Constraint Contextual Rewriting[*]

Alessandro Armando[†]
DIST, Università di Genova & LORIA-INRIA-Lorraine

Silvio Ranise[‡]
DIST, Università di Genova

Abstract

We are interested in the problem of integrating decision procedures with rewriting as in many state-of-the-art verification systems. We define *Constraint Contextual Rewriting* (CCR) as a generalization of contextual rewriting, whereby the rewriting context is processed by the available decision procedures. We show how CCR accounts for some of the most important integration schemas adopted in state-of-the-art verification systems. The rule-based presentation of CCR given in this paper contrasts the practice of describing the integration either by examples or in informal ways with high-level ideas intermixed with implementation details. Important properties (e.g. soundness) of the proposed integration schema can be formally stated and proved. Moreover, the approach is amenable of operationalization. This has allowed us to easily fast-prototype and validate the integration schemas described in this paper.

1 Introduction

We are interested in the problem of integrating decision procedures with rewriting as in many state-of-the-art verification systems such as NQTHM [4], PVS [13], Tecton [9] and STeP [7]. The key factors in the success of such systems are (a) a tight integration schema for the cooperation of decision procedures, and (b) a carefully designed integration schema between the decision procedures and rewriting. While abstract accounts of (a) can be found in the literature (see, e.g., [12, 14]), only informal descriptions often intermixed with implementation details are available for (b). This situation is exemplified in [12], where the paradigm for cooperating decision procedures is abstractly described whilst the integration schema with the simplifier is described at a much lower level of detail.

In this paper, we propose a general schema for the integration of decision procedures in formula simplification. The key idea is to generalize *contextual rewriting* (see, e.g., [15]), to allow the available decision procedures to access and manipulate the rewriting context. We call *Constraint Contextual Rewriting* (CCR) this extended form of rewriting. We show how CCR easily accounts for some of the most important integration schemas adopted in state-of-the-art verification systems. This is exemplified by encoding in our framework

[*]We wish to thank Alan Smaill and Ian Green for their very useful comments on an earlier version of this paper. We also would like to thank Olga Caprotti and Deepak Kapur for pointing us to some of the relevant works in the literature.

[†]Viale Causa 13 – 16145 Genova – Italia, `armando@dist.unige.it` & 615, rue du Jardin Botanique, BP 101 – 54602 Villers les Nancy Cedex – France, `armando@loria.fr`

[‡]Viale Causa 13 – 16145 Genova – Italia, `silvio@dist.unige.it`

both contextual rewriting and two sophisticated integration schemas: the simplifier of NQTHM [3], and the simplifier of Tecton [9]. Furthermore, we identify the set of the interface functionalities that the decision procedure must provide for the integration to be effective. The rule-based specification of CCR given in this paper contrasts the practice of describing the integration by examples or in informal ways with high-level ideas intermixed with implementation details. As a result, important properties (e.g. soundness) of the integration schemas of interest can be formally stated and discussed. Finally, the rule-based formalization is amenable of mechanization. This has allowed us to easily fast-prototype and validate the integration schemas described in this paper.

The paper is structured as follows. In Section 2, we put CCR in the context of formula simplification, we discuss the rules for CCR and constraint simplification, and give hints on the soundness of the integration schema. In Section 3, we show that contextual rewriting as well as the integration schemas of NQTHM and Tecton are all instances of CCR. In Section 4 we compare with related work, and in Section 5 we draw some final conclusions.

Formal Preliminaries. By Σ, Π (possibly subscribed) we denote collections of function and predicate symbols (with their arity), respectively. By V we denote a set of variables. $\tau(\Sigma \cup V)$ and $\tau(\Sigma)$ denote the set of *terms* and *ground terms* built on Σ and V. Furthermore, we assume that $\tau(\Sigma \cup V)$ is the smallest set containing V s.t. $f(t_1, \ldots, t_n) \in \tau(\Sigma \cup V)$ whenever $f \in \Sigma$ and $t_i \in \tau(\Sigma \cup V)$ $(i = 1, \ldots, n)$. A term $t \in \tau(\Sigma \cup V)$ may be viewed as a finite ordered tree, whose leaves are labeled by constants (i.e. function symbols of zero arity) or variables and whose internal nodes are labeled with function symbols of positive arity with out-degree equal to the arity of the label. A *position* within a term may be represented as a sequence of positive integers, describing the path from the outermost symbol to the head of the sub-term at that position. With $t|_p$ we denote the sub-term of t at position p. The term t with its sub-term $t|_p$ replaced by a term s is denoted by $t[s]_p$ and we call $t[]_p$ the *context* of s. A *substitution* is a mapping from variables to terms and it is written out as $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$, s.t. there are only finitely many x_i $(i = 1, \ldots, n)$ not mapped to themselves. If σ is a substitution and $s \in \tau(\Sigma \cup V)$, then $s\sigma$ denotes the result of applying σ to s. We also call a substitution a function $\hat{\sigma}$ from $\tau(\Sigma \cup V)$ to itself, obtained as the extension of a substitution σ from V to $\tau(\Sigma \cup V)$ in such a way that $f(t_1, \ldots, t_n)\hat{\sigma} = f(t_1\hat{\sigma}, \ldots, t_n\hat{\sigma})$, for each $f \in \Sigma$, for all terms $t_i \in \tau(\Sigma \cup V)$ $(i = 1, \ldots, n)$ and $x\hat{\sigma} = x\sigma$, for each $x \in V$. A term t *matches* a term t' if there exists a substitution σ s.t. $t'\sigma = t$.

A (Σ, Π, V)*-atom* is either an expression $q(t_1, \ldots, t_n)$ where $q \in \Pi$ and $t_i \in \tau(\Sigma \cup V)$ $(i = 1, \ldots, n)$ or an expression $t_1 = t_2$ where $t_1, t_2 \in \tau(\Sigma \cup V)$. A (Σ, Π, V)*-literal* is either a (Σ, Π, V)-atom or a negated (Σ, Π, V)-atom. The variables in a (Σ, Π, V)-atom (-literal) are understood to be universally quantified. We write (Σ, Π)-atom (-literal) instead of $(\Sigma, \Pi, \emptyset)$-atom (-literal). The set of (Σ, Π)*-expressions* is the union of $\tau(\Sigma)$ and the set of (Σ, Π)-atoms. A (Σ, Π, V)*-formula* is the smallest set containing the (Σ, Π, V)-literals and if α is a (Σ, Π, V)-formula then $\neg(\alpha)$ is a (Σ, Π, V)-formula, and if α_1, α_2 are two (Σ, Π, V)-formulae then $(\alpha_1 \diamond \alpha_2)$ is a (Σ, Π, V)-formula, where "$\diamond$" is one of the following logical connectives $\wedge$, $\vee$, $\rightarrow$ and $\leftrightarrow$. (Below, $a \neq b$ abbreviates $\neg(a = b)$.) We assume that the language contains the propositional constants *true* and *false* denoting truth and falsity, respectively. A (Σ, Π, V)*-theory* is a collection of (Σ, Π, V)-formulae.

In order to distinguish the symbols interpreted by the different modules of the combination/integration, we partition the sets of function and predicate symbols in the following sub-sets: Σ_j, Σ_c, Π_j and Π_c, s.t. $\Sigma_c \subseteq \Sigma_j$ and $\Pi_c \subseteq \Pi_j$. The subscript "j" labels the sets of function and predicate symbols of the (fragment of the) logic of the prover, where the combination/integration of decision procedures and contextual rewriting takes place. The subscript "c" labels the sets of function and predicate symbols handled by a (combination of) decision procedure(s). The assumptions $\Pi_c \subseteq \Pi_j$ and $\Sigma_c \subseteq \Sigma_j$ mean that a subset of the logic of the prover is handled by the decision procedure(s) and that contextual rewriting is able to handle the whole class of formulae in the logic. Finally, we call *constraints* the (Σ_j, Π_c)-literals.

Below, we use sets of literal with different meanings. If P is a set of literals then $\overline{P}$ is the set of the negations of the literals in P. The logical reading of a set of literals $\{l_1, \dots, l_n\}$ is $l_1 \vee \dots \vee l_n$, whereas the logical reading of its negation is $\neg l_1 \wedge \dots \wedge \neg l_n$. *Constraint stores* are denoted with sets of (Σ_j, Π_c)-atoms, such as $\{c_1, \dots, c_n\}$ whose logical reading is $c_1 \wedge \dots \wedge c_n$. The ambiguous use of brackets for both sets of literals and constraint stores is made clear by the context. Below, sometimes, a set of literal and/or a constraint store is used in a logical formula: that set should be replaced by the conjunction of each literal in it.

A *conditional rewrite rule* is an implication in which the equation in the conclusion is oriented, for which we write $(p_1, \dots, p_n \to l \Rightarrow r)$, where $l \in \tau(\Sigma_j, V)$, $r \in \tau(\Sigma_j, V')$ with $V' \subseteq V$, and for $i = 1, \dots, n$ $(n \geq 0)$ p_i is a (Σ_j, Π_j, V_i)-atom with $V_i \subseteq V$. The logical reading of a conditional rewrite rule is $(p_1 \wedge \dots \wedge p_n \to l = r)$.[1] A *conditional constraint rule* is an implication in which the conclusion belongs to the set of (Σ_j, Π_c)-literals, for which we write $(p_1, \dots, p_n \to c)$, where c is a (Σ_j, Π_c, V)-atom, and for $i = 1, \dots, n$ $(n \geq 0)$ p_i is a (Σ_j, Π_j, V_i)-atom with $V_i \subseteq V$. The logical reading of a conditional constraint rule is $(p_1 \wedge \dots \wedge p_n \to c)$.[2] In the following, we consider a set R of conditional rewrite rules and a set CR of conditional constraint rules. We assume these two sets to be fixed during the simplification of a formula.

Above, we have assumed a quantifier-free first-order language. However, this assumption is not fundamental to our approach. The crucial point is to be able to partition the set of formulae so to recognize those that can be handled by the decision procedure and those that must be handled with the help of the rewriting mechanism. With this respect even a higher-order language with a binding operator (i.e. a language containing the lambda-calculus) can be used, if a suitable form of rewriting can be defined and at least one decision procedure for a fragment of the logic is available.

In order to precisely present the functionalities of the modules as well as their integration, we introduce the notion of *contextual reduction system* (CRS). The overall specification will consist of a set of mutually inductive defined CRSs mirroring the deep interplay of the modules. This concept allows us to unambiguously and modularly specify the integration/combination of decision procedures. Formally, a contextual reduction system is a family of structures $\langle Q_i, S_i, R_i \rangle_{i \in I}$, where Q_i and S_i are sets and R_i a set of *rules of inference*, i.e. subsets of $T_i \times T_{k_1} \times \cdots \times T_{k_n}$, where T_z abbreviates $(Q_z \times S_z \times S_z)$, $k_j \in I$ for $j = 1, \dots, n$, and $n \geq 0$. If $q \in Q_i$, and $s, s' \in S_i$, then we abbreviate $\langle q, s, s' \rangle$ with $q :: s \underset{\mathsf{i}}{\to} s'$ and use the term *sequent* to refer to objects of this form. We write $s \underset{\mathsf{i}}{\to} s'$ in place of $q :: s \underset{\mathsf{i}}{\to} s'$ when q plays no significant role. The sequent $q :: s \underset{\mathsf{i}}{\to} s'$ is read as follows: the object s reduces to s' under the context q by using the functionalities offered by module i. As an example, take i to be a simplification module, s the first-order atom $P(f(f(g(a))))$ and c the following conjunction $f(f(a)) = f(a) \wedge f(g(a)) = a$, then s rewrites to $P(a)$

[1] For $n = 0$, we obtain a *rewrite rule* for which we write $(l \Rightarrow r)$, whose logical reading is $l = r$.

[2] For $n = 0$, we obtain a *constraint rule* for which we write (c), where (Σ_j, Π_c)-atom.

under the context c.[3]
As usual, inference rules will be presented schematically:

$$\frac{c_{k_1} :: s_{k_1} \xrightarrow[\mathsf{k_1}]{} s'_{k_1} \quad \cdots \quad c_{k_n} :: s_{k_n} \xrightarrow[\mathsf{k_n}]{} s'_{k_n}}{c_i :: s_i \xrightarrow[\mathsf{i}]{\mathsf{r_i}} s'_i}$$

We inductively define the notion of derivation of the sequent $q :: a_0 \xrightarrow[\mathsf{i}]{} a_{2n}$ to be a sequence $a_0, a_1, \ldots, a_{2n}$ such that $a_k, a_{k+2} \in S_i$, $q :: s_k \xrightarrow[\mathsf{i}]{} s_{k+2}$ is the conclusion of a rule, say r_{k+1}, in R_i, and a_{k+1} is a sequence of derivations of the premises of r_{k+1}, for $k = 0, \ldots, 2n-2$. We will use the notation $q :: a_0 \overset{a_1}{\xrightarrow[\mathsf{i}]{\mathsf{r_1}}} a_2 \overset{a_3}{\xrightarrow[\mathsf{i}]{\mathsf{r_3}}} \cdots \overset{a_{2n-3}}{\xrightarrow[\mathsf{i}]{\mathsf{r_{2n-3}}}} a_{2n-2} \overset{a_{2n-1}}{\xrightarrow[\mathsf{i}]{\mathsf{r_{2n-1}}}} a_{2n}$ in place of $a_0, a_1, a_2, a_3, \ldots, a_{2n-3}, a_{2n-2}, a_{2n-1}, a_{2n}$, and use the notation $q :: a \xrightarrow[\mathsf{i}]{}^{*} a'$ to indicate that there is a derivation of $q :: a \xrightarrow[\mathsf{i}]{} a'$.

A *constraint domain* is a (Σ_c, Π_c)-structure $\mathcal{D}$, where a (Σ, Π)-structure consists of a set D and an assignment of functions and relations on D to the symbols of Σ and Π (resp.) which respect the arity of the symbols. A *model* of a (Σ, Π)-theory T is a (Σ, Π)-structure under which all the formulae in T evaluate to true. A *$\mathcal{D}$-model* of a theory T is a model of T extending $\mathcal{D}$ (this requires that the signature of $\mathcal{D}$ is contained in the signature of T). We write $T \models_{\mathcal{D}} \phi$ to denote that ϕ is valid in all $\mathcal{D}$-models of T. In the sequel we assume the existence of a background (Σ_j, Π_j)-theory T such that *(i)* the models of T are $\mathcal{D}$-models, *(ii)* if $(p_1, \ldots, p_n \to l \Rightarrow r) \in R$ then $T \models_{\mathcal{D}} (p_1 \wedge \ldots \wedge p_n \to l = r)$, and *(iii)* if $(p_1, \ldots, p_n \to c) \in CR$ then $T \models_{\mathcal{D}} (p_1 \wedge \ldots \wedge p_n \to c)$.

2 Combining Rewriting and Decision Procedures

A well established approach to formula simplification amounts to rewriting the sub-expressions of the input formula using the sub-formulae surrounding the expression being rewritten as the *context* of rewriting. For instance, when the formulae to simplify are clauses the negation of all the literals in the clause (but the one in which the expression being rewritten occurs) can be used as context of rewriting.
We abstract from the language employed by postulating the existence of a function $\mathbf{Cxt}(\cdot,\cdot)$ s.t. $(\bigwedge \mathbf{Cxt}(\Phi, u) \to (s \sim t)) \to (\Phi[s]_u \leftrightarrow \Phi[t]_u)$ holds in the background theory, for every formula Φ and for every legal position u in Φ, where $\mathbf{Cxt}(\Phi, u)$ denotes a finite set of literals, $\bigwedge \mathbf{Cxt}(\Phi, u)$ denotes the conjunction of the literals in this set and "$\sim$" is a congruence relation – e.g. "$=$" and "$\leftrightarrow$". Then, the simplification activity which relies on rewriting can be modeled as follows:

$$\frac{\mathbf{Cxt}(\Phi, u) :: C^{\circ} \xrightarrow[\mathsf{cs-fe}]{} C \quad C :: s \xrightarrow[\mathsf{ccr}]{} t}{\Phi[s]_u \xrightarrow[\mathsf{simp}]{\mathsf{cl-simp}} \Phi[t]_u} \text{ if } \mathtt{cs\text{-}init\text{-}state}(C^{\circ})$$

where $\mathsf{cl\text{-}simp}$ is the rule name (standing for clause simplification), simp is one of the module defining the CRS formalizing simplification of formulae by means of constraint contextual

[3]The context and atoms of the example could be extracted out of the following clause $P(f(f(g(a)))) \vee f(f(a) \neq f(a) \vee f(g(a)) \neq a$, by focusing on the first literal and assuming the negation of the remaining ones.

rewriting, $C :: s \xrightarrow[\mathsf{ccr}]{} t$ models constraint contextual rewriting, i.e. the activity of rewriting s to t using C as the context of rewriting (a formal definition is given in Section 2.1), and $\mathbf{Cxt}(\Phi, u) :: C^{\circ} \xrightarrow[\mathsf{cs-fe}]{} C$ models the activity of extending the constraint store C° with the facts contained in the context $\mathbf{Cxt}(\Phi, u)$ (indeed, cs-fe stands for constraint store's fact extension). Finally, `cs-init-state` is a predicate which is true only for the constraint store C° equivalent to the empty set of constraints. Thus, we could read the above rule as follows: expression s at position u in the formula Φ can be replaced by expression t, provided that t is obtained by constraint contextual rewriting s in the context obtained by extending the empty context with the expressions surrounding s in the formula Φ.
For the sake of simplicity, in the rest of the paper we will assume that $\xrightarrow[\mathsf{simp}]{}$ is defined so as to formalize clause simplification. Therefore, $\mathbf{Cxt}(\Phi, u)$ can be thought of returning the conjunction of the negation of each literal in the clause Φ, but the one within which the position u occurs.

2.1 Constraint Contextual Rewriting

Given a set of conditional rewrite rules R, the (ternary) *constraint contextual rewriting* relation $\_ :: \_ \xrightarrow[\mathsf{ccr}]{} \_$ is defined by the following rules:

$$C :: p \xrightarrow[\mathsf{ccr}]{\mathsf{cs-unsat}} \mathit{true} \quad \text{if } p \text{ is a } (\Sigma_j, \Pi_j)\text{-literal and } \mathtt{cs\text{-}unsat}(C)$$

i.e. rewrite the literal p to *true* under the context C if this is not consistent (inconsistency of the constraint store is checked by `cs-unsat`),

$$\frac{\overline{p} :: C \xrightarrow[\mathsf{cs-fe}]{} C'}{C :: p \xrightarrow[\mathsf{ccr}]{\mathsf{cxt-entails}} \mathit{true}} \quad \text{if } p \text{ is a } (\Sigma_j, \Pi_c)\text{-literal and } \mathtt{cs\text{-}unsat}(C')$$

i.e. rewrite the literal p to *true* under the context C, if after adding C the negation of the literal being rewritten, we get an inconsistent constraint store,

$$\frac{C :: e \xrightarrow[\mathsf{cs-canon}]{} e'}{C :: e \xrightarrow[\mathsf{ccr}]{\mathsf{canon}} e'}$$

i.e. rewrite the expression e to its canonical form e' under the context C, if the decision procedure is able to derive a canonical form out of e and its context C,

$$\frac{C :: \{p_1\sigma, \ldots, p_n\sigma\} \xrightarrow[\mathsf{rlv}]{} \emptyset}{C :: s[l\sigma]_u \xrightarrow[\mathsf{ccr}]{\mathsf{crew}} s[r\sigma]_u} \text{ if } \ (p_1, \ldots, p_n \rightarrow l \Rightarrow r) \in R$$

i.e. rewrite the sub-term $l\sigma$ at position u in the expression s to the sub-term $r\sigma$ under the context C, if there exist a substitution σ and a conditional rewrite rule in R whose conclusion is of the form $l \Rightarrow r$ s.t. the sub-term at position u in s is $l\sigma$ and, once the hypotheses of the conditional rewrite rule have been added to the constraint store C, we get the empty constraint store (we say that we have *relieved* the hypotheses of the constraint rule).

Finally, the activity of relieving the hypotheses of a conditional rewrite rule is modeled as follows:

$$\frac{C :: p \xrightarrow[\mathsf{ccr}]{} \mathit{true}}{C :: P \cup \{p\} \xrightarrow[\mathsf{rlv}]{\mathsf{relieve}} P}$$

i.e. relive the hypothesis p under the context C, if p rewrites to *true* by constraint contextual rewriting under the context C.

2.2 Constraint Simplification

We are now in the position to make precise some of the previously introduced concepts. Mainly, we are going to precisely characterize the predicate `cs-init-state`, the test for constraint store inconsistency, the relation returning the canonical form of a given expression w.r.t. the constraint store and, finally, the relation computing the extension of the constraint store with new facts. Formally, we assume the decision procedure provides the rewriter with the following set of functionalities:

- `cs-init-state`(C) holds if and only if C is equivalent to the empty set of constraints.
- `cs-unsat`(C) only if C is unsatisfiable.
- $C :: e \xrightarrow[\mathsf{cs-canon}]{} e'$ if and only if e' is the canonical representation of the expression p w.r.t. the constraint store C. If there is no canonical representation for the input expression e then e' is e itself or a normal form computed w.r.t. the information contained in the constraint store. As an example of canonical representation of an expression w.r.t. a constraint store, consider a procedure for congruence closure of a certain equivalence relation. This procedure builds a graph encoding the equivalence classes for the relation. Once a term is presented as the input, the procedure returns the chosen witness for the equivalence class to which the term belongs.
- $P :: C \xrightarrow[\mathsf{cs-fe}]{} C'$ if and only if C' is the constraint store obtained by extending the constraint store C with the literals in P.

By assuming that the definition of $P :: C \xrightarrow[\mathsf{cs-fe}]{} C'$ relies on a constraint simplification relation $C \xrightarrow[\mathsf{cs-simp}]{} C'$,[4] we can model the activity of heuristically augmenting the constraint store with selected instances of constraints in the following way:

$$\frac{P :: C \xrightarrow[\mathsf{cs-fe}]{} C'}{C \xrightarrow[\mathsf{cs-simp}]{\mathsf{augment}} C'} \text{ if } \mathtt{oracle}(C, P)$$

where P is a set of valid constraints "guessed" by the `oracle` relation. As we will see in Section 3, different definitions of `oracle` provides us with forms of augmentation of different strength. For instance, it is possible in this way to inform the decision procedure about properties of interpreted function symbols, which otherwise the decision procedure does not know anything about.

[4]An example of constraint simplification relation is given by the variable elimination process performed on a set of inequalities C. An inequality is an atom of the form $a_1 * x_1 + \ldots + a_n * x_n \leq 0$, where the a_i's are the coefficients (e.g. real numbers) and the x_i's are distinct variables. The variable elimination process consists of cross-multiplying coefficients and adding inequalities so as to cancel out one variable at a time until no more variable can be eliminated, thus obtaining C'.

Soundness. The soundness of the simplification mechanism amounts to proving that if $\Phi \xrightarrow[\mathsf{simp}]{} \Phi'$ then $T \models_{\mathcal{D}} (\Phi \leftrightarrow \Phi')$ and can be easily inferred from the soundness of CCR. The soundness of CCR amounts to showing that if $C :: e \xrightarrow[\mathsf{ccr}]{} e'$, then $T, C \models_{\mathcal{D}} e \sim e'$ (where "$\sim$" is "$\leftrightarrow$" if e and e' are (Σ_j, Π_j)-literals and "$=$" if $e, e' \in \tau(\Sigma_j)$). The soundness of CCR can be proved under the assumption that the interface functionalities enjoy the following properties: (i) if $\mathtt{cs\text{-}init\text{-}state}(C)$ then $\models_{\mathcal{D}} C$, (ii) if $\mathtt{cs\text{-}unsat}(C)$ then $C \models_{\mathcal{D}} \mathit{false}$, (iii) if $C :: e \xrightarrow[\mathsf{cs-canon}]{} e'$ then $C \models_{\mathcal{D}} e \sim e'$, where "$\sim$" is "$\leftrightarrow$" if e and e' are (Σ_j, Π_j)-literals and "$=$" if $e, e' \in \tau(\Sigma_j)$, (iv) if $P :: C \xrightarrow[\mathsf{cs-fe}]{} C'$ then $T \models_{\mathcal{D}} C' \leftrightarrow (C \cup P)$, and finally (v) if $\mathtt{oracle}(C, P)$ then $T \models_{\mathcal{D}} P$. The proof proceeds by straightforward mutual rule induction.

3 Case Studies

In this section, we show that contextual rewriting as well as the integration schemas of NQTHM and Tecton are all instances of CCR.

3.1 Contextual Rewriting

Contextual rewriting is the instance of CCR where the decision procedure is taken to be an (incremental) satisfiability procedure for the theory of equality. The constraint store is a pair $\langle A, G\rangle$ where A is a set of literals and G (G°) is the (initial) state of the decision procedure and keeps in some internal form a representation of the equivalence classes induced by the ground equalities fed to the procedure. $\mathtt{cs\text{-}init\text{-}state}(C)$ holds if and only if $C = \langle \emptyset, G^\circ\rangle$. $\mathtt{cs\text{-}unsat}(\langle A, G\rangle)$ if and only if $(s \neq s) \in A$ for some term s, or $p \in A$ and $\neg p \in A$ for some atom p.

$$\langle A, G\rangle :: e \xrightarrow[\mathsf{cs-normal}]{\mathsf{cc-canon}} \mathtt{cc\text{-}canon}(G, e) \qquad \frac{\langle A \cup P, G\rangle \xrightarrow[\mathsf{cs-simp}]{} \langle A', G'\rangle}{P :: \langle A, G\rangle \xrightarrow[\mathsf{cs-ext}]{\mathsf{cs-ext}} \langle A', G'\rangle}$$

where $\xrightarrow[\mathsf{cs-simp}]{}$ is defined by:

$$\langle A \cup \{a = b\}, G\rangle \xrightarrow[\mathsf{cs-simp}]{\mathsf{cc-merge}} \langle A, \mathtt{cc\text{-}merge}(G, a, b)\rangle$$

$$\langle A \cup \{p\}, G\rangle \xrightarrow[\mathsf{cs-simp}]{\mathsf{cc-canon}} \langle A \cup \{\mathtt{cc\text{-}canon}(G, p)\}, G\rangle$$

where $\mathtt{cc\text{-}canon}(G, p)$ returns the representative of the equivalence class of p in G, and $\mathtt{cc\text{-}merge}(G, a, b)$ denotes the state obtained from G by merging the equivalence classes of a and b in G.

To illustrate consider the following example. Let $R = \{u \neq 0 \to \mathit{rem}(u * v, u) \Rightarrow 0\}$. It can be shown that $\{(\mathit{rem}(y * z, x) = 0), (x * y \neq y * z), (x = 0)\} \xrightarrow[\mathsf{simp}]{}^{*} \{\mathit{true}\}$. The sub-derivation associated to CCR is as follows (where G_0 is the constraint store representing $\{(x * y = y * z)\}$):

$$\langle \{x \neq 0\}, G_0\rangle :: (\mathit{rem}(y * z, x) = 0) \xrightarrow[\mathsf{ccr}]{\overset{\Pi_1}{\mathsf{normal}}} (\mathit{rem}(x * y, x) = 0) \xrightarrow[\mathsf{ccr}]{\overset{\Pi_2}{\mathsf{crew}}} (0 = 0) \xrightarrow[\mathsf{ccr}]{\overset{\Pi_3}{\mathsf{entails}}} \mathit{true}$$

where Π_1 is the singleton sequence containing the following sub-derivation:

$$\frac{(\mathit{rem}(y * z, x) = 0) :: \langle \{x \neq 0\}, G_0\rangle \xrightarrow[\mathsf{cs-se}]{\mathsf{cs-se}} \langle \{x \neq 0\}, G_0\rangle}{\langle \{x \neq 0\}, G_0\rangle :: (\mathit{rem}(y * z, x) = 0) \xrightarrow[\mathsf{ccr}]{\mathsf{normal}} (\mathit{rem}(x * y, x) = 0)}$$

and Π_2 is the singleton sequence containing the following sub-derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\langle\{(x=0),x\neq 0\},G_0\rangle \xrightarrow[\text{cs-simp}]{\text{cc-merge}} \langle\{x\neq 0\},G_1\rangle \xrightarrow[\text{cs-simp}]{\text{cc-canon}} \langle\{0\neq 0\},G_1\rangle}{\{(x=0)\} :: \langle\{x\neq 0\},G_0\rangle \xrightarrow[\text{cs-ext}]{\text{cxt-fe}} \langle\{0\neq 0\},G_1\rangle}}{\langle\{x\neq 0\},G_0\rangle :: (x\neq 0) \xrightarrow[\text{ccr}]{\text{cxt-entails}} \mathit{true}}}{\langle\{x\neq 0\},G_0\rangle :: \{(x\neq 0)\} \xrightarrow[\text{rlv}]{\text{relieve}} \emptyset}}{\langle\{x\neq 0\},G_0\rangle :: (\mathit{rem}(x*y,x)=0) \xrightarrow[\text{ccr}]{\text{crew}} (0=0)}$$

(G_1 is the constraint store representing $\{(x=0),(x*y=y*z)\}$.) Since Π_3 can be readily constructed using the rules given in Section 2 it is omitted here. No augmentation is needed for the above example to succeed.

3.2 NQTHM

The way NQTHM incorporates an (incremental) decision procedure for linear arithmetic (LA for short) can be easily formalized in our framework.[5] The state of the decision procedure is taken to be a pair $\langle A,L\rangle$ where A is a finite set of literals, and L (L°) represent the (initial) state of the decision procedure, and keeps in some internal representation the linear facts issued by the simplifier. `cs-init-state`(C) holds if and only if $C=\langle\emptyset,L^\circ\rangle$. `cs-unsat` and $\xrightarrow[\text{cs-ext}]{}$ are defined as in Section 3.1, whereas $\xrightarrow[\text{cs-normal}]{}$ and $\xrightarrow[\text{cs-simp}]{}$ are defined in the following way:

$$\langle A,L\rangle :: e \xrightarrow[\text{cs-normal}]{\text{id}} e \qquad \dfrac{\texttt{linearize}(p) :: \langle L,\emptyset\rangle \xrightarrow[\text{push-polys}]{} \langle L',E\rangle}{\langle A\cup\{p\},L\rangle \xrightarrow[\text{cs-simp}]{\text{la-ext}} \langle A\cup E,L'\rangle} \quad \text{if } p \text{ is a } (\Sigma_j,\Pi_c)\text{-literal}$$

where `linearize`(p) returns a finite set of linear inequalities which logically imply p, and $\xrightarrow[\text{push-polys}]{}$ is defined as follows:

$$P :: \langle L,\emptyset\rangle \xrightarrow[\text{push-polys}]{\text{add-polys}} \langle L',E\rangle$$

where L' is the state resulting from the extension of L with P, and if L' is consistent then E is a (possibly empty) set of equalities entailed by L', otherwise $E=\{a\neq a\}$ for some term a.

To illustrate, consider a simple variant of the first example of Section 3.1, where $R=\{u>0\rightarrow \mathit{rem}(u*v,u)\Rightarrow 0\}$. It is easy to show that $\{(\mathit{rem}(x*y,x)=0),(x\leq 0)\} \xrightarrow[\text{simp}]{}^* \{\mathit{true}\}$.

More sophisticated forms of simplifications can be obtained if augmentation is used, and `oracle`$(\langle A,L\rangle,P)$ is defined to hold whenever *(i)* there exists a constraint rule $(p_1,\ldots,p_n\rightarrow r(\vec{s}))\in \mathit{CR}$, *(ii)* $\langle A,G\rangle :: \{p_1\sigma,\ldots,p_n\sigma\}\xrightarrow[\text{rlv}]{}\emptyset$, and *(iii)* $\langle A,G\rangle :: \vec{s}\sigma \xrightarrow[\text{ccr}]{} \vec{t}$, and *(iv)* $P=\{r(\vec{t})\}$. The following complex example (from [3]) shows the power of the resulting simplification mechanism. By taking $\mathit{CR}=\{((0<i)\rightarrow j\leq i*j),(0<\mathit{ms}(x))\}$ it can be shown that $\{\mathit{ms}(c)+\mathit{ms}^2(a)+\mathit{ms}^2(b)<\mathit{ms}(c)+\mathit{ms}^2(b)+2*(\mathit{ms}^2(a)*\mathit{ms}(b))+\mathit{ms}^4(a)\} \xrightarrow[\text{simp}]{}^* \{\mathit{true}\}$ where $\mathit{ms}^2(x)$ abbreviates $\mathit{ms}(x)*\mathit{ms}(x)$ and $\mathit{ms}^{n+1}(x)$ abbreviates $\mathit{ms}(x)*\mathit{ms}^n(x)$, for $n>1$.

[5]For the lack of space, we do not consider some features of the actual system (i.e. the use of the *type-set specialist* and the (weak) form of reasoning about equalities carried by the simplifier). However there is no serious obstacle towards extending our current formulation to include such features.

3.3 Tecton

The decision procedure is a combination of a decision procedure for the theory of equality and a decision procedure for LA. The constraint store is a triple $\langle A, G, L\rangle$ where A is a set of literals, L and G (L° and G°) represent the (initial) state of the decision procedures for LA and the theory of equality, respectively. $\texttt{cs-init-state}(C)$ holds if and only if $C = \langle \emptyset, G^\circ, L^\circ\rangle$.

$$\langle A, G, L\rangle :: e \xrightarrow[\mathsf{cs-normal}]{\mathsf{cc-canon}} \mathsf{cc\text{-}canon}(G, e) \qquad \frac{\langle A \cup \texttt{separate}(P), G, L\rangle \xrightarrow[\mathsf{cs-simp}]{} \langle A', G', L'\rangle}{P :: \langle A, G, L\rangle \xrightarrow[\mathsf{cs-ext}]{\mathsf{cs-ext}} \langle A', G', L'\rangle}$$

$\texttt{separate}(P)$ returns a set of literals which are either $(\Sigma_c \cup \Delta, \Pi_c)$-literals or equalities between terms in $\tau(\Sigma_p \cup \Delta)$ where $\Sigma_p = \Sigma_j \setminus \Sigma_c$ and Δ is a new set of constants (called *abstraction constants*) such that $\Sigma_c \cap \Delta = \emptyset$ and $\Sigma_p \cap \Delta = \emptyset$. It is assumed that the set of literals returned by $\texttt{separate}(P)$ is unsatisfiable if and only if P is.

$$\langle A \cup \{s = t\}, G, L\rangle \xrightarrow[\mathsf{cs-simp}]{\mathsf{cc-merge}} \langle A, \texttt{cc-merge}(G, s, t), L\rangle \qquad \text{if } s, t \in \tau(\Sigma_p \cup \Delta)$$

$$\langle A \cup \{p\}, G, L\rangle \xrightarrow[\mathsf{cs-simp}]{\mathsf{cc-canon}} \langle A \cup \{\texttt{cc-canon}(G, p)\}, G, L\rangle$$

$$\frac{\texttt{linearize}(p) :: \langle L, \emptyset\rangle \xrightarrow[\mathsf{push-polys}]{} \langle L', E\rangle}{\langle A \cup \{p\}, G, L\rangle \xrightarrow[\mathsf{cs-simp}]{\mathsf{la-ext}} \langle A \cup E, G, L'\rangle} \qquad \text{if } p \text{ is a } (\Sigma_c \cup \Delta, \Pi_c)\text{-literal}$$

$\texttt{oracle}(\langle A, G, L\rangle, P)$ is defined to hold whenever: *(i)* there exists a constraint rule $(p_1, \ldots, p_n \rightarrow r(\vec{s})) \in CR$, *(ii)* $\langle A, G, L\rangle :: \{p_1\sigma, \ldots, p_n\sigma\} \xrightarrow[\mathsf{rlv}]{} \emptyset$, *(iii)* $\langle A, G, L\rangle :: \vec{s}\sigma \xrightarrow[\mathsf{ccr}]{} \vec{t}$, and *(iv)* $P = \{r(\vec{t})\}$, or *(1)* there exists a term u occurring in A or G, *(2)* $\langle A, G, L\rangle :: u \xrightarrow[\mathsf{ccr}]{}{}^{*} v$, and *(3)* $P = \{(u = v)\}$.

The following example (from [9]) shows the strength of the resulting simplification mechanism. Let $R = \{max(x, y) = x \rightarrow min(x, y) \Rightarrow y\}$ and $CR = \{p(x) \rightarrow f(x) \leq g(x)\}$. It can be shown that the clause $\{\neg p(x), \neg(z \leq f(max(x, y))), \neg(0 < min(x, y)), \neg(x \leq max(x, y)), \neg(max(x, y) \leq x), z < g(x) + y\}$ can be simplified to $\{true\}$.

Implementation. Our framework can be readily built on top of several implementation languages and logical frameworks. We found convenient to build a first prototype in Prolog, following the methodology envisaged in [8] (for more on this, see section 4). This amounted to a careful design of the sequents and the rules, as well as the implementation of the interpreter for the strategies specifying the application of the rules. Next we refined the rules described in Sections 2 and 3 to encode control information (e.g. a field storing the literal being rewritten so as to avoid looping). As far as the strategy of application of the rules is concerned, a simple depth-first strategy is sufficient to tackle all the examples shown in this paper.

4 Related Work

In [3], Boyer and Moore carefully report the integration of a decision procedure for LA within their prover. In [10], Kapur and Nie describe the integration of a decision procedure for LA with contextual rewriting. This results in a powerful form of simplification. As shown in Section 3 both integration schemas have been neatly recasted in our framework. The advantage is that CCR provides us with an abstract characterization of the functionalities provided by the decision procedure thereby providing an integration schema which is independent from the specific decision procedure employed. PVS [13] comes with a decision procedure for equality and LA (among many others). The decision procedures are tightly integrated following the schema proposed in [14]. A preliminary analysis of PVS suggests that also such an integration schema can be recast in our framework. STeP [7] implements a form of contextual rewriting integrated with a semi-decision procedure for first-order logic and a bunch of decision procedures extended to deal with non-ground constraints. The encoding of the integration schema used in STeP in our framework seems to be a very interesting experiment. [8] describes a framework and a methodology (Open Mechanized Reasoning System, OMRS) to compose and to enhance reasoning modules. As an application of OMRS, a rational reconstruction of Boyer and Moore's integration schema is reported in [6]. Following OMRS, we adopt a rule-based approach to specify reasoning modules, and in the definition of the rules we stress the distinction between logic and control.

Many extensions to constraint solving aiming at a better trade-off between declarativity and efficiency have been put forward in Constraint Programming (CP). In particular, SoleX [11] is a mechanism for the domain independent extension of constraint solvers so to deal with programmer-defined constraints. This work resembles ours in the rule-based presentation, in the semantical extension of the constraint rules by means of guarded rewrite rules, and in being domain independent. In [5], a general framework for the extension/combination of solvers using rewrite rules and strategies is proposed. In [1], Apt gives a proof theoretic account of constraint solving. This work is similar to ours in the rule based presentation, it is more fundational in nature since it aims at giving an abstract account of CP in the "proof as computation" paradigm, but it does not deal with integration issues. In [2], we compare the Constraint Logic Programming paradigm (CLP) with CCR, discussing potentials for cross-fertilization between the fields of Automated Theorem Proving and CLP on the problem of integrating decision procedures.

5 Conclusions

We have proposed an extension to contextual rewriting (called *Constraint Contextual Rewriting*), where contextual information can be accessed by the available decision procedures. The interface functionalities between the rewriting engine and the decision procedures have been precisely described. These functionalities specify how the parts interact, yielding an effective simplification mechanism. The soundness of the integration schema has been hinted. The encoding in our framework of the Boyer and Moore's integration schema and the one adopted in Tecton gives evidence of the flexibility and generality of the approach. Prototypes of these two integration schemas have been easily derived from the rule-based presentation and used to validate the models. Further properties (e.g. termination) are under investigation.

References

[1] K. R. Apt. A Proof Theoretic View of Constraint Programming. *Fundamenta Informaticae*, 33(1):1–27, 1998.

[2] A. Armando, E. Melis, and S. Ranise. Constraint Solving in Logic Programming and in Automated Deduction: a Comparison. In *Proc. 8th Intl. Conf. on Artificial Intelligence: Methodology, Systems, Applications (AIMSA'98). Sozopol, Bulgaria, September 21 - 23.*, 1998.

[3] R. S. Boyer and J S. Moore. Integrating Decision Procedures into Heuristic Theorem Provers: A Case Study of Linear Arithmetic. *Machine Intelligence*, 11:83–124, 1988.

[4] R.S. Boyer and J S. Moore. *A Computational Logic.* Academic Press, 1979.

[5] C. Castro. Building Constraint Solvers Using Rewrite Rules and Strategies. Submitted for publication. Preliminar version available at `http://www.loria.fr/-~castro/PAPERS/publications.html`, 1998.

[6] A. Coglio, F. Giunchiglia, P. Pecchiari, and C. L. Talcott. A Logic Level Specification of the NQTHM Simplification Process. Technical report, IRST, 1997.

[7] Z. Manna *et. al.* STeP: The Stanford Temporal Prover. Technical Report CS-TR-94-1518, Stanford University, June 1994.

[8] F. Giunchiglia, P. Pecchiari, and C. L. Talcott. Reasoning Theories: Towards an Architecture for Open Mechanized Reasoning Systems. Technical Report TR-9409-15, IRST, Nov. 1994.

[9] D. Kapur, D.R. Musser, and X. Nie. An Overview of the Tecton Proof System. *Theoretical Computer Science*, Vol. 133, October 1994.

[10] D. Kapur and X. Nie. Reasoning about Numbers in Tecton. In *Proc. 8th Inl. Symp. Methodologies for Intelligent Systems*, pages 57–70, Charlotte (North Carolina), October 1994.

[11] E. Monfroy and C. Ringeissen. SoleX: a Domain-Independent Scheme for Constraint Solver Extension. In 4th *Intl. Conf. on Artificial Intelligence and Symbolic Computation (AISC'98). Plattsburgh, New York, September*, 1998.

[12] G. Nelson and D.C. Oppen. Simplification by Cooperating Decision Procedures. Technical Report STAN-CS-78-652, Stanford Computer Science Department, April 1978.

[13] S. Owre, J. M. Rushby, and N. Shankar. PVS: A Prototype Verification System. In *Proc. 11th Intl. Conf. on Automated Deduction (CADE'92). Saratoga, June 748 – 752.*, 1992.

[14] R.E. Shostak. Deciding Combination of Theories. *Journal of the ACM*, 31(1):1–12, 1984.

[15] H. Zhang. Contextual Rewriting in Automated Reasoning. *Fundamenta Informaticae*, 24(1/2):107–123, 1995.