Model Checking as Constraint Solving

Andreas Podelski

Max-Planck-Institut für Informatik
Im Stadtwald, 66123 Saarbrücken, Germany
podelski@mpi-sb.mpg.de

Abstract. We show how model checking procedures for different kinds of infinite-state systems can be formalized as a generic constraint-solving procedure, viz. the saturation under a parametric set of inference rules. The procedures can be classified by the solved form they are to compute. This solved form is a recursive (automaton-like) definition of the set of states satisfying the given temporal property in the case of systems over stacks or other symbolic data.

1 Introduction

In the large body of work on model checking for infinite-state systems (see e.g. [2–12, 15, 17, 20, 21, 24–26, 31–33]), we can distinguish two basic cases according to the infinite data domain for the program variables. In the first case, we have pushdown stacks or other 'symbolic' data implemented by pointer structures. These data are modeled by words or trees, and sets of states are represented by word automata or tree automata. In the second case, program variables range over reals or other numeric data, and sets of states are represented by arithmetic constraints. Accordingly, the respective model checking procedures operate on automata or on arithmetic constraints. Whereas they are essentially fixpoint procedures based on the predecessor or the successor operator in the second case, they seem to require ad-hoc reasoning techniques in the first case. In this paper, we show how all these procedures can be formalized as one generic constraint-solving procedure, viz. the saturation under a parametric set of inference rules; the procedures can be compared by the solved form they are to compute.

We will use constraints φ (such as $x = y + 1$ over the domain of reals or $x = a.y$ over the domain of words) in order to form Constraints Φ (such as $X = \{x \mid \exists y \in Y : \; x = y + 1\}$ or $X = \{x \mid \exists y \in Y : \; x = a.y\}$).[1] A specification of a transition system by a guarded-command program, together with a specification of a temporal property by a modal μ-calculus formula, translates effectively to a Constraint Φ such that the intended solution of Φ is the temporal property (i.e., the value of a variable X of Φ is the set of states satisfying the property). Model checking for the transition system and the temporal property amounts to solving that Constraint Φ. Solving Φ means transforming Φ into an equivalent Constraint Φ' in *solved form.*

In each Constraint solving step, a direct consequence under a logical inference rule is added as a conjunct to the Constraint ("saturation"). When no more new

[1] Notation: constraints φ are first-order, Constraints Φ are second-order.

conjuncts can be added, the Constraint is in solved form. This generic model checking procedure is parametrized by the set of inference rules.

The purpose of the solved form of a Constraint is to exhibit its set of solutions; technically, this means that known (and algorithmically more or less pleasing) tests for emptiness and for membership are applicable. (Strictly speaking, the test of the membership of the initial state of the system in this set is still part of the model checking procedure.)

We can compare the two cases of 'symbolic' vs. numeric data structures by the two basic cases of solved forms. The solved form is a recursive definition of sets of states in the first case, and a non-recursive definition (viz. a symbolic state enumeration) in the second case.

Our uniform definition of model checking and the classification according to solved forms allows us to contrast the two cases with each other. Model checking over the reals is 'harder' than over the words in the sense that solving a Constraint Φ over the reals means 'eliminating the recursion'. For example, the Constraint $X = \{x \mid \exists y \in X : \ x = y+1\} \cup \{0\}$ gets transformed into in its solved form $X = \{x \mid x \geq 0\}$; however, the Constraint $X = \{x \mid \exists y \in X : \ x = a.y\} \cup \{\varepsilon\}$ is already in solved form (representing the set $a^\star$ of all words $a.a \ldots a$ including the empty word ε). The first Constraint arises from the CTL formula EF(0) and the program consisting of the loop with the one instruction `x:=x-1`, the second from the CTL formula EF(ε) and the program with `pop(a,x)`.

Our technical setting uses concepts from logic programming. In order to address the reader who is not familiar with those concepts, we will first consider two concrete examples of infinite-state systems and use them to illustrate our approach (Sections 2 and 3). We then generally characterize the temporal property of a system in terms of a solution of a Constraint Φ (Section 4). In our formalization of the model checking procedure, the first step in the design of a concrete procedure is to define the solved form of a Constraint; we give a generic definition and its two instances corresponding to symbolic and numeric systems, respectively (Section 5). The second step is to instantiate the parameter of the generic Constraint-solving procedure, namely the set of inference rules, for concrete examples of infinite-state systems (Section 6).

2 Finite Automata as Infinite-State Systems

We will rephrase well-known facts about finite automata in order to illustrate our notions of constraints φ and Constraints Φ and the correspondence between a specific solution of Φ and a temporal property.

The notion of constraints φ. A finite automaton is given essentially by a *finite* edge-labeled directed graph $\mathcal{G}$. Some nodes of the graph $\mathcal{G}$ are marked as *initial* and some as *final.* We will write $Q = \{1, \ldots, n\}$ for its set of nodes and Σ for its set of edge labels. The graph defines a transition system. The states are pairs $\langle i, x \rangle$ consisting of a node i and a word x (i.e. x is an element of the free monoid $\Sigma^\star$ over the alphabet Σ). The edge $\langle i, a, j \rangle$ from node i to node j labeled by a defines the following state transitions:

state $\langle i, x\rangle$ can take a transition to state $\langle j, y\rangle$ if $\underline{x = a.y}$ holds.

The formula $x = a.y$ is an example of a constraint φ (over the variables x and y). The constraint is satisfied by words x and y if the first letter of x is a, and y is obtained from x by removing the first letter.

We note that a finite automaton defines a special case of an *infinite-state* transition system where each execution sequence is finite (infinite with self-loops on $\langle i, \varepsilon\rangle$). This view will be useful when we extend it to pushdown systems in Section 3.

The notion of Constraints Φ. It is well known that a finite automaton can be associated with a *regular system of equations*, which is a conjunction of equations of the form $p_i = [\varepsilon \cup] \bigcup_{\dots} a.p_j$ where the union ranges over all edges $\langle i, a, j\rangle$ from node i to node j; the 'empty word' ε is a member of the union if the node i is marked final.

A regular system of equations is an example of a Constraint Φ (over the variables $p_1, \dots, p_n$). Its variables range over sets of words. Being interested in the *least* solution, we can it equivalently as the conjunction of the following (superset) inclusions:

$$\begin{array}{ll} p_i \supseteq a.q_i & \text{(for each edge } \langle i, a, j\rangle\text{)}, \\ p_i \supseteq \varepsilon & \text{(if the node } i \text{ is marked final).} \end{array}$$

We next introduce a syntactic variant of the inclusion $p_i \supseteq a.q_i$ that makes explicit the role of constraints:

$$p_i \supseteq \{x \in \Sigma^\star \mid \exists y \in \Sigma^\star\ (\underline{x = a.y},\ y \in p_j)\}.$$

Identifying sets and unary predicates, we write the above inclusion in yet another syntactic variant, namely as a *clause* of a constraint data base or constraint logic program:[2]

$$p_i(x) \leftarrow\ x = a.y,\ p_j(y).$$

This notation leaves implicit the universal quantification of the variables x and y for each clause. Thus, the only free variables in the clause above are the *set variables* p_i and p_j. An inclusion of the form $p_i \supseteq \varepsilon$ translating the fact that the node i is marked final can be written as a special case of clause called *fact*:

$$p_i(x) \leftarrow\ x = \varepsilon.$$

Solutions of Constraints Φ and Temporal Properties. The correspondence between the least solution of a regular system of equations and the language recognized by a finite automaton is well known. If we note $[\![p_i]\!]$ the value of the set variable p_i in the least solution of the regular system (viz. the Constraint Φ),

[2] The close relationship between a clause $p(x) \leftarrow\ \dots$ and an inclusion $p \supseteq \dots$ underlines the fact that the "predicate symbols" p in clauses stand for second-order variables; they range over sets of (tuples of) data values.

then the recognized language is the union of all sets of words $[\![p_i]\!]$ such that the node i is marked initial.

The set $[\![p_i]\!]$ consists of all words accepted by the finite automaton when starting from the node i. Now, we only need to realize that acceptance is a special case of a temporal property, namely the reachability of an *accepting state* which is of the form $\langle j, \varepsilon \rangle$ for a final node j:

$x \in \Sigma^\star$ is accepted from node $i \in Q$ iff $\langle i, x \rangle \longrightarrow^\star \langle j, \varepsilon \rangle$ for a final node j.

We introduce *accept* as the symbol for the *atomic proposition* that holds for all accepting states. Then, the temporal property is specified by the formula $\mathrm{EF}(\mathit{accept})$ in the syntax of CTL, or by the formula $\mu X.(\mathit{accept} \vee \Diamond X)$ in the syntax of the modal μ-calculus. We can now rephrase the above correspondence as the following identity between the value $[\![p_i]\!]$ of the variable p_i in the least solution of the Constraint Φ and the temporal property:

$$x \in [\![p_i]\!] \;\text{ iff }\; \langle i, x \rangle \in \mathrm{EF}(\mathit{accept}). \tag{1}$$

In Section 4, we will generalize this correspondence (which holds also for other systems than just finite automata, other temporal properties than just reachability, and, accordingly, other solutions than just the least one).

3 Pushdown Systems

In the previous section, we have shown that a temporal property corresponds to a specific solution of a Constraint Φ for a special example. In that example, the Constraint Φ that is associated with the given transition system and the given temporal property is already in what we define to be the *solved form*. There is no reasonable way to simplify it any further; the tests for emptiness or membership are linear in the size of Φ (we carry over the standard algorithms for automata).

In contrast, in the case of *pushdown systems* to be introduced next, the associated Constraint Φ is not in solved form. The purpose of this section is to illustrate that model checking for pushdown systems (in the style of e.g. [6, 21]) is done by solving Φ, i.e. by bringing Φ into an equivalent solved form. Our example temporal property is again reachability.

If we view the word x in the second component of a state $\langle i, x \rangle$ of the transition system induced by a finite automaton as the representation of a *stack*, then each edge $\langle i, a, j \rangle$ defines a *pop* operation (at node i, if the top symbol is a, pop it and go to node j). It is now natural to extend the notion of transition graphs by allowing edges that define *push* operations (at node i, push a on top of the stack and go to node j). Formally, the edge $\langle i, !a, j \rangle$ from node i to node j labeled by a together with "!" defines the following state transitions:

state $\langle i, x \rangle$ can take a transition to state $\langle j, y \rangle$ if $a.x = y$ holds.

In contrast with the previous case, infinite execution sequences are possible in the more general kind of transition system.

We extend the notion of regular systems of equations accordingly. Each edge $\langle i, !a, j\rangle$ corresponds to an inclusion of the form

$$p_i \supseteq \{x \mid \exists y \in \Sigma^\star \; (\underline{a.x = y}, \; y \in p_j)\}$$

which we will write equivalently as the clause

$$p_i(x) \leftarrow \; a.x = y, \; p_j(y).$$

The new kind of clause contains constraints of a new form (the letter a is appended to the left of the variable x appearing in the head atom).

As before, each edge $\langle i, a, j\rangle$ translates to a clause $p_i(x) \leftarrow \; x = a.y, \; p_j(y)$. If we are again interested in the reachability of accepting states defined as above (wrt. a given set of nodes marked final), we translate each marking of a node j as final to a clause $p_j(x) \leftarrow \; x = \varepsilon$ (we say that these clauses express the atomic proposition *accept*).

The Constraint Φ whose least solution characterizes the temporal property EF(*accept*) in the same sense as in (1) is formed of the conjunction of the two kinds of clauses translating pop resp. push edges of the transition graph, and of the third kind of clauses expressing the atomic proposition *accept*.

We do not know of any algorithms for the tests for emptiness or membership that apply directly to Constraints containing the three kinds of conjuncts. We now define the *solved form* of a Constraint Φ as the smallest conjunction containing all conjuncts of Φ and being closed under the two inference rules below.

$$\left.\begin{array}{l} p(x) \leftarrow \; a.x = y, \; q(y) \\ q(x) \leftarrow \; q'(x) \\ q'(x) \leftarrow \; x = a.y, \; r(y) \end{array}\right\} \vdash \; p(x) \leftarrow \; r(x) \qquad (2)$$

$$\left.\begin{array}{r} p(x) \leftarrow \; q(x) \\ q(x) \leftarrow \; r(x) \end{array}\right\} \vdash \; p(x) \leftarrow \; r(x)$$

We note that the first of the two inference rules is obtained by applying logical operations to constraints φ over the logical structure of words. The conjunction of the two clauses $p(x) \leftarrow \; a.x = y, \; q(y)$ and $q(y) \leftarrow \; y = a.z, \; r(z)$ (the second clause is obtained by applying α-renaming to the clause $p(x) \leftarrow \; a.x = y, \; q(y)$ with the universally quantified variables x and y) yields the clause

$$p(x) \leftarrow \; \underline{a.x = y, \; y = a.z,} \, r(z).$$

The logical operations that we now apply are: forming the conjunction of constraints (here, $a.x = y \wedge y = a.z$), testing its satisfiability and transforming it into the equivalent constraint $x = z$.

Given a Constraint Φ, we define the Constraint Φ' as the part of Φ without conjuncts of the form $p_i(x) \leftarrow \; a.x = y, \; p_j(y)$ (i.e. without the clauses that translate push edges). If Φ is in solved form, then the least solution of Φ is equal to the least solution of Φ' (as can be shown formally). Hence, the tests of emptiness or membership for the least solution of Φ can be restricted to the Constraint Φ'. The conjuncts of Φ' are of the form $p_i(x) \leftarrow\ x = a.y,\ p_j(y)$ (translating edges of a finite automaton) or $p_j(x) \leftarrow\ x = \varepsilon$ (translating the marking of final nodes) or $p(x) \leftarrow\ r(x)$ (translating "ε-transitions of a finite automaton); thus, Φ' corresponds to a finite automaton with ε-transitions (for which linear algorithms are again well known).

The model checking procedure for pushdown systems is simply the *Constraint solving procedure* that we define as the iterative addition of new conjuncts obtained by the inference step above (until no more new conjuncts can be added).

The *cubic* complexity bound for this procedure, which can be described by inference rules, can be inferred directly using the techniques of McAllester [27]; these techniques work by transferring known complexity bounds for deductive database queries.

Atomic Propositions specified by Regular Sets. It should be clear by now how we form the Constraint Φ corresponding to the temporal property $\mathrm{EF}(ap_{\mathcal{L}})$ where the atomic proposition $ap_{\mathcal{L}}$ is given by a family $\mathcal{L} = (L_i)_{i\in Q}$ of regular word languages $L_i \subseteq \Sigma^\star$ for each node $i \in Q$; the atomic proposition holds for all states $\langle i, w\rangle$ where $w \in L_i$ (the atomic proposition *accept* is the special case where all languages L_i consist of the empty word ε). Each set L_i is recognized by a finite automaton with the set of nodes Q_i such that $Q \cap Q_i = \{i\}$ and L_i is the set of all words accepted from node i (all nodes different from i are new). We translate the finite automaton into a Constraint Φ_i such that value of the variable p_i in its least solution is L_i (all other variables are new). We now form the Constraint Φ as the conjunction of all clauses translating pop and push edges of the transition graph of the pushdown system and of all Constraints Φ_i.

Automata and Guarded Command Programs. The purpose of Sections 2 and 3is to convey the intuition behind the general framework through two concrete examples. Since the general setting uses guarded command programs, we need to relate those with transition graphs of finite automata. An edge $\langle i, a, j\rangle$ from node i to node j labeled a specifies the same transition steps as the following guarded command:

$$z = i,\ head(x) = a \ [\!]\ z := j,\ x := tail(x)$$

The guarded command program obtained by the translation of a transition graph has the program variables z (ranging over the finite set Q of program points) and x (ranging over words denoting the stack contents).

The guarded command above can be represented in another form, where primed variables stand for the value of the variable after the transition step. Here, the guard constraint $\alpha(z, x)$ and the action constraint $\gamma(z, x, z', x')$ are logical formulas (we indicate their free variables in parenthesis; the variable y is quantified existentially at the outset of the guarded command).

$$\underbrace{z = i,\ x = a.y}_{\alpha(z,x)}\ [\!]\ \underbrace{z' = j,\ x' = y}_{\gamma(z,x,z',x')}$$

In the setting of Section 4, the guarded command will be translated to the conjunct $p(z,x) \leftarrow z = i,\ x = a.y,\ z' = j,\ x' = y,\ p(z',x')$ of the Constraint $\varPhi$; here p is a generic symbol for the (only one) variable of $\varPhi$. This translation is equivalent to the one given in Section 2, in the sense that $p_i(x)$ is equivalent to $p(i,x)$.

4 Temporal Properties and Constraints $\varPhi$

Given a specification of a transition system (possibly with an infinite state space) in form of a guarded command program $\mathcal{P}$ and of a temporal property in form of a modal μ-calculus formula $\varDelta$, we will construct a Constraint $\varPhi$ whose solution (least, greatest or intermediate, according to the quantifier prefix of $\varDelta$) is the temporal property.

The program $\mathcal{P}$ is a set of guarded commands $\alpha \ [\!] \ \gamma$; the free variables of the guard constraint α are the program variables $x_1, \dots, x_n$; its existentially quantified variables may be 'shared' with the action constraint γ which is a conjunction of equalities $x' = e(x_1, \dots, x_n)$ with an expression e for each program variable x (we omit any further formalization). We use $\mathbf{x}$ for the tuple $x_1, \dots, x_n$.

The formula $\varDelta$ consists of a sequence of quantifiers μX or νX applied to a set of declarations of the form $X = \delta$, where the language of expressions δ is defined as follows. We assume the usual restrictions for the (closed, well-formed) formula $\varDelta$ in positive normal form (negation is pushed to the atomic propositions; as usual, we close the set of atomic propositions under negation).

$$\delta \ \equiv \ ap \mid X_1 \vee X_2 \mid X_1 \wedge X_2 \mid \Diamond X \mid \Box X$$

The Constraint $\varPhi$ is defined as the conjunction of the following clauses, where p_X is a new symbol for a variable of $\varPhi$ (for each X occurring in $\varDelta$).

$$\varPhi \ \equiv \ \left\{ \begin{array}{lllll} p_X(\mathbf{x}) \leftarrow ap & \text{for } X = ap & \text{in } \varDelta, & \\ p_X(\mathbf{x}) \leftarrow p_{X_i}(\mathbf{x}) & \text{for } X = X_1 \vee X_2 & \text{in } \varDelta, & i = 1,2 \\ p_X(\mathbf{x}) \leftarrow p_{X_1}(\mathbf{x}),\ p_{X_2}(\mathbf{x}) & \text{for } X = X_1 \wedge X_2 & \text{in } \varDelta, & \\ p_X(\mathbf{x}) \leftarrow \alpha,\ \gamma,\ p_{X'}(\mathbf{x}') & \text{for } X = \Diamond X' & \text{in } \varDelta, & \alpha \ [\!] \ \gamma \text{ in } \mathcal{P} \\ p_X(\mathbf{x}) \leftarrow \bigwedge_j \forall \dots \alpha_j,\ \gamma_j,\ p_{X'}(\mathbf{x_j}) & \text{for } X = \Box X' & \text{in } \varDelta & \end{array} \right\}$$

In the last kind of clause, the conjunction $\bigwedge_j$ ranges over all guarded commands of the program $\mathcal{P}$; the primed variables in each guarded command are renamed apart (from $\mathbf{x}'$ to $\mathbf{x}'_\mathbf{j}$; the renamed version of $\alpha \ [\!] \ \gamma$ is $\alpha_j \ [\!] \ \gamma_j$), and the universal quantifier ranges over all variables other than $\mathbf{x}$ and $\mathbf{x_j}$.[3]

We assume that the quantifier prefix of $\varDelta$ is $\xi_1 X_1 \dots \xi_m X_m$ where ξ_i is either μ or ν; i.e., the formula is of the form

$$\varDelta \ \equiv \ \xi_1 X_1 \dots \xi_m X_m \ \{X_i = \delta_i \mid i = 1, \dots, m\}.$$

[3] The universal quantification in the body of clauses goes beyond the usual notion of Horn clauses that is used in related approaches (see e.g. [14, 16, 18, 23, 29]). It is needed when successor values can be chosen nondeterministically. The direct-consequence operator $T_\varPhi$ is still defined.

Then, the free variables of the Constraint Φ form the tuple $\langle p_{X_1}, \ldots, p_{X_m} \rangle$, and a solution of Φ can be represented as a tuple $\langle S_1, \ldots, S_m \rangle$ of sets of states (states are value tuples $\langle v_1, \ldots, v_n \rangle$ for the program variables $\langle x_1, \ldots, x_n \rangle$).

The intended solution $\langle [\![p_{X_1}]\!], \ldots, [\![p_{X_m}]\!] \rangle$ of Φ according to the quantifier prefix of Δ is defined as the fixpoint of the *logical consequence operator* T_Φ,

$$\langle [\![p_{X_1}]\!], \ldots, [\![p_{X_m}]\!] \rangle \;=\; \xi_1 p_{X_1} \ldots \xi_m p_{X_m} \; T_\Phi(\langle p_{X_1}, \ldots, p_{X_m} \rangle).$$

In order to define alternating fixpoints, we reformulate the usual logical consequence operator for constraint logic programs as a fixpoint operator T_Φ over tuples $\langle S_1, \ldots, S_m \rangle$ of sets of states (see also [13]); its application is defined by $T_\Phi(\langle S_1, \ldots, S_m \rangle) = (\langle S_1', \ldots, S_m' \rangle)$ where

$$S_j' = \{ \langle v_1, \ldots, v_n \rangle \mid \Phi \cup p_{X_1}(S_1) \cup \ldots \cup p_{X_m}(S_m) \vdash p_{X_j}(\langle v_1, \ldots, v_n \rangle) \}.$$

Here, $p_{X_k}(S_k)$ stands for the conjunction of formulas $p_{X_k}(\langle w_1, \ldots, w_n \rangle)$ where $\langle w_1, \ldots, w_n \rangle \in S_k$ (we always implicitely use the identification of sets and unary predicates p_X). The symbol $\vdash$ here refers to one single step of logical inference (the *modus ponens* rule, essentially). When using clauses with constraints φ (here, conjunctions $\alpha \wedge \gamma$ of guard and action constraints), the inference step is taken wrt. the logical structure for the specific constraints (the domain of reals, the monoid of words, etc.). The alternating fixpoints are well-defined due to our assumption that Δ is well-formed; we omit any further formal details.

Theorem 1. *Given a specification of a transition system in form of a guarded command program $\mathcal{P}$ and of a temporal property in form of a modal μ-calculus formula Δ, the set of all states satisfying the temporal property is the value $[\![p_{X_1}]\!]$ of the variable p_{X_1} under the solution of Φ specified by the quantifier prefix of Δ.*

Proof. The proof works by a logical formulation of the construction of an alternating tree automaton as in [1], expressing the next-state relation used there by the first-order constraints $\alpha \wedge \gamma$ that correspond to the guarded commands. These constraints are inserted into the tree-automaton clauses (see Figure 1) without changing their logical meaning; the clauses are then transformed into the form as they occur in Φ; again, one can show this transformation logically correct. $\square$

Reachability for Guarded Command Programs. As an example, we consider the most simple (and practically most important) temporal property, i.e. reachability, which is specified by the CTL formula EF(ap) or the modal μ-calculus formula

$$\Delta \;\equiv\; \mu X \mu X_1 \mu X_2 \left\{ \begin{array}{l} X \;= X_1 \vee X_2, \\ X_1 = ap, \\ X_2 = \Diamond X \end{array} \right\}.$$

Given Δ and a guarded command program $\mathcal{P}$ whose program variables form the tuple $\mathbf{x}$, we form the Constraint

$$\Phi \;\equiv\; \left\{ \begin{array}{l} p_X(\mathbf{x}) \leftarrow p_{X_1}(\mathbf{x}) \\ p_X(\mathbf{x}) \leftarrow p_{X_2}(\mathbf{x}) \\ p_{X_1}(\mathbf{x}) \leftarrow ap \end{array} \right\} \cup \{ p_{X_2}(\mathbf{x}) \leftarrow \alpha, \; \gamma, \; p_X(\mathbf{x}') \mid \; \alpha \, [\!] \, \gamma \text{ in } \mathcal{P} \}$$

Following Theorem 1, we deduce that the set of states satisfying the temporal property Δ is the value $[\![p_X]\!]$ of the variable p_X under the least solution of Φ. The least solution is the fixpoint $\mu p_X \mu p_{X_1} \mu p_{X_2}\ T_\Phi(\langle p_X, p_{X_1}, p_{X_2}\rangle))$.

Equivalently, the value $[\![p_X]\!]$ is defined by the least solution of Φ' (defined as $\mu p_X\ T_{\Phi'}(p_X))$, where Φ' is a simplified version of Φ defined by

$$\Phi' \equiv \{p_X(\mathbf{x}) \leftarrow ap\} \cup \{p_X(\mathbf{x}) \leftarrow \alpha,\ \gamma,\ p_X \mid \alpha \,[\!]\, \gamma \text{ in } \mathcal{P}\}.$$

Inevitability. The greatest solution of the Constraint

$$\Phi'' \equiv \{p_X(\mathbf{x}) \leftarrow ap,\ \alpha,\ \gamma,\ p_X \mid \alpha \,[\!]\, \gamma \text{ in } \mathcal{P}\}$$

is the property defined by the CLT formula EG(ap) or the modal μ-calculus formula $\nu X\ \{ap \wedge \Diamond X\}$, which is the dual of the simplest case of a liveness property, namely inevitability.

Clark's completion. The conjunction of all clauses $p(\mathbf{x}) \leftarrow body_i$ defining the predicate p is, in fact, only a syntactic sugaring for the formula that expresses the logical meaning correctly, namely the equivalence (here, the existential quantification ranges over all variables but the ones in $\mathbf{x}$)

$$p(\mathbf{x}) \leftrightarrow \bigvee_i \exists \ldots\ body_i.$$

The two forms are equialent wrt. the least solution. The greatest solution, however, refers to the second form with equivalences (the so-called *Clark's completion*), or, equivalently, to the greatest fixpoint of T_Φ. All intermediate solutions are defined by intermediate fixpoints of T_Φ.

5 Solved Forms for Constraints Φ

We first give a general definition of solved forms (for all cases of data structures) that is only parametrized by a subclass of *solved-form clauses*. We will then instantiate the definition by specifying concrete subclasses for the two basic cases of data structures.

The idea behind solved-form clauses is that they form a fragment of monadic second-order logic (over the respective data structures) for which procedures implementing tests for emptiness and membership are available. Note that it makes sense here to admit also procedures that are possibly non-terminating (but hopefully practically useful); e.g., one may trade this possibility with a termination guarantee for the constraint solving procedure.

Definition 1 (General Solved Form). *Given a class of* solved-form clauses*, a Constraint Φ is said to be in solved form if it is equivalent to the Constraint Φ' that consists of all solved-form clauses of Φ.*

As always, the equivalence between Constraints Φ and Φ' refers to the solution specified by a given fixpoint (least, greatest, ..., possibly alternating).

$p(x) \leftarrow x = a.y,\ p(y)$	finite automaton (on words)
$p(x) \leftarrow x = \varepsilon$	
$p(x) \leftarrow x = f(y,z),\ q(y),\ r(z)$	tree automaton
$p(x) \leftarrow x = a$	
$p(x) \leftarrow x = f(y,z),\ y = z,\ q(y),\ r(z)$	"equality on brother terms"
$p(x) \leftarrow q(x),\ r(x)$	alternation
$p(x) \leftarrow \neg q(x)$	negation
stratified	"weak alternating"
$\ldots \nu p \mu q \ldots$	automata with parity condition

Fig. 1. Automaton clauses and corresponding notions of automata

Definition 2 (Solved Form (1) for Words). *The class of solved-form clauses defining the solved form of Constraints Φ for systems over words consists of all clauses of one of the three forms:*

$$\begin{aligned} p(x) &\leftarrow x = a.y,\ q(y), \\ p(x) &\leftarrow x = \varepsilon, \\ p(x) &\leftarrow q(x). \end{aligned}$$

In Section 2, we have seen that the class of clauses defined above corresponds to the notion of a finite automaton. (As noted in Section 4, we can write these clauses using a generic predicate symbol p, i.e. writing $p(i,x)$ instead of $p_i(x)$.)

Generally, we say that a class of clauses (then called *automaton clauses*) corresponds to a given class of automata if each Constraint Φ consisting of such clauses can be translated into an equivalent automaton in the class (i.e. such that the recognized languages and the values under the specific solution of the Constraint Φ coincide). This kind of correspondence holds between several notions of automata and their 'corresponding' form of *automaton clauses* (see Figure 1). In each case, one can define a new class of solved-form clauses.

We have usually in mind automata on finite words or finite trees, but one can consider also infinite objects (e.g. in order to model cyclic pointer structures), terms in algebras other than the tree algebra, certain forms of graphs etc..

The results in [13] imply the correspondence between fixpoints and acceptance conditions for Constraints Φ over words and trees; i.e., every alternating fixpoint specifying a solution of a conjunction of Horn clauses corresponds to a specific acceptance condition (on infinite runs, i.e. based on the *parity condition*) for the 'corresponding' automaton over words resp. trees. Thus, if Φ is in solved form, algorithms implementing emptiness and membership tests are known.

The correspondence between fixpoints and acceptance conditions for Constraints Φ generalizes from the domain of words or trees to any constraint domain. However, emptiness and membership are undecidable for (any interesting subclass of) *recursive* Constraints Φ over the reals (i.e. with conjuncts of the form $p(\mathbf{x}) \leftarrow \varphi,\ p(\mathbf{x}'))$, even if we consider the *least* solution only.

This leads to the following definition of *non-recursive* solved-form clauses. The definition means that the solution of a Constraint in solved form (2) is essentially presented as a finite union of infinite sets of states, these sets being denoted by constraints φ (whose free variables form the tuple $\mathbf{x} = \langle x_1, \dots, x_n \rangle$).

Definition 3 (Solved Form (2), for Reals). *The class of solved-form clauses defining the solved form of Constraints Φ for systems over reals consists of all clauses of the form (where φ is a constraint over reals)*

$$p(\mathbf{x}) \leftarrow \varphi.$$

Definition 3 is parametric wrt. a given notion of constraints φ; it can be reformulated for other domains such as the integers or the rationals. The free variables of the constraints φ correspond to the program variables; some program variables (the control variables) range over a finite domain of program locations; we can choose that domain as a finite subset of the constraint domain. We have in mind linear or non-linear arithmetic constraints over the reals or over the integers, as they are used in model checkers for network protocols with counters, timed, linear-hybrid or hybrid systems, etc.. The class of constraints is closed under conjunction and existential quantification; it may or may not be closed under disjunction. It comes with a test of satisfiability ("$\models \exists \mathbf{x}\ \varphi(\mathbf{x})$?"), entailment ("$\models \varphi(\mathbf{x}) \rightarrow \varphi'(\mathbf{x})$?") and satisfaction for a given tuple of values of the constraint domain ("$\models \varphi(v_1, \dots, v_n)$?").

6 Solving Constraints Φ

A *Constraint solving procedure* can be defined generically wrt. to a given set of inference rules: iteratively add direct consequences as conjuncts; start from the Constraint Φ constructed for the model checking problem (viz. for the program $\mathcal{P}$ and the temporal formula Δ); terminate when no more *new* consequences can be inferred. It is part of the inference system to specify whether 'new' refers to the semantics of all of Φ or the semantics or the syntax of one of the conjuncts of Φ.

Thus, a model checking procedure is specified by a set of inference rules. These define transformations of constraints into equivalent constraints (as always in this text, equivalence refers to the intended solutions).

The *soundness* of the model checking procedure holds by definition. The *completeness* (the solution of the subpart Φ' contains already the solution of the solved form of Φ) is trivial for the inference rule (3) given below for systems over reals. It requires more intricate reasoning in the case of the inference rule (2) for pushdown systems.

A possible alternative to ensure completeness is to check whether the solved-form clauses subsume all the other ones. To our knowledge, this alternative has not yet been explored in practical systems.

In the remainder of this section we show that one can express the main ideas of the model checking procedures for different examples of infinite-state systems by means of inference rules (the parameter of our generic procedure).

In Section 3, we have seen the set of inference rules (2) for *pushdown systems.* In the case of more general temporal properties (e.g. expressed with nested fixpoints), the inference rules must be extended to memorize the priority of the fixpoint operator for the 'eliminated' predicate q; this memorization technique is described in [13]. The solved form is here an alternating Rabin automaton with ε-transitions. Applying known results for those automata, we obtain the complexity result of [33] in a direct way.

In passing, we observe that the model checking problem for the subclass of *finite-state automata* viewed as infinite-state systems (with pop operations only) has the same complexity.

Real-Valued Systems. The set of inference rules that accounts for the symbolic model checking procedure for system over reals or integers (based on backward analysis) consists of the following rule. (Forward analysis is accounted for differently; see e.g. [22, 18]).

$$\left.\begin{array}{l} p(\mathbf{x}) \leftarrow \alpha,\ \gamma,\ p(\mathbf{x}') \\ p(\mathbf{x}) \leftarrow \varphi \end{array}\right\} \vdash\ p(\mathbf{x}) \leftarrow \varphi[\mathbf{x}'/\mathbf{x}],\ \alpha,\ \gamma \tag{3}$$

The application of the inference rule includes the test of satisfiability of the constraint $\varphi[\mathbf{x}'/\mathbf{x}] \wedge \alpha \wedge \gamma$. Note our conventions about notation: conjuncts in clauses are separated by commas; the constraint $\varphi[\mathbf{x}'/\mathbf{x}]$ is obtained by renaming the tuple $\mathbf{x}$ (of free variables of φ) to $\mathbf{x}'$ (recall that the free variables in the guard constraint α and the action constraint γ form the tuples $\mathbf{x}$ and $\langle x_1, \ldots, x_n, x_1', \ldots, x_n' \rangle$, respectively).

Meta-Transitions. The ultimate goal of a constraint solving procedure is to add enough 'interesting' conjuncts, i.e. conjuncts forming the part Φ' which is relevant according to the definition of a solved form (i.e., the part to which the emptiness or memberships tests refer). Other conjuncts may be inferred and added, however, in order to *accelerate* the inference of 'interesting' conjuncts. To give a simple example, the guarded command $z = \ell,\ x \geq 0 \ [\!]\ z' = \ell,\ x' = x + 1$ (an increment loop for the integer variable x at the program location ℓ) corresponds to the clause

$$p_X(z, x) \leftarrow\ z = \ell,\ x \geq 0,\ z' = \ell,\ x' = x + 1,\ p_X(z, x'). \tag{4}$$

This clause entails the clause (where the variable k is implicitely existentially quantified in $x' = x + k$)

$$p_X(z, x) \leftarrow\ z = \ell,\ x \geq 0,\ z' = \ell,\ x' = x + k,\ p_X(z, x'). \tag{5}$$

Boigelot [2] uses Presburger arithmetic in order to derive guarded commands called *meta-transitions* corresponding to clauses such as (5). One application of the inference rule (3) to the clause (5) yields a clause that subsumes all clauses obtained by its application to the clause in (4) in an infinite iteration.

Queue Systems. A system with one queue is similar to a pushdown system in that a dequeue operation corresponds to a pop operation and, hence, can be translated to a clause of the same form (for better legibility, we return to our notation of Section 3). A guarded command specifying an enqueue operation, however, is translated to a clause with the constraint $x.a = y$ expressing the concatenation to the right of the word x modeling the queue contents.

$$\begin{array}{ll} p(x) \leftarrow\ x = a.y,\ q(y) & \text{(dequeue).} \\ p(x) \leftarrow\ x.a = y,\ q(y) & \text{(enqueue).} \end{array}$$

Model checking for systems with queues is a topic of ongoing research; see e.g. [2, 3, 7]. One possible (quite insufficient) inference rule is

$$\left.\begin{array}{l} p(x) \leftarrow\ x.a = y,\ q(y) \\ q(x) \leftarrow\ x = a.y,\ r(y) \\ r(x) \leftarrow\ x = \varepsilon \end{array}\right\} \vdash\ p(x) \leftarrow\ x = \varepsilon.$$

This rule can be generalized to any set of clauses specifying a finite automaton that accepts only words ending with the letter a (here, $q'_1, \ldots, q'_n$ are new).

$$\left.\begin{array}{r} p(x) \leftarrow \underline{x.a = y},\ q_1(y) \\ q_1(x) \leftarrow x = b_1.y,\ q_2(y) \\ \vdots\quad \\ q_{n-1}(x) \leftarrow x = b_{n-1}.y,\ q_n(y) \\ q_n(x) \leftarrow \underline{x = a.y},\ r(y) \\ r(x) \leftarrow x = \varepsilon \end{array}\right\} \vdash \left\{\begin{array}{r} p(x) \leftarrow \underline{x = y},\ q'_1(y) \\ q'_1(x) \leftarrow x = b_1.y,\ q'_2(y) \\ \vdots\quad \\ q'_{n-1}(x) \leftarrow x = b_{n-1}.y,\ q'_n(y) \\ q'_n(x) \leftarrow \underline{x = \varepsilon} \end{array}\right.$$

This schematic inference rule is used by Boigelot and Godefroid (see [2, 3]).

7 Related Work and Conclusion

Since a fixpoint equation is a constraint over sets of states, the existence of a characterization of a temporal property by a second-order Constraint is not surprising. Our characterization (in Theorem 1) using clausal syntax with first-order constraints seems to be the first one, however, that is useful for *symbolic* model checking (where 'symbolic' refers to first-order constraints) for a very general class of (nondeterministic) infinite-state systems. The characterization holds for the full modal μ-calculus and for arbitrary guarded command programs and is thus more general than in related approaches [1, 16, 14, 18, 19, 21, 23, 29] (see also Footnote 3). In the case of finite-state systems, Φ is the alternating automaton constructed by Kupfermann, Vardi and Wolper [1] in a logical formulation; we generalize that construction and its extensions for pushdown systems [21] and for timed automata [19].

We have formalized model checking as a generic constraint-solving procedure that is parametrized by logical inference rules. This allows us to classify the two basic cases by the solved form, and to express the main ideas of model checking procedures concisely.

Our formalization provides a formal support for proving a model checking procedure correct (by checking soundness and completeness of the inference rules, possibly employing proof-theoretic (as opposed to graph-theoretic) techniques) and for analyzing its complexity (e.g. by writing the inference rules as bottom-up logic programs and applying syntactic criteria as suggested by McAllester [27]; see e.g. [20]).

Regarding future work, we note that the emptiness test or the test whether an initial state $\langle v_1, \ldots, v_n \rangle$ is a member of $[\![p_X]\!]$ (in the notation of Theorem 1) can be implemented by applying a *refutation* procedure to the conjunction of the formula $\neg\exists \mathbf{x} p_X(\mathbf{x})$ (or of the formula $\neg p_X(\langle v_1, \ldots, v_n \rangle)$, respectively) with the Constraint Φ. This is related to the procedures e.g. in [14,26,28,32,30]. Hybrid forms combining that procedure and the one given in Section 6 and the relation to ordered resolution have to be explored.

Acknowledgement. We thank Harald Ganzinger for discussions and for his suggestion of a general solved form, and Giorgio Delzanno and Jean-Marc Talbot for comments.

References

1. O. Bernholtz, M. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. In D. Dill, editor, *CAV 94: Computer-aided Verification*, LNCS, pages 142–155. Springer, 1994.
2. B. Boigelot. *Symbolic Methods for Exploring Infinite State Spaces.* PhD thesis, University of Liège, May 1998.
3. B. Boigelot and P. Godefroid. Symbolic verification of communications protocols with infinite state spaces using QDDs. In *Proceedings of CAV'96*, volume 1102 of *LNCS*, Berlin, 1996. Springer.
4. B. Boigelot and P. Wolper. Symbolic Verification with Periodic Sets. In D. L. Dill, editor, *Proceedings of CAV'94: Computer-aided Verification*, volume 818 of *LNCS*, pages 55–67. Springer, 1994.
5. B. Boigelot and P. Wolper. Verifying Systems with Infinite but Regular State Space. In A. J. Hu and M. Y. Vardi, editors, *Proceedings of CAV'98: Computer-aided Verification*, volume 1427 of *LNCS*, pages 88–97. Springer, 1998.
6. A. Bouajjani, J. Esparza, and O. Maler. Reachability Analysis of Pushdown Automata: Application to Model Checking. In A. W. Mazurkiewicz and J. Winkowski, editors, *CONCUR '97: Concurrency Theory,*, volume 1243 of *LNCS*, pages 135–150. Springer, 1997.
7. A. Bouajjani and P. Habermehl. Symbolic reachability analysis of FIFO-channel systems with nonregular sets of configuarations. *Theoretical Computer Science*, 221:211–250, 1999.
8. T. Bultan, R. Gerber, and W. Pugh. Symbolic Model Checking of Infinite-state Systems using Presburger Arithmetics. In O. Grumberg, editor, *Proceedings of CAV'97: Computer-aided Verification*, volume 1254 of *LNCS*, pages 400–411. Springer, 1997.

9. O. Burkart, D. Caucal, F. Moller, and B. Steffen. Verification on infinite structures. In S. S. J. Bergstra, A. Ponse, editor, *Handbook of Process Algebra.* Elsevier Science Publisher B.V., 1999. to appear.
10. O. Burkart and B. Steffen. Composition, decomposition and model checking optimal of pushdown processes. *Nordic Journal of Computing*, 2(2):89–125, 1995.
11. O. Burkart and B. Steffen. Model–checking the full modal mu–calculus for infinite sequential processes. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *International Colloquium on Automata, Languages, and Programming (ICALP'97)*, volume 1256 of *LNCS*, pages 419–429. Springer, 1997.
12. W. Chan, R. Anderson, P. Beame, and D. Notkin. Combining Constraint Solving and Symbolic Model Checking for a Class of Systems with Non-linear Constraints. In O. Grumberg, editor, *Proceedings of the Ninth Conference on Computer Aided Verification (CAV'97)*, volume 1254 of *LNCS*, pages 316–327. Springer, 1997.
13. W. Charatonik, D. McAllester, D. Niwinski, A. Podelski, and I. Walukiewicz. The Horn mu-calculus. In V. Pratt, editor, *Proceedings of LICS'98: Logic in Computer Science*, pages 58–69. IEEE Computer Society Press, 1998.
14. W. Charatonik, S. Mukhopadhyay, and A. Podelski. The Sμ-calculus. Submitted to this conference.
15. W. Charatonik and A. Podelski. Set-based analysis of reactive infinite-state systems. In B. Steffen, editor, *Proceedings of TACAS'98: Tools and Algorithms for the Construction and Analysis of Systems*, volume 1384 of *LNCS*, pages 358–375. Springer, 1998.
16. W. Charatonik and A. Podelski. Set-based analysis of reactive infinite-state systems. In B. Steffen, editor, *Fourth International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1384 of *LNCS*, pages 358–375, Lisbon, Portugal, March-April 1998. Springer-Verlag.
17. G. Delzanno and A. Podelski. Model checking in CLP. In R. Cleaveland, editor, *Proceedings of TACAS'99: Tools and Algorithms for the Construction and Analysis of Systems*, volume 1579 of *LNCS*, pages 223–239. Springer, 1999.
18. G. Delzanno and A. Podelski. Model Checking in CLP. In R. Cleaveland, editor, *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'99)*, volume 1579 of *LNCS*, pages 223–239, Amsterdam, The Netherlands, January 1999. Springer-Verlag.
19. M. Dickhöfer and T. Wilke. Timed alternating tree automata: The automata-theoretic solution to the TCTL model checking problem. In J. Widermann, P. van Emde Boas, and M. Nielsen, editors, *ICALP: Automata, Languages and Programming*, volume 1644 of *LNCS*, pages 281–290. Springer-Verlag, 1999.
20. J. Esparza and A. Podelski. Efficient algorithms for pre* and post* on interprocedural parallel flow graphs. In T. Reps, editor, *Proceedings of POPL'00: Principles of Programming Languages*, pages 1–11. IEEE, ACM Press, January 2000.
21. A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems. Electronic Notes in Theoretical Computer Science 9, www.elsevier.nl/locate/entcs, 13 pages, 1997.
22. L. Fribourg and J. Richardson. Symbolic Verification with Gap-order Constraints. Technical Report LIENS-93-3, Laboratoire d'Informatique, Ecole Normale Superieure, Paris, 1996.
23. G. Gottlob, E. Grädel, and H. Veith. Datalog LITE: Temporal versus deductive reasoning in verification. Technical Report DBAI-TR-98-22, Institut für Informationssysteme, Technische Universität Wien, December 1998.

24. S. Graf and H. Saidi. Verifying invariants using theorem proving. In *Proceedings of CAV'96: Computer-aided Verification*, volume 1102 of *LNCS*, pages 196–207. Springer, 1996.
25. T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: a Model Checker for Hybrid Systems. In O. Grumberg, editor, *Proceedings of CAV'97: Computer Aided Verification*, volume 1254 of *LNCS*, pages 460–463. Springer, 1997.
26. K. G. Larsen, P. Pettersson, and W. Yi. Compositional and symbolic model checking of real-time systems. In *Proceedings of the 16th Annual Real-time Systems Symposium*, pages 76–87. IEEE Computer Society Press, 1995.
27. D. McAllester. On the complexity analysis of static analyses. In A. Cortesi and G. Filé, editors, *SAS'99: Static Analysis Symposium*, volume 1694 of *LNCS*, pages 312–329. Springer, 1999.
28. S. Mukhopadhyay and A. Podelski. Model checking in Uppaal and query evaluation. In preparation.
29. Y. Ramakrishna, C. Ramakrishnan, I. Ramakrishnan, S. Smolka, T. Swift, and D. Warren. Efficient model checking using tabled resolution. In *Computer Aided Verification (CAV'97)*, volume 1254 of *LNCS*. Springer-Verlag, June 1997.
30. Y. S. Ramakrishnan, C. R. Ramakrishnan, I. V. Ramakrishnan, S. A. Smolka, T. Swift, and D. S. Warren. Efficient Model Checking using Tabled Resolution. In O. Grumberg, editor, *Proceedings of CAV'97: Computer-aided Verification*, volume 1254 of *LNCS*, pages 143–154. Springer, 1997.
31. T. R. Shiple, J. H. Kukula, and R. K. Ranjan. A Comparison of Presburger Engines for EFSM Reachability. In A. J. Hu and M. Y. Vardi, editors, *Proceedings of CAV'98: Computer-aided Verification*, volume 1427 of *LNCS*, pages 280–292. Springer, 1998.
32. H. B. Sipma, T. E. Uribe, and Z. Manna. Deductive Model Checking. In R. Alur and T. Henzinger, editors, *Proceedings of CAV'96: Computer-aided Verification*, volume 1102 of *LNCS*, pages 208–219. Springer, 1996.
33. I. Walukiewicz. Pushdown processes: Games and model checking. In *Proceedings of CAV'96: Computer-aided Verification*, volume 1102 of *LNCS*, pages 62–74. Springer, 1996.