On Unit-Refutation Complete Formulae with Existentially Quantified Variables*

Lucas Bordeaux
Microsoft Research, Cambridge
lucasb@microsoft.com

Mikoláš Janota
INESC-ID, Lisboa
mikolas@sat.inesc-id.pt

Joao Marques-Silva
CASL, UCD & IST/INESC-ID
jpms@ucd.ie

Pierre Marquis
CRIL-CNRS, U. d'Artois
marquis@cril.fr

Abstract

We analyze, along the lines of the knowledge compilation map, both the tractability and the succinctness of the propositional language URC−C of *unit-refutation* complete propositional formulae, as well as its disjunctive closure URC−C[∨, ∃], and a superset of URC−C where variables can be existentially quantified and unit-refutation completeness concerns only consequences built up from free variables.

Introduction

This paper is about the representation of propositional knowledge as logical formulae in the classical *Conjunctive Normal Form* (CNF). This representation formalism is the input language of SAT solvers, and is also increasingly used in other constraint solving tools as a target language into which higher-level constraints are translated (see, e.g. (Bessiere, Hebrard, and Walsh 2003; Bacchus 2007; Brand et al. 2007; Quimper and Walsh 2008; Huang 2008; Feydy and Stuckey 2009; Bessiere et al. 2009)).

The key challenge when representing problems in CNF is to reach the two goals of *tractability* and *succinctness*. Many automated reasoning tasks on CNF formulae have high computational complexity and can only be performed efficiently if the problem is well-posed in a certain sense. A way to obtain a well-posed CNF encoding is to add information to it in the form, notably, of extra clauses; but this process can, in bad cases, blow-up the formula exponentially.

Unit-Refutation Completeness To shed light on the elusive notion of "well-posed CNF encoding", we consider the key notion of *Unit-Refutation Completeness*. A formula is unit-refutation complete iff any of its implicates can be refuted by unit propagation. Refutation refers to the process of proving an implication $\alpha \models \beta$ by proving that $\alpha \wedge \neg\beta \models \bot$. Unit resolution is the restriction of the well-known resolution rule where at least one of the two clauses resolved upon is a unit clause, i.e., a clause containing a single literal. A CNF formula α is unit-refutation complete if all clauses δ that are implied by α can be proved by refutation using a unit-resolution proof, i.e., there exists a finite sequence of clauses $\delta_1, \ldots, \delta_n = \bot$, where each δ_i is a clause of α, or the complementary literal of a literal of δ, or is obtained by unit resolution from two previous clauses of the sequence. Hence, for a unit-refutation complete α, whether a clause δ is implied by α or not is decided by determining whether performing unit resolution on $\alpha \wedge \neg\delta$ leads to the empty clause.

Example 1 *The* CNF *formula* $\alpha = (a \vee c) \wedge (\neg a \vee c) \wedge d$ *is unit-refutation complete: for each of its implicates* δ *(here, every clause implied by* c *or by* d*), there exists a unit refutation of* $\alpha \wedge \neg\delta$*. In contrast, the* CNF *formula* $\beta = (a \vee b \vee c) \wedge (a \vee \neg b \vee c) \wedge (\neg a \vee b \vee c) \wedge (\neg a \vee \neg b \vee c) \wedge d$*, though equivalent to* α*, is not unit-refutation complete.*

The class URC−C of unit-refutation complete CNF formulae has well-known tractability properties for the consistency query and more generally for clausal entailment; indeed, determining whether a URC−C formula α implies a clause δ amounts to determining whether a unit refutation of $\alpha \wedge \neg\delta$ exists, which can be decided in time $\mathcal{O}(\mid \alpha \mid + \mid \delta \mid)$, while the problem of clausal entailment is coNP-complete for unrestricted CNF formulae. Of course, there is some price to be paid: turning CNF formulae into equivalent URC−C ones is always feasible but the size of the resulting formulae is exponential in the input size in the worst case. Unless the polynomial hierarchy collapses, the same problem occurs whenever the target language of the translation offers tractable clausal entailment.

Unit-refutation complete CNF formulae are increasingly used in the analysis of CNF encodings: for instance a particular case of URC−C formulae ("unit-*propagation* completeness", or UPC−C) is considered by the aforementioned papers as well as in recent papers on Clause Learning in SAT (Atserias, Fichte, and Thurley 2011; Pipatsrisawat and Darwiche 2011) and knowledge compilation (Bordeaux and Marques-Silva 2012). Additionally, Sinz (Sinz 2002) explores the pre-processing of CNF formulae into URC−C to speed-up the resolution of configuration problems.

*Pierre Marquis benefited from the support of the project BR4CP ANR-11-BS02-008 of the French National Agency for Research - Agence Nationale de la Recherche. Joao Marques-Silva and Mikoláš Janota were supported by SFI grant BEACON (09/PI/I2618), and by FCT grants ATTEST (CMU-PT/ELE/0009/2009) and POLARIS (PTDC/EIA-CCO/123051/2010).

Knowledge Compilation The language URC−C is but one specific kind of *knowledge compilation* language. In fact, it is historically among the first knowledge compilation languages introduced in the literature since del Val showed in (del Val 1994) how to render a CNF formula unit-refutation complete by conjoining it with some of its prime implicates. However, in many AI applications, tractable clausal entailment is not enough: other queries and transformations are expected to be feasible in polynomial time. Furthermore, the more queries are supported by a knowledge compilation language, the larger problem encodings in this language tend to be. Therefore another crucial aspect of URC−C is to understand its succinctness. In order to evaluate more accurately the attractiveness of URC−C as a target language for knowledge compilation, in terms of tractability and succinctness, we study it along the lines of the knowledge compilation map (Darwiche and Marquis 2002).

In CNF encodings, it is well-known that the introduction of existentially quantified variables can have a big impact on the size. A basic example is the Boolean function $\mathit{XOR}(x_1 \cdots x_n)$ requiring that the number of true values among n variables be odd. Representing this function as a CNF formula requires 2^{n-1} clauses, but with the addition of extra variables it is easy to represent it as a CNF formula of size linear in n and whose solutions are essentially the same (i.e., up to projecting away the introduced variables). Similarly, it is well-known that when encoding constraints, introducing extra variables is often necessary to obtain concise *well-posed* encodings.

In knowledge compilation, languages that allow the introduction of existentially quantified variables are called *existential closures*, as defined in (Fargier and Marquis 2008). More generally, closures with $\exists$ and/or $\vee$ are collectively referred to as *disjunctive* closures (Fargier and Marquis 2008) since existential quantifications can be viewed as a form of generalized disjunctions. The well-known Tseitin transformation (Tseitin 1968) shows how every propositional formula can be turned in polynomial time into CNF[$\exists$], the existential closure of CNF, which simply amounts to the set of CNF formulae where some variables are existentially quantified. However, CNF (and a fortiori CNF[$\exists$]) cannot be considered as an interesting target language for knowledge compilation since it is does not offer a polynomial-time consistency test unless P = NP.

Hence, an important issue is to identify new propositional languages based on CNF encodings which are more tractable than CNF[$\exists$] and CNF in the sense that they offer a polynomial-time consistency test, but also offer succinct encodings via the introduction of existentially quantified variables.

Contributions Towards this objective, we introduce in this paper the language $\exists$URC−C. This is a subset of CNF[$\exists$] consisting mainly of formulae of the form $\exists X.\alpha$, where X is a finite (and possibly empty) set of propositional variables, α is a CNF formula and for every implicate δ of $\exists X.\alpha$ there exists a unit-refutation from $\alpha \wedge \neg\delta$. While URC−C[$\exists$] is a subset of $\exists$URC−C, it turns out that URC−C[$\exists$] and $\exists$URC−C are not equal. Let us slightly modify Example 1 by introducing existential variables in order to illustrate the difference:

Example 2 *The* CNF[$\exists$] *formula* $\exists\{a, b, c\}.\beta = \exists\{a, b, c\}.((a \vee b \vee c) \wedge (a \vee \neg b \vee c) \wedge (\neg a \vee b \vee c) \wedge (\neg a \vee \neg b \vee c) \wedge d)$ *is an* $\exists$URC−C *formula, since for any implicate* δ *(a clause implied by* d*) unit propagation on* $\beta \wedge \neg\delta$ *yields a conflict (denoted as* $\beta \wedge \neg\delta \vdash_u \bot$*). However,* β *is not a* URC−C *formula, as explained before.*

In the following, we analyze the tractability and the succinctness of URC−C, its disjunctive closure URC−C[$\vee, \exists$] and $\exists$URC−C along the lines of the knowledge compilation map. We show that the three languages have both a polynomial-time consistency test, and a polynomial-time clausal entailment test. Unlike URC−C, URC−C[$\vee, \exists$] and $\exists$URC−C offer also polynomial time forgetting and closure by disjunction. Furthermore, they are strictly more succinct than URC−C. We also show that $\exists$URC−C compares favorably with the influential DNNF language, and its subsets.

Formal Preliminaries

Propositional Logic

We consider subsets $\mathcal{L}$ of the propositional language QDAG of quantified propositional DAGs.

Definition 1 (QDAG) *Let* PS *be a denumerable set of propositional variables.* QDAG *is the set of all finite, single-rooted DAGs* α *where:*

- *each leaf node of* α *is labeled by a literal* l *over* PS*, or by a Boolean constant* $\top$ *(always true) or* $\bot$ *(always false) ;*
- *each internal node of* α *is labeled by a connective* $c \in \{\wedge, \vee, \neg, \oplus\}$ *and has as many children as required by* c *(*$\neg$ *is a unary connective while the three other connectives admit finitely many arguments), or is labeled by* $\exists x$ *(where* $x \in PS$*) and has a single child.*

All the propositional languages considered in the following are subsets of QDAG; DAG is the subset of QDAG where no node labeled by a quantification is allowed (each DAG formula thus corresponds to a Boolean circuit.)

A literal (over $V \subseteq PS$) is an element $x \in V$ (a positive literal) or a negated one $\neg x$ (a negative literal), or a Boolean constant. $\bar{l}$ is the complementary literal of literal l, so that $\overline{\top} = \bot$, $\overline{\bot} = \top$, $\overline{x} = \neg x$ and $\overline{\neg x} = x$. For a literal l different from a Boolean constant, $var(l)$ denotes the corresponding variable: for $x \in PS$, we have $var(x) = x$ and $var(\neg x) = x$. A clause (resp. a term) is a finite disjunction (resp. conjunction) of literals. CLAUSE (resp. TERM) is the subset of DAG consisting of all clauses (resp. term). A CNF formula is a finite conjunction of clauses, while a DNF formula is a finite disjunction of terms.

Each element α of QDAG is called a QDAG formula. $Var(\alpha)$ denotes the set of free variables x of α, i.e., those variables x for which there exists a leaf node n_x of α labeled by a literal l such that $var(l) = x$ and there is a path from the root of α to n_x such that no node from it is labeled by $\exists x$. The size $|\alpha|$ of a QDAG formula $|\alpha|$ is the number of nodes plus the number of arcs in the DAG.

A key deductive technique on CNF formulae is the well-known *resolution* rule whereby we deduce $A \vee B$ from two clauses $A \vee x$ and $\neg x \vee B$, where x is a variable. *Unit* resolution is the restriction when one of the clauses is reduced

to a single literal, i.e., one of the clauses A or B is empty. A unit derivation from a CNF α is a finite sequence of clauses $\delta_1, \ldots, \delta_n$, where each δ_i is a clause of α, or is obtained by unit resolution from two previous clauses of the sequence. We write $\alpha \vdash_u c$ if there is a unit derivation ending with $\delta_n = c$. We say that α is unit-refutable if $\alpha \vdash_u \bot$.

The Knowledge Compilation Map

Within the knowledge compilation map, propositional languages are evaluated w.r.t. their ability to support some queries and transformations in polynomial time and w.r.t. their succinctness. The following queries and transformations are considered.

Definition 2 (queries) *Let $\mathcal{L} \subseteq$ QDAG.*

- *$\mathcal{L}$ satisfies* **CO** *(consistency) iff there exists a polytime algorithm that maps every formula α from $\mathcal{L}$ to 1 if α is consistent, and to 0 otherwise.*
- *$\mathcal{L}$ satisfies* **VA** *(validity) iff there exists a polytime algorithm that maps every formula α from $\mathcal{L}$ to 1 if α is valid, and to 0 otherwise.*
- *$\mathcal{L}$ satisfies* **CE** *(clausal entailment) iff there exists a polytime algorithm that maps every formula α from $\mathcal{L}$ and every clause γ to 1 if $\alpha \models \gamma$ holds, and to 0 otherwise.*
- *$\mathcal{L}$ satisfies* **EQ** *(equivalence) iff there exists a polytime algorithm that maps every pair of formulae α, β from $\mathcal{L}$ to 1 if $\alpha \equiv \beta$ holds, and to 0 otherwise.*
- *$\mathcal{L}$ satisfies* **SE** *(sentential entailment) iff there exists a polytime algorithm that maps every pair of formulae α, β from $\mathcal{L}$ to 1 if $\alpha \models \beta$ holds, and to 0 otherwise.*
- *$\mathcal{L}$ satisfies* **IM** *(implicant) iff there exists a polytime algorithm that maps every formula α from $\mathcal{L}$ and every term γ to 1 if $\gamma \models \alpha$ holds, and to 0 otherwise.*
- *$\mathcal{L}$ satisfies* **CT** *(model counting) iff there exists a polytime algorithm that maps every formula α from $\mathcal{L}$ to a nonnegative integer that represents the number of models of α over $Var(\alpha)$ (in binary notation.)*
- *$\mathcal{L}$ satisfies* **ME** *(model enumeration) iff there exists a polynomial $p(.,.)$ and an algorithm that outputs all models of an arbitrary formula α from $\mathcal{L}$ in time $p(n, m)$, where n is the size of α and m is the number of its models (over $Var(\alpha)$.)*
- *$\mathcal{L}$ satisfies* **MC** *(model checking) iff there exists a polytime algorithm that maps every formula α from $\mathcal{L}$ and every interpretation ω over $Var(\alpha)$ (represented as a term) to 1 if ω is a model of α, and to 0 otherwise.*

Definition 3 (transformations) *Let $\mathcal{L} \subseteq$ QDAG.*

- *$\mathcal{L}$ satisfies* **CD** *(conditioning) iff there exists a polytime algorithm that maps every formula α from $\mathcal{L}$ and every consistent term γ to a formula from $\mathcal{L}$ that is logically equivalent to the conditioning $\alpha \mid \gamma$ of α on γ, i.e., the formula obtained by replacing each free occurrence of variable x of α by $\top$ (resp. $\bot$) if x (resp. $\neg x$) is a positive (resp. negative) literal of γ.*
- *$\mathcal{L}$ satisfies* **FO** *(forgetting) iff there exists a polytime algorithm that maps every formula α from $\mathcal{L}$ and every subset X of variables from PS to a formula from $\mathcal{L}$ equivalent to $\exists X.\alpha$. If the property holds for each singleton X, we say that $\mathcal{L}$ satisfies* **SFO** *(single forgetting).*
- *$\mathcal{L}$ satisfies $\wedge$***C** *(conjunction) iff there exists a polytime algorithm that maps every finite set of formulae $\alpha_1, \ldots, \alpha_n$ from $\mathcal{L}$ to a formula of $\mathcal{L}$ that is logically equivalent to $\alpha_1 \wedge \cdots \wedge \alpha_n$.*
- *$\mathcal{L}$ satisfies $\vee$***C** *(disjunction) iff there exists a polytime algorithm that maps every finite set of formulae $\alpha_1, \ldots, \alpha_n$ from $\mathcal{L}$ to a formula of $\mathcal{L}$ that is logically equivalent to $\alpha_1 \vee \cdots \vee \alpha_n$.*
- *$\mathcal{L}$ satisfies $\wedge$***BC** *(bounded conjunction) iff there exists a polytime algorithm that maps every pair of formulae α and β from $\mathcal{L}$ to a formula of $\mathcal{L}$ that is logically equivalent to $\alpha \wedge \beta$.*
- *$\mathcal{L}$ satisfies $\vee$***BC** *(bounded disjunction) iff there exists a polytime algorithm that maps every pair of formulae α and β from $\mathcal{L}$ to a formula of $\mathcal{L}$ that is logically equivalent to $\alpha \vee \beta$.*
- *$\mathcal{L}$ satisfies $\neg$***C** *(negation) iff there exists a polytime algorithm that maps every formula α from $\mathcal{L}$ to a formula of $\mathcal{L}$ logically equivalent to $\neg\alpha$.*

Succinctness is defined as follows.

Definition 4 (succinctness) *Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two subsets of* QDAG. *$\mathcal{L}_1$ is at least as succinct as $\mathcal{L}_2$, denoted $\mathcal{L}_1 \leq_s \mathcal{L}_2$, iff there exists a polynomial p such that for every formula $\alpha \in \mathcal{L}_2$, there exists an equivalent formula $\beta \in \mathcal{L}_1$ where $|\beta| \leq p(|\alpha|)$.*

It turns out that the succinctness relation is a pre-order (i.e., a reflexive and transitive relation). One also often takes advantage of the following restriction of succinctness, where the translation must be achieved in polynomial time (instead of polynomial space.)

Definition 5 (polynomial translation) *Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two subsets of* QDAG. *$\mathcal{L}_1$ is said to be* polynomially translatable *into $\mathcal{L}_2$, noted $\mathcal{L}_2 \leq_p \mathcal{L}_1$, iff there exists a polytime algorithm f such that for every $\alpha \in \mathcal{L}_1$, we have $f(\alpha) \in \mathcal{L}_2$ and $f(\alpha) \equiv \alpha$.*

Thus, whenever $\mathcal{L}_1$ is polynomially translatable into $\mathcal{L}_2$, $\mathcal{L}_2$ is at least as succinct as $\mathcal{L}_1$. Furthermore, when $\mathcal{L}_1$ is polynomially translatable into $\mathcal{L}_2$, every query which is supported in polynomial time in $\mathcal{L}_2$ also is supported in polynomial time in $\mathcal{L}_1$; and conversely, every query which is not supported in polynomial time in $\mathcal{L}_1$ unless P = NP (resp. unless the polynomial hierarchy PH collapses) cannot be supported in polynomial time in $\mathcal{L}_2$, unless P = NP (resp. unless PH collapses.)

$\sim_s$ (resp. $\sim_p$) is the symmetric part of $\leq_s$ (resp. $\leq_p$.) $<_s$ (resp. $<_p$) is the asymmetric part of $\leq_s$ (resp. $\leq_p$.) When $\mathcal{L}_1 \sim_p \mathcal{L}_2$, $\mathcal{L}_1$ and $\mathcal{L}_2$ are said to be polynomially equivalent. Obviously enough, polynomially equivalent fragments are equally efficient (and succinct) and possess the same set of tractable queries and transformations. $\mathcal{L}_1 \not\leq_s^* \mathcal{L}_2$ means that $\mathcal{L}_1$ is not at least as succinct as $\mathcal{L}_2$ unless PH collapses.

Disjunctive Closures

Intuitively, a closure principle applied to a propositional language $\mathcal{L}$ defines a new propositional language, called a closure of $\mathcal{L}$, through the application of "operators" (i.e., connectives or quantifications.) The resulting closure is said to be *disjunctive* when the operators are among $\vee$ and $\exists x$ with $x \in PS$. It is called the *disjunction closure* $\mathcal{L}[\vee]$ of $\mathcal{L}$ when the operator is $\vee$, and the *existential closure* $\mathcal{L}[\exists]$ of $\mathcal{L}$ when the operators are of the form $\exists x$.

Definition 6 (disjunctive closures) *Let $\mathcal{L} \subseteq$ QDAG and $\triangle \subseteq \{\vee, \exists\}$. The* closure $\mathcal{L}[\triangle]$ *of $\mathcal{L}$ by $\triangle$ is the subset of* QDAG *inductively defined as follows:*

1. *if $\alpha \in \mathcal{L}$, then $\alpha \in \mathcal{L}[\triangle]$,*
2. *if $\vee \in \Delta$ and $\alpha_i \in \mathcal{L}[\triangle]$ for each $i \in 1, \ldots, n$, then $\vee(\alpha_1, \ldots, \alpha_n) \in \mathcal{L}[\triangle]$;*
3. *if $\exists \in \Delta$, $x \in PS$, and $\alpha \in \mathcal{L}[\triangle]$, then $\exists x.\alpha \in \mathcal{L}[\triangle]$.*

In order to avoid heavy notations, when $\triangle = \{\delta_1, \ldots, \delta_n\}$, we write $\mathcal{L}[\delta_1, \ldots, \delta_n]$ instead of $\mathcal{L}[\{\delta_1, \ldots, \delta_n\}]$.

Thus, an element of $\mathcal{L}[\triangle]$ can be viewed as a tree in which internal nodes are labeled by quantifications of the form $\exists x$ or by $\vee$ and leaf nodes are labeled by elements of $\mathcal{L}$. Accordingly, the formulae α_i considered in item 2 of Definition 6 do not share any common subgraphs.

(Fargier and Marquis 2008) provide several general-scope characterization results for disjunctive closures. Especially, they show that for a given $\mathcal{L} \subseteq$ QDAG, under very weak conditions on $\mathcal{L}$ (stability by renaming), only three disjunctive closures are worth considering since $\mathcal{L}[\exists][\exists] = \mathcal{L}[\exists]$, $\mathcal{L}[\vee][\vee] = \mathcal{L}[\vee]$ and $\mathcal{L}[\exists][\vee] \sim_p \mathcal{L}[\vee][\exists] \sim_p \mathcal{L}[\vee, \exists]$ (see item 4 of Proposition 1 and item 1 of Proposition 2 in (Fargier and Marquis 2008).) They also study the disjunctive closures of the languages $\mathcal{L}$ among $\text{OBDD}_<$, DNF, DNNF, CNF, PI, IP, MODS considered in (Darwiche and Marquis 2002). (Fargier and Marquis 2008) study the disjunction closures $\mathcal{L}[\vee]$ of some incomplete fragments $\mathcal{L}$, mainly the KROM−C language (the subset of CNF consisting of conjunctions of binary clauses), the HORN−C language (the subset of CNF consisting of conjunctions of Horn clauses), and the AFF language (the set of affine formulae, which are conjunctions of XOR-clauses.) (Marquis 2011) investigates the existential closures of those languages $\mathcal{L}$ and of their disjunction closures $\mathcal{L}[\vee]$.

URC−C, URC−C$[\vee, \exists]$, and $\exists$URC−C

Definition 7 (URC−C) *(del Val 1994)* URC−C *is the subset of* CNF *consisting of formulae α which are unit-refutation complete, i.e., for every implicate $\delta = l_1 \vee \ldots \vee l_k$ of α, we have $\alpha \wedge \bar{l}_1 \wedge \ldots \wedge \bar{l}_k \vdash_u \bot$.*

The following lemma shows that focusing on the *prime* implicates of a CNF formula α is enough when one wants to determine whether it belongs to URC−C. The main advantage is that a propositional formula always has finitely many prime implicates (up to logical equivalence), while it has infinitely many implicates (up to logical equivalence) when PS is denumerable.

Lemma 1 *A* CNF *formula α is a* URC−C *formula iff for every prime implicate $\delta = l_1 \vee \ldots \vee l_k$ of α, we have $\alpha \wedge \bar{l}_1 \wedge \ldots \wedge \bar{l}_k \vdash_u \bot$.*

Example 3 *The* CNF *formula $\alpha = (a \vee b) \wedge (\neg b \vee c) \wedge (\neg a \vee c)$ belongs to* URC−C*. c is a prime implicate of it and we have $\alpha \wedge \neg c \vdash_u \bot$; indeed, the sequence $\neg c$, $\neg b \vee c$, $\neg b$, $\neg a \vee c$, $\neg a$, $a \vee b$, a, $\bot$ is a unit refutation of $\alpha \wedge \neg c$.*

The language URC−C is not closed under logical equivalence within CNF. For instance, the formula $(a \vee b \vee c) \wedge (a \vee b \vee \neg c) \wedge (a \vee \neg b \vee c) \wedge (\neg a \vee \neg b \vee c) \wedge (\neg a \vee b \vee c)$ is not URC−C while equivalent to α from Example 3. Nevertheless, it is easy to show that a CNF α is URC−C iff the formula obtained by removing in α every clause that is strictly implied by another clause of α also is a URC−C formula. Furthermore, a CNF α is URC−C iff the saturation of α by unit-propagation is also URC−C (the formula is obtained by repeatedly applying unit-resolution on α and replacing every clause by its resolvent when the latter subsumes the former.)

These two easy results show that URC−C satisfies a certain form of stability by simplification.

In order to compare URC−C with other subsets of CNF, like PI (the Blake formulae, also known as prime implicate formulae), the following easy lemma is useful.

Lemma 2 *Let α be a* CNF *formula containing each of its prime implicates. Then α is an* URC−C *formula.*

As a direct consequence, we have the inclusion PI $\subseteq$ URC−C. This shows that URC−C is a complete propositional language: for every Boolean function, there exists a URC−C formula representing it. Another easy consequence is that MONO−C $\subseteq$ URC−C, where MONO−C is the subset of CNF consisting of monotone formulae (a CNF formula α being monotone iff for every variable $x \in Var(\alpha)$, either x occurs as a literal in α or $\neg x$ occurs as a literal in α.) Indeed, a monotone CNF formula contains each of its prime implicate.

Other influential subsets of CNF are KROM−C, HORN−C, renH−C (the set of CNF formulae α which are renamable Horn, i.e. such that there exists a subset V of $Var(\alpha)$ for which the formula obtained by replacing in α every literal l over a variable of V by the complementary literal $\bar{l}$ is a HORN−C formula.) Clearly, renH−C is a subset of URC−C. Indeed, it is well-known that a clause $\delta = l_1 \vee \ldots \vee l_k$ is an implicate of a renH−C formula α iff $\alpha \wedge \bar{l}_1 \wedge \ldots \wedge \bar{l}_k \vdash_u \bot$. Since HORN−C is a subset of renH−C, we obtain that HORN−C is a subset of URC−C. This is also the case of CLAUSE (another subset of renH−C) but this does not extend to KROM−C; for instance the KROM−C formula $(a \vee b) \wedge (\neg a \vee b) \wedge (a \vee \neg b) \wedge (\neg a \vee \neg b)$ is not a URC−C formula since it is contradictory but no unit refutation of it exists. However, since every consistent KROM−C formula is a renH−C formula and since KROM−C satisfies **CO**, we immediately obtain that KROM−C is polynomially translatable into URC−C. The inclusion renH−C $\subset$ URC−C is a strict one since for instance the URC−C formula $(a \vee b \vee c) \wedge (\neg a \vee \neg b \vee \neg c)$ is an URC−C formula but not a renH−C one.

This shows that many interesting subsets of CNF are also subsets of URC−C (or at least are polynomially translatable into it.) In particular, URC−C is "bounded" by the two com-

plete languages PI and CNF, in the sense that

$$\mathtt{PI} \subset \mathtt{URC\text{-}C} \subset \mathtt{CNF}.$$

Both inclusions are strict ones since $a \wedge (\neg a \vee b)$ is URC−C formula but not a PI one, while $(a \vee b) \wedge (\neg a \vee b) \wedge (a \vee \neg b) \wedge (\neg a \vee \neg b)$ is a CNF formula but not a URC−C one. Furthermore, we will show in the following that URC−C is strictly more succinct then PI, while being strictly less succinct than CNF unless the PH collapses.

We now introduce formally the language ∃URC−C.

Definition 8 (∃URC−C) *∃URC−C is the subset of CNF[∃] consisting of URC−C formulae, and formulae of the form $\exists X.\alpha$ where X is a finite subset of PS and α is a CNF formula, such that for every implicate $\delta = l_1 \vee ... \vee l_k$ of $\exists X.\alpha$, we have $\alpha \wedge \bar{l}_1 \wedge ... \wedge \bar{l}_k \vdash_u \bot$.*

Obviously enough, URC−C is a subset of ∃URC−C. We also have that URC−C[∃] is a subset of ∃URC−C. Indeed, consider a URC−C[∃] formula of the form $\exists X.\alpha$; then for every implicate $\delta = l_1 \vee ... \vee l_k$ of α, we have $\alpha \wedge \bar{l}_1 \wedge ... \wedge \bar{l}_k \vdash_u \bot$ since α is a URC−C formula. Then the fact that the implicates of $\exists X.\alpha$ are among the implicates of α completes the proof.

We have for ∃URC−C direct counterparts to Lemmas 1 and 2.

Lemma 3 *A CNF[∃] formula α is an ∃URC−C formula iff for every prime implicate $\delta = l_1 \vee ... \vee l_k$ of α, we have $\alpha \wedge \bar{l}_1 \wedge ... \wedge \bar{l}_k \vdash_u \bot$.*

Lemma 4 *Let α be a CNF[∃] formula containing each of its prime implicates. Then α is a URC−C[∃] formula.*

Note that the (prime) implicates of an existentially quantified formula are clauses built on the *free* variables of this formula, therefore these lemmas allow us to reason in terms of free variables only.

Tractability and Succinctness

We now prove a number of results that shed light on the tractability and succinctness of URC−C, URC−C[∨, ∃], and ∃URC−C.

Relating ∃URC−C[∨] with ∃URC−C

We first explain why considering the disjunctive closures of ∃URC−C yields a language that is equivalent to ∃URC−C from the tractability and succinctness point of view. While it is obvious that ∃URC−C =∃URC−C[∃], we also show that:

Proposition 1 ∃URC−C $\sim_p$ ∃URC−C[∨].

As a consequence, we have that ∃URC−C $\sim_p$ ∃URC−C[∨, ∃]. Prop. 1 hides a subtlety. The idea is to eliminate the disjunctions from a ∃URC−C[∨] (or URC−C[∨, ∃]) formulae, in order to construct a new CNF formula (possibly with more existentially quantified variables) and that is, *importantly, also unit-refutation complete*. There are two standard ways to eliminate such disjunctions. The brute-force approach takes advantage of the distributivity of ∨ over ∧; when two (or, more generally, a fixed number of) CNF formulae are considered, this can be achieved in polynomial time (stated otherwise, CNF satisfies ∨**BC**), but this does not extend to an unbounded number of CNF formulae. The other approach exploits the familiar *Tseitin encoding* (Tseitin 1968) which introduces new (existentially quantified) variables (and this approach runs in polynomial time for an unbounded number of CNF formulae to be disjoined.) However, none of the methods preserves unit-refutation completeness in the general case:

Example 4 *Consider the formula $\alpha_1 \vee \alpha_2$ where $\alpha_1 \equiv (\neg a \vee b) \wedge (a \vee b)$ and $\alpha_2 \equiv (\neg c \vee d) \wedge (c \vee d)$. Both α_1 and α_2 are URC−C formulae, hence ∃URC−C formulae. Using the brute-force approach, $\alpha_1 \vee \alpha_2$ is turned into the equivalent CNF formula $\alpha_b = (\neg a \vee b \vee \neg c \vee d) \wedge (\neg a \vee b \vee c \vee d) \wedge (a \vee b \vee \neg c \vee d) \wedge (a \vee b \vee c \vee d)$. It turns out that α_b does not belong to ∃URC−C since the clause $(b \vee d)$ is an implicate of $\alpha_1 \vee \alpha_2$ but $\alpha_b \wedge \neg b \wedge \neg d \not\vdash_u \bot$. Similarly, to express $\alpha_1 \vee \alpha_2$ in CNF using Tseitin encoding, we can introduce two new variables τ_1, τ_2 that capture the truth value of each disjunct. We obtain the CNF[∃] formula: $\alpha_T = \exists \tau_1, \tau_2.\ ((\tau_1 \vee \tau_2) \wedge (\neg\tau_1 \vee \neg a \vee b) \wedge (\neg\tau_1 \vee a \vee b) \wedge (\neg\tau_2 \vee \neg c \vee d) \wedge (\neg\tau_2 \vee c \vee d))$. Again, this formula α_T is not ∃URC−C: the clause $(b \vee d)$ is an implicate of it but $\alpha_T \wedge \neg b \wedge \neg d \not\vdash_u \bot$.*

To prove Prop. 1 we need a more sophisticated encoding that produces an ∃URC−C formula. Intuitively we need to process the Tseitin encoding in a way that simulates "constructive disjunction": unit propagation on the processed formula should be enhanced so as to directly infer any literal that would be obtained in each and every branch of a case reasoning on the literals of the selected clause. Such encodings are defined as follows.

Definition 9 ("Constructive Disjunction" Encoding) *Given a satisfiable CNF formula α and a selected clause $(d_1 \vee \cdots \vee d_p)$ of α, a* constructive disjunction encoding *of α w.r.t. the selected clause is a CNF formula δ with $Var(\delta) \supseteq Var(\alpha)$ such that, given any set of literals $l_1 \cdots l_k$ built on $Var(\alpha)$*

1. *$\alpha \wedge l_1 \wedge \cdots \wedge l_k$ is satisfiable iff $\delta \wedge l_1 \wedge \cdots \wedge l_k$ is satisfiable;*
2. *if, $\forall i \in 1, \cdots, p$, $\alpha \wedge l_1 \wedge \cdots \wedge l_k \wedge d_i \vdash_u \bot$ then $\delta \wedge l_1 \wedge \cdots \wedge l_k \vdash_u \bot$*

Prop. 1 relies on the following proposition, which states that constructive encodings indeed exist, and can be computed in a tractable way.

Proposition 2 *Given any CNF α we can construct a constructive disjunction encoding of α w.r.t. any selected clause in time polynomial in $|\alpha|$.*

The formal definition of the encoding is given in the proof of Prop. 2. Here we simply give a flavour of what the encoding looks like for a simple example:

Example 5 *Consider the formula $(x \vee y), (\neg x \vee z), (\neg y \vee z)$. Assume we want a constructive disjunction w.r.t. the clause $(x \vee y)$. The encoding introduces variables $x_1^-, x_1^+, y_1^-, y_1^+, z_1^-, z_1^+$ that, intuitively, simulate unit propagation conditionally to the first disjunct, namely x. (We have a positive and a negative version of each variable so that the simulation never causes an inconsistency but, instead, detects it by setting e.g. both x_1^- and x_1^+ to true.)*

$\mathcal{L}$	CO	VA	CE	IM	EQ	SE	CT	ME	MC
$\exists$URC−C	√	○	√	○	○	○	○	√	√
URC−C[∨, ∃]	√	○	√	○	○	○	○	√	√
URC−C	√	√	√	√	√	√	○	√	√

Table 1: Queries. √ means "satisfies" and ○ means "does not satisfy unless P = NP."

$\mathcal{L}$	CD	FO	SFO	∧C	∧BC	∨C	∨BC	¬C
$\exists$URC−C	√	√	√	○	○	√	√	○
URC−C[∨, ∃]	√	√	√	○	○	√	√	○
URC−C	√	•	?	○	○	•	?	•

Table 2: Transformations. √ means "satisfies," • means "does not satisfy," ○ means "does not satisfy unless P=NP.".

Similarly for the second disjunct y we introduce variables $x_2^-, x_2^+, y_2^-, y_2^+, z_2^-, z_2^+$. We have 4 groups of clauses;

1. *clauses that simulate propagation in the first context under the assumption that x is true:* $(x_1^+), (\neg x_1^- \vee y_1^+), (x_1^+ \vee \neg y_1^-)(\neg x_1^+ \vee z_1^+), (x_1^- \vee \neg z_1^-), (\neg y_1^+ \vee z_1^+), (y_1^- \vee \neg z_1^-)$*;*
2. *clauses that simulate propagation in the 2nd context under the assumption that y is true:* $(y_2^+), (\neg x_2^- \vee y_2^+), (x_2^+ \vee \neg y_2^-)(\neg x_2^+ \vee z_2^+), (x_2^- \vee \neg z_2^-), (\neg y_2^+ \vee z_2^+), (y_2^- \vee \neg z_2^-)$*;*
3. *clauses that "inject" unit literals from the original formula into each simulation, e.g.* $(\neg x \vee x_1^+), (x \vee x_1^-)$ *(similarly for y and z and for the second context);*
4. *clauses that detect when a literal is* true under all cases of the disjunction*, e.g.* $(\neg x_1^+ \vee \neg x_2^+ \vee x), (\neg x_1^- \vee \neg x_2^- \vee \neg x)$ *(similarly for y and z).*

To see what the encoding brings, observe, for instance, that the literal z is deduced from this formula by unit propagation.

Queries and Transformations

As to queries and transformations, we have obtained the following results.

Proposition 3 *The results given in Table 1 and in Table 2 hold.*

From Proposition 3, it turns out that URC−C[∨, ∃] and $\exists$URC−C are equally tractable. URC−C is more tractable than $\exists$URC−C w.r.t. queries (since it offers the same queries as $\exists$URC−C plus **VA**, **IM**, **EQ** and **SE**). Conversely, $\exists$URC−C is more tractable than URC−C w.r.t. transformations since beyond ∨**C**, it offers the very significant **FO** transformation (indeed, forgetting is a key transformation in a number of AI problems, with (among others) applications to diagnosis, planning, reasoning about change, reasoning under inconsistency.) We ignore whether URC−C satisfies ∨**BC** or **SFO** (since it satisfies **CD**, if it satisfies ∨**BC**, then it satisfies **SFO**). Anyway, both transformations are offered by $\exists$URC−C.

Let us now report some results concerning the succinctness of URC−C, URC−C[∨, ∃] and $\exists$URC−C.

Proposition 4 *The following succinctness results hold:*

1. $\exists$URC−C $\leq_s$ URC−C[∨, ∃] $<_s$ URC−C $<_s$ PI.
2. URC−C $\not\leq_s^*$ CNF *and* CNF $\leq_s$ URC−C.
3. $\exists$URC−C $\not\leq_s^*$ CNF *and* CNF $\not\leq_s$ $\exists$URC−C.
4. URC−C $\not\leq_s$ DNF, URC−C $\not\leq_s$ SDNNF, *and* URC−C $\not\leq_s$ d−DNNF,
5. DNF $\not\leq_s$ URC−C, SDNNF $\not\leq_s$ URC−C, *and* FBDD $\not\leq_s$ URC−C.
6. $\exists$URC−C $\leq_s$ DNNF.
7. $\exists$URC−C $<_s$ DNF.
8. $\exists$URC−C $<_s$ SDNNF.
9. $\exists$URC−C $<_s^*$ d−DNNF.

Point 1. shows that $\exists$URC−C is at least as succinct as URC−C[∨, ∃], which is strictly more succinct than URC−C. Especially, since URC−C[∨, ∃] is just as tractable as $\exists$URC−C, $\exists$URC−C appears definitely as a language which is at least as interesting as URC−C[∨, ∃] from the knowledge compilation point of view. As a consequence, we refrained from comparing it with other languages from the succinctness standpoint and focus instead on URC−C and $\exists$URC−C.

Points 2. and 3. give succinctness results concerning URC−C, $\exists$URC−C compared to CNF, which is in some sense the most standard propositional language and the basic language on which URC−C and $\exists$URC−C have been built. While URC−C is strictly less succinct than CNF (unless PH collapses), moving from URC−C to the strictly more succinct $\exists$URC−C language leads to language which is incomparable with CNF as to succinctness.

The remaining points report succinctness results between URC−C, $\exists$URC−C and the influential DNNF language and some of its subsets. We considered the three subsets DNF, SDNNF, and d−DNNF of DNNF because existing compilers to DNNF actually target one of those three languages; furthermore, they include other interesting subsets of DNNF (especially, FBDD and its well-known subset OBDD$_<$ are subsets of d−DNNF).

The results show that, as to succinctness, URC−C is incomparable with each of DNF, SDNNF, and FBDD. For sure, URC−C is not at least as succinct as d−DNNF (hence not at least as succinct as DNNF), but whether d−DNNF $\leq_s$ URC−C remains an open issue. Again, considering $\exists$URC−C dramatically changes the succinctness picture since $\exists$URC−C is at least as succinct as DNNF, and is strictly more succinct than each of DNF, SDNNF, and d−DNNF.[1]

Conclusion

This article shows that $\exists$URC−C is a very interesting target language for knowledge compilation. Indeed, while it offers the same tractable queries and transformations as URC−C[∨, ∃], $\exists$URC−C is at least as succinct as URC−C[∨, ∃]. More importantly, it offers the same queries and transformations and is at least as succinct as the influential DNNF language from (Darwiche 2001). Furthermore, $\exists$URC−C is strictly more succinct than DNF, SDNNF, and

[1] Unless PH collapses for d−DNNF.

d−DNNF which are the three most succinct subsets of DNNF for which compilers have been developed.

The main issue for further research will concern the design and the evaluation of compilation algorithms targeting ∃URC−C. Note that we can already take advantage of the algorithms pointed out in (del Val 1994) in order to compile some CNF formulae into equivalent URC−C formulae. One can also take advantage of some recent algorithms (Bordeaux and Marques-Silva 2012) targeting UPC−C, the subset of CNF consisting of formulae α which are *unit-propagation complete*, i.e., for every implicate $\delta = l_1 \vee ... \vee l_k$ of α which is not valid, we have $\alpha \wedge \bar{l}_1 \wedge ... \wedge \bar{l}_{k-1} \vdash_u \bot$ or $\alpha \wedge \bar{l}_1 \wedge ... \wedge \bar{l}_{k-1} \vdash_u l_k$. Indeed, UPC−C is a proper subset of URC−C.

Interestingly, any existing CNF2URC−C compilation algorithm gives rise immediately to a DAG2∃URC−C compilation algorithm. Indeed, given a DAG formula α_1, the approach first consists in turning it in polynomial time into an equivalent CNF[∃] formula $\alpha_2 = \exists X.\alpha_3$ using Tseitin encoding. Then α_3 can be turned into an equivalent URC−C formula α_4 using the CNF2URC−C compilation algorithm. By construction, the formula $\exists X.\alpha_4$ is a URC−C[∃] formula equivalent to α_1, hence an ∃URC−C formula. Note that the approach also works if α_1 is a DAG[∃] formula.

Appendix

Proof:[Lemma 1] If for every implicate $\delta = l_1 \vee \ldots \vee l_k$ of α, we have $\alpha \wedge \bar{l}_1 \wedge \ldots \wedge \bar{l}_k \vdash_u \bot$, then this holds for the prime implicates of α. Conversely, let δ be an implicate of α. By primality, there exists a prime implicate $\gamma = l_1 \vee \ldots \vee l_k$ of α such that $\gamma \models \delta$. By assumption, $\alpha \wedge \bar{l}_1 \wedge \ldots \wedge \bar{l}_k \vdash_u \bot$. Since $\vdash_u$ is monotonic w.r.t. $\subseteq$ (i.e., if a unit refutation of a CNF formula β exists, then for every CNF formula γ, a unit refutation of the CNF $\beta \wedge \gamma$ exists), adding to $\alpha \wedge \bar{l}_1 \wedge \ldots \wedge \bar{l}_k$ the conjunction of the complementary literals of those occurring in δ but not in γ does not affect the existence of a unit refutation. This concludes the proof. ∎

Proof:[Lemma 2] Let $\delta = l_1 \vee \ldots \vee l_k$ be an implicate of α. If α contains each of its prime implicates, then there is a clause γ of α s.t. $\gamma \models \delta$. Obviously enough, we have $\gamma \wedge \bar{l}_1 \wedge \ldots \wedge \bar{l}_k \vdash_u \bot$, hence $\alpha \wedge \bar{l}_1 \wedge \ldots \wedge \bar{l}_k \vdash_u \bot$ as well. ∎

Proof:[Lemmas 3 and 4] Analogous to the proofs of Lemmas 1 and 2. ∎

Proof:[Proposition 1] We prove the direction that is non-trivial, i.e., ∃URC−C $\leq_p$ ∃URC−C[∨]. We are given a disjunction α of ∃URC−C formulae. We put this disjunction of existentially quantified formulae into prenex form, obtaining a formula β of the form $\exists y_1 \cdots y_t.\ \alpha_1 \vee \cdots \vee \alpha_p$.

We rewrite β into a conjunctive normal form by a Tseitin encoding as in Example 4: we introduce p variables $\tau_1, \ldots, \tau_p$, such that each τ_i will "trigger" formula α_i when set to true. To this effect, for each α_i $(i \in 1, \ldots, p)$ we define $\beta_i = \bigwedge_{c \in \mathcal{C}_i} (\neg\tau_i \vee c)$ where $\mathcal{C}_i$ denotes the set of clauses of α_i. We obtain a formula ϕ defined as: $\exists y_1 \cdots y_t.\ \exists \tau_1 \cdots \tau_p.\ ((\tau_1 \vee \cdots \vee \tau_p) \wedge \bigwedge_{i \in 1}^{p} \beta_i)$.

We now build a *constructive disjunction encoding* of this formula w.r.t. the disjunction $(\tau_1 \vee \cdots \vee \tau_p)$, following Def. 9. This transformation introduces new existentially quantified variables, which we here call $z_1, \cdots, z_q$. We obtain a CNF[∃] formula ψ the prefix of which is the form $\exists y_1 \cdots y_t.\ \exists \tau_1 \cdots \tau_p.\ \exists z_1 \cdots z_q$.

We now show that ψ is ∃URC−C. Consider any implicate $\mathcal{I} = (l_1 \vee \cdots \vee l_k)$ of ψ built over $Var(\psi) = Var(\alpha)$. Equivalently it is an implicate of α, therefore it is an implicate of each formula $\exists y_1 \cdots y_t.\ \alpha_i$, for $i \in 1, \ldots, p$. Let L_i denote the clause containing only the literals from $\mathcal{I}$ that are in $Var(\alpha_i)$. We have $\alpha_i \models L_i$. Hence, $\alpha_i \wedge \neg L_i \vdash_u \bot$ and $\alpha_i \wedge \neg\mathcal{I} \vdash \bot$, for each $i \in 1, \ldots, p$. By property (2) of the Def. 9 of "constructive disjunction encoding" it follows that $\psi \wedge \overline{l_1} \wedge \cdots \wedge \overline{l_k} \vdash_u \bot$. ∎

Proof:[Proposition 2] We first introduce the following gadget, used in previous literature and in particular in (Bessiere et al. 2009): we associate to each variable x of the initial CNF α two "indicator" variables x^- and x^+, together with clauses $(\neg x \vee x^+), (x \vee x^-)$. Then each initial clause c of length k of formula α is translated into k clauses that represent the k possible ways the clause can trigger. More precisely, let c be of the form $(l_1 \vee \cdots \vee l_k)$, then for each literal $i \in 1, \ldots, k$ we add the clause

$$\left(\bigvee_{j \in 1,\ldots,k,\ j \neq i} \neg \mathit{neg}(l_j) \right) \vee \mathit{pos}(l_i)$$

where for each variable x, $\mathit{pos}(x) = x^+$, $\mathit{pos}(\neg x) = x^-$, $\mathit{neg}(x) = x^-$, and $\mathit{neg}(\neg x) = x^+$. It can be shown that the constructed gadget γ is such that, given any satisfiable set of literals $l_1 \cdots l_k$ built on $Var(\alpha)$, we have

1. $\gamma \wedge l_1 \wedge \cdots \wedge l_k$ is always satisfiable;
2. $\gamma \wedge l_1 \wedge \cdots \wedge l_{k-1} \vdash_u \mathit{pos}(l_k)$ *iff* $\alpha \wedge l_1 \wedge \cdots \wedge l_{k-1} \vdash_u l_k$.

To build a constructive disjunction encoding, we duplicate p gadgets $\gamma_1 \cdots \gamma_p$. By "duplication" we mean that in each new copy we use fresh indicator variables x^-, x^+, for all x. The role of each γ_i is, intuitively, to simulate unit propagation under each and every hypothesis $d_1 \cdots d_p$ of the selected disjunction. For any literal l built on $Var(\alpha)$, and $i \in 1, \ldots, p$, let $\mathit{pos}_i(l)$ and $\mathit{neg}_i(l)$ denote the positive and negative indicator variables for l, as above, but taken from the i^{th} gadget, γ_i. We add a unit clause $\mathit{pos}_i(d_i)$, for $i \in 1, \ldots, p$, to indicate that in each γ_i we reason under the extra assumption d_i. Now for each literal l built on $Var(\alpha)$ we enhance unit propagation by adding a clause δ_l that triggers exactly when literal l is true under all assumptions $d_1 \cdots d_p$:

$$\left(\bigvee_{i \in 1,\ldots,p} \neg \mathit{pos}_i(l) \right) \vee l$$

Formula δ is defined as $\left(\bigwedge_{i \in 1,\ldots,p} \gamma_i \wedge \mathit{pos}_i(d_i)\right) \wedge \left(\bigwedge_{x \in Var(\alpha)} \delta_x \wedge \delta_{\neg x}\right)$ and has the defining properties of a constructive disjunction encoding (Def. 9). ∎

Proof:[Proposition 3] We start with the queries:

- **CO**: Since URC−C is a subset of URC−C[$\vee, \exists$], and URC−C[$\vee, \exists$] is polynomially translatable into $\exists$URC−C, it is enough to prove the result for $\exists$URC−C. Let $\alpha = \exists X.\beta$ be an $\exists$URC−C formula. α is inconsistent iff $\bot$ is an implicate of it. If $\bot$ is an implicate of α, then since $\alpha \in \mathtt{URC} - \mathtt{C}$, we have $\beta \vdash_u \bot$, which can be decided in time linear in the size of β. If $\bot$ is not an implicate of α, then $\bot$ is not an implicate of β, hence we cannot have $\beta \vdash_u \bot$ in this case.
- **VA**:
  - $\exists$URC−C: DNNF is polynomially translatable into $\exists$URC−C using Tseitin's extension rule (Jung et al. 2008). The fact that DNNF does not satisfy **VA** unless P = NP (Darwiche and Marquis 2002) completes the proof.
  - URC−C[$\vee, \exists$]: Direct since DNF is a subset of it (since every term is an URC−C formula) and DNF does not satisfy **VA** unless P = NP (Darwiche and Marquis 2002).
  - URC−C: Direct since URC−C $\subseteq$ CNF and CNF satisfies **VA** (Darwiche and Marquis 2002).
- **CE**: Comes directly from the fact that each of URC−C, URC−C[$\vee, \exists$], and $\exists$URC−C satisfies **CO** and **CD**.
- **ME**: Direct from item 2 of Proposition 4 from (Fargier and Marquis 2008) given the fact that each language under consideration satisfies **CO** and **CD**, and consists of proper formulae.
- **IM**:
  - $\exists$URC−C, URC−C[$\vee, \exists$]: Direct since neither of these two languages satisfies **VA** unless P = NP.
  - URC−C: Direct since URC−C $\subseteq$ CNF and CNF satisfies **IM** (Darwiche and Marquis 2002).
- **EQ**, **SE**:
  - $\exists$URC−C, URC−C[$\vee, \exists$]: Because HORN−C[$\exists$] satisfies neither **EQ**, nor **SE** unless P = NP (Marquis 2011), and HORN−C[$\exists$] is a subset of URC−C[$\exists$] (since HORN−C is a subset of URC−C), and URC−C[$\exists$] is a subset of both URC−C[$\vee, \exists$] and $\exists$URC−C.
  - URC−C: Direct since URC−C satisfies **CE**.
- **CT**: Direct from the fact that the set of positive Krom formulae (e.g., CNF formulae where each clause contains at most two literals, and those literals are positive ones) does not satisfy **CT** (Roth 1996), given that this set is included in MONO−C, hence it is a subset of each of the languages under consideration.
- **MC**: Direct from item 3 of Proposition 4 from (Fargier and Marquis 2008) showing that when a subset $\mathcal{L}$ of QDAG satisfies **CO** and **CD**, then it satisfies **MC** as well, plus the fact that each of URC−C, URC−C[$\vee, \exists$], and $\exists$URC−C satisfies **CO** and **CD**.

As to the transformations:

- **CD**: From item 1 of Proposition 4 from (Fargier and Marquis 2008), we know that when a subset $\mathcal{L}$ of QDAG satisfies **CD**, then $\mathcal{L}[\vee, \exists]$ satisfies **CD** as well. Hence it is enough to show that URC−C satisfies **CD**, and that $\exists$URC−C satisfies **CD**.

  We show that whenever α is a URC−C formula and γ is a consistent term, then $\alpha \mid \gamma$ is itself a URC−C formula. Consider any prime implicate $l_1 \vee \cdots \vee l_k$ of $\alpha \mid \gamma$. We have $(\alpha \mid \gamma) \wedge \overline{l_1} \wedge \cdots \wedge \overline{l_k} \models \bot$. This implies $\alpha \wedge \gamma \wedge \overline{l_1} \wedge \cdots \wedge \overline{l_k} \models \bot$. Note that $Var(l_1 \vee \cdots \vee l_k) \subseteq Var(\alpha) \setminus Var(\gamma)$, therefore the term $\gamma \wedge \overline{l_1} \wedge \cdots \wedge \overline{l_k}$ is consistent and since α is URC−C we have $\alpha \wedge \gamma \wedge \overline{l_1} \wedge \cdots \wedge \overline{l_k} \vdash_u \bot$. By the properties of unit propagation it follows that $(\alpha \mid \gamma) \wedge \overline{l_1} \wedge \cdots \wedge \overline{l_k} \vdash_u \bot$.

  (The proof relies on the properties: given any CNF α and term γ we have: $\alpha \mid \gamma \vdash_u \bot$ iff $\alpha \wedge \gamma \vdash_u \bot$. We also have $\alpha \mid \gamma \models \bot$ iff $\alpha \wedge \gamma \models \bot$.)

  The proof for $\exists$URC−C is similar: if we are given an $\exists$URC−C formula α and a consistent term γ, then any of the prime implicates of $\alpha \mid \gamma$ is shown unit refutable, following exactly the same reasoning.
- **FO**:
  - $\exists$URC−C: If $\alpha = \exists X.\beta$ is an $\exists$URC−C formula then for every implicate δ of α, we have $\alpha \wedge \neg\delta \vdash_u \bot$. Whatever Y (a finite subset of PS), since the implicates of $\exists Y.\alpha$ are also implicates of α, we have as expected that for every implicate δ of $\exists Y.\alpha$, $\alpha \wedge \neg\delta \vdash_u \bot$ holds.
  - URC−C[$\vee, \exists$]: Obvious since this language is an existential closure.
  - URC−C: Direct since URC−C[$\exists$] $<_s$ URC−C.
- **SFO**: Obvious since each of $\exists$URC−C and URC−C[$\vee, \exists$] satisfies **FO**.
- $\wedge$**C**: Direct since neither of these languages satisfies $\wedge$**BC** unless P = NP.
- $\wedge$**BC**: Comes from the fact that every CNF formula α can be turned in polynomial time into a conjunction $\beta \wedge \gamma$ of two URC−C formulae (hence β and γ are $\exists$URC−C, and URC−C[$\vee, \exists$] formulae as well) such that $\beta \wedge \gamma$ is consistent iff α is consistent, and the fact that each of these languages satisfies **CO**, while CNF satisfies it only if P = NP. Indeed, let α be a CNF formula over n variables $x_1, \ldots, x_n$. Let β be the MONO−C formula obtained by replacing every positive literal x_i in α by the negative literal $\neg not - x_i$ (where each $not - x_i$ is a fresh variable), conjoined with n additional negative clauses $\neg x_i \vee \neg not - x_i$ ($i \in 1, \ldots, n$). Let γ be the MONO−C formula $\bigwedge_{i=1}^{n} x_i \vee not - x_i$. β and γ are MONO−C formulae, hence URC−C formulae. Finally, by construction, α is consistent iff $\beta \wedge \gamma$ is consistent. This concludes the proof.
- $\vee$**C**:
  - $\exists$URC−C: Comes from the fact that $\exists$URC−C[$\vee$] and $\exists$URC−C are polynomially equivalent.
  - URC−C[$\vee, \exists$]: By construction, as a disjunctive closure based on $\vee$.
  - URC−C: Every term is a URC−C formula. If $\vee$**C** were satisfied by URC−C, then DNF would be polynomially translatable into URC−C hence it would be polynomially translatable into its superset CNF. But CNF $\not\leq_s$ DNF (Darwiche and Marquis 2002), contradiction.

- $\vee\mathbf{BC}$: Obvious since each of the languages satisfies $\vee\mathbf{C}$.
- $\neg\mathbf{C}$:
  - $\exists$URC−C, URC−C$[\vee,\exists]$: Every term is an URC−C formula, hence an $\exists$URC−C formula. As a consequence, DNF is a subset of each of URC−C$[\vee,\exists]$ and of $\exists$URC−C$[\vee]$ which is polynomially equivalent to $\exists$URC−C. Assume that one of $\exists$URC−C, URC−C$[\vee,\exists]$, say $\mathcal{L}$, satisfies $\neg\mathbf{C}$. Then for every DNF formula α we could compute in polynomial time an $\mathcal{L}$ formula β equivalent to $\neg\alpha$. Since $\mathcal{L}$ satisfies **CO**, we could check in polynomial time whether β is consistent or not. But β is inconsistent iff α is valid, and as a consequence, we would have P = NP.
  - URC−C: Towards a contradiction: consider the following CNF formula $\alpha = \bigwedge_{i=0}^{n-1}(x_{2i} \vee x_{2i+1})$; this is a MONO−C formula hence a URC−C formula. Suppose that URC−C satisfies $\neg\mathbf{C}$. Then a URC−C formula equivalent to $\neg\alpha$ could be computed in time polynomial in the size of α. This is impossible since the formula $\neg\alpha$ (equivalent to the DNF formula $\bigvee_{i=0}^{n-1}(\neg x_{2i} \wedge \neg x_{2i+1})$) has no CNF representation of size polynomial in n (Darwiche and Marquis 2002).

∎

Proof:[Proposition 4]

1. We first have the inclusions PI $\subseteq$ URC−C $\subseteq$ URC−C$[\exists]$ $\subseteq$ URC−C$[\vee,\exists]$. Since URC−C$[\vee,\exists]$ $\sim_p$ (URC−C$[\exists]$)$[\vee]$, and since URC−C$[\exists]$ is a subset of $\exists$URC−C, we get that URC−C$[\vee,\exists]$ is polynomially translatable into $\exists$URC−C$[\vee]$, which is polynomially equivalent to $\exists$URC−C. Hence URC−C$[\vee,\exists]$ is polynomially translatable into $\exists$URC−C. Altogether, this shows that $\exists$URC−C$\leq_s$ URC−C$[\vee,\exists]$ $\leq_s$ URC−C $\leq_s$ PI.

   Proposition 6 from (Bordeaux and Marques-Silva 2012) shows that PI $\not\leq_s$ UPC−C. Since UPC−C $\subseteq$ URC−C, we get that PI $\not\leq_s$ URC−C.

   Now, every term is a URC−C formula. As a consequence, DNF is a subset of URC−C$[\vee,\exists]$, so that we have URC−C$[\vee,\exists]$ $\leq_s$ DNF. Since URC−C is a subset of CNF, we have CNF $\leq_s$ URC−C. Hence, if URC−C $\leq_s$ URC−C$[\vee,\exists]$ were the case, then by transitivity of $\leq_s$ we would have that CNF $\leq_s$ DNF, which is known to be false (Darwiche and Marquis 2002).
2. Since URC−C $\subseteq$ CNF, we have CNF $\leq_s$ URC−C. The fact that URC−C $\not\leq_s^*$ CNF comes from the fact that URC−C satisfies **CE** while the problem is coNP-complete for CNF(this is a standard result in knowledge compilation.)
3. From URC−C $\not\leq_s^*$ CNF and $\exists$URC−C $<_s$ URC−C, we get that $\exists$URC−C $\not\leq_s^*$ CNF. Now, we have that $\exists$URC−C $\leq_s$ DNF. If CNF $\leq_s$ $\exists$URC−C were the case, then by transitivity of $\leq_s$, we would have CNF $\leq_s$ DNF, which is known to be false (Darwiche and Marquis 2002).
4. First, URC−C $\not\leq_s$ DNF since CNF $\leq_s$ URC−C and CNF $\not\leq_s$ DNF (Darwiche and Marquis 2002). Then URC−C $\not\leq_s$ SDNNFsince CNF $\leq_s$ URC−C, SDNNF $\leq_s$ OBDD$_<$ and OBDD$_<$ $\not\leq_s$ CNF (Darwiche and Marquis 2002). Finally, URC−C $\not\leq_s$ d−DNNF since CNF $\leq_s$ URC−C, d−DNNF $\leq_s$ OBDD$_<$, and CNF $\not\leq_s$ OBDD$_<$ (Darwiche and Marquis 2002).
5. DNF $\not\leq_s$ URC−C, SDNNF $\not\leq_s$ URC−C, and FBDD $\not\leq_s$ URC−C. First, DNF $\not\leq_s$ URC−C because URC−C $\leq_s$ PI, but DNF $\not\leq_s$ PI (Darwiche and Marquis 2002). Then SDNNF $\not\leq_s$ URC−C because the circular bit shift functions do not have polynomial-sized SDNNF representations (Pipatsrisawat and Darwiche 2010), but have polynomial-sized KROM−C representations (hence polynomial-sized PI representations, which are URC−C formulae.) Finally, FBDD $\not\leq_s$ URC−C because URC−C $\leq_s$ PI, but FBDD $\not\leq_s$ PI (Darwiche and Marquis 2002).
6. (Jung et al. 2008) shows that DNNF is polynomially translatable into $\exists$URC−C.
7. Since DNF is a subset of URC−C$[\vee,\exists]$ and since URC−C$[\vee,\exists]$ is polynomially translatable into $\exists$URC−C, we get that $\exists$URC−C $\leq_s$ DNF. That DNF $\not\leq_s$ $\exists$URC−C comes from the fact that $\exists$URC−C $<_s$ URC−C, and DNF $\not\leq_s$ URC−C.
8. $\exists$URC−C $\leq_s$ SDNNF comes from the fact that SDNNF is a subset of DNNF and that $\exists$URC−C $\leq_s$ DNNF; That SDNNF $\not\leq_s$ $\exists$URC−C comes from the fact that $\exists$URC−C $<_s$ URC−C, and SDNNF $\not\leq_s$ URC−C.
9. $\exists$URC−C $\leq_s$ d−DNNF comes from the fact that d−DNNF is a subset of DNNF and that $\exists$URC−C $\leq_s$ DNNF; That d−DNNF $\not\leq_s^*$ $\exists$URC−C comes from the fact that d−DNNF $\not\leq_s^*$ DNF (Darwiche and Marquis 2002) and that $\exists$URC−C $\leq_s$ DNF (which comes also directly from the inclusion DNF $\subseteq$ URC−C$[\vee,\exists]$ and that URC−C$[\vee,\exists]$ is polynomially translatable into $\exists$URC−C).

∎

References

Atserias, A.; Fichte, J. K.; and Thurley, M. 2011. Clause-learning algorithms with many restarts and bounded-width resolution. *Journal of Artificial Intelligence Research* 40:353–373.

Bacchus, F. 2007. GAC via unit propagation. In *Proc. of CP*, 133–147.

Bessiere, C.; Katsirelos, G.; Narodytska, N.; and Walsh, T. 2009. Circuit complexity and decompositions of global constraints. In *Proc. of IJCAI*, 412–418.

Bessiere, C.; Hebrard, E.; and Walsh, T. 2003. Local consistencies in SAT. In *Proc. of SAT*, 299–314.

Bordeaux, L., and Marques-Silva, J. 2012. Knowledge compilation with empowerment. In *Proc. of SOFSEM*, To appear.

Brand, S.; Narodytska, N.; Quimper, C.-G.; Stuckey, P.-J.; and Walsh, T. 2007. Encodings of the sequence constraint. In *Proc. of CP*.

Darwiche, A., and Marquis, P. 2002. A knowledge compilation map. *Journal of Artificial Intelligence Research* 17:229–264.

Darwiche, A. 2001. Decomposable negation normal form. *Journal of the ACM* 48(4):608–647.

del Val, A. 1994. Tractable databases: How to make propositional unit resolution complete through compilation. In *Proc. of KR'94*, 551–561.

Fargier, H., and Marquis, P. 2008. Extending the knowledge compilation map: Closure principles. In *Proc. of ECAI'08*, 50–54.

Feydy, T., and Stuckey, P. J. 2009. Lazy clause generation reengineered. In *Proc. of CP*, 352–366.

Huang, J. 2008. Universal Booleanization of constraint models. In *Proc. of CP*, 144–158.

Jung, J.-C.; Barahona, P.; Katsirelos, G.; and Walsh, T. 2008. Two encodings of DNNF theories. In *ECAI'08 Workshop on Graphical Structures of Knowledge*.

Marquis, P. 2011. Existential closures for knowledge compilation. In *Proc. of IJCAI*, 996–1001.

Pipatsrisawat, K., and Darwiche, A. 2010. A lower bound on the size of decomposable negation normal form. In *Proc. of AAAI'10*.

Pipatsrisawat, K., and Darwiche, A. 2011. On the power of clause-learning SAT solvers as resolution engines. *Artificial Intelligence* 175(2):512–525.

Quimper, C.-G., and Walsh, T. 2008. Decompositions of grammar constraints. In *Proc. of AAAI*, 1567–1570.

Roth, D. 1996. On the hardness of approximate reasoning. *Artificial Intelligence* 82(1–2):273–302.

Sinz, C. 2002. Knowledge compilation for product configuration. In *ECAI Workshop on Configuration*.

Tseitin, G. 1968. *On the complexity of derivation in propositional calculus*. Steklov Mathematical Institute. chapter Structures in Constructive Mathematics and Mathematical Logic, 115–125.