*Electronic Notes in Theoretical Computer Science 50 No. 4 (2001) { Proc. VEPAS 2001 URL:* [*http://www.elsevier.nl/locate/entcs/volume50.html*](http://www.elsevier.nl/locate/entcs/volume50.html) *15 pages*

*An Assertional Language for the Veri cation of* Systems Parametric in Several Dimensions (Preliminary Results)

*G. Delzanno 1*

*Dipartimento di Informatica e Scienze dell'Informazione Universit a di Genova, Via Dodecaneso 35, 16146 Genova, Italy*

*Abstract*

*We propose a rich assertional language to be used for symbolic veri cation of systems with several parametric dimensions. Our approach combines notions coming from dif- ferent elds. We use Colored Petri Nets [16] to describe nets of processes carrying structured data. We combine concepts coming from constraint programming [23] and multiset rewriting [19] to nitely and concisely represent transitions and in nite col- lection of states of Colored Petri Nets. Finally, we incorporate these concepts in the veri cation technique based on backward reachability and upward-closed sets of [1,12]. We obtain a procedure that can be used as an automatic support for attacking pa- rameterized veri cation problems. We apply these ideas to verify safety properties of a parameterized mutual exclusion algorithm. A number of open questions arise from our preliminary experiments, nding an adequate counterpart of our framework in the world of automated deduction being among the more interesting ones.*

# *1 Introduction*

*Given a speci cation of a concurrent system, the challenge of parameterized veri-*

*cation is to verify a property for any of its possible initial con gurations see (e.g.* [2,6,10,8,14]). To illustrate the problem on a practical example, let us consider Petri Nets as abstract model of concurrent systems. Via the abstraction that maps processes to tokens, we can describe the behaviour of our system via places and transitions, and a possible initial con guration as an instance of a parametric initial marking (e.g., where K 1 tokens are in a given place). The goal here is to verify a given property for any value of the parameters in the initial marking (e.g. for any value of K). For safety properties, the previous problem can be

*1 Email:* [*giorgio@disi.unige.it*](mailto:giorgio@disi.unige.it)

*c 2001 Published by Elsevier Science B. V. Open access under* [*CC BY-NC-ND license*](http://creativecommons.org/licenses/by-nc-nd/3.0/)*.*

*solved using techniques like Karp-Miller's coverability tree [17]. Parameterized* veri cation problems for a wider range of concurrent systems like Lossy FIFO Systems [1,12], Timed Petri Nets [2], Lossy Vector Addition Systems [4], Broad- cast Protocols [11], and Coherence Protocols [8] can be solved using the theory of well-structured systems [1,12]. This approach ts well in the general framework of symbolic model checking with rich assertional languages proposed in [18]. An assertional language is a formalism used to symbolically represent sets of states (e.g. BDDs for nite-state veri cation) and equipped with algorithmic procedures for the operations needed to automatically reason on the speci cation (e.g. to compute preconditions). The method of [1] combines in fact backward reachabil- ity analysis and symbolic representations of in nite collection of states. One of the more interesting feature of the `backward' approach comes from the following observation: violations of safety properties can often be represented via upward- closed sets of states. This is a desirable property since in several cases it allows to give nite representations of in nite set of states. For instance, an upward-closed set of Petri Net markings can be represented via the nite collection of its minimal points. Despite of all these nice properties, models based on Petri Nets are often too abstract: processes here are represented as black tokens. However, in many practical cases like mutual exclusion and security protocols processes carry along information like identi ers or time-stamps that are essential for establishing the correctness of the whole system. When data cannot be abstracted away, we can say that a speci cation becomes parametric in several dimensions (e.g. number of processes and values of the time-stamps on each process as in [2]). Though in the general case fully automatic veri cation of parameterized problems becomes impossible, porting the backward approach to systems with several parametric dimensions is a possible way to enrich the set of tools (e.g. automated deduc- tion, bounded model checking, simulation, and testing) used to reason over these complex systems.

*Following this line of thoughts and taking inspiration from [2,9], in this paper*

*we propose an assertional language to symbolically represent in nite collections* of states for systems parametric in several dimensions. For this purpose, we found technically convenient to combine notions coming from di erent elds like high level Petri Nets [16], constraints [23], multiset rewriting [19], and Constraint Logic Programming [15]. In particular, the notion of constraint is central to our construction. Following the terminology of [15], constraints can be viewed as formulas interpreted over a xed domain, and for which one uses specialized decision procedures, e.g., to test satis ability, and entailment. More in detail, our approach works as follows. To smoothly extend the ideas used in [1,2,8,14], we adopt Colored Petri Nets (CPNs) [16] as our general model of parameterized concurrent systems. Basically, CPNs are Petri Nets in which tokens can be colored with structured data. Di erently from [16] however, our presentation of CPNs is based on a combination of constraints with a restricted form of multiset rewriting. As shown in [19], multiset rewriting allows one to locally specify the behaviour of

*processes in a natural way. Annotating multiset rewriting rules with constraints* allows us to nitely and concisely (e.g. without the need of axioms for arithmetic operations) represent transition relations of CPNs.

*Building upon these ideas, we will introduce the new concept of contrained* con guration as a symbolic representation of upward-closed sets of con gurations of CPNs: here constraints are used to nitely represent data attached to processes, whose minimal distribution on the net is described via a multiset of atomic for- mulas. Constrained con gurations play the role of the minimal points used to represent upward-closed sets of Petri Nets markings. Following [18], in order to make an assertional language out of the previous notions, we need symbolic oper- ations to manipulate `properties', i.e., to compute the pre-image and the union of sets of CPNs states, and for checking containment bewteen sets of CPNs states. In this paper we will show how to build these operations upon the operations of the underlying constraint language. As for CLP systems [15], our assertional language is de ned for any constraint language equipped with the operations of entailment, satis ability and existential quanti cation. Our assertional language can be used as `constraint system' in the veri cation framework of [1]. This way we obtain a new backward reachability procedure, that represents an alternative to `forward' techniques based on the construction of the occurrece graph of CPNs [16]. Given the generality of the approach, the resulting procedure is not guar- anteed to terminate. However in the same philosophy as simulation, testing, and bounded model checking, it can be used as an automatic support for the analysis of complex protocols. Furthermore, the procedure is robust w.r.t. the backward approach for Petri Nets: if we abstract away the data from all components of our assertional language, we obtain the backward reachability algorithm for Petri Nets, whose termination is guaranteed by Dicskon's lemma [1,12]. Following [9], we have implemented the library of symbolic operations and the backward reach- ability procedure using a CLP system [15] equipped with a constraint solver for linear arithmetic constraints over the reals. As a preliminary experiment, we proved safety properties for a distributed test-and-lock protocol parameterized on the number of processes, monitors, and resource identi ers.

*In our view, multiset rewriting is a possible way to connect parameterized*

*veri cation and automated deduction. In fact, at least in principle, it is possible* to express veri cation problems as the ones considered in the paper using AC Rewriting and specialized theories to represent and handle colors. As we will discuss at the end of the paper, it turns out, however, that the strategies used in existing automated deduction tools are not well-suited to solve problems like the one we are interested in (e.g. computing P re starting from and upward closed set of states). We believe that the application of automated theorem provers to solve parameterized veri cation problems, as well as nding decidable fragments and accelerations techniques are among the more interesting future direction of our research. In Section 2 and Section 3, we will brie y recall all concepts we need for our assertional language. The language is de ned in Section 4. In Section *5*

*we will discuss how to incorporate the language in the backward approach, and* discuss the analysis of our case-study. In Section 6 we discuss related works and discuss a number of open problems we plan to address in our future work. Finally, in Section 7 we will draw some conclusions.

# *2 Constraints, Multiset Rewriting, and Colored Nets*

*Our presentation of CPNs is technically but not substantially di erent from* Jensen's de nition in [16]. Let us rst introduce the notion of constraint, and consider a restricted form of multiset (or AC) rewriting.

*Constraints. Following [15], a constraint ' is a conjunction c1;::: ; cn of atomic* formulas with free variables (implicitly existentially quanti ed) from a denu- merable set V. The interpretation domain D of the constraints is xed a pri- ori. As an example, linear arithmetic constraints are conjunctions of formulas k1 x1 + ::: + kn xn rel k, where k, and ki are an integer constants, xi 2 V r i : 1;::: ;n and rel is a relational operator. An evaluation is an assignment which maps variables in V to values in D. The application of an evaluation is extended to a formula ' containing variables in V, written ('), in the natural way. A solution for a constraint ' is an evaluation (restricted to the variables in ') such that (') true (in the example hx; yi ; h2; 1i is a solution for '). The set of solutions of a constraints is denoted as Sol('). In this paper we will restrict our attention to constraint languages and domains equipped with decision procedures for testing entailment and satis ability. These operations are de ned as follows: ' is satis able if and only if Sol(') 6= ;; ' entails if and only if Sol(') Sol( ). Furthermore, we will assume to have an algorithm for variable elimination such that given ' and a variable x it returns a constraint 9x:' whose solutions are obtained by projecting the solutions of ' over V ar(') n fxg. For in- stance, let ' be x y; x 1;y 1, then ' is satis able and entails x 1;y 1; furthermore 9y:' is equivalent to the constraint x 1. For linear constraints, we can use the Symplex and Fourier-Motzkin as procedures for satis ability, entail- ment, and variable elimination [23].

*Multiset Rewriting. In this paper we consider the following restricted form of* rewriting. Let P be a nite set of symbols, V be a denumerable set of vari- ables, and D be a set of values. An atomic formula p(~x) is such that p 2 P, and ~x = hx1;::: ; xni is a vector of variables in V. A ground atomic formula is obtained by substituting values D to variables. We will use j as multiset constructor (associative and commutative), being the empty multiset. Further- more, will denote the multiset union; and will denote the multiset di erence (e.g. (pjpjq) (p) = (pjq)). In the rest of the paper we will use M, N , ::: to denote multisets of atomic formulas. A multiset rewriting rule annotated with constraints has the following form:

*p1(x~1) j ::: j pn(x~n) ! q1(y~1) j ::: j qm(y~m) : ';*

*where pi; qj 2 P, and ' is a constraint with variables in x~1;::: ; x~n; y~1;::: ; y~m.* A rule M ! M0 : ' denotes a (possibly) in nite collection of ground rules formally de ned as follows:

*0 0*

*Inst(M ! M : ') = f (M) ! (M ) j 2 Sol(')g:*

*For instance, let R be the rule p(x) j r(y) ! t(x0 ) : x 1;y 0; x0 = x 1,* then, e.g, p(1) j r(0) ! t(0) 2 Inst(R). Finally, given two multisets of ground atoms M1 and M2, one step of rewriting is de ned via the following relation:

*M1 ) M2 if and only if there exists a multiset of ground atomic formulas Q*

*s.t. M1 = N1 Q, M2 = N2 Q, and N1 ! N2 2 Inst(R).*

*Colored Petri Nets (CPNs). Based on the previous de nition, a Colored Petri Net* can be viewed as a tuple hP; Ri, where P is a set of predicates representing place names, and R is a set of multiset rewriting rules representing transitions. In the rest of the paper we will call con guration a multiset of ground atomic formulas. Con gurations are the `generalization' of Petri Net markings, where a token in

*place p and data d~ is represented via the atomic formula p(d~). Multiset rewrit-*

*ing rules allow us to locally model rendez-vous and internal actions of processes* (tokens), independently from the global state of the system. The constraint in a transition denotes the relation between the the data of di erent tokens. A transi- tion can be red only if the current con guration satis es '. By construction, a CPN implicitly de nes a family of transition systems, each one obtained by xing an initial con guration. An initialized CPN is a tuple hP; R; Ii where hP; Ri is a CPN, and I is the set of initial con gurations. The occurrence sequences (runs) of an initialized CPN are then de ned via the rewriting relation ) de ned in the previous section. Let S be a set of con gurations, then the immediate successor operator is de ned as

*P ost(S) = f M j M ) M ; M 2 S g;* whereas the immediate predecessor operator P re is de ned as

*0 0*

*P re(S) = f M j M ) M ; M 2 S g:*

*0 0*

*Finally, the reachability set is de ned using the transitive closure of the P ost* operator as follows O = P ost (I):

*2.1 A Case-study.*

*We consider here a distributed test-and-lock protocol for a net with multiple re-* sources each one controlled by a monitor. Each resource is labelled with a noneg- ative integer. A process can non-deterministically request any resource. We use the predicates think, wait(x), and use(x) (x=resource id) to denote the current state of processes, while a predicate m(x; t) is used to specify a monitor for the resource x with value t for the semaphor. As constraint language, we use linear constraints interpreted over nonegative numbers. We give the speci cation in two steps. First, we specify all the initial con gurations using the predicate start as

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *follows.* | *start* | *!* | *start j think :* | *true:* | *(1)* |
|  | *start* | *!* | *start j m(x; t) :* | *x 0;t = 0:* | *(2)* |
|  | *start* | *!* | *:* | *true:* | *(3)* |

*A run from start generates a con guration with an arbitrary number of thinking* processes, and idle monitors. As from the rule (3), start is not needed after the initialization phase. The core of the protocol is as follows:

*think ! wait(x) : x 0: (4)*

*wait(x) ! think : x 0: (5)* wait(x) j m(x; t) ! use(x) j m(x; t0 ) : t = 0; t0 = 1: (6)

*use(x) j m(x; t) ! think j m(x; t0 ) : t = 1; t0 = 0: (7)*

*The rules work as follows: (4) a thinking process asks for a resource with identi-*

*er x (a nonnegative integer, chosen non-deterministically), moving to the state* wait(x); (5) waiting processes can choose to go back thinking; (6) a resource x is assigned to a waiting process provided the semaphore of its monitor is not locked (i.e. the monitor is idle); (7) when a process releases the resource the monitor resets the lock. Note that in the previous speci cation we simply express the local interaction between one process and one monitor (the power of multiset rewrit- ing). The initial con guration of the protocol consists of the con guration start, that in turn generates a con gurations in which all processes are thinking, and all monitors are idle. For instance, a possible run from start is as follows (we will use )i to indicate the application of rule (i)):

*start )1 start j think )2 start j think j m(1; 0) )1 start j think j think j m(1; 0)*

*)2 start j think j thinkj m(1; 0) j m(4; 0) )3 think j thinkj m(1; 0) j m(4; 0):*

*Starting form the initial con guration of the previous example, a run involving* the other rules is as follows.

*think j think j m(1; 0) j m(4; 0) )4 wait(1) j think j m(1; 0) j m(4; 0)*

*)6 use(1) j think j m(1; 1) j m(4; 0) )4 use(1) j wait(1) j m(1; 1) j m(4; 0):*

# *3 Parameterized Veri cation*

*In this paper we will restrict ourselves to consider veri cation of safety properties.* A safety property for a CPN with initial con gurations I can be represented via two sets Sgood and Sbad, that represent the con gurations that respectively do and do not satisfy the property. In the forward approach to veri cation (in the style of Karp and Miller [17,10,16]) one tries to prove that P ost (I) Sgood, whereas in the backward approach of [1,12] one tries to prove that P re (Sbad) \

*I = ; (where P ost =P re is the transitive closure of P ost=P re). When applied* to parameterized veri cation, the backward approach has two advantages: (1) computing P re can be done independently from the initial con guration; (2) bad con gurations often form upward-closed sets. Let us explain this intuition using our example. The safety property of the multiple test-and-lock protocol is that only one process per time can use a given resource; thus, Sbad is the set of con gurations containing the minimal violations: use(n) j use(n) for some resource-id n. To formalize the idea of `minimality' of a set, let us introduce the following ordering between con gurations:

*M 4 N if and only if OccA(M) OccA(N ) for any atom A;*

*where OccA(M) is the number of occurrences of A in M (e.g. Occuse(1)(M) = 2* for M = use(1) j use(1) j think(2)). A set of con gurations S generates its upward closure U p(S) de ned as follows:

*U p(S) = fN j M 4 N ; M 2 Sg:*

*A set S is upward-closed whenever U p(S) = S (e.g. Sbad in our example). Upward-* closed sets of con gurations have interesting properties w.r.t. the predecessor operator P re of a CPN.

*Proposition 3.1 U p(P re(S)) P re(U p(S)) for any set S of con gurations.*

*In general the reverse implication does not hold. As a counter-example, simply* take the rule p ! q1 j q2 and the singleton set S consisting of the multiset q1. Then, P re(S) = ;, whereas the multiset p belongs to P re(U p(S)). However, the following property holds.

*Corollary 3.2 If S is upward-closed, then U p(P re(S)) = P re(U p(S)).*

*In other words, the class of upward-closed sets of con gurations is closed under* the computation of the pre-image. In the following section we will try to exploit these properties to de ne assertions for veri cation of CPNs.

# *4 The Assertional Language*

*Following [18], a rich assertional language (in the context of automatic, or semi-* automatic veri cation) should allow one to symbolically represent and manipulate properties expressed via (possibly in nite) sets of states. Boolean formulas and BDDs are classical examples from nite-state veri cation. In our setting we will use constraints, this time to nitely represent minimal con gurations of upward- closed sets. For this purpose, we introduce the notion of constrained con guration de ned as

*p1(x~1) j ::: j pn(x~n) : '*

*where p1;::: ; pn 2 P, and ' is a constraint over the variables x~1;::: ; x~n (free vari-* ables are implicitly existentially quanti ed). Given a constrained con guration M : ' the set of its ground instances is de ned as

*Inst(M : ') = f (M) j 2 Sol(')g:*

*This de nition can be extended to sets of constrained con gurations with disjoint* variables (indicated as S; S0 ;:: :) in the natural way. For instance, if S is the singleton containing use(x)juse(y) : x = y, then Inst(Sbad) is the set of con gurations having the form use(n)juse(n) for any n. Note that Inst(Sbad) does not model our intuition that constrained con gurations should generate all violations. Thus, instead of taking the set of instances as ` at' denotation of a set of constrained con guration S, we choose the rich denotation:

*bad*

*[[S]] = U p(Inst(S)):*

*For instance, in our rst example we have that Sbad = [[Sbad]] as desired. In the rest* of this section we will show how to formulate operations on sets of con gurations at the symbolic level of constrained con gurations w.r.t. the `rich' denotation [[ ]]. We anticipate here that all operations will be parametric on the constraint language used in the speci cation.

*Pre-image computation. In order to handle our symbolic representation, we need* a new operator Pre such that [[Pre(S)]] = P re([[S]]). We rst introduce a new operator working on sets of con gurations as follows:

*P re(S) = f A M0 j A ! B 2 Inst(R); M 2 S; M0 = M B g:*

*Intuitively, P re treats a con gurations as `representation' of its upward-closure.* For instance, consider D = f0; 1g, and the rule p(x) ! q(x) : true. Then, P re(fq(1)g) returns p(1) as well as p(0) j q(1). In fact, p(0) rewtites into q(1), whereas p(0) j q(1) rewrites into q(0) j q(1) that belongs to the upward-closure of q(1). The new operator satis es the following property.

*Proposition 4.1 P re(U p(S)) U p(P re(S)) for any set S of con gurations.*

*To lift the de nition of P re to the symbolic level, we introduce the notion of* uni cation between constrained con gurations:

*(p1(x~1) j ::: j pn(x~n) : ') uni es with (q1(y~1) j ::: j qm(y~m) : ) via*

*if and only if: (i) m = n; (ii) there exist two permutations i1;::: ; in and j1;::: ; jn* of 1;::: ;n such that pik = qjk , and x~ik and y~jk are tuples of the same size for

*k : 1;::: ; n; (iii) the constraint de ned as ' ^ ^n*

*k=1*

*x~ik*

*= y~ik*

*is satis able*

*w.r.t. the domain D taken into consideration.*

*The operator Pre is de ned on a set S containing constrained multisets (with* disjoint variables) as follows

*Pre(S) = f (A N : 9~x: ) j (A ! B : ) 2 R; (M : ') 2 S;*

*M0 4 M; B0 4 B;*

*(M0 : ') uni es with (B0 : ) via ; N = M M0 ;*

*and ~x are all variables not in A N g:*

*For instance, consider p(x; y) ! q(x; y) : x 0;y = 1. Given the singleton S* with q(u; w) : u = 1;w 0, Pre(S) should contain p(x; y) : x = 1;y = 1 as well as p(x; y) j q(u; w) : x 0;y = 1;u = 1;w 0 (e.g., p(4; 1) j q(1; 5) rewrites into q(4; 1) j q(1; 5) 2 [[S]].) The latter constrained multiset can be obtained by setting M0 = B0 = (the empty multiset) when applying Pre to S. The new operator enjoys the following properties

*Proposition 4.2 For any CPN and any set S of constrained con gurations, P re(Inst(S)) Inst(Pre(S)) and Inst(Pre(S)) U p(P re(Inst(S))).*

*In general Inst(Pre(S)) P re(Inst(S)) does not hold. As a counter-example,* take D = f0; 1g, the rule p(x) ! q(x) : true, and a set S containing (q(y) : y =

*0). Then, p(x) j q(y) : y = 0 (x free) is in Pre(S), while its instance p(0) j q(0)* is not in P re(Inst(S)) (while it is in U p(P re(Inst(S)))). From the previous proposition, it follows however that

*Theorem 4.3 [[Pre(S)]] = P re([[S]]) for any S.*

*It is easy to verify that [[S1]] [ [[S2]] = [[S1 [ S2]]. As a consequence of the* previous observation and of Theorem 4.3, it follows that we can use Pre in order to symbolically compute P re (S) = S [ P re(S) [ ::: (the transitive closure of P re). Formally, if S is the symbolic representation of the upward-closed set S (i.e. S = [[S]]), then [[Pre (S)]] = P re ([[S]]).

*An E ective Containment Test. To build a procedure for computing (or simply* approximating) Pre (S), we need to de ne a procedure to check the containment of intermediate results. Speci cally, given two sets S and S0 , we need to test whether [[S]] [[S0 ]] holds. We rst note that the previous condition cannot be tested pointwise as shown by the following counter-example. Let S1 be the singleton containing the constrained con guration p(x) : x 0, and S2 be the set containing p(x) : x = 0 and p(x) : x 1. Let D be the set of nonnegative integers. Clearly, [[S1]] [[S2]] holds, however there exists no M 2 S2 such that [[p(x) : x 0]] [[M]]. Though the pointwise subsumption test is not complete, it gives us a suÆcient condition to check containment between the denotations of sets of con gurations. Furthermore, we can make it `e ective' by introducing

*P roc Pre (U : set of constrained con gurations) S := U; R := ;;*

*while S 6= ; do*

*remove (M : ') from S;*

*if there are no (N : ) 2 R s:t: (M : ') entails (N : ) then add (M : ') to R and set S := S [ Pre(fM : 'g);*

*endif ; end while:*

*Fig. 1. Symbolic backward reachability for CPNs. the following operator working on constrained con gurations:*

*(M : ') entails (N : ) if and only if there exists 9M0 s.t. M0 4 M, (M0 : ')*

*uni es with (N : ) via , and (9~x: ) entails , ~x being all variables not* occurring in M0 and N .

*For instance, p(x) j q(z) : x 1;z 1 entails p(x0 ) : x0 0, since the constraint* 9z:x 1 entails x 0 (modulo renaming, x 1 is the uni er for p(x) : x *1*

*and p(x0 ) : x0 0). Then, the following proposition holds.*

*Theorem 4.4 Let (M : ') and (N : ) be two constrained con gurations. Sup-* pose (M : ') entails (N : ) holds, then [[M : ']] [[N : ]].

*Note that, the reverse implication does not hold. As a counter-example, con-* sider D as the set of terms built over a, b, and f , and the two constrained con g- urations p(x; y) j p(z; w) : x = a; y = z; w = f (u), and p(x; w) : x = a; w = f (u) (the denotations of the former are contained in those of the latter, while they are not in the entail relation).

# *5 An Automatic Support for Parameterized Veri cation*

*Given the generality of the notion of color, most of the veri cation problems for* CPNs are undecidable. However as advocated in [16], automatic techniques can still be useful as a support for their validation. Automated deduction, bounded model checking, simulation and testing are all examples of techniques one could use in this sense. Our assertional language can incorporated in the procedure of [1], as shown by the program skeleton in Fig. 1. Fixing the depth of the back- ward search, the resulting procedure be used, e.g., as an automatic support for searching bugs. Interestingly, the procedure can be reduced to the usual back- ward reachability algorithm for Petri Nets by abstracting away colors in the data structures and operations. Following [9], we directly implement the operations on constrained multisets, and the procedure of Fig. 1 in the CLP system SICStus

*m1 start : true:*

*m2 think j think j m(x; 0) j m(x; 0) : x 0:*

*m3 use(x) j m(x; 1) j use(x) : x 0:*

*m4 think j m(x; 0) j wait(x) j m(x; 0) : x 0:*

*m5 wait(x) j m(x; 0) j wait(x) j m(x; 0) : x 0:*

*m6 think j m(x; 0) j use(x) : x 0:*

*m7 wait(y) j use(x) j use(x) : x 0;y 0:*

*m8 wait(x) j m(x; 0) j use(x) : x 0:*

*m9 think j use(x) j use(y) : x 0:*

*m10 use(x) j use(x) : x 0:*

*Fig. 2. Pre (S) with S = f use(x) j use(x) : x 0 g.*

*Prolog, providing symbolic manipulations of terms via uni cation, and contraint-* operations for di erent domains (e.g. linear constraints over reals). We used the implementation to study safety properties for our case-study.

*Analysis of the case-study. The goal is to verify mutual exclusion for any num-* ber of processes, monitors, and resources for the distributed test-and-lock pro- tocol of Section 2, i.e., start 62 [[Pre (S)]] where S is the singleton containing use(x) j use(y) : x = y; x 0. On such an input, the procedure of Fig. 1 terminates in 8 steps. Counting all redundancies, the xpoint consists of *413*

*constrained multisets (making manual computations highly error-prone). In our*

*implementation we automatically remove redundant elements. The result consists* of ten elements, as shown in Fig. 2. Observing the history of the backward com- putation, we note that start (generated in the last step) can be derived (applying 4 times Pre) from m2 (generated in the fourth step); however m2 represents ini- tial con gurations only if we let two monitors control the same resource. If we assume that all monitors control distinct resources, then we can discharge m2 and m0 (automatically, if we apply the invariant to cut the search), and con- clude that our protocol ensures mutual exclusion for any number of processes, and resources. Though in the example the precondition on the monitors could be derived by looking at the speci cation, the example shows that the approach could have interesting applications to synthetized invariants for parameterized systems. Finally, it is important to note that the previous property cannot be proved by abstracting away data. In fact, the `abstract' unsafe con gura- tions containing the marking use j use (the abstraction of Sbad) can always be reached from abstractions of admissible con gurations (e.g. from the abstraction of wait(1) j wait(2) j m(1; 0) j m(2; 0)).

# *6 Related Work and Open Questions*

*In their seminal paper [2], Abdulla and Jonsson proposed an assertional language* for Timed Petri Nets in which they use dedicated data structures to symbolically represent markings parametric in the number of tokens and in the age associated to tokens. In [3], Abdulla and Nyl en formulate a symbolic algorithm using di er- ence constraints to represent the state-space of Time Petri Nets. Our approach is an attempt to generalize the ideas of [2,3] to problems and constraint systems that do not depend strictly on time. In our opinion the combination of multiset rewriting and constraints is an elegant way to achieve the goal.

*Relation with Constraint-based Veri cation. Our ideas are related to previous* works connecting Constraint Logic Programming and veri cation [13]. In this setting transition systems are encoded via CLP programs of the form A:-B; ' where A and B are atomic formulas used to encode the global state of a system, and ' is a constraint modeling state updates, see e.g. [9]. Contrary, here we re ne this idea by using multiset rewriting and constraints to locally specify the update of the global state, and the synchronization between di erent processes (i.e., we only keep the information in A and B strictly relevant to the rule by splitting an atomic formula into a multiset of formulas). The notion of constrained multiset extends naturally the notion of constrained atom of [9]. However, the locality in the representation of rules allows us to consider rich denotations (upward- closures) instead of at ones (instances) like in [9]. This way, we can lift the approach to the parameterized case. Concerning extensions of logic programs with aspects of concurrency, the use of contrained formulas to de ne non-ground semantics for linear logic programs has been investigated in [5].

*Relation with AC Rewriting. The use of constraints and the independence from* the initial con guration are two distinguished features of our approach compared to applications of AC rewriting and rewrite logic to simulation and analysis (see e.g. [21]). Backward analysis share some similarities with saturation-based pro- cedures for rst-order logic, with associative and commutative (AC) operators: computing P re here amounts to saturate the theory with the set of the logical consequences of a theory. More precisely, we rst express a multiset rewriting rule M ! M0 (where M = a j ::: j a , and M0 = b j ::: j b ) as the rewriting rule:

*1 m 1 n*

*M j X ! M0 j X;*

*where j is an AC symbol, and X is a free variable used to obtain an upward* closed set of ground rule instances. A veri cation problem expressible in terms of upward-closed sets can be represented as the `reachability' problem: the initial state, say start, reaches an instance of the term tunsafe having the form

*t1 j ::: j tn j Y;*

*where t1 j ::: j tn denotes the minimal violations, and Y is a free variable. As an* experiment, we have encoded our example, and tested on the saturation-based

*theorem prover daTac [22] specialized in AC theories. The prover returned a don't* know answer to our problem.

*Apart from the decidability results for the veri cation problems of Timed Petri* Nets given in [2,3], and some results for ground AC-rewriting systems [20], we are not aware of theoretical results concerning decidability of reachability problems for multiset rewriting systems with `colors'. Though the notion of contraint (to symbolically represent data) is not peculiar of systems like daTac, we plan to in- vestigate more deeply the applicability of automated deduction problems like the one presented in this paper.

*Need of Accelerations. The framework presented in the paper allows us to express* examples of mutual exclusion algorithms, with structured colors. As an example, consider the CPN model of the well-know ticket mutual exclusion algorithm. Dif- ferently from our previous example, the ticket algorithm deals with one common critical section. Before entering the critical section, each process gets a ticket and waits until its turn comes. Turns are selected by incrementing a variable everytime a process leaves the critical section. In this example, tickets will be the colors of our CPNs. The protocol is parametric on the number of processes and on the values of tickets. We use the predicates fthink; wait; useg to denote the state of processes, count to emit new tickets, and turn to establish the current turn. Note that while we need multiset rewriting to handle processes, we should consider only one copy of the two counters (we will discuss later this point). As constraint language we will use linear arithmetic constraints interpreted over the nonnegative integers. The speci cation is given as follows.

*think(x) j count(t) ! wait(x0 ) j count(t0 ) : x0 = t; t0 = t + 1:* wait(x) j turn(s) ! use(x) j turn(s) : x s:

*use(x) j turn(s) ! think(x0 ) j turn(s0 ) : x0 = x; s0 = s + 1:*

*With the rst rule, a process in state think represented as gets the current ticket* t (stored in couint(t)). A new ticket is prepared incrementing t. With the second rule a process requesting enters the critical section whenever its ticket t has a value less or equal than the current turn s (stored in turn(s)). With the third rule, a process releases the resource an the current turn is updated. In the initial state of the system all processes are thinking (the initial value of their ticket do not matter), while turn and count must store the same initial value. values of the identi ers of the resources. The set of violations can be represented through the constraint multiset use(x) j use(y) : x 0;y 0 denoting all con gurations with at least two processes in the critical section. Our procedure (enriched with the use of invariants like every reachable state has at most one occurrence of turn and count) does not terminate on this example. Furthermore, acceleration operations built on the top of widenings used in for static analysis [7] did not return useful approximations. Finding stronger acceleration operators seems to be an interesting direction for future research.

# *7 Conclusions*

*In this paper we made a rst attempt to merge notions coming from di erent* research elds (constraints, rewriting, Petri Nets) in order to attack one of the more diÆcult problem of (manual as well as automated) veri cation: proving parameterized properties. Our conceptual contribution is the de nition of an assertional language for expressing properties of Colored Petri Nets. The con- cept of constraint allows us to de ne the language on the basis of the operations (entailment, etc.) supported by any system for programming over constraints [23]. The language is designed for the backward approach with upward-closed sets proposed in [1]. This approach represents an alternative to previous valida- tion techniques used for CPNs based on the construction of the occurrence graph (forward reachability), and on structural analysis [16].

*Acknowledgments. The author would like to thank Sandro Etalle, Marco Boz-* zano and Gianna Reggio for useful discussions, and Alessandro Armando and Michael Rusinowitch for the experiments with daTac.

# *References*

*[1] Abdulla, P. A., K. Cerans, B. Jonsson and Y.-K. Tsay, General Decidability Theorems for In nite-State Systems, in: Proc. LICS'96 (1996), pp. 313{321.*

*[2] Abdulla, P. A. and B. Jonsson, Verifying networks of timed processes, in: Proc. TACAS '98, LNCS 1384 (1998), pp. 298{312.*

*[3] Abdulla, P. A. and A. Nyl en, Better is Better than Well: On EÆcient Veri cation of In nite-State Systems , in: Proc. LICS 2000 (2000), pp. 132{140.*

*[4] Bouajjani, A. and R. Mayr, Model checking lossy vector addition systems, in: Proc. STACS '99, LNCS 1563 (1999), pp. 323{333.*

*[5] Bozzano, M., G. Delzanno and M. Martelli, An e ective xpoint semantics for*

*rst-order linear logic programs, in: Proc. FLOPS 01', LNCS 2024 (2001), pp. 138{152.*

*[6] Clarke, E., O. Grumberg and S. Jha, Verifying Parameterized Networks, TOPLAS*

*19 (1997), pp. 726{750.*

*[7] Cousot, P. and R. Cousot, Abstract Interpretation: A Uni ed Lattice Model for Static Analysis of Programs by Construction or Approximation of Fix-Points, in: Proc. POPL'77 (1977), pp. 238{252.*

*[8] Delzanno, G., Automatic Veri cation of Parameterized Cache Coherence Protocols, in: Proc. CAV 2000, LNCS 1855 (2000), pp. 53{68.*

*[9] Delzanno, G. and A. Podelski, Model Checking in CLP, in: Proc. TACAS '99, LNCS 1579 (1999), pp. 223{239.*

*[10] Emerson, E. A. and K. S. Namjoshi, On the veri cation of broadcast protocols, in: Proc. LICS '98 (1998), pp. 70{80.*

*[11] Esparza, J., A. Finkel and R. Mayr, On Model Checking for Non-deterministic In nite-state Systems, in: Proc. LICS '99 (1999), pp. 352{359.*

*[12] Finkel, A. and P. Schnoebelen, Well-structured transition systems everywhere!, Theoretical Computer Science 256 (2001), pp. 63{92.*

*[13] Fribourg, L., Constraint Logic Programming Applied to Model Checking, in: Proc. LOPSTR '99, LNCS 1817 (1999), pp. 30{41.*

*[14] German, S. M. and A. P. Sistla, Reasoning about Systems with Many Processes, JACM 39 (1992), pp. 675{735.*

*[15] Ja ar, J. and M. J. Maher, Constraint Logic Programming: A Survey, Journal of Logic Programming 19-20 (1994), pp. 503{582.*

*[16] Jensen, K., An Introduction to the Theoretical Aspects of Coloured Petri Nets, in: A Decade of Concurrency, LNCS 803 (1994), pp. 230{272.*

*[17] Karp, R. M. and R. E. Miller, Parallel program schemata, Journal of Computer and System Sciences 3 (1969), pp. 147{195.*

*[18] Kesten, Y., O. Maler, M. Marcus, A. Pnueli and E. Shahar, Symbolic model checking with rich assertional languages, in: Proc. CAV '97, LNCS 1254 (1997), pp. 424{435.*

*[19] Meseguer, J., Conditioned Rewriting Logic as a Uni ed Model of Concurrency, Theoretical Computer Science 96 (1992), pp. 73{155.*

*[20] Narendran, P. and M. Rusinowitch, Any ground associative-commutative theory has a nite canonical system, Journal of Automated Reasoning 17 (1996), pp. 131{143.*

*[21] O lveczky, P. C. and J. Meseguer, Real-Time Maude: A Tool for Simulating and Analyzing Real-Time and Hybrid Systems, in: Proc. Workshop on Rewriting Logic and its Applications '00, ENTCS 36 (2000).*

*[22] Rusinowitch, M. and L. Vigneron, Automated Deduction with Associative and Commutative Operators, Applicable Algebra in Engineering, Communication and Computing 6 (1995), pp. 23{56.*

*[23] Saraswat, V. and P. Van Henteryck, \Principles and Practice of Constraint Programming," MIT Press, 1995.*