Representing Reductions of NP-Complete Problems in Logical Frameworks — A Case Study

Jatin Shah
Yale University
jatin.shah@yale.edu

Carsten Schürmann
Yale University
carsten@cs.yale.edu

Abstract

Under the widely believed conjecture P≠NP, NP-complete problems cannot be solved exactly using efficient polynomial time algorithms. Furthermore, any instance of a NP-complete problem can be converted to an instance of another problem in NP in polynomial time. Thus, identifying NP-complete problems is very important in algorithm design and can help computer scientists and engineers redirect their efforts towards finding approximate solutions to these problems. As a first step towards a digital library for NP-complete problems, we describe a case study involving two well-known NP-complete problems 3-SAT and CHROMATIC together with a reduction and the corresponding soundness proof in a logical framework.

1 Introduction

Variables are omnipresent in mathematics and programming languages, computer algebra systems and proof assistants. In the past a significant amount of work has been about the treatment of variables in semantics, type theory, and concurrency theory. In this paper we shift our focus of attention towards a different discipline, complexity theory, and examine the requirements of a logical framework designed for representing graphs, formulas, and reductions.

Representing complexity theoretical problems from the area of NP-complete problems require concise formulations of the underlying mathematical domains, structures, and of course, reductions. Variables range over scalars, graphs, edges, and nodes, with respective operations such as substitution, expansion, deletion, or complement, most of which are not directly supported by any existing logical framework. Of course, one can encode graphs as adjacency matrices, or lists of edges and vertices, but these are not the techniques that we strive to use in this paper. Instead we pursue the design a powerful and elegant logical framework that is tailored for complexity theoretic problems, especially related to NP completeness, with an internalized concept of graph. To this end we report here on a case study on two NP complete problems, 3-satisfiability (3-SAT) and the chromatic number of a graph (CHROMATIC) and their encodings in the linear logical framework LLF [CP96].

When representing mathematical constructs, it is not only important how variables are encoded, but also what operations are provided, and what meta theoretical behavior can be expected. A standard technique for representing variables in logical frameworks with induction principles, for example, often employ de Bruijn indices or strings, and consequently environments, contexts, and substitutions are encoded as lists. In this setting, common operations, such as the creation of new variable names, lookup, and substitution application are left to the user, as is the derivation of necessary but often unwieldy meta-theoretic consequences. Other logical frameworks, that have evolved out of the simply typed λ-calculus, such as LF [HHP93] and LLF advocate the use of (linear) higher-order functions to encode variable binders to encode object language variables unless they behave differently. Closed under congruence, $\beta\eta$ [Coq91] are the only rules of equality supported and therefore neither of the two frameworks help directly with the encoding of graphs and their associated functionality.

In a first preliminary case study we have encoded the reduction from the satisfiability of conjunctive normal form (CNF) to the satisfiability problem of a conjunctive normal form where each clause consists of exactly three literals (3-SAT) in LF, which does not depend on graphs at all. One direction was proven with the meta theorem prover that is part of the Twelf system [PS99]. The other direction required some amount of manual interaction and eventually was also verified. The problem described in this paper of reducing from 3-SAT to CHROMATIC is much more challenging because it relies crucially on the interaction with graphs including the insertion of edges as well as the union of graphs. The reduction described below is well-known [GJ79] and turns Boolean formulas in conjunctive normal form into graphs by inserting two vertices for each variable (one that corresponds to the positive form of the literal, the other to its negation) and a vertex for each individual clause. Depending on the literals contained in each clause, new edges are inserted into the graph.

Upon completion the chromatic number for the resulting graph is at least as complex to find as a model for the original Boolean formula. Because the reduction is polynomially computable in the size of the formula , the NP completeness of CHROMATIC follows directly from 3-SAT being NP complete. The development presented in this paper gives a deductive account for all concepts involved and describes an implementation that has been type-checked in linear Twelf. Linear Twelf is a preliminary prototype implementation of a extension of Twelf to constructs present in LLF. Unfortunately, linear Twelf is lacking a termination and coverage checkers necessary to ensure that the present encoding really constitutes a proof. We have however convincingly and satisfactorily verified these conditions by hand. The source code of this development can be accessed from the our webpage `http://www.cs.yale.edu/~jds58/chromatic.elf`.

This paper is organized in the following way. In Section 2, we describe two problems, 3-SAT and CHROMATIC, respectively, and formulate them in form of a inference system. How to convert 3-SAT to CHROMATIC is described in Section 3 followed by Section 4 on how to encode the problems and the respective reduction in LLF. In Section 5 we give a correctness proof of the reduction, and show that it is indeed total. We assess results and conclude with future work in Section 6. Excerpts of the implementation in linear Twelf is given in the Appendix.

Boolean variables		u, u_n
Boolean formulas	$f, f_n ::=$	$\mathsf{pos}\ u \mid \mathsf{neg}\ u \mid \mathsf{new}\ u.f \mid f_m \wedge f_n \mid f_m \vee f_n$
Vertex variables		$v, w, x, \ldots, v_n, w_n, x_n, \ldots$
Edges	$e, e_n ::=$	$\mathsf{edge}\ v\ w$
Graphs	$G, G_n ::=$	$\# \mid \mathsf{newv}\ v.G \mid \mathsf{newe}\ e : (v_m, v_n).G \mid G_m \cup G_n$

Figure 1: Description of instances of Boolean formulas and graphs.

2 Two NP-Complete Problems

The two fundamental problems in NP that are studied in this paper are 3-satisfiability (3-SAT) and chromatic number (CHROMATIC). We proceed by presenting them in the familiar standard theoretical computer science discourse below followed by a reformulation as an inference system. For more information about these and other NP-complete problems, the reader is referred to Garey and Johnson [GJ79].

Definition 2.1 (3-SAT) *Given a set $U = \{u_1, u_2, \ldots, u_n\}$ of Boolean variables and a conjunctive normal form formula $f = c_1 \wedge c_2 \wedge \ldots c_m$ on the Boolean variables in U such that $c_i = l_{i1} \vee l_{i2} \vee l_{i3}, \forall i = 1, \ldots, m$ and $l_{i1}, l_{i2}, l_{i3} \in U \cup \overline{U}$ where $\overline{U} = \{\bar{u}_1, \bar{u}_2, \ldots, \bar{u}_m\}$. Is there a truth assignment to the Boolean variables such that every clause in f is satisfied?*

Definition 2.2 (CHROMATIC) *Given a graph $G = (V, E)$ where V is the set of vertices and E is the set of edges, and a positive integer C. Is G C-colorable, i.e., does there exist a function $\chi : V \to \{1, 2, \ldots, C\}$ such that $\chi(u) \neq \chi(v)$ whenever $\{u, v\} \in E$?*

3-SAT's domain is that of propositional formulas with the usual connectives whereas CHROMATIC's domain is that of graphs consisting of vertices and edges. Boolean variables are schematic and denoted with $u_1 \ldots u_n$. We omit the informal pictorial presentation of graphs. For the purpose of representation in a formal system, however, the standard mathematical discourse of describing Boolean formulas, variables, and graphs is too informal and too imprecise. Hence, a more formal way of representing these domains is in order, and this is what is described in Figure 1. Besides Boolean variables, edges and vertices are considered to be variables as well, giving rise to three respective binding constructs new, newv, and newe. Without loss of generality, we assume that the individual conjuncts in a Boolean formula do not contain references to new. Colors C can be thought of as integers.

Next, we capture the essence of 3-SAT and CHROMATIC, formally. Of course, each of the definition can simply be expressed as a first-order formula enriched with predicates that describe formulas and graphs. When using a logical framework, however, it is easier to capture the respective meaning in form of two inference systems that are depicted in Figures 2 and 3.

The statement that an instance of 3-SAT is a "Yes" instance, i.e. the Boolean formula F has a model, is written as $\eta \vdash F\ \mathsf{SAT}$ where environments η contain assignments for the free variables in F.

$$\text{Environments:} \quad \eta \quad ::= \quad \cdot \mid \eta, u \to \mathsf{true} \mid \eta, u \to \mathsf{false}$$

$$\frac{}{\eta, u \rightarrow \mathsf{true} \vdash (\mathsf{pos}\ u)\ \mathsf{SAT}}\ \mathit{satp} \qquad \frac{}{\eta, u \rightarrow \mathsf{false} \vdash (\mathsf{neg}\ u)\ \mathsf{SAT}}\ \mathit{satn}$$

$$\frac{\eta \vdash F_1\ \mathsf{SAT} \quad \eta \vdash F_2\ \mathsf{SAT}}{\eta \vdash (F_1 \wedge F_2)\ \mathsf{SAT}}\ \mathit{sat}\wedge$$

$$\frac{\eta \vdash F_1\ \mathsf{SAT}}{\eta \vdash (F_1 \vee F_2)\ \mathsf{SAT}}\ \mathit{sat}\vee 1 \qquad \frac{\eta \vdash F_2\ \mathsf{SAT}}{\eta \vdash (F_1 \vee F_2)\ \mathsf{SAT}}\ \mathit{sat}\vee 2$$

$$\frac{\eta, u \rightarrow \mathsf{true} \vdash F\ \mathsf{SAT}}{\eta \vdash \mathsf{new}\ u.F\ \mathsf{SAT}}\ \mathit{satt} \qquad \frac{\eta, u \rightarrow \mathsf{false} \vdash F\ \mathsf{SAT}}{\eta \vdash \mathsf{new}\ u.F\ \mathsf{SAT}}\ \mathit{satf}$$

Figure 2: Inference rules for "Yes" instances of 3-SAT.

$$\frac{}{\eta \vdash \#\ C\ \mathsf{COLORING}}\ \mathit{cgempty}$$

$$\frac{C' \leq C \quad \eta, v \rightarrow C' \vdash G\ C\ \mathsf{COLORING}}{\eta \vdash \mathsf{newv}\ v.G\ C\ \mathsf{COLORING}}\ \mathit{cgvertex}$$

$$\frac{C_1 \leq C \quad C_2 \leq C \quad C_1 \neq C_2 \quad \eta, A \rightarrow C_1, B \rightarrow C_2 \vdash G\ C\ \mathsf{COLORING}}{\eta, A \rightarrow C_1, B \rightarrow C_2 \vdash \mathsf{newe}\ e : (A, B).G\ C\ \mathsf{COLORING}}\ \mathit{cgedge}$$

$$\frac{\eta \vdash G_1\ C\ \mathsf{COLORING} \quad \eta \vdash G_2\ C\ \mathsf{COLORING}}{\eta \vdash (G_1 \cup G_2)\ C\ \mathsf{COLORING}}\ \mathit{cgunion}$$

Figure 3: Inference rules for "Yes" instances of CHROMATIC.

Environments also satisfy the standard properties of intuitionistic contexts, such as *weakening*, *strengthening* and *permutation*, allowing us to use higher-order encodings to represent them in a logical framework as discussed in Section 4.

Similarly, every "Yes" instance satisfying Definition 2.2 can be expressed as a derivation in the inference system depicted in Figure 3. For any graph G and color C, the judgment $\eta \vdash G\ C\ \mathsf{COLORING}$ means that the graph G can be colored with at most C colors where colors for free vertices in G are colored as defined in η and no edge is connected to vertices of same color. In this setting, we extend η to binds colors to vertices as well.

$$\text{Environments:} \quad \eta \quad ::= \quad \cdots \mid \eta, A \rightarrow C$$

It is not hard to see that an instance of 3-SAT or CHROMATIC will have a deduction if and only if it is a "Yes" instance. As an example, the proof that the Boolean formula $\bar{u}_1 \wedge u_2$ is satisfiable is written as

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\cdot, u_1 \longrightarrow \mathsf{false}, u_2 \longrightarrow \mathsf{true} \vdash (\mathsf{neg}\ u_1)\ \mathsf{SAT}}\,\mathit{satn} \qquad \dfrac{}{\cdot, u_1 \longrightarrow \mathsf{false}, u_2 \longrightarrow \mathsf{true} \vdash (\mathsf{pos}\ u_2)\ \mathsf{SAT}}\,\mathit{satp}}{\cdot, u_1 \longrightarrow \mathsf{false}, u_2 \longrightarrow \mathsf{true} \vdash (\mathsf{neg}\ u_1) \wedge (\mathsf{pos}\ u_2)\ \mathsf{SAT}}\,\mathit{sat}\wedge}{\cdot, u_1 \longrightarrow \mathsf{false} \vdash \mathsf{new}\ u_2.(\mathsf{neg}\ u_1) \wedge (\mathsf{pos}\ u_2)\ \mathsf{SAT}}\,\mathit{satt}}{\cdot \vdash \mathsf{new}\ u_1.\mathsf{new}\ u_2.(\mathsf{neg}\ u_1) \wedge (\mathsf{pos}\ u_2)\ \mathsf{SAT}}\,\mathit{satf}$$

However, it is important to note that the intractability of an NP-complete problem has a mirror image in the world of inference systems as well. A graph G and a color C forms an instance of Definition 2.2 if and only if a derivation of $\cdot \vdash G\ C$ $\mathsf{COLORING}$ exists, which may involve checking all possible color assignments for vertices in the instance.

3 3-SAT CHROMATIC Reduction

A *polynomial time reduction* from 3-SAT to CHROMATIC consists in showing that there exists a algorithm polynomial in the size of the Boolean formula that converts every instance of 3-SAT to an instance of CHROMATIC such that all "Yes" instances of 3-SAT are mapped to "Yes" instances of CHROMATIC and vice-versa. Instead of mapping a Boolean formula F to a graph G, we shall use the inference system formulation from the previous section to represent the *polynomial time reduction* as a total function mapping derivation of $\eta \vdash F$ SAT into derivations of $\eta \vdash G\ C$ $\mathsf{COLORING}$.

Following Karp [Kar72][1] and Lewis [Lew] we sketch the reduction first informally before formalizing it further. Suppose, we are given an instance of 3-SAT as described in Definition 2.1.

1. For every variable u_i, create vertices v_i, v'_i and x_i. For every clause c_j, create a vertex c_j in the graph.
2. Connect the edges between these vertices as below:
    (a) For every i, add an edge $\{v_i, v'_i\}$.
    (b) For every i and j, add an edge $\{x_i, x_j\}$ when $i \neq j$.
    (c) For every i and j, add an edge $\{v_i, x_j\}$ and $\{v'_i, x_j\}$ when $i \neq j$.
    (d) For every i and j, add an edge $\{c_i, v_j\}$ if u_j does not appear in c_i and an edge $\{c_i, v'_j\}$ if $\bar{u}_j$ does not appear in c_i.

It is not hard to see that if the Boolean formula with n variables has a truth assignment then the graph has a $n+1$-coloring and vice versa. Essentially, the construction given above – connecting v_i's and v'_i's to the clique on x_i's – forces creation of n true colors and one false color.

A formalization of this reduction, again in form of a inference system is given in Figure 4. The main judgment is of the form $\Gamma; \Delta \vdash K \diamond F \Rightarrow_C C', G$, where Γ is a list of assumptions

[1]Karp's reduction as printed is incorrect.

$$\frac{\Gamma,(u,v,v',x);\Delta,u\vdash K\diamond F\Rightarrow_{C+1}C',G}{\Gamma;\Delta\vdash K\diamond\mathsf{new}\ u.F\Rightarrow_C C',\mathsf{newv}\ v\ v'\ x.\mathsf{newe}\ e:(v,v').G}\ \mathit{convnew}$$

$$\frac{\Gamma;\Delta\vdash K;F\diamond F'\Rightarrow_C C',G}{\Gamma;\Delta\vdash K\diamond F\wedge F'\Rightarrow_C C',G}\ \mathit{conv}\wedge$$

$$\frac{\Gamma;\Delta\vdash K;(F_1\vee F_2\vee F_3)\Rightarrow G_1\quad \Gamma;\Delta\vdash G_2\ \mathsf{CLIQUE}\quad \Gamma;\Delta\vdash G_3\ \mathsf{VARS\text{-}TO\text{-}CLIQUE}}{\Gamma;\Delta\vdash K\diamond(F_1\vee F_2\vee F_3)\Rightarrow_C C,G_1\cup G_2\cup G_3}\ \mathit{convb}$$

$$\frac{}{\Gamma;\Delta\vdash\cdot\Rightarrow\#}\ \mathit{cconv\_base}\qquad\frac{\Gamma;\Delta\vdash K\Rightarrow G_1\quad\Gamma;\Delta\vdash F\Rightarrow G_2}{\Gamma;\Delta\vdash K;F\Rightarrow G_1\cup G_2}\ \mathit{cconv\_cont}$$

$$\frac{\Gamma,(u_1,v_1,v_1',x_1),(u_2,v_2,v_2',x_2),(u_3,v_3,v_3',x_3);\Delta\vdash c\downarrow G}{\begin{array}{c}\Gamma,(u_1,v_1,v_1',x_1),(u_2,v_2,v_2',x_2),(u_3,v_3,v_3',x_3);\Delta,u_1,u_2,u_3\vdash(\mathsf{pos}\ u_1)\vee(\mathsf{pos}\ u_2)\vee(\mathsf{pos}\ u_3)\\ \Rightarrow\mathsf{newv}\ c.\mathsf{newe}\ e_1:(c,v_1')\ e_2:(c,v_2')\ e_3:(c,v_3').G\end{array}}\ \mathit{conv5}$$

(39 SIMILAR RULES OMITTED)

$$\frac{}{\Gamma;\cdot\vdash C\downarrow\#}\ \mathit{conv\_base}\qquad\frac{\Gamma,(u,v,v',x);\Delta\vdash C\downarrow G}{\Gamma,(u,v,v',x);\Delta,u\vdash C\downarrow\mathsf{newe}\ e:(C,v)\ e':(C,v')}\ \mathit{conv\_cont}$$

$$\frac{}{\Gamma;\cdot\vdash\#\ \mathsf{CLIQUE}}\ \mathit{clique\_\#}\qquad\frac{\Gamma,(u,v,v',x);\Delta\vdash G_1\ \mathsf{CLIQUE}\quad\Gamma;\Delta\vdash\mathsf{CONNECTX}\ x\ G_2}{\Gamma,(u,v,v',x);\Delta,u\vdash(G_1\cup G_2)\ \mathsf{CLIQUE}}\ \mathit{clique\_vtx}$$

$$\frac{}{\Gamma;\cdot\vdash\#\ \mathsf{VARS\text{-}TO\text{-}CLIQUE}}\ \mathit{vars2clique\_\#}$$

$$\frac{\begin{array}{rr}\Gamma,(u,v,v',x);\Delta\vdash G_1\ \mathsf{VARS\text{-}TO\text{-}CLIQUE}&\Gamma,(u,v,v',x);\Delta\vdash\mathsf{CONNECTX}\ v\ G_2\\ &\Gamma,(u,v,v',x);\Delta\vdash\mathsf{CONNECTX}\ v'\ G_3\\ &\Gamma,(u,v,v',x);\Delta\vdash\mathsf{CONNECTV}\ x\ G_4\end{array}}{\Gamma,(u,v,v',x);\Delta,u\vdash(G_1\cup G_2\cup G_3\cup G_4)\ \mathsf{VARS\text{-}TO\text{-}CLIQUE}}\ \mathit{vars2clique\_vtx}$$

$$\frac{}{\Gamma;\cdot\vdash\mathsf{CONNECTV}\ X\ \#}\ \mathit{connectV\_\#}$$

$$\frac{\Gamma,(u,v,v',x');\Delta\vdash\mathsf{CONNECTV}\ X\ G}{\Gamma,(u,v,v',x');\Delta,u\vdash\mathsf{CONNECTV}\ X\ \mathsf{newe}\ e:(X,v)\ e':(X,v').G}\ \mathit{connectV\_vtx}$$

$$\frac{}{\Gamma;\cdot\vdash\mathsf{CONNECTX}\ X\ \#}\ \mathit{connectX\_\#}$$

$$\frac{\Gamma,(u,v,v',x');\Delta\vdash\mathsf{CONNECTX}\ X\ G}{\Gamma,(u,v,v',x');\Delta,u\vdash\mathsf{CONNECTX}\ X\ \mathsf{newe}\ e:(X,x').G}\ \mathit{connectX\_vtx}$$

Figure 4: Linear LF representation of 3-SAT CHROMATIC reduction.

of the form (u, v, v', x) representing a relationship between a free Boolean variable in F and its corresponding free graph vertices in G. Δ is a list of Boolean variables accumulated during the traversal of a Boolean formula when the reduction algorithm runs. Eventually, it will contain all free Boolean variables in F. We also maintain two variables C and C': C is incremented every time we see a new variable and C' corresponds to the total number of variables. Intuitively, the C contains the information which vertex to color with which color. All clauses that are contained in the Boolean formula prompt the insertion of edges into the graph corresponding to step (d) of the conversion algorithm. We achieve this by maintaining a "continuation" stack of clauses that were already encountered but not yet processed. The language of continuation stack is given below. Here init is the initial continuation, indicating that we have no more clauses left. Pfenning [Pfe01] describes continuations and their usage in compilation of expressions in considerable detail.

$$\text{Continuations} \quad K \quad ::= \quad \mathsf{init} \mid K; f$$

Thus, these inference rules allow us to build a valid deduction for a judgment $\Gamma; \Delta \vdash K \diamond F \Rightarrow_C C', G$ if and only if the conversion algorithm given above converts the Boolean formula represented by combining the clauses in F and K to the graph G; C should always be more than the free Boolean variables in F and C' is the total number of Boolean variables in F.

The edges in step (a) are added immediately when we encounter a new variable in rule *convnew*, the edges in step (b) are added through the inference rules associated with judgment $\Gamma; \Delta \vdash G$ CLIQUE, the edges in step (c) are added through the inference rules associated with $\Gamma; \Delta \vdash G$ $\mathsf{VARS\text{-}TO\text{-}CLIQUE}$.

In step (d), we create a vertex corresponding to every clause and add edges connecting the clause to vertices corresponding to literals not in the clause. These edges are added through the inference rules associated with $\Gamma\ \Delta \vdash K; F \Rightarrow G$. We are only considering clauses with three literals and hence there are 40 different kinds of clauses: each of the 3 literals can have a variable appearing as itself or as its complement, giving us 8 choices and each clause can have up to 3 distinct variables, giving us 5 choices[2]. For the sake of conciseness we give only one representative rule *conv5* in Figure 4, the reader may guess what the other 39 rules look like.

The predicate $\mathsf{CONNECTX}$ adds an edge between its first argument and every vertex among the resource in Δ. We note that once we access a vertex in Δ, it is subsequently removed from it (see for example rules *conv5*, *conv_cont*,*clique_vtx*, *vars2clique_vtx*,*connectV_vtx*, and *connectX_vtx*). We build the clique recursively. First, we access a vertex among all resources in Δ, thereby removing it from the context. We add an edge between this vertex and all other vertices in the context. Then we merge it with the clique created recursively from the remaining vertices in the context (see rule *clique_vtx*). Consequently, Δ's properties are best described as those of the linear context in the sense of linear logic [Gir87].

If a Boolean formula F has n variables, 0 free variables and m clauses, then the number of inference rules used in the derivation $\cdot; \cdot \vdash \mathsf{init} \diamond F \Rightarrow_{\mathsf{z}} C, G$ are $m + n + 1$ (each new variable corresponds to the inference rule *convnew*, each clause corresponds to the inference rule

[2]When variables appear only positively in each of the 3 literals, the 5 choices are: $(\mathsf{pos}\ u_1) \vee (\mathsf{pos}\ u_1) \vee (\mathsf{pos}\ u_1)$, $(\mathsf{pos}\ u_1) \vee (\mathsf{pos}\ u_1) \vee (\mathsf{pos}\ u_2)$, $(\mathsf{pos}\ u_1) \vee (\mathsf{pos}\ u_2) \vee (\mathsf{pos}\ u_1)$, $(\mathsf{pos}\ u_2) \vee (\mathsf{pos}\ u_1) \vee (\mathsf{pos}\ u_1)$, $(\mathsf{pos}\ u_1) \vee (\mathsf{pos}\ u_2) \vee (\mathsf{pos}\ u_3)$

conv$\wedge$ and there is one base case). Further, the deductions for the judgments $\Gamma; \Delta \vdash K \Rightarrow G_1$, $\Gamma; \Delta \vdash G_2$ CLIQUE, and $\Gamma; \Delta \vdash G_3$ $\mathsf{VARS\text{-}TO\text{-}CLIQUE}$ have height $O(n)$. Hence, the total number of inference rules used in the derivation of the reduction is $O(m + n)$. Thus, the proposed reduction algorithm is in P, moreover, the way how the reduction algorithm is specified renders it directly amenable to being implemented in a logical framework which we discuss next.

4 Representation in a Logical Framework

A logical framework is a meta-language that serves the representation of deductive system defined in terms of judgments and inference rules in a type theory. Several frameworks are available, we mention only few such as Isabelle [Pau94], Lego [LP92], LF [HHP93] or LLF [CP96]. For a good overview about logical framework research, consult [Pfe99]. In pursuit of the overall goal of this work, the design of a digital library for complexity theoretic problems, special attention must be paid to the simplicity with which complexity theoretic problems are to be formulated by the user. Therefore, for this particular case study, our choice has fallen on LLF, mostly because the representation of Γ and Δ behave just like the intuitionistic and the linear context provided by the framework. And indeed, the entire the development so far through Sections 2 and 3 has been implemented and type checked in a prototype implementation for LLF and the interested reader can get a feel for the encoding in Appendix A – C. Also the correctness proof still to be discussed has been implemented in linear Twelf.

LLF is a conservative extension over LF and incorporates three connectives from linear logic, namely *multiplicative implication* ($\multimap$), *additive conjunction* (&) and *additive truth* ($\top$). It is a two zone system that explicitly distinguishes between *intuitionistic* assumptions (that play the role of Γ), and *linear* assumptions (that play the role of resources Δ). In a derivation, linear assumptions can only be used exactly once. The few main axiom and introduction rules of linear logic connectives and their intuitionistic counterparts are given in Figure 5. Note that the *additive conjunction* (&) allows the use of the same set of linear assumptions for proving both the conjuncts and *multiplicative implication* ($\multimap$) puts a linear assumption into the linear context. The rule *Iaxiom* expresses that an intuitionistic assumption can only be used if no linear assumptions are present as opposed to the *Laxiom* rule that consumes one single linear assumption.

LLF supports the *judgments-as-types* methodology for representation and incorporates the aforementioned linear connectives as type constructors. In addition, each rule is endowed with a proof objects that correspond to the introduction and elimination forms as shown below.

$$\begin{array}{lll} \textit{Objects } M \quad ::== & \hat{\lambda} x : A.M \mid M_1 \hat{\ } M_2 & \text{(Linear functions)} \\ & \mid \langle M_1, M_2 \rangle \mid \mathsf{FST}\ M \mid \mathsf{SND}\ M & \text{(Additive conjunction)} \\ & \mid \langle\rangle & \text{(Additive unit element)} \end{array}$$

Thus, a linear LF representation for the judgment $\Gamma; \Delta \vdash J$ is $\ulcorner\Gamma\urcorner \to \ulcorner\Delta\urcorner \multimap \ulcorner J\urcorner$ where $\ulcorner\Gamma\urcorner$, $\ulcorner\Delta\urcorner$ and $\ulcorner J\urcorner$ are linear LF representations of Γ, Δ and J respectively. And

$$\frac{}{\Gamma, A; \cdot \vdash A}\ Iaxiom \qquad \frac{}{\Gamma; \Delta, A \vdash A}\ Laxiom$$

$$\frac{}{\Gamma; \Delta \vdash \top}\ Top$$

$$\frac{\Gamma; \Delta_1 \vdash A \quad \Gamma; \Delta_2 \vdash B}{\Gamma; \Delta_1, \Delta_2 \vdash A \times B}\ \times - I \qquad \frac{\Gamma; \Delta \vdash A \quad \Gamma; \Delta \vdash B}{\Gamma; \Delta \vdash A \& B}\ \& - I$$

$$\frac{\Gamma, A; \Delta \vdash B}{\Gamma; \Delta \vdash A \rightarrow B}\ \rightarrow -I \qquad \frac{\Gamma; \Delta, A \vdash B}{\Gamma; \Delta \vdash A \multimap B}\ \multimap -I$$

Figure 5: A fragment of linear logic.

finding a proof is equivalent to finding an object of the corresponding type generated from the language of LF objects augmented as above.

Although LLF supports quite elegant representations of Boolean formulas and graphs given in Figure 1, there is something unsatisfying about the way the reduction is laid out in Figure 4. The linear context is used as an auxiliary concept for computing cliques, and to connect a vertex to all remaining vertices in a graph. Additive conjunction is used to pass this auxiliary information to the other relevant judgments. The graph isomorphism problem is not relevant for this particular case study, but might be in the general case, as are other operations such as the intersection of two graphs, or the expansion of a node in graph by another graph. To decide it in LLF would require the user to encode it explicitly — a complicated operation that is prone for error and difficult to reason about. Our graph representation stores all vertices and edges in a graph together with the superfluous history in which order vertices and edges were inserted. Thus, adequacy of the representation is guaranteed.

The linear Twelf code for all inference rules shown so far can be directly derived from the *judgments-as-types* methodology underlying LLF. The translations where the environment mapping is extended such as the rules *satt* and *satf* for 3-SAT and *cgvertex* and *cgedge* for CHROMATIC use hypothetical judgments. These assignments are represented by type families `hyp` and `colorvertex` for the case of Boolean variables and graph vertices respectively. We give some of these rules here in Figure 6 the complete code is given in the Appendices A and B. Let U and V be LLF types, and M an LLF object. In concrete syntax, we write `{x:`U`}`V for the dependent type $\Pi x : U.V$, and `[x:`U`]`M for the higher-order term $\lambda x : U.M$. As usual, where convenient we omit the leading block of Π quantifiers from constant declarations to improve readability. Furthermore we write U `-o` V for the linear function type $U \multimap V$, and U `&` V for the additive conjunction $U \& V$. Figure 7 depicts an encoding of the rules that construct a graph clique. Note, how the harmless looking `&` in `clique_vtx` is responsible for duplicating the context Δ in rule *clique_vtx* in Figure 4.

```
hyp  : v -> b -> type.          colorvertex : vertex -> nat -> type.
sat  : o -> type.               coloring  : nat -> graph -> type.
satp  : sat (pos A)             cg# : coloring C #.
         <- hyp A true.         cgvertex : coloring C (newv [v] (G v))
sat/\ : sat (F1 /\ F2)                     <- (C' <= C)
         <- sat F1                         <- (v:vertex colorvertex v C'
         <- sat F2.                            -> coloring C (G v)).
satnewt : sat (new F)
          <- (v:v hyp v true -> sat (F v)).
```

Figure 6: Encoding of selected rules in LLF.

```
clique : graph -> type.
connectX : vertex -> graph -> type.
clique_vtx : (var U -o clique (G + G'))
                    <- clique G & connectX X G'
                    <- relate U _ _ X.
clique_#  : clique #.
connectX_vertex : (var U -o connectX X (newe [e:edge X X'] G))
                   <- relate U _ _ X'
                   <- connectX X G.
connectX_#      : connectX X #.
```

Figure 7: Encoding of cliques in LLF.

5 Correctness

In this section, we show that the conversion from 3-SAT to CHROMATIC maps all "Yes" instances of 3-SAT to "Yes" instances of CHROMATIC and vice-versa. This section presents a sequence of lemmas cumulating in Theorem 5.7. All lemmas and theorems presented in this Section have been encoded as relations in LLF and type checked in the prototype implementation of linear Twelf. The source code for this development is given in Appendix D. Although type checking is not enough to guarantee that the source code constitutes a proof, we have convinced ourselves that all cases are covered and the underlying induction principle is sound.

With the deductive description of the reduction algorithm from Figure 4, we first establish the invariant property about the assumptions in Γ with respect to the context η as used in Figure 2 and 3. First, it relates the truth value of each Boolean variable in 3-SAT with the color assignment to each vertex in CHROMATIC and second it guarantees that all vertices in the context are assigned distinct colors. That the intuitionistic LLF context as image of Γ satisfies the second part at all times is a global property of the context and can currently only be enforced by manual inspection.

Definition 5.1 (Valid environments) *For any Boolean variables* $u_1, u_2, \ldots, u_n$ *and vertices* $v_1, v_2, \ldots, v_n; v'_1, v'_2, \ldots, v'_n$ *and* $x_1, x_2, \ldots, x_n$*, let*

$$\Gamma = (u_1, v_1, v'_1, x_1), (u_2, v_2, v'_2, x_2), \ldots, (u_n, v_n, v'_n, x_n)$$

and $\eta = u_1 \to B_1, u_2 \to B_2, \ldots, u_n \to B_n$ *where* $B_i \in \{\mathsf{true}, \mathsf{false}\}$ *and* C *be any color. An environment* η' *is said to be valid under environments* Γ *and* η *and color* C *if*

$$\eta' = \begin{cases} x_1 \to C_1, x_2 \to C_2, \ldots, x_n \to C_n \\ v_1 \to C'_1, v_2 \to C'_2, \ldots, v_n \to C'_n \\ v'_1 \to C''_1, v_2 \to C''_2, \ldots, v'_n \to C''_n \end{cases}$$

C_i*'s are distinct,* $C_i \leq C$ *and*

$$C'_i = \begin{cases} C_i & \textit{if } B_i = \mathsf{true} \\ \mathsf{c}_0 & \textit{if } B_i = \mathsf{false} \end{cases} \quad C''_i = \begin{cases} \mathsf{c}_0 & \textit{if } B_i = \mathsf{true} \\ C_i & \textit{if } B_i = \mathsf{false} \end{cases}$$

We begin now with the presentation of the individual lemmas. First, we describe an infrastructure lemma about properties of colors. If integers were available as constraint domain in linear Twelf as they are in the non-linear version of Twelf (which they are not), colors could have been encoded directly as integers rendering this lemma unnecessary.

Lemma 5.2 (Properties of colors) *Let* C *and* C' *be colors. The following properties hold: (1) If* $C < C'$ *then* $C < C'+1$*. (2) If* $C' \leq C$ *then* $C' \leq C+1$*. (3) If* $C = C'$ *then* $C < C'+1$*. (4) If* $C \leq C$*. (5) If* $C < C+1$*. (6) If* $C < C'$ *then* $C \neq C'$*. (7) If* $C < C'$ *then* $C \leq C'$*. (8) If* $C < C'$ *then* $C < C'+1$*.*

The proofs are straightforward, and omitted (even from the appendix). We refer, however, to these properties in the appendix in form of type families `lemma1` – `lemma8`. Next, we prove the basic property of graph coloring that is a graph that is colorable with C colors is also colorable with $C+1$ colors.

Theorem 5.3 (Coloring preservation) *For any graph* G *and a color* C*, if* $\mathcal{D} :: \eta \vdash G\ C\ \mathsf{COLORING}$ *then there exists* $\mathcal{D}' :: \eta \vdash G\ (C+1)\ \mathsf{COLORING}$*.*

Proof: The proof proceeds by induction over the height of derivation $\mathcal{D}$. We have four cases depending on whether the derivation $\mathcal{D}$ ends in *cg#*, *cgvertex*, *cgedge* or *cgunion* (see figure 3); the case *cg#* is the base case. The proof follows naturally using Lemma 5.2 (2). □

In LLF, this proof is represented as a relation over $\mathcal{D}$ and $\mathcal{D}'$, i.e. $\ulcorner\mathcal{D}\urcorner$ `:coloring C G` and $\ulcorner\mathcal{D}'\urcorner$ `:coloring (s C) G`. This relation is implemented as a type family

```
increase_color : coloring C G -> coloring (s C) G -> type.
```

where each of the cases of the proof is represented as a declaration as given in Appendix C. We show some excerpts from the signature below. The declarations of `increase_vtx` and `increase_union` correspond to the cases when the derivation $\mathcal{D}$ ends in *cgvertex* and *cgunion* respectively.

```
increase_vtx : increase_color (cgvertex CG E) (cgvertex CG' E')
                <- ({v:vertex}{c:colorvertex v C'}
                      increase_color (CG v c) (CG' v c))
                <- lemma2 E E'.

increase_union : increase_color (cgunion CG1 CG2) (cgunion CG1' CG2')
                  <- increase_color CG1 CG1'
                  <- increase_color CG2 CG2'.
```

The lemmas given below prove that the edges added in the steps (b), (c) and (d) of the conversion algorithm do not connect vertices assigned the same color.

Lemma 5.4 (Clique Coloring) *For any graph G: Let* $x_1, x_2, \ldots, x_n$ *be the free variables in* G *and* $u_1, u_2, \ldots, u_n$ *be Boolean variables. Let* $\Gamma = (u_1, \_, \_, x_1), (u_2, \_, \_, x_2), \ldots, (u_n, \_, \_, x_n)$ *and* $\Delta = u_1, u_2, \ldots, u_n$.

If $\mathcal{D} :: \Gamma; \Delta \vdash G$ CLIQUE *and* $C \geq \mathsf{c}_0 + n$ *then there exists* $\mathcal{G} :: \eta \vdash G\ C$ $\mathsf{COLORING}$ *where* $\eta = x_1 \to C_1, x_2 \to C_2, \ldots, x_n \to C_n$*,* C_i*'s are distinct and* $C_i \leq C$*.*

Proof: Since, the inference rules for CLIQUE only connect the different x_i's and these vertices are colored with distinct colors, the proof follows by induction. An encoding of the individual cases of type family `clique_color` is given in Appendix D. □

Lemma 5.5 (Coloring of graph created from edges between variables and clique) *For any graph G: Let* $v_1, v_2, \ldots, v_n; v'_1, v'_2, \ldots, v'_n$ *and* $x_1, x_2, \ldots, x_n$ *be the free variables in* G *and* $u_1, u_2, \ldots, u_n$ *be Boolean variables. Let*

$$\Gamma = (u_1, v_1, v'_1, x_1), (u_2, v_2, v'_2, x_2), \ldots, (u_n, v_n, v'_n, x_n)$$

and $\Delta = u_1, u_2, \ldots, u_n$*. Let* $\eta = u_1 \to B_1, u_2 \to B_2, \ldots, u_n \to B_n$ *where* $B_i \in \{\mathsf{true}, \mathsf{false}\}$*. If* $\mathcal{D} :: \Gamma; \Delta \vdash G$ $\mathsf{VARS\text{-}TO\text{-}CLIQUE}$ *and* $C \geq \mathsf{c}_0 + n$ *then there exists* $\mathcal{G} :: \eta' \vdash G\ C$ $\mathsf{COLORING}$ *where* η' *is valid under environments* Γ*,* η *and color* C*.*

Proof: The inference rules for $\mathsf{VARS\text{-}TO\text{-}CLIQUE}$ only connect v_i to x_j and v'_i to x_j if $i \neq j$. These pairs of vertices are never assigned same colors if η' is a valid environment. The proof follows again by induction. An encoding of the individual cases of type family `connect2clique` is given in Appendix D. □

Lemma 5.6 (Coloring of graph created from the clauses) *For any continuation K and a graph G: Let* $u_1, u_2, \ldots, u_n$ *be the free variables in* K *and* $v_1, v_2, \ldots, v_n; v'_1, v'_2, \ldots, v'_n$ *be the free variables in* G*. Let*

$$\Gamma = (u_1, v_1, v'_1, \_), (u_2, v_2, v'_2, \_), \ldots, (u_n, v_n, v'_n, \_)$$

and $\Delta = u_1, u_2, \ldots, u_n$*. Let* $\eta = u_1 \to B_1, u_2 \to B_2, \ldots, u_n \to B_n$ *where* $B_i \in \{\mathsf{true}, \mathsf{false}\}$*. If* $\mathcal{D} :: \Gamma; \Delta \vdash K \Rightarrow G$ *and* $C \geq \mathsf{c}_0 + n$ *then*

$$\mathcal{E} :: \eta \vdash K\ SAT$$

iff

$$\mathcal{G} :: \eta' \vdash G\ C\ \textsf{COLORING}$$

where η' is valid under environments Γ, η and color C.

Proof: If the truth assignment in η satisfies all the clauses in K, then we color every clause vertex c_j with the color assigned to the literal which satisfies that clause, i.e. if the literal u_i appears in c_j and is $\textsf{true}$ then we assign c_j with color of v_i (as defined in η') and if the literal $\bar{u}_i$ appears in c_i and is $\textsf{false}$ then we assign c_j with color of v'_i (as defined in η'). In other words, we color each c_j with a $\textsf{true}$ color. Since, we have n distinct $\textsf{true}$ colors and c_j is connected to vertices corresponding to literals not in c_j, we do not add any edges between vertices with same color. The other direction is similar. The formal proof is by induction and each of the cases of the *only if* part is encoded into declarations of the type family `conv'_color` in Appendix D. □

Theorem 5.7 (Main Theorem) *For any continuation K, Boolean formula F and graph G: Let $u_1, u_2, \ldots, u_n$ be the free variables in F and K. Let $v_1, v_2, \ldots, v_n$;$v'_1, v'_2, \ldots, v'_n$ and $x_1, x_2, \ldots, x_n$ be the free variables in G. Let*

$$\Gamma = (u_1, v_1, v'_1, x_1), (u_2, v_2, v'_2, x_2), \ldots, (u_n, v_n, v'_n, x_n)$$

and $\Delta = u_1, u_2, \ldots, u_n$. Let $\eta = u_1 \to B_1, u_2 \to B_2, \ldots, u_n \to B_n$ where $B_i \in \{\textsf{true}, \textsf{false}\}$. If $\mathcal{D} :: \Gamma; \Delta \vdash K \diamond F \Rightarrow_C C', G$ and $C \geq \textsf{c}_0 + n$ then

$$\mathcal{E} :: \eta \vdash F\ \textsf{SAT}, \mathcal{F} :: \eta \vdash K\ \textsf{SAT}$$

iff

$$\mathcal{G} :: \eta' \vdash G\ C'\ \textsf{COLORING}$$

where η' is valid under environments Γ and η and color C'.

Proof: (**Sketch**) We note that C' is at least n, so we always have n distinct colors. The proof follows by induction on the height of the derivation $\mathcal{D}$ using lemmas 5.4, 5.5 and 5.6. The *only if* part of the proof is represented in linear Twelf by a type family

```
reductionf : {C:nat} conv F C C' K G -> sat F -> satK K
             -> coloring C' G -> type.
```

It encodes the relation between color C, representations of the derivations $\mathcal{D}$, $\mathcal{E}$, $\mathcal{F}$ and $\mathcal{G}$, i.e. $\ulcorner\mathcal{D}\urcorner$ `: conv F C C' K G`, $\ulcorner\mathcal{E}\urcorner$ `: sat F`, $\ulcorner\mathcal{F}\urcorner$ `: satK K`, $\ulcorner\mathcal{G}\urcorner$ `: coloring C G`. The implementation of each case is direct and the corresponding encoding is given in Appendix D. Because each Γ is valid "by construction", it need not be included into the formulation of the theorem, and we can always take advantage of it whenever needed in the proof; the relevant type families are `necolor` and `lecolor` shown in the Appendix D. □

6 Conclusion and Future Work

In this paper, we have given a deductive account for a reduction from 3-satisfiability to chromatic number. All inference systems and all proofs presented in this paper have been implemented in the linear logical framework LLF. We believe this to be the first attempt to employ logical framework technology for representing and reasoning with NP-complete problems. Eventually we plan to develop a digital library of NP-complete problems, that contains problems domains, reductions, and correctness proofs, and a powerful query language to retrieve them.

Central to the representation of a whole range of NP-complete and many other problems in computer science is the notion of a graph. Graphs are not first class citizens neither in logical frameworks nor in computer algebra systems. On the other hand they are so common in this particular problem domain that a search for a graph oriented logical framework is warranted. This case study has exposed some of short-comings of our way of encoding graphs. First, variables that range over graphs behave differently than variables of the λ-calculus. What does it mean for a graph to be substituted into a variable that ranges over graphs? How shall incoming edges be relinked if a subgraph is replaced by a variable? Second, linearity plays an essential role in our construction of cliques, which makes graph related operations somewhat clumsy to express. Other operations, such as deciding if two graphs are isomorphic are neither primitive nor easy to implement, if definitional equality is based on $\beta\eta$-rules exclusively. In this light, the work described in this paper should not be seen as an answer, more as an initial step towards understanding the basic design requirements of such a graph oriented logical framework.

The nature of this case study exposes a challenge for coverage checking algorithms [SP03] predominantly used in logical frameworks as well. For example, an environment was defined to be valid only if all colors assigned to vertices are pairwise different. This is a global property that cannot easily be enforced locally as required in the proof of main theorem. Currently, invariants of this kind are not supported neither by the coverage checker implemented in Twelf nor the meta-logics for LF [Sch00] or LLF [MS03]. Consequently, further research and implementation work are necessary to provide coverage checking and automated theorem proving support for all lemmas and theorems contained in this paper.

And finally, any valid reduction between two NP complete problems must be a polynomial time reduction. It is possible to argue that this is indeed the case by examining the size of the derivations involved. What is left to do is to develop polytime checker for LF and LLF signatures, respectively.

Acknowledgements. We would like to thank Dana Angluin for raising our interest in studying the relationship between logical frameworks and complexity theory and the many discussions related to the subject.

References

[Coq91] Thierry Coquand. An algorithm for testing conversion in type theory. In Gérard Huet and Gordon Plotkin, editors, *Logical Frameworks*, pages 255–279. Cambridge

University Press, 1991.

[CP96] Iliano Cervesato and Frank Pfenning. A linear logical framework. In E. Clarke, editor, *Proceedings of the Eleventh Annual Symposium on Logic in Computer Science — LICS'96*, pages 264–275, New Brunswick, New Jersey, 27–30 July 1996. IEEE Computer Society Press.

[Gir87] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman and Co., 1979.

[HHP93] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, January 1993.

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of computer computations*, pages 85–103. Plenum Press, New York, NY, 1972.

[Lew] F. D. Lewis. *Solving NP-Complete Problems*. World Wide Web, http://cs.engr.uky.edu/ lewis/cs-heuristic/text/contents.html. This is a web-version of a work in progress.

[LP92] Zhaohui Luo and Robert Pollack. The LEGO proof development system: A user's manual. Technical Report ECS-LFCS-92-211, University of Edinburgh, May 1992.

[MS03] Andrew McCreight and Carsten Schürmann. A meta linear logical framework. Draft, 2003.

[Pau94] Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover*. Springer-Verlag LNCS 828, 1994.

[Pfe99] Frank Pfenning. Logical frameworks. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*. Elsevier Science Publishers, 1999. In preparation.

[Pfe01] Frank Pfenning. *Computation and Deduction*. Cambridge University Press, 2001.

[PS99] Frank Pfenning and Carsten Schürmann. System description: Twelf — a meta-logical framework for deductive systems. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction (CADE-16)*, pages 202–206, Trento, Italy, July 1999. Springer-Verlag LNAI 1632.

[Sch00] Carsten Schürmann. *Automating the Meta-Theory of Deductive Systems*. PhD thesis, Carnegie Mellon University, 2000. CMU-CS-00-146.

[SP03] Carsten Schürmann and Frank Pfenning. A coverage algorithm for LF. In *Proceedings of Theorem Proving in Higher-order Logics (TPHOLs)*, Rome, 2003. To appear.

A Encoding of 3-SAT

```
% Boolean variables
v : type.                               % Variables
b : type.                               % Boolean Domain
true : b.
false : b.

% Boolean Formulae
o : type.
/\ : o -> o -> o.                       %infix right 10 /\.
\/ : o -> o -> o.                       %infix right 10 \/.
pos : v -> o.
neg : v -> o.
new : (v -> o) -> o.

% Encoding 'Yes' instances of 3-SAT
hyp : v -> b -> type.
sat : o -> type.
satp : sat (pos A)
         <- hyp A true.
satn : sat (neg A)
         <- hyp A false.
sat/\: sat (F1 /\ F2)
        <- sat F1
        <- sat F2.
sat\/1: sat (F1 \/ F2)
         <- sat F1.
sat\/2: sat (F1 \/ F2)
         <- sat F2.
satnewt: sat (new F)
          <- ({v:v} hyp v true -> sat (F v)).
satnewf: sat (new F)
          <- ({v:v} hyp v false -> sat (F v)).
```

B Encoding of CHROMATIC

```
% Representation of graphs
vertex : type.
edge   : vertex -> vertex -> type.
graph  : type.
# : graph.                                         % Empty Graph
newv  : (vertex -> graph) -> graph.
newe  : (edge A B -> graph) -> graph.
+ : graph -> graph -> graph.                         %infix right 10 +.

% Colors -- Proofs Omitted
nat : type.
z : nat.
s : nat -> nat.
!= : nat -> nat -> type.                         %infix right 10 !=.
< : nat -> nat -> type.                          %infix right 10 <.
```

```
== : nat -> nat -> type.                         %infix right 10 ==.
<= : nat -> nat -> type.                         %infix left 10 <=.
% Some theorems about colors
lemma1 : C < C' -> C < (s C') -> type.
lemma2 : C <= C' -> C <= (s C') -> type.
lemma3 : C == C' ->  C < (s C') -> type.
lemma4 : C <= C -> type.
lemma5 : C < (s C) -> type.
lemma6 : C < C' -> C != C' -> type.
lemma7 : C < C' -> C <= C' -> type.
lemma8 : C < C' -> C < (s C') -> type.

% Encoding 'Yes' instances of CHROMATIC
colorvertex : vertex -> nat  -> type.
coloring : nat -> graph -> type.
cg# : coloring N #.
cgvertex : coloring C (newv [v] (G v))
           <- (C' <= C)
           <- ({v:vertex} colorvertex v C' -> coloring C (G v)).
cgedge : coloring C (newe [e: edge A B] G)
          <- colorvertex A C1
          <- colorvertex B C2
          <- C1 != C2
          <- C1 <= C
          <- C2 <= C
          <- coloring C G.
cgunion : coloring C (G1 + G2)
           <- coloring C G1
           <- coloring C G2.

% A property of graph coloring
increase_color : coloring C G -> coloring (s C) G -> type.
increase_# : increase_color cg# cg#.
increase_vtx : increase_color (cgvertex CG E) (cgvertex CG' E')
                <- ({v:vertex}{c:colorvertex v C'} increase_color (CG v c) (CG' v c))
                <- lemma2 E E'.
increase_edge : increase_color (cgedge CG E2 E1 E3 C1 C2) (cgedge CG' E2' E1' E3 C1 C2)
                 <- increase_color CG CG'
                 <- lemma2 E2 E2'
                 <- lemma2 E1 E1'.
increase_union : increase_color (cgunion CG1 CG2) (cgunion CG1' CG2')
                  <- increase_color CG1 CG1'
                  <- increase_color CG2 CG2'.
```

C Encoding of 3-SAT CHROMATIC Reduction

```
% Continuations
stack : type.
empty : stack.
; : stack -> o -> stack.                         %infix right 10 ;.

% Satisfiability for continuations
```

```
satK : stack -> type.
satKempty : satK empty.
satK; : satK (K ; F)
         <- satK K
         <- sat F.

% Encoding of 3-SAT CHROMATIC reduction
var : v -> type.
relate : v -> vertex -> vertex -> vertex -> type.
conv : o -> nat -> nat -> stack -> graph -> type.
conv' : stack -> graph -> type.
conv'' : o -> graph -> type.
conv'''  : vertex -> graph -> type.
clique : graph -> type.
connectX : vertex -> graph -> type.
connectV : vertex -> graph -> type.
connect2clique : graph -> type.
convnew: conv (new F) C C' K (newv [v] newv [v'] newv [x] newe [e:edge v v'] (G v v' x))
          <- ({u:v} {v:vertex} {v':vertex} {x:vertex} relate u v v' x -> var u
              -o conv (F u) (s C) C' K (G v v' x)).
conv/\: conv (F /\ F') C C' K G
        <- conv F' C C' (K ; F) G.
conv3: conv (F1 \/ F2 \/ F3) C C K (G + G' + G'')
       <- conv' (K ; (F1 \/ F2 \/ F3)) G & clique G' & connect2clique G''.

% Adding edges between clauses and all literals not in the clause
conv'init: conv' empty # o- <T>.                   % Consume rest of the linear context
conv'cont: conv' (K ; F) (G' + G'')
            <- conv' K G' & conv'' F G''.

conv''1: (var U -o conv'' ((pos U) \/ (pos U) \/ (pos U))
                   (newv [c] newe [e:edge c V'] (G c)))
          <- ({c:vertex} conv''' c (G c))
          <- relate U V V' _.
conv''2: (var U1 -o var U2 -o conv'' ((pos U1) \/ (pos U1) \/ (pos U2))
                           (newv [c] newe [e1:edge c V1'] newe [e2:edge c V2'] (G c)))
          <- ({c:vertex} conv''' c (G c))
          <- relate U1 V1 V1' _
          <- relate U2 V2 V2' _.
conv''3: (var U1 -o var U2 -o conv'' ((pos U1) \/ (pos U2) \/ (pos U1))
                           (newv [c] newe [e1:edge c V1'] newe [e2:edge c V2'] (G c)))
          <- ({c:vertex} conv''' c (G c))
          <- relate U1 V1 V1' _
          <- relate U2 V2 V2' _.
conv''4: (var U1 -o var U2 -o conv'' ((pos U2) \/ (pos U1) \/ (pos U1))
                           (newv [c] newe [e1:edge c V1'] newe [e2:edge c V2'] (G c)))
          <- ({c:vertex} conv''' c (G c))
          <- relate U1 V1 V1' _
          <- relate U2 V2 V2' _.
conv''5: (var U1 -o var U2 -o var U3 -o conv'' ((pos U1) \/ (pos U2) \/ (pos U3))
                           (newv [c] newe [e1:edge c V1'] newe [e2:edge c V2']
                                  newe [e3: edge c V3'] (G c)))
          <- ({c:vertex} conv''' c (G c))
          <- relate U1 V1 V1' _
          <- relate U2 V2 V2' _
          <- relate U3 V3 V3' _.
% There are 35 more similar inference rules: Each 'pos' can be replaced by
% 'neg' independently of others

conv'''_vtx: (var U -o conv''' C (newe [e: edge C V] newe [e': edge C V'] G))
                 <- (conv''' C G)
                 <- relate U V V' _.
conv'''_#: conv''' C #.


% Encoding of clique construction
clique_vtx : (var U -o clique (G + G'))
                    <- clique G & connectX X G'
                    <- relate U _ _ X.
clique_#  : clique #.

% Adds edges between the vertex X and all x-vertices in the context
connectX_vertex : (var U -o connectX X (newe [e:edge X X'] G))
                   <- relate U _ _ X'
                   <- connectX X G.
connectX_#      : connectX X #.

% Connect vertices corresponding to the variables to the clique
connect2clique_vtx : (var U -o connect2clique (G + G' + G'' + G'''))
                     <- connect2clique G & connectX V G' & connectX V' G'' &
                        connectV X G'''
                     <- relate U V V' X.
connect2clique_#   : connect2clique #.

%
connectV_vertex : (var U -o connectV X (newe [e:edge V X] newe [e':edge V' X] G))
                    <- relate U V V' _
                    <- connectV X G.
connectV_#      : connectV X #.

% A Property of 3-SAT CHROMATIC Reduction
conv_lemma : conv F C C' K G -> (C <= C') -> type.
convlem_base : conv_lemma (conv3 D) E
                <- lemma4 E.
convlem_/\ : conv_lemma (conv/\ D) E
                <- conv_lemma D E.
convlem_new : conv_lemma (convnew D) E
               <- ({u:v}{v:vertex}{v':vertex}{x:vertex}{r:relate u v v' x}{var: var u}
                     conv_lemma (D u v v' x r ^ var) (<=s E'))
               <- lemma2 E' E.                   % Proof of: (s C) <= C' ==> C <= C'
```

D Correctness of Encoding of 3-SAT CHROMATIC Reduction

```
% Correctness of CONNECTX: CONNECTX does not add edges between vertices
% having same color
connectX_color : {C:nat} colorvertex X C' -> (C < C') -> connectX X G
                                 -> coloring C' G -> type.
connectXc_base:  connectX_color z CGX E connectX_# cg#.
connectXc_cont:  {R:relate _ _ _ X1}{CGX1:colorvertex X1 (s C)}
                 connectX_color (s C) CGX (<s E) (connectX_vertex D R ^ V)
                                             (cgedge CG F2 F1 NE CGX1 CGX)
                 <- connectX_color C CGX E' D CG
                 <- lemma8 E E'
                 <- lemma4 F1
                 <- lemma6 (<s E) NE
                 <- lemma7 (<s E) F2.
connectX_color' : {C:nat} colorvertex X z -> connectX X G -> coloring (s C) G -> type.
connectXc'_base: connectX_color' z CGX connectX_# cg#.
connectXc'_cont: {R:relate _ _ _ X1}{CGX1:colorvertex X1 (s C)}
                 connectX_color' (s C) CGX (connectX_vertex D R ^ V)
                                         (cgedge CG E' <=z !=z1 CGX1 CGX)
                 <- connectX_color' C CGX D CG'
                 <- increase_color CG' CG
                 <- lemma5 E
                 <- lemma7 E E'.

% Correctness of encoding of Clique
clique_color : {C:nat} clique G -> coloring C G -> type.
cliquec_base : clique_color z clique_# cg#.
cliquec_cont : {R:relate _ _ _ X}{CGX:colorvertex X (s C)}
                clique_color (s C) (clique_vtx R (D1 , D2) ^ V) (cgunion CG1 CG2)
                <- clique_color C D1 CG1'
                <- increase_color CG1' CG1
                <- lemma5 E                      % Proof of C < (s C)
                <- connectX_color C CGX E D2 CG2.

% Correctness of CONNECTV
connectV_color : {C:nat} colorvertex X C' -> (C < C') -> connectV X G
                                                  -> coloring C' G -> type.
connectVc_base: connectV_color z CGX E connectV_# cg#.
connectVc_cont1: {R:relate _ V1 V1' X1}{CGV1:colorvertex V1 (s C)}
                 {CGV1':colorvertex V1' z}
                connectV_color (s C) CGX (<s E) (connectV_vertex D R ^ V)
                   (cgedge (cgedge CG F2 <=z !=z1 CGX CGV1') F2 F1 NE CGX CGV1)
                 <- connectV_color C CGX E' D CG
                 <- lemma8 E E'
                 <- lemma6 (<s E) NE
                 <- lemma7 (<s E) F2
                 <- lemma4 F1.

connectVc_cont2: {R:relate _ V1 V1' X1}{CGV1:colorvertex V1 z}
                 {CGV1':colorvertex V1' (s C)}
                connectV_color (s C) CGX (<s E) (connectV_vertex D R ^ V)
                      (cgedge (cgedge CG F2 F1 NE CGX CGV1') F2 <=z !=z1 CGX CGV1)
                 <- connectV_color C CGX E' D CG
                 <- lemma8 E E'
                 <- lemma6 (<s E) NE
                 <- lemma7 (<s E) F2
                 <- lemma4 F1.

% Correctness of connect2clique (VARS-TO-CLIQUE): Edges added between x_j and v_i,
% and x_j and v_i' do not connect vertices with same color when i!=j
connect2clique_color : {C:nat} connect2clique G -> coloring C G -> type.
connect2cliquec_base : connect2clique_color z connect2clique_# cg#.
connect2cliquec_cont1 : {R:relate U V V' X}{CGV:colorvertex V z}
                        {CGV':colorvertex V' (s C)}{CGX:colorvertex X (s C)}
                        connect2clique_color (s C)
                            (connect2clique_vtx R (D1 , D2, D3, D4) ^ VAR)
                            (cgunion CG1 (cgunion CG2 (cgunion CG3 CG4)))
                        <- connect2clique_color C D1 CG1'
                        <- increase_color CG1' CG1
                        <- connectX_color' C CGV D2 CG2
                        <- connectX_color C CGV' E D3 CG3
                        <- lemma5 E
                        <- connectV_color C CGX E D4 CG4.
connect2cliquec_cont2 : {R:relate U V V' X}{CGV:colorvertex V (s C)}
                        {CGV':colorvertex V' z}{CGX:colorvertex X (s C)}
                        connect2clique_color (s C)
                            (connect2clique_vtx R (D1 , D2, D3, D4) ^ VAR)
                            (cgunion CG1 (cgunion CG2 (cgunion CG3 CG4)))
                        <- connect2clique_color C D1 CG1'
                        <- increase_color CG1' CG1
                        <- connectX_color C CGV E D2 CG2
                        <- lemma5 E
                        <- connectX_color' C CGV' D3 CG3
                        <- connectV_color C CGX E D4 CG4.


% For verifying that environment is valid
necolor : {M:nat} {N:nat} M != N -> type.
necolor1: necolor z (s M) !=z1.
necolor2: necolor (s M) z !=z2.
necolor3: necolor (s M) (s N) (!=s D)
            <- necolor M N D.
lecolor : {M:nat}{N:nat} M <= N -> type.
lecolor1: lecolor z M <=z.
lecolor2: lecolor (s M) (s N) (<=s D)
           <- lecolor M N D.

% Correctness of coloring of the graph created by connecting a clause vertex with
% all literals not in the clause
conv'''_color : {C:nat} {C':nat} {C'':nat} colorvertex V (s C'') ->
                                            conv''' V G -> coloring C' G -> type.
conv'''c_base: conv'''_color z C' C'' CGV conv'''_# cg#.
conv'''c_cont1:{R:relate U1 V1 V1' _}{CGV1:colorvertex V1 (s C1)}
```

```
               {CGV1':colorvertex V1' z}
               conv'''_color (s C) C' C'' CGV (conv'''_vtx R D ^ V)
                          (cgedge (cgedge CG <=z F !=z2 CGV1' CGV) F1 F NE CGV1 CGV)
                <- conv'''_color C C' C'' CGV D CG
                <- necolor (s C'') (s C1) NE
                <- lecolor (s C'') C' F
                <- lecolor (s C1) C' F1.

conv''_color : {C:nat} conv'' F G -> sat F -> coloring C G -> type.

% Case: F=(pos U) \/ (pos U) \/ (pos U)
%       U=True
conv''c_1 : {R:relate U V V' _}{CGV:colorvertex V (s C')}{CGV':colorvertex V' z}
            conv''_color (s C) (conv''1 R D ^ VAR) E
                     (cgvertex ([c] [cgc] (cgedge (CG' c cgc) <=z F !=z2 CGV' cgc)) F)
            <- ({c:vertex}{cgc:colorvertex c (s C')}
                      conv'''_color C (s C) C' cgc (D c) (CG c cgc))
            <- lecolor (s C') (s C) F.

% Case: F=(pos U1) \/ (pos U1) \/ (pos U2)
%       U1=True, U2=True
conv''c_2A :{R1:relate U1 V1 V1' _}{CGV1:colorvertex V1 (s C1')}
            {CGV1':colorvertex V1' z}{R2:relate U2 V2 V2' _}
            {CGV2:colorvertex V2 (s C2')}{CGV2':colorvertex V2' z}
            conv''_color (s (s C)) (conv''2 R2 R1 D ^ VAR2 ^ VAR1) E
                      (cgvertex ([c] [cgc] (cgedge (cgedge
                          (CG c cgc) <=z F !=z2 CGV2' cgc) <=z F !=z2 CGV1' cgc)) F)
            <- ({c:vertex}{cgc:colorvertex c (s C1')}
                      conv'''_color C (s (s C)) C1' cgc (D c) (CG c cgc))
            <- lecolor (s C1') (s (s C)) F.

% Case: F=(pos U1) \/ (pos U1) \/ (pos U2)
%       U1=False, U2=True
conv''c_2B: {R1:relate U1 V1 V1' _}{CGV1:colorvertex V1 z}
            {CGV1':colorvertex V1' (s C1')}{R2:relate U2 V2 V2' _}
            {CGV2:colorvertex V2 (s C2')}{CGV2':colorvertex V2' z}
            conv''_color (s (s C)) (conv''2 R2 R1 D ^ VAR2 ^ VAR1) E
                      (cgvertex ([c] [cgc] (cgedge (cgedge
                          (CG c cgc) <=z F2 !=z2 CGV2' cgc) F1 F2 NE CGV1' cgc)) F2)
            <- ({c:vertex}{cgc:colorvertex c (s C2')}
                      conv'''_color C (s (s C)) C2' cgc (D c) (CG c cgc))
            <- lecolor (s C2') (s (s C)) F2
            <- lecolor (s C1') (s (s C)) F1
            <- necolor (s C2') (s C1') NE.

% Case: F=(pos U1) \/ (pos U1) \/ (pos U2)
%       U1=True, U2=False
conv''c_2C: {R1:relate U1 V1 V1' _}{CGV1:colorvertex V1 (s C1')}
            {CGV1':colorvertex V1' z}{R2:relate U2 V2 V2' _}
            {CGV2:colorvertex V2 z}{CGV2':colorvertex V2' (s C2')}
            conv''_color (s (s C)) (conv''2 R2 R1 D ^ VAR2 ^ VAR1) E
                      (cgvertex ([c] [cgc] (cgedge (cgedge
                          (CG c cgc) F2 F1 NE CGV2' cgc) <=z F1 !=z2 CGV1' cgc)) F1)
            <- ({c:vertex}{cgc:colorvertex c (s C1')}
                   conv'''_color C (s (s C)) C2' cgc (D c) (CG c cgc))
            <- lecolor (s C2') (s (s C)) F2
            <- lecolor (s C1') (s (s C)) F1
            <- necolor (s C1') (s C2') NE.

% The other 37 forms of clause F have similar proofs.

% Correctness of coloring of all graphs corresponding to all clauses
% on the continuation
conv'_color : {C:nat} conv' K G -> satK K -> coloring C G -> type.
conv'c_base : conv'_color C (conv'init ^ ()) satKempty cg#.
conv'c_cont : conv'_color C (conv'cont (D1 , D2)) (satK; E2 E1) (cgunion CG1 CG2)
              <- conv'_color C D1 E1 CG1
              <- conv''_color C D2 E2 CG2.

% Proof of the Reduction - from 3-SAT to Chromatic Number
reductionf : {C:nat} conv F C C' K G -> sat F -> satK K -> coloring C' G -> type.
reductionnew_t : reductionf C (convnew D) (satnewt E) F
                    (cgvertex ([v] [cgv] (cgvertex ([v'][cgv']
                    (cgvertex ([x][cgx] (cgedge (CG v v' x cgv cgv' cgx)
                              <=z H !=z2 cgv' cgv)) H)) <=z)) H)
                <- ({u:v} {hypt: hyp u true}{v:vertex}{v':vertex}{x:vertex}
                     {r:relate u v v' x}{var: var u}{cgv :colorvertex v (s C)}
                     {cgv':colorvertex v' z}{cgx :colorvertex x (s C)}
                     reductionf (s C) (D u v v' x r ^ var) (E u hypt) F
                                          (CG v v' x cgv cgv' cgx))
                <- ({u:v}{v:vertex}{v':vertex}{x:vertex}
                     {r:relate u v v' x}{var: var u}
                        conv_lemma (D u v v' x r ^ var) H).

reductionnew_f : reductionf C (convnew D) (satnewf E) F
                    (cgvertex ([v] [cgv] (cgvertex ([v'][cgv']
                    (cgvertex ([x][cgx] (cgedge (CG v v' x cgv cgv' cgx)
                              H <=z !=z1 cgv' cgv)) H)) H)) <=z)
                <- ({u:v} {hypf: hyp u false}{v:vertex}{v':vertex}{x:vertex}
                     {r:relate u v v' x}{var: var u}{cgv :colorvertex v z}
                     {cgv':colorvertex v' (s C)}{cgx :colorvertex x (s C)}
                     reductionf (s C) (D u v v' x r ^ var) (E u hypf) F
                                           (CG v v' x cgv cgv' cgx))
                <-({u:v}{v:vertex}{v':vertex}{x:vertex}
                    {r:relate u v v' x}{var: var u}
                      conv_lemma (D u v v' x r ^ var) H).

reduction/\ : reductionf C (conv/\ D) (sat/\ E1 E2) F CG
              <- reductionf C D E2 (satK; E1 F) CG.

reduction_base : reductionf C (conv3 (D1 , D2 , D3)) E F (cgunion CG1 (cgunion CG2 CG3))
                <- conv'_color C D1 (satK; E F) CG1
                <- clique_color C D2 CG2
                <- connect2clique_color C D3 CG3.
```