Stable Models for Stubborn Sets

Kimmo Varpaaniemi
Helsinki University of Technology
Laboratory for Theoretical Computer Science
P.O. Box 9700, FIN-02015 HUT, Finland
(kimmo.varpaaniemi@hut.fi)

Abstract. The stubborn set method is one of the methods that try to relieve the state space explosion problem that occurs in state space generation. Spending some time in looking for "good" stubborn sets can pay off in the total time spent in generating a reduced state space. This paper shows how the method can exploit tools that solve certain problems of logic programs. The restriction of a definition of stubbornness to a given state can be translated into a variable-free logic program. When a stubborn set satisfying additional constraints is wanted, the additional constraints should be translated, too. It is easy to make the translation in such a way that each acceptable stubborn set of the state is represented by at least one stable model of the program, each stable model of the program represents at least one acceptable stubborn set of the state, and for each pair in the representation relation, the number of certain atoms in the stable model is equal to the number of enabled transitions of the represented stubborn set. So, in order to find a stubborn set which is good w.r.t. the number of enabled transitions, it suffices to find a stable model which is good w.r.t. the number of certain atoms.
Keywords: reachability analysis, reduced state space generation, stubborn sets, variable-free logic programs, stable models

1. Introduction

Reachability analysis, also known as *exhaustive simulation* or *state space generation*, is a powerful formal method for detecting errors in concurrent and distributed finite state systems. Strictly speaking, infinite state systems can be analyzed, too, but reachability analysis methods are typically such that they cannot process more than a finite set of states. Nevertheless, we can quite well try to find errors even in cases where we do not know whether or not the complete state space of the system is finite.

Anyway, reachability analysis suffers from the so called *state space explosion problem*, i.e. the complete state space of a system can be far too large w.r.t. the resources needed to inspect all states in the state space. Fortunately, in a variety of cases we do not have to inspect all reachable states of the system in order to get to know whether or not errors of a specified kind exist.

The *stubborn set method* [12, 13, 14, 15, 10], and the *sleep set method* [2] are state search techniques that are based on the idea that when two executions of action sequences are sufficiently similar to each other, it is not necessary to investigate both of the executions. *Persistent sets* [2] and *ample sets* [1, 7] are strikingly similar to stubborn sets, at least if we consider the actual construction algorithms that have been suggested for stubborn, persistent and ample sets. This similarity is made explicit in [4] where a set is said to be a *stamper set* whenever the set is stubborn or ample or persistent in some way. This paper is concentrated on the construction of stubborn sets.

Place/transition nets [8] are the formalism to which the stubborn set method is applied in this presentation. Due to similarities between logical structures of definitions of stubbornness, the approach can easily be extended to cover all conventional formalisms where the stubborn set method is applicable.

A reduced state space for a net can be constructed by starting from the initial state of the net and by taking into account some but not necessarily all possible transitions at those states that become encountered. Assuming that we want a mechanically computed answer to the question if terminal states exist in the complete state space, we can let the stubborn set method construct a reduced state space graph. The obtained space actually contains all the terminal states of the complete state space [12, 15].

Given any useful definition of stubbornness, it is likely if not provably at least NP-hard [7] to find a cardinality minimal reduced state space among the the alternatives induced by the definition. No algorithm for finding such a minimum "within a reasonable time" has been presented either. Minimization of branching at each encountered state does not guarantee anything about the cardinality of the reduced state space [14, 15], but ways of doing it are known and some of the ways are of practical interest while there are not very many arguments against minimal branching itself.

In the stubborn set method, minimization of branching at a state means finding a stubborn set which is cardinality minimal w.r.t. enabled transitions. Even this cardinality minimization may be at least NP-hard, but incomplete cardinality minimization can still be quite practical. Though the "incomplete minimization algorithm" in [15] can be taken seriously as such, it has much room for optimizations. Much has been done in solving similar problems in the domain of logic programs, so we now suggest an approach on the basis of such solutions.

The restriction of a definition of stubbornness to a given state can be translated into a variable-free logic program. When a stubborn set satisfying additional constraints is wanted, the additional constraints should be translated, too. It is easy to make the translation in such a way that each acceptable stubborn set of the state is represented by at least one *stable model* [5, 6, 11] of the program, each stable model of the program represents at least one acceptable stubborn set of the state, and for each pair in the representation relation, the number of certain atomic formulas in the stable model is equal to the number of enabled transitions of the represented stubborn set. So, in order to find a stubborn set which is good w.r.t. the number of enabled transitions, it suffices to find a stable model which is good w.r.t. the number of certain atomic formulas.

The rest of this paper has been organized as follows. *And/or-graphs*, an intermediate formalism in the above mentioned translation, are described in Section 2, and a principally known complexity result is presented for them. Section 3 defines the concepts for logic programs and shows how an and/or-graph optimization problem can be transormed into a stable model optimization problem. Section 4 defines place/transition nets and stubbornness and fills in the gap in the translation. (Stubbornness is actually defined in terms of and/or-graphs.) Conclusions are then drawn in Section 5.

We shall use N to denote the set of non-negative integer numbers, 2^X to denote the set of subsets of the set X, X^* (respectively, X^ω) to denote the set of finite (respectively, infinite) words over the alphabet X, and ε to denote the empty word. For any nonempty finite word x, x^ω is the infinite word $xxx\ldots$. (As usual, exponentation has a higher precedence than concatenation in the presentation of words.) For any alphabet X and for any $\rho \in X^\omega$, ρ is thought of as a function from N to X in such a way that $\rho = \rho(0)\rho(1)\rho(2)\ldots$. When $R \subseteq A \times B$ and $x \in A$, $R(x)$ is the set $\{y \in B \mid \langle x, y\rangle \in R\}$.

2. And/or-graphs

And/or-graphs are used for various purposes e.g. in the research of artificial intelligence. We consider them up to an extent that is needed from the point of view of the stubborn set method. The notion of legality defined below is essentially the same as in [13]. The notion of H-solidity is defined in order to support compact describing of things.

Definition 2.1 An *and/or-graph* is a quadruple $\langle V_\otimes, V_\oplus, \kappa, F\rangle$ such that $V_\otimes$ is the set of *and-vertices*, $V_\oplus$ is the set of *or-vertices*, $V_\otimes \cap V_\oplus = \emptyset$, $V_\otimes \cup V_\oplus$ is finite, $\kappa \in V_\otimes \cup V_\oplus$, $F \subseteq (V_\otimes \cup V_\oplus) \times (V_\otimes \cup V_\oplus)$ is the set of *edges*, and $\forall y \in V_\oplus\ F(y) \neq \emptyset$. A set $L \subseteq V_\otimes \cup V_\oplus$ is *legal* iff (i) $\kappa \in L$, (ii) $\forall x \in V_\otimes \cap L\ F(x) \subseteq L$, and (iii) $\forall y \in V_\oplus \cap L\ F(y) \cap L \neq \emptyset$.

For any $H \subseteq V_{\otimes} \cup V_{\oplus}$, a set $B \subseteq V_{\otimes} \cup V_{\oplus}$ is H-*solid* iff there exists a legal set $L \subseteq V_{\otimes} \cup V_{\oplus}$ such that $B = H \cap L$. □

For any H and B, it can be checked in time $O(|V_{\otimes}| + |V_{\oplus}| + |F|)$ whether or not B is H-solid. The algorithm in Figure 1 makes such a check.

The pseudo-code is a mixture of mathematical expressions, English expressions and the C programming language [3]. Words belonging to the control structure of C are written in boldface. A construct of the form "**for (** each x in A such that $\psi(x)$ **)**", where A is a set and $\psi(x)$ a truth-valued expression on x, is apparently against the syntax of the C language but corresponds to a valid "for-construct" where a cursor moves through a data structure and skips "uninteresting" elements. The type **AO_vertex**, implementing each vertex of the and/or-graph, is assumed to have been defined appropriately. Each vertex has links to the immediate predecessor vertices. (Links to successor vertices are not needed.) Also, each vertex is coloured with a single colour, and each or-vertex has a counter. The and/or-graph data structure is assumed to have a global scope.

The function SolidCheck returns 1 if B is H-solid. Otherwise 0 is returned. We first check that $B \subseteq H$ and $\kappa \notin H \setminus B$. Then we initialize the colours and counters. When we come to the line 31, each or-vertex has the number of outgoing edges in the counter, the value being non-zero due to Definition 2.1. Whenever we come to the line 33, the set of non-grey vertices is legal. All this can be proven quite mechanically, as well as the overall correctness of the algorithm and the above claim concerning the time taken by the algorithm.

It is easy to modify the algorithm of Figure 1 in such a way that we get an algorithm for checking of whether or not B has a H-solid subset (possibly but not necessarily B itself). It suffices to make the line 24 blank and erase the part "and all the vertices of B" of the line 31. Such an operation clearly does not change the time complexity, whereas the correctness of the obtained algorithm can be proven in the same way as the correctness of the original algorithm.

By modifying the algorithm of Figure 1 in another way, an algorithm for computing an inclusion minimal H-solid set can be obtained. Such an algorithm is easy to guess by looking at the work done in [13, 14, 15]. The time taken by the obtained algorithm is $O((|V_{\otimes}|+|V_{\oplus}|+|F|)\cdot|H|)$. However, instead of being satisfied by mere inclusion minimality, we like to deal with the following problem: "Given an and/or-graph and a subset H of its vertices, compute a cardinality minimal H-solid set." This problem can be solved in time $O((|V_{\otimes}|+|V_{\oplus}|+|F|)\cdot 2^{|H|})$ since it suffices to execute the algorithm in Figure 1 for each subset of H in turn.

```
void Initialize (AO_vertex x ) {
    mark x white;
    for ( each immediate predecessor vertex y of x ) {
        if ( y is a black and-vertex ) Initialize(y);
        else if ( y is an or-vertex ) {
            add 1 to the counter of y;
            if ( y is black ) Initialize(y); } } }
int Removal (AO_vertex x ) {
    if ( x is black ) return 0;
    mark x grey;
    for ( each non-grey immediate predecessor vertex
        y of x ) {
        if ( y is an and-vertex ) {
            if ( Removal(y) is equal to 0 ) return 0;
        } else /* y is an or-vertex. */ {
            subtract 1 from the counter of y;
            if ( the counter of y is equal to 0 ) {
                if ( Removal(y) is equal to 0 ) return 0;
            } } } return 1; }
int SolidCheck (AO_vertex_set H, AO_vertex_set B ) {
    mark all vertices black;
    mark all vertices of H white;
    if ( B contains at least one black vertex ) return 0;
    if ( κ is white but not in B ) return 0;
    mark all vertices black;
    set the counters of all or-vertices to 0;
    while ( at least one black vertex exists ) {
        let v be some black vertex; Initialize(v);
    }
    mark κ and all the vertices of B black;
    while ( H contains at least one white vertex ) {
        let v be some white vertex of H;
        if ( Removal(v) is equal to 0 ) return 0;
    } return 1; }
```

Figure 1. Checking of whether or not B is H-solid.

Though some more intelligent solutions certainly exist, there is not much hope for dramatic reduction in the worst-case time complexity.

The above problem has the subproblem: "Given an and/or-graph, a subset H of its vertices and a non-negative integer number n, compute a H-solid set of cardinality less than or equal to n, or show that such a H-solid set does not exist." This problem is NP-complete since (i) we can check the correctness of any positive answer by using the algorithm in Figure 1, and (ii) the famous NP-complete problem SAT can be reduced to this problem in polynomial time, by constructing the same and/or-graph as is constructed in [9], κ being the unique root vertex and H the set of terminal vertices of the acyclic graph in question.

Knowing all this, we are relieved by the fact that the context where we look for small H-solid sets does not force us to find cardinality minimal H-solid sets.

3. Logic programs and stable models

Definition 3.1 A *logic program* is a pair $\langle Q, R\rangle$ where Q is the set of *atomic formulas* and R is the set of *rules* such that Q is finite and $R \subseteq Q \times (2^Q) \times (2^Q)$. The *interpretation* of the program is the function ι from $R \times 2^Q$ to 2^Q such that for any $\langle a, B, C\rangle \in R$ and for any $Q_s \subseteq Q$, $\iota(\langle a, B, C\rangle, Q_s)$ is (i) $\{a\} \cup Q_s$ if $(B \subseteq Q_s) \wedge (C \cap Q_s = \emptyset)$, and (ii) Q_s in the other case. The *inference graph* of the program is such that the set of vertices in the graph is 2^Q and the set of edges is $\{\langle Q_1, Q_2\rangle \in (2^Q) \times (2^Q) \mid Q_2 \neq Q_1 \wedge \exists r \in R\, Q_2 = \iota(r, Q_1)\}$. Let $Q_s \subseteq Q$. The *reduct of the program w.r.t.* Q_s is the logic program $\langle Q, R'\rangle$ such that $R' = \{\langle a, B, \emptyset\rangle \mid \exists C \in 2^Q\ (\langle a, B, C\rangle \in R) \wedge (C \cap Q_s = \emptyset)\}$. □

When $B = \{b_1, \ldots, b_m\}$ and $C = \{c_1, \ldots, c_n\}$, a conventional way to write the rule $\langle a, B, C\rangle$ is "$a \leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n$". The inference graph is easily seen to be acyclic. When all rules are of the form $\langle a, B, \emptyset\rangle$, then for each vertex Q_s, only one terminal vertex is accessible from Q_s via the paths of the graph, and we then say that the terminal vertex accessible from the empty set is the *deductive closure of the program.*

We define the notion of a *stable model* as follows: a set $Q_s \subseteq Q$ is a stable model of a logic program $\langle Q, R\rangle$ iff the deductive closure of the reduct of the program w.r.t. Q_s is Q_s itself.

Let us then build a connection between and/or-graphs and logic programs. Let $\langle V_\oplus, V_\otimes, \kappa, F\rangle$ be an and/or-graph and $H \subseteq V_\otimes \cup V_\oplus$. Let $\langle Q, R\rangle$ be a logic program such that (i) $Q = H \cup J \cup \{\alpha, \beta\}$, (ii) $\alpha \neq \beta$, (iii) $(V_\otimes \cup V_\oplus \cup J) \cap \{\alpha, \beta\} = \emptyset$, (iv) $(V_\otimes \cup V_\oplus) \cap J = \emptyset$, (v) there is a bijection γ from $V_\otimes \cup V_\oplus$ to J, and

(vi) $R = \{\langle \alpha, \{\beta\}, \{\alpha\}\rangle, \langle \beta, \{\gamma(\kappa)\}, \emptyset\rangle\} \cup$
$\{\langle \gamma(x), \{\gamma(z)\}, \emptyset\rangle \mid x \in V_\otimes \wedge z \in E(x)\} \cup$
$\{\langle \gamma(y), \{\gamma(u) \mid u \in E(y)\}, \emptyset\rangle \mid y \in V_\oplus\} \cup$
$\{\langle \gamma(h), \emptyset, \{h\}\rangle \mid h \in H\} \cup \{\langle h, \emptyset, \{\gamma(h)\}\rangle \mid h \in H\}$.

By strictly following the definitions, it is relatively easy to show that a set $B \subseteq V_\otimes \cup V_\oplus$ is H-solid if and only if the above program has a stable model Q_s such that $B = H \cap Q_s$. The bijection γ simulates negation. The atomic formulas α and β have the effect that no stable model contains $\gamma(\kappa)$. This is perhaps the most difficult part to prove, so we prove it now. Every reduct of the program has the rule $\langle \beta, \{\gamma(\kappa)\}, \emptyset\rangle$. So, if $\gamma(\kappa)$ is in a stable model, β is in the model as well. If both α and β are in a stable model, the reduct w.r.t. the model does not contain any rule that refers to α, and consequently, the deductive closure of the reduct does not contain α, which is a contradiction with the stability of the model. If β is in a stable model without α, the reduct w.r.t. the model contains the rule $\langle \alpha, \{\beta\}, \emptyset\rangle$, and consequently, the deductive closure of the reduct contains α or does not contain β, both alternatives being in contradiction with the stability of the model. We thus conclude that $\gamma(\kappa)$ cannot be in any stable model. (The rules referring to α and β actually implement an *integrity constraint* in the way recommended by [6].)

Due to the above translation, small H-solid sets can be obtained by looking for stable models that are small w.r.t. the intersection with H. A big effort has been made to get good algorithms for the latter purpose. The Smodels tool [11, 17] implements some of such algorithms, even an algorithm which is able to compute a cardinality minimal stable model w.r.t. H. The bound mentioned in Section 2.1 does not become essentially reduced in this way, but the algorithm is well prepared against typical sources of unnecessary combinatorial explosion.

4. Place/transition nets and stubborn sets

Definition 4.1 A *place/transition net* is a quadruple $\langle S, T, W, M_0\rangle$ such that S is the set of *places*, T is the set of *transitions*, $S \cap T = \emptyset$, W is a function from $(S \times T) \cup (T \times S)$ to N, and M_0 is the *initial marking* (*initial state*), $M_0 \in \mathcal{M}$ where $\mathcal{M}$ is the set of *markings* (*states*), i.e. functions from S to N. The net is *finite* iff $S \cup T$ is finite. If $x \in S \cup T$, then the set of *input elements* of x is ${}^\bullet x = \{y \mid W(y,x) > 0\}$, the set of *output elements* of x is $x^\bullet = \{y \mid W(x,y) > 0\}$, and the set of *adjacent elements* of x is $x^\bullet \cup {}^\bullet x$. A transition t *leads (can be fired) from a marking M to a marking M'* ($M[t\rangle M'$ for short) iff
$\forall s \in S\ M(s) \geq W(s,t) \wedge M'(s) = M(s) - W(s,t) + W(t,s)$.

A transition t is *enabled at a marking* M iff t leads from M to some marking. A marking M is *terminal* iff no transition is enabled at M. $\Box$

In our figures, places are circles, transitions are rectangles, and the initial marking is shown by the distribution of tokens, black dots, onto places. A directed arc, i.e. an arrow, is drawn from an element x to an element y iff x is an input element of y. Then $W(x, y)$ is called the *weight* of the arc. As usual, the weight is shown iff it is not equal to 1.

Definition 4.2 Let $\langle S, T, W, M_0 \rangle$ be a place/transition net. The set T^* (respectively, T^ω) is called the *set of finite* (respectively, *infinite*) *transition sequences of the net.* Let f be a function from $\mathcal{M}$ to 2^T. A finite transition sequence σ *f-leads (can be f-fired) from a marking M to a marking M'* iff $M[\sigma\rangle_f M'$, where $\forall M \in \mathcal{M}\ M[\varepsilon\rangle_f M$, and
$\forall M \in \mathcal{M}\ \forall M' \in \mathcal{M}\ \forall \delta \in T^*\ \forall t \in T$
$M[\delta t\rangle_f M' \Leftrightarrow (\exists M'' \in \mathcal{M}\ M[\delta\rangle_f M'' \wedge t \in f(M'') \wedge M''[t\rangle M')$.
A finite transition sequence σ is *f-enabled at a marking M* ($M[\sigma\rangle_f$ for short) iff σ f-leads from M to some marking. An infinite transition sequence σ is f-enabled at a marking M ($M[\sigma\rangle_f$ for short) iff all finite prefixes of σ are f-enabled at M. A marking M' is *f-reachable from a marking M* iff some finite transition sequence f-leads from M to M'. A marking M' is an *f-reachable marking* iff M' is f-reachable from M_0. The *f-reachability graph* of the net is the pair $\langle V, A \rangle$ such that the set of vertices V is the set of f-reachable markings, and the set of edges A is $\{\langle M, t, M' \rangle \mid M \in V \wedge M' \in V \wedge t \in f(M) \wedge M[t\rangle M'\}$. $\Box$

Let Ψ be the function from $\mathcal{M}$ to 2^T such that for each marking M, $\Psi(M) = T$. From now on in this paper, we use a plain "$\rangle$" instead of "$\rangle_\Psi$", and as far as the notions of Definition 4.2 are concerned, we replace "Ψ-xxx" by "xxx" (where xxx is any word), with the exception that the Ψ-reachability graph of the net is called the *full reachability graph* of the net. When f is clear from the context or is implicitly assumed to exist and be of a kind that is clear from the context, then the f-reachability graph of the net is called the *reduced reachability graph* of the net.

Definition 4.3 defines functions needed in the definition of stubbornness. Intuitively, $E_1(M, s)$ is the set of transitions that could increase the contents of s and are not disabled by s at M. Correspondingly, $E_2(M, t, s)$ is the set of transitions that could decrease the contents of s or get disabled because of the firing of t at M. Respectively, $E_3(M, t, s)$ is the set of transitions that are not disabled by s at M and could increase the contents of s or have a greater output flow to s than t has. Finally, $E_4(s)$ is the set of transitions that could decrease the contents of s.

Definition 4.3 Let $\langle S, T, W, M_0 \rangle$ be a finite place/transition net. The function E_1 from $\mathcal{M} \times S$ to 2^T, the functions E_2 and E_3 from $\mathcal{M} \times T \times S$ to 2^T, and the function E_4 from S to 2^T are defined as follows. Let $M \in \mathcal{M}$, $t \in T$, and $s \in S$. Then

$$\begin{aligned}
E_1(M,s) &= \{t' \in {}^\bullet s \mid M(s) \geq W(s,t') \wedge W(t',s) > W(s,t')\}, \\
E_2(M,t,s) &= E_4(s) \cup \{t' \in s^\bullet \mid \ W(s,t) > W(t,s) \wedge \\
&\qquad\qquad\qquad\qquad W(s,t') > M(s) - W(s,t) + W(t,s)\}, \\
E_3(M,t,s) &= E_1(M,s) \cup \{t' \in {}^\bullet s \mid \ M(s) \geq W(s,t') \\
&\qquad\qquad\qquad\qquad \wedge W(t',s) > W(t,s)\}, \text{ and} \\
E_4(s) &= \{t' \in s^\bullet \mid W(s,t') > W(t',s)\}.
\end{aligned}$$

□

We define stubbornness in terms of and/or-graphs, instead of taking the extra pain of a translation. Definition 4.4 has the same effect as Definition 4.21 of [15] which in turn is close to (but not equivalent to) Definition 2.3 of [12].

Definition 4.4 Let $\langle S, T, W, M_0 \rangle$ be a finite place/transition net and M a nonterminal marking of the net. The *basic and/or-graph at M* is the and/or-graph $\langle V_\otimes, V_\oplus, \kappa, F \rangle$ such that the set of and-vertices $V_\otimes$ is

$$\begin{aligned}
&\{s \mid \exists t \in T \ s \in {}^\bullet t \wedge M(s) < W(s,t)\} \cup \{t \in T \mid M[t\rangle\} \cup \\
&\{\langle t, s, i\rangle \mid t \in T \wedge M[t\rangle \wedge s \in {}^\bullet t \wedge W(s,t) > W(t,s) \wedge i \in \{2,3\}\} \cup \\
&\{\langle \kappa, t\rangle \mid t \in T \wedge M[t\rangle\},
\end{aligned}$$

the set of or-vertices $V_\oplus$ is

$$\{t \in T \mid \neg M[t\rangle\} \cup \{\langle t, s\rangle \mid t \in T \wedge M[t\rangle \wedge s \in {}^\bullet t \wedge W(s,t) > W(t,s)\} \cup \{\kappa\},$$

κ is not expressible in terms of the elements of the net, and the set of edges F is

$$\begin{aligned}
&\{\langle t, s\rangle \mid t \in T \wedge s \in {}^\bullet t \wedge M(s) < W(s,t)\} \cup \\
&\{\langle s, t'\rangle \mid \exists t \in T \ s \in {}^\bullet t \wedge M(s) < W(s,t) \wedge t' \in E_1(M,s)\} \cup \\
&\{\langle \langle t, s\rangle, \langle t, s, i\rangle\rangle \mid \ t \in T \wedge M[t\rangle \wedge s \in {}^\bullet t \wedge W(s,t) > W(t,s) \wedge \\
&\qquad\qquad\qquad\qquad i \in \{2,3\}\} \cup \\
&\{\langle t, \langle t, s\rangle\rangle \mid t \in T \wedge M[t\rangle \wedge s \in {}^\bullet t \wedge W(s,t) > W(t,s)\} \cup \\
&\{\langle \langle t, s, i\rangle, t'\rangle \mid \ t \in T \wedge M[t\rangle \wedge s \in {}^\bullet t \wedge W(s,t) > W(t,s) \wedge \\
&\qquad\qquad\qquad\quad i \in \{2,3\} \wedge t' \in E_i(M,t,s)\} \cup \\
&\{\langle \kappa, \langle \kappa, t\rangle\rangle \mid t \in T \wedge M[t\rangle\} \cup \\
&\langle \langle \kappa, t\rangle, t'\rangle \mid t \in T \wedge M[t\rangle \wedge (t' = t \vee \exists s \in {}^\bullet t \quad t' \in E_4(s))\}.
\end{aligned}$$

A set $T_s \subseteq T$ is *stubborn at M* iff T_s is a T-solid set of vertices. A function f from $\mathcal{M}$ to 2^T is a *stubborn set generator* iff for each nonterminal marking M', $f(M')$ is stubborn at M'. □

The stubborn set method constructs an f-reachability graph for some stubborn set generator f. If $T_e = \{t \in T \mid M[t\rangle\}$, we can simply say that we actually want a T_e-solid set which is as small as possible.

When stubborn sets satisfying some additional constraints are wanted, the additional constraints should be included in some way in the and/or-graphs. The attribute “basic” in Definition 4.4 indicates that no additional constraint is included. Many typical constraints, even non-local conditions such as the search stack condition in [1], can be included in the and/or-graphs. (The stack condition can be transformed into a local condition by “marking” the enabled transitions which lead from the current state to the states in the stack.) If for some reason, some constraint is not included, the and/or-graph abstraction remains unaware of the constraint, but it is of course possible to develop algorithms that use the abstraction in some subproblems only, as is done in [15].

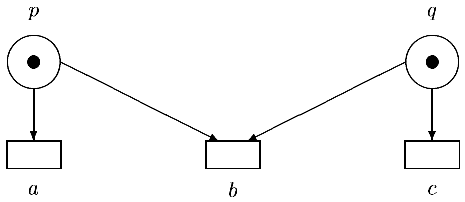

Figure 2. The stubborn sets at M_0 are $\{a, b\}$, $\{b, c\}$ and $\{a, b, c\}$.

Let us consider the net in Figure 2, the initial marking M_0 being displaued. The basic and/or-graph at M_0 is presented in Figure 3. From the basic and/or-graph it follows that the stubborn sets at M_0 are $\{a, b\}$, $\{b, c\}$ and $\{a, b, c\}$.

If the outgoing edges of the immediate successor vertices of κ in the definition of the basic and/or-graph did not exist, T-solidity would be *semistubborrness* [12]. The definition of stubbornness in [10] actually considers semisstubborn sets as stubborn sets. Since the empty set is semistubborn, some constraint is needed to prevent the empty set from being chosen too easily, but such constraints are defined in all of the analysis problems considered in [10].

5. Conclusions

The transformation of a stubborn set computation problem into a stable model computation problem can be made on a tool level. Since August 1999, the reachability analysis tool PROD [15, 16] has had an option which makes a reachability graph generator program call Smodels whenever a stubborn set is to be computed. However, it is

vertex	kind	immediate successors	vertex	kind	immediate successors
κ	or	$\langle\kappa,a\rangle$, $\langle\kappa,b\rangle$, $\langle\kappa,c\rangle$	$\langle\kappa,a\rangle$	and	a, b
$\langle\kappa,b\rangle$	and	a, b, c	$\langle\kappa,c\rangle$	and	b, c
a	and	$\langle a,p\rangle$	b	and	$\langle b,p\rangle$, $\langle b,q\rangle$
c	and	$\langle c,q\rangle$	$\langle a,p\rangle$	or	$\langle a,p,2\rangle$, $\langle a,p,3\rangle$
$\langle b,p\rangle$	or	$\langle b,p,2\rangle$, $\langle b,p,3\rangle$	$\langle b,q\rangle$	or	$\langle b,q,2\rangle$, $\langle b,q,3\rangle$
$\langle c,q\rangle$	or	$\langle c,q,2\rangle$, $\langle c,q,3\rangle$	$\langle a,p,2\rangle$	and	a, b
$\langle b,p,2\rangle$	and	a, b	$\langle b,q,2\rangle$	and	b, c
$\langle c,q,2\rangle$	and	b, c	$\langle a,p,3\rangle$	and	none
$\langle b,p,3\rangle$	and	none	$\langle b,q,3\rangle$	and	none
$\langle c,q,3\rangle$	and	none			

Figure 3. The basic and/or-graph of the net of Figure 2 at M_0.

equally fruitful to take algorithms from the "stable model side" and implement them in any tool. One interesting research problem is how to handle a case where a net has enormously many transitions, possibly enormously many of them being enabled simultaneously.

Acknowledgements

The translation from and/or-graphs to logic programs was assisted by M.Sc.(Eng.) Keijo Heljanko, Docent Dr.Tech. Ilkka Niemelä, Lic.Tech. Patrik Simons, and Stud.Tech. Tommi Syrjänen (each being from HUT/TCS, i.e. from the same laboratory as the author). A reference to [9] was found by M.Sc.(Eng.) Tommi Junttila (from HUT/TCS).

The work has been supported by The Technology Development Centre of Finland, Nokia Research Center, Nokia Telecommunications, Helsinki Telephone Corporation, and Finnish Rail Administration.

References

1. R. Gerth, R. Kuiper, D. Peled, and W. Penczek, "A Partial Order Approach to Branching Time Logic Model Checking," *Information and Computation*, Vol. 150, No. 2, pp. 132–152, May 1999.
2. P. Godefroid, *Partial-Order Methods for the Verification of Concurrent Systems — An Approach to the State-Explosion Problem*, LNCS 1032, Springer-Verlag, 1996, 143 p.
3. B.W. Kernighan and D.M. Ritchie, *The C Programming Language*, 2nd edition, Prentice-Hall, 1988, 272 p.
4. I. Kokkarinen, D. Peled, and A. Valmari, "Relaxed Visibility Enhances Partial Order Reduction," in O. Grumberg (Ed.), *Computer Aided Verification (CAV '97)*, LNCS 1254, Springer-Verlag, 1997, pp. 328–339.
5. I. Niemelä, "Logic Programs with Stable Model Semantics as a Constraint Programming Framework," in *Proceedings of the Workshop on Computational Aspects of Nonmonotonic Reasoning*, Trento, Italy, 1998, pp. 72–79.
6. I. Niemelä, *Logic Programs with Stable Model Semantics as a Constraint Programming Paradigm*, manuscript (an extended version of [5]).
7. D. Peled, "All from One, One for All: on Model Checking Using Representatives," in C. Courcoubetis (Ed.), *Computer Aided Verification (CAV '93)*, LNCS 697, Springer-Verlag, 1993, pp. 409–423.
8. W. Reisig, *Petri Nets: An Introduction*, EATCS Monographs on Theoretical Computer Science, Vol. 4, Springer-Verlag, 1985, 161 p.
9. S. Sahni, "Computationally Related Problems," *SIAM Journal on Computing*, Vol. 3, No. 4, pp. 262–279, December 1974.
10. K. Schmidt, "Stubborn Sets for Standard Properties," in S. Donatelli and J. Kleijn (Eds.), *Application and Theory of Petri Nets 1999*, LNCS 1639, Springer-Verlag, 1999, pp. 46–65.
11. P. Simons, *Towards Constraint Satisfaction Through Logic Programs and the Stable Model Semantics*, Helsinki University of Technology, Digital Systems Laboratory Report A 47, 1997, 49 p.
12. A. Valmari, "Error Detection by Reduced Reachability Graph Generation," in *Proceedings of the IX European Workshop on Applications and Theory of Petri Nets*, Venice, Italy, 1988, pp. 95–112.
13. A. Valmari, "Heuristics for Lazy State Space Generation Speeds up Analysis of Concurrent Systems," in M. Mäkelä, S. Linnainmaa, and E. Ukkonen (Eds.), *STeP-88 (Proceedings of the Finnish Artificial Intelligence Symposium)*, Vol. 2, Helsinki, 1988, pp. 640–650.
14. A. Valmari, *State Space Generation: Efficiency and Practicality*, Doctoral thesis, Tampere University of Technology, Publications 55, 1988, 170 p.
15. K. Varpaaniemi, *On the Stubborn Set Method in Reduced State Space Generation*, Doctoral thesis, Helsinki University of Technology, Digital Systems Laboratory Report A 51, 1998, 105 p.
16. Worldwide web, page http://saturn.tcs.hut.fi/pub/prod/index.html.
17. Worldwide web, page http://saturn.tcs.hut.fi/pub/smodels/index.html.