially used** to improve query-answering (maximise S):

$$comp(\Sigma) = \langle \Sigma_{glb}, \Sigma_{lub} \rangle$$

Horn LUBs and GLBs

- The Horn LUB of Σ is unique up to logical equivalence but it has no representation of size polynomial in the size of Σ in the worst case
- Every Horn consequence γ of Σ 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 Σ

Computing Horn GLBs

Σ CNF $= \gamma_1 \wedge \ldots \wedge \gamma_n$

- A **Horn strengthening** γ_s of a clause γ is a Horn clause γ that is maximal w.r.t. $\models$ among the Horn clauses which imply γ
- A **Horn strengthening** $\gamma_{s1} \wedge \ldots \wedge \gamma_{sn}$ of a CNF formula $\Sigma = \gamma_1 \wedge \ldots \wedge \gamma_n$ is a HORN-CNF formula s.t. for $i \in 1 \ldots n$, γ_{si} is a Horn strengthening of γ_i

One can prove that every Horn GLB of Σ is logically equivalent to a Horn strengthening of it

Computing Horn GLBs

Input: a $\mathtt{CNF}$ Σ
Output: a Horn GLB of Σ
$L \leftarrow$ the lexicographically first Horn-strengthening of Σ
while *a lexicographically next Horn-strengthening L' of Σ exists*
do

  if $L \models L'$ **then** $L \leftarrow L'$

endwhile
remove subsumed clauses from L
Return(L)

Computing Horn LUBs

Input: a $\mathtt{CNF}$ $\Sigma = \Sigma_H$ (Horn subset) $\wedge \Sigma_{\bar{H}}$ (non-Horn)
Output: the Horn LUB of Σ
while *a resolvent* γ *of a clause of* Σ_H *and a clause of* $\Sigma_{\bar{H}}$ *that is not subsumed by any clause from* $\Sigma_H \cup \Sigma_{\bar{H}}$ *exists* **do**
remove from $\Sigma_H \cup \Sigma_{\bar{H}}$ every clause subsumed by γ
if γ *is Horn* **then**
add γ to Σ_H
else
add γ to $\Sigma_{\bar{H}}$
endif
endwhile
Return(Σ_H)

Example: Horn Approximations

- $\Sigma = (\neg a \vee b \vee c) \wedge (\neg b \vee d) \wedge (\neg c \vee d)$ is a CNF formula
- $\Sigma_{glb} = (\neg a \vee b) \wedge (\neg b \vee d) \wedge (\neg c \vee d)$ is a Horn GLB of Σ
- $\Sigma_{lub} = (\neg b \vee d) \wedge (\neg c \vee d) \wedge (\neg a \vee d)$ is the Horn LUB of Σ
- $\Sigma \not\models \neg a \vee c$ since $\Sigma_{glb} \not\models \neg a \vee c$
- $\Sigma \models \neg a \vee b \vee d$ since $\Sigma_{lub} \models \neg a \vee b \vee d$
- $\neg a \vee b \vee c \notin S$

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]

Exact KC

- Many *comp* functions and the corresponding targeted (complete) fragments satisfying **CE**

  $\mathtt{DNNF}$, $\mathtt{d\text{-}DNNF}$, $\mathtt{DNF}$, $\mathtt{OBDD}_<$, $\mathtt{PI}$, $\mathtt{IP}$, etc.

- Focus on simple fragments: $\mathtt{PI}$ and $\mathtt{DNF}$
- Separability properties
- Theory reasoning for KC:
  - Theory prime implicates
  - Tractable covers

Query-Answering from PI Formulas

Reiter and de Kleer [AAAI'87]
$comp(\Sigma) = PI(\Sigma)$

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

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 δ is one of the logically strongest clauses which can be derived from a CNF formula Σ using a resolution strategy complete in consequence-finding
- **Distribution-based algorithms:**

$$PI(\Sigma_1 \vee \ldots \vee \Sigma_k) = min(\{\delta_1 \vee \ldots \vee \delta_k \mid \delta_i \in PI(\Sigma_i)\}, \models)$$

Computing Prime Implicates: Tison's Algorithm

Input: a $\mathtt{CNF}$ formula Σ
Output: $PI(\Sigma)$
$PI \leftarrow \Sigma$
foreach $x \in Var(\Sigma)$ **do**
    foreach *pair of clauses* $x \vee \gamma_1, \neg x \vee \gamma_2$ *of* PI **do**
        if $\gamma_1 \vee \gamma_2$ *is not entailed by any clause of* PI **then**
            remove from PI every clause entailed by $\gamma_1 \vee \gamma_2$
            $PI \leftarrow PI \wedge (\gamma_1 \vee \gamma_2)$
        endif
    endforeach
    ReturnPI
endforeach

Query-Answering from DNF Formulas

Castell [ICTAI'96], Schrag [AAAI'96]
$comp(\Sigma) =$ a DNF formula equivalent to Σ

- A **linear-time query answering** algorithm exists for querying DNF formulas with clausal queries
- $\delta_1 \vee \ldots \vee \delta_k \models \gamma$ iff γ is a valid clause or for every $i \in 1 \ldots k$, δ_i contains a literal occurring in γ
- Many different DNF formulas equivalent to Σ may exist
- But **none of size polynomial** in $|\Sigma|$ for some Σ, like $\bigwedge_{i=0}^{n-1}(\neg x_{2i} \vee x_{2i+1})$
- The smallest ones contains only **prime implicants** of Σ, i.e., terms maximal w.r.t. $\models$ among those implying Σ
- **Adaptation of DPLL algorithms** can be used for computing DNF formulas

A DPLL-like Algorithm for Computing DNF Formulas

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

Input: a CNF formula Σ
Output: a DNF formula equivalent to Σ
DNF $\leftarrow \emptyset$
DPLL*(Σ, $\emptyset$)
Return(DNF)

DPLL*

Input: a CNF formula Σ, a partial assignment (term) τ
Output: false
if Σ *is the empty set of clauses* **then**
  DNF $\leftarrow$ **DNF** $\cup\{\tau\}$

endif
Return(*false*)
if Σ *contains* $\Box$ **then Return**(*false*)
UNIT-PROPAGATE(Σ, τ)
select a branching literal l using a branching rule
DPLL*(simp(Σ, $\tau \cup \{l\}$))
DPLL*(simp(Σ, $\tau \cup \{\sim l\}$))

Computing Prime Implicants Covers

- DPLL* can be improved to compute **prime implicants covers**, i.e., DNF formulas $comp(\Sigma)$ in which each term is a prime implicant of Σ
- Interesting since DNF of minimal size are prime implicants covers
- Before adding a term τ to DNF, **extract a prime implicant** of Σ 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 γ is a logical consequence of a conjunctive formula $\alpha_1 \wedge \ldots \wedge \alpha_n$ if and only if there exists $i \in 1 \ldots n$ s.t. γ is a logical consequence of α_i
- Satisfied by `PI` formulas and by `DNNF` formulas

The Disjunctive Separability Property

- **Disjunctive separability:**

  A formula α is a logical consequence of a disjunctive formula $\alpha_1 \vee \ldots \vee \alpha_n$ if and only if for every $i \in 1 \ldots n$ s.t. α is a logical consequence of α_i.
- Satisfied by every NNF formula

The Power of Separability Properties

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

  $$(\neg x \wedge (\Sigma \mid \neg x)) \vee (x \wedge (\Sigma \mid x))$$

  Such a formula is equivalent to Σ and **exhibits several separable subformulas**
- Shannon decompositions can be computed **using DPLL-like algorithms**

Theory Reasoning for KC

- 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 Φ s.t. $\Sigma \models \Phi$ and Φ belongs to a fragment satisfying **CE**

Theory Prime Implicates

Marquis [IJCAI'95]
Let Σ, Φ be two propositional formulas

- $\Sigma \models_{\Phi} \gamma$ iff $\Sigma \wedge \Phi \models \gamma$
- A **theory implicate** of Σ w.r.t. Φ is a clause γ s.t. $\Sigma \models_{\Phi} \gamma$
- A **theory prime implicate (tpi)** of Σ w.r.t. Φ is a theory implicate δ of Σ w.r.t. Φ that is minimal w.r.t. $\models_{\Phi}$
- $TPI(\Sigma, \Phi)$ is the set of all tpis of Σ w.r.t. Φ (assuming that one representative per equivalence class is kept, only)

Properties of Theory Prime Implicates

- The theory prime implicates form of Σ is **unique** (when clauses are considered up to Φ-equivalence)
- The prime implicates of Σ 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 δ of Σ w.r.t. Φ s.t. $\delta \models_{\Phi} \gamma$
- Σ is Φ-equivalent to the conjunction of all its tpis w.r.t. Φ
- Strengthening Φ may only **decrease** the number of tpis of Σ w.r.t. Φ

Query-Answering from Theory Prime Implicates

$comp(\Sigma) = \langle TPI(\Sigma, \Phi), \Phi \rangle$

- $\Sigma \models_{\Phi} \gamma$ iff $\exists \delta \in TPI(\Sigma, \Phi)$, $\delta \models_{\Phi} \gamma$
- $\delta \models_{\Phi} \gamma$ iff $\forall l \in \delta$, $\Phi \models \sim l \vee \gamma$
- If $\Sigma \models \Phi$ and Φ belongs to a fragment satisfying **CE**, then determining whether $\Sigma \models \gamma$ holds can be **achieved in polynomial time** from $comp(\Sigma)$
- **Many Φ can be selected or computed** (select the subset of Horn clauses from Σ, compile a subset of Σ, etc.)‘

Example: Theory Prime Implicates

- $\Sigma = (a \vee b \vee d) \wedge (a \vee e) \wedge (b \vee e) \wedge (b \vee d \vee f) \wedge \Phi$
- $\Phi = (\neg a \vee b) \wedge (\neg b \vee c) \wedge (\neg e \vee f)$
- $a \vee e$ and $b \vee d$ are the tpis of Σ w.r.t. Φ (up to Φ-equivalence)
- $\Sigma \models b \vee f$ since $a \vee e \models_{\Phi} b \vee f$

Computing Theory Prime Implicates

- We have $TPI(\Sigma, \Phi) = min(PI(\Sigma \wedge \Phi), \models_\Phi)$
- **Algorithms for computings 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])

Tractable Cover Compilations

Boufkhad, Grégoire, Marquis, Mazure, Saïs [IJCAI'97]

Use several classes of theories instead of a preset theory

- Let τ be a partial assignment (term) s.t. *simp*(Σ, τ) belongs to a fragment **L** satisfying **CE**
- Σ and *simp*(Σ, τ) are τ-equivalent
- For a clause γ, determining whether $\tau \wedge$ *simp*$(\Sigma, \tau) \models \gamma$ can be done **in polynomial time:**
  - **conjunctive separability** $\tau \wedge$ *simp*$(\Sigma, \tau) \models \gamma$ iff $\tau \models \gamma$ or *simp*$(\Sigma, \tau) \models \gamma$
  - **theory reasoning:** each *simp*$(\Sigma, \tau) \in$ **L** is a tractable theory (**L** satisfies **CE**)

Tractable Cover Compilations

- **disjunctive separability:** search for a **valid** DNF $\tau_1 \vee \ldots \vee \tau_k$ s.t. each $simp(\Sigma, \tau_i)$ ($i \in 1 \ldots k$) belongs to a fragment satisfying **CE** in order to derive a **tractable cover** $\bigvee_{i=1}^{k}(\tau_i \wedge simp(\Sigma, \tau_i))$ of Σ
- $\Sigma \equiv (\tau_1 \vee \ldots \vee \tau_k) \wedge \Sigma \equiv \bigvee_{i=1}^{k}(\tau_i \wedge \Sigma) \equiv \bigvee_{i=1}^{k}(\tau_i \wedge simp(\Sigma, \tau_i))$
- $comp(\Sigma) = \langle \tau_1 \vee \ldots \vee \tau_k, \Sigma \rangle$: storing $\tau_1, \ldots, \tau_k$ and Σ is sufficient since *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 $\Sigma = (a \vee b \vee c) \wedge (\neg a \vee \neg b \vee \neg c)$
- $comp(\Sigma) = \langle a \vee (\neg a \wedge b) \vee (\neg a \wedge \neg b), \Sigma \rangle$ represents a **tractable cover** of Σ
- The `HORN-CNF` fragment is targeted
  - $simp(\Sigma, a) = (\neg b \vee \neg c)$
  - $simp(\Sigma, \neg a \wedge b) = true$
  - $simp(\Sigma, \neg a \wedge \neg b) = c$
- $[a \wedge (\neg b \vee \neg c)] \vee [\neg a \wedge b] \vee [\neg a \wedge \neg b \wedge c]$ is a `HORN-CNF`$[\vee]$ formula equivalent to Σ

The Size of the Compiled Form

- For all target fragments for KC considered before, there exist Σ such that $|comp(\Sigma)|$ 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 Σ 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

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

  CLAUSE ENTAILMENT = $\{\langle \Sigma, \alpha \rangle \mid \Sigma$ a NNF formula and α a clause s.t. $\Sigma \models \alpha\}$

compC

- C = a complexity class closed under polynomial reductions and admitting complete problems for such reductions
- A language of pairs L **belongs to** compC if and only if there exists a **polysize function** *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 $\langle comp(\Sigma), \alpha \rangle \in L'$
- For every admissible complexity class C, we have the inclusion C $\subseteq$ 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 $\mathbb{N}$ 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** NP $\subseteq$ P/poly **then** $\Pi_2^p = \Sigma_2^p$ **(hence** PH **collapses at the second level)**
- If NP $\subseteq$ coNP/poly then $\Pi_3^p = \Sigma_3^p$ (hence PH collapses at the third level)

CLAUSE ENTAILMENT $\notin$ compP

Kautz and Selman [AAAI'92]

- Let n be any non-negative integer
- Let Σ_n^{max} be the CNF formula

$$\bigwedge_{\gamma_i \in 3-C_n} \neg holds_i \vee \gamma_i$$

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

CLAUSE ENTAILMENT $\notin$ compP

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

$$\Sigma_n^{max} \models \gamma_{\alpha_n} = \bigvee_{holds_i \in S_{\alpha_n}} \neg holds_i$$

CLAUSE ENTAILMENT $\notin$ 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 α be a 3-CNF formula
  - If $|Var(\alpha)| = n$, then the machine loads

$$A(n) = comp(\Sigma_n^{max})$$

  - Finally it determines in polynomial time whether $comp(\Sigma_n^{max}) \models \gamma_\alpha$
- 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 Σ, there are only **polynomially many** possible varying parts α, then the corresponding language of pairs *L* belongs to compP
- During the off-line phase, consider successively every α and store it in a lookup-table *comp*(Σ) whenever $\langle\Sigma, \alpha\rangle$ belongs to *L*
- For every Σ, $|$*comp*$(\Sigma)|$ is polynomially bounded in the size of Σ and determining on-line whether $\langle\Sigma, \alpha\rangle \in L$ amounts to a lookup operation

Example: LITERAL ENTAILMENT $\in$ compP

- $\{\langle \Sigma, \alpha \rangle \mid \Sigma \in \mathtt{NNF}, \alpha \in L_{Var(\Sigma)}, \Sigma \models \alpha\} \in$ 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 α are possible is **not a necessary condition** for the membership to compP
- TERM ENTAILMENT $\in$ compP
- A **separability property** at the query level (the dual of the disjunctive one)

$$\Sigma \models \alpha_1 \wedge \ldots \wedge \alpha_n \text{ iff } \forall i \in 1 \ldots n, \Sigma \models \alpha_i$$

compC are not General Enough!

- 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 compC, it is enough **to prove that it is** nu-compC'**-hard, where** C' **is located higher than** C in the polynomial hierarchy.
- **Complete problems** for any nu-compC class can be easily derived from complete problems for C
- Hence nu-compC-complete problems appear as a very interesting tool for proving non-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

The KC 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
- $\wedge$ **C** ($\wedge$**BC**) (closure under $\wedge$)
- $\vee$**C** ($\vee$**BC**) (closure under $\vee$)
- $\neg$**C** (closure under $\neg$)
- **FO** (**SFO**) (forgetting)
- ...

Forgetting

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

- Let Σ be a propositional formula and let X be a subset of variables from PS
- The *forgetting* of **X** from Σ, denoted $\exists X.\Sigma$, is the **most general consequence** of Σ that is **independent** of X
- Example: $\exists\{b\}.((\neg a \vee b) \wedge (\neg b \vee c)) \equiv \neg a \vee c$

Forgetting via Conditioning

An **inductive** characterization:

- $\exists\emptyset.\Sigma \equiv \Sigma$
- $\exists\{x\}.\Sigma \equiv (\Sigma \mid \neg x) \vee (\Sigma \mid x)$
- $\exists(X \cup \{x\}).\Sigma \equiv \exists X.(\exists\{x\}.\Sigma)$

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 `NNF`

- If a language **L** satisfies **SE**, then it satisfies **CE** and **EQ**
- If **L** satisfies **ME**, then it satisfies **CO**
- If **L** satisfies **CO** and **CD**, then it satisfies **CE**
- If **L** satisfies **CT**, then it satisfies **CO** and **VA**
- If **L** satisfies **CO**, $\wedge$**C** and $\neg$**C**, then it satisfies **SE**
- If **L** satisfies **VA**, $\vee$**C** and $\neg$**C**, then it satisfies **SE**
- If **L** contains L_{PS} and satisfies $\wedge$**C** and $\vee$**BC**, then it does not satisfy **CO** unless P = NP
- If **L** satisfies **FO**, then it satisfies **CO**
- ...

Succinctness

Succinctness captures **the ability of a language to represent information using little space**

- $\mathbf{L}_1$ is **at least as succinct as** $\mathbf{L}_2$, denoted $\mathbf{L}_1 \leq_s \mathbf{L}_2$, iff there exists a polynomial p such that for every formula $\alpha \in \mathbf{L}_2$, there exists an equivalent formula $\beta \in \mathbf{L}_1$ where $|\beta| \leq p(|\alpha|)$
- A **pre-order** $\leq_s$ over the subsets of NNF

Fragments, Queries and Transformations

"Positive" results

- `DNNF` satisfies **CO**, **CE**, **ME**, **CD**, **FO**, $\vee$**C**
- `d-DNNF` satisfies **CO**, **VA**, **CE**, **IM**, **CT**, **ME**, **CD**
- $\texttt{OBDD}_<$ satisfies **CO**, **VA**, **CE**, **IM**, **EQ**, **CT**, **ME**, **CD**, **SFO**, $\wedge$**BC**, $\vee$**BC**, $\neg$**C**
- `DNF` satisfies **CO**, **CE**, **ME**, **CD**, **FO**, $\wedge$**BC**, $\vee$**C**
- `PI` satisfies **CO**, **VA**, **CE**, **IM**, **EQ**, **SE**, **ME**, **CD**, **FO**, $\vee$**BC**
- `IP` satisfies **CO**, **VA**, **CE**, **IM**, **EQ**, **SE**, **ME**, **CD**, $\wedge$**BC**

DNNF satisfies **FO**

An **inductive** characterization:

- $\exists X.\mathit{false} \equiv \mathit{false}$
- $\exists X.\mathit{true} \equiv \mathit{true}$
- $\exists X.l \equiv \mathit{true}$ if $\mathit{var}(l) \in X$, $\equiv l$ otherwise
- $\exists X.(\alpha_1 \vee \ldots \vee \alpha_n) \equiv (\exists X.\alpha_1) \vee \ldots \vee (\exists X.\alpha_n)$
- $\exists X.(\alpha_1 \wedge \ldots \wedge \alpha_n) \equiv (\exists X.\alpha_1) \wedge \ldots \wedge (\exists X.\alpha_n)$ since $\alpha_1 \wedge \ldots \wedge \alpha_n$ is decomposable

Fragments, Queries and Transformations

“Negative” results

- `DNNF` **does not satisfy** any of **VA**, **IM**, **EQ**, **SE**, **CT**, $\wedge$**BC**, $\neg$**C** unless P = NP
- **VA**: the validity problem for `DNF` formulas is coNP-complete
- ...

The Succinctness of Propositional Fragments

- $\texttt{DNNF} <_s \texttt{d-DNNF} <_s \texttt{OBDD}_<$
- $\texttt{CNF} \not\leq_s \texttt{DNF}$
- $\texttt{DNNF} \not\leq_s \texttt{CNF}$
- ...

Succinctness vs. Non-Succinctness Results

Different kinds of proof

- $\mathtt{DNNF} \leq_s \mathtt{DNF}$: Easy since $\mathtt{DNNF} \supseteq \mathtt{DNF}$
- $\mathtt{DNF} \not\leq_s \mathtt{DNNF}$: **Combinatorial arguments**

$$\bigwedge_{i=0}^{n-1} (\neg x_{2i} \vee x_{2i+1}) \in \mathtt{DNNF}$$

- $\mathtt{DNNF} \not\leq_s \mathtt{CNF}$: **Exploit non-compilability results!**
  - DNNF satisfies **CE**
  - CLAUSE ENTAILMENT from CNF formulas Σ 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 *el al.* [AAAI'07] (`tree-of-BDDs`)
- Pipatsrisawat and Darwiche [AAAI'08] ($\mathtt{DNNF}_T$)
- 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