Satisfiability Modulo Structures as Constraint Satisfaction: An Introduction

Hassan Aït-Kaci[1] & Bruno Berstel[2] & Ulrich Junker[2] & Michel Leconte[2] & Andreas Podelski[3]

1: ILOG, Inc.
hak@ilog.com
2: ILOG S.A.
{bberstel,ujunker,mleconte}@ilog.fr
3: University of Freiburg
podelski@informatik.uni-freiburg.de

Abstract

Constraint Programming (CP) and Satisfiability Modulo Theories (SMT) both generalize SAT w.r.t. expressiveness and propagation power. CP uses domain propagators based on mathematical structures, whereas SMT uses constraint propagators based on logical theories. In this paper, we incorporate SMT reasoning into CP and illustrate the benefits of the resulting system for the problem of rule verification.

1. Introduction

SAT methods have intensively been applied to hardware verification and are gaining in importance for software verification. As programs involve operations on different data types such as integers, floating-points, booleans, strings, arrays, lists, and objects, satisfiability problems in software verification need to take the mathematical properties of those data types into account. This can be achieved by dedicated decision procedures, which check the satisfiability of a formula w.r.t. different mathematical theories (*e.g.*, linear arithmetics, symbolic functions, array theory etc). An important topic is then to find correct and complete combinations of those decision procedures. Nelson and Oppen [6] showed that equality propagation between decision procedures achieves completeness if equality is the only operation that is shared by two theories. This pioneering work is fundamental for software verification and has influenced many program verifiers as explained in [5].

Whereas the Nelson-Oppen method is able to prove the unsatisfiability of formulas such as $x = y \wedge f(x) = f(y) + 1$, it does not detect the unsatisfiability of the formula $x.\texttt{startsWith}(y) \wedge x.\texttt{length} + 1 = y.\texttt{length}$, which combines string operations with integer arithmetics. Only recent research by Tinelli and Ringeissen [11, 10] has shown how to combine decision procedures for theories that share more than equality into a semi-decision procedure for unsatisfiability. Craig interpolation based on conjunctions of clauses over the shared variables is one way to achieve this combination [10].

Craig interpolation is also used in the DPLL(T)-method [9], which combines propositional logic with a decision procedure for constraints that are attached to propositional variables. From the stand-point of Constraint Programming (CP), DPLL(T) as in [9] attaches a Boolean variable to each literal and maintains a global consistency among those Boolean variables w.r.t. a given theory. As maintaining global consistency is expensive, it may be more efficient to use incomplete forms of propagation [2].

Satisfiability modulo theories (SMT) addresses problems with infinite domains (as for integer arithmetics) or unknown domains (as for uninterpreted functions). Whereas SMT provides a semi-decision procedure for unsatisfiability [11, 5] and requires that mathematical structures have decision procedures or first-order axiomatizations, the interest of CP is to provide semi-decision procedures for satisfiability, while requiring only decision procedures for each function and each predicate in the mathematical structure. Informally, we can say that SMT is able to enumerate proofs, while CP is able to enumerate solutions. SMT achieve this by propagating literals (*i.e.*, constraints) among decision procedures, whereas CP propagates reduced variable domains among constraints.

As the CP framework is quite general, it is possible to incorporate SMT-based reasoning by creating all relevant constraints subject to SMT-propagation in advance and by posting a global constraint on them that encapsulates a decision procedure. However, this approach does not work for uninterpreted functions and other purely symbolic theories, which do not have a fixed semantical counterpart. In this paper, we address this issue of combining symbolic and semantic reasoning within CP, which is motivated by a verification problem, namely that of the consistency and coherence of production rules.

The major obstacle is that uninterpreted and interpreted functions work on different universes such as Herbrand terms and integer domains and that the same expression may be interpreted differently by different decision procedures. In order to obtain an effective propagation, we need to combine propagations on various domains, such as equality propagation over Herbrand terms or interval propagation over numerical constraints. A recent paper of Baader and Ghilardi [1] uses an explicit homomorphism to connect different copies of the constraints, which can be interpreted differently. We introduce copies of the interpreted functions. We interpret the original symbols such as $+$ syntactically (on Herbrand terms) and the copy $+_i$ semantically by the integer sum. We connect both via a function h, which is a homomorphism since we add the constraint $h(t_1 + t_2) =_i h(t_1) +_i h(t_2)$ for each subterm $t_1 + t_2$ of the given formula. Unlike [1], we are not connecting theories, but mathematical structures and we do not need to split the initial formula into separate subformulas. Structure connection turns out to be very naturally compared to theory connection as it just attaches meaning to syntactic terms via the well-known concept of a homomorphism between structures. We show that this is sufficient to test the satisfiability of a quantifier-free formula within a CP approach.

2. Verification and Constraint Programming

In this section, we show that even very simple verification problems can profit from different types of propagation including Boolean propagation, equality propagation, and the propagation of numeric intervals. We explain why Constraint Programming can be beneficial for verification and why it needs to be extended by other propagation methods for this purpose.

We consider a simple problem occurring in rule verification. Business rule management systems are becoming a popular way to manage business policies in business applications such as pricing, claim processing, loan accrediting, and others. As business rules can be entered by different experts, verification and validation methods are needed to consolidate the given rule set. Suppose that an expert dealing with a customer fidelity program has entered a pricing rule that adds a discount of 20% to customer orders of at least 10000 Euros. Formally, the rule is expressed as a production rule consisting of a condition, which is applied to given objects, and an action, which modifies those objects. Our rule is applicable if there is an object `o` which is an instance of type `Order` and which has a value greater than or equal to 10000. If the rule is applied then the discount of the object is increased by 20:

```
r1: if o: Order(value >= 10000) then o.discount += 20;
```

The manager of the rule base now imposes that discounts must always be within the range 0 to 30%. She asks whether there is any rule that can move a discount out of this domain. We thus obtain a

verification problem, which consists in checking whether a safety property can be violated. A typical approach to solve this problem is to see whether there is a scenario in which the rule is applied and the property is violated. For this purpose, we need to model the rule execution, which represents a transition of a state s_1 to a state s_2. We model this transition between unknown states by a quantifier-free first-order formula. We use a constant symbol $\mathtt{o}$ for the matched object and unary terms indexed by the state for the value and discount attributes.

$$\begin{array}{l} \mathtt{value}_{\mathtt{s1}}(\mathtt{o}) \geq 10000 \\ \mathtt{discount}_{\mathtt{s2}}(\mathtt{o}) = \mathtt{discount}_{\mathtt{s1}}(\mathtt{o}) + 20 \end{array} \tag{1}$$

We also state that the initial discount of the order is within $[0, 30]$:

$$\mathtt{discount}_{\mathtt{s1}}(\mathtt{o}) \geq 0 \wedge \mathtt{discount}_{\mathtt{s1}}(\mathtt{o}) \leq 30 \tag{2}$$

We also state that the discount does not change for any other object that is different to $\mathtt{o}$. In the simple example, we consider one unknown object in addition to $\mathtt{o}$ and use the variable $\mathtt{x}$ to denote it:

$$\neg(\mathtt{x} = \mathtt{o}) \Rightarrow \mathtt{discount}_{\mathtt{s2}}(\mathtt{x}) = \mathtt{discount}_{\mathtt{s1}}(\mathtt{x}) \tag{3}$$

Furthermore, we suppose that the transition moves the discount of the object $\mathtt{x}$ out of the range $[0, 30]$:

$$\begin{array}{l} \mathtt{discount}_{\mathtt{s1}}(\mathtt{x}) \geq 0 \wedge \mathtt{discount}_{\mathtt{s1}}(\mathtt{x}) \leq 30 \\ \mathtt{discount}_{\mathtt{s2}}(\mathtt{x}) < 0 \vee \mathtt{discount}_{\mathtt{s2}}(\mathtt{x}) > 30 \end{array} \tag{4}$$

We then combine (1), (2),(4), and (3) into a conjunction ϕ and ask whether ϕ it is satisfiable. In first-order logic, this requires to find a model of ϕ. A model consists of a domain and of functions and relations over this domain that interpret the given symbols such that the formula evaluates to true. It is important to note that the symbols for equality, sum, and order have a fixed interpretation.

We now discuss informally how the problem can be solved by different forms of propagation. We associate an (infinite) interval domain to every numerical expression such as $\mathtt{discount}_{\mathtt{s2}}(\mathtt{x})$. Furthermore, we associate a Boolean domain $\{\mathrm{T}, \mathrm{F}\}$ to each atomic formula such as $\mathtt{x} = \mathtt{o}$ and $\mathtt{discount}_{\mathtt{s2}}(\mathtt{x}) < 0$. Initially, interval propagation reduces the interval domain of $\mathtt{value}_{\mathtt{s1}}(\mathtt{o})$ to $[10000, \infty)$ and the interval domains of $\mathtt{discount}_{\mathtt{s1}}(\mathtt{x})$ and of $\mathtt{discount}_{\mathtt{s1}}(\mathtt{o})$ to $[0, 30]$. Moreover, the sum constraint in (1) reduces the domain of $\mathtt{discount}_{\mathtt{s2}}(\mathtt{o})$ to $[20, 50]$. This means that any model of ϕ will evaluate those expressions to values within the reduced domains.

We now ask whether a model of ϕ may evaluate $\mathtt{x} = \mathtt{o}$ to false. We therefore set the Boolean domain of $\mathtt{x} = \mathtt{o}$ to $\{\mathrm{F}\}$. Boolean propagation of (3) will then set the Boolean domain of

$$\mathtt{discount}_{\mathtt{s2}}(\mathtt{x}) = \mathtt{discount}_{\mathtt{s1}}(\mathtt{x}) \tag{5}$$

to $\{\mathrm{T}\}$. As this constraint is true now, it will propagate the interval $[0, 30]$ from $\mathtt{discount}_{\mathtt{s1}}(\mathtt{x})$ to $\mathtt{discount}_{\mathtt{s2}}(\mathtt{x})$. Hence, neither the constraint $\mathtt{discount}_{\mathtt{s2}}(\mathtt{x}) < 0$, nor $\mathtt{discount}_{\mathtt{s2}}(\mathtt{x}) > 30$ can be satisfied, meaning that their Boolean domains are reduced to $\{\mathrm{F}\}$. The disjunction in (4) is thus violated and there is no model of ϕ that evaluates $\mathtt{x} = \mathtt{o}$ to false.

Therefore, the constraint $\mathtt{x} = \mathtt{o}$ must be true and we set its domain to $\{\mathrm{T}\}$. Neither Boolean propagation, nor interval propagation can now proceed with problem solving. However, equality propagation can do so as we can deduce that $\mathtt{discount}_{\mathtt{s2}}(\mathtt{x})$ must be equal to $\mathtt{discount}_{\mathtt{s2}}(\mathtt{o})$ since $\mathtt{x}$ is equal to $\mathtt{o}$. Hence, equality propagation posts a new constraint:

$$\mathtt{discount}_{\mathtt{s2}}(\mathtt{x}) = \mathtt{discount}_{\mathtt{s2}}(\mathtt{o}) \tag{6}$$

Interval propagation via this constraint sets the domain of $\mathtt{discount}_{\mathtt{s2}}(\mathtt{x})$ to $[20, 50]$. As a consequence, the constraint $\mathtt{discount}_{\mathtt{s2}}(\mathtt{x}) < 0$ is violated and its Boolean domain is set to $\{\mathrm{F}\}$. Boolean

propagation via (4) then sets the domain of $\texttt{discount}_{\texttt{s2}}(\texttt{x}) > 30$ to {T}. Interval propagation then reduces the domain of $\texttt{discount}_{\texttt{s2}}(\texttt{x})$ to $[31, 50]$. This interval is propagated via (6) to $\texttt{discount}_{\texttt{s2}}(\texttt{o})$ and the sum constraint in (1) reduces the domain of $\texttt{discount}_{\texttt{s1}}(\texttt{o})$ to $[11, 30]$. In this particular example, we can find a model of ϕ for any value in this domain. Hence, a safety violation can occur if `x` is equal to `o` and the initial discount of `o` is within $[11, 30]$. The rule verification system therefore alerts the manager:

> *When applying rule* `r1` *to an order* `o` *with discount strictly greater than* 10 *the discount of* `o` *will be outside the domain of* $[0, 30]$.

As a consequence, the manager of the rule base modifies the rule as follows:

```
r1': if o: Order(value >= 10000; discount <= 10) then o.discount += 20;
```

The safety violation check for this new rule adds the constraint $\texttt{discount}_{\texttt{s1}}(\texttt{o}) \leq 10$ to the conjunction ϕ. This new conjunction is unsatisfiable as we already deduced that $\texttt{discount}_{\texttt{s1}}(\texttt{o})$ must be strictly greater than 10 when ϕ is given.

This example shows that different kinds of propagations are useful for finding models of a quantifier-free formula. In the example, we used interval propagation, Boolean propagations, and equality propagations. Other forms of propagations based on modular arithmetics for integers [4], regular expressions for strings, and others are also useful.

It is important that these propagations are used altogether and operate on a single problem space. As the propagations reduce domains that are linked together, they can interact very tightly. We have seen that a Boolean propagation can trigger an equality propagation which triggers an interval propagations. The interval propagation may trigger Boolean propagations and so on. It is thus important to combine the different propagations within a single solver which maintains a single problem space in form of domains. This guarantees a good synergy among the propagations. If multiple solvers with multiple problem spaces were used, then significant effort for coordinating these spaces would be needed. Constraint Programming (CP) provides a well-elaborated framework in which different forms of propagation can interact quickly. We will see that we gain a semi-decision procedure for satisfiability of quantifier-free formulas (modulo given mathematical structures) within CP (supposed that domains are enumerable and constraints are decidable), which is very well-suited for verification problems such as safety checks.

3. Satisfiability Modulo Structures

In the last section, we gave an example of a rule verification problem and showed how it can be expressed as the satisfiability problem of a quantifier-free first-order formula. Many other problems in Computer Science, Operations Research, and Artificial Intelligence can be formulated as satisfiability problems in a logical language. It is often the case that this language mixes symbols that can freely be interpreted with symbols that have a fixed interpretation for the given problem. In the last section, we used constant symbols such as `o` and function symbols such as $\texttt{discount}_{\texttt{s1}}$ to express statements about objects. The interpretation of these symbols can freely be chosen. Furthermore, we used the binary function symbol $+$ to express a sum of integers and the binary predicate symbol $\geq$ for the greater-than-or-equal-to relation among integers. These symbols have a fixed mathematical meaning for the problem of consideration. In other terms, we use a fixed mathematical structure such as integer arithmetics to interpret this second class of symbols.

There are two approaches to solve satisfiability problems where some symbols are interpreted within a given mathematical structure. The first approach, called *satisfiability modulo theories* (SMT), requires an axiomatization of the mathematical structure in form of a first-order theory. The problem

then consists in finding a model of this first-order theory that also satisfies the original constraints. The second approach, which we call *satisfiability modulo structures* (SMS), tries to extend the given mathematical structure by an interpretation of the yet uninterpreted symbols. Hence, the problem consists in finding a model of the original constraints that extends the given mathematical structure.

SMT requires an axiomatization of the mathematical structure and thus can address only limited forms of integer arithmetics, such as Presburger arithmetics, for which axiomatizations exist. As SMT adds universally quantified formulas (the axioms) to the given constraints, it leads to a semi-decision procedure for unsatisfiability. This means that an SMT procedure can prove the inconsistency of an unsatisfiable problem within a finite number of steps, but may run forever if the problem is satisfiable. The SMS approach can address arbitrary mathematical structures that have enumerable universes and decidable predicates and functions. It provides a semi-decision procedure for satisfiability as it only needs to choose values for the subterms of the quantifier-free formula. Furthermore, it can profit from specific solving procedures that are available for the mathematical structure. An example are the interval propagation methods explained in the last section.

Satisfiability procedures as used in Mathematical Programming and Constraint Programming follow an SMS approach, but require that all symbols are interpreted by a mathematical structure. In this case, the satisfiability procedure just needs to determine values of the variables. In the general SMS setting, the satisfiability procedure also needs to choose the interpretation of terms involving uninterpreted functions. A convenient way is to use a *Herbrand structure* and to interpret those terms by themselves. However, we encounter two technical difficulties. Firstly, our constraints make statements about the equality and disequality of objects. We may write formulas such as $\mathrm{a} = \mathrm{b} \wedge \neg(\texttt{discount}(\mathrm{a}) = \texttt{discount}(\mathrm{b}))$. This formula is unsatisfiable if we interpret the syntactic equality by the identity relation. In this case, we interpret a and b by the same value. As a consequence, the ground terms $\texttt{discount}(\mathrm{a})$ and $\texttt{discount}(\mathrm{b})$ are interpreted by the same values as well and the formula is unsatisfiable. Structures which interpret the syntactic equality by the identity relation are called *equality structures* (cf. [8], section 3.1). Now consider the formula $\neg(\mathrm{a} = \mathrm{b}) \wedge \texttt{discount}(\mathrm{a}) = \texttt{discount}(\mathrm{b})$. Although a and b represent different objects, their discounts may be equal and the formula should be satisfiable. However, a Herbrand structure is not a suitable equality structure for our purpose. It is based on a free algebra (*cf. e.g.*, [8], section 3.1), which means that the equality of two terms $f(t_1) = f(t_2)$ implies the equality of their arguments $t_1 = t_2$. As a consequence, the second formula is unsatisfiable in a Herbrand structure which interprets the syntactic equality by the identity relation.

Our problem consists in finding a model of a first-order formula that interprets the syntactic equality by an identity relation. However, we can transform this satisfiability problem into a different satisfiability problem which uses a Herbrand structure. Herbrand structures have the advantage that less values need to be chosen by the satisfiability procedure. If only uninterpreted function and predicate symbols are given, the satisfiability procedure just needs to determine the interpretation of the predicate symbols, including the syntactic equality. We limit the interpretation of the syntactic equality to a congruence relation, i.e. an equivalence relation $\approx$ that has the following two properties:

$$\begin{array}{l} t_1 \approx t'_1, \ldots, t_k \approx t'_k \text{ imply } f(t_1, \ldots, t_k) \approx f(t'_1, \ldots, t'_k) \text{ for all functions } f \\ t_1 \approx t'_1, \ldots, t_k \approx t'_k \text{ imply } p(t_1, \ldots, t_k) = p(t'_1, \ldots, t'_k) \text{ for all predicates } p \end{array} \tag{7}$$

If a Herbrand structure interprets the syntactic equality by a congruence relation, then we call it a *Herbrand congruence structure*. If the given quantifier-free formula only contains uninterpreted function and predicate symbols in addition to the syntactic equality, then it is satisfiable in an equality structure if and only if it is satisfiable in a Herbrand congruence structure. This correspondence follows from well-known results about equality structures and congruence relations (*cf. e.g.*, [8], section 3.1). We are thus able to use Herbrand structures for equality constraints over uninterpreted functions.

A second difficulty stems from the fact that the quantifier-free formula may contain interpreted as well as uninterpreted function and predicate symbols. An example is $\mathrm{a} = \mathrm{b} \wedge 10 \geq \texttt{discount}(\mathrm{a}) \wedge$

`discount`(b) ≥ 20. Whereas the predicate symbol $\geq$ represents the greater-than-or-equal-to relation over integers, the function symbol `discount` can be interpreted freely. Following the discussion above, we will interpret the ground terms `discount`(a) and `discount`(b) by themselves. However, these terms occur as arguments of the greater-than-or-equal-to relation that expects integer arguments. This means that we need two interpretations of the terms `discount`(a) and `discount`(b), namely in form of Herbrand terms and in form of integers. If we chose only the integer interpretation, then we would have difficulties to interpret terms such as `isGood`(`discount`(b)) which apply other uninterpreted functions to an integer-valued term.

Recent work of Baader and Ghilardi [1] on theory combination allows the coexistence of multiple interpretations of the same symbols within a mathematical structure. Baader and Ghilardi use a many-sorted first-order logic. They introduce copies f' of certain function symbols f and they ensure that f and f' range over different sorts S, S'. These sorts can be interpreted by different universes U, U'. Thanks to this, the copy f' can be interpreted differently than the origin f. Furthermore, Baader and Ghilardi introduce an explicit connection function from the sort S to the sort S'. This function connects a term $f(t)$ to its copy $f'(t')$ and represents a homomorphism between the two interpretations since it has to satisfy the formula $h(f(t)) = f'(h(t))$.

We apply this approach in the following way. We consider a many-sorted signature Ω, which consists of a set of sorts, a set of constant symbols, a set of function symbols, and a set of predicate symbols. Furthermore, we consider a quantifier-free formula ϕ formulated with the symbols in Ω. We distinguish a sub-signature Ω_1, which contains those sorts and symbols from Ω which have a fixed meaning for our problem. We consider a mathematical structure $\mathcal{A}_1$ that interprets the symbols in Ω_1. In order to permit unlimited equality reasoning over Herbrand terms, we will interpret all the symbols in Ω by a Herbrand congruence structure as explained above. This Herbrand structure will interpret the symbols in Ω_1 in a different way as specified by $\mathcal{A}_1$. We therefore introduce a copy Ω_2 of the signature Ω_1. We interpret the copies in the same way as the original sorts and symbols. Let $\mathcal{A}_2$ be the corresponding mathematical structure. We can safely combine a Herbrand congruence structure that interprets Ω with the structure $\mathcal{A}_2$ that interprets Ω_2 as the two signatures Ω and Ω_2 do not share any sort and symbol. Furthermore, we introduce connection functions h from the sorts of Ω_1 to the sorts of Ω_2. As a result, we obtain a combined signature Ω^* which consists of the original signature Ω, the copy Ω_2, and the connection functions h. We consider mathematical structures for Ω^* that interpret Ω by a Herbrand congruence structure, Ω_2 by $\mathcal{A}_2$, and the connection functions h by homomorphisms that ensure that the following properties are satisfied:

$$\begin{array}{l} h(f(t_1, \ldots, t_k)) = f'(h(t_1), \ldots, h(t_k)) \text{ for all function symbols } f \text{ and their copies } f' \\ .p(t_1, \ldots, t_k) \equiv p'(h(t_1), \ldots, h(t_k)) \text{ for all predicate symbols } p \text{ and their copies } p' \end{array} \quad (8)$$

We call the resulting structure an Ω-*Herbrand expansion* of $\mathcal{A}_1$ over (Ω_1, Ω_2). This notion provides our second transformation result: A quantifier-free formula over Ω is satisfiable in an equality structure that coincides with $\mathcal{A}_1$ on the symbols in Ω_1 if and only if there is an Ω-Herbrand expansion of $\mathcal{A}_1$ over (Ω_1, Ω_2). Whereas the original satisfiability problem consists in finding an equality structure that extends $\mathcal{A}_1$, we can now solve our problem by seeking a Herbrand structure which is linked to $\mathcal{A}_1$ via a homomorphism. In the next section, we describe how this problem can be solved by Constraint Programming.

4. Constraint Networks for Satisfiability Modulo Structures

Given a quantifier-free formula ϕ, we build a constraint network to determine a model for the formula if it is satisfiable. A constraint network consists of a set of variables and a set of constraints. Each variable x has a domain $D(x)$. Each constraint consists of a tuple $(x_1, \ldots, x_k)$ of variables and a k-ary relation R which is a subset of $D(x_1) \times \ldots \times D(x_k)$. A constraint network represents a constraint

satisfaction problem. This problem consists in finding an assignment of values to the variables such that the value of each variable belongs to its domain and all the constraints are satisfied. An assignment satisfies a constraint with variables $(x_1, \ldots, x_k)$ and relation R if the vector of values of those variables is an element of R.

We introduce variables for the unknown parts of the Ω-Herbrand expansion of $\mathcal{A}_1$. As the subterms t of ϕ are interpreted by themselves, we do not introduce variables for the subterms themselves. If f is an interpreted function symbol from $\mathcal{A}_1$, then we introduce the variables $\chi_{h(f(t_1,\ldots,t_k))}, \chi_{h(t_1)}, \ldots, \chi_{h(t_k)}$ for the unknown interpretation of the terms $h(f(t_1, \ldots, t_k))$, $h(t_1), \ldots, h(t_k)$. Furthermore, we introduce a variable χ_ψ with Boolean domain for each subformula ψ of ϕ. For equality propagation, we furthermore need a variable $\chi_{t_1=t_2}$ with Boolean domain for two arbitrary subterms[1] t_1 and t_2 of ϕ. We then express the following constraints:

1. *Logical constraints:* If ϕ contains a subformula of the form $\neg\psi$, then we introduce a binary constraint with variables $\chi_\psi, \chi_{\neg\psi}$ and the relation $R^\neg := \{(\mathrm{F},\mathrm{T}),(\mathrm{T},\mathrm{F})\}$. If ϕ contains a subformula of the form $\psi_1 \wedge \psi_2$ then we introduce a ternary constraint with variables $\chi_{\psi_1}, \chi_{\psi_2}, \chi_{\psi_1\wedge\psi_2}$ and the relation $R^\wedge := \{(\mathrm{F},\mathrm{F},\mathrm{F}),(\mathrm{F},\mathrm{T},\mathrm{F}),(\mathrm{T},\mathrm{F},\mathrm{F}),(\mathrm{T},\mathrm{T},\mathrm{T})\}$. If ϕ contains a subformula of the form $\psi_1 \vee \psi_2$ then we introduce a ternary constraint with variables $\chi_{\psi_1}, \chi_{\psi_2}, \chi_{\psi_1\vee\psi_2}$ and the relation $R^\vee := \{(\mathrm{F},\mathrm{F},\mathrm{F}),(\mathrm{F},\mathrm{T},\mathrm{T}),(\mathrm{T},\mathrm{F},\mathrm{T}),(\mathrm{T},\mathrm{T},\mathrm{T})\}$.

2. *Congruence constraints:* If ϕ contains two subterms $f(t_1, \ldots, t_k)$ and $f(t'_1, \ldots, t'_k)$, then we introduce a constraint with variables $\chi_{t_1=t'_1}, \ldots, \chi_{t_k=t'_k}, \chi_{f(t_1,\ldots,t_k)=f(t'_1,\ldots,t'_k)}$ and a relation that ensures that $\chi_{f(t_1,\ldots,t_k)=f(t'_1,\ldots,t'_k)}$ is true iff all the $\chi_{t_1=t'_1}, \ldots, \chi_{t_k=t'_k}$ are true. We introduce similar congruence constraints for subformulas $p(t_1, \ldots, t_k)$ and $p(t'_1, \ldots, t'_k)$ of ϕ. Furthermore, we have to ensure that the syntactic equality is interpreted by an equivalence relation. We ensure reflexivity by setting $\chi_{t_1=t_2}$ to T whenever t_1 and t_2 are the same terms. The symmetry and transitivity can be guaranteed by a congruence constraint for the equality predicate for all subterms t_1, t_2, t'_1, t'_2 of ϕ. This means we introduce a ternary implication constraint with variables $\chi_{t_1=t'_1}$,$\chi_{t_2=t'_2}$,$\chi_{(t_1=t'_1)=(t_2=t'_2)}$. The congruence constraints can be encoded efficiently in form of a global constraint that encapsulates the congruence closure algorithm of Nelson and Oppen [7].

3. *Homomorphism constraints:* If ϕ contains a subterm of the form $f(t_1, \ldots, t_k)$ and f is a function symbol in Ω_1 then we introduce a constraint on variables $\chi_{h(t_1)}, \ldots, \chi_{h(t_k)}$, $\chi_{h(f(t_1,\ldots,t_k))}$ that has the relation $R(f) := \{(v_1, \ldots, v_k, v) \mid v = f^{\mathcal{A}_1}(v_1, \ldots, v_k)\}$ where $f^{\mathcal{A}_1}$ is the interpretation of f in the structure $\mathcal{A}_1$. If ϕ contains a subformula of the form $p(t_1, \ldots, t_k)$ and p is a predicate symbol in Ω_1 then we introduce a constraint on variables $\chi_{h(t_1)}, \ldots, \chi_{h(t_k)}, \chi_{p(t_1,\ldots,t_k)}$ that has the relation $R(p) := \{(v_1, \ldots, v_k, v) \mid v = p^{\mathcal{A}_1}(v_1, \ldots, v_k)\}$ where $p^{\mathcal{A}_1}$ is the interpretation of p in the structure $\mathcal{A}_1$. For any two subterms t_1, t_2 of ϕ that both have a sort S from Ω_1, we introduce a constraint on the variables $\chi_{h(t_1)}, \chi_{h(t_2)}, \chi_{t_1=t_2}$ that ensures that $\chi_{t_1=t_2}$ is true whenever $\chi_{h(t_1)}$ and $\chi_{h(t_2)}$ are equal.

4. *Satisfaction constraint:* finally we introduce a constraint on χ_ϕ with relation $\{\mathrm{T}\}$ stating that ϕ must be true,

The resulting CSP has a solution if and only if there is a Ω-Herbrand expansion of $\mathcal{A}_1$ over (Ω_1, Ω_2). The CSP can be solved by different methods such as systematic search, interval propagation, propagation of modular arithmetics [4], and other methods dedicated to the mathematical structure under consideration.

[1] For the sake of brevity, we do not discuss the possible optimizations of this constraint network.

5. Conclusion

We explained the principles of how to integrate SMT reasoning capabilities into CP, which enables CP methods to test the satisfiability of quantifier-free formulas modulo given mathematical theories, as it is required by software verification. A major difficulty is the incorporation of equality reasoning over uninterpreted functions. We achieve this by a double representation in the sense of [1]. Equality propagation is done over Herbrand terms and domain propagation is done by constraints. We showed how to construct a hybrid constraint network coupling those propagations and illustrated the possible synergy obtained by this connection. A topic of future work is the incorporation of advanced forms of conflict learning. CP allows more compact forms of conflicts and thus promises faster interpolation methods [3].

Bibliography

[1] Franz Baader and Silvio Ghilardi. Connecting many-sorted theories. In *CADE*, pages 278–294, 2005.

[2] Harald Ganzinger, George Hagen, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. DPLL(T): Fast decision procedures. In *CAV*, pages 175–188, 2004.

[3] Ulrich Junker and Olivier Lhomme. Return of the JTMS: Preferences orchestrate conflict learning and solution synthesis. In *ECAI*, pages 123–127, 2006.

[4] Michel Leconte and Bruno Berstel. Extending a CP solver with congruences as domains for program verification. In *Proceedings of the 1st workshop on Constraints in Software Testing, Verification and Analysis*, pages 22–33, 2006.

[5] Zohar Manna and Calogero G. Zarba. Combining decision procedures. In *Formal Methods at the Crossroads. From Panacea to Foundational Support*, volume 2757 of *Lecture Notes in Computer Science*, pages 381–422. Springer, 2003.

[6] Greg Nelson and Derek C. Oppen. Simplification by cooperating decision procedures. *ACM Trans. Program. Lang. Syst.*, 1(2):245–257, 1979.

[7] Greg Nelson and Derek C. Oppen. Fast decision procedures based on congruence closure. *J. ACM*, 27(2):356–364, 1980.

[8] Michael M. Richter. *Logikkalküle*. Teubner, 1978.

[9] Cesare Tinelli. A DPLL-based calculus for ground satisfiability modulo theories. In *JELIA*, volume 2424 of *Lecture Notes in Computer Science*, pages 308–319, 2002.

[10] Cesare Tinelli. Cooperation of background reasoners in theory reasoning by residue sharing. *Journal of Automated Reasoning*, 30:1–31, 2003.

[11] Cesare Tinelli and Christophe Ringeissen. Unions of non-disjoint theories and combinations of satisfiability procedures. *Theoretical Computer Science*, 290:291–353, 2003.