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[V, E], 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 \( \alpha \land \neg \beta \models \bot \).

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 \( \alpha \) is unit-refutation complete if all clauses \( \delta \) that are implied by \( \alpha \) 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 \( \delta_i \) is a clause of \( \alpha \) or the complementary literal of a literal of \( \delta \), or is obtained by unit resolution from two previous clauses of the sequence. Hence, for a unit-refutation complete \( \alpha \), whether a clause \( \delta \) is implied by \( \alpha \) or not is decided by determining whether performing unit resolution on \( \alpha \land \neg \delta \) leads to the empty clause.

Example 1 The CNF formula \( \alpha = (a \lor c) \land (\neg a \lor c) \land d \) is unit-refutation complete: for each of its implicates \( \delta \) (here, every clause implied by \( c \) or by \( d \)), there exists a unit refutation of \( \alpha \land \neg \delta \). In contrast, the CNF formula \( \beta = (a \lor b \lor c) \land (a \lor \neg b \lor c) \land (\neg a \lor b \lor c) \land (\neg a \lor \neg b \lor c) \land d \), though equivalent to \( \alpha \), 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 \( \alpha \) implies a clause \( \delta \) amounts to determining whether a unit refutation of \( \alpha \land \neg \delta \) exists, which can be decided in time \( O((|\alpha| + |\delta|)) \), while the problem of clausal entailment is \textsf{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 URC–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/12618), and by FCT grants ATTEST (CMU-PT/ELE/0009/2009) and POLARIS (PTDC/EIA-CCO/123051/2010).

Copyright © 2012, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.
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 implicants. 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 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 $\lor$ 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, $\alpha$ is a CNF formula and for every implicant $\delta$ of $\exists X. \alpha$ there exists a unit-refutation from $\alpha \land \neg \delta$. 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 $\exists\{a, b, c\}.$ $\beta = \exists\{a, b, c\}. (a \lor b \lor c) \land (a \lor \neg b \lor c) \land (\neg a \lor b \lor c) \land (\neg a \lor \neg b \lor c) \land (\neg a \lor \neg b \lor c) \land (\neg a \lor \neg b \lor c)$. This is an $\exists$URC–C formula, since for any implicate $\delta$ (a clause implied by $d$) unit propagation on $\beta \land \neg \delta$ yields a conflict (denoted as $\beta \land \neg \delta \vdash \bot$). However, $\beta$ 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[V, $\exists$] 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[V, $\exists$] 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 $\text{QDAG}$ 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 $\alpha$ where:

- each leaf node of $\alpha$ is labeled by a literal $l$ over $PS$, or by a Boolean constant $\top$ (always true) or $\bot$ (always false);
- each internal node of $\alpha$ is labeled by a connective $c \in \{\land, \lor, \neg, \to\}$ and has as many children as required by $c$ (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 $\text{QDAG}$; $\text{DAG}$ is the subset of $\text{QDAG}$ 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 $T$ is the complementary literal of literal $l$, so that $\overline{T} = \bot$, $\overline{T} = T$, $\overline{x} = \neg x$ and $\overline{\neg x} = x$. For a literal $l$ different from a Boolean constant, $\text{var}(l)$ denotes the corresponding variable: for $x \in PS$, we have $\text{var}(x) = x$ and $\text{var}(\neg x) = x$. A clause (resp. a term) is a finite disjunction (resp. conjunction) of literals. CLAUSE (resp. TERM) is the subset of $\text{DAG}$ 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 $\alpha$ of $\text{QDAG}$ is called a $\text{QDAG}$ formula. $\text{Var}(\alpha)$ denotes the set of free variables $x$ of $\alpha$, i.e., those variables $x$ for which there exists a leaf node $n_x$ of $\alpha$ labeled by a literal $l$ such that $\text{var}(l) = x$ and there is a path from the root of $\alpha$ to $n_x$ such that no node from it is labeled by $\exists x$. The size $|\alpha|$ of a $\text{QDAG}$ 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 $A \lor B$ from two clauses $A \lor x$ and $\neg x \lor B$, 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 $\alpha$ is a finite sequence of clauses $\delta_1, \ldots, \delta_n$, where each $\delta_i$ is a clause of $\alpha$, or is obtained by unit resolution from two previous clauses of the sequence. We write $\alpha \vdash u \perp$ if there is a unit derivation ending with $\delta_n = c$. We say that $\alpha$ is unit-refutable if $\alpha \vdash u \perp$.

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

**Definition 3 (transformations)** Let $\mathcal{L} \subseteq \text{QDAG}$.  
- $\mathcal{L}$ satisfies CD (conditioning) iff there exists a polytime algorithm that maps every formula $\alpha$ from $\mathcal{L}$ and every consistent term $\gamma$ to a formula from $\mathcal{L}$ that is logically equivalent to the conditioning $\alpha \models \gamma$ of $\alpha$ on $\gamma$, i.e., the formula obtained by replacing each free occurrence of variable $x$ of $\alpha$ by $\top$ (resp. $\bot$) if $x$ (resp. $\neg x$) is a positive (resp. negative) literal of $\gamma$.
- $\mathcal{L}$ satisfies FO (forgetting) iff there exists a polytime algorithm that maps every formula $\alpha$ 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 $\land \mathcal{C}$ (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 $\alpha_1 \land \cdots \land \alpha_n$.
- $\mathcal{L}$ satisfies $\lor \mathcal{C}$ (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 $\alpha_1 \lor \cdots \lor \alpha_n$.
- $\mathcal{L}$ satisfies $\land \mathcal{BC}$ (bounded conjunction) iff there exists a polytime algorithm that maps every pair of formulae $\alpha$ and $\beta$ from $\mathcal{L}$ to a formula of $\mathcal{L}$ that is logically equivalent to $\alpha \land \beta$.
- $\mathcal{L}$ satisfies $\lor \mathcal{BC}$ (bounded disjunction) iff there exists a polytime algorithm that maps every pair of formulae $\alpha$ and $\beta$ from $\mathcal{L}$ to a formula of $\mathcal{L}$ that is logically equivalent to $\alpha \lor \beta$.
- $\mathcal{L}$ satisfies $\neg \mathcal{C}$ (negation) iff there exists a polytime algorithm that maps every formula $\alpha$ 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 $\text{QDAG}$. $\mathcal{L}_1$ 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 $\alpha \in \mathcal{L}_1$, there exists an equivalent formula $\beta \in \mathcal{L}_2$ 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 $\text{QDAG}$. $\mathcal{L}_1$ is said to be polynomially translatable into $\mathcal{L}_2$, noted $\mathcal{L}_1 \preceq_p \mathcal{L}_2$, 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 $\mathcal{L}_1$ is also supported in polynomial time in $\mathcal{L}_1$; and conversely, every query which is not supported in polynomial time in $\mathcal{L}_1$ unless $\mathcal{P} = \mathcal{NP}$ (resp. unless the polynomial hierarchy $\mathcal{PH}$ collapses) cannot be supported in polynomial time in $\mathcal{L}_2$, unless $\mathcal{P} = \mathcal{NP}$ (resp. unless $\mathcal{PH}$ collapses.)

$\sim_s$ (resp. $\sim_p$) is the symmetric part of $\leq_s$ (resp. $\leq_p$) $\leq_s$ (resp. $\leq_p$) 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. $\mathcal{L}_1 \leq_s \mathcal{L}_2$ means that $\mathcal{L}_1$ is not at least as succinct as $\mathcal{L}_2$ unless $\mathcal{PH}$ 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 \( \lor \) and \( \exists x \) with \( x \in PS \). It is called the disjunction closure \( \mathcal{L}[\lor] \) of \( \mathcal{L} \) when the operator is \( \lor \), and the existential closure \( \mathcal{L}[\exists] \) of \( \mathcal{L} \) when the operators are of the form \( \exists x \).

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

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

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

Thus, an element of \( \mathcal{L}[\Delta] \) can be viewed as a tree in which internal nodes are labeled by quantifications of the form \( \exists x \) or by \( \lor \) and leaf nodes are labeled by elements of \( \mathcal{L} \). Accordingly, the formulas \( \alpha_i \) 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 \( \mathcal{L} \subseteq \text{QDAG} \), under very weak conditions on \( \mathcal{L} \) (stability by renaming), only three disjunctive closures are worth considering since \( \mathcal{L} \) shows that for a given characterization results for disjunctive closures. Especially, do not share any common subgraphs.

Internal nodes are labeled by quantifications of the form \( \alpha \), and leaf nodes are labeled by elements of \( \mathcal{L} \). Accordingly, the formulae \( \alpha_i \) considered in item 2 of Definition 6 do not share any common subgraphs.

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

The following lemma shows that focusing on the prime implicates of a CNF formula \( \alpha \) is enough when one wants to determine whether it belongs to \( \mathcal{L}[\lor] \). 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 \( \alpha \) is a \( \text{URC-C} \) formula iff for every prime implicate \( \delta = l_1 \lor \ldots \lor l_k \) of \( \alpha \), we have \( \alpha \wedge \bar{l}_1 \wedge \ldots \wedge \bar{l}_k \models \bot \).

Example 3 The CNF formula \( \alpha = (a \lor b) \land \neg(b \lor c) \land (\neg a \lor c) \) belongs to \( \mathcal{L}[\lor] \). It is a prime implicate of it and we have \( \alpha \wedge \neg c \models \bot \); indeed, the sequence \( \neg c, \neg b \lor c, \neg b, \neg a \lor c, \neg a, a \lor b, \bot \) is a unit refutation of \( \alpha \land \neg c \).

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

These two easy results show that \( \mathcal{L}[\lor] \) satisfies a certain form of stability by simplification.

In order to compare \( \mathcal{L}[\lor] \) with other subsets of CNF, like \( \text{PI} \) (the Blake formulae, also known as prime implicates formulae), the following easy lemma is useful.

Lemma 2 Let \( \alpha \) be a CNF formula containing each of its prime implicates. Then \( \alpha \) is an \( \text{URC-C} \) formula.

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

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

This shows that many interesting subsets of CNF are also subsets of \( \mathcal{L}[\lor] \) (or at least are polynomially translatable into it.) In particular, \( \mathcal{L}[\lor] \) is “bounded” by the two com-
plete languages \(\text{PI}\) and \(\text{CNF}\), in the sense that \[\text{PI} \subset \text{URC-C} \subset \text{CNF}.\]

Both inclusions are strict ones since \(a \land (\neg a \lor b)\) is \(\text{URC-C}\) formula but not a \(\text{PI}\) one, while \((a \lor b) \land (\neg a \lor b) \land (a \lor \neg b) \land (\neg a \land \neg b)\) is a \(\text{CNF}\) formula but not a \(\text{URC-C}\) one. Furthermore, we will show in the following that \(\text{URC-C}\) is strictly more succinct than \(\text{PI}\), while being strictly less succinct than \(\text{CNF}\) unless the \(\text{PH}\) collapses.

We now introduce formally the language \(\text{URC-C}\).

**Definition 8** (\(\text{URC-C}\)) \(\text{URC-C}\) is the subset of \(\text{CNF}[\exists]\) consisting of \(\text{URC-C}\) formulae, and formulae of the form \(\exists X.\alpha\) where \(X\) is a finite subset of \(\text{PS}\) and \(\alpha\) is a \(\text{CNF}\) formula, such that for every implicate \(\delta = l_1 \lor \ldots \lor l_k\) of \(\exists X.\alpha\), we have \(\alpha \land \neg l_1 \land \ldots \land \neg l_k \vdash \bot\).

Obviously enough, \(\text{URC-C}\) is a subset of \(\exists \text{URC-C}\). We also have that \(\text{URC-C}[\exists]\) is a subset of \(\exists \text{URC-C}\). Indeed, consider a \(\text{URC-C}[\exists]\) formula of the form \(\exists X.\alpha\); then for every implicate \(\delta = l_1 \lor \ldots \lor l_k\) of \(\alpha\), we have \(\alpha \land \neg l_1 \land \ldots \land \neg l_k \vdash \bot\). Then the fact that the implicates of \(\exists X.\alpha\) are among the implicates of \(\alpha\) completes the proof.

We have for \(\exists \text{URC-C}\) direct counterparts to Lemmas 1 and 2.

**Lemma 3** A \(\text{CNF}[\exists]\) formula \(\alpha\) is an \(\exists \text{URC-C}\) formula iff for every prime implicate \(\delta = l_1 \lor \ldots \lor l_k\) of \(\alpha\), we have \(\alpha \land \neg l_1 \land \ldots \land \neg l_k \vdash \bot\).

**Lemma 4** Let \(\alpha\) be a \(\text{CNF}[\exists]\) formula containing each of its prime implicates. Then \(\alpha\) is a \(\exists \text{URC-C}[\exists]\) 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 \(\text{URC-C}, \text{URC-C}[\forall, \exists],\) and \(\exists \text{URC-C}\).

**Relating \(\exists \text{URC-C}[\forall]\) with \(\exists \text{URC-C}\)**

We first explain why considering the disjunctive closures of \(\exists \text{URC-C}\) yields a language that is equivalent to \(\exists \text{URC-C}\) from the tractability and succinctness point of view. While it is obvious that \(\exists \text{URC-C} \equiv \exists \text{URC-C}[\exists]\), we also show that:

**Proposition 1** \(\exists \text{URC-C} \sim_p \exists \text{URC-C}[\forall]\).

As a consequence, we have that \(\exists \text{URC-C} \sim_p \exists \text{URC-C}[\forall, \exists]\). Prop. 1 hides a subtlety. The idea is to eliminate the disjunctions from a \(\exists \text{URC-C}[\forall]\) (or \(\exists \text{URC-C}[\forall, \exists]\)) formulae, in order to construct a new \(\text{CNF}\) 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 \(\lor\) over \(\land\); when two (or, more generally, a fixed number of) \(\text{CNF}\) formulae are considered, this can be achieved in polynomial time (stated otherwise, \(\text{CNF}\) satisfies \(\forall \text{BC}\)), but this does not extend to an unbounded number of \(\text{CNF}\) 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 \(\text{CNF}\) formulae to be disjoined.) However, none of the methods preserves unit-refutation completeness in the general case:

**Example 4** Consider the formula \(\alpha_1 \lor \alpha_2\) where \(\alpha_1 \equiv (\neg a \lor b) \land (a \lor b)\) and \(\alpha_2 \equiv (\neg a \lor d) \land (c \lor d)\). Both \(\alpha_1\) and \(\alpha_2\) are \(\text{URC-C}\) formulae, hence \(\exists \text{URC-C}\) formulae. Using the brute-force approach, \(\alpha_1 \lor \alpha_2\) is turned into the equivalent \(\text{CNF}\) formula \(\alpha_0 = (\neg a \lor b \lor c \lor d) \land (\neg a \lor b \lor c \lor d) \land (a \lor b \lor c \lor d) \land (a \lor b \lor c \lor d)\). It turns out that \(\alpha_0\) does not belong to \(\exists \text{URC-C}\) since the clause \((b \lor d)\) is an implicate of \(\alpha_1 \lor \alpha_2\) but \(\alpha_0 \land \neg b \land \neg d \not\vdash \bot\). Similarly, to express \(\alpha_1 \lor \alpha_2\) in \(\text{CNF}\) using Tseitin encoding, we can introduce two new variables \(\tau_1, \tau_2\) that capture the truth value of each disjunct. We obtain the \(\text{CNF}\) formula \(\alpha_T = \tau_1, \tau_2, (\tau_1 \lor \tau_2) \land (\neg \tau_1 \lor a \lor b) \land (\neg \tau_2 \lor c \lor d) \land (\neg \tau_2 \lor c \lor d)\).

Again, this formula \(\alpha_T\) is not \(\exists \text{URC-C}\): the clause \((b \lor d)\) is an implicate of it but \(\alpha_T \land \neg b \land \neg d \not\vdash \bot\).

To prove Prop. 1 we need a more sophisticated encoding that produces an \(\exists \text{URC-C}\) 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 \(\text{CNF}\) formula \(\alpha\) and a selected clause \((d_1 \lor \cdots \lor d_p)\) of \(\alpha\), a constructive disjunction encoding of \(\alpha\) w.r.t. the selected clause is a \(\text{CNF}\) formula \(\delta\) with \(\text{Var}(\delta) \supseteq \text{Var}(\alpha)\) such that, given any set of literals \(l_1 \cdots l_k\) built on \(\text{Var}(\alpha)\):

1. \(\alpha \land l_1 \land \cdots \land l_k\) is satisfiable if \(\delta \land l_1 \land \cdots \land l_k \land d_i \vdash \bot\) for all \(i\) in \(1, \cdots, p\);
2. if \(\forall i \in 1, \cdots, p\), \(\alpha \land l_1 \land \cdots \land l_k \land d_i \vdash \bot\) then \(\delta \land l_1 \land \cdots \land l_k \land d_i \vdash \bot\) for all \(i\) in \(1, \cdots, p\).

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 \(\text{CNF}\) \(\alpha\) we can construct a constructive disjunction encoding of \(\alpha\) 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 \((x \lor y), (\neg x \lor z), (\neg y \lor z)\). Assume we want a constructive disjunction w.r.t. the clause \((x \lor y)\). The encoding introduces variables \(x_1, x_2, y_1, y_1^+, z_1, z_1^+\) 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.)
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: \((x_1^+, \neg x_1 \lor y_1^+, (x_1^+ \lor \neg y_1^+ \lor \neg z_1), (x_1^+ \lor \neg y_1^+ \lor z_1^+), (y_1^+ \lor \neg z_1^+)\);
2. clauses that simulate propagation in the second context under the assumption that \( y \) is true: \((y_2^+, (x_2 \lor y_2^+, (x_2^+ \lor \neg y_2^+ \lor \neg z_2), (x_2^+ \lor y_2^+ \lor z_2^+), (y_2^+ \lor \neg z_2^+))\);
3. clauses that “inject” unit literals from the original formula into each simulation, e.g. \((\neg x \lor x_1^+),(x \lor x_1^+)\) (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. \((\neg x_1^+ \lor \neg x_2^+ \lor x),(\neg x_1 \lor \neg x_2^+ \lor \neg x)\) (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 \( \mathbb{URC}^*[V, 3] \) and \( \exists \mathbb{URC}^* \) are equally tractable. \( \mathbb{URC}^* \) is more tractable than \( \exists \mathbb{URC}^* \) w.r.t. queries (since it offers the same queries as \( \exists \mathbb{URC}^* \) plus \( \mathbb{VA}, \mathbb{IM}, \mathbb{EQ} \) and \( \mathbb{SE} \)). Conversely, \( \exists \mathbb{URC}^* \) is more tractable than \( \mathbb{URC}^* \) w.r.t. transformations since beyond \( \mathbb{VC} \), it offers the very significant \( \mathbb{FO} \) transformation (indeed, forgetting is a key transformation in the number of AI problems, with (among others) applications to diagnosis, planning, reasoning about change, reasoning under inconsistency.) We ignore whether \( \mathbb{URC}^* \) satisfies \( \mathbb{VC} \) or \( \mathbb{SFO} \) (since it satisfies \( \mathbb{CD} \), if it satisfies \( \mathbb{VC} \), then it satisfies \( \mathbb{SFO} \)). Anyway, both transformations are offered by \( \exists \mathbb{URC}^* \).

Let us now report some results concerning the succinctness of \( \mathbb{URC}^*, \mathbb{URC}^*[V, 3] \) and \( \exists \mathbb{URC}^* \).

**Proposition 4** The following succinctness results hold:

1. \( \exists \mathbb{URC}^* \leq_s \mathbb{URC}^*[V, 3] \leq_s \mathbb{URC}^* \leq_s \mathbb{PI} \).
2. \( \mathbb{URC}^* \leq_s \mathbb{CNF} \) and \( \mathbb{CNF} \leq_s \mathbb{URC}^* \).
3. \( \mathbb{URC}^* \leq_s \mathbb{CNF} \) and \( \mathbb{CNF} \leq_s \mathbb{URC}^* \).
4. \( \mathbb{URC}^* \leq_s \mathbb{DFN}, \mathbb{URC}^* \leq_s \mathbb{SDNNF}, \mathbb{DFN} \leq_s \mathbb{SDNNF} \).
5. \( \mathbb{URC}^* \leq_s \mathbb{URC}^*, \mathbb{SDNNF} \leq_s \mathbb{URC}^* \), and \( \mathbb{FBDD} \leq_s \mathbb{URC}^* \).
6. \( \mathbb{URC}^* \leq_s \mathbb{DFN} \).
7. \( \mathbb{URC}^* \leq_s \mathbb{DFN} \).
8. \( \mathbb{URC}^* \leq_s \mathbb{SDNNF} \).
9. \( \mathbb{URC}^* \leq_s \mathbb{d-DFN} \).

Point 1 shows that \( \exists \mathbb{URC}^* \) is at least as succinct as \( \mathbb{URC}^*[V, 3] \), which is strictly more succinct than \( \mathbb{URC}^* \). Especially, since \( \mathbb{URC}^*[V, 3] \) is just as tractable as \( \exists \mathbb{URC}^* \), \( \exists \mathbb{URC}^* \) appears definitely as a language which is at least as interesting as \( \mathbb{URC}^*[V, 3] \) 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 \( \mathbb{URC}^* \) and \( \exists \mathbb{URC}^* \).

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

The remaining points report succinctness results between \( \mathbb{URC}^* \) and \( \exists \mathbb{URC}^* \) and the influential \( \mathbb{DNNF} \) language and some of its subsets. We considered the three subsets \( \mathbb{DFN}, \mathbb{SDNNF}, \mathbb{d-DFN} \) of \( \mathbb{DNNF} \) because existing compilers to \( \mathbb{DNNF} \) actually target one of those three languages; furthermore, they include other interesting subsets of \( \mathbb{DNNF} \) (especially, \( \mathbb{OBDD} \) and its well-known subset \( \mathbb{OBDD} \) are subsets of \( \mathbb{d-DFN} \)).

The results show that, as to succinctness, \( \mathbb{URC}^* \) is incomparable with each of \( \mathbb{DFN}, \mathbb{SDNNF}, \) and \( \mathbb{DNNF} \). For sure, \( \mathbb{URC}^* \) is not at least as succinct as \( \mathbb{d-DFN} \) (hence not at least as succinct as \( \mathbb{DNNF} \)), but whether \( \mathbb{d-DFN} \leq_s \mathbb{URC}^* \) remains an open issue. Again, considering \( \exists \mathbb{URC}^* \) dramatically changes the succinctness picture since \( \exists \mathbb{URC}^* \) is at least as succinct as \( \mathbb{DNNF} \), and is strictly more succinct than each of \( \mathbb{DFN}, \mathbb{SDNNF}, \) and \( \mathbb{DNNF} \).

**Conclusion**

This article shows that \( \exists \mathbb{URC}^* \) is a very interesting target language for knowledge compilation. Indeed, while it offers the same tractable queries and transformations as \( \mathbb{URC}^*[V, 3] \), \( \exists \mathbb{URC}^* \) is at least as succinct as \( \mathbb{URC}^*[V, 3] \). More importantly, it offers the same queries and transformations and is at least as succinct as the influential \( \mathbb{DNNF} \) language from (Darwiche 2001). Furthermore, \( \exists \mathbb{URC}^* \) is strictly more succinct than \( \mathbb{DFN}, \mathbb{SDNNF}, \) and

\(^1\)Unless \( \mathbb{PH} \) collapses for \( \mathbb{d-DFN} \).
∃-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 which are unit-propagation complete, i.e., for every implication δ = l₁ ∨ ... ∨ l_k of α which is not valid, we have α ∧ T₁ ∧ ... ∧ T_k ⊢ l_u ⊥ or α ∧ T₁ ∧ ... ∧ T_k−1 ⊢ 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 α₁, the approach first consists in turning it in polynomial time into an equivalent CNF[3] formula α₂ = ∃X. α₃ using Tseitin encoding. Then α₃ can be turned into an equivalent URC-C formula α₄ using the CNF2URC-C compilation algorithm. By construction, the formula ∃X. α₄ is a URC-C[3] formula equivalent to α₁, hence an ∃URC-C formula. Note that the approach also works if α₁ is a DAG[3] formula.

Appendix

Proof:[Lemma 1] If for every implication δ = l₁ ∨ ... ∨ l_k of α, we have α ∧ T₁ ∧ ... ∧ T_k ⊢ l_u ⊥, then this holds for the prime implications of α. Conversely, let δ be an implication of α. By primality, there exists a prime implication γ = l₁ ∨ ... ∨ l_k of α such that γ ⊢ δ. By assumption, α ∧ T₁ ∧ ... ∧ T_k ⊢ γ ⊥. Since ⊢ is monotonic w.r.t. (i.e., if a unit refutation of a CNF formula β exists, then for every CNF formula γ, a unit refutation of the CNF β ∧ γ exists), adding to α ∧ T₁ ∧ ... ∧ T_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 δ = l₁ ∨ ... ∨ l_k be an implication of α. If α contains each of its prime implications, then there is a clause γ of α s.t. γ ⊢ δ. Obviously enough, we have γ ∧ T₁ ∧ ... ∧ T_k ⊢ l_u ⊥, hence α ∧ T₁ ∧ ... ∧ T_k ⊢ l_u ⊥ 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 ≤ₚ ∃URC-C[3]. We are given a disjunction α of ∃URC-C formulae. We put this disjunction of existentially quantified formulae in prefix form, obtaining a formula β of the form ∃y₁ · · · yₘ. α₁ ∨ · · · ∨ αₚ.

We rewrite β into a conjunctive normal form by a Tseitin encoding as in Example 4: we introduce p variables τ₁, . . . , τₚ, such that each τᵢ will “trigger” formula αᵢ, when set to true. To this effect, for each αᵢ (i ∈ 1, . . . , p) we define βᵢ = ∨c∈Cᵢ (¬τᵢ ∨ c) where Cᵢ denotes the set of clauses of αᵢ. We obtain a formula φ defined as: ∃y₁ · · · yₘ. ∃τ₁ · · · τₚ. ((τ₁ ∨ · · · ∨ τₚ) ∧ ∨i∈1...p βᵢ).

We now build a constructive disjunction encoding of this formula w.r.t. the disjunction (τ₁ ∨ · · · ∨ τₚ), following Def. 9. This transformation introduces new existentially quantified variables, which we here call z₁, . . . , zₚ. We obtain a CNF[3] formula ψ the prefix of which is the form ∃z₁ · · · zₚ. ∃τ₁ · · · τₚ. ∃z₁ · · · zₚ.

We now show that ψ is ∃URC-C. Consider any implication I = (l₁ ∨ ... ∨ l_k) of ψ built over Var(ψ) = Var(α₁). Equivalently it is an implication of α, therefore it is an implication of each formula ∃z₁ · · · zₚ. αᵢ, for i ∈ 1, . . . , p. Let Lᵢ denote the clause containing only the literals from I that are in Var(αᵢ). We have αᵢ ⊢ Lᵢ. Hence, αᵢ ∧ ¬Lᵢ ⊢ ℓ_u ⊥ and αᵢ ∧ ¬I ⊢ ℓ_u ⊥, for each i ∈ 1, . . . , p. By property (2) of the Def. 9 of “constructive disjunction encoding” it follows that ψ ∧ T₁ ∧ · · · ∧ T_k ⊢ ℓ_u ⊥.

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 (¬x ∨ x+), (x ∨ 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 (l₁ ∨ ... ∨ l_k), then for each literal i ∈ 1, . . . , k we add the clause

\[
\left( \bigvee_{j \neq i} \neg l_j \right) \lor l_i
\]

where for each variable x, pos(x) = x+, pos(¬x) = x−, neg(x) = x−, and neg(¬x) = x+. It can be shown that the constructed gadget γ is such that, given any satisfiable set of literals l₁ · · · l_k built on Var(α), we have

1. γ ∧ l₁ ∧ ... ∧ l_k is always satisfiable;
2. γ ∧ l₁ ∧ ... ∧ l_k−1 ⊢ l_u pos(l_k) iff α ∧ l₁ ∧ ... ∧ l_k−1 ⊢ l_u l_k.

To build a constructive disjunction encoding, we duplicate p gadgets γ₁ · · · γₚ. By “duplication” we mean that in each new copy we use fresh indicator variables x−, x+, for all x. The role of each γᵢ is, intuitively, to simulate unit propagation under each and every hypothesis d₁ · · · dₚ of the selected disjunction. For any literal l built on Var(α), and i ∈ 1, . . . , p, let pos(l) and neg(l) denote the positive and negative indicator variables for l, as above, but taken from the i th gadget, γᵢ. We add a unit clause pos(δ_i), for i ∈ 1, . . . , p, to indicate that in each γᵢ we reason under the extra assumption dᵢ. Now for each literal l built on Var(α) we enhance unit propagation by adding a clause δᵢ that triggers exactly when literal l is true under all assumptions d₁ · · · dₚ:

\[
\left( \bigvee_{i \in 1,...,p} \neg pos(δ_i) \right) \lor l
\]

Formula δ is defined as \((∧_{i \in 1,...,p} \gamma_i ∧ pos(δ_i)) \land (∧_{x \in Var(α)} δ_x ∧ δ−x)\) 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[V, E], and URC-C[V, E] 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. $\alpha$ is an implicate of it. If $\perp$ is an implicate of $\alpha$, then $\perp$ is an implicate of $\beta$, hence we cannot have $\beta \perp \perp$ 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: 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 $\exists$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[V, E], 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[V, E]: Direct since neither of these two languages satisfies VA unless $P = NP$.
  - URC-C: Direct since $\exists$URC-C $\subseteq$ CNF and CNF satisfies IM (Darwiche and Marquis 2002).

• EQ, SE:
  - $\exists$URC-C, URC-C[V, E]: Because HORN-C[E] satisfies neither EQ, nor SE unless $P = NP$ (Marquis 2011), and HORN-C[E] is a subset of $\exists$URC-C[V, E] (since HORN-C is a subset of URC-C), and URC-C[E] is a subset of both URC-C[V, E] 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 $L$ of QDAG satisfies CO and CD, then it satisfies MC as well, plus the fact that each of URC-C, URC-C[V, E], 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 $L$ of QDAG satisfies CD, then $L[V, E]$ 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 $\alpha$ is a URC-C formula and $\gamma$ is a consistent term, then $\alpha \upharpoonright \gamma$ is itself a URC-C formula. Consider any prime implicate $l_1 \lor \ldots \lor l_k$ of $\alpha \upharpoonright \gamma$. We have $(\alpha \upharpoonright \gamma) \land \Gamma_1 \land \cdots \land \Gamma_k \models \perp$. This implies $\alpha \land \gamma \land \Gamma_1 \land \cdots \land \Gamma_k \models \perp$. Note that $Var(l_i \lor \ldots \lor l_k) \subseteq Var(\alpha) \setminus Var(\gamma)$, therefore the term $\gamma \land \Gamma_1 \land \cdots \land \Gamma_k$ is consistent and since $\gamma$ is an URC-C we have $\alpha \land \gamma \land \Gamma_1 \land \cdots \land \Gamma_k \models \perp$. By the properties of unit propagation it follows that $(\alpha \upharpoonright \gamma) \land \Gamma_1 \land \cdots \land \Gamma_k \models \perp$.

(The proof relies on the properties: given any $\alpha \land \gamma$ $\models \perp$ since $\alpha \land \gamma$ is a subset of both $\alpha \land \gamma \land \Gamma_1 \land \cdots \land \Gamma_k \models \perp$ as well, plus $\alpha \land \gamma \land \Gamma_1 \land \cdots \land \Gamma_k \models \perp$. We also have $\alpha \land \gamma \models \perp$ iff $\alpha \land \gamma \models \perp$.)

The proof for $\exists$URC-C is similar: if we are given an $\exists$URC-C formula $\alpha$ and a consistent term $\gamma$, then any of the prime implicants of $\alpha \upharpoonright \gamma$ 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 $\delta$ of $\alpha$, we have $\alpha \land \delta \models \perp$. Whatever $Y$ (a finite subset of $PS$), since the implicants of $\exists Y \alpha$ are also implicants of $\alpha$, we have as expected that for every implicate $\delta$ of $\exists Y \alpha$, $\alpha \land \delta \models \perp$ holds.
  - URC-C[V, E]: Obvious since this language is an existential closure.
  - URC-C: Direct since URC-C[E] is a subset of URC-C.

• SFO: Obvious since each of $\exists$URC-C and URC-C[V, E] satisfies FO.

• $\land C$: Direct since neither of these languages satisfies $\land C$ unless $P = NP$.

• $\land ABC$: Comes from the fact that every CNF formula $\alpha$ can be turned in polynomial time into a conjunction $\beta \land \gamma$ of two URC-C formulae (hence $\beta$ and $\gamma$ are $\exists$URC-C, and URC-C[V, E] formulae as well) such that $\beta \land \gamma$ is consistent iff $\alpha$ is consistent, and the fact that each of these languages satisfies CO, while CNF satisfies it only if $P = NP$. Indeed, let $\alpha$ be a CNF formula over $n$ variables $x_1, \ldots, x_n$. Let $\beta$ be the MONO-C formula obtained by replacing every positive literal $x_i$ in $\alpha$ by the negative literal $\lnot x_i$ (where each $\lnot x_i$ is a fresh variable), conjoined with $n$ additional negative clauses $\lnot x_i \lor \lnot x_i$ (i.e., $i \in \{1, \ldots, n\}$). Let $\gamma$ be the MONO-C formula $\bigwedge_{i=1}^{n} x_i \lor \lnot x_i$, $\beta$ and $\gamma$ are MONO-C formulae, hence URC-C formulae. Finally, by construction, $\alpha$ is consistent iff $\beta \land \gamma$ is consistent. This concludes the proof.

• $\lor C$:
  - $\exists$URC-C: Comes from the fact that $\exists$URC-C[V] and $\exists$URC-C are polynomially equivalent.
  - URC-C[V, E]: By construction, as a disjunctive closure based on $\lor$.

• URC-C: Every term is a URC-C formula. If $\lor C$ 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 $\not\subseteq$ DNF (Darwiche and Marquis 2002), contradiction.
• vBC: Obvious since each of the languages satisfies vC.
• ¬C:
  - \text{URC-C}, \text{URC-C}[v, 3]: Every term is an URC-C formula, hence an URC-C formula. As a consequence, DNF is a subset of each of URC-C[v, 3] and of \text{URC-C}[v] which is polynomially equivalent to \text{URC-C}. Assume that one of \text{URC-C}, \text{URC-C}[v, 3], say \mathcal{L}, satisfies ¬C. Then for every DNF formula \(\alpha\) we could compute in polynomial time an \(\mathcal{L}\) formula \(\beta\) equivalent to ¬\(\alpha\). Since \(\mathcal{L}\) satisfies CO, we could check in polynomial time whether \(\beta\) is consistent or not. But \(\beta\) is inconsistent iff \(\alpha\) is valid, and as a consequence, we would have \(P = \text{NP}\).
  - URC-C: Towards a contradiction: consider the following CNF formula \(\alpha = \bigwedge_{i=0}^{n-1} (x_{2i} \lor x_{2i+1})\); this is a \text{MONO-C} formula hence a URC-C formula. Suppose that URC-C satisfies ¬C. Then a URC-C formula equivalent to ¬\(\alpha\) could be computed in time polynomial in the size of \(\alpha\). This is impossible since the formula ¬\(\alpha\) (equivalent to the DNF formula \(\bigvee_{i=0}^{n-1} (\neg x_{2i} \land \neg x_{2i+1})\)) has no CNF representation of size polynomial in \(n\) (Darwiche and Marquis 2002).

Proof:[Proposition 4]

1. We first have the inclusions \(\Pi \subseteq \text{URC-C} \subseteq \text{URC-C}[v, 3] \subseteq \text{URC-C}[v]\). Since URC-C[v, 3] \(\sim_p\) (URC-C[v, 3]) in \(\mathcal{L}\) and since URC-C[3] is a subset of \(\exists \text{URC-C}\), we get that URC-C[v, 3] is polynomially translatable into \(\exists \text{URC-C}[v]\), which is polynomially equivalent to URC-C. Hence URC-C[v, 3] is polynomially translatable into \(\exists \text{URC-C}\). Altogether, this shows that \(\exists \text{URC-C} \subseteq \exists \text{URC-C}[v, 3] \subseteq \exists \text{URC-C} \subseteq \Pi\).

Proposition 6 from (Bordeaux and Marques-Silva 2012) shows that \(\Pi \not\subseteq \text{UPC-C}\). Since UPC-C \(\subseteq\) URC-C, we get that \(\Pi \not\subseteq \text{URC-C}\).

Now, every term is a URC-C formula. As a consequence, DNF is a subset of URC-C[v, 3], so that we have URC-C[v, 3] \(\not\subseteq\) DNF. Since URC-C is a subset of CNF, we have \(\text{CNF} \subseteq \text{URC-C}\). Hence, if URC-C \(\subseteq\) URC-C[v, 3] were the case, then by transitivity of \(\subseteq\), we would have that \(\text{CNF} \subseteq\) DNF, which is known to be false (Darwiche and Marquis 2002).

2. Since URC-C \(\subseteq\) CNF, we have \(\text{CNF} \subseteq \text{URC-C}\). The fact that URC-C \(\not\subseteq\) 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 \(\not\subseteq\) CNF and \(\exists \text{URC-C} \subseteq \text{URC-C}\), we get that \(\exists \text{URC-C} \not\subseteq\) CNF. Now, we have that URC-C \(\not\subseteq\) DNF. If CNF \(\subseteq\) \(\exists \text{URC-C}\) were the case, then by transitivity of \(\subseteq\), we would have \(\text{CNF} \subseteq\) DNF, which is known to be false (Darwiche and Marquis 2002).

4. First, URC-C \(\not\subseteq\) DNF since \(\text{CNF} \not\subseteq\) URC-C and CNF \(\not\subseteq\) DNF (Darwiche and Marquis 2002). Then URC-C \(\not\subseteq\) SDNF since \(\text{CNF} \not\subseteq\) URC-C, SDNF \(\not\subseteq\) OBDD\(<\), OBDD\(<\) \(\not\subseteq\) CNF (Darwiche and Marquis 2002). Finally, URC-C \(\not\subseteq\) d-DNF since \(\text{CNF} \not\subseteq\) URC-C, d-DNF \(\not\subseteq\) OBDD\(<\), and CNF \(\not\subseteq\) OBDD\(<\) (Darwiche and Marquis 2002).

5. DNF \(\not\subseteq\) URC-C, SDNF \(\not\subseteq\) URC-C, and FBDD \(\not\subseteq\) URC-C. First, DNF \(\not\subseteq\) URC-C because URC-C \(\not\subseteq\) PI, but DNF \(\not\subseteq\) PI (Darwiche and Marquis 2002). Then SDNF \(\not\subseteq\) URC-C because the circular bit shift functions do not have polynomial-sized SDNF representations (Pipatsrisawat and Darwiche 2010), but have polynomial-sized KROM-C representations (hence polynomial-sized PI representations.) Finally, FBDD \(\not\subseteq\) URC-C because URC-C \(\not\subseteq\) PI, but FBDD \(\not\subseteq\) PI (Darwiche and Marquis 2002).

6. (Jung et al. 2008) shows that DNNF is polynomially translatable into \(\exists\text{URC-C}\).

7. Since DNF is a subset of URC-C[v, 3] and since URC-C[v, 3] \(\not\subseteq\) DNF, we get that \(\exists\text{URC-C} \not\subseteq\) DNF. That DNF \(\not\subseteq\) \(\exists\text{URC-C}\) comes from the fact that \(\exists\text{URC-C} \not\subseteq\) URC-C, and DNF \(\not\subseteq\) URC-C.

8. \(\exists\text{URC-C} \not\subseteq\) SDNNF comes from the fact that SDNNF is a subset of DNNF and that \(\exists\text{URC-C} \not\subseteq\) DNNF. That SDNNF \(\not\subseteq\) \(\exists\text{URC-C}\) comes from the fact that \(\exists\text{URC-C} \not\subseteq\) URC-C, and SDNNF \(\not\subseteq\) URC-C.

9. \(\exists\text{URC-C} \not\subseteq\) d-DNNF comes from the fact that d-DNNF is a subset of DNNF and that \(\exists\text{URC-C} \not\subseteq\) DNNF. That d-DNNF \(\not\subseteq\) \(\exists\text{URC-C}\) comes from the fact that d-DNNF \(\not\subseteq\) DNF (Darwiche and Marquis 2002) and that \(\exists\text{URC-C} \not\subseteq\) DNF (which comes also directly from the inclusion DNF \(\subseteq\) URC-C[v, 3] and that URC-C[v, 3] is polynomially translatable into \(\exists\text{URC-C}\).

References


