# Rewriting Needs Constraints and Constraints Need Rewriting

José Meseguer

Department of Computer Science, UIUC

NSF Workshop on Symbolic Computation and Constraints
14 November 2008

# Symbolic Computation, Rewriting, and Constraints

Rewriting is a very general mechanism for symbolic computation; therefore, research on rewriting techniques is an essential part of symbolic computation research.

Similarly, solving of constraints is a very essential part of symbolic computation, so that constraint solving is likewise an essential part of symbolic computation research.

This talk will try to make obvious that rewriting and constraints are intimately related, need each other, and can help each other in fundamental ways.

After a quick review of basic concepts for constraints and rewriting, I will focus on the mutual help and inter-dependence between rewriting and constraint solving.

# Validity, Satisfiability, and Constraints

Let $\mathcal{C}$ be a class of first-order formulas, and let $T$ be a first-order theory such that $\mathcal{C} \subseteq \mathcal{L}(T)$. We say that $\varphi \in \mathcal{C}$ is $T$-valid iff

$$T \vdash \varphi$$

We say that $\varphi \in \mathcal{C}$ is $T$-satisfiable iff

$$T \cup \varphi \not\vdash \bot$$

Obviously, validity is a special case of satisfiability since (for $\varphi$ a sentence) we have,

$$T \vdash \varphi \quad \Leftrightarrow \quad T \cup \neg\varphi \vdash \bot$$

Let $M$ be a model of a theory $T$. For an appropriate class $\mathcal{C}$ of formulas, called constraints, the constraint $M$-satisfaction problem is the problem of whether, given $\varphi \in \mathcal{C}$, we have

$$M \models \varphi$$

# Satisfiability and Constraint Satisfaction Procedures

It is obviously very important to find theories $T$ and classes of formulas $\mathcal{C}$ such that the problem of whether $\varphi \in \mathcal{C}$ is $T$-satisfiable is decidable.

Likewise it is very important to find for a model of interest $M$ a class $\mathcal{C}$ of constraints such that the problem of whether $M \models \varphi$ is decidable.

These two problems are different; however, in some cases they may be equivalent. For example, for $T = E$ a set of $\Sigma$-equations, and $u = v$ an equation, we have equivalences:

$$\forall \vec{x}\ u \neq v \ \ E{-}invalid \quad \Leftrightarrow \quad E \vdash \exists \vec{x}\ u = v \quad \Leftrightarrow \quad T_{\Sigma/E}(X) \models \exists \vec{x}\ u = v$$

which is the $E$-unifiability problem.

# Rewriting

Giving a signature $\Sigma$ of function symbols, a rewite rule is a sequent $l \longrightarrow r$ with $l, r$ $\Sigma$-terms, with $vars(r) \subseteq vars(l)$.

A collection $R$ of rewrite rules defines a rewriting relation $t \longrightarrow_R^* t'$ between $\Sigma$-tems as the reflexive-transitive closure of the relation

$$t[\theta(l)] \longrightarrow_R t[\theta(r)]$$

for some substitution $\theta$ and some $l \longrightarrow r$ in $R$.

For example, for $R$ the single rule $x + 0 \longrightarrow x$ performs the rewrite $4 + (3 + 0) \longrightarrow_R 4 + 3$. with $\theta = \{x \mapsto 3\}$.

More generally, for rules $R$ and equations $A$, the rewriting modulo $A$, $t \longrightarrow_{R/A}^* t'$, is the reflexive-transitive closure of

$$t[u] \longrightarrow_{R/A} t[\theta(r)]$$

for some $\theta$ and some $l \longrightarrow r$ in $R$ such that $u =_A \theta l$. E.g., for the above rule we get $4 + (0 + 3) \longrightarrow_{R/C} 4 + 3$ when $C = \{x + y = y + x\}$.

# Two Semantics for Rewriting

Given a *rewrite theory* $\mathcal{R} = (\Sigma, A, R)$ two different semantics are possible for $\mathcal{R}$:

1. An equational semantics, in which we read $l \longrightarrow r$ as an oriented equation. For example, $x + 0 \longrightarrow x$ is read as the oriented form of $x + 0 = x$. If the rules $R$ are confluent, terminating, and coherent modulo $A$, this semantics gives a decision procedure for the $R \cup A$-word problem, provided $A$-matching is decidable.

2. A rewriting logic semantics, in which we read $l \longrightarrow r$ as a transition rule in a concurrent system or, alternatively, as an inference rule in a logic. For example:

$$credit(A, M)\ ,\ [A : Accnt \mid bal : N] \longrightarrow [A : Accnt \mid balN + M]$$

$$A\ ,\ (A \Rightarrow B) \longrightarrow B$$

where in both cases the operation $\_ , \_$ is associative and commutative.

# Rewriting Strategies

Under an equational semantics with confluent and terminating rules all rewriting sequences end in a unique canonical form. Therefore, rewriting strategies are not essential and only affect performance.

However, under a rewriting logic semantics, the rules need not be confluent or terminating, and strategies become very important. We may assume that rules are labeled, e.g., $a : l \longrightarrow r$, and can define a strategy as a regular-like expression on labels such as, for example,

$$(a! \,.\, b) + (c^{+} .\, a)!$$

where $a!$ applies rule $a$ "to the bitter end," and the other regular expression notations have the obvious meaning.

We can then define a rewriting algorithm as a pair $(\mathcal{R}, strat)$, with $\mathcal{R}$ a rewrite theory, and $strat$ a strategy for $\mathcal{R}$. For example, $\mathcal{R}$ can be an inference system and $strat$ a strategy for it. This has the advantage of allowing us to reason about the correctness of $\mathcal{R}$ quite independently of the strategy $strat$.

# Rewriting Needs Constraints

Let $\mathcal{R} = (\Sigma, A, R)$ be a rewrite theory. I will give three instances in which constraint solving is essential for rewriting: (1) the rewriting relation, (2) the narrowing relation, and (3) inductive reasoning in initial algebras.

Consider the rewriting modulo $A$ relation

$$t[u] \longrightarrow_{R/A} t[\theta(r)]$$

with a rule $l \longrightarrow r$ in $R$. For this relation to be decidable, we need to decide whether there is a substitution $\theta$ with $u =_A \theta l$.

This is just the $A$-matching problem, which is the special case of the $A$-unification problem consisting of the solving of the constraint

$$\exists \vec{x}\; l =_A u$$

where $\vec{x} = vars(l)$ and we assume that $u$ has no variables (ground term). For example, Maude supports rewriting modulo $A$ by efficient $A$-matching algorithms for $A$ any combination of associativity and/or commutativity and/or identity axioms.

# Narrowing

We can think of narrowing as doubly symbolic rewriting. Given $\mathcal{R} = (\Sigma, A, R)$ we define the narrowing modulo $A$ relation $t \leadsto^*_{R/A} t'$ as the reflexive-transitive closure of the relation

$$t[u]_p \leadsto_{R/A} t[\theta(r)]_p$$

where $p$ is a nonvariable position in $t$, and there is a rule $l \longrightarrow r$ in $R$ and a substitution $\theta$ which is a unifier of $u$ and $l$ modulo $A$, i.e., $\theta(u) =_A \theta l$. For example, $4 + (y + 3) \leadsto_{R/C} 4 + 3$ with $x + 0 \longrightarrow x$ in $R$, $\theta = \{x \mapsto 3, y \mapsto 0\}$, modulo $C = \{x + y = y + x\}$.

Obviously, to perform narrowing modulo $A$ we need to solve $A$-unification constraints of the form $\exists \vec{x}\ u =_A l$.

Narrowing is a crucial mechanisms in many ways, including: (1) functional-logic programming languages; (2) Knuth-Bendix completion (KB) of a rewrite theory $\mathcal{R}$ with equational semantics; (3) $R \cup A$-unification; and (4) symbolic reachability analysis with transition semantics. For example, the Maude-NPA tool performs complete symbolic reachability analysis of cryptographic protocols modulo algebraic properties of the cryptographic functions by narrowing.

# Inductive Reasoning in Initial Models

Very often we need to reason not just in a theory $T$, but in the standard model of such a theory. Inductive reasoning is precisely sound reasoning in such a model. For example, if $T$ is an equational theory or a Horn theory, $T$ has an initial model called, respectively, the initial algebra, or the Herbrand model of $T$. A rewrite theory $\mathcal{R}$ has also an initial model.

Many theorem proving systems (e.g., PVS) use a variety of constraint solvers and satisfiability decision procedures for reasoning that is often inductive. For example, decision procedures for data structures are precisely procedures to reason about equalities and inequalities between constructor terms in an initial model. Similarly for linear arithmetic.

The Maude formal environment performs inductive reasoning using:

- order-sorted unification modulo commutativity and associativity-commutativity;
- tree automata modulo any combination of associativity and/or commutativity and/or identity axioms;
- sat solvers, linear arithmetic, and uninterpreted function symbols;
- an LTL model checker for finite-state rewrite theories.

# Constraints Need Rewriting

Constraint solving and satisfiability procedures need rewriting techniques in several ways:

- To reason about their correctness and about alternative algorithms.
- To cover more applications by integrating theory-specific algorithms with theory-generic ones.
- To easily develop prototypes and even efficient implementations.

Traditional descriptions of constraint solving and decision procedures, have tended to be implementation-oriented. Although obviously hepful to develop efficient implementations, such descriptions are unnecessarily complex and low-level, posing serious challenges to both correctness and understandability.

For example, it has taken about 20 years to fully understand Shostak's algorithm and its correctness issues. The point is that it is hard from the jungle of pointers of an implementation-oriented description to disentangle the essentail correctness aspects form the optimizations.

# Rewriting Logic Specification of Decision Procedures

Since the seminal Martelli&Montanari paper on unification it has been more and more clearly understood that the essential aspects of a constraint solver or decision procedure should be captured declaratively by means of an inference system, whereas the control and optimization aspects can be then specified as strategies to apply the inference rules.

This is exactly what rewriting logic provides as a logical framework in which an inference system is specified as a rewrite theory $\mathcal{R}$, and various alternative implementations of the inference system are captured by alternative strategies, giving rise to algorithms:

$$(\mathcal{R}, strat_1), (\mathcal{R}, strat_2), \ \ldots \ (\mathcal{R}, strat_n)$$

Reasoning about the correctness of the inference system can be isolated to the declarative level of the rewrite rules in $\mathcal{R}$, whereas efficiency, complexity, and implementation issues are dealt with at the level of the strategies $strat_1, strat_2, \ldots, strat_n$.

This can often be done without disregarding implementation and complexity issues.

# Rewriting Logic Specification of Decision Procedures (II)

This approach has been followed for example to specify and reason about:

- Unification algorithms modulo different theories $A$ (Jouannaud and Kirchner).
- Abstract congruence closure, specified as a KB-like inference system, so that: (i) the various Shostak, Downey-Sethi-Tarjan, and Nelson-Oppen algorithms appear precisely as alternative strategies; and (ii) congruence closure is generalized modulo $AC$ and $ACU$ (Bachmair-Tiwari-Vigneron, building upon Kapur).
- Groebner basis computation, understood and generalized as a KB-like completion method, modulo the theory of rings (Kapur, Tiwari).
- Combinations of Decision Procedures, understood as an inference system so that the different combination procedures such as Shostak, Nelson-Oppen, and Shankar-Ruess appear as alternative strategies (Conchon&Krstić, building upon Tinelli&Harandi).
- Abstract $DPLL$ and $DPLL(T)$, specified both as rewrite theories, so that different sat-solving and SMT-solving algorithms are then understood as alternative strategies (Nieuwenhuis-Oliveras-Tinelli).

# Combining Theory-Specific and Theory-Generic Procedures

The advantage of theory-specific procedures is their decidability, but often they deal only with a fragment of the entire problem. It is therefore very useful to combine them with theory-generic procedures to cover a much broader range of applications. Narrowing modulo $A$ provides good examples of this method:

1. In the equational interpretation of $\mathcal{R} = (\Sigma, A, E)$, if $A$ has a unification algoritm, and $E$ is confluent, terminating, and coherent modulo $A$, then narrowing provides a complete $A \cup E$-unification algorithm, which is finitary under some assumptions (Escobar-Meseguer-Sasse in the modulo $A$ case, building upon Comon-Delaune, Jouannaud-Kirchner-Kirchner, and Hullot).
2. In the transition interpretation of $\mathcal{R} = (\Sigma, B, R)$, under appropriate assumptions narrowing provides a complete procedure to answer symbolic reachability questions of the form:

$$\exists \vec{x}\ t \longrightarrow^{*}_{R/B} t'$$

(Thati&Meseguer, exploited by Escobar-Meadows-Meseguer in the Maude-NPA).

# Declarative Does not Mean Inefficient

Giving declarative specifications of decision procedures for constraint solving and formula satisfiability is not only an easy way to prototype such procedures. In many cases, a good rewriting specification running on a high-performance rewrite engine can compete with or even outperform well-engineered implementations.

For example, a 30-line semantic definition in rewriting logic of Milner's unification-based type checking algorithm by Roșu and his students at UIUC outperforms the SML compiler when executed in Maude:

| - | Average | Stress test | | | | |
|---|---|---|---|---|---|---|
| | Speed | n = 10 | n = 11 | n = 12 | n = 13 | n = 14 |
| OCaml | 0.6s | 0.6s | 2.1s | 7.9s | 30.6s | 120.5s |
| Haskell | 1.2s | 0.5s | 0.9s | 1.5s | 2.5s | 5.8s |
| SML | 4.0s | 5.1s | 14.6s | 110.2s | 721.9s | - |
| W in Maude | 1.1s | 2.6s | 7.8s | 26.9s | 103.1s | 373.2s |
| W+ in Maude | 2.0s | 2.6s | 7.7s | 26.1s | 96.4s | 360.4s |
| W in PLT/Redex[1] | 134.8s | >1h | - | - | - | - |
| W in OCaml | 49.8s | 105.9s | 9m14 | >1h | - | - |

# Conclusions

Both constraint satisfaction and rewriting are essential for symbolic computation.

Furthermore, as I have barely hinted at but will be further developed in other workshop talks, the range of applications supported by constraint satisfaction and by rewriting techniques is immense: they greatly broaden the applicability of symbolic computation methods.

I hope I have given ample evidence for the conclusion that may talk title suggests:

*Rewriting Needs Constraints and Constraints Need Rewriting*.