A New 3-CNF Transformation by Parallel-Serial Graphs [1]

Uwe Bubeck * , Hans Kleine Büning

University of Paderborn, Computer Science Institute, 33098 Paderborn, Germany

Abstract

For propositional formulas we present a new transformation into satisfiability equivalent 3-CNF formulas of linear length. The main idea is to represent formulas as *parallel-serial graphs*. This is a subclass of directed acyclic multigraphs where the edges are labeled with literals and the AND operation (respectively, the OR operation) is expressed as parallel (respectively, serial) connection.

Key words: algorithms, propositional logic, satisfiability, conjunctive normal form

1 Introduction

Proof systems and satisfiability algorithms for propositional formulas have often been designed specifically for formulas in conjunctive normal form (CNF). Transforming an arbitrary propositional formula into a logically equivalent CNF formula is very space consuming. This can be avoided by providing only a satisfiability equivalent formula, which is sufficient for most applications. Recall that logical equivalence $\alpha \approx \beta$ requires that α and β evaluate to the same truth value for each assignment to the variables, whereas satisfiability equivalence $\alpha \approx_{sat} \beta$ only needs α being satisfiable if and only if β is satisfiable.

The transformation of a propositional formula ϕ into a satisfiability equivalent CNF formula ϕ' usually involves significant changes to the structure of ϕ and the addition of new auxiliary variables. In this paper, we present a new transformation that naturally preserves the structure of the input formula.

* Corresponding author: phone +49 5251 603361, fax +49 5251 603338
Email addresses: bubeck@upb.de (Uwe Bubeck), kbcsl@upb.de (Hans Kleine Büning).

[1] Supported by the German Research Foundation (DFG), grant KL 529 / QBF.

Postprint, Information Processing Letters 109(7), 2009, ©*Elsevier*

At the same time, it requires less auxiliary variables than existing approaches and produces no clauses that contain only such new variables. This is not only beneficial to satisfiability solvers or proof systems, but the structure-preserving graph representation on which the transformation is based also provides a new way of proving structural properties of formulas. In addition, the graphs allow a comprehensible visualization of the algorithm and illustrate a close relationship between exponential CNF transformation by distributing terms and linear CNF transformation with auxiliary variables.

Among existing CNF transformations, the *Tseitin procedure* [5] is one of the best known. Given a formula in negation normal form (NNF), the procedure replaces any subformula $\alpha \vee (\beta \wedge \sigma)$ with $(\alpha \vee x) \wedge (\neg x \vee \beta) \wedge (\neg x \vee \sigma)$ for a new variable x. This can be understood as introducing an abbreviation $x \leftrightarrow (\beta \wedge \sigma)$, which can be simplified to $x \rightarrow (\beta \wedge \sigma)$ [4]. The process continues as long as such subformulas exist. The resulting formula is satisfiable if and only if the initial formula is satisfiable, so we have $\phi \approx_{sat} \phi'$, and it can be shown that the length of ϕ' (denoted as $|\phi'|$) is linear in the length of ϕ.

These ideas can be adapted in a straightforward way to quantified Boolean formulas, an extension of propositional logic with quantifiers over propositional variables [2,3]. In general, quantified Boolean formulas have the form $\Phi = Q\phi$ with a quantifier prefix Q and a propositional matrix ϕ, but in this paper, we only consider existentially quantified formulas $\Phi = \exists\phi$. Their semantics is very similar to propositional formulas, because $\exists y\ \phi(y)$ is satisfiable if and only if $\phi(0)$ is satisfiable or $\phi(1)$ is satisfiable. By definition, two quantified Boolean formulas Φ and Ψ are logically equivalent ($\Phi \approx \Psi$) if and only if they evaluate to the same truth value for each truth assignment *to the unquantified (free) variables*, in contrast to the propositional case where all variables are taken into consideration. This allows us to achieve full logical equivalence through the Tseitin procedure: let ϕ be the initial propositional formula and ϕ' the output of the Tseitin procedure with newly introduced variables $x_1, \ldots, x_n$. Then it holds that $\phi \approx \exists x_1 \ldots \exists x_n\ \phi'$. The advantage of this quantified notation is that it becomes immediately clear which variables are the auxiliary variables.

A different approach to CNF transformation is based on the representation of formulas as circuits over $\{\wedge, \vee, \neg\}$. Such a circuit is a directed acyclic graph where the nodes are labeled with $\wedge, \vee$, or $\neg$. It has some source nodes labeled with literals as inputs and exactly one outgoing edge (the sink) to provide an output. Furthermore, we assume that the circuits are in negation normal form (sometimes also called standard form), which means the inputs can be literals x or $\neg x$, but the negation $\neg$ may not occur as an inner node.
It is well-known [1] that we can associate to any such circuit a satisfiability equivalent formula in 3-CNF whose length is linear in the length of the circuit: First, we label each edge with a new variable. Say the set of these variables is $\{y_1, \ldots, y_n, y\}$, where y denotes the sink.

For a $\wedge$-node $\xrightarrow{y_i} \wedge \xrightarrow{y_j} = \xrightarrow{y_r}$, we obtain the clauses $y_i \to y_r$ and $y_j \to y_r$.
For a $\vee$-node $\xrightarrow{y_i} \vee \xrightarrow{y_j} = \xrightarrow{y_r}$, we obtain the clause $(y_i \wedge y_j) \to y_r$.
For an input edge $L \xrightarrow{y_i}$ with a literal L, we add the clause $L \vee y_i$.
Finally, the sink $\xrightarrow{y}$ is represented by the unit clause $\neg y$.
The result is the conjunction of these clauses. Notice that the original literals only occur in one clause each, and all other clauses are purely made of new variables. A lot of these are needed: one for each edge. The graph representation that we now present avoids these problems and preserves more structural information by labeling edges with original literals from the input formula.

2 Parallel-Serial Graphs

With *parallel-serial graphs* (*PS-graphs*), we denote a restricted class of directed acyclic multigraphs. They have only one source and one sink and edges labeled with formulas or literals. We write $(x \to y : \alpha)$ for an edge from x to y labeled with α. PS-graphs are defined inductively by parallel and serial connections.

Definition 1 *(PS-graph)*

(1) Let $V = \{x, y\}$ be a set of nodes, α a label and $E = \{(x \to y : \alpha)\}$. Then $G = (V, E)$ is a PS-graph.
(2) Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be PS-graphs that share the same source x and sink y, with $V_1 \setminus \{x, y\} \cap V_2 \setminus \{x, y\}$ being empty. Then $G = (V_1 \cup V_2, E_1 \cup E_2)$ is a PS-graph.
(3) Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be PS-graphs, such that z is both the sink of G_1 and the source of G_2 and $V_1 \setminus \{z\} \cap V_2 \setminus \{z\}$ is empty. Then $G = (V_1 \cup V_2, E_1 \cup E_2)$ is a PS-graph.

In the definition, we require disjoint sets of vertices to avoid cycles and "crossings" of connections. Examples of well-formed PS-graphs are shown in Figure 1 on page 7.

The semantics of PS-graphs is given implicitly by associating with every PS-graph G a propositional formula Φ_G. It encodes every edge $(u \to w : \alpha)$ as a clause $(u \to w) \vee \alpha \approx \neg u \vee w \vee \alpha$, and two additional unit clauses represent source and sink.

Definition 2 *(PS-graph semantics)*

Let $G = (V, E)$ be a PS-graph with source x, sink y and inner nodes $\mathbf{z} = z_1, ..., z_t$. Then we associate with G the following formula:

$$\Phi_G = \exists x \exists y \exists z_1 ... \exists z_t \; x \wedge \neg y \wedge \bigwedge_{(u \to w : \alpha) \in E} ((u \to w) \vee \alpha)$$

Interestingly, this semantics definition coincides with two different intuitive interpretations of PS-graphs. The first one considers all possible paths from the source to the sink and takes the disjunction of labels on such a path.

Theorem 3 *(Path semantics)*

Let $G = (V, E)$ be a PS-graph with source x and sink y, and let Φ_G be the associated formula. Then

$$\Phi_G \approx \bigwedge_{p \text{ path from } x \text{ to } y} \left(\bigvee_{(u \to w:\alpha) \in p} \alpha \right) .$$

Proof:
Let $p = (x \to z_1 : \alpha_1), (z_1 \to z_2 : \alpha_2), ..., (z_{t-1} \to z_t : \alpha_t), (z_t \to y : \alpha_{t+1})$ be a path from the source x to the sink y. Then Φ_G contains the clauses $x, \neg x \vee z_1 \vee \alpha_1, \neg z_1 \vee z_2 \vee \alpha_2, \ldots, \neg z_{t-1} \vee z_t \vee \alpha_t, \neg z_t \vee y \vee \alpha_{t+1}, \neg y$. Omitting the labels α_i leads to an unsatisfiable formula. Therefore, this set of clauses implies the clause $\alpha_1 \vee \ldots \vee \alpha_{t+1}$. That shows the direction from left to right. For the other direction, let v be a satisfying truth assignment to the right hand formula. We can also apply this truth assignment to the free variables of Φ_G and simplify the resulting formula Φ_G^*. Suppose Φ_G^* is unsatisfiable. Then there must be a chain $x \to z_1, z_1 \to z_2, \ldots, z_{t-1} \to z_t, z_t \to y$ in Φ_G^*, since $\neg y$ is the only negative clause. But that chain represents a path from the source x to the sink y in contradiction to our assumption that the right hand formula is true. $\square$

Another intuitive interpretation of PS-graphs clarifies the difference between parallel and serial connections: parallel connections in the PS-graph can be considered as AND operations, and serial connections can be understood as OR operations.

Theorem 4 *(AND-OR semantics)*

Let $G = (V_1 \cup V_2, E_1 \cup E_2)$ be a PS-graph with source x and sink y that is composed of two smaller PS-graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with associated formulas $\Phi_{G_1} = \exists \phi_{G_1}$ and $\Phi_{G_2} = \exists \phi_{G_2}$.

(1) Let G_1 and G_2 be arranged in a parallel connection where both share the same source x and sink y, with $V_1 \setminus \{x, y\} \cap V_2 \setminus \{x, y\}$ being empty, as in Definition 1 (2).
Then $\Phi_G \approx \exists\ \phi_{G_1} \wedge \phi_{G_2}$.

(2) Let G_1 and G_2 be arranged in a serial connection where z is both the sink of G_1 and the source of G_2 and $V_1 \setminus \{z\} \cap V_2 \setminus \{z\}$ is empty, as in Definition 1 (3).
Then $\Phi_G \approx \exists\ \phi_{G_1} \vee \phi_{G_2}$.

Proof:

(1) In Φ_G, we can duplicate the unit clauses for source and sink and partition the edges into two disjoint sets:

$$\begin{aligned}\Phi_G &= \exists x \exists y \exists \mathbf{z} \quad x \wedge \neg y \wedge \textstyle\bigwedge_{(u \to w:\alpha) \in E}((u \to w) \vee \alpha) \\ &\approx \exists x \exists y \exists \mathbf{z} \quad x \wedge \neg y \wedge \textstyle\bigwedge_{(u \to w:\alpha) \in E_1}((u \to w) \vee \alpha) \\ &\qquad\qquad \wedge\ x \wedge \neg y \wedge \textstyle\bigwedge_{(u \to w:\alpha) \in E_2}((u \to w) \vee \alpha) \\ &\approx \exists x \exists y \exists \mathbf{z} \quad \phi_{G_1} \wedge \phi_{G_2}\end{aligned}$$

(2) According to Theorem 3, we have $\Phi_G \approx \bigwedge_{p \text{ path from } x \text{ to } y} \left(\bigvee_{(u \to w:\alpha) \in p} \alpha\right)$. The construction implies that every path from x to y must pass through z. Consider one single path from x to z. If Φ_G is true, one of the labels on this path is satisfied, or every path from z to y has one satisfied label. Since this applies to each path from x to z, we have the following implication:

$$\Phi_G \Rightarrow \left(\bigwedge_{p \text{ path from } x \text{ to } z} \left(\bigvee_{(u \to w:\alpha) \in p} \alpha\right)\right) \vee \left(\bigwedge_{p \text{ path from } z \text{ to } y} \left(\bigvee_{(u \to w:\alpha) \in p} \alpha\right)\right)$$

The implication in the other direction is obvious: if all paths in one half of the graph are satisfiable then all paths in the whole graph are satisfiable. Thus it follows that both formulas are equivalent. Once again applying Theorem 3, the right hand formula is equivalent to $\Phi_{G_1} \vee \Phi_{G_2}$. The claim follows after moving all quantifiers to the front (notice that $(\exists v\ \phi_{G_1}) \vee (\exists v\ \phi_{G_2}) \approx \exists v\ \phi_{G_1} \vee \phi_{G_2}$). $\square$

This theorem covers cases (2) and (3) of the inductive definition of PS-graphs. For the remaining case (1) of Definition 1, it is easy to see that a graph with only one edge $(x \to y : \alpha)$ simply encodes α, because $\Phi_G = \exists x \exists y\ x \wedge \neg y \wedge (\neg x \vee y \vee \alpha) \approx \alpha$.

3 3-CNF Transformation and Properties

Our approach to CNF transformation is based on the following idea: given an input formula ψ in NNF, we construct a PS-graph G. From G, we can then extract an associated existentially quantified formula $\Phi_G = \exists \phi$, where ϕ is in 3-CNF and $\Phi_G \approx \psi$. According to Definition 2, Φ_G encodes edges $(u \to w : \alpha)$ of a PS-graph into clauses $\neg u \vee w \vee \alpha$. In order for these to be in 3-CNF, the labels α must be literals, so we need a way to map a formula ψ to an equivalent PS-graph $G = ps(\psi)$ that is only labeled with literals. It turns out that Theorem 4 is not only useful for interpreting a PS-graph, but

it can also help us build such a graph, because it establishes a correspondence between the propositional operators AND and OR and the structure of the graph. The idea of our top-down approach is as follows: we start with one single edge that is labeled with the whole formula ψ. Depending on whether ψ is a conjunction or a disjunction of subformulas α_1 and α_2, we then split this edge into two parallel or serial edges labeled with α_1 and α_2. The process continues on the newly created edges until all labels are literals. From G, we can then extract the resulting 3-CNF formula Φ_G. Listing 1 provides the complete transformation procedure.

Listing 1. Algorithm PS-transform

Input propositional formula ψ in NNF

Initialize $G := (V, E) := (\{x, y\}, \{(x \to y : \psi)\})$
// Graph with new nodes x and y and one edge labeled with ψ

while G has an edge $(u \to w : \alpha)$ with a non–literal formula α {
  if $\alpha = \alpha_1 \wedge \alpha_2$
    $E := E \setminus \{(u \to w : \alpha)\} \cup \{u \to w : \alpha_1,\ \ u \to w : \alpha_2\}$
  else if $\alpha = \alpha_1 \vee \alpha_2$
    $E := E \setminus \{(u \to w : \alpha)\} \cup \{u \to z_i : \alpha_1,\ \ z_i \to w : \alpha_2\}$
    $V := V \cup \{z_i\}$ for a new variable z_i
}

$\Phi_G := \exists x \exists y \exists z_1 ... \exists z_t\ \ x \wedge \neg y \wedge \bigwedge_{(u \to w : \alpha) \in E} (\neg u \vee w \vee \alpha)$

Output PS–graph G with associated 3–CNF formula $\Phi_G \approx \psi$

Consider the example $\psi = \neg a \wedge ((b \wedge \neg c) \vee (d \wedge e))$. Figure 1 shows the construction of G, and we obtain the following associated formula:

$$\Phi_G = \exists x \exists y \exists z\ x \wedge \neg y \wedge (\neg x \vee y \vee \neg a) \wedge (\neg x \vee z \vee b) \wedge (\neg x \vee z \vee \neg c) \\ \wedge (\neg z \vee y \vee d) \wedge (\neg z \vee y \vee e)$$

The unit clauses for source and sink can always be propagated. Then our example requires only one helper variable:

$$\Phi_G = \exists z\ \neg a \wedge (z \vee b) \wedge (z \vee \neg c) \wedge (\neg z \vee d) \wedge (\neg z \vee e)$$

On the other hand, the Tseitin procedure needs two quantified variables to deal with this example:

$$\begin{aligned} \psi &\approx \exists x\ \neg a \wedge (x \vee (d \wedge e)) \wedge (\neg x \vee b) \wedge (\neg x \vee \neg c) \\ &\approx \exists x \exists y\ \neg a \wedge (x \vee y) \wedge (\neg x \vee b) \wedge (\neg x \vee \neg c) \wedge (\neg y \vee d) \wedge (\neg y \vee e) \end{aligned}$$

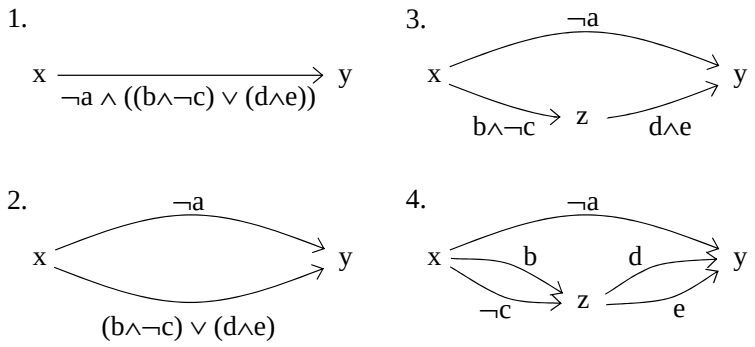

Figure 1. Construction of PS-graph for $\psi = \neg a \wedge ((b \wedge \neg c) \vee (d \wedge e))$

Theorem 5 *(Correctness and linearity)*

Let ψ be a propositional formula in NNF which is mapped by the algorithm PS-transform to the PS-graph $G = ps(\psi)$ with associated formula Φ_G. Then we have $\psi \approx \Phi_G$ and $|\Phi_G| = 3\,|\psi| + 2$.

Proof:
The correctness of PS-transform, i.e. $\psi \approx \Phi_G$, follows directly from Theorem 4 and the subsequent remark about single-edge graphs.
To verify that the associated formula Φ_G has linear length, notice that the number of edges in G equals the number of occurrences of literals in the input formula ψ, which is commonly defined to be the length of ψ. Moreover, Φ_G has one 3-clause for each edge in G, plus two unit clauses for source and sink. □

The algorithm can be modified to produce CNF formulas with arbitrary clause length by labeling edges not only with literals, but also with disjunctions of literals. That means we only have to split edges $(u \to w : \alpha)$ if α is neither a literal nor a disjunction of literals. Or from a bottom-up viewpoint, we can merge two single edges in a serial connection, so that G has no inner nodes for which both indegree and outdegree are 1. With longer clauses, less additional existentially quantified variables are needed: one for each disjunction in the formula tree that has a conjunction as child. This is less than or equal to the number of additional variables that the Tseitin algorithm needs (one for each conjunction that is child of a disjunction). The circuit-based approach that was presented in the introduction clearly needs more additional variables (one for each edge in the circuit). Another advantage of PS-transform is that it guarantees every clause in Φ_G to have at least one literal of the original formula. That should later provide more guidance to solvers or proof systems and avoid extensive reasoning only on helper variables.

Due to the direct correspondence between the propositional connectives and the graph layout, the graph $G = ps(\psi)$ quite naturally represents the structure of the input formula ψ. This is shown by the interesting observation that G encodes both the linear and the exponential CNF transformation of ψ. By

Theorem 3, ψ is equivalent to a CNF that is obtained by adding for every possible path from the source to the sink a clause which contains the disjunction of labels on such a path. This CNF is the result of the application of the distributivity law to ψ. Consider the example from Figure 1. The possible paths are $p_1 = (x \to y : \neg a)$, $p_2 = (x \to z : b), (z \to y : d)$, $p_3 = (x \to z : b), (z \to y : e)$, $p_4 = (x \to z : \neg c), (z \to y : d)$ and $p_5 = (x \to z : \neg c), (z \to y : e)$. Then the resulting formula is $\neg a \wedge (b \vee d) \wedge (b \vee e) \wedge (\neg c \vee d) \wedge (\neg c \vee e)$. We can also observe that if a graph $G = ps(\psi)$ has t possible paths from the source to the sink then ψ is equivalent to a CNF of at most t clauses.

4 Conclusion and Future Work

We have introduced PS-graphs as a new representation of propositional formulas that naturally preserves their structure. This leads to a novel linear 3-CNF transformation which is easy to understand and visualize and nicely illustrates the relationship to exponential CNF transformation by the distributivity law.

An interesting topic for future work is the expressiveness of PS-graphs and their generalization to arbitrary directed acyclic multigraphs with multiple sources and sinks, called *m-DAG*s. When assigning them formulas as in Definition 2 and Theorem 3, are they as expressive as circuits with arbitrary fanout where internal nodes may have more than one outgoing edge? We can show that for every m-DAG, there exists a linear-size circuit with fanout greater than 1 which encodes an equivalent formula. But it is unknown whether the direction from a circuit with arbitrary fanout to a poly-size m-DAG also holds.

References

[1] M. Bauer, D. Brand, M. Fischer, A. Meyer, M. Paterson, A Note on Disjunctive Form Tautologies, SIGACT Newss, 5(2):17–20, 1973.

[2] H. Kleine Büning, T. Lettmann, Propositional Logic: Deduction and Algorithms, Cambridge University Press, Cambridge, UK, 1999.

[3] A. Meyer, L. Stockmeyer, Word Problems Requiring Exponential Time, Preliminary Report, Proc. 5th ACM Symp. on Theory of Computing (STOC'73), pp. 1–9, 1973.

[4] D. Plaisted, S. Greenbaum, A Structure-preserving Clause Form Translation, Journal of Symbolic Computation, 2(3):293–304, 1986.

[5] G. Tseitin, On the Complexity of Derivation in Propositional Calculus, In A. Silenko (ed.): Studies in Constructive Mathematics and Mathematical Logic, Part II, pp. 115–125, 1970.