NP(FD): A Proof System for Finite Domain Formulas

Björn Carlson
Xerox PARC, 3333 Coyote Hill Rd
Palo Alto, CA, 94304

Mats Carlsson
Swedish Institute of Computer Science (SICS)
Box 1263, S-164 28 Kista, Sweden

Gunnar Stålmarck
Logikkonsult NP AB
Swedenborgsg. 2
118 48, Stockholm, Sweden

`bcarlson@parc.xerox.com` `matsc@sics.se` `gunnar@lk.se`

April 12, 1997

Abstract

We define a proof system for a language of finite domain (bounded arithmetic) formulas, closed under the propositional connectives. The system includes simple rules for propagating logical consequences and the Dilemma rule for assumption-based reasoning. Using the system, a saturation algorithm is given which checks the satisfiability of any formula in the language, and hence, it can also be used for verifying tautologies. The algorithm is a combination of two previous propagation algorithms, of which one is for propositional logic and one is for finite domain constraints.

Contents

Chapter 1

Introduction

In this report we define a sound and complete proof system NP(FD) for constraints in FDC, based on simple rules and the Dilemma rule. Using the proof rules, a saturation algorithm is developed, based on triplets, which can be used for consistency and truth checking of constraints in FDC. The algorithm has two primary areas of usage: as an extension of a previously defined algorithm for verification of propositional specifications [SS90], such as control system specifications, and as an improved constraint solver for use in finite domain constraint programming.

There are still other propagation techniques, in particular numerical methods for interval propagation, which should be added to NP(FD), that would further increase the power of the system. As shown in an evaluation, the arithmetic reasoning enables larger problems to be verified, and also makes the system appropriate for combinatorial satisfaction problems.

This report contains two major chapters: one the theory of NP(FD), and one on the evaluation of NP(FD).

The first chapter is structured as follows. Section 2.2 defines FDC and Section 2.3 defines the proof rules and show their soundness and completeness. In Section 2.4 we give the saturation algorithms for constraint propagation in FDC. Section 2.5 contains a discussion of related works.

The second chapter is structured as: Section 3.1 introduces two systems that exploits the arithmetic reasoning of NP(FD) to varying degrees. Section 3.2 compares the systems performances. Section 3.3 describes the benchmark sets used in the evaluation, and finally in Section 3.4 we discuss the results.

Chapter 2

The theory of NP(FD)

2.1 Introduction

Propagation algorithms have turned out to be quite powerful in proving theorems and solving combinatorial search problems [Hen89, HSD92b, JM94, DC93b, Car95]. In particular, algorithms specialized to discrete constraint satisfaction problems, so called finite domain problems, have been carefully studied [HDT92].

Similarly, propagation techniques tailored for propositional logic have been designed which compare favorably with other approaches to propositional theorem proving [DC93a, Rau94, SS90]. We focus in this article on the saturation algorithm developed and patented by Stålmarck for verifying propositional formulas [Stå94]. The method is based on a natural deduction style proof system fulfilling the so called subformula property [Pra65, Stå91, Har96], and resembles the semantic tableaux proof system of Mondadori [Mon88, D'A92]. The efficiency of the saturation algorithm compares favorably with binary-decision-diagram procedures [Bry86] and the Davis-Putnam [DP60] procedure. The saturation algorithm has also been used extensively and successfully in practice, where the applications have been mainly concerned with verifying the correctness of control system specifications [SÅ92, Bra94].

However, when applying the algorithm to formulas which contain (bounded) arithmetic the need for "clausifying" (skolemizing) the expressions is a bottleneck due to the rapid growth of formula sizes. Therefore, there is a strong demand for extending the algorithm with better support for arithmetic. In this paper we combine the algorithms from [Car95] with the abovementioned algorithm, hence, realizing a more efficient treatment of formulas combining propositional and arithmetic (finite domain) reasoning.

In short, we define a language FDC of finite domain constraints, which is closed under the propositional connectives and the arithmetic constraints. A sound and complete proof system NP(FD) for FDC is given, based on simple rules and the Dilemma rule. The system is further used as a basis for a saturation algorithm which checks the satisfiability of formulas in FDC, hence, it can also be used for verifying tautologies in FDC.

The proof system has been implemented in two variants, and the addition of finite domain propagation improves the efficiency of the system when applied to combinatorial problems and problems where the input domains are large.

2.2 FDC

Let FDC be the language that contains domain constraints, arithmetic constraints, and propositional formulas.

2.2.1 Domain constraints

A *domain constraint* is a formula t **in** $\{n_1, \ldots, n_k\}$, where n_i is an integer, $1 \leq i \leq k$. We say that t is a *finite domain term*, where t is either a variable, or an arithmetic term (see next section). If t is a variable x, x is called a finite domain variable.

2.2.2 Arithmetic constraints

A *linear arithmetic constraint* $x \cdot y$ is an equation $(x = y)$ or an inequation $(x \leq y)$ over linear expressions x and y, where a linear expression is of the form $n_1 x_1 \pm \ldots \pm n_k x_k \pm n_0$, where n_i is a constant and x_i a finite domain variable $(0 \leq i \leq k)$.

A *nonlinear arithmetic constraint* $x \cdot y$ is an equation $(x = y)$ or an inequation $(x \leq y)$ over nonlinear expressions x and y, where a nonlinear expression is of the form $n_1 x_1 y_1 \pm \ldots \pm n_k x_k y_k \pm n_0$, where n_i is a constant, x_i and y_i are finite domain variables $(0 \leq i \leq k)$. We only consider products of two variables since products of three or more variables can be transformed into products of only two variables by replacing $xyz = t$ with $xz' = t \wedge yz = z'$, for some new variable z'.

An arithmetic constraint (linear or nonlinear) $c(x_1, \ldots, x_k)$ thus defines a k-ary relation R_c as a subset of $\mathcal{Z}_1 \times \cdots \times \mathcal{Z}_k$.

An arithmetic constraint c is in *normal form* if

$$c \equiv (S + j) \cdot (T + j')$$

where $S = a_1 * x_1 + \ldots + a_k * x_k$, $T = a_{k+1} * x_{k+1} + \ldots + a_n * x_n$, each x_i is a distinct variable, each coefficient a_i is an integer > 0, j and j' are integers such that $j = 0 \wedge j' \geq 0 \vee j \geq 0 \wedge j' = 0$, and $\cdot \in \{\leq, \geq, <, >\}$.

2.2.3 Propositional formulas

A *propositional formula* is either a propositional variable p, an arithmetic constraint a (assumed to be in normal form), or a formula $A \wedge B$, $A \vee B$, $\neg A$, $A \rightarrow B$, $A \leftrightarrow B$, where A and B are propositional formulas.

2.3 NP(FD)

NP(FD) is a proof system for FDC based on simple rules for constraint propagation, and the Dilemma rule for assumption-based reasoning [SS90]. The underlying idea of the proof system is to saturate a set of formulas by applying simple rules until either an inconsistency is detected, or a fixed-point is reached. If a fixed-point is reached, branching is done by applying the Dilemma rule, which selects a variable and tries assigning it the value "true" and "false" (or a number in the case of an arithmetic variable) respectively. For each assignment, the new set of formulas is saturated, and the initial formula set is extended with the intersection of the saturated sets thus generated. This is repeated until the intersection steps only compute subsets of the formula set. At this point, the selection step instead considers two variables each time. The above description can now easily be generalized to the case where k variables are chosen at each selection step.

Let Γ be a set of of propositional formulas. We assume that for each domain variable x in Γ, there is a domain constraint x **in** d in ψ.

In the following we use Δ and Γ to denote sets, and A and B to denote formulas.

2.3.1 Simple rules

Each simple rule `R` is given as

$$\frac{A_1 \ \ldots \ A_k}{B} \quad \texttt{(R)}$$

which is read as: given A_i, $1 \leq i \leq k$, B follows.

An *interpretation* M assigns a value $M(x)$ to each variable x, and interprets the logical and arithmetical operators in the "obvious" way.

`R` is *sound*, i.e. $A_1, \ldots, A_k \models B$, if whenever A_i is true in a interpretation M, $1 \leq i \leq k$, B is true in M. A formula A is true in M if

- $A \equiv t$ **in** d, and $M(t) \in d$, i.e. t is evaluated in M to a number in its domain.
- $A \equiv x \cdot y$, where $\cdot \in \{=, \neq, \leq, \geq, <, >\}$ and $M(x) \cdot M(y)$ is true.
- $A \equiv p$, for some propositional variable, and $M(p)$ is true.
- $A \equiv B_1 \cdot B_2$, where $\cdot \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$ and $M(B_1) \cdot M(B_2)$ is true.
- $A \equiv \neg B$, and $M(B)$ is false.

If A is true in M, we say that M is a model for A.

A set of simple rules $\mathcal{R}$ for a propositional formula A is *model-complete*, if A or $\neg A$ can be derived from $l_1, \ldots, l_n, x_1$ **in** $\{n_1\}, \ldots, x_m$ **in** $\{n_m\}$ using rules in $\mathcal{R}$ only, where $l_i = p_i$ or $l_i = \neg p_i$, $n_j \in N$, and p_i and x_j, $1 \leq i \leq n$, $1 \leq j \leq m$, are those variables that occur in A.

Hence, the rules for falsity are defined as:

$$\frac{A \quad \neg A}{\bot} \quad (\mathtt{E}_{1\bot})$$

$$\frac{t \textbf{ in } \emptyset}{\bot} \quad (\mathtt{E}_{2\bot})$$

and

$$\frac{\bot}{A} \quad (\mathtt{I}_{\bot})$$

The simple rules are partitioned into introduction and elimination rules for domain constraints, arithmetic and propositional constraints.

Domains

The rule for domain constraints is defined as:

$$\frac{t \textbf{ in } d_1 \quad t \textbf{ in } d_2}{t \textbf{ in } d_1 \cap d_2} \quad (\textbf{ in })$$

We only consider rules for $+$ and $*$, since all normal form arithmetic constraints are thereby treated.

$$\frac{t_2 \textbf{ in } d_2 \quad t_1 + t_2 \textbf{ in } d}{t_1 \textbf{ in } d - d_2} \quad (\mathtt{E}_{1+}) \qquad \frac{t_1 \textbf{ in } d_1 \quad t_1 + t_2 \textbf{ in } d}{t_2 \textbf{ in } d - d_1} \quad (\mathtt{E}_{2+})$$

$$\frac{t_2 \textbf{ in } d_2 \quad t_1 * t_2 \textbf{ in } d}{t_1 \textbf{ in } \lceil d/d_2 \rceil \cap \lfloor d/d_2 \rfloor} \quad (\mathtt{E}_{1*}) \qquad \frac{t_1 \textbf{ in } d_1 \quad t_1 * t_2 \textbf{ in } d}{t_2 \textbf{ in } \lceil d/d_1 \rceil \cap \lfloor d/d_1 \rfloor} \quad (\mathtt{E}_{2*})$$

$$\frac{t_1 \textbf{ in } d_1 \quad t_2 \textbf{ in } d_2}{t_1 + t_2 \textbf{ in } d_1 + d_2} \quad (\mathtt{I}_{+}) \qquad \frac{t_1 \textbf{ in } d_1 \quad t_2 \textbf{ in } d_2}{t_1 * t_2 \textbf{ in } d_1 * d_2} \quad (\mathtt{I}_{*})$$

Arithmetic

$$\frac{t = n}{t \textbf{ in } \{n\}} \quad (\mathtt{I}_{N})$$

$$\frac{t_1 = t_2 \quad t_2 \textbf{ in } d_2}{t_1 \textbf{ in } d_2} \quad (\mathtt{E}_{1=}) \qquad \frac{t_1 = t_2 \quad t_1 \textbf{ in } d_1}{t_2 \textbf{ in } d_1} \quad (\mathtt{E}_{2=})$$

$$\frac{t_1 \neq t_2 \quad t_1 \textbf{ in } d \quad t_2 \textbf{ in } \{n\}}{t_1 \textbf{ in } d \setminus \{n\}} \quad (\mathtt{E}_{1\neq}) \qquad \frac{t_1 \neq t_2 \quad t_1 \textbf{ in } \{n\} \quad t_2 \textbf{ in } d}{t_2 \textbf{ in } d \setminus \{n\}} \quad (\mathtt{E}_{2\neq})$$

$$\frac{t_1 \leq t_2 \quad t_1 \textbf{ in } d_1 \quad t_2 \textbf{ in } d_2}{t_1 \textbf{ in } \{\textbf{min}(d_1), \ldots, \textbf{max}(d_2)\}} \quad (\mathtt{E}_{1\leq}) \qquad \frac{t_1 \leq t_2 \quad t_1 \textbf{ in } d_1 \quad t_2 \textbf{ in } d_2}{t_2 \textbf{ in } \{\textbf{min}(d_1), \ldots, \textbf{max}(d_2)\}} \quad (\mathtt{E}_{2\leq})$$

$$\frac{t_1 \textbf{ in } \{n\} \quad t_2 \textbf{ in } \{n\}}{t_1 = t_2} \quad (\mathtt{I}_{=}) \qquad \frac{t_1 \textbf{ in } d_1 \quad t_2 \textbf{ in } d_2 \quad d_1 \cap d_2 = \emptyset}{t_1 \neq t_2} \quad (\mathtt{I}_{\neq})$$

$$\frac{t_1 \textbf{ in } d_1 \; t_2 \textbf{ in } d_2 \; \textbf{max}(d_1) {\leq} \textbf{min}(d_2)}{t_1 \leq t_2} \quad (\mathtt{I}_{1\leq}) \qquad \frac{t_1 \textbf{ in } d_1 \; t_2 \textbf{ in } d_2 \; \textbf{min}(d_1) {>} \textbf{max}(d_2)}{t_1 > t_2} \quad (\mathtt{I}_{2\leq})$$

Negation

$$\frac{\neg\neg A}{A}\ (\mathtt{E}_{\neg}) \qquad \frac{A}{\neg\neg A}\ (\mathtt{I}_{\neg})$$

Conjunction

$$\frac{A \wedge B}{A}\ (\mathtt{E}_{1\wedge}) \qquad \frac{A \wedge B}{B}\ (\mathtt{E}_{2\wedge})$$

$$\frac{\neg(A \wedge B) \quad B}{\neg A}\ (\mathtt{E}_{3\wedge}) \qquad \frac{\neg(A \wedge B) \quad A}{\neg B}\ (\mathtt{E}_{4\wedge})$$

$$\frac{\neg A}{\neg(A \wedge B)}\ (\mathtt{I}_{1\wedge}) \qquad \frac{\neg B}{\neg(A \wedge B)}\ (\mathtt{I}_{2\wedge})$$

$$\frac{A \quad B}{A \wedge B}\ (\mathtt{I}_{3\wedge})$$

Disjunction

$$\frac{\neg(A \vee B)}{\neg A}\ (\mathtt{E}_{1\vee}) \qquad \frac{\neg(A \vee B)}{\neg B}\ (\mathtt{E}_{2\vee})$$

$$\frac{A \vee B \quad \neg B}{A}\ (\mathtt{E}_{3\vee}) \qquad \frac{A \vee B \quad \neg A}{\neg B}\ (\mathtt{E}_{4\vee})$$

$$\frac{A}{A \vee B}\ (\mathtt{I}_{1\vee}) \qquad \frac{B}{A \vee B}\ (\mathtt{I}_{2\vee})$$

$$\frac{\neg A \quad \neg B}{\neg(A \vee B)}\ (\mathtt{I}_{3\vee})$$

Implication

$$\frac{\neg(A \rightarrow B)}{A}\ (\mathtt{E}_{1\rightarrow}) \qquad \frac{\neg(A \rightarrow B)}{\neg B}\ (\mathtt{E}_{2\rightarrow})$$

$$\frac{A \rightarrow B \quad \neg B}{\neg A}\ (\mathtt{E}_{3\rightarrow}) \qquad \frac{A \rightarrow B \quad A}{B}\ (\mathtt{E}_{4\rightarrow})$$

$$\frac{\neg A}{A \rightarrow B}\ (\mathtt{I}_{1\rightarrow}) \qquad \frac{B}{A \rightarrow B}\ (\mathtt{I}_{2\rightarrow})$$

$$\frac{A \quad \neg B}{\neg(A \rightarrow B)}\ (\mathtt{I}_{3\rightarrow})$$

Equivalence

$$\frac{A \leftrightarrow B \quad \neg A}{\neg B}\ (\mathtt{E}_{1\leftrightarrow}) \qquad \frac{A \leftrightarrow B \quad \neg B}{\neg A}\ (\mathtt{E}_{2\leftrightarrow})$$

$$\frac{A \leftrightarrow B \quad A}{B}\ (\mathtt{E}_{3\leftrightarrow}) \qquad \frac{A \leftrightarrow B \quad B}{A}\ (\mathtt{E}_{4\leftrightarrow})$$

$$\frac{\neg(A \leftrightarrow B) \quad \neg A}{B}\ (\mathtt{E}_{5\leftrightarrow}) \qquad \frac{\neg(A \leftrightarrow B) \quad \neg B}{A}\ (\mathtt{E}_{6\leftrightarrow})$$

$$\frac{\neg(A \leftrightarrow B) \quad A}{\neg B}\ (\mathtt{E}_{7\leftrightarrow}) \qquad \frac{\neg(A \leftrightarrow B) \quad B}{\neg A}\ (\mathtt{E}_{8\leftrightarrow})$$

$$\frac{A \quad B}{A \leftrightarrow B}\ (\mathtt{I}_{1\leftrightarrow}) \qquad \frac{\neg A \quad \neg B}{A \leftrightarrow B}\ (\mathtt{I}_{2\leftrightarrow})$$

$$\frac{A \quad \neg B}{\neg(A \leftrightarrow B)}\ (\mathtt{I}_{3\leftrightarrow}) \qquad \frac{\neg A \quad B}{\neg(A \leftrightarrow B)}\ (\mathtt{I}_{4\leftrightarrow})$$

Soundness

The following lemma is easily proven by inspection of each rule.

Lemma 2.3.1 *The simple rules are sound.*

We now prove that the simple rules are model-complete for any propositional formula.

Lemma 2.3.2 *Let A be a propositional formula. Then, A or $\neg A$ can be derived from $\Gamma = l_1, \ldots, l_n, x_1$ **in** $\{n_1\}, \ldots, x_m$ **in** $\{n_m\}$ using simple rules only, where $l_i = p_i$ or $l_i = \neg p_i$, n_j is some integer, and p_i and x_j, $1 \leq i \leq n$, $1 \leq j \leq m$, are those variables that occur in A.*

Note that Γ has a model M_Γ, constructed by letting $M_\Gamma(x_j) = n_j$, $M_\Gamma(p_i)$ to be false if $l_i = \neg p_i$, and $M_\Gamma(p_i)$ to be true otherwise. Thus, we say Γ is a *model-set*.

Proof: We now proceed by induction on A, where we consider only a few cases. Let A be a propositional variable p, then either p or $\neg p$ is contained in Γ. Hence, the proposition follows trivially.

Let $A \equiv t_1 \cdot t_2$, where $\cdot$ is an arithmetic equality symbol. Using the introductory rules for terms (Section 2.3.1), t_1 **in** d_1 and t_2 **in** d_2 are derivable for two singleton domains $d_1 = \{n_1\}$ and $d_2 = \{n_2\}$. Suppose $\cdot$ is $=$. Then, if $n_1 \neq n_2$, $t_1 \neq t_2$ is derivable using the rule $\mathtt{I}_{\neq}$, and if $n_1 = n_2$, $t_1 = t_2$ is derivable using the rule $\mathtt{I}_{=}$. Similar reasoning proves the case for the other equality symbols.

Let $A \equiv \neg\neg B$. By induction it follows that either B or $\neg B$ is derivable from Γ. Hence, either $\neg\neg B$ or $\neg\neg\neg B$ is derivable using the rule $\mathtt{I}_{\neg}$.

Finally, let $A \equiv A_1 \cdot A_2$ or $A \equiv \neg(A_1 \cdot A_2)$, where $\cdot \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$. Then, by induction, it follows that either A_i or $\neg A_i$ is derivable from Γ, $i = 1, 2$. Suppose

A_1 and A_2 are both derivable, and that $\cdot$ is $\wedge$. Then, by the introduction rule $\mathtt{I}_3\wedge$ for $\wedge$, A is derivable. Suppose instead $\neg A_1$ is derivable. Then, $\neg(A_1 \wedge A_2)$ is derivable, using $\mathtt{I}_1\wedge$. The other cases for A_i follows similarly, as well as the other cases for $\cdot$. $\Box$

Lemma 2.3.3 *Let Γ be a model-set for Δ and A. Suppose $\Delta \models A$. Then A is provable from Γ.*

Proof: Since the simple rules are model-complete, either A or $\neg A$ is provable from Γ, and since the rules are sound it immediately follows that A is provable from Γ. $\Box$

2.3.2 Dilemma rule

The Dilemma rule is used for reasoning with assumptions, and can be seen as an extension of Reductio of natural deduction. In the following, let Δ and Γ be sets. We denote Δ by $\perp$ whenever $\perp \in \Delta$, and furthermore, we let $\perp \cap \Delta = \Delta$, for any Δ.

The Dilemma rule is defined as:

$$\frac{\dfrac{\Delta}{\begin{array}{cc} \Delta, A & \Delta, \neg A \\ \vdots & \vdots \\ \Gamma & \Gamma' \end{array}}}{\Gamma \cap \Gamma'} \quad \texttt{(D)}$$

Proposition 2.3.1 *Dilemma is sound.*

Proof: By an inductive argument on the number of applications of Dilemma, using the soundness of the simple rules, $\Delta, A \models \Gamma$ and $\Delta, \neg A \models \Gamma'$. Hence, suppose M is a model for Δ. Then, either A is true in M, i.e. M is a model for Γ, or A is false in M, i.e. M is a model for Γ'. Thus, in any case M is a model for $\Gamma \cap \Gamma'$. $\Box$

We say that an application of Dilemma is *open* until its assumptions are closed by a matching conclusion.

In the following we also use two special versions of Dilemma:

$$\frac{\dfrac{\Delta, t \ \mathbf{in} \ \{n_1, \ldots, n_k\}}{\begin{array}{ccc} \Delta, t \ \mathbf{in} \ \{n_1\} & & \Delta, t \ \mathbf{in} \ \{n_k\} \\ \vdots & \cdots & \vdots \\ \Gamma_1 & & \Gamma_k \end{array}}}{\Gamma_1 \cap \cdots \cap \Gamma_k} \quad \texttt{(Dd)}$$

derivable from the original Dilemma as follows:

$$
\begin{array}{c}
\Delta, t \textbf{ in } \{n_1, \ldots, n_k\} \\
\hline
\begin{array}{lcl}
\dfrac{\Delta, t = n_1}{t \textbf{ in } \{n_1\}}\ (\mathtt{I}_N,\mathtt{E}_{1=}) & \cdots & \dfrac{\Delta, t \neq n_1}{t \textbf{ in } \{n_2, \ldots, n_k\}}\ (\mathtt{E}_{1\neq}) \\
\vdots & & \vdots \\
\Gamma_1 & & \Gamma_2 \cap \cdots \cap \Gamma_k
\end{array} \\
\hline
\Gamma_1 \cap \cdots \cap \Gamma_k
\end{array}\ (\mathtt{D})
$$

and Reductio, `Red`, where we require that either Γ or Γ' is $\perp$ in the above definition of `D`.

A finite set Γ is a *derivation* of Γ from Γ. Let Π be a derivation of Γ' from Γ. If Ω can be formed from Γ' either by the application of a simple rule or by Dilemma, Π together with the application is a derivation of Ω from Γ.

A *refutation* of a set Γ is a derivation of $\perp$ from Γ.

A *proof* of a formula A is a refutation of $\neg A$.

A is derivable from Γ if there exists a derivation of Γ' from Γ such that $A \in \Gamma'$.

Let Π be a derivation and $d(\Pi)$ be the number of simultaneously open assumptions in Π. Also, let Γ be refutable and $h(\Gamma)$ denote the smallest n for which $n = d(\Pi)$, where Π is a refutation of Γ. We call $d(\Pi)$ the *degree* and $h(\Gamma)$ the *hardness* of Γ.

$\Delta \vdash^i A$ is defined as: A can be proven from Δ by a derivation of degree i.

Lemma 2.3.4 *Suppose A can be proven from Γ using a derivation Π of degree n, where the assumptions are made using* `D`*. Then there is a derivation using* `Red` *instead of A from Γ of degree $2n$.*

Proof: Consider the special case of `D` as:

$$
\begin{array}{c}
\hline
\begin{array}{cc}
A & \neg A \\
\vdots & \vdots \\
B & B
\end{array} \\
\hline
B
\end{array}\ (\mathtt{D})
$$

The same derivation in `Red` comes out as:

$$
\begin{array}{c}
\hline
\begin{array}{cc}
\begin{array}{c} B \\ \vdots \\ B \end{array} &
\begin{array}{c}
\neg B \\
\hline
\begin{array}{cc} A & \neg A \\ \vdots & \vdots \\ \perp & \perp \end{array} \\
\hline
\perp
\end{array}\ (\mathtt{Red})
\end{array} \\
\hline
B
\end{array}\ (\mathtt{Red})
$$

Clearly, the degree is 1 and 2 in each case respectively. The restriction can now easily be removed, and the lemma follows inductively over n. $\square$

2.3.3 Completeness

Let $\Delta \vdash A$ denote that A is provable from Δ using simple rules and Dilemma only.

Proposition 2.3.2 *If $\Delta \models A$, then $\Delta \vdash A$.*

Proof: Let $p_1, \ldots, p_n$, and $x_1, \ldots, x_m$ be the variables that occur in Δ and A. The proof is based on using Dilemma to generate different model-sets Γ, and then applying Lemma 2.3.3 to derive $\perp$ or A in each branch. Thus, consider the following proof outline, where Δ contains x_1 **in** $\{n_1, \ldots, n_k\}$:

$$\dfrac{\dfrac{\Delta}{\Delta, A \qquad \dfrac{\Delta, \neg A}{\Pi_1 \quad \Pi_2}}}{A}$$

where Π_1 is outlined as:

$$\dfrac{\dfrac{\Delta, \neg A, p_1}{\begin{array}{lcl} \Delta, \neg A, p_1, x_1 \textbf{ in } \{n_1\} & & \Delta, \neg A, p_1, x_1 \textbf{ in } \{n_k\} \\ \vdots & \cdots & \vdots \\ \perp & & \perp \end{array}}}{\perp} \; \texttt{(Dd)}$$

and Π_2 is outlined as:

$$\dfrac{\dfrac{\Delta, \neg A, \neg p_1}{\begin{array}{lcl} \Delta, \neg A, \neg p_1, x_1 \textbf{ in } \{n_1\} & & \Delta, \neg A, \neg p_1, x_1 \textbf{ in } \{n_k\} \\ \vdots & \cdots & \vdots \\ \perp & & \perp \end{array}}}{\perp} \; \texttt{(Dd)}$$

Each sub-tree, rooted in the node for $\Delta, \neg A$, thus ends with a model-set Γ for Δ, A. From Lemma 2.3.3 it follows that A is provable from Γ, thus, $\perp$ follows from A and $\neg A$. Thereby, the final Dilemma application introduces A. $\square$

2.3.4 Upper-bound complexity

A set of formulas Δ in NP(FD) is *i-saturated* if $\Delta = \{A : \Delta \vdash^i A\}$. Hence, a 1-saturated set Δ is such that each formula in Δ is proven by a derivation where any application of Dilemma does not itself use Dilemma.

In the following we only consider proofs from Δ where all terms and formulas in the proofs are subexpressions of some terms and formulas in Δ. Given a term or formula t, the number of subexpressions of t is bounded by $2i+1$, where i is the number of occurrences of any functor or equality symbol in t, i.e. $|t| = 2i+1$. We let the size of Δ, $|\Delta|$, be defined as the accumulated size of all the formulas in Δ. The size of a derivation Π is defined by the number of derivation steps in Π.

Proposition 2.3.3 *Consider a derivation Π of $\Delta \vdash^k A$, and let m, where $m \geq 2$, be the domain size of the subterm in Δ with the largest domain. Then, Π is $O(m^{2k+1}|\Delta|^{k+1})$ in size.*

Proof: The proof is made by induction over k. Let $k = 0$. Thus: Π is on the form

$$\Pi : \quad \frac{\dfrac{\Delta}{\quad}}{\vdots \atop A}$$

where only simple rules are used. By the assumption that only subexpressions of any domain constraint in Δ can occur in Π, using `dom` we can assume that for each t that occurs in Π, there exists a largest d such that t **in** d occurs in Π. Furthermore, we assume that no t **in** d occurs twice or more as the conclusion of any simple rule.

Consider the elimination rules. For each application of an elimination rule, the domain inferred is smaller or equal in size to the domains in the assumption of the rule. Hence, there can only be $m|\Delta|$ applications of elimination rules in Π, using the subexpression assumption.

Consider now the introduction rules. For each application of an introduction rule, a subformula of a formula in Δ is introduced, using the subexpression assumption, and thereby introduction rules can only be applied $|\Delta|$ times. Hence, Π is $O(m|\Delta|)$ in size.

Suppose $k = l + 1$. We thereby have the following derivation Π:

$$\begin{array}{cc}
\Pi_1 : & \dfrac{\dfrac{\Delta_0 = \Delta}{\vdots}}{\Delta_1}\ \texttt{(D)} \\
\hline
 & \vdots \\
\hline
\Pi_n : & \dfrac{\dfrac{\Delta_n}{\vdots}}{\Delta_{n+1} = A}\ \texttt{(D)}
\end{array}$$

where Π_i is an application of Dilemma as:

$$\Pi_i : \quad \dfrac{\dfrac{\Delta_{i-1}, t \textbf{ in } \{n_1, \ldots, n_m\}}{\begin{array}{lcl} \Delta_{i-1}, t \textbf{ in } \{n_1\} & & \Delta_{i-1}, t \textbf{ in } \{n_m\} \\ \vdots & \cdots & \vdots \\ \Gamma_{1i} & & \Gamma_{mi} \end{array}}}{\Delta_i = \Gamma_{1i} \cap \cdots \cap \Gamma_{mi}} \; \texttt{(Dd)}$$

or as:

$$\Pi_i : \quad \dfrac{\dfrac{\Delta_{i-1}}{\begin{array}{ll} \Delta_{i-1}, A & \Delta_{i-1}, \neg A \\ \vdots & \vdots \\ \Gamma_i & \Gamma_i' \end{array}}}{\Delta_i = \Gamma_i \cap \Gamma_i'} \; \texttt{(D)}$$

By induction, the $\vdash^l$ derivation of Γ_{ji}, $1 \leq j \leq m$, is $O(m^{2l+1}|\Delta|^{l+1})$ in size. Hence, the total size of the derivation Π_i is $O(mm^{2l+1}|\Delta|^{l+1})$. The number n of proofs Π_i is bounded by $m|\Delta|$, since each Δ_i contains only subformulas of formulas in Δ. Therefore Π is $O(m|\Delta|mm^{2l+1}|\Delta|^{l+1})$ in size, i.e. $O(m^{2k+1}|\Delta|^{k+1})$ in size.

The second case follows similarly, since $m \geq 2$. □

2.3.5 The hierarchy theorems

This section concerns lower bounds of proofs in NP(FD). First we show that the degrees of $\vdash$ collapse if we make no assumption on what formulas that can be used in a proof.

Proposition 2.3.4 *Suppose $\Delta \vdash^i A$ for some i. Then there is a derivation $\Delta \vdash^1 A$.*

Proof: The proposition is proven by recasting $\vdash$ in terms of derivability in a complete Hilbert system. The Hilbert system H is defined by the following axioms

A_1 $A \rightarrow (B \rightarrow A)$

A_2 $(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$

A_3 $(\neg A \rightarrow \neg B) \rightarrow (B \rightarrow A)$

and the following proof rules:

$$\frac{\Delta}{\Delta \cup \{A_i\}} \; \texttt{(Ax)}$$

where $i \in \{1,2,3\}$, and

$$\frac{\Delta \cup \{A, A \to B\}}{\Delta \cup \{A, A \to B, B\}} \quad \texttt{(MP)}$$

The system H is complete [Bar77].

We now extend H with new axioms corresponding to the simple rules of NP(FD), and thus the Ax-rule above is extended too. Hence, H is extended with

$$\{A \to B \colon \frac{A}{B} \quad \text{is a simple rule}\}$$

and

$$\{A \to (B \to C) \colon \frac{A \quad B}{C} \quad \text{is a simple rule}\}$$

Let $\vdash_H$ denote derivability in H.

Claim: If $\Delta \vdash^i A$, then $\Delta \vdash_H A$. Let Π be a derivation $\Delta \vdash^i A$. We show how to transform Π into a proof in $\vdash_H$ instead.

Consider the application of a simple rule $\frac{A}{B}$ in Π. The simple rule is thus applied in H as:

$$\cfrac{\cfrac{\Delta \cup \{A\}}{\Delta \cup \{A, A \to B\}} \quad \texttt{(Ax)}}{\Delta \cup \{A, A \to B, B\}} \quad \texttt{(MP)}$$

Similarly, consider the application of a simple rule $\frac{A \quad B}{C}$ in Π. The rule is applied in Π' as:

$$\cfrac{\cfrac{\cfrac{\Delta \cup \{A, B\}}{\Delta \cup \{A, B, A \to (B \to C)\}} \quad \texttt{(Ax)}}{\Delta \cup \{A, B, A \to (B \to C), (B \to C)\}} \quad \texttt{(MP)}}{\Delta \cup \{A, B, A \to (B \to C), (B \to C), C\}} \quad \texttt{(MP)}$$

Consider now an application of Dilemma:

$$\cfrac{\cfrac{\Delta}{\Delta, A \qquad \Delta, \neg A} \\ \begin{array}{cc} \vdots & \vdots \\ \Gamma & \Gamma' \end{array}}{\Gamma = \Gamma \cap \Gamma'} \quad \texttt{(D)}$$

By the soundness of $\vdash$, $\Delta \models \Gamma$. Also, $\Delta \models \Gamma$ iff for any formula B; $\Delta, B \models \Gamma$ and $\Delta, \neg B \models \Gamma$. By the completeness of $\vdash$; $\Delta, B/\neg B \vdash \Gamma$. Inductively, thereby $\Delta, B/\neg B \vdash_H \Gamma$. Hence, by the completeness of H; $\Delta \vdash_H \Gamma$. Thus, $\Delta \vdash^i A$ implies $\Delta \vdash_H A$.

We now show that any proof in H can be transformed into a derivation in $\vdash^1$. There are two rules we need to consider: `Ax` and `MP`. The latter rule is identical to $\mathtt{E}_{4\to}$, so it remains to show how applications of `Ax` are transformed.

Suppose we are given an instance:

$$\frac{\Delta}{\Delta \cup \{A\}} \quad (\mathtt{Ax})$$

for some axiom A, where A is either A_i, for some $i \in \{1, 2, 3\}$, or one of the axioms added from a simple rule.

Such an instance can be transformed into a $\vdash^1$ derivation as follows. We consider only two examples of which the other are simple variants.

First, consider the use of axiom A_1.

$$\dfrac{\dfrac{\Delta}{A \to (B \to A) \qquad \dfrac{\dfrac{\dfrac{\neg(A \to (B \to A))}{A \quad \neg(B \to A)}\ \mathtt{E}_{1\to},\mathtt{E}_{2\to}}{\neg A}\ \mathtt{E}_{1\to}}{\bot}\ (\mathtt{E}_{1\bot})}}{\Delta, A \to (B \to A)}\ (\mathtt{D})$$

Clearly, the derivation in NP(FD) is $\vdash^1$.

Then consider a use of a simple rule $\dfrac{A \quad B}{C}$ $(\mathtt{R})$.

$$\dfrac{\Delta}{A \to (B \to C) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\neg(A \to (B \to C))}{A \quad \neg(B \to C)}\ \mathtt{E}_{1\to},\mathtt{E}_{2\to}}{\dfrac{A \quad B}{C}\ (\mathtt{R}) \quad \neg C}\ \mathtt{E}_{1\to},\mathtt{E}_{2\to}}{\bot}\ (\mathtt{E}_{1\bot})}{\Delta, A \to (B \to C)}\ (\mathtt{D})}$$

Clearly, the derivation in NP(FD) is $\vdash^1$. □

We now assume that only subformulas of Δ can be used in a proof from Δ, which implies that the degrees no longer collapse.

Proposition 2.3.5 *Suppose any derivation starting with Δ contains only subformulas of Δ. Then for all $i \geq 1$, there exists Δ and A such that $\Delta \vdash^i A$, but for no $j < i$, $\Delta \vdash^j A$.*

In the following we use Dilemma for domain constraints, since the original Dilemma does not improve the degree for this problem instance.

Proof: Consider the formulas $\Gamma_n = \{x_i \in \{a_1, \ldots, a_n\} : 0 \leq i \leq n\}$, for some natural numbers a_k, and $\Gamma = \{x_i \neq x_j : 0 \leq i \neq j \leq n\}$. Let $\Delta = \Gamma \cup \Gamma_n$, and

$A = \bot$. Clearly, $\Delta \models A$. We prove the proposition by showing that for $n \geq 2$, each derivation of $\Delta \vdash A$, where only subformulas of Δ are used, is of degree $n-1$ or more. This is done by induction on n.

Suppose $n = 2$. The we can use the following Dilemma application to prove A. Let i be some number between 0 and n.

$$\dfrac{\dfrac{\Delta = \Gamma \cup \Gamma_2}{\begin{array}{lcl} \Delta, x_i \textbf{ in } \{a_1\} & & \Delta, x_i \textbf{ in } \{a_2\} \\ \vdots & \cdots & \vdots \\ \bot & & \bot \end{array}}}{\bot = A}\ \mathtt{(Dd)}$$

where a 0-degree derivation of $\Delta, x_i \textbf{ in } \{a_1\} \vdash \bot$ is (assume $i \neq j \neq k \neq i$)

$$\dfrac{\dfrac{\dfrac{\dfrac{x_i \neq x_j \quad x_i \textbf{ in } \{a_1\} \quad x_j \textbf{ in } \{a_1, a_2\}}{x_j \neq x_k \quad x_j \textbf{ in } \{a_2\} \quad x_k \textbf{ in } \{a_1, a_2\}}\ \mathtt{(E_{2\neq})}}{x_i \neq x_k \quad x_i \textbf{ in } \{a_1\} \quad x_k \textbf{ in } \{a_1\}}\ \mathtt{(E_{2\neq})}}{x_i \textbf{ in } \emptyset}\ \mathtt{(E_{2\neq})}}{\bot}\ \mathtt{I_\bot}$$

Consider now proving $\bot$ from Δ without using Dilemma. Thus, only simple rules can be applied. However, the only formulas in Δ are of the kind $x_i \neq x_j$ and $x_i \textbf{ in } \{a_1, a_2\}$. By inspection of the rules for $\neq$ it follows immediately that no simple rule applies.

Assume $n > 2$. Again, by inspection of the simple rules for $\neq$, no simple rule applies to Δ, since each for each $x \textbf{ in } d \in \Gamma_n$, $\mid d \mid > 1$, and thus no rule for equality applies. Hence, we can assume that any derivation of A from Δ starts with an application of Dilemma, i.e. (let i be some number between 1 and n)

$$\dfrac{\dfrac{\Delta = \Gamma \cup \Gamma_n}{\begin{array}{lcl} \Delta, x_i \textbf{ in } \{a_1\} & & \Delta, x_i \textbf{ in } \{a_n\} \\ \vdots & \cdots & \vdots \\ \bot & & \bot \end{array}}}{\bot = A}\ \mathtt{(Dd)}$$

Each of the proofs

$$\begin{array}{l} \Delta, x_i \textbf{ in } \{a_j\} \\ \vdots \\ \bot \end{array}$$

reduces to a derivation of $\Gamma \cup \Gamma_i^j \vdash \bot$ as follows, where $\Gamma_i^j = \{x_k \textbf{ in } \{a_1, \ldots, a_n\} \setminus a_j : 1 \leq k \neq i \leq n\}$.

First, we can assume that by using the $\mathtt{E}_{2\neq}$ rule $n-1$ times we reduce $\Delta, x_i \textbf{ in } \{a_j\}$ to $\Delta \cup \Gamma_i^j, x_i \textbf{ in } \{a_j\}$ as

$$\cfrac{\cfrac{\cfrac{\Delta, x_i \textbf{ in } \{a_j\}}{\Delta, x_i \textbf{ in } \{a_j\}, x_j \textbf{ in } \{a_1, \ldots, a_n\} \setminus a_j}\ \mathtt{E}_{2\neq}}{\vdots}\ \mathtt{E}_{2\neq}}{\Delta \cup \Gamma_i^j, x_i \textbf{ in } \{a_j\}}\ \mathtt{E}_{2\neq}$$

by $n-1$ applications of the $\mathtt{E}_{2\neq}$ rule. Since $\Delta = \Gamma \cup \Gamma_n$ and $\Gamma_i^j \models \Gamma_n$, we can thus simplify $\Delta \cup \Gamma_i^j, x_i \textbf{ in } \{a_j\}$ to $\Gamma' = \Gamma \cup \Gamma_i^j, x_i \textbf{ in } \{a_j\}$. The only formulas in $\Gamma \cup \Gamma_i^j$ that contain x_i are $x_i \neq x_j$, $1 \leq i \neq j \leq n$, and since for each $x_k \textbf{ in } d_k \in \Gamma_i^j$, $a_j \notin d_k$, it follows that $\Gamma' \vdash A$ iff $\Gamma \cup \Gamma_i^j \vdash A$. Finally, since $(\Gamma \cup \Gamma_i^j) \cap (\Gamma \cup \Gamma_i^k) = \Gamma$, the application of Dilemma at this stage does not add any new formula to Δ.

Now, it follows inductively that there exists no derivation of $\Gamma \cup \Gamma_i^j \vdash \perp$ of degree less than $n-2$. Hence, no derivation of $\Delta \vdash A$ is of degree less than $n-1$. □

Finally, we compare the lengths of proofs where only atomic assumptions can be made in each Dilemma application, with proofs where any assumption can be made.

Let $\vdash'$ denote proofs in NP(FD) where only atomic assumptions are made.

Proposition 2.3.6 *For all $n \geq 2$, there exists Δ and A such that $\Delta \vdash^1 A$, $\Delta \vdash'^n A$, but for no $k < n$, $\Delta \vdash'^k A$.*

Proof: Let $A = \perp$, and $\Delta = \Gamma \wedge \Omega$ be defined as

$$(\vee_{i=1}^n \wedge_{j=1}^n p_{ij}) \wedge (\vee_{i=1}^n \wedge_{j=1}^n \neg p_{ij})$$

i.e. Δ is true iff there exists i s.t. for all j, p_{ij} is true, and there exists i' s.t. for all j', $p_{i'j'}$ is false. Thus, $\Delta \models \perp$. Let Γ_i denote $\wedge_{j=1}^n p_{ij}$, and Ω_i denote $\wedge_{j=1}^n \neg p_{ij}$. Hence, consider the following outline:

$$\cfrac{\cfrac{\cfrac{\cfrac{\Delta}{\begin{matrix}\Gamma_1, \Omega & \\ \vdots & \neg\Gamma_1 \\ \perp & \end{matrix}}}{\neg\Gamma_1}}{\begin{matrix}\vdots\end{matrix}}}{\cfrac{\neg\Gamma_1 \wedge \cdots \wedge \neg\Gamma_n, \Gamma}{\perp}}$$

i.e. first Γ_1 is assumed, which together with Ω proves $\perp$, since from Γ_1 the elimination rules for $\wedge$ derives p_{1j}, for each j between 1 and n. Hence, using the introduction rules for $\wedge$, $\neg\Omega_i$ is proven, $1 \leq i \leq n$. Thereby, the introduction rules for $\vee$ proves $\neg\Omega$. Consequently, $\neg\Gamma_1$ is proven using a 1-proof. Similarly, $\neg\Gamma_i$ is proven, $1 \leq i \leq n$. Thus, by the introduction rules for $\wedge$, $\neg\Gamma_1 \wedge \cdots \wedge \neg\Gamma_n$ is proven, which together with Γ proves $\perp$ using the elimination rules for $\wedge$ and $\vee$. That is, $\Delta \vdash^1 \perp$.

Consider now proving $\perp$ using only atomic assumptions, i.e. any assumption A must be a propositional variable. For example, consider a $\vdash'^1$ proof:

$$\dfrac{\dfrac{\Delta}{\begin{array}{cc} \Delta, p & \Delta, \neg p \\ \vdots & \vdots \\ \Gamma & \Gamma' \end{array}}}{\Gamma \cap \Gamma'}$$

where the choice of p is arbitrary, p_{ij} say. Given Δ and p_{ij}, it follows, by inspection, that no simple rule applies since both Γ and Ω are disjunctions of conjunctions. Hence, only when n atomic assumptions $l_{i1}, \ldots, l_{in}$ are made, for some i between 1 and n, where $l_{ij} = p_{ij}$ or $l_{ij} = \neg p_{ij}$, the simple rules for $\wedge$ apply.

Now, let $\Delta' = \{l_1, \ldots, l_n\}$ be any one of these n atomic assumptions, i.e. Δ' is a model-set for Δ. Hence, by Lemma 2.3.3, $\Delta, \Delta' \vdash \perp$ using simple rules only. Thus, for any derivation of $\Delta \vdash'^k \perp$, it follows that $k \geq n$. $\square$

2.3.6 Equality rules

We now introduce equality rules which improve lower bounds of derivation degrees and lengths.

We assume in the following that the substitution rules for (in)equality holds. Also, we identify each formula A with a domain variable x_A as $x_A = 1 \leftrightarrow A$, where x_A **in** $\{0, 1\}$ holds. Hence, the equality rules for the propositional combinators are:

$$\frac{x_A \neq x_B}{x_A = x_{\neg B}} \;(\mathtt{Eq}_\neg 1) \qquad \frac{x_A = x_{\neg B}}{x_A \neq x_B} \;(\mathtt{Eq}_\neg 2)$$

$$\frac{x_A \textbf{ in } \{1\}}{x_B = x_{A \wedge B}} \;(\mathtt{Eq}_{1\wedge}) \qquad \frac{x_B \textbf{ in } \{1\}}{x_A = x_{A \wedge B}} \;(\mathtt{Eq}_{2\wedge})$$

$$\frac{x_A = x_B}{x_A = x_{A \wedge B}} \;(\mathtt{Eq}_{3\wedge}) \qquad \frac{x_A = x_B}{x_B = x_{A \wedge B}} \;(\mathtt{Eq}_{4\wedge})$$

$$\frac{x_A \neq x_B}{x_{A \wedge B} \textbf{ in } \{0\}} \;(\mathtt{Eq}_{5\wedge})$$

$$\frac{x_A \textbf{ in } \{0\}}{x_B = x_{A\vee B}}\ (\mathtt{Eq}_{1\vee}) \qquad \frac{x_B \textbf{ in } \{0\}}{x_A = x_{A\vee B}}\ (\mathtt{Eq}_{2\vee})$$

$$\frac{x_A = x_B}{x_A = x_{A\vee B}}\ (\mathtt{Eq}_{3\vee}) \qquad \frac{x_A = x_B}{x_B = x_{A\vee B}}\ (\mathtt{Eq}_{4\vee})$$

$$\frac{x_A \neq x_B}{x_{A\vee B} \textbf{ in } \{1\}}\ (\mathtt{Eq}_{5\vee})$$

$$\frac{x_A \textbf{ in } \{1\}}{x_B = x_{A\to B}}\ (\mathtt{Eq}_{1\to}) \qquad \frac{x_B \textbf{ in } \{0\}}{x_A \neq x_{A\to B}}\ (\mathtt{Eq}_{2\to})$$

$$\frac{x_A \neq x_B}{x_A \neq x_{A\to B}}\ (\mathtt{Eq}_{3\to}) \qquad \frac{x_A \neq x_B}{x_B = x_{A\to B}}\ (\mathtt{Eq}_{4\to})$$

$$\frac{x_A = x_B}{x_{A\to B} \textbf{ in } \{1\}}\ (\mathtt{Eq}_{5\to})$$

$$\frac{x_A = x_B}{x_{A\leftrightarrow B} \textbf{ in } \{1\}}\ (\mathtt{Eq}_{1\leftrightarrow}) \qquad \frac{x_A \neq x_B}{x_{A\leftrightarrow B} \textbf{ in } \{0\}}\ (\mathtt{Eq}_{2\leftrightarrow})$$

$$\frac{x_{A\leftrightarrow B} \textbf{ in } \{1\}}{x_A = x_B}\ (\mathtt{Eq}_{3\leftrightarrow}) \qquad \frac{x_{A\leftrightarrow B} \textbf{ in } \{0\}}{x_A \neq x_B}\ (\mathtt{Eq}_{4\leftrightarrow})$$

$$\frac{x_A \textbf{ in } \{1\}}{x_{A\leftrightarrow B} = x_B}\ (\mathtt{Eq}_{5\leftrightarrow}) \qquad \frac{x_A \textbf{ in } \{0\}}{x_{A\leftrightarrow B} \neq x_B}\ (\mathtt{Eq}_{6\leftrightarrow})$$

We now show how the derivation degrees are separated using the equality rules. In the following we will use Reductio rather than Dilemma for the sake of simplicity, but by lemma 2.3.4, the separation holds for both the rules.

Let U_n be the formula[Urq87]

$$p_1 \leftrightarrow (p_2 \leftrightarrow \ldots (p_n \leftrightarrow (p_1 \leftrightarrow (p_2 \leftrightarrow \ldots (p_{n-1} \leftrightarrow p_n) \ldots)))),$$

where the subformulas are labeled x_i and y_i as:

$$\begin{array}{l} x_i \leftrightarrow (p_i \leftrightarrow x_{i+1}), 1 \leq i < n, \\ x_n \leftrightarrow (p_n \leftrightarrow y_1), \\ y_i \leftrightarrow (p_i \leftrightarrow y_{i+1}), 1 \leq i < n-1, \\ y_{n-1} \leftrightarrow (p_{n-1} \leftrightarrow p_n) \end{array}$$

or written as a *table*:

$$\begin{array}{ll} x_1 \leftrightarrow (p_1 \leftrightarrow x_2) & y_1 \leftrightarrow (p_1 \leftrightarrow y_2) \\ x_2 \leftrightarrow (p_2 \leftrightarrow x_3) & y_2 \leftrightarrow (p_2 \leftrightarrow y_3) \\ \vdots & \vdots \\ x_{n-1} \leftrightarrow (p_{n-1} \leftrightarrow x_n) & y_{n-1} \leftrightarrow (p_{n-1} \leftrightarrow p_n) \\ x_n \leftrightarrow (p_n \leftrightarrow y_1) & \end{array}$$

Note that $x \leftrightarrow y \leftrightarrow z$ is true iff either one or three of x,y, and z are true. As a consequence, if one of the variables is true, other two variables are assigned the same value, whenever $x \leftrightarrow y \leftrightarrow z$ is true. It follows that U_n is true for any assignment of p_i, $1 \leq i \leq n$. Hence, $\neg U_n$ is inconsistent.

If we allow the use of the equality rules in the proofs, the hardness of $\neg U_n$ is no more than 2.

Proposition 2.3.7 *Suppose we consider proofs where the equality rules and Dilemma are allowed. Then $h(\neg U_n) \leq 2$, for all $n > 2$.*

Proof: We will use the mapping of subformulas to variables as given by the table for U_n. The derivation first assumes y_1 and $\neg y_1$ respectively.

Then, by the assumption y_1 and the formula $x_n \leftrightarrow (p_n \leftrightarrow y_1)$, it follows by simple rules only ($\mathtt{Eq}_{\leftrightarrow}$) that $x_n = p_n$. Hence, by the formulas $x_{n-1} \leftrightarrow (p_{n-1} \leftrightarrow x_n)$ and $y_{n-1} \leftrightarrow (p_{n-1} \leftrightarrow p_n)$, it follows by the equality rules only that $x_{n-1} = y_{n-1}$.

From $x_{i+1} = y_{i+1}$, the formulas $x_i \leftrightarrow (p_i \leftrightarrow x_{i+1})$ and $y_i \leftrightarrow (p_i \leftrightarrow y_{i+1})$, and the assumption of either p_i or $\neg p_i$ it follows in each case by simple rules only that $x_i = y_i$. Hence, by Dilemma, $x_i = y_i$ follows.

Thereby, for $i = 1$, we reach a contradiction since y_1 and $\neg x_1$ are true.

Similarly, from $(\neg x_1, \neg y_1)$, in steps of decreasing i, we prove $x_i \neq y_i$. Again, for $i = 1$ we reach a contradiction. It follows by Dilemma, that $\neg x_1 \vdash \perp$, i.e. $\neg U_n$ is contradictory.

An outline of the derivation is:

$$
\begin{array}{c}
\neg x_1 \\ \hline
\begin{array}{cc}
\begin{array}{c}
y_1 \\ \hline
\begin{array}{cc} p_{n-1} & p_{n-1} \\ \vdots & \vdots \\ x_{n-1} = y_{n-1} & x_{n-1} = y_{n-1} \end{array} \\ \hline
x_{n-1} = y_{n-1} \\ \vdots \\
\dfrac{x_1 = y_1}{\perp}\ \mathtt{E}_{\perp},\mathtt{E}_{=}
\end{array} \ (\mathtt{D})
&
\begin{array}{c}
\neg y_1 \\ \hline
\begin{array}{cc} p_{n-1} & \neg p_{n-1} \\ \vdots & \vdots \\ x_{n-1} \neq y_{n-1} & x_{n-1} \neq y_{n-1} \end{array} \\ \hline
x_{n-1} \neq y_{n-1} \\ \vdots \\
\dfrac{x_1 \neq y_1}{\perp}\ \mathtt{E}_{\perp},\mathtt{E}_{=}
\end{array} \ (\mathtt{D})
\end{array} \\ \hline
\perp
\end{array} \ (\mathtt{D})
$$

Since, at any point there are at most two open assumptions, one on y_1 and one on p_i, the degree of the derivation is 2. □

We show in the following that for increasing n the hardness of U_n increases logarithmically in the proof system based on the simple rules and `Red` but with no equality rules.

Lemma 2.3.5 *There is a subformula A of U_n such that $h(\neg U_n, y_1, A) < h(\neg U_n, y_1)$ or $h(\neg U_n, y_1, \neg A) < h(\neg U_n, y_1)$, $n > 2$.*

Proof: It follows by inspection of the simple rules that $(\neg U_n, y_1)$ is 0-saturated if $n > 2$. Hence, each refutation Ψ (using `Red`) of $(\neg U_n, y_1)$ has the form:

$$\dfrac{\dfrac{\neg U_n, y_1}{\begin{array}{ll} \neg U_n, y_1, A & \neg U_n, y_1, \neg A \\ \Pi & \Pi' \\ \Gamma & \Gamma' \end{array}}}{\vdots}\ (\texttt{Red})$$

where A is a subformula of U_n, where either Γ or Γ' is $\perp$. Assume that $h(\neg U_n, y_1) = d(\Psi) = m$, for some m. Now, suppose either $h(\neg U_n, y_1, A) = m$ or $h(\neg U_n, y_1, \neg A) = m$. Therefore, $d(\Pi) \geq m$ or $d(\Pi') \geq m$, since either Γ or Γ' is $\perp$. It then follows immediately that $d(\Pi) > m$, which is a contradiction. Hence, $h(\neg U_n, y_1, A) < h(\neg U_n, y_1)$ or $h(\neg U_n, y_1, \neg A) < h(\neg U_n, y_1)$. □

Proposition 2.3.8 *For $n > 2$, $h(\neg U_{2n+1}, y_1) > h(\neg U_n, y_1)$.*

Proof: We will prove this by showing how $(\neg U_{2n+1}, y_1, A)$ can be reduced to $(\neg U_n, y_1)$ for any subterm A of U_{2n+1}.

Consider the table of U_{2n+1} as follows:

$$\begin{array}{ll}
x_1 \leftrightarrow (p_1 \leftrightarrow x_2) & y_1 \leftrightarrow (p_1 \leftrightarrow y_2) \\
\vdots & \vdots \\
x_{n-1} \leftrightarrow (p_{n-1} \leftrightarrow x_n) & y_{n-1} \leftrightarrow (p_{n-1} \leftrightarrow y_n) \\
x_n \leftrightarrow (p_n \leftrightarrow x_{n+1}) & y_n \leftrightarrow (p_n \leftrightarrow y_{n+1}) \\
x_{n+1} \leftrightarrow (p_{n+1} \leftrightarrow x_{n+2}) & y_{n+1} \leftrightarrow (p_{n+1} \leftrightarrow y_{n+2}) \\
x_{n+2} \leftrightarrow (p_{n+2} \leftrightarrow x_{n+3}) & y_{n+2} \leftrightarrow (p_{n+2} \leftrightarrow y_{n+3}) \\
\vdots & \vdots \\
x_{2n} \leftrightarrow (p_{2n} \leftrightarrow x_{2n+1}) & y_{2n} \leftrightarrow (p_{2n} \leftrightarrow p_{2n+1}) \\
x_{2n+1} \leftrightarrow (p_{2n+1} \leftrightarrow y_1) &
\end{array}$$

Note that the table of U_n is contained as a subtable in the above if any of the following holds:

1. $x_{n+1} = y_1$ and $y_n = p_n$, or
2. $x_{n+2} = x_1$ and $y_{n+2} = y_1$, and where z_{n+1+i} is mapped to z_i in the table for U_n, where $1 \leq i \leq n$ and z is either of x, y, or p.

In particular, when considering the table for $(\neg U_{2n+1}, y_1)$ the above equations come out as:

1. $x_{n+1} = 1$ and $y_{n+1} = 1$, or
2. $x_{n+2} = 0$ and $y_{n+2} = 1$.

Suppose now we extend $(\neg U_{2n+1}, y_1)$ with a consistent (non-refutable) Δ which asserts subformulas A_i, where $i \geq n + 1$, such that case 1 is true. Then, the hardness of $(\neg U_{2n+1}, y_1, \Delta)$ is at least that of $(\neg U_n, y_1)$, since because Δ is consistent and does not involve formulas in the upper half of the table, any refutation must be based on the upper half of the formulas, i.e. the refutation is that of $(\neg U_n, y_1)$.

A similar argument can be made for case 2.

We will now show how for any subterm A of U_{2n+1}, we can extend $\Gamma = (\neg U_{2n+1}, y_1, A)$ with a Δ as above, thereby proving that $h(\neg U_{2n+1}, y_1, A) \geq h(\neg U_n, y_1)$.

Suppose $i > n+1$. Then we add x_{n+1} and y_{n+1} to Γ. Because of x_{n+1}, y_{n+1}, and y_1 being true, it follows that any set of positive or negative formulas, consistent with the rows in the table, that occur in the lower part of the table, i.e. the index of each formula is greater than $n+1$, which includes A, and which satisfies $p_{n+1} = x_{n+2} = y_{n+2}$ will give us case 1. Clearly, such a set exists.

Suppose $i < n+1$. Then we add $\neg x_{n+2}$ and y_{n+2} to Γ. Because of y_{n+2} being true, it follows that any set of positive or negative formulas, consistent with the rows in the table, that occur in the lower part of the table, i.e. the index of each variable is greater than $n+1$, which includes A, and which satisfies $y_{n+1} = p_{n+1}$ will give us case 2. Clearly, such a set exists.

Finally, suppose $i = n+1$. If x_{n+1} is true, we can add Δ such that Δ is consistent, and such that case 1 applies. Similar arguments can be made for when x_{n+1} is false, and for y_{n+1} being either true or false.

If p_{n+1} is true, we can add Δ such that Δ is consistent, contains x_{n+1}, and such that case 1 applies. If p_{n+1} is false, we can add Δ such that Δ is consistent, contains $\neg x_{n+2}$, and such that case 2 applies.

Hence, it now follows that for each subterm A of U_{2n+1}, $h(\neg U_{2n+1}, y_1, A) \geq h(\neg U_n, y_1)$. By Lemma 2.3.5, for some A, $h(\neg U_{2n+1}, y_1) > h(\neg U_{2n+1}, y_1, A) \geq h(\neg U_n, y_1)$. □

Also, for some 0-degree proofs the lower bound of the derivation length is proportional to the size of the domains of the formulas. This is a well-known phenomenon well-known in the context of finite domain and temporal constraint satisfaction problems (see for example [DMP91, JMSY94]). In the following we give axioms, that, combined with the transitivity of $<$, will improve the derivation lengths for a class of arithmetic problems.

$$\frac{}{\neg x \leq y \leftrightarrow x > y}\ (\mathtt{A}_{1<}) \qquad \frac{}{\neg x < x}\ (\mathtt{A}_{2<}) \qquad \frac{x < y \quad y < z}{x < z}\ (\mathtt{T}_<)$$

Proposition 2.3.9 *For each n, there exists Δ and A such that any 0-degree derivation of A from Δ using $\vdash$ has kn or more inferences, where k is the number of variables in Δ, but for which there is a 0-degree derivation using $\mathtt{A}_<$ of length k.*

Proof: For simplicity, we assume $k = 2$. Let $\Delta = y \leq x, x < y, x$ **in** d, y **in** d, $A = \bot$, and n be the size of d. Clearly, $\Delta \models A$. By inspection, it follows that the only elimination rules that apply for Δ are $\mathtt{E}_{1\leq}$ and $\mathtt{E}_{2\leq}$. Hence, from Δ we can derive x **in** $d \setminus \mathbf{max}(d)$ and y **in** $d \setminus \mathbf{max}(d)$. Again, by inspection, the only rules that apply are the elimination rules for $\leq$. Thereby, it follows that it takes $2n$ applications of simples rules until $\perp$ is derived. However, by the axiom added above, we immediately derive the inconsistency from $y \leq x$, by applying $\mathtt{A}_{<}$ and $\mathtt{E}_{1\perp}$ once. $\square$

The above argument can easily be generalized to larger k using the transitivity of $<$.

2.4 Saturation

In the following we define algorithms for saturating a set of formulas to a certain degree i.

2.4.1 Triplets

Let A be a formula in FDC. Then Σ_A is the set of all subformulas and subterms of A.

Let t **in** Σ_A. Then, we define ν as:

$$\nu(t) = \begin{cases} t & \text{if } t \text{ is a propositional or a domain variable} \\ b_t & \text{where } b_t \text{ is a new propositional variable and } t \text{ is a formula} \\ x_t & \text{where } x_t \text{ is a new domain variable and } t \text{ is a term} \end{cases}$$

Hence, given $t \in \Sigma_A$, the triplet corresponding to t is defined by τ as:

$$\tau(t) = \begin{cases} \langle \nu(t), \nu(t_1) \cdot \nu(t_2) \rangle & \text{if } t \equiv t_1 \cdot t_2, \cdot \in \{\wedge, \vee, \rightarrow, =, \leq, +, *\} \\ \langle \nu(t), \neg\nu(t') \rangle & \text{if } t \equiv \neg t' \end{cases}$$

A triplet T *contains* a variable x if x occurs in T.

2.4.2 Tables of simple rules

Each set of simple rules is coded by a corresponding table (Table 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8), and 2.9, such that input to the table is a set of constraints constituting the assumptions of some rules, and output from the table is the set of constraints being the conclusions of those rules. Each triplet T thereby matches exactly one table.

Hence, evaluating a triplet T in Δ results in some constraints that are added to Δ. If a constraint x **in** d is added to Δ such that for no x **in** $d' \in \Delta$, $d' \subseteq d$, we say that x is *pruned*. Furthermore, in some cases new equality constraints $x \cdot y$ are generated for which $\tau(x \cdot y)$ is added to the set of triplets. For example, when evaluating $\langle x, x_1 \wedge x_2 \rangle$ in $\Delta = \{x_1 \textbf{ in } \{1\}\}$, the triplet $\tau(x = x_2)$ is generated, and ν is extended to include $x = x_2$.

Also, not shown here, are the triplets for substitution.

$T = \langle x, x_1 \wedge x_2 \rangle$	T_Δ
x_1 **in** $\{0\} \in \Delta$	x **in** $\{0\}$
x_2 **in** $\{0\} \in \Delta$	x **in** $\{0\}$
x_1, x_2 **in** $\{1\} \in \Delta$	x **in** $\{1\}$
x **in** $\{1\} \in \Delta$	x_1, x_2 **in** $\{1\}$
x **in** $\{0\} \in \Delta$	$\nu(x_1 \neq x_2)$ **in** $\{1\}$
x_1 **in** $\{1\} \in \Delta$	$\nu(x = x_2)$ **in** $\{1\}$
x_2 **in** $\{1\} \in \Delta$	$\nu(x = x_1)$ **in** $\{1\}$
$\nu(x_1 \neq x_2)$ **in** $\{1\} \in \Delta$	x **in** $\{0\}$
$\nu(x_1 = x_2)$ **in** $\{1\} \in \Delta$	$\nu(x = x_1), \nu(x = x_2)$ **in** $\{1\}$

Table 2.1: Evaluating conjunctions

$T = \langle x, x_1 \vee x_2 \rangle$	T_Δ
x_1 **in** $\{1\} \in \Delta$	x **in** $\{1\}$
x_2 **in** $\{1\} \in \Delta$	x **in** $\{1\}$
x_1, x_2 **in** $\{0\} \in \Delta$	x **in** $\{0\}$
x **in** $\{0\} \in \Delta$	x_1, x_2 **in** $\{0\}$
x **in** $\{1\} \in \Delta$	$\nu(x_1 \neq x_2)$ **in** $\{1\}$
x_1 **in** $\{0\} \in \Delta$	$\nu(x = x_2)$ **in** $\{1\}$
x_2 **in** $\{0\} \in \Delta$	$\nu(x = x_1)$ **in** $\{1\}$
$\nu(x_1 \neq x_2)$ **in** $\{1\} \in \Delta$	x **in** $\{1\}$
$\nu(x_1 = x_2)$ **in** $\{1\} \in \Delta$	$\nu(x = x_1), \nu(x = x_2)$ **in** $\{1\}$

Table 2.2: Evaluating disjunctions

2.4.3 Saturation algorithms

We will in the following define algorithms which saturates Δ to a given level i. Suppose we are given a set of formulas $\mathcal{C}$ in FDC. Then, $\Delta_\mathcal{C}$ is the following set of domain constraints:

$$\Delta_\mathcal{C} = \{\nu(A) \textbf{ in } \{1\} \colon A \in \mathcal{C}\} \cup \{x \textbf{ in } d \colon x \text{ a domain variable in } \mathcal{C} \text{ with domain } d\}$$

In the following, assume that Q is initialized by $Q_\mathcal{C}$, where $Q_\mathcal{C} = \{\nu(A) \colon A \in \mathcal{C}\} \cup \{x \colon x \text{ a domain variable in } \mathcal{C}\}$, Δ by $\Delta_\mathcal{C}$, and L by $L_\mathcal{C}$, where $L_\mathcal{C} = \{\nu(t) \colon t \in \Sigma_\mathcal{C}\}$. Furthermore, suppose that for all compound formulas in $\Sigma_\mathcal{C}$, the corresponding triplets are defined.

0-saturation

0-saturation is thus defined as an algorithm for propagating applications of simple rules.

We use a queue of triplets, where Q is initialized by $\{T \colon T \text{ contains } \nu(A) \wedge A \in \mathcal{C}\} \cup \{T \colon T \text{ contains } x, \text{ a domain variable in } \mathcal{C}\}$:
Algorithm 2.4.3:

$T = \langle x, x_1 \to x_2 \rangle$	T_Δ
x_1 **in** $\{0\} \in \Delta$	x **in** $\{1\}$
x_2 **in** $\{1\} \in \Delta$	x **in** $\{1\}$
x_1 **in** $\{1\}, x_2$ **in** $\{0\} \in \Delta$	x **in** $\{0\}$
x **in** $\{0\} \in \Delta$	x_1 **in** $\{1\}, x_2$ **in** $\{0\}$
x **in** $\{1\} \in \Delta$	$\nu(x_1 = x_2)$ **in** $\{1\}$
x_1 **in** $\{1\} \in \Delta$	$\nu(x = x_2)$ **in** $\{1\}$
x_2 **in** $\{0\} \in \Delta$	$\nu(x \neq x_1)$ **in** $\{1\}$
$\nu(x_1 = x_2)$ **in** $\{1\} \in \Delta$	x **in** $\{1\}$
$\nu(x_1 \neq x_2)$ **in** $\{1\} \in \Delta$	$\nu(x \neq x_1)$ **in** $\{1\}, \nu(x = x_2)$ **in** $\{1\}$

Table 2.3: Evaluating implications

$T = \langle x, x_1 \leftrightarrow x_2 \rangle$	T_Δ
x **in** $\{1\} \in \Delta$	$\nu(x_1 = x_2)$ **in** $\{1\}$
x **in** $\{0\} \in \Delta$	$\nu(x_1 \neq x_2)$ **in** $\{1\}$
x_1 **in** $\{1\} \in \Delta$	$\nu(x = x_2)$ **in** $\{1\}$
x_1 **in** $\{0\} \in \Delta$	$\nu(x = x_2)$ **in** $\{0\}$
x_2 **in** $\{1\} \in \Delta$	$\nu(x = x_1)$ **in** $\{1\}$
x_2 **in** $\{0\} \in \Delta$	$\nu(x = x_1)$ **in** $\{0\}$
$\nu(x_1 = x_2)$ **in** $\{1\} \in \Delta$	x **in** $\{1\}$
$\nu(x_1 \neq x_2)$ **in** $\{1\} \in \Delta$	x **in** $\{0\}$

Table 2.4: Evaluating equivalences

```
sat(Δ, Q, L, 0) →
{
  while Q not empty do {
    set Q = Q \ {T};
    let Γ = T_Δ \ Δ;
    if Γ = ⊥ return ⊥;
    set Δ = Δ ∪ Γ;
    set Q = Q ∪ {S: S contains x ∧ ∃d.x in d ∈ Γ};
  }
  return Δ;
}
```

The algorithm for 0-saturation is a version of Waltz filtering algorithm [Wal75], and the complexity follows thereof.

Complexity 2.4.1 *Δ can be 0-saturated in $O(m^3ne)$ time, where m is the maximum size of any domain d that occurs in Δ, n is the number of variables, and e the maximum number of triplets that constrain some particular variable.*

$T = \langle x, \neg x' \rangle$	T_Δ
	$\nu(x \neq x')$ **in** $\{1\}$

Table 2.5: Evaluating negations

$T = \langle x, x_1 + x_2 \rangle$	T_Δ
x_1 **in** d_1, x_2 **in** $d_2 \in \Delta$	x **in** $d_1 + d_2$
x_2 **in** d_2, x **in** $d \in \Delta$	x_1 **in** $d - d_2$
x_1 **in** d_1, x **in** $d \in \Delta$	x_2 **in** $d - d_1$

$T = \langle x, x_1 * x_2 \rangle$	T_Δ
x_1 **in** d_1, x_2 **in** $d_2 \in \Delta$	x **in** $d_1 * d_2$
x_2 **in** d_2, x **in** $d \in \Delta$	x_1 **in** $\lceil d/d_2 \rceil \cap \lfloor d/d_2 \rfloor$
x_1 **in** d_1, x **in** $d \in \Delta$	x_2 **in** $\lceil d/d_1 \rceil \cap \lfloor d/d_1 \rfloor$

Table 2.6: Evaluating the arithmetic operators

i-saturation

i-saturation is thus based on Dilemma (Dd):

Algorithm 2.4.3:

```
sat(Δ, Q, L, i) i > 0 →
{
  while L not empty do {
    set L = L \ {x}, where x in {n_1, ..., n_k} ∈ Δ;
    let Γ_1 = sat(Δ ∪ {x in {n_1}}, Q, L, i − 1), ...,
        Γ_k = sat(Δ ∪ {x in {n_k}}, Q, L, i − 1), and Γ = Γ_1 ∩ ··· ∩ Γ_k;
    if Γ ⊆ Δ continue;
    if Γ = ⊥ return ⊥;
    return sat(Δ ∪ Γ, Q, L_C, i);
  }
  return Δ;
}
```

Complexity 2.4.2 *Δ can be i-saturated in $O(m^{2i+3}n^{2i+1}e)$ time, where m is the maximum size of any domain d that occurs in Δ, n is the number of variables, and e the maximum number of triplets that constrain some variable.*

Proof: By induction on i. The base case follows from the complexity of 0-saturation. Thus, let $i = j+1$. For each dequeueing from L at most m recursive calls are made, each of complexity $O(m^{2j+3}n^{2j+1}e)$. The size of L is bounded by n, hence, one iteration through L can be done in order $O(nmm^{2j+3}n^{2j+1}e) = O(m^{2j+3+1}n^{2j+1+1}e)$. Each time the recursive calls generates a pruning, the procedure is restarted at i, with $L = L_{\mathcal{C}}$. There can be atmost $n(m-1)$ prunings, hence, there can be atmost $n(m-1)$ restarts, each $O(m^{2j+3+1}n^{2j+1+1}e)$ in time. Thereby, i-saturation is $O(nmm^{2j+3+1}n^{2j+1+1}e) = O(m^{2j+2+3}n^{2j+2+1}e)$ in time, i.e. $O(m^{2i+3}n^{2i+1}e)$ in time. □

$T = \langle x, x_1 = x_2 \rangle$	T_Δ
x_1 **in** $\{n\}, x_2$ **in** $\{n\} \in \Delta$	x **in** $\{1\}$
x **in** $\{1\}, x_2$ **in** $d_2 \in \Delta$	x_1 **in** d_2
x **in** $\{1\}, x_1$ **in** $d_1 \in \Delta$	x_2 **in** d_1
x_1 **in** d_1, x_2 **in** $d_2 \in \Delta$ and $d_1 \cap d_2 = \emptyset$	x **in** $\{0\}$
x **in** $\{0\}, x_1$ **in** d, x_2 **in** $\{n\} \in \Delta$	x_1 **in** $d \setminus \{n\}$
x **in** $\{0\}, x_1$ **in** $\{n\}, x_2$ **in** $d \in \Delta$	x_2 **in** $d \setminus \{n\}$

$T = \langle x, x_1 \leq x_2 \rangle$	T_Δ
x_1 **in** d_1, x_2 **in** $d_2 \in \Delta$ and $\mathbf{max}(d_1) \leq \mathbf{min}(d_2)$	$x \in \{1\}$
x_1 **in** d_1, x_2 **in** $d_2 \in \Delta$ and $\mathbf{min}(d_1) > \mathbf{max}(d_2)$	$x \in \{0\}$
x **in** $\{1\}, x_1$ **in** d_1, x_2 **in** $d_2 \in \Delta$	x_1, x_2 **in** $\{\mathbf{min}(d_1), \ldots, \mathbf{max}(d_2)\}$

Table 2.7: Evaluating equalities

Δ	T_Δ
x **in** $\emptyset \in \Delta$	$\perp$
x **in** d_1, x **in** $d_2 \in \Delta$	x **in** $d_1 \cap d_2$

Table 2.8: Evaluating domain constraints

2.5 Related and Future Work

Our proof system goes beyond resolution and Davis-Putnam-like procedures [Rob65, DP60] by combining propagation of arithmetic constraints [Mac88, DHS+88] with the propagation of propositional constraints. Dilemma lets us use speculative work constructively by asserting formulas shared between disjunctive branches, somewhat like the splitting rule of Davis-Putnam [DP60, US94].

Boolean formulas over linear constraints have been investigated before. Conjunctions of linear constraints are straightforward, but disjunctions either lead to combinatorial explosion or can only be used passively unless treated specially. Beringer and De Backer use a combination of bound propagation, Balas' linear relaxation, and the Simplex method to solve disjunctive formulas over reals and integers [BdB93].

Boolean formulas over linear constraints have been studied in the OR community [Wil78]. A system for translating such formulas into 0-1 mixed integer programs was described by Mitra et al. [MLMH94].

Withing the constraint logic programming framework, different approaches for solving Boolean formulas over linear constraints have been investigated, for example the CLAIRE-D language by Beringer and De Backer [dBB93]. The usual approach is to link the truth value of the linear constraint to a Boolean variable, and then to use Boolean propagation techniques [DC93a, Rau94] to handle the Boolean constraints. Constructive disjunction [HSD92a, JS93], a reactive variant of Dilemma, was introduced in the cc(FD) language in order to exploit disjunctive constraints.

Interval arithmetic have been extensively studied as an alternative to domain

Δ	T_Δ
$\nu(x < y)$ **in** $\{1\} \in \Delta$	$\nu(x \geq y)$ **in** $\{0\}$
$\nu(x \leq y)$ **in** $\{1\} \in \Delta$	$\nu(x < y)$ **in** $\{0\}$
$\nu(x = y)$ **in** $\{1\} \in \Delta$	$\nu(x \neq y)$ **in** $\{0\}$
$\nu(x \neq y)$ **in** $\{1\} \in \Delta$	$\nu(x = y)$ **in** $\{0\}$
$\nu(x = y + c)$ **in** $\{1\} \in \Delta, c \neq 0$	$\nu(x \neq y)$ **in** $\{1\}$
$\nu(x = y + c)$ **in** $\{1\} \in \Delta, c > 0$	$\nu(x < y)$ **in** $\{1\}$

Table 2.9: Evaluating the axioms

reasoning. The BNR-Prolog language, for example, uses a narrowing algorithm to solve constraints over reals, integers and Booleans [OV93].

Hard optimization problems [GJ79] are typically solved with techniques such as integer/linear programming [Sch87] or local satisfaction, hill-climbing techniques [SLM92]. These techniques are tailored towards computing a particular solution rather than verifying all solutions. We do believe that our proof system combined with constrained, branch-and-bound driven search would produce an efficient optimizer but this is still to be shown.

The saturation algorithms are being deployed in a commercial system for verification and modeling (Prover, Logikkonsult NP AB), and currently we are investigating the benefits of the arithmetic rules.

In the near future, we also plan to extend the system with more rules. One set of rules will be for propagation-based temporal reasoning, so that many-worlds models can be analyzed. Another set of rules will be added so that numerical methods for interval propagation and equation solving can be used as well [Cle87, OV93, BMvH94]. This lets us model and analyze analog devices, as well as apply powerful methods for root-finding from numerical analysis.

Chapter 3

The Evaluation of NP(FD)

3.1 Introduction

This chapter summarizes results from running benchmark formulae under Logikkonsult's Prover version 3.2.25 as well as under the NP(FD) prototype.

3.1.1 The Prover proof system

Prover is a finely tuned implementation of a proof system similar to NP(FD) for a language containing arithmetic constraints and propositional formulae; thus Prover does not provide for domain constraints or domain reasoning. Briefly, the differences between the proof systems of Prover and NP(FD) are the following:

- No domain constraint rule.
- The elimination and introduction rules for $*$ and $+$ are replaced by rules for equations such as $x + y = z$ which reduce to assignments if two of the unknowns become known.
- No elimination rules for $\neq$ and $\leq$.
- The introduction rules for $=$ and $\leq$ only apply if all variables in the antecedent have known values.
- The implementation maintains the transitive closure of equalities for all variables and their negations. Equalities between integer variables are used by the saturation algorithm. This goes beyond NP(FD), which only maintains the transitive closure of equalities for propositional variables and their negations. Equalities between integer variables are not used by saturation in NP(FD).

Furthermore, Prover adds redundant triplets to the formula to achieve stronger arithmetic reasoning. The quotient, modulo, min and max operators are also supported.

3.1.2 The NP(FD) prototype

The NP(FD) prototype implements the NP(FD) proof system with minor deviations, which can be summarized as:

- The introduction rules for $=$, $\neq$ and $\leq$ only apply if the (dis)equality being introduced occurs explicitly in the formula.
- A number of variants of the dilemma rule have been implemented; see 3.2.1.
- The arithmetic has been extended with support for quotient, modulo, min and max.

Implementation outline

NP(FD) is an amalgamation of a finite domain constraint solver [ea95] and a Boolean solver based on the Stålmarck method. The finite domain solver handles the arithmetic using domain and interval reasoning. Its constraint store consists of variables ranging over the integers and their associated domains. The constraint store of the Boolean solver consists of propositional variables and equivalences between variables, propositional constants, and their relations.

The two solvers interact as coroutines as needed. By combining them, we can handle propositional formulae over variables and finite domain constraints. This is done by transforming such formulae to a conjunction of a single Boolean constraint and a number of reified finite domain constraints. For example, the formula

$$x < y - 1 \vee x > y + 1 \vee (q \wedge r)$$

is flattened into the conjunction

$$\begin{array}{l}(t_1 \vee t_2 \vee (q \wedge r)) \wedge \\ (x < y - 1 \Leftrightarrow t_1) \wedge \\ (x > y + 1 \Leftrightarrow t_2)\end{array}$$

where t_1 and t_2 form links between the finite domain constraints and the Boolean constraint.

The prototype also performs simple normalization and common subexpression elimination on the arithmetic formulae. For example, all arithmetical relations are normalized to $=$ and $\leq$ and their negations. Surrogate constraints are generated if certain pairs of normalized subformulae occur:

subformulae	surrogate
$x = y, x \leq y$	$x = y \rightarrow x \leq y$
$y \leq x, x \leq y$	$y \leq x \vee x \leq y$

3.2 Performance evaluation

The purpose of the benchmarking reported here is to gauge the power of extending the Stålmarck method with domain reasoning, as well as the execution overhead involved. The comparison is somewhat biased, as Prover is a finely tuned implementation written in C whereas NP(FD) is an experimental prototype written in Prolog. Prover was run on a HP 700. NP(FD) was run on a SparcCenter 2000.

Each formula was subjected to 1-saturation followed by a search for a particular model. For Prover, this means searching for a partial model where the arithmetic constraints are not guaranteed to hold, leaving integer variables uninstantiated. For NP(FD), the search includes searching for particular values for all integer variables that have bounded domains.

The performance data is presented as tables containing a single column for Prover, and four columns for different strategies for the extended dilemma rule used in 1-saturation in NP(FD).

3.2.1 Strategies for the extended dilemma rule

The dilemma rule is applied on all propositional variables and on such finite domain variables for which the domain is declared explicitly in the formula.

If x is a propositional variable, the dilemma rule is applied once, computing stores for the assumptions $x = 0$ and $x = 1$ respectively. If x is a finite domain variable, the extended dilemma rule is applied one or more times depending on the chosen strategy. The following strategies were tried, where $\{i_1, \ldots, i_n\}$ is the domain of x:

none: The dilemma rule is not used.

bisect: The dilemma rule is applied once, computing stores for the assumptions $x \leq m$ and $x > m$ respectively, where $m = (i_1 + i_n)2/$.

binary: The dilemma rule is applied n times, each time computing stores for the assumptions $x = i_j$ and $x \neq i_j$ respectively, $1 \leq j \leq n$.

nary: The dilemma rule is applied once, computing stores for the assumptions $x = i_1, \ldots, x = i_n$ respectively.

3.2.2 Search strategies

Prover used a static variable order, ordering propositional variables (including variables reflecting arithmetic constraints) by decreasing degree (number of occurrences). Under some circumstances, Prover can enumerate variables corresponding to non-atomic propositional formulae also, but this feature was not used in our experiments.

NP(FD) was used with a static and a dynamic variable order. The static variable order was the same as above. For propositional variables, the dynamic order chose at each step the variable with the greatest degree. For finite domain variables, the the dynamic order chose at each step the variable with the smallest domain, breaking ties by choosing the one with the greatest degree. The search may leave finite domain variables with unbounded domains uninstantiated.

False before true was used as value order for propositional variables. Values for finite domain variables were enumerated in ascending order.

3.2.3 Performance parameters

The following data is extracted from each run. Internally generated temporary variables that correspond to subexpressions are not included in the statistics. Two tables were generated from each run corresponding to the saturation and search phases.

Saturation data:

Msec: The time in milliseconds to perform 1-saturation in the two systems.

Iterations: The number of iterations needed in 1-saturation.

Steps: The number of times a triple or a primitive finite domain constraint was resumed during 1-saturation.

Constraints: The total number of triples or primitive finite domain constraints created.

Constraints Left: The total number of triples or primitive finite domain constraints not yet entailed after 1-saturation.

V/C Bindings: The total number of variables bound to constants after 1-saturation.

V/V Bindings: The total number of variables bound to other variables or to their negations after 1-saturation.

Singleton Classes: The total number of variables that form singleton equivalence classes after 1-saturation.

Non-Singleton Classes: The total number of variables that form non-singleton equivalence classes after 1-saturation.

Infinite Domains: For NP(FD), the number of variables that still have an infinite domain after 1-saturation.

Finite Domain Elements: For NP(FD), the total domain size of variables that have a finite domain after 1-saturation. Only finite domain variables that were not bound to constants are considered.

Prunings: This number gives a measure of the amount of domain reduction achieved by 1-saturation. Only finite domain variables that were not bound to constants are considered.

For Prover, this is the total number of variable-constant disequations found. For NP(FD), given as a pair (b, e) where b is the total number of upper or lower bounds found by for variables that were not bounded above or below, and e is the total number of domain elements that were excluded.

Search data:

Msec: The time in milliseconds to search for a particular model.

Assumptions The number of assumptions on variables done in a search branch leading to a model. For NP(FD), given as a pair $b + f$ where b is the number of assumptions on propositional variables and f is the number of assumptions on finite domain variables.

Backtracks The number of backtracks during search for a particular model. For NP(FD), given as a pair $b + f$ where b is the number of backtracks during assignment of propositional variables and f is the number of backtracks during assignment of finite domain variables.

3.3 Benchmark Sets

Two types of formulae were used. The first type of formulae are STERNOL formulae, modelling different features of railway stations. Such formulae exhibit shallow arithmetic and rich logic, and are typical for the industrial environment in which Prover is being used. The STERNOL formulae are identified by a trailing "H" of "A" in their name. Their integer variables range over $[0, 10500]$ except for variables that have an explicit domain in the formula.

The second type of formulae are combinatorial puzzles, commonly used as benchmarks for finite domain constraint solvers. The arithmetic dominates in these benchmarks, the logic merely expressing a conjunction of constraints except in one case.

3.3.1 Benchmark: AUTOMATON H

This formula contains 278 variables.

Saturation data

Count	Prover	none	bisect	binary	nary
Msec	160	27520	28520	31900	30770
Iterations	2	2	2	2	2
Steps	172318	134975	141095	160353	152527
Constraints	2163	1154	1154	1154	1154
Constraints Left	922	698	698	698	692
V/C Bindings	54	54	54	54	54
V/V Bindings	51	47	47	47	51
Singleton Classes	122	130	130	130	122
Non-Singleton Classes	51	47	47	47	51
Infinite Domains	-	5	5	5	5
Finite Domain Elements	-	231288	231288	231288	231288
Prunings	0	(0,461956)	(0,461956)	(0,461956)	(0,461956)

Search data

Count	Prover	static	dynamic
Msec	1	150	300
Assumptions	127	117+16	111+16
Backtracks	0	0+0	0+0

3.3.2 Benchmark: FICTIVESIGNAL H

This formula contains 63 variables. Prover was able to merge two equivalence classes that NP(FD) could not, due to its more complete handling of equivalence classes of integer variables.

Saturation data

Count	Prover	none	bisect	binary	nary
Msec	34	3540	3930	4800	4550
Iterations	2	2	2	2	2
Steps	24835	15939	16697	20287	19285
Constraints	555	285	285	285	285
Constraints Left	304	230	230	230	230
V/C Bindings	5	5	5	5	5
V/V Bindings	8	7	7	7	7
Singleton Classes	44	44	44	44	44
Non-Singleton Classes	6	7	7	7	7
Infinite Domains	-	0	0	0	0
Finite Domain Elements	-	63114	63114	63114	63114
Prunings	0	(0,62994)	(0,62994)	(0,62994)	(0,62994)

Search data

Count	Prover	static	dynamic
Msec	0	80	110
Assumptions	71	56+13	54+13
Backtracks	1	1+0	1+0

3.3.3 Benchmark: LEVELCROSSING H

This formula contains 77 variables.

Saturation data

Count	Prover	none	bisect	binary	nary
Msec	26	3860	4260	4600	4560
Iterations	2	2	2	2	2
Steps	17366	8853	9633	10383	10383
Constraints	531	237	237	237	237
Constraints Left	242	185	185	185	185
V/C Bindings	10	10	10	10	10
V/V Bindings	12	12	12	12	12
Singleton Classes	44	44	44	44	44
Non-Singleton Classes	11	11	11	11	11
Infinite Domains	-	6	6	6	6
Finite Domain Elements	-	89	89	89	89
Prunings	2	(39,1)	(39,1)	(39,1)	(39,1)

Search data

Count	Prover	static	dynamic
Msec	0	50	80
Assumptions	34	31+7	30+7
Backtracks	0	0+0	0+0

3.3.4 Benchmark: LINEBLOCK H

This formula contains 24 variables.

Saturation data

Count	Prover	none	bisect	binary	nary
Msec	8	160	180	240	210
Iterations	2	2	2	2	2
Steps	1666	774	918	1268	1100
Constraints	158	59	59	59	59
Constraints Left	58	28	28	28	28
V/C Bindings	5	5	5	5	5
V/V Bindings	5	5	5	5	5
Singleton Classes	9	9	9	9	9
Non-Singleton Classes	5	5	5	5	5
Infinite Domains	-	0	0	0	0
Finite Domain Elements	-	37	37	37	37
Prunings	1	(0,62994)	(0,62994)	(0,62994)	(0,62994)

Search data

Count	Prover	static	dynamic
Msec	0	20	20
Assumptions	18	9+4	9+4
Backtracks	0	0+0	0+0

3.3.5 Benchmark: LOCKER H

This formula contains 182 variables.

Saturation data

Count	Prover	none	bisect	binary	nary
Msec	148	21540	15150	17010	16240
Iterations	3	3	2	2	2
Steps	146676	88869	62060	68984	66431
Constraints	1960	691	691	691	691
Constraints Left	1202	620	620	620	620
V/C Bindings	8	8	8	8	8
V/V Bindings	18	17	17	17	18
Singleton Classes	138	140	140	140	138
Non-Singleton Classes	18	17	17	17	18
Infinite Domains	-	1	1	1	1
Finite Domain Elements	-	483451	483451	483451	462457
Prunings	0	(0,136483)	(0,136483)	(0,136483)	(0,157477)

Search data

Count	Prover	static	dynamic
Msec	2	190	430
Assumptions	220	191+35	193+35
Backtracks	0	0+0	0+0

3.3.6 Benchmark: POINT H

This formula contains 266 variables.

Saturation data

Count	Prover	none	bisect	binary	nary
Msec	215	24450	25270	29910	29480
Iterations	3	2	2	2	2
Steps	245598	139896	144924	173532	167892
Constraints	2362	1374	1374	1374	1374
Constraints Left	1138	903	903	903	903
V/C Bindings	44	44	44	44	44
V/V Bindings	46	45	45	45	46
Singleton Classes	133	135	135	135	133
Non-Singleton Classes	43	42	42	42	43
Infinite Domains	-	2	2	2	2
Finite Domain Elements	-	157888	157888	157888	157888
Prunings	1	(1,314944)	(1,314944)	(1,314944)	(1,314944)

Search data

Count	Prover	static	dynamic
Msec	2	230	540
Assumptions	177	163+22	161+22
Backtracks	0	0+0	0+0

3.3.7 Benchmark: ROUTE H

This formula contains 556 variables.

Saturation data

Count	Prover	none	bisect	binary	nary
Msec	1518	433460	424800	299320	322560
Iterations	3	3	3	2	2
Steps	1646801	1535611	1557747	1100309	1102821
Constraints	7251	4263	4263	4263	4263
Constraints Left	4894	3760	3760	3760	3758
V/C Bindings	62	62	62	62	62
V/V Bindings	45	45	45	45	45
Singleton Classes	404	404	404	404	404
Non-Singleton Classes	45	45	45	45	45
Infinite Domains	-	3	3	3	3
Finite Domain Elements	-	1502266	1502266	1502266	1502266
Prunings	7	(0,157478)	(0,157478)	(0,157478)	(0,157478)

Search data

Count	Prover	static	dynamic
Msec	8	1420	4980
Assumptions	853	678+75	696+81
Backtracks	0	0+0	0+0

3.3.8 Benchmark: SIGNAL H

This formula contains 377 variables.

Saturation data

Count	Prover	none	bisect	binary	nary
Msec	466	55780	56350	62030	61330
Iterations	3	2	2	2	2
Steps	540494	253800	263554	298546	289178
Constraints	3384	1822	1822	1822	1822
Constraints Left	1868	1452	1452	1452	1448
V/C Bindings	18	18	18	18	18
V/V Bindings	71	70	70	70	71
Singleton Classes	244	246	246	246	244
Non-Singleton Classes	44	43	43	43	44
Infinite Domains	-	2	2	2	2
Finite Domain Elements	-	483574	483574	483574	483574
Prunings	3	(0,314964)	(0,314964)	(0,314964)	(0,314964)

Search data

Count	Prover	static	dynamic
Msec	3	420	850
Assumptions	323	247+84	240+84
Backtracks	5	4+0	7+0

3.3.9 Benchmark: TRACK H

This formula contains 137 variables.

Saturation data

Count	Prover	none	bisect	binary	nary
Msec	85	12070	12430	15650	15490
Iterations	3	2	2	2	2
Steps	139677	86487	89229	111023	107805
Constraints	1304	806	806	806	806
Constraints Left	572	490	490	490	490
V/C Bindings	31	31	31	31	31
V/V Bindings	19	19	19	19	19
Singleton Classes	68	68	68	68	68
Non-Singleton Classes	19	19	19	19	19
Infinite Domains	-	1	1	1	1
Finite Domain Elements	-	42212	42212	42212	42212
Prunings	0	(0,125956)	(0,125956)	(0,125956)	(0,125956)

Search data

Count	Prover	static	dynamic
Msec	0	110	170
Assumptions	104	68+1	69+1
Backtracks	0	0+0	0+0

3.3.10 Benchmark: DISTANTSIGNAL A

This formula contains 259 variables.

Saturation data

Count	Prover	none	bisect	binary	nary
Msec	275	5680	5880	7960	7360
Iterations	3	2	2	2	2
Steps	209032	33143	34643	44181	41706
Constraints	2395	896	896	896	896
Constraints Left	1167	412	412	412	403
V/C Bindings	81	81	81	81	81
V/V Bindings	59	59	59	59	61
Singleton Classes	67	67	67	67	65
Non-Singleton Classes	52	52	52	52	52
Infinite Domains	-	1	1	1	1
Finite Domain Elements	-	646821	646821	646821	604861
Prunings	13	(0,235858)	(0,235858)	(0,235858)	(0,277818)

Search data

Count	Prover	static	dynamic
Msec	2	140	600
Assumptions	346	84+50	87+48
Backtracks	0	0+2	0+2

3.3.11 Benchmark: ENDBLOCK A

This formula contains 53 variables.

Saturation data

Count	Prover	none	bisect	binary	nary
Msec	53	550	620	1110	950
Iterations	2	2	2	2	2
Steps	49621	2147	2277	3585	3171
Constraints	611	58	58	58	58
Constraints Left	380	55	55	55	55
V/C Bindings	21	21	21	21	21
V/V Bindings	3	3	3	3	3
Singleton Classes	26	26	26	26	26
Non-Singleton Classes	3	3	3	3	3
Infinite Domains	-	0	0	0	0
Finite Domain Elements	-	94691	94691	94691	94691
Prunings	0	(0,41996)	(0,41996)	(0,41996)	(0,41996)

Search data

Count	Prover	static	dynamic
Msec	1	30	130
Assumptions	165	23+19	22+19
Backtracks	0	0+0	0+0

3.3.12 Benchmark: HELPBLOCK A

This formula contains 224 variables.

NP(FD) was able to bind two more propositional variables than Prover.

Saturation data

Count	Prover	none	bisect	binary	nary
Msec	307	33490	34460	39600	38630
Iterations	3	3	3	3	3
Steps	369251	145168	149540	168113	162413
Constraints	2138	669	669	669	669
Constraints Left	1197	467	467	467	467
V/C Bindings	19	21	21	21	21
V/V Bindings	66	64	64	64	64
Singleton Classes	83	84	84	84	84
Non-Singleton Classes	56	55	55	55	55
Infinite Domains	-	1	1	1	1
Finite Domain Elements	-	497414	497414	497414	497414
Prunings	19504	(0,311738)	(0,311738)	(0,311738)	(0,311738)

Search data

Count	Prover	static	dynamic
Msec	2	120	490
Assumptions	371	78+45	74+44
Backtracks	0	0+0	0+0

3.3.13 Benchmark: LEVELCROSSING A

This formula contains 418 variables.

Saturation data

Count	Prover	none	bisect	binary	nary
Msec	784	208090	208870	171190	179500
Iterations	4	4	4	3	3
Steps	950895	675907	688107	575056	574326
Constraints	4116	2086	2086	2086	2086
Constraints Left	2533	1719	1719	1719	1700
V/C Bindings	24	22	22	22	24
V/V Bindings	69	69	69	69	69
Singleton Classes	256	258	258	258	256
Non-Singleton Classes	69	69	69	69	69
Infinite Domains	-	7	7	7	7
Finite Domain Elements	-	1248284	1248284	1248284	1248284
Prunings	7504	(0,232938)	(0,232938)	(0,232938)	(0,232938)

Search data

Count	Prover	static	dynamic
Msec	4	430	1550
Assumptions	439	197+81	202+80
Backtracks	0	0+2	0+2

3.3.14 Benchmark: LINEBLOCK A

This formula contains 279 variables.

NP(FD) was able to bind two more propositional variables than Prover. Prover was able to merge two equivalence classes that NP(FD) could not, probably due to its more complete handling of equivalence classes of integer variables.

Saturation data

Count	Prover	none	bisect	binary	nary
Msec	589	91560	91680	97830	97750
Iterations	3	2	2	2	2
Steps	824446	392715	396923	430563	427032
Constraints	2512	1410	1410	1410	1410
Constraints Left	1123	1111	1111	1111	1115
V/C Bindings	31	33	33	33	33
V/V Bindings	21	19	19	19	19
Singleton Classes	206	208	208	208	208
Non-Singleton Classes	21	19	19	19	19
Infinite Domains	-	2	2	2	2
Finite Domain Elements	-	945347	945347	945347	945347
Prunings	1	(0,104991)	(0,104991)	(0,104991)	(0,104991)

Search data

Count	Prover	static	dynamic
Msec	3	250	920
Assumptions	280	149+68	148+68
Backtracks	1	0+0	0+0

3.3.15 Benchmark: POINT A

This formula contains 670 variables. Prover was able to infer the value of one more propositional variable than NP(FD). This phenomenon is not understood.

Saturation data

Count	Prover	none	bisect	binary	nary
Msec	7621	2352540	1885630	1467720	999320
Iterations	4	5	4	3	2
Steps	8498548	7191858	5798903	4536674	3029488
Constraints	7470	4345	4345	4345	4345
Constraints Left	4923	3788	3788	3788	3788
V/C Bindings	19	18	18	18	18
V/V Bindings	107	107	107	107	107
Singleton Classes	455	456	456	456	456
Non-Singleton Classes	89	89	89	89	89
Infinite Domains	-	2	2	2	2
Finite Domain Elements	-	1550443	1550443	1550443	1529471
Prunings	0	(0,728963)	(0,728963)	(0,728963)	(0,749935)

Search data

Count	Prover	static	dynamic
Msec	8	?	2220
Assumptions	665	349+?	351+76
Backtracks	0	0+?	0+2

3.3.16 Benchmark: SIGNAL A

This formula contains 737 variables.

NP(FD) was able to bind two more propositional variables than Prover. Prover failed to find a partial model, whereas NP(FD) found a model in five backtracks while assigning propositional variables.

Saturation data

Count	Prover	none	bisect	binary	nary
Msec	4923	1978370	1972570	1646820	1245940
Iterations	4	5	5	4	3
Steps	5686169	6848544	6903369	5738084	4364783
Constraints	7747	5182	5182	5182	5182
Constraints Left	5162	4339	4339	4339	4322
V/C Bindings	38	39	39	39	40
V/V Bindings	81	81	81	81	81
Singleton Classes	541	541	541	541	540
Non-Singleton Classes	77	76	76	76	76
Infinite Domains	-	5	5	5	5
Finite Domain Elements	-	2034372	2034372	2034372	2034372
Prunings	25504	(0,601888)	(0,601888)	(0,601888)	(0,601888)

Search data

Count	Prover	static	dynamic
Msec	?	?	11330
Assumptions	?	375+?	370+113
Backtracks	?	2+?	4+2

3.3.17 Benchmark: TRACKCROSSING A

This formula contains 525 variables.

Prover and NP(FD) both found partial models without backtracking, but the partial model found by NP(FD) contained an inconsistent system of finite domain constraints, which led to thrashing while assigning the finite domain variables.

Saturation data

Count	Prover	none	bisect	binary	nary
Msec	1438	143610	144890	109430	105350
Iterations	4	3	3	2	2
Steps	1722415	675931	688447	525770	515714
Constraints	5596	2462	2462	2462	2462
Constraints Left	3510	2166	2166	2166	2166
V/C Bindings	0	0	0	0	0
V/V Bindings	100	100	100	100	100
Singleton Classes	325	325	325	325	325
Non-Singleton Classes	100	100	100	100	100
Infinite Domains	-	0	0	0	0
Finite Domain Elements	-	1272530	1272530	1272530	1272530
Prunings	0	(0,492870)	(0,492870)	(0,492870)	(0,492870)

Search data

Count	Prover	static	dynamic
Msec	6	?	?
Assumptions	728	226+?	219+?
Backtracks	0	0+?	0+?

NP(FD) found a partial model which turned out to contain a contradictory system of linear arithmetic constraints. Searching for a complete model resulted in thrashing. No improvement was obtained by using other heuristics during the search, such as domain splitting.

3.3.18 Benchmark: TRANSMISSION A

This formula contains 49 variables.

NP(FD) was able to merge two singleton classes that Prover could not.

Saturation data

Count	Prover	none	bisect	binary	nary
Msec	50	130	140	410	270
Iterations	2	2	2	2	2
Steps	47816	601	637	1029	911
Constraints	602	32	32	32	32
Constraints Left	362	25	25	25	25
V/C Bindings	21	21	21	21	21
V/V Bindings	3	4	4	4	4
Singleton Classes	22	20	20	20	20
Non-Singleton Classes	3	4	4	4	4
Infinite Domains	-	0	0	0	0
Finite Domain Elements	-	31685	31685	31685	31685
Prunings	0	(0,41996)	(0,41996)	(0,41996)	(0,41996)

Search data

Count	Prover	static	dynamic
Msec	1	10	90
Assumptions	160	5+16	5+16
Backtracks	0	0+0	0+0

3.3.19 Benchmark: Alpha

This benchmark is a conjunction of arithmetic constraints. The goal of this benchmark is to find an assignment of 1..26 to the letters of the alphabet that satisfies a system of equations.

NP(FD) achieved significant pruning by 1-saturation, but not enough to deduce the value of any of the problem variables. Prover and NP(FD) both found models with little or no backtracking.

Saturation data

Count	Prover	none	bisect	binary	nary
Msec	315	0	570	15780	11720
Iterations	2	1	4	6	5
Steps	299881	403	13809	367204	286094
Constraints	3424	408	408	408	408
Constraints Left	1708	408	408	408	408
V/C Bindings	0	0	0	0	0
V/V Bindings	0	0	0	0	0
Singleton Classes	26	26	26	26	26
Non-Singleton Classes	0	0	0	0	0
Infinite Domains	-	0	0	0	0
Finite Domain Elements	-	676	555	431	419
Prunings	0	(0,0)	(0,121)	(0,245)	(0,257)

Search data

Count	Prover	static	dynamic
Msec	285	190	430
Assumptions	0	0+8	0+6
Backtracks	0	0+13	0+3

3.3.20 Benchmark: Bridge

This benchmarks is a jobshop scheduling problem, where the objective is to construct a bridge in 104 days. It is expressed as a conjunction of precedence constraints (inequalities) and disjunctive resource constraints (exclusive ors of two inequalities).

NP(FD) achieved significant pruning by 1-saturation, but not enough to deduce the value of any of the problem variables.

Prover could not find a model in reasonable time whereas NP(FD) found the unique model in 21 backtracks.

Saturation data

Count	Prover	none	bisect	binary	nary
Msec	5134	670	1120	36550	35340
Iterations	2	2	2	2	2
Steps	4614324	22929	36295	1162872	645792
Constraints	19911	287	287	287	287
Constraints Left	287	287	287	287	
V/C Bindings	0	0	0	0	0
V/V Bindings	0	0	0	0	0
Singleton Classes	46	46	46	46	46
Non-Singleton Classes	0	0	0	0	0
Infinite Domains	-	0	0	0	0
Finite Domain Elements	-	2265	2265	2225	2225
Prunings	65	(0,276)	(0,276)	(0,316)	(0,316)

Search data

Count	Prover	static	dynamic
Msec	-	-	1400
Assumptions	-	15+20	15+21
Backtracks	-	21+0	21+0

3.3.21 Benchmark: Magic3

This benchmarks is the 3x3 Magic Square puzzle. The objective is to assign 1..9 to the slots of a 3x3 square so that all horizontal, vertical and main diagonal sums are equal.

NP(FD) achieved some pruning by 1-saturation, but not enough to deduce the value of any of the problem variables.

Prover and NP(FD) found models in 53 and 2 backtracks, respectively.

Saturation data

Count	Prover	none	bisect	binary	nary
Msec	21	0	20	310	260
Iterations	2	1	1	2	2
Steps	12276	80	449	9586	8928
Constraints	409	52	52	52	52
Constraints Left	204	52	52	52	52
V/C Bindings	0	0	0	0	0
V/V Bindings	0	0	0	0	0
Singleton Classes	9	9	9	9	9
Non-Singleton Classes	0	0	0	0	0
Infinite Domains	-	0	0	0	0
Finite Domain Elements	-	81	81	67	67
Prunings	0	0	0	14	14

Search data

Count	Prover	static	dynamic
Msec	22	20	60
Assumptions	12	0+4	0+4
Backtracks	53	0+2	0+2

3.3.22 Benchmark: Magic4

This benchmarks is the 4x4 Magic Square puzzle. The objective is to assign 1..16 to the slots of a 4x4 square so that all horizontal, vertical and main diagonal sums are equal.

No pruning was achieved by 1-saturation.

Prover and NP(FD) found models in 5775 and 20 backtracks, respectively.

Saturation data

Count	Prover	none	bisect	binary	nary
Msec	78	10	30	970	830
Iterations	2	1	1	2	2
Steps	69147	130	490	18650	18170
Constraints	1281	150	150	150	150
Constraints Left	640	150	150	150	150
V/C Bindings	0	0	0	0	0
V/V Bindings	0	0	0	0	0
Singleton Classes	16	16	16	16	16
Non-Singleton Classes	0	0	0	0	0
Infinite Domains	-	0	0	0	0
Finite Domain Elements	-	256	256	256	256
Prunings	0	0	0	0	0

Search data

Count	Prover	static	dynamic
Msec	489	190	240
Assumptions	46	0+14	0+11
Backtracks	5775	0+60	0+20

3.3.23 Benchmark: Partit24

The objective of this benchmark is to partition the numbers 1..24 into two sets A and B such that:

- $|A| = |B|$
- $\sum_{x \in A} x = \sum_{x \in B} x$
- $\sum_{x \in A} x^2 = \sum_{x \in B} x^2$

NP(FD) achieved significant pruning by 1-saturation, but not enough to deduce the value of any of the problem variables.

Prover could not find a model in reasonable time whereas NP(FD) found the unique model in 1334 backtracks using a static variable order and in 113 backtracks using a dynamic order.

Saturation data

Count	Prover	none	bisect	binary	nary
Msec	253	10	500	7610	5260
Iterations	2	1	3	7	5
Steps	236483	2878	14422	231769	163564
Constraints	3021	371	371	371	371
Constraints Left	1523	371	371	371	371
V/C Bindings	0	0	0	0	0
V/V Bindings	2	2	2	2	2
Singleton Classes	64	64	64	64	64
Non-Singleton Classes	2	2	2	2	2
Infinite Domains	-	0	0	0	0
Finite Domain Elements	-	20814	20814	20814	20814
Prunings	0	(88,0)	(88,159)	(88,596)	(88,708)

Search data

Count	Prover	static	dynamic
Msec	-	8220	980
Assumptions	-	0+16	0+29
Backtracks	-	0+1334	0+113

3.3.24 Benchmark: Send

This benchmark is a conjunction of arithmetic constraints. The goal of this benchmark is to find an assignment of the digits 0..9 to some letters satisfying the equation SEND+MORE=MONEY.

NP(FD) completely solved this problem by the the stronger variants of 1-saturation, and partially solved it by the weaker variants.

Prover found a model in 19276 backtracks. 1-saturation did not help.

Saturation data

Count	Prover	none	bisect	binary	nary
Msec	21	0	30	40	20
Iterations	2	1	2	2	2
Steps	11639	326	850	1588	1101
Constraints	390	49	49	49	49
Constraints Left	194	25	27	0	0
V/C Bindings	0	3	3	8	8
V/V Bindings	0	0	0	0	0
Singleton Classes	8	5	5	0	0
Non-Singleton Classes	0	0	0	0	0
Infinite Domains	-	0	0	0	0
Finite Domain Elements	-	35	35	0	0
Prunings	2	0	0	0	0

Search data

Count	Prover	static	dynamic
Msec	1218	0	0
Assumptions	31	0+0	0+0
Backtracks	19276	0+0	0+0

3.3.25 Benchmark: Zebra

This benchmarks is known as the Zebra puzzle. The goal is to find five permutations of the numbers 1..5 satisfying certain constraints.

1-saturation completely solved this problem. This is not too surprising, as it is well known that the Zebra problem can be solved by making the problem path consistent.

Saturation data

Count	Prover	none	bisect	binary	nary
Msec	16	10	40	30	20
Iterations	4	2	3	2	2
Steps	5469	137	590	450	388
Constraints	511	62	62	62	62
Constraints Left	0	49	0	0	0
V/C Bindings	25	3	25	25	25
V/V Bindings	0	8	0	0	0
Singleton Classes	0	6	0	0	0
Non-Singleton Classes	0	8	0	0	0
Infinite Domains	-	0	0	0	0
Finite Domain Elements	-	83	0	0	0
Prunings	100	(0,1)	(0,0)	(0,0)	(0,0)

Search data

Count	Prover	static	dynamic
Msec	0	0	0
Assumptions	0	0+0	0+0
Backtracks	0	0+0	0+0

3.4 Conclusions

We have presented results from running benchmark formulae under Logikkonsult's Prover version 3.2.25 as well as under the experimental NP(FD) theorem prover. A number of formulae were subjected to 1-saturation followed by search for an explicit model.

In the comparison, we have focussed on the following aspects:

equivalence classes of Boolean variables: It was hoped that the extra power brought to bear by domain reasoning would yield more variable bindings and generally fewer and larger equivalence classes than the basic Stålmarck method. This has only been observed on four cases, where NP(FD) was able to bind 1-2 variables that Prover could not. In three cases, however, NP(FD) yielded fewer variable bindings than Prover, since NP(FD) maintains equivalence classes of finite domain variables in a weaker sense than Prover does.

dilemma rule strategies: The "nary" strategy was generally the strongest one, in terms of number of variable bindings and number of prunings, although the difference in effect between the strategies was rather small.

The stronger strategies were also the more costly per iteration during saturation. However, in many cases a stronger strategy could save one or more iterations during saturation. In those cases, the stronger strategies were also the fastest ones.

domain information: The main advantage of the NP(FD) approach is the fact that domain information is available for the finite domain variables after saturation. This is a qualitative advantage rather than a quantitative one, and is likely to enable a deeper understanding of the problems being analysed.

Domain information and domain reasoning is highly relevant in problems dominated by arithmetic, especially if the domains are not too big. This is illustrated by the results of the combinatorial benchmarks.

search: We have compared a static variable order with a dynamic one when searching for models. The variable orders were based on domain size and degree (number of attached constraints).

It is well known for combinatorial problems that a dynamic variable order can be vastly more efficient than a static one; witness the results of Partit24. However, the efficiency of a dynamic variable order has not been investigated on formulae similar to the STERNOL ones.

In two STERNOL cases, a model was only found using the dynamic variable order. In other cases, no significant difference in the number of backtracks required to find a model was observed. The execution time was several times higher for the dynamic variable order, but in all cases where a model was found, the time spent in search was insignificant compared to the time spent in 1-saturation. The search depth (number of assumptions) for the first model found was slightly lower for the dynamic variable order, which may reflect the fact that the dynamic order is a more precise heuristic.

execution time: For STERNOL formulae, execution times is NP(FD) are about 200 times greater than in Prover. To some extent this is due to different implementation levels. A finely tuned implementation of NP(FD) could perhaps bring the performance gap down by an order of magnitude, but hardly by more.

Bibliography

[Bar77] J. Barwise. *Handbook of Mathematical Logic*. North Holland, 1977.

[BdB93] Henri Beringer and Bruno de Backer. Satisfiability of boolean formulas over linear constraints. In *IJCAI'93*, pages 296–301, 1993.

[BMvH94] F. Benhamou, D. McAllester, and P. van Hentenryck. clp(Intervals) revisited. In *Proc. of the International Symposium on Logic Programming*. MIT Press, 1994.

[Bra94] L. Bramstång. Formal Verification of Hardware. Master's thesis, Lund University, 1994.

[Bry86] R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transaction on computers*, 35:8:677–691, 1986.

[Car95] B. Carlson. *Compiling and Executing Finite Domain Constraints*. Uppsala theses in computing science 21, Uppsala University, May 1995. SICS Dissertation Series 18.

[Cle87] J. G. Cleary. Logical arithmetic. *Future Generation Computing Systems*, 2:2:125–149, 1987.

[D'A92] D'Agostino. Investigations into the complexity of some propositional calculi. Technical monograph, Oxford University, 1992.

[dBB93] Bruno de Backer and Henri Beringer. A clp language handling disjunctions of linear constraints. In *ICLP'93*, pages 550–563, 1993.

[DC93a] D. Diaz and P. Codognet. A Boolean Extension of clp(FD). In *Proceedings of the International Symposium on Logic Programming*. MIT Press, 1993.

[DC93b] D. Diaz and P. Codognet. A Minimal Extension of the WAM for clp(FD). In *Proceedings of the International Conference on Logic Programming*. MIT Press, 1993.

[DHS+88] M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier. The Constraint Logic Programming Language CHIP. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, 1988.

[DMP91] R. Dichter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.

[DP60] M. Davis and H. Putnam. A Computing Procedure for Quantification Theory. *Journal of the ACM*, 7:201–215, 1960.

[ea95] Mats Carlsson et al. *SICStus Prolog User's Manual.* Swedish Institute of Computer Science, June 1995. Release 3.

[GJ79] M. R. Garey and D.S. Johnson. *Computers and Intractability.* W.H. Freeman and Company, 1979.

[Har96] J. Harrison. Stålmarck's method as a hol derived rule. In *TPHOL'96*, 1996.

[HDT92] P. Van Hentenryck, Y. Deville, and C.-M. Teng. A Generic Arc Consistency Algorithm and Its Specializations. *Artificial Intelligence*, 57:291–321, 1992.

[Hen89] P. Van Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, 1989.

[HSD92a] P. Van Hentenryck, V. Saraswat, and Y. Deville. Constraint Logic Programming over Finite Domains: the Design, Implementation, and Applications of cc(FD). Technical report, Computer Science Department, Brown University, 1992.

[HSD92b] P. Van Hentenryck, H. Simonis, and M. Dincbas. Constraint satisfaction using constraint logic programming. *Artificial Intelligence*, 58:113–159, 1992.

[JM94] J. Jaffar and M. Maher. Constraint Logic Programming: A Survey. *Journal of Logic Programming*, 19/20:503–581, 1994.

[JMSY94] J. Jaffar, M. Maher, P. Stuckey, and R. Yap. Beyond finite domains. In *PPCP'94: Second Workshop on Principles and Practice of Constraint Programming*, 1994.

[JS93] J. Jourdan and T. Sola. The versatility of handling disjunctions as constraints. In *Proceedings of the Programming Language Implementation and Logic Programming Conference, LNCS 714*. Springer Verlag, 1993.

[Mac88] A. Mackworth. *Encyclopedia of AI*, chapter Constraint Satisfaction. John Wiley and Sons, 1988.

[MLMH94] G. Mitra, C. Lucas, S. Moody, and E. Hadjiconstantinou. Tools for reformulating logical forms into zero-one mixed integer programs. *European Journal of Operations Research*, 72:262–276, 1994.

[Mon88] M. Mondadori. *Classical analytical deduction.* Annali dell'Universita di Ferrara, 1988.

[OV93] W. Older and A. Vellino. Constraint arithmetic on real intervals. In *Constraint Logic Programming: Selected Research (eds. Benhamou and Colmerauer).* MIT Press, 1993.

[Pra65] D. Prawitz. *Natural Deduction.* Almqvist & Wiksell, 1965.

[Rau94] A. Rauzy. Notes on the Design of an Open Boolean Solver. In *Proceedings of the Eleventh International Conference on Logic Programming.* MIT Press, 1994.

[Rob65] J. A. Robinson. A Machine Oriented Logic based on the Resolution Principle. *Journal of ACM*, 12 (1):23–41, 1965.

[SÅ92] G. Stålmarck and O. Åkerlund. Formal Verification of Hardware and Software Systems using NP-Circuit. In *Reliability and Safety of Processes and Manufacturing Systems.* Elsevier Applied Science, 1992.

[Sch87] A. Schrijver. *Theory of integer and linear programming.* John Wiley and sons, 1987.

[SLM92] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proc. of the 10th National Conference on Artifial Intelligence.* AAAI, 1992.

[SS90] G. Stålmarck and M. Säflund. Modeling and Verifying Systems and Software in Propositional Logic. In *Safety of Computer Control Systems.* IFAC Symposia Series, 1990.

[Stå91] G. Stålmarck. Normalization proofs for full first order classical natural deduction. *Journal of Symbolic Logic*, 55:1, 1991.

[Stå94] G. Stålmarck. A system for determining propositional logic theorems by applying values and rules to triplets generated from a formula. US Patent # 5 276 897, Swedish Patent # 467 076, European Patent # 0403 454, 1994.

[Urq87] A. Urquhart. Hard examples for resolution. *Journal of the ACM*, 34:1:209–219, 1987.

[US94] T. Uribe and M. E. Stickel. Ordered binary decision diagrams and the davis-putnam procedure. In *1st International Conference on Constraints in Computational Logics.* Springer-Verlag, 1994. Lecture Notes in Computer Science, vol 845.

[Wal75] D. Waltz. *The Psychology of Computer Vision (Ed. P. Winston)*, chapter Understanding line drawings of scenes with shadows. McGraw-Hill, New York, 1975.

[Wil78] H. P. Williams. *Model Building in Mathematical Programming.* J. Wiley and sons, 1978.