Incremental Model Checking for Decomposable Structures

J.A. MAKOWSKY* and E.V. RAVVE **

Department of Computer Science
Technion - Israel Institute of Technology
Haifa, Israel
e-mail: {selenajanos}@cs.technion.ac.il

Extended Abstract, January 1995
Submitted to MFCS’95

Abstract. Assume we are given a transition system which is composed from several well identified components. We propose a method which allows us to reduce the model checking of formulas in the complex system to model checking of derived formulas in Monadic Second Order Logic in the components. Our method applies to all practically used hardware specification languages, although at a certain cost. The basic idea goes back to a method proposed first by Feferman and Vaught in 1959, which we adapt and generalize to the specific context of model checking of transition systems. Our method allows for a precise definition of incremental model checking. We give also estimates on when our incremental method starts to be better than traditional methods.

1 Introduction

In hardware verification we find the following situation: We are given a mathematical model of a finite state device in form of a finite relational structure $A$ (transition system) and a formalized property $\phi$. Usually $\phi$ is given in advance and $A$ is being built with the aim to satisfy $\phi$. Checking whether $\phi$ holds in $A$ is to be atomized. This process is called model checking. The literature is rich in papers addressing this problem, [Eme90].

Very often $A$ is built from components $A_i$ where $i \in I$ is some index set or structure. In the process of building $A$ several candidate structures $A^j$ have to be checked for $\phi$, where $j$ denotes the $j$th attempt of designing $A$. Often $A^j$ differs from $A^{j+1}$ in one component $A_i^{j+1}$.

The problem we address here, is: How can we exploit this modularity to make the model checking process more efficient. To make this question meaningful, we have to make various aspects more precise.

* Partially supported by a Grant of the French-Israeli Binational Foundation and by the Fund for Promotion of Research of the Technion–Israel Institute of Technology
** This paper contains parts of the M.Sc.thesis of the second author, written under the supervision of the first author
Choice of the logic. We shall argue in section 2, that Monadic Second Order Logic (MSOL) is a good choice to deal with this problem for three reasons: It allows for a workable definition of modularity of structures to be checked, and hence also of the incrementality of model checking. A theorem due to Courcelle and Niwinski [Cou93, CN94] allows us to apply our method, at a certain cost, to all other specification languages used in the literature.

Precise definition of modularity. We propose a notion of modularity based on a generalization of disjoint unions of structures with additional links between the components, which depend only on the index structure and specified nodes in the components as described by some table. The table will be represented syntactically by a translation scheme $\Phi$, defined in section 3. Examples are discussed in detail in section 5 and the appendix. Such generalized disjoint unions have a long history in mathematical logic, [Mak85], but were, so far, not exploited for model checking purposes. We shall call them $\Phi$-sums. In section A we shall discuss realistic examples which fall under this definition, and exhibit others which provably do not.

Complexity measures. To measure the advantage of this method we shall differ from the usual approach, which looks at the cost of checking once, whether $A$ satisfies $\phi$, and uses either the size of $A$, the size of $\phi$ or the sum of the two as the relevant input size. We shall ask, what our method can gain by repeating this process many times, with small changes at a time. For this purpose we also look at the size of the changed component and the number of iterations. A detailed discussion of these cost evaluations is given in section 6.

Our main result shows how the truth of $\phi$ in $A$ depends on the components $A_i$ of $A$ and the index structure $I$. The exact formulation of this is rather involved and is explained in detail in section 4. It is an extension of the Feferman-Vaught Theorem, [FV59] for First Order Logic to MSOL, which seems to be new. Special cases have been considered in the literature ad hoc in [Gur79, She92]. For First Order Logic the Feferman-Vaught theorem covers a very wide class of generalized products and sums of structures (here representing transition systems or processes) and is extremely powerful. Our extension of this theorem for Monadic Second Order Logic works only for a more restricted class of sum-like operations. As a negative result we show in section 5 that our theorem does not apply to certain synchronized product construction used in building parallel synchronized transition systems.

From the theorem we derive a method for checking $\phi$ in $A$ which proceeds as follows:

Preprocessing. Given $\phi$ and $\Phi$, but no $A$, we construct a sequence of formulas $\psi_{i,j}$ and a boolean function $F_{\Phi,\phi}$. This construction is polynomial in the size of $\phi$ and $\Phi$.

Initialization. Assume our first structure $A^0$ has to be checked. In a first run we compute the boolean values $b_{i,j}$ defined by

$$b_{i,j} = 1 \text{ iff } A_i \models \psi_{i,j}.$$
Checking $\phi$. The theorem now states that $A^0 \models \phi$ iff $F_{\phi,\emptyset}(\vec{b}) = 1$.

**Iteration.** If a new $A^1$ has to be checked differing from $A^0$ only in, say, $A_1^1$, we only have to recompute the values $a_{1,j}$ and $F_{\phi,\emptyset}$.

A close analysis in section 6 of this process reveals that, even if the model checking procedure is already polynomial, considerable gains in efficiency are possible under our scenario of incremental, i.e. stepwise building and verification of the hardware.

**Acknowledgements**

We are would like to thank S. Shelah, who gave us his manuscript [She92]. The first author had stimulating discussions with Y. Hirshfeld and E. Clarke in the early stages of this research. In later stages, B. Courcelle, A. Arnold, D. Niwiński and O. Grumberg helped us in finding more examples to which we could apply our method.

## 2 Why Monadic Second Order Logic

### 2.1 Basics

Our choice of hardware specification language is Monadic Second Order Logic (MSOL). Recall that Second Order Logic (SOL) is like first order logic, but allows also variables and quantification over relation variables of various but fixed arities. Monadic Second Order Logic is the sublogic of SOL where relation variables are restricted to be unary. We also consider the fragment of SOL which is restricted to formulas of the form $\exists X \phi$ where $X$ is a finite sequence of relation variables and $\phi$ is first order. This fragment is denoted by $\text{ESOL}$.

The meaning function of formulas is explained for arbitrary $\tau$-structures where $\tau$ is the vocabulary, i.e. a finite set of relation and constant symbols. We do not consider here the case of adding function symbols. We denote structures by $A, B$ and their underlying sets by $A, B$.

The relation variables will range also over infinite sets, when the structures considered are infinite. The restriction to finite sets and relations when the structures are infinite is called Weak (Monadic) Second Order Logic, and will be used only once below. On finite structures this distinction is irrelevant.

The quantifier rank of formulas is defined as usual taking the maximum for $\lor, \land$ and $\neg$, and augmenting by 1 in case of quantification.

Two structures $A$ and $B$ are $n$-equivalent with respect to some logic $\mathcal{L}$, which we denote by $A \equiv_n B(\mathcal{L})$, if all the formulas of logic $\mathcal{L}$ with quantifier depth $n$ have the same truth value in these structures. Similarly, We denote by $Th^0_{\mathcal{L}}(A)$ a set of all formulas with $n$ quantifier of some logic $\mathcal{L}$, which hold in $A$, and by $Th^c_{\mathcal{L}}(A)$ a set of all formulas of $\mathcal{L}$, which are true in $A$. If the context makes it clear we shall omit $\mathcal{L}$. 

---

3
2.2 Expressive Power of Monadic Second Order Logic.

Monadic Second Order Logic has considerable expressive power. Most of the logics used in hardware verification are sublogics of it. On finite structures we have:

(i) Fixed Point Logic \( FPL \) (with inflationary or, alternatively, monotone, fixed points). The two versions have equal expressive power, as shown in [GS86]. Fixed Point Logic, in general, is definable in \( SOL \). For model checking usually only Monadic \( FPL \) is used. Monadic \( FPL \) is \( MSOL \) expressible on the transition system and the translation into \( MSOL \) is straightforward.

(ii) Propositional Dynamic Logic, [Har84]. Actually, it is definable in Monadic Fixed Point Logic on the transition system. The translation is straightforward.

Very often in hardware specification one models the device by a finite relational structure \( A \) and associates with it an infinite structure \( bhv(A) \) the behaviour of \( A \). By abuse of notation, one speaks of formulas true in \( A \), but they are really explained on \( bhv(A) \). Among these we have:

(i) The fixed point definable operations on the power set algebra of trees, [A.A92]. They are \( MSOL \) expressible on the transition system.

(ii) Linear Temporal Logic \( (LTL) \). Actually, \( LTL \) expressible already in First Order Logic on the behavior of the transition system, as was shown by Kamp, [Bur84].

(iii) Computation Tree Logic \( (CTL) \), [OL85]. Actually, it is in Monadic Fixed Point Logic on behavior of the transition system.

(iv) The operations, which can be defined without alternation on \( \mu \) and \( \nu \), [A.A92]. Actually, they are definable in Weak Monadic Second Order Logic on behavior of the transition system.

However, using powerful tools based on ideas related to Rabin’s theorem on the decidability of \( MSOL \) on infinite trees, [Rab69], Courcelle [Cou93], and Courcelle and Walukiewicz [CW95] proved:

**Theorem 1 (Courcelle and Walukiewicz).**

*Every \( MSOL \) expressible property of behavior of the transition system is equivalent to some \( MSOL \) expressible property of the transition system.*

This theorem justifies our restriction to \( MSOL \) over finite structures (transition systems). However, the translation of formulas over the behaviour to formulas over the transition system comes at a considerable cost and is at best feasible for small formulas.

3 Translation Schemes

In this section we introduce the general framework for syntactically defined translation schemes. We introduce also the notion of abstract translation schemes according to Rabin.
**Definition 2.** Translation Schemes $\Phi$.
Let $\tau$ and $\sigma$ be two vocabularies and $\mathcal{L}$ be a logic, such as $FOL$ or $MSOL$. Let $\sigma = \{R_1, \ldots, R_m\}$ and let $\rho(R_i)$ be the arity of $R_i$. Let $\Phi = \langle \phi, \psi_1, \ldots, \psi_m \rangle$ be formulas of $\mathcal{L}(\tau)$. $\Phi$ is $k$-feasible for $\sigma$ over $\tau$ if $\phi$ has exactly $k$ distinct free first order variables and each $\psi_i$ has $k\rho(R_i)$ distinct free first order variables. Such a $\Phi = \langle \phi, \psi_1, \ldots, \psi_m \rangle$ is also called a $k$-$\tau$-$\sigma$-translation scheme or, shortly, a translation scheme, if the parameters are clear in the context.

With a translation scheme $\Phi$ we can naturally associate a (partial) function $\Phi^*$ from $\tau$-structures to $\sigma$-structures.

**Definition 3.** The induced map $\Phi^*$.
Let $A$ be a $\tau$-structure and $\Phi$ be $k$-feasible for $\sigma$ over $\tau$. The structure $A_\Phi$ is defined as follows:

(i) The universe of $A_\Phi$ is the set $A_\Phi = \{ \bar{a} \in A^k : A \models \phi(\bar{a}) \}$;
(ii) The interpretation of $R_i$ in $A_\Phi$ is the set

$$A_\Phi(R_i) = \{ \bar{a} \in A_\Phi^{\rho(R_i) \times k} : A \models \psi_i(\bar{a}) \}.$$

Note that $A_\Phi$ is a $\sigma$-structure of cardinality at most $|A|^k$.

(iii) The partial function $\Phi^*: \text{Str}(\tau) \to \text{Str}(\sigma)$ is defined by $\Phi^*(A) = A_\Phi$. Note that $\Phi^*(A)$ is defined iff $A \models \exists \bar{a}\phi$.

With a translation scheme $\Phi$ we can also naturally associate a function $\Phi^\#$ from $\mathcal{L}(\sigma)$-formulas to $\mathcal{L}(\tau)$-formulas.

**Definition 4.** The induced map $\Phi^\#$.
Let $\theta$ be a $\sigma$-formula and $\Phi$ be $k$-feasible for $\sigma$ over $\tau$. The formula $\theta_\Phi$ is defined inductively as follows:

(i) For $R_i \in \sigma$ and $\theta = R(x_1, \ldots, x_m)$ let $x_{i,h}$ be new variables with $i \leq m$ and $h \leq k$ and denote by $\bar{x}_i = \langle x_{i,1}, \ldots, x_{i,k} \rangle$. We put $\theta_\Phi = \psi_i(\bar{x}_1, \ldots, \bar{x}_m)$. This works also for relation variables.

(ii) For the boolean connectives the translation distributes, i.e., if $\theta = (\theta_1 \lor \theta_2)$ then $\theta_\Phi = (\theta_{1,\Phi} \lor \theta_{2,\Phi})$ and if $\theta = \neg \theta_1$ then $\theta_\Phi = \neg \theta_{1,\Phi}$, and similarly for $\land$.

(iii) For the existential quantifier, we use relativization, i.e., if $\theta = \exists y \theta_1$, let $\bar{y} = \langle y_1, \ldots, y_k \rangle$ be new variables. We put $\theta_\Phi = \exists \bar{y}(\phi(\bar{y}) \land (\theta_{1,\Phi}))$.

(iv) For second order variables $U$ of arity $\ell$ and $\bar{a}$ a vector of length $\ell$ of first order variables or constants we translate $U(\bar{a})$ by treating $U$ like a relation symbol above and put

$$\theta_\Phi = \exists V(\forall \bar{v}(V(\bar{v}) \rightarrow (\phi(\bar{v}_1) \land \ldots \phi(\bar{v}_\ell) \land (\theta_{1,\Phi}))).$$

(v) The function $\Phi^\#: \mathcal{L}(\sigma) \to \mathcal{L}(\tau)$ is defined by $\Phi^\#(\theta) = \theta_\Phi$.

**Observation 5.** (i) $\Phi^\#(\theta) \in FOL$ (SOL) provided $\theta \in FOL$ (SOL), even for vectorized $\Phi$.

(ii) $\Phi^\#(\theta) \in MSOL$ provided $\theta \in MFOL$, but only for non-vectorized $\Phi$. 

5
In the sequel, we assume that $\Phi^*$ is not vectorized, unless stated otherwise.

The following facts hold (for proof see [EFT94]).

**Proposition 6.**
Let $\Phi = \{\phi, \psi_1, \ldots, \psi_m\}$ be a $k$-$\tau$-$\sigma$-translation scheme, $A$ a $\tau$-structure and $\theta$ a $L(\sigma)$-formula. Then $A \models \Phi^*(\theta)$ iff $\Phi^*(A) \models \theta$.

**Proposition 7.**
Let $\Phi = \{\phi, \psi_1, \ldots, \psi_m\}$ be a $k$-$\tau$-$\sigma$-translation scheme, $A$ a $\tau$-structure. In this case if $Th(A)$ is decidable then $Th(\Phi^*(A))$ is decidable too.

## 4 Piecing Structures Together

In this section we discuss various ways of obtaining transition systems from smaller components.

### 4.1 Disjoint Union

The *Disjoint Union* of a family of structures is the simplest example of juxtaposing structures where none of the components are linked to each other.

**Definition 8 (Disjoint Union).**
Let $\tau_i = \langle R_{i1}, \ldots, R_{i\ell}, c_{i1}, \ldots, c_{i\ell}\rangle$ be a vocabulary of $A_i$. In the general case the resulting structure is $A = \bigsqcup_{l \in I} A_l = \langle I \cup \bigsqcup_{l \in I} A_l, P(i, x), I(x), R_{i1}^1, \ldots, R_{i\ell}^1, R_{i1}^2, \ldots, R_{i\ell}^2, \ldots, R_{i1}^\ell, \ldots, R_{i\ell}^\ell, c_{i1}, \ldots, c_{i\ell}, c_{i1}^1, \ldots, c_{i\ell}^1, \ldots, c_{i1}^\ell, \ldots, c_{i\ell}^\ell, \ldots \rangle$
for all $i \in I$, where $P(i, x)$ is true iff $x$ came from $A_i$ and $I(x)$ is true iff $x$ came from $I$.

**Definition 9 (Partitioned Index Structure).** Let $I$ be an index structure. $I$ is called *finitely partitioned* into $\ell$ parts if there are unary predicates $I_{a\beta}, a < \ell$, in the vocabulary of $I$ such that their interpretation forms a partition of the universe of $I$.

Using Ehrenfeucht-Fraïssé games for $MSOL$, [Ehr61], it is easy to see that

**Theorem 10.** Let $I, J$ be two partitioned index structures over the same vocabulary such that for $i, j \in I_\ell$ and $i', j' \in J_\ell$ $A_i$ and $A_j$ ($B_i$ and $B_j$) are isomorphic.

(i) If $I \equiv^n_{MSOL} J$, and $A_i \equiv^n_{MSOL} B_i$ then $\cup_{i \in I} A_i \equiv^n_{MSOL} \cup_{j \in J} B_j$.

(ii) If $I \equiv^n_{MSOL} J$, and $A_i \equiv^n_{FOL} B_i$ then $\cup_{i \in I} A_i \equiv^n_{FOL} \cup_{j \in J} B_j$.

If, as in our applications, there are only finitely many different components, we can prove a stronger statement, dealing with formulas rather than theories.
**Theorem 11.** Let $I$ be a finitely partitioned index structure. Let $A = \bigcup_{i \in I} A_i$ be a $\tau$-structure, where each $A_i$ is isomorphic to some $B_1, \ldots, B_\ell$ over the vocabularies $\tau_1, \ldots, \tau_\ell$, in accordance to the partition. For every $\phi \in MSOL(\tau)$ there is a boolean function $F_\phi(b_{1,1}, \ldots, b_{1,i}, \ldots, b_{\ell,1}, \ldots, b_{\ell,j}, b_{1,1}, \ldots, b_{1,j})$ and formulas $\psi_{1,1}, \ldots, \psi_{1,i}, \ldots, \psi_{\ell,1}, \ldots, \psi_{\ell,j}, \psi_{I,1}, \ldots, \psi_{I,j}$ such that for every $A, I$ and $B_\ell$ as above with $B_\ell \models \psi_{I,j}$ iff $b_{I,j} = 1$ and $B_1 \models \psi_{I,j}$ iff $b_{I,j} = 1$ we have $A \models \phi$ iff $F_\phi(b_{1,1}, \ldots, b_{1,i}, \ldots, b_{\ell,1}, \ldots, b_{\ell,j}, b_{1,1}, \ldots, b_{1,j}) = 1$. Moreover, $F_\phi$ and the $\psi_{I,j}$ are computable from $\phi$ alone, but are exponential in the quantifier depth of $\phi$.

Proof: By analyzing the proof of theorem 10 and tedious book keeping. \hfill \Box

### 4.2 Sum-like Structures

The disjoint union as such is not very interesting. However, combining it with translation schemes gives as a rich repertoire of patching techniques.

**Definition 12** Sum-like Structures. Let $I$ be a finitely partitioned index structure. Let $A = \bigcup_{i \in I} A_i$ be a $\tau$-structure, where each $A_i$ is isomorphic to some $B_1, \ldots, B_\ell$ over the vocabularies $\tau_1, \ldots, \tau_\ell$, in accordance with the partition.

Furthermore let $\Phi$ be a non-vectorized $\tau$-$\sigma$ MSOL-translation scheme. The $\Phi$-sum of $B_1, \ldots, B_\ell$ over $I$ is the structure $\Phi^*(A)$, or rather any structure isomorphic to it.

**Theorem 13.** Let $I$ be a finitely partitioned index structure and let $A$ be the $\Phi$-sum of $B_1, \ldots, B_\ell$ over $I$, as above. For every $\phi \in MSOL(\tau)$ there is a boolean function $F_{\Phi,\phi}(b_{1,1}, \ldots, b_{1,i}, \ldots, b_{\ell,i}, b_{1,1}, \ldots, b_{1,j}, b_{1,1}, \ldots, b_{1,j})$ and formulas $\psi_{1,1}, \ldots, \psi_{1,i}, \ldots, \psi_{\ell,1}, \ldots, \psi_{\ell,j}, \psi_{I,1}, \ldots, \psi_{I,j}$ such that for every $A, I$ and $B_\ell$ as above with $B_\ell \models \psi_{I,j}$ iff $b_{I,j} = 1$ and $B_1 \models \psi_{I,j}$ iff $b_{I,j} = 1$ we have $A \models \phi$ iff $F_{\Phi,\phi}(b_{1,1}, \ldots, b_{1,i}, \ldots, b_{\ell,1}, \ldots, b_{\ell,j}, b_{1,1}, \ldots, b_{1,j}) = 1$. Moreover, $F_{\Phi,\phi}$ and the $\psi_{I,j}$ are computable from $\Phi^*$ and $\phi$, but are exponential in the quantifier depth of $\phi$.

Proof: By analyzing the proof of theorem 11 and using theorem 6. \hfill \Box

The corresponding theorem for vectorizing translation schemes does hold for SOL, but not for MSOL.

### 5 Real Life Examples

Now we will consider several examples taken from every day life. The first few are asynchronous and synchronous parallel composition of hardware respectively, including the pipeline design. The last two are taken from parallel numeric computations.
5.1 Pipelines

Let us consider the following composition of two input graphs $H$ and $G$. $G$ can be viewed as a display graph, where on each node we want to have a copy of $H$, such that certain additional edges are added. In practice this is a model on how a pipeline works. The nodes marked with $L^j$ are the latches.

Let $G = (D_G, R)$ and $H = (D_H, S, L^j(j \in J))$ be two relational structures ($J$ is finite), then their composition $C = (D_C, L^1_C, \ldots, L^n_C, S_C, R^j_C(j \in J))$ is defined as following:

- $D_C = \bigcup_{g \in G} D_H$;
- $L^j_C(w)$ is true if $w$ belongs to $L^j$;
- $S_C = \{(w, v) : w \in D_H, v \in D_H, S(w, v)\}$;
- $R^j_C = \{(w, v) : L^j(w), L^j(v), P(i, w), P(i', v), R(i, i')\}$.

It is easy to see that this construction can be obtained from the cartesian product $G \times H$ by a FOL translation scheme without vectorization. However, the cartesian product cannot be obtained from $\bigcup_{g \in G} H$ without vectorization. However, $C$ can also be obtained from the disjoint union $\bigcup_{g \in G} H$ by a FOL translation scheme without vectorization. The following proposition makes precise.

**Proposition 14.**

$C$ is isomorphic to $\Phi^*(\bigcup_{g \in G} H)$ with $\Phi = \langle \phi, \psi_{L^1_C}, \ldots, \psi_{L^n_C}, \psi_S, \psi_{R^1_C}, \ldots, \psi_{R^n_C} \rangle$ and $\phi = \exists i (P(i, x) \land I(i)) \land \psi_{L^j_C} = \exists i ((P(i, x) \land I(i)) \land L^j(x)) \land \psi_S = \exists i ((P(i, x) \land I(i)) \land S(w, v)) \land \psi_{R^j_C} = \exists i \exists i' (((P(i, x) \land I(i'))) \land R(i, i')) \land (L^j(w) \land L^j(v)) \land (P(i, w) \land P(i', v)))$.

In this example, depending on the choice of the interpretation of the $L^j$'s, more sophisticated parallel computations can be modelled, but not all.

5.2 Synchronous Parallel Composition

Our next example is a synchronized parallel transition system, which is not sum-like. We consider the parallel composition of transition systems as studied in [GL91]. The transition systems here are given as follows.

**Definition 15.** Given a structure $A = \langle S, S_0, R, X_\varphi(\varphi \in \mathcal{R}) \rangle$, where.

(i) $S$ is a finite set of states.
(ii) $S_0$ is a relation for set of initial states.
(iii) $R \subseteq S \times S$ is a transition relation.
(iv) $X_\varphi = \{ s \in S : s \vDash \varphi \}$ where $\mathcal{R}$ is a finite set of atomic propositions.

Now the synchronous product is defined as follows:

**Definition 16.** Let $A$ and $A'$ be two structures as above. The composition of $A$ and $A'$, denoted $A \parallel A'$, is the structure $A''$ defined by
Atomic Propositions $\mathcal{AP} = \mathcal{R} \cup \mathcal{V}$.

States $S^{\mu} = \{(s, s') \in S \times S' \mid \exists(s) \cap \mathcal{R} = \exists(s') \cap \mathcal{R}\}$.

Initial States $S_0^{\mu} = (S_0 \times S_0') \cap S^{\mu}$.

Transition Relation $(s, s') \in X^{\mu}_0$ iff $s \in X^0_0$ or $s' \in X^0_0$. 

Unary Predicates $R^{\mu}((s, s'), (t, t'))$ iff $R(s, t)$ and $R'(s', t')$.

**Theorem 17.** There are no unary $\Phi$, 1 and $A_v$, such that $A^{\mu} = A \parallel A'$ is isomorphic to $\Phi[\bigcup_{v \in 1} A_v]$. In other words, the synchronized parallel composition of two transition systems is not, in general, sum-like.

Proof:
The simplest case of this composition is the Cartesian Product of components. Now, by Büchi’s theorem, the MSOL-theory of one successor is decidable, whereas, the MSOL-theory of the cartesian product of two copies of one successor relation is not. For both facts, see, e.g., [Tho90].

6 Complexity Consideration.

In this section we discuss under what conditions theorem 11 improves the complexity of model checking, when measured in the size of the composed structure only. Our scenarios are as follows: A formula (set of formulas) $\phi$ is given in advance. A structure (transition system) is now submitted to the model checker and we want to know, how long it takes to check whether $\phi$ is true. The first scenario (Scenario A) consists of checking one structure. More realistically, in Scenario B, we check several structures, which differ from each other only in small components.

6.1 Complexity of Model Checking for MSOL

Theorems 10 and 11 hold for MSOL and, with restrictions, also for FOL. Model checking for FOL is polynomial (even in logarithmic space), whereas model checking for MSOL is likely to be non-polynomial, as it sits fully in the polynomial hierarchy PH. More precisely: The complexity of model checking (in the size of the structure) of Second Order Logic can be described as follows:

**Fagin:** The class NP of nondeterministic polynomial-time problems is the set of properties, which are expressible by existential second-order logic (ESOL) on finite structures, [Fag74].

**Stockmeyer–Lynch:** Model checking for SOL is in the polynomial hierarchy PH, cf. [GJ79]. Moreover, for every level of the polynomial hierarchy there is a problem, expressible in SOL, that belongs to this class. The same fact hold for MSOL, too, as observed in [MP93].

**Vardi:** Model checking for Fixed Point Logic is polynomial, [Var82]. All the problems, which are expressible by CTL* can be checked in polynomial time [Eme90].
Most Hardware Verification Languages are stronger than FOL but weaker than MSOL, and their model checking complexity is polynomial. However, theorem 11 does not hold for them. We now discuss these scenarios for various logics in detail.

6.2 General analysis

From theorem 11 we conclude that:

Observation 18. (i) Function $F_{\phi_0}$ gives a Model Checker of $A^*$ with the help of Model Checkers of $A_i$ and index structure $I$.
(ii) Theorem 11 is useless if the $A_i$ and index structure $I$ are not "easy" obtained form $A$.

Assume that $A$ is sum-like. Its components are $A_i$ with index structure $I$, and we want to check whether $\phi$ is true in $A$. Assume that:

- $T(N)$ denotes time to solve the problem by the usual way ($N$ denotes the size of structure $A$);
- $\mathcal{E}_I$ denotes time to extract index structure $I$ from $A$;
- $\mathcal{E}_i$ denotes time to extract each $A_i$ from $A$;
- $C_I(n_I)$ denotes time to compute all values of $b_{I,j}$, where $n_I$ is the size of $I$;
- $C_i(n_i)$ denotes time to compute all values of $b_{i,j}$, where $n_i$ is the size of $A_i$;
- $T_{F_{\phi_0}}$ denotes time to build $F_{\phi_0}$;
- $T_{\text{search}} = T_S$ denotes time to search one result in the table of $F_{\phi_0}$.

According to these symbols new checking time is:

$$T_{new} = \mathcal{E}_I + \sum_{i \in I} \mathcal{E}_i + \sum_{i \in I} C_i + + T_{F_{\phi_0}} + T_S$$

and the question to answer is: where $T > T_{new}$.

6.3 Scenario A: Single Model Checking

Assume that some structure $A$ is sum-like, where $A_i = \bar{A}_i$. Example of such a structure $A$ is shown in A.1. Assume that:

- $N$ is a size of $A$, $n$ is a size of $\bar{A}$ and $l$ is a size of index structure $I$.
- $\mathcal{E}_I = \mathcal{E}_i = 0$.
- $T = f(x) = e^g(x)$.

In this case $T_{new} = P^p(T(n), T(l))$, where $P^p$ denotes polynomial of degree $p$, and $T_{\text{search}} = T(l \cdot n)$. The question to answer is: when $f(n \cdot l) > P^p(f(n), f(l))$.

According to our assumptions we obtain: $e^{g(n \cdot l)} > a_p(e^{g(n)} + e^{g(l)})$.

Assume that $n = l$. Then $g(n^2) > p \cdot g(n) + ln2 + ln(a_p)$. Assume that $g(x) = ln^2(x)$, then $f(x) = x^{ln(x)}$. In this case we obtain that:

$ln^2(n^2) > p \cdot ln^2(n) + ln2 + ln(a_p)$ or $ln^2(n) > \frac{ln^2(a_p)}{2^p}.

On the other hand, if we use some FOL-definable logic, then Model Checking procedure is polynomial in $A$ and in $I$ and each $A_i$ too. In this case we do not obtain time gain.
6.4 Scenario B: Incremental Model Checking

Assume that our aim is as follows: we need to change several times (let us denote the number of the times by $l$) some fixed component of $A$ called $A_i$. Assume that each time we need to check whether $A$ satisfies some fixed formula $\phi$. We use the notations as above.

Let $T_{old}$ be time to solve the present problem by the usually applied way. It should be clear that $T_{old} = l \cdot T(N)$. Let $T_{new}$ be time to solve the same problem, when structure $A$ is viewed as $F$-sum generalized sum. It should be clear that $T_{new}(N,n) = T(N-n) + l \cdot T(n) + T_{F_{\phi,\alpha}} + l \cdot T_S$. The question to answer is: which value of $n$ provides that $T_{old} > T_{new}$. Assume that we deal with $FOL$-definable logic and $T(x) = x^2$, then:

$$1 \cdot N^2 > (N-n)^2 + l \cdot n^2 + T_{F_{\phi,\alpha}} + l \cdot T_S$$

$$N^2 - 2 \cdot n \cdot N + n^2(l + 1) + T_{F_{\phi,\alpha}} + l \cdot T_S - l \cdot N^2 < 0$$

$$n_{1,2} = \frac{N \pm \sqrt{N^2+(l+1)(N^2(l-1)-T_{F_{\phi,\alpha}}-l \cdot T_S)}}{(l+1)}.$$

If $n_1 \leq n \leq n_2$ then $T_{old} > T_{new}$.

$$n_2 = \frac{N + \sqrt{N^2+(l+1)(N^2(l-1)-T_{F_{\phi,\alpha}}-l \cdot T_S)}}{(l+1)}$$

$$\lim_{n \to \infty} n_2 = \sqrt{N^2 - T_S}.$$

The same consideration can be done for other polynomial dependencies $T(x)$ for $FOL$-definable logics.

Let $L$ be any proper sublogic of MSOL stronger that First Order Logic. Theorem 11 does not hold in the following: if we apply it then $\psi_{i,j}$ are not necessary in $L$. So, the next open problem arises: how can we exploit the complexity gain described before for such a $L$?

The complexity considerations are true for logics over the transition systems. The use of theorem 1 (see page 4) helps here, but the formula obtained may be very much bigger then the original formula. This affects in our scenarios only the constants, but it does dramatically. Note that, although $LTL$ is $FOL$ definable over the behavior, the formula obtained via theorem 1 is generally in MSOL.

References


A Domain Decomposition Method for Solving of Partial Differential Equations

A.1 Poisson Equation

Let us consider the decomposition method for solving the following partial differential equation which stems from the Poisson problem.

Given the following equation:

\[
\frac{\partial^2 u}{\partial t^2} - \frac{\partial^2 u}{\partial x^2} = 0
\]

and the following conditions: \( t \geq 0, 0 \leq x \leq 1, \)
\( u(0, t) = 0, u(1, t) = 0, u(x, 0) = f(x), u_t(x, 0) = 0. \)

The approximate solution can be obtained with the help of the following computation scheme:

\[
U_j^{n+1} = 2 \cdot (1 - \mu^2) \cdot U_j^n + \mu^2 \cdot (U_{j+1}^n + U_{j-1}^n) - U_j^{n-1}, \quad \text{where:}
\]

\( j = 1, ..., N - 1; n = 1, 2, 3, ... \) and \( U_j^n \approx u(j \cdot \Delta x, n \cdot \Delta t) \) and \( \Delta x = \frac{1}{N}, \mu = \frac{\Delta t}{\Delta x} \)

The boundary conditions are given by: \( U_0^n = 0, U_N^n = 0, U_j^0 = f(j \cdot \Delta x), U_j^{-1} = U_j^0. \)

The computation domain can be divided among \((N - 1)\) elementary computation elements \( A_j \), each computing \( U_j^{n+1} \) (see Fig.1). Each of the \( A_j (j \neq 1, j \neq N - 1) \) has to communicate with the other only when it requires inputs \( U_{j+1}^n \) or \( U_{j-1}^n \) from \( A_{j+1} \) and \( A_{j-1} \). We see that this example is similar to Pipelines. Hence,

**Proposition 19.** The computation network \( W_1 \) for the 1-dimensional wave equation is sum-like.
A.2 Heat Equation

Let us consider the decomposition method for solving the following partial differential equation. Given:

\[
\begin{align*}
\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} &= f(x, y) \quad \text{in} \quad \Omega = [0, b] \times [0, a] \\
u &= \phi(x, y) \quad \text{on} \quad \partial \Omega
\end{align*}
\]

The approximate solution can be obtained by the following way: \( x_i = i \cdot h, 0 \leq i \leq p \) and \( y_j = j \cdot k, 0 \leq j \leq q \); \( U_{i,j} \approx u(x_i, y_j) \)

\[
\frac{1}{h^2}(U_{i-1,j} + U_{i+1,j}) - 2 \cdot \left(\frac{1}{h^2} + \frac{1}{k^2}\right) \cdot U_{i,j} + \frac{1}{k^2}(U_{i,j-1} + U_{i,j+1}) = f_{i,j}
\]

Fig. 2. Computation Scheme for the Second Domain Decomposition Method.
Assume that $a = 1, b = 1, p = q = N, h = k, f(x, y) = 0$. In this case the approximate solution is: $4 \cdot U_{i,j} - U_{i+1,j} - U_{i-1,j} - U_{i,j+1} - U_{i,j-1} = 0$; $U_{i,j} \approx u(i\Delta x, j\Delta y); \Delta x = \Delta y = \frac{1}{N}; 1 \leq i, j \leq N - 1$

Let us consider one iterative method to compute $U_{i,j}$.

\[
U_{i,j}^{k+1} = \frac{U_{i-1,j}^k - U_{i+1,j}^k - U_{i,j+1}^k - U_{i,j-1}^k}{4},
\]
where $1 \leq i, j \leq N - 1$.

In this case the computation procedure can be presented as shown on Fig. 2.

**Proposition 20.** The computation network $H_2$ for the 2-dimensional heat equation is not sum-like.

Proof: It is easy to see that there is a translation scheme from $H_2$ to Grids. \(\square\)