Constraint CNF: SAT and CSP Language Under One Roof

Broes De Cat[1] **and Yuliya Lierler**[2]

1 **Independent Researcher, Londerzeel, Belgium**
`broes.decat@gmail.com`

2 **University of Nebraska at Omaha, Omaha, USA**
`ylierler@unomaha.edu`

Abstract

A new language, called constraint CNF, is proposed. It integrates propositional logic with constraints stemming from constraint programming. A family of algorithms is designed to solve problems expressed in constraint CNF. These algorithms build on techniques from both propositional satisfiability and constraint programming. The result is a uniform language and an algorithmic framework, which allow us to gain a deeper understanding of the relation between the solving techniques used in propositional satisfiability and in constraint programming and apply them together.

1998 ACM Subject Classification I.2.4 Knowledge Representation Formalisms and Methods

Keywords and phrases Propositional Satisfiability, Constraint Programming

Digital Object Identifier 10.4230/OASIcs.ICLP.2016.12

1 Introduction

Propositional satisfiability (SAT) and constraint programming (CP) are two areas of automated reasoning that focus on finding assignments that satisfy declarative specifications. However, the typical declarative languages, solving techniques and terminology in both areas are quite different. As a consequence, it is not straightforward to see their relation and how they could benefit from eachother. In this work, we introduce a language called constraint CNF, which will allow a formal study of this relation. We propose a graph-based algorithmic framework suitable to describe a family of algorithms designed to solve problems expressed in constraint CNF or, in other words, to find models of constraint CNF formulas. The described algorithms build on ideas coming from both SAT and CP. We view constraint CNF as a uniform, simple language that allows us to conglomerate solving techniques of SAT and CP.

The idea of connecting CP with SAT is not novel. Many solving methods investigated in CP fall back on realizing the connection between the two fields and, in particular, on devising translations from constraint satisfaction problem (CSP) specifications to SAT problem specifications, e.g. [12]. Also, methods that combine CP and SAT in a more sophisticated manner exist [10, 2]. Somewhat orthogonal to these efforts is constraint answer set programming (CASP) [5], which attempts to enhance the SAT-like solving methods that are available for processing logic programs under stable model semantics with CP algorithms. It is reasonable to believe that the two distinct research areas CASP and CP, coming from different directions, move towards a common ground. Yet, capturing the common ground is difficult. Research on SAT, CP, CASP each rely on their established terminology and classical results in earlier literature. This makes it difficult to borrow on the knowledge discovered in one of the communities and yet not available in another. Here we undertake the effort of introducing a language that can serve as a unifying ground for the investigation in different automated reasoning communities. We believe that this language will foster and promote new insights and breakthroughs in research communities that consider the computational task of model building.

The paper starts by introducing syntax and semantics of constraint CNF and relating the language to propositional logic and constraint satisfaction problems. We then adapt a graph-based framework, pioneered by Nieuwenhuis et al. [9] for describing backtrack-search algorithms, and design a family of algorithms suitable to solve problems expressed in constraint CNF. We conclude the paper by discussing specific instantiations of such algorithms.

2 Constraint CNF

A *domain* is a set of values. For example, the *Boolean* domain consists of truth values $\mathbb{B} = \{\mathbf{t}, \mathbf{f}\}$, whereas some possible *integer* domains include $\mathbb{Z}$, $\mathbb{N}$, and $\{1, 2, 4\}$. A *signature* is a set of *function symbols*. Each function symbol f is assigned

- a nonnegative integer called the *arity* and denoted ar^f, and
- a non-empty domain for every integer i such that $0 \leq i \leq ar^f$, denoted $\hat{d}^f$ when $i = 0$ and $\hat{d}^f_i$ when $1 \leq i \leq ar^f$.

We call $\hat{d}^f$ the *range* of f. In addition, a function symbol f of nonzero arity can be assigned a specific function from $\hat{d}^f_1 \times \cdots \times \hat{d}^f_n$ to $\hat{d}^f$, in which case we say that f is *interpreted* and denote its interpretation by ι^f. A function symbol is *Boolean* if its range is Boolean. A *propositional symbol* is a 0-ary Boolean function symbol. For a signature Σ, its *domain* is defined as the union of all involved domains. A signature is *finite-domain* when its domain is finite.

For signature Σ, *terms* over Σ are defined recursively:

- a 0-ary function symbol is a term, and
- for an n-ary ($n > 0$) function symbol f, if $t_1, \ldots, t_n$ are terms then $f(t_1, \ldots, t_n)$ is a term.

For a term τ of the form $f(t_1, \ldots, t_n)$, by τ^0 we denote its function symbol f. We define the *range* of a term τ, denoted as $\hat{d}^\tau$, as the domain $\hat{d}^{\tau^0}$. For a term $f(t_1, \ldots, t_n)$, we associate each argument position $1 \leq i \leq n$ with the domain $\hat{d}^f_i$ and assume that $\hat{d}^{t_i} \subseteq \hat{d}^f_i$. A term is *Boolean* if its range is (a subset of) the Boolean domain.

Let Σ be a signature. An *atom* over Σ is a Boolean term over Σ, a *literal* over Σ is an atom a over Σ or its negation $\neg a$. A *clause* over Σ is a finite nonempty set of literals over Σ. A *formula* over Σ is a finite set of clauses over Σ. We sometime drop the explicit reference to the signature. For an atom a we call its negation $\neg a$ a *complement.* For a literal $\neg a$ we call an atom a a *complement.* By $\bar{l}$, we denote the complement of a literal l.

▶ **Example 1.** Consider the following sample problem specification. *Two items, named* 1 *and* 2*, are available for sale, with associated prices of* 100 *and* 500*. We want to buy at least one item, but we should not exceed our budget of* 200. A possible signature to express these statements consists of 0-ary function symbols i_1, i_2 and bgt (abbreviating budget), unary function symbols pr (abbreviating price), buy, and binary interpreted function symbols $+$ and $\leq$, so that

$$\hat{d}^{i_1} = \{1\}, \hat{d}^{i_2} = \{2\}, \hat{d}^{pr} = \{100, 500\},$$
$$\hat{d}^{bgt} = \{200\}, \hat{d}^{buy} = \hat{d}^{\leq} = \mathbb{B},$$
$$\hat{d}^{+} = \{200, 600, 1000\},$$
$$\hat{d}^{pr}_1 = \hat{d}^{buy}_1 = \{1, 2\}, \hat{d}^{\leq}_1 = \{200, 600, 1000\},$$
$$\hat{d}^{+}_1 = \hat{d}^{+}_2 = \{100, 500\}, \hat{d}^{\leq}_2 = \{200\}.$$

Function symbols $+$ and $\leq$ are assigned respective arithmetic functions. We name the described signature Σ_1. In the following, terms that are composed using these functions are often written in a common infix-style. We also drop parenthesis following the common conventions. For instance, expression

$$pr(i_1) + pr(i_2) \leq bgt$$

stands for the term

$$\leq (+(pr(i_1), pr(i_2)), bgt).$$

The requirements of the problem are expressed by formula φ_1, consisting of four clauses:[1]

$$\{buy(i_1), buy(i_2)\} \tag{1}$$
$$\{\neg buy(i_1), \neg buy(i_2), pr(i_1) + pr(i_2) \leq bgt\} \tag{2}$$
$$\{\neg buy(i_1), pr(i_1) \leq bgt\} \tag{3}$$
$$\{\neg buy(i_2), pr(i_2) \leq bgt\}. \tag{4}$$

Intuitively, clause (1) expresses that at least one item is bought. Clause (2) states that if two items are bought then the sum of their prices should not exceed the budget. Clauses (3) (respectively, clause (4)) state that if an item i_1 (respectively, item i_2) is bought then its price should not exceed the budget.

For an n-ary function symbol f, we call any function $\hat{d}_1^f \times \ldots \times \hat{d}_n^f \mapsto 2^{\hat{d}^f}$ *approximating*. For instance, for the function symbol pr of Σ_1 defined in Example 1, functions α_1, α_2, α_3, presented below

$$\begin{array}{llll} \alpha_1: & (1) \mapsto \{100\} & \alpha_2: & (1) \mapsto \{100\} \\ & (2) \mapsto \{100, 500\} & & (2) \mapsto \{500\} \\ \alpha_3: & (1) \mapsto \{100\} & & \\ & (2) \mapsto \emptyset & & \end{array}$$

are approximating. An approximating function is *defining* if, for all possible arguments, it returns a singleton set. For example, function α_2 is defining. We identify defining functions with functions that, instead of a singleton set containing a domain value, return the domain value itself. Thus, we can represent α_2 as a function that maps 1 to 100 and 2 to 500. We call a function *inconsistent* if for some arguments it returns the empty set. Function α_3 is inconsistent. We say that a function $\alpha : D_1 \times \cdots \times D_n \mapsto 2^D$ *reifies* a function $\alpha' : D_1 \times \cdots \times D_n \mapsto 2^D$ if for any n-tuple $\vec{x} \in D_1 \times \cdots \times D_n$ it holds that $\alpha(\vec{x}) \subseteq \alpha'(\vec{x})$. For example, functions α_2, α_3 reify α_1, and α_3 reifies α_2. The reification-relation is transitive and reflexive.

We are now ready to define a semantics of the constraint CNF formulas. An *interpretation* I over a signature Σ consists of an approximating function for every function symbol in Σ; for a function symbol $f \in \Sigma$, by f^I we denote the approximating function of f in I.

We call f^I an *interpretation of a function symbol* f in I. An interpretation that contains only defining functions is *total*. An interpretation that contains an inconsistent function is

[1] By introducing additional function symbols, the representation can be made more elaboration tolerant with regards to increasing the number of items. For simplicity of the example, a different representation was chosen here.

inconsistent. For interpretations I and I' over Σ we say that I *reifies* I' (or, I is a *reification of* I') if for every function symbol $f \in \Sigma$, f^I reifies $f^{I'}$.

Let Σ be a signature, τ be a term over Σ, and I be a total interpretation over Σ. By τ^I we denote the value *assigned* to a term τ by I, defined recursively as

- f^I if τ has the form f, and
- $f^I(t_1^I, \dots, t_n^I)$ if τ has the form $f(t_1, \dots, t_n)$.

We say that I *satisfies*

- an atom a over Σ, denoted $I \models a$, if $a^I = \mathbf{t}$,
- a literal $\neg a$ over Σ, denoted $I \models \neg a$, if $a^I = \mathbf{f}$,
- a clause $C = \{l_1, \dots, l_n\}$ over Σ, denoted $I \models C$, if I satisfies any literal l_i in C,
- a formula $\varphi = \{C_1, \dots, C_n\}$ over Σ, denoted $I \models \varphi$, if I satisfies every clause C_i in φ.

We say that I is a *model* of a formula φ if

- I satisfies φ and
- for any interpreted function symbol f in Σ, f^I coincides with ι^f.

We say that formula φ is satisfiable when φ has a model and *unsatisfiable*, otherwise.

▶ **Example 2** (Continued from Example 1). Consider signature Σ_1 and formula φ_1. All models of φ_1 interpret function symbols i_1, i_2, *bgt* as follows:

$$i_1 \quad [\mapsto 1] \quad i_2 \quad [\mapsto\ 2] \quad bgt \quad [\mapsto\ 200].$$

They differ in their interpretation of *pr* and *buy*. Indeed, there are five models, one of which is the following:

$$\begin{array}{lll} model_1 & pr & [(1) \mapsto 100, (2) \mapsto 500] \\ & buy & [(1) \mapsto \mathbf{t}, (2) \mapsto \mathbf{f}]. \end{array}$$

2.1 Relation to propositional logic and constraint programming

It is easy to see that in case when the signature is composed only of propositional symbols, constraint CNF formulas coincide with classic propositional logic formulas in conjunctive normal form (classic CNF formulas). Indeed, we can identify a clause $\{l_1, \dots, l_n\}$ in constraint CNF with a clause $l_1, \cdots, l_n$ in propositional logic, whereas a constraint CNF formula corresponds to conjunction of its elements in propositional logic.

A *constraint satisfaction problem* (CSP) is a triple $\langle V, D, C\rangle$, where V is a set of variables, D is a finite set of values, and C is a set of constraints. Every constraint is a pair $\langle (v_1, \dots, v_n), R\rangle$, where $v_i \in V$ $(1 \le i \le n)$ and R is an n-ary relation on D. An *assignment* is a function from V to D. An assignment ν *satisfies* a constraint $\langle (v_1, \dots, v_n), R\rangle$ if $(\nu(v_1), \dots, \nu(v_n)) \in R$. A *solution* to $\langle V, D, C\rangle$ is an assignment that satisfies all constraints in C. We map a CSP $\mathcal{C} = \langle V, D, C\rangle$ to an "equivalent" constraint CNF theory $F_{\mathcal{C}}$ as follows. We define a signature $\Sigma_{\mathcal{C}}$ to be composed of

- 0-ary function symbols f_v so that $\hat{d}^{f_v} = D$ for each variable $v \in V$, and
- interpreted n-ary Boolean function symbols f_c, one for each constraint $c = \langle (x_1, \dots, x_n), R\rangle \in C$.

Function ι^{f_c} maps n-tuple d^n in Cartesian product D^n to $\mathbf{t}$ if $d^n \in R$, otherwise ι^{f_c} maps $d^n \in D^n$ to $\mathbf{f}$. For each constraint $c = \langle (v_1, \dots, v_n), R\rangle \in C$, the constraint CNF $F_{\mathcal{C}}$ includes a clause $\{f_c(f_{v_1}, \dots, f_{v_n})\}$. Models of $F_{\mathcal{C}}$ are in one-to-one correspondence with solutions of $\mathcal{C}$: indeed, an interpretation I is a model of $F_{\mathcal{C}}$ if and only if an assignment ν defined as follows $\nu(v) = f_v^I$ for each variable $v \in V$ is a solution to $\mathcal{C}$.

3 DPLL Approach for Constraint CNF

The DPLL decision algorithm [1] and its enhancement CDCL [8] are at the heart of most modern SAT solvers. These algorithms also became a basis for some of the search procedures in related areas such as satisfiability modulo theories [9], answer set programming [4], constraint answer set programming [5] and constraint programming [12, 10]. Here, we present an extension of DPLL that is applicable in the context of the constraint CNF language.

The DPLL algorithm is applied to a classic CNF formula. Let F be such a formula. Informally, the search space of DPLL on F consists of all assignments of the symbols in F. During its application, DPLL maintains a record of its computation state that corresponds to a currently considered family of assignments. When DPLL terminates, it either indicates that given formula F is unsatisfiable or the current state of computation corresponds to a model of F. Nieuwenhuis et al. [9] pioneered a graph-based (or transition system based) approach for representing the DPLL procedure (and its enhancements). They introduced the "Basic DPLL system", which is a graph so that its nodes represent possible states of computation of DPLL, while the edges represent possible transitions between the states. As a result, any execution of the DPLL algorithm can be mapped onto a path of the Basic DPLL system. Here we introduce a graph that captures a backtrack-search procedure for establishing whether a constraint CNF formula is satisfiable, an "entailment graph". We refer to a procedure captured by this graph as "an ENTAIL procedure". The relation between the entailment graph and an ENTAIL algorithm is similar to that between the Basic DPLL system and the DPLL algorithm. Before presenting the entailment graph we introduce two key concepts used in its definition: coherent encoding and entailment.

3.1 Coherent encodings and entailment

We begin by presenting some required terminology. An atom is *propositional* if it is a propositional symbol, a literal is *propositional* if it is a propositional atom or a negation of a propositional atom. We say that a signature is *propositional* if it is composed of propositional symbols only. For a propositional signature Σ, we define $\hat{\Sigma}$ as

$$\Sigma \cup \{\neg a \mid a \in \Sigma\}$$

It is easy to identify interpretations over a propositional signature Σ with sets of propositional literals over $\hat{\Sigma}$. Indeed, consider a mapping $\mathcal{L}$ from interpretations over Σ to $2^{\hat{\Sigma}}$ so that for an interpretation I over Σ, $\mathcal{L}(I)$ results in a set

$$\{f,\ \neg f \mid f^I = \emptyset\} \cup \{f \mid f^I = \mathbf{t}\} \cup \{\neg f \mid f^I = \mathbf{f}\}.$$

$\mathcal{L}^{-1}$ is a mapping from $2^{\hat{\Sigma}}$ to interpretations over Σ so that for a set M of literals over Σ, $\mathcal{L}^{-1}(M)$ is an interpretation where for every symbol $f \in \Sigma$

$$f^I = \begin{cases} \mathbf{t}, & \text{if } f \in M, \neg f \notin M \\ \mathbf{f}, & \text{if } f \notin M, \neg f \in M \\ \emptyset, & \text{if } f, \neg f \in M \\ \mathbb{B}, & \text{otherwise.} \end{cases}$$

A state of the DPLL procedure is meant to capture the family of assignments currently being explored. These families of assignments of classic CNF formulas can be referred to by means of sets of propositional literals. For instance, a state $\{a\ \neg b\}$ over a propositional signature $\{a\ b\ c\}$ intuitively suggests that assignments captured by the sets $\{a\ \neg b\ c\}$ and $\{a\ \neg b\ \neg c\}$ of literals are of immediate interest. The signature of a general constraint CNF formula goes beyond propositional symbols. To adapt the "propositional states" of DPLL to the constraint CNF formalism one has to ensure that the maintained state of computation can be mapped into an interpretation for a signature that includes non-propositional symbols. One approach to achieve this goal is to use auxiliary propositional symbols to encode the state of the approximating functions of non-propositional symbols in the formula's signature. This method is sometimes used by constraint programming solvers, for example, see [11]. We follow this approach in developing ENTAIL procedures.

From now on we assume only finite-domain signatures. We start by presenting a generalized concept of an "encoding" and summarize the important properties it should exhibit to be applicable in the scope of ENTAIL procedures that we present next. In the following section we illustrate that the equality or direct encoding studied in CP satisfies such properties.

An *encoding* is a 4-tuple (Σ, Σ', m, m'), where Σ is a signature, Σ' is a propositional signature, m is a function that maps interpretations in Σ into interpretations in Σ', and m' is a function that maps interpretations in Σ' into interpretations in Σ. We say that an encoding (Σ, Σ', m, m') is *coherent* when the following conditions (properties) hold

1. For a total interpretation I over Σ, $m(I)$ results in a total interpretation over Σ' and $I = m'(m(I))$.
2. For a total interpretation I' over Σ', $m'(I')$ results in either a total interpretation over Σ so that $I' = m(m'(I'))$ or an inconsistent interpretation over Σ.
3. For any interpretations I'_1, I'_2, and a literal l over Σ' such that (i) l is in I'_1 and its complement is in I'_2, and (ii) interpretations $m'(I'_1)$ and $m'(I'_2)$ are consistent, it holds that $m'(I'_1)$ does not reify $m'(I'_2)$.
4. For any consistent interpretations I'_1, I'_2 over Σ' such that I'_2 reifies I'_1, $m'(I'_2)$ reifies $m'(I'_1)$.
5. For any consistent interpretations I_1, I_2 over Σ such that I_2 reifies I_1, $m(I_2)$ reifies $m(I_1)$.
6. For any total interpretation I over Σ, a non-total interpretation I' over Σ' such that I reifies $m'(I')$, and any atom a in Σ' such that neither a nor $\neg a$ in I', it holds that I reifies $m'(I' \cup \{a\})$ or I reifies $m'(I' \cup \{\neg a\})$.
7. For any literal l over Σ', $\{l\} \subseteq m(m'(\{l\}))$.

The properties of coherent encodings allow us to shift between the interpretations in two signatures Σ and Σ' in a manner that proves to be essential to design of ENTAIL procedures for constraint CNF formulas.

Let Σ be a signature, φ be a formula over Σ, I be an interpretation over Σ, and f be a function symbol in Σ. Formula φ *entails* an approximating function f^α, denoted as $\varphi \models f^\alpha$, if for every model J of φ, f^J reifies f^α. Formula φ *entails* an approximating function f^α *with respect to interpretation* I, denoted as $\varphi \models_I f^\alpha$, when for every model J of φ that is a reification of I, f^J reifies f^α and f^α reifies f^I. Formula φ *entails* interpretation I, denoted as $\varphi \models I$ if $\varphi \models g^I$ for every function symbol g in Σ. Formula φ *entails* an interpretation I' over Σ *with respect to interpretation* I, denoted as $\varphi \models_I I'$ if $\varphi \models_I g^{I'}$ for every function symbol g in Σ. We now remark on some properties about entailment: (i) when there is no model of φ that reifies I then any approximating function is entailed w.r.t. I, (ii) when there is at least one model of φ that is a reification of I then no inconsistent approximating function is entailed, (iii) φ entails any interpretation including inconsistent ones when φ has no models, (iv) φ entails any interpretation (including inconsistent) I' w.r.t. I when φ has no models that reify I, and (v) φ entails any interpretation with respect to inconsistent interpretation I.

3.2 Abstract Constraint CNF Solver.

We are now ready to define nodes of the entailment graph and its transitions. For a set $\mathcal{B}$ of propositional atoms (which is also a propositional signature), a *state* relative to $\mathcal{B}$ is either the distinguished state *Failstate* or a (possibly empty) list M of literals over $\mathcal{B}$, where (i) no literal is repeated twice and (ii) some literals are possibly annotated by Δ. For instance, list $a\ \neg a^\Delta$ is a state relative to $\{a, b\}$, while $a\ a^\Delta$ is not. The tag Δ marks literals as *decision* literals. Frequently, we consider M as a set of literals and hence as an interpretation over a propositional signature, ignoring the annotations and the order among its elements. We say that M is *inconsistent* if some atom a and its negation $\neg a$ occur in it. E.g., states $b^\Delta\ \neg b$ and $b\ a\ \neg b$ are inconsistent.

Given an encoding $E = (\Sigma, \Sigma', m, m')$, we define the *entailment graph* $\text{ENT}_{\varphi,E}$ for a formula φ over Σ as follows. The set of nodes of $\text{ENT}_{\varphi,E}$ consists of the states relative to Σ'. The edges of the graph $\text{ENT}_{\varphi,E}$ are specified by four transition rules:

Entailment Propagate:	$M \Rightarrow M\ l$	if $\varphi \models_{m'(M)} I$ and $l \in m(I)$
Decide:	$M \Rightarrow M\ l^\Delta$	if $l \notin M$ and $\bar{l} \notin M$
Fail:	$M \Rightarrow$ *Failstate*	if $\begin{cases} m'(M) \text{ is inconsistent, and} \\ \text{no decision literal is in } M \end{cases}$
Backtrack:	$P\ l^\Delta\ Q \Rightarrow P\ \bar{l}$	if $\begin{cases} m'(P\ l^\Delta\ Q) \text{ is inconsistent,} \\ \text{and no decision literal is in } Q. \end{cases}$

A node (state) in the graph is *terminal* if no edge originates in it. The following proposition gathers key properties of the graph $\text{ENT}_{\varphi,E}$ under assumption that E is a coherent encoding.

▶ **Proposition 3.** *For a signature Σ, a formula φ over Σ, and a coherent encoding $E = (\Sigma, \Sigma', m, m')$,*

(a) *graph $\text{ENT}_{\varphi,E}$ is finite and acyclic,*

(b) *any terminal state M of $\text{ENT}_{\varphi,E}$ other than Failstate is such that $m'(M)$ is a model of φ,*

(c) *state Failstate is reachable from $\emptyset$ in $\text{ENT}_{\varphi,E}$ if and only if φ has no models.*

Thus, to decide whether a CNF formula φ over Σ has a model, it is sufficient to (i) find any coherent encoding $E = (\Sigma, \Sigma', m, m')$ and (ii) find a path leading from node $\emptyset$ to a terminal node M in $\text{ENT}_{\varphi,E}$. If $M = \textit{Failstate}$, φ has no models. Otherwise, M is a model of φ. Conditions (b) and (c) of Proposition 3 ensure the correctness of this procedure, while condition (a) ensures that this procedure terminates. We refer to any algorithm of this kind as an ENTAIL procedure.

An implementation of an ENTAIL algorithm in its full generality is infeasible due to the complexity of the condition of the *Entailment Propagate* transition rule. Yet, for various special cases, efficient methods exist. The DPLL algorithm for classic CNF formulas relies on this observation. Recall that classic CNF formulas can be viewed as constraint CNF formulas over a propositional signature. We now define the graph DP_F that coincides with aforementioned Basic DPLL system. Let E_p denote the encoding (Σ, Σ, id, id), where Σ is a propositional signature and id is an identity function from Σ to Σ. The set of nodes of DP_F coincide with the nodes of ENT_{F,E_p}. The edges of the graph DP_F are specified by the three transition rules of ENT_{F,E_p}, namely, *Decide*, *Fail*, *Backtrack*, and the clause-specific propagate rule

$$\textit{Unit Propagate}:\quad M \Rightarrow M\ l \quad \text{if} \quad \begin{cases} \{l_1, \dots, l_n, l\} \in F \text{ and} \\ \{\overline{l_1}, \dots, \overline{l_n}\} \subseteq M \end{cases}$$

It turns out that if the condition of the transition rule *Unit Propagate* holds then the condition of *Entailment Propagate* in the graph ENT_{F,E_p} also holds. The converse is not true. The DP_F graph is a subgraph of ENT_{F,E_p}. Yet, Proposition 3 holds for the graph DP_F. Proof of this claim was presented in [9, 7].

We now present several incarnations of the $\text{ENT}_{\varphi,E}$ framework that encapsulate the *Unit Propagate* rule of DPLL in a meaningful way.

Let Σ be a signature, $E = (\Sigma, \Sigma', m, m')$ a coherent encoding, φ a formula over Σ, and F a classic CNF formula over Σ'. We say that F *respects* φ when every model I of φ is such that $m(I)$ is also a model of F; we also say that F *captures* φ when F respects φ and every model M of F is such that $m'(M)$ is a model of φ. It is obvious that the graph DP_F can be used to decide whether formula φ has a model when F captures φ. We define a graph $\text{ENT-UP}_{\varphi,E,F}$ as follows: (i) its nodes are the nodes of $\text{ENT}_{\varphi,E}$, and (ii) its edges are defined by the transition rules of $\text{ENT}_{\varphi,E}$ and the transition rule *Unit Propagate*. It turns out that when F respects φ, the graphs $\text{ENT-UP}_{\varphi,E,F}$ and $\text{ENT}_{\varphi,E}$ coincide:

▶ **Proposition 4.** *For a coherent encoding $E = (\Sigma, \Sigma', m, m')$, a formula φ over Σ, a classic CNF formula F over Σ' that respects φ, and some nodes M and $M\ l$ in the graph $\text{ENT}_{\varphi,E}$, if the transition rule Unit Propagate suggests the edge between M and $M\ l$ then this edge is present in $\text{ENT}_{\varphi,E}$ (due to the transition rule Entailment Propagate).*

Consider a new graph $\text{ENT}'_{\varphi,E,F}$ constructed from $\text{ENT}_{\varphi,E}$ by dropping some of its edges. In particular, given a node M in $\text{ENT}_{\varphi,E}$ to which *Unit Propagate* is applicable, we drop all of the edges from M that cannot be characterized by the application of *Unit Propagate*. It turns out that Proposition 3 holds for the graph $\text{ENT}'_{\varphi,E,F}$, when F respects φ. This suggests that the "more respecting" the propositional formula is to a given constraint CNF formula, the more we can rely on the *Unit Propagate* rule of DPLL and the less we have to rely on propagations that go beyond Boolean reasoning. Another interesting propagator based on *Entailment Propagate* is due to the transition rule

$$\textit{Model Check}:\quad M \Rightarrow M\ l \quad \text{if} \quad \begin{cases} M \text{ is a model of } F, \\ \varphi \models_{m'(M)} I, \text{ and } l \in m(I) \end{cases}$$

This propagator is such that it is only applicable to the states that represent total interpretations over Σ'. It is easy to see that any edge due to *Model Check* is also an edge due to *Entailment Propagate*. It turns out that given classic CNF formula F that respects φ, Proposition 3 also holds for the graph $\text{ENT}''_{\varphi,E,F}$ constructed from $\text{ENT}'_{\varphi,E,F}$ by dropping all of the edges not due to *Unit Propagate* or *Model Check*. The essence of this graph is in the following: to adapt the DPLL algorithm for solving a constraint CNF formula φ over signature Σ, it is sufficient to (i) find some coherent encoding of the form $E = (\Sigma, \Sigma', m, m')$, (ii) find some classic CNF formula F over Σ' that respects φ, (iii) apply DPLL to F, and (iv) implement a check that given any model of F can verify whether that model is also a model of φ. Next section presents one specific coherent encoding and a family of mappings that given a constraint CNF formula produces a classic CNF formula respecting it.

4 Equality Encoding

Walsh [12] describes a mapping from CSP to SAT that he calls "direct encoding". Similar ideas are applicable in the realm of constraint CNF for producing a coherent encoding and classic CNF formulas respecting and capturing given constraint CNF formulas. We make this statement precise by (a) defining a coherent "equality" encoding $\mathcal{E}$ and, (b) introducing mappings from a constraint CNF formula φ to classic CNF formulas that respect φ.

For a function symbol f, we denote the Cartesian product $\hat{d}^f_1 \times \cdots \times \hat{d}^f_{ar f}$ by $\hat{D}_f$. For a non-propositional function symbol $f \in \Sigma$, by $f^{\equiv}$ we denote the set of propositional symbols constructed as follows:

$$\{[f^{\vec{x}} \doteq v] \mid \vec{x} \in \hat{D}_f \text{ and } v \in \hat{d}^f\}.$$

For a signature Σ, by $\Sigma^{\equiv}$ we denote the signature that consists of all propositional symbols in Σ and the propositional symbols in $f^{\equiv}$ for every non-propositional function symbol f in Σ. For example, for Σ_1 defined in Example 1 signature $\Sigma_1^{\equiv}$ includes, among others, following elements

$$\begin{aligned}
&[i_1 \doteq 1];\ [i_2 \doteq 2];\ [bgt \doteq 200];\\
&[pr^1 \doteq 100];\ [pr^1 \doteq 500];\ [pr^2 \doteq 100];\ [pr^2 \doteq 500];\\
&[buy^1 \doteq \mathbf{t}];\ [buy^1 \doteq \mathbf{f}];\ [buy^2 \doteq \mathbf{t}];\ [buy^2 \doteq \mathbf{f}];\\
&[+^{100,100} \doteq 200];\ [+^{100,500} \doteq 200];\ [+^{200,100} \doteq 500];\\
&[\leq^{100,200} \doteq \mathbf{t}];\ [\leq^{600,200} \doteq \mathbf{t}];\ [\leq^{100,200} \doteq \mathbf{f}].
\end{aligned}$$

Intuitively, the collection of the symbols of the form $[f^{\vec{x}} \doteq v]$ in $f^{\equiv}$ is meant to "capture" the approximating function of non-propositional function symbol f in Σ by means of approximating functions for the elements of $f^{\equiv}$ in $\Sigma^{\equiv}$.

We now present a mapping ϵ_* from an approximating function α for a non-propositional function symbol f into an interpretation M over signature $f^{\equiv}$: every symbol $[f^{\vec{x}} \doteq v]$ in $f^{\equiv}$ is interpreted as

$$[f^{\vec{x}} \doteq v]^M = \begin{cases} \mathbf{t}, & \text{if } \alpha(\vec{x}) = \{v\} \\ \mathbf{f}, & \text{if } v \notin \alpha(\vec{x}) \\ \mathbb{B}, & \text{otherwise.} \end{cases}$$

For an interpretation I over Σ, by $\epsilon(I)$ we denote the interpretation over $\Sigma^{\equiv}$ constructed as follows (i) for every propositional symbol f in Σ, $f^{\epsilon(I)} = f^I$, and (ii) for every non-propositional function symbol f in Σ, $\epsilon(I)$ includes the elements of $\epsilon_*(f^I)$.

Similarly, for a non-propositional symbol f, we define a mapping $\epsilon^{\equiv}_*$ which, given an interpretation M over $f^{\equiv}$, maps M into an approximating function α for f: for every $\vec{x} \in \hat{D}_f$,

$$\alpha(\vec{x}) = \begin{cases} \emptyset, \text{if } [f^{\vec{x}} \doteq v]^M = [f^{\vec{x}} \doteq v']^M = \mathbf{t} \text{ and } v \neq v' \\ \{v\} \text{ otherwise, if } [f^{\vec{x}} \doteq v]^M = \mathbf{t} \\ \hat{d}^f \setminus \{v \mid [f^{\vec{x}} \doteq v]^M = \mathbf{f}\} \text{ otherwise.} \end{cases}$$

For an interpretation I and signature Σ, by $I[\Sigma]$ we denote the set of all approximating functions of Σ-elements in I: $\{f^I | f \in \Sigma\}$. For a signature Σ and an interpretation M over $\Sigma^{\equiv}$, by $\epsilon^{\equiv}(M)$ we denote the interpretation over Σ constructed as follows (i) for every propositional symbol f in Σ, $f^{\epsilon^{\equiv}(M)} = f^M$, and (ii) for every non-propositional function symbol f in Σ, $f^{\epsilon^{\equiv}(M)} = \epsilon^{\equiv}_*(M[f^{\equiv}])$.

For a signature Σ, we call an encoding $(\Sigma, \Sigma^{\equiv}, \epsilon, \epsilon^{\equiv})$ an *equality* encoding.

▶ **Proposition 5.** *For a signature Σ, its equality encoding is coherent.*

We now present several mappings from constraint CNF formulas to classic CNF formulas based on equality encoding. Consider a formula φ over signature Σ and the equality encoding $\mathcal{E} = (\Sigma, \Sigma^{\equiv}, \epsilon, \epsilon^{\equiv})$. By $F_{\varphi,\Sigma^{\equiv}}$ we denote a propositional formula constructed as the union of the following sets of clauses:

- for every interpreted function symbol f in Σ and every tuple $\vec{x}$ in $\hat{D}_f$, a set consisting of unit clauses over $\Sigma^{\equiv}$ that ensures that f is interpreted according to ι^f:
$$\bigcup_{v \in \hat{d}^f} \{\neg[f^{\vec{x}} \doteq v] \mid \iota^f(\vec{x}) \neq v\} \cup$$
$$\{[f^{\vec{x}} \doteq v] \mid \iota^f(\vec{x}) = v\}.$$
- for every other function symbol f in Σ and every tuple $\vec{x}$ in $\hat{D}_f$ (i) a clause over $\Sigma^{\equiv}$ that ensures that f is associated with an approximating function:
$$\{[f^{\vec{x}} \doteq v] \ \mid \ v \in \hat{d}^f\}.$$
and (ii) a set of clauses over $\Sigma^{\equiv}$ that ensures that each functional symbol is associated with a defining approximating function:
$$\bigcup_{v,v' \in \hat{d}^f, v \neq v'} \{\neg[f^{\vec{x}} \doteq v], \neg[f^{\vec{x}} \doteq v']\}.$$

▶ **Proposition 6.** *For a signature Σ, a constraint CNF formula φ over Σ, and respective equality encoding $\mathcal{E} = (\Sigma, \Sigma^{\equiv}, \epsilon, \epsilon^{\equiv})$, propositional formula $F_{\varphi,\Sigma^{\equiv}}$ (as well as any formula over $\Sigma^{\equiv}$ constructed from $F_{\varphi,\Sigma^{\equiv}}$ by dropping some of its clauses) respects φ.*

This proposition tells us that we can use the graph $\text{ENT}'_{\varphi,E,F}$ and $\text{ENT}''_{\varphi,E,F}$ for verifying whether formula φ is satisfiable.

5 Proofs for "Constraint CNF"

Proof of Proposition 5.

Proof of Property 1. This property is apparent from the constructions of the mappings.

Proof of Property 2. By contradiction. Assume that for a total interpretation I' over Σ', $m'(I')$ resulted in a consistent non-total interpretation I over Σ. Thus, there is a function symbol $f \in \Sigma$ so that for some $\vec{x} \in \hat{D}_f$, $f^I(\vec{x})$ results in a set whose cardinality if greater than 1. From f^I construction it follows that (i) there is no $v \in \hat{d}^f$ such that $[f^{\vec{x}} \doteq v] = \mathbf{t}$ and (ii) there are at least two values $v, v' \in \hat{d}^f$ such that $v \neq v'$, $[f^{\vec{x}} \doteq v] \neq \mathbf{f}$ and $[f^{\vec{x}} \doteq v] \neq \mathbf{f}$. Consequently, atoms $[f^{\vec{x}} \doteq v]^{I} = [f^{\vec{x}} \doteq v']^{I'} = \mathbb{B}$. This contradicts the fact that I' is total.

Proof of Property 3. Consider interpretations I'_1, I'_2, and a literal l over Σ' such that (i) l is in I'_1 and its complement is in I'_2, and (ii) interpretations $\epsilon^{\equiv}(I'_1)$ and $\epsilon^{\equiv}(I'_2)$ are consistent. We show that it holds that $\epsilon^{\equiv}(I'_1)$ does not reify $\epsilon^{\equiv}(I'_2)$. Recall that an interpretation $\epsilon^{\equiv}(I'_1)$ reifies $\epsilon^{\equiv}(I'_2)$ if any function in $\epsilon^{\equiv}(I'_1)$ reifies a corresponding function in $\epsilon^{\equiv}(I'_2)$.

Case 1. Literal l is of the form $[f^{\vec{x}} \doteq v]$. Since $\epsilon^{\equiv}(I'_1)$ is consistent we derive that $[f^{\vec{x}} \doteq v]^{\epsilon^{\equiv}(I'_1)} = \{v\}$. By the definition of $\epsilon^{\equiv}$, $\epsilon^{\equiv}(I'_2)$ satisfies the following requirement $[f^{\vec{x}} \doteq v]^{\epsilon^{\equiv}(I'_2)} \subseteq \hat{d}^f \setminus \{v\}$. We derive that $\epsilon^{\equiv}(I'_1)$ does not reify $\epsilon^{\equiv}(I'_2)$.

Case 2. Literal l is of the form $\neg[f^{\vec{x}} \doteq v]$. By the definition of $\epsilon^{\equiv}$, we derive that $[f^{\vec{x}} \doteq v]^{\epsilon^{\equiv}(I'_1)} \subseteq \hat{d}^f \setminus \{v\}$. Since $\epsilon^{\equiv}(I'_1)$ is consistent, we also derive that $[f^{\vec{x}} \doteq v]^{\epsilon^{\equiv}(I'_1)} \neq \emptyset$. Since $\epsilon^{\equiv}(I'_2)$ is consistent we derive that
$$[f^{\vec{x}} \doteq v]^{\epsilon^{\equiv}(I'_2)} = \{v\} \tag{5}$$

Proof of Property 4. Take any two consistent interpretations I'_1 and I'_2 over $\Sigma^{\equiv}$ such that I'_2 reifies I'_1. We illustrate that $\epsilon^{\equiv}(I'_2)$ reifies $\epsilon^{\equiv}(I'_1)$. This is the case if for every nonpropositional function symbol $f \in \Sigma$, $f^{\epsilon^{\equiv}(I'_2)} \subseteq f^{\epsilon^{\equiv}(I'_1)}$.

Take any nonpropositional function symbol $f \in \Sigma$. Recall that atoms $f^{\equiv}$ are the ones that are used to define approximating function of f via mapping $\epsilon^{\equiv}$. Take any argument list $\vec{x} \in \hat{D}_f$. We illustrate that $f^{\epsilon^{\equiv}(I'_2)}(\vec{x}) \subseteq f^{\epsilon^{\equiv}(I'_1)}(\vec{x})$.

Case 1. I'_2 is such that $[f^{\vec{x}} \doteq v]^{I'_2} = [f^{\vec{x}} \doteq v']^{I'_2} = \mathbf{t}$ for some values $v \neq v'$. Then, $f^{\epsilon^{\equiv}(I'_2)}(\vec{x}) = \emptyset$. Condition $f^{\epsilon^{\equiv}(I'_2)} \subseteq f^{\epsilon^{\equiv}(I'_1)}$ trivially holds.

Case 2. I'_2 is such that $[f^{\vec{x}} \doteq v]^{I'_2} = \mathbf{t}$ for some value v, and there is no other value $v' \neq v$ such that $[f^{\vec{x}} \doteq v']^{I'_2} = \mathbf{t}$. By the $\epsilon^{\equiv}$ construction, $f^{\epsilon^{\equiv}(I'_2)}(\vec{x}) = \{v\}$. Since I'_2 is consistent and I'_2 reifies I'_1 it follows that (i) for v either $[f^{\vec{x}} \doteq v]^{I'_1} = \mathbf{t}$ or $[f^{\vec{x}} \doteq v]^{I'_1} = \mathbb{B}$ and (ii) there is no other value $v' \neq v$ such that $[f^{\vec{x}} \doteq v']^{I'_2} = \mathbf{t}$. Consequently, by $\epsilon^{\equiv}_*$ mapping definition, it is either $f^{\epsilon^{\equiv}(I'_2)}(\vec{x}) = \{v\}$ or $f^{\epsilon^{\equiv}(I'_2)}(\vec{x}) = \{v\} \cup S$ where S is some subset of $\hat{d}^f$. Consequently, condition $f^{\epsilon^{\equiv}(I'_2)} \subseteq f^{\epsilon^{\equiv}(I'_1)}$ holds.

Case 3. I'_2 is such that $[f^{\vec{x}} \doteq v]^{I'_2} = \mathbf{f}$ or $[f^{\vec{x}} \doteq v]^{I'_2} = \mathbb{B}$ for any $v \in \hat{d}^f$. Since I'_2 reifies I'_1, it also holds that $[f^{\vec{x}} \doteq v]^{I'_1} = \mathbf{f}$ or $[f^{\vec{x}} \doteq v]^{I'_1} = \mathbb{B}$ for any $v \in \hat{d}^f$ so that when $[f^{\vec{x}} \doteq v]^{I'_1} = \mathbf{f}$ it follows that $[f^{\vec{x}} \doteq v]^{I'_2} = \mathbf{f}$. By $\epsilon^{\equiv}_*$ construction (forth case apply), and it is apparent that $f^{\epsilon^{\equiv}(I'_2)}(\vec{x}) \subseteq f^{\epsilon^{\equiv}(I'_1)}(\vec{x})$.

Proof of Property 5. Consider consistent interpretations I_1, I_2 over Σ such that I_2 reifies I_1. We illustrate that $\epsilon(I_2)$ reifies $\epsilon(I_1)$. This is the case when for every (propositional) function symbol $f \in \Sigma^{\equiv}$, $f^{\epsilon(I_2)} = f^{\epsilon(I_1)}$ or $f^{\epsilon(I_1)} = \mathbb{B}$. This trivially holds for all propositional symbols f that are in $\Sigma \cap \Sigma^{\equiv}$. We consider here propositional symbols in $\Sigma^{\equiv} \setminus \Sigma$. Consider any symbol a of this kind. Symbol a is of the form $[f^{\vec{x}} \doteq v]$. From the fact that I_2 reifies I_1 following cases are possible:

Case 1. $f^{I_2}(\vec{x}) = f^{I_1}(\vec{x})$. From ϵ-mapping definition it follows that $a^{\epsilon(I_2)} = a^{\epsilon(I_1)}$.

Case 2. $f^{I_2}(\vec{x}) \subset f^{I_1}(\vec{x})$. From ϵ-mapping definition following cases are possible:

Case 2.1. $v \in f^{I_2}(\vec{x})$. It follows that there is also $v' \neq v$ so that $v, v' \in f^{I_1}(\vec{x})$. Consequently, $a^{\epsilon(I_1)} = \mathbb{B}$.

Case 2.2. $v \notin f^{I_2}(\vec{x})$ and $v \notin f^{I_2}(\vec{x})$. From ϵ-mapping definition it follows that $a^{\epsilon(I_2)} = a^{\epsilon(I_1)} = \mathbf{f}$.

Case 2.3. $v \notin f^{I_2}(\vec{x})$ and $v \in f^{I_1}(\vec{x})$.

Case 2.3.1. $f^{I_1}(\vec{x}) = \{v\}$. Then $f^{I_2}(\vec{x}) = \emptyset$. Impossible as I_2 is a consistent interpretation.

Case 2.3.2. Since, cardinality of $f^{I_1}(\vec{x})$ is greater than one and v in $f^{I_1}(\vec{x})$ it follows that $a^{\epsilon(I_1)} = \mathbb{B}$.

Proof of Property 6. Atom a is of the form $[f^{\vec{x}} \doteq v]$. It is easy to see that $\epsilon^{\equiv}(I' \cup \{a\})$ and $\epsilon^{\equiv}(I' \cup \{\neg a\})$, only differ from $\epsilon^{\equiv}(I')$ in how approximation function for function symbol f is defined. Thus, for any other function symbol $f' \neq f$ in Σ, approximating function for f' in I, $f^{I'}$, reifies approximating function for f' in both $\epsilon^{\equiv}(I' \cup \{a\})$ and $\epsilon^{\equiv}(I' \cup \{\neg a\})$. Even more it only differs in how approximating function for f is defined on $\vec{x}$ arguments. We only have to show that approximating function f^I reifies an approximating function for f in $\epsilon^{\equiv}(I' \cup \{a\})$ or in $\epsilon^{\equiv}(I' \cup \{\neg a\})$ for the case of $\vec{x}$. Recall that I reifies $\epsilon^{\equiv}(I')$.

Case 1. $f^I(\vec{x}) = \{v\}$. Then $v \in f^{\epsilon^{\equiv}(I')}(\vec{x})$. From the fact that $a \notin I'$ and $\epsilon^{\equiv}$ construction we derive that there is no single atom of the form $[f^{\vec{x}} \doteq v']$ such that $v' \neq v$ and $[f^{\vec{x}} \doteq v']^{I'} = \mathbf{t}$. From $\epsilon^{\equiv}$ construction, it follows that $f^{\epsilon^{\equiv}(I' \cup \{a\})}(\vec{x}) = \{v\}$. Obviously, $f^I(\vec{x})$ reifies $f^{\epsilon^{\equiv}(I' \cup \{a\})}(\vec{x})$.

Case 2. I, $f^I(\vec{x}) = \{v'\}$ so that $v' \neq v$. Then $v' \in f^{\epsilon^{\equiv}(I')}(\vec{x})$.

Case 2.1. $f^{\epsilon^{\equiv}(I')}(\vec{x}) = \{v'\}$. It is easy to see from $\epsilon^{\equiv}$ construction that $f^{\epsilon^{\equiv}(I' \cup \{\neg a\})}(\vec{x}) = \{v'\}$ as well.

Case 2.2. $f^{\epsilon^{\equiv}(I')}(\vec{x}) = \{v'\} \cup S$ where cardinality $|S| \geq 1$. Since $\neg a \notin I'$, we derive that $v \in f^{\epsilon^{\equiv}(I')}(\vec{x})$. It is easy to see that $f^{\epsilon^{\equiv}(I' \cup \{\neg a\})}(\vec{x}) = f^{\epsilon^{\equiv}(I')}(\vec{x}) \setminus \{v\}$. It holds that $v' \in f^{\epsilon^{\equiv}(I' \cup \{\neg a\})}(\vec{x})$. Thus, $f^{I}(\vec{x})$ reifies $f^{\epsilon^{\equiv}(I' \cup \{a\})}(\vec{x})$.

Proof of Property 7. Recall that $\{l\}$ corresponds to an interpretation over $\Sigma^{\equiv}$ where all but one atom is assigned $\mathbb{B}$.

Case 1. l has the form $[f^{\vec{x}} \doteq v]$. $\epsilon^{\equiv}(\{l\}$ results in interpretation where $f^{\epsilon^{\equiv}(\{l\}}(\vec{x}) = \{v\}$ whereas all other approximating functions as well as approximating function for f on different arguments that $\vec{x}$ are mapped to $\mathbb{B}$. By ϵ construction, $\{l\} \in \epsilon(\epsilon^{\equiv}(\{l\}))$. Indeed, $\epsilon(\epsilon^{\equiv}(\{l\}))$ contains l as well as literals of the form $[f, v', \vec{x}^{f} \doteq]$ or all $v' \in \hat{d}^{f}$, where $v' \neq v$.

Case 2. l has the form $\neg[f^{\vec{x}} \doteq v]$ By the construction of $\epsilon^{\equiv}$ and ϵ, it is easy to see that $\{l\} = \epsilon(\epsilon^{\equiv}(\{l\}))$. ◀

We now present a lemma that captures important conditions that help to illustrate the correctness of Proposition 3.

▶ **Lemma 7.** *For any formula φ, a coherent encoding $E = (\Sigma, \Sigma', m, m')$, and a path from $\emptyset$ to a state $l_1 \ldots l_n$ in* ENT$_{\varphi,E}$*, every model I of φ is such that $l_i \in m(I)$ if I reifies $m'(\{l_j^{\Delta} | j \leq i\})$.*

Proof. By induction on the length of a path. Since the property trivially holds in the initial state $\emptyset$, we only need to prove that all transition rules of ENT$_{\varphi,E}$ preserve it.

Consider an edge $M \Rightarrow M'$ where M is a sequence $l_1 \ldots l_k$ such that every model I of φ is such that $l_i \in m(I)$ if I reifies $m'(\{l_j^{\Delta} | j \leq i\})$.

Entailment Propagate: M' is $M\ l_{k+1}$. Take any model I of φ such that I reifies $m'(\{l_j^{\Delta} | j \leq k+1\})$. It is easy to see that $\{l_j^{\Delta} | j \leq k+1\} = \{l_j^{\Delta} | j \leq k\}$. By the inductive hypothesis, since I reifies $m'(\{l_j^{\Delta} | j \leq k\})$, $M \subseteq m(I)$. We only have to illustrate that $l_{k+1} \in m(I)$. This trivially follows from the application condition of *Entailment Propagate.*

Decide: M' is $M\ l_{k+1}^{\Delta}$. Take any model I of φ such that I reifies $m'(\{l_j^{\Delta} | j \leq k+1\})$. By Property 4 (of coherent encoding), interpretation $m'(\{l_j^{\Delta} | j \leq k+1\})$ reifies $m'(\{l_j^{\Delta} | j \leq k\})$. Since reification is a transitive relation we derive that I reifies $m'(\{l_j^{\Delta} | j \leq k\})$. By the inductive hypothesis $M \subseteq m(I)$. We only have to illustrate that $l_{k+1} \in m(I)$. Obviously $l_{k+1} \in \{l_j^{\Delta} | j \leq k+1\}$.

Since I is a model and hence a total interpretation it may only reify consistent interpretations. Hence, $m'(\{l_j^{\Delta} | j \leq k+1\})$ is consistent. Interpretation $m(I)$ is a total over $\Sigma^{\equiv}$, by Property 1. Thus, either $l_{k+1} \in m(I)$ or $\overline{l_{k+1}} \in m(I)$. Assume $\overline{l_{k+1}} \in m(I)$. By Property 1, $I = m'(m(I))$ and reifies $m'(\{l_j^{\Delta} | j \leq k+1\})$. By Property 3, we derive a contradiction. Thus, $l_{k+1} \in m(I)$.

Fail: Obvious.

Backtrack: M has the form $P\ l_i^{\Delta}\ Q$ where Q contains no decision literals. M' is $P\ \overline{l_i}$. Take any model I of φ such that I reifies $m'(\{l_j^{\Delta} | j < i\})$.

By Property 6 two following cases are possible: I reifies $m'(\{l_j^{\Delta} | j < i\} \cup \{l_i\})$ or I reifies $m'(\{l_j^{\Delta} | j < i\} \cup \{\overline{l_i}\})$.

Case 1. I reifies $m'(\{l_j^\Delta | j < i\} \cup \{l_i\})$. By inductive hypothesis we derive that $M \subseteq m(I)$. Since I is a model and hence a total interpretation, by Property 1 $m(I)$ is total. We derive a contradiction since M is inconsistent interpretation. Hence, Case 2 must hold.

Case 2. I reifies $m'(\{l_j^\Delta | j < i\} \cup \{\overline{l_i}\})$. By Property 4, interpretation $m'(\{l_j^\Delta | j < i\} \cup \{\overline{l_i}\})$ reifies $m'(\{l_j^\Delta | j < i\})$. Since reification is a transitive relation we derive that I reifies $m'(\{l_j^\Delta | j \leq i\})$. By the inductive hypothesis $P \subseteq m(I)$. We only have to illustrate that $\bar{l} \in m(I)$. By Property 4, interpretation $m'(\{l_j^\Delta | j < i\} \cup \{\overline{l_i}\})$ reifies $m'(\{\overline{l_i})\})$ (the fact that set $\{l_j^\Delta | j < i\} \cup \{\overline{l_i}\}$ of literals is consistent follows from the properties of the *Decide* rules and simple inductive argument). Since reification is a transitive relation we derive that I reifies $m'(\{\overline{l_i}\})$. Since I is a model, it follows that $m'(\{\overline{l_i}\})$ is consistent. By Property 5, $m(I)$ reifies $m(m'(\{\overline{l_i}\}))$. By Property 7 $\{\overline{l_i}\} \subseteq m(m'(\{\overline{l_i}\}))$. By Property 1, $m(I)$ is a total interpretation and hence $\{\overline{l_i}\} \subseteq m(I)$. ◀

Proof of Proposition 3. Part (a) is proven following the arguments for Proposition 1 (a) in the paper by Lierler [6].

(b) Consider any terminal state M other than *Failstate*. From the fact that *Decide* is not applicable, we derive that M assigns all literals over Σ'. Similarly, since neither *Backtrack* nor *Fail* is applicable, M is consistent. By Property 2 of coherent encoding, it follows that $m'(M)$ is either (i) a total interpretation over Σ' or (ii) an inconsistent interpretation.

We now show that (i) holds: $m'(M)$ is a total interpretation. Assume the other case (ii): $m'(M)$ is inconsistent. Take any literal l over Σ' not in M (since M is consistent set of literals such l exists). Interpretation $m'(\{l\})$ is such that $\varphi \models_{m'(M)} m'(\{l\})$ since $m'(M)$ is inconsistent (recall that formula entails any interpretation w.r.t. any inconsistent interpretation). By Property 7 of coherent encoding, $l \in m(m'(\{l\}))$. It follows that *Entailment Propagate* is applicable in M. This contradicts the fact that M is terminal. Consequently, $m'(M)$ is a total interpretation.

We now illustrate that $m'(M)$ is a model of φ. By contradiction. Assume $m'(M)$ is not a model. Since $m'(M)$ is a total interpretation there is no other total interpretation that reifies it. Hence, there is no model that reifies $m'(M)$. It follows that φ entails any interpretation w.r.t. $m'(M)$. Consequently, interpretation $m'(\{l\})$ is such that $\varphi \models_{m'(M)} m'(\{l\})$. Following the argument presented in previous paragraph, we derive that rule *Entailment Propagate* is applicable in M that contradicts the fact that M is terminal. Consequently, $m'(M)$ is a model of φ.

(c) Left-to-right: Since *Failstate* is reachable from $\emptyset$, there is a state M without decision literals so that (i) M is inconsistent, and (ii) there exists a path from $\emptyset$ to M. By Lemma 7, any model I of φ is such that $M \subseteq m(I)$. Since I is a model it is also a total interpretation. By Property 1, $m(I)$ is also a total interpretation. This contradicts the facts that $M \subseteq m(I)$ and M is inconsistent. Indeed, there is an element τ in M such that $\tau^M = \emptyset$, whereas $\tau^{m(I)} = \mathbf{t}$ or $\tau^{m(I)} = \mathbf{f}$.

Right-to-left: From (a) it follows that there is a path from $\emptyset$ to some terminal state. By (b), this state cannot be different from *Failstate*, because φ is unsatisfiable. ◀

Proof of Proposition 6. Sketch: The second claim follows immediately from the proof of the former claim. It is sufficient to illustrate a more general statement, i.e., any interpretation I of φ is such that $\epsilon(I)$ is a model of $F_{\varphi,\Sigma^{\equiv}}$. This is easy to see by following the construction of ϵ and illustrating that every clause in $F_{\varphi,\Sigma^{\equiv}}$ is satisfied by $\epsilon(I)$ for any interpretation I. ◀

Proof of Proposition 4. Sketch: Follows from the properties of *Unit Propagate* and the respective formulas. ◀

6 Conclusions

In this paper we introduced the uniform language constraint CNF which integrates languages from SAT and CP. We also introduced a graph-based framework for a class of algorithms for constraint CNF. We generalized the concept of encoding and identified its essential properties. In the future, we will extend the framework with clause learning, non-finite domains, and constraint-based propagation rules as well as investigate the properties of other non-equality encodings available in CP literature. We will also extend constraint CNF to the case of logic programs so that the algorithms behind answer set solvers and constraint answer set solvers can be captured. This will allow us to formulate algorithms stemming from CP and constraint answer set programming in a uniform fashion to clarify their differences and similarities and facilitate cross-fertilization between the fields. An ultimate goal of this work is to illustrate how advanced solvers stemming from different research sub-communities can be captured as an algorithm for solving search problems stated in constraint CNF. The SAT solver MINISAT [3] is a true success story in model search automated reasoning. *MiniSAT Hack-track* has been an official track since 2009 at the SAT competition – a prime research venue for presenting and comparing state-of-the-art SAT solvers and techniques. The MINISAT authors envisioned such a future for the solver. Their motivation behind the development of the solver was to produce a middle-ware for a SAT solver design. This MINISAT middle-ware incorporates major SAT techniques and also allows a simple integration mechanism for investigating new features. We view constraint CNF as a step in the direction of designing middle-ware that incorporates not only advances in SAT but also other related areas.

Acknowledgments. We would like to thank Marc Denecker, Vladimir Lifschitz, and Mirek Truszczynski for valuable discussions related to this work. The work has been partially supported by FRI (Faculty Research International) of the University of Nebraska at Omaha.

References

1 Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, 1962.

2 Broes De Cat, Bart Bogaerts, Jo Devriendt, and Marc Denecker. Model expansion in the presence of function symbols using constraint programming. In *ICTAI*, pages 1068–1075. IEEE, 2013.

3 Niklas Een and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In *SAT*, 2005.

4 Martin Gebser, Benjamin Kaufmann, and Torsten Schaub. Conflict-driven answer set solving: From theory to practice. *Artif. Intell.*, 187:52–89, 2012.

5 Martin Gebser, Max Ostrowski, and Torsten Schaub. Constraint answer set solving. In Patricia M. Hill and David Scott Warren, editors, *ICLP*, volume 5649 of *LNCS*, pages 235–249. Springer, 2009.

6 Yuliya Lierler. Abstract answer set solvers. In *Proceedings of International Conference on Logic Programming (ICLP)*, pages 377–391. Springer, 2008.

7 Yuliya Lierler. Abstract answer set solvers with backjumping and learning. *Theory and Practice of Logic Programming*, 11:135–169, 2011.

8 João P. Marques Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 131–153. IOS Press, 2009.

9 Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *J. ACM*, 53(6):937–977, 2006.

10 Olga Ohrimenko, Peter J. Stuckey, and Michael Codish. Propagation via lazy clause generation. *Constraints*, 14(3):357–391, 2009.

11 Naoyuki Tamura, Akiko Taga, Satoshi Kitagawa, and Mutsunori Banbara. Compiling finite linear CSP into SAT. *Constraints*, 14(2):254–272, 2009.

12 Toby Walsh. Sat v csp. In Rina Dechter, editor, *Principles and Practice of Constraint Programming, CP 2000*, volume 1894 of *Lecture Notes in Computer Science*, pages 441–456. Springer Berlin Heidelberg, 2000.