Downward Closed Language Generators

Parosh Aziz Abdulla   Johann Deneux   Pritha Mahata   Aletta Nylén

Dept. of Information Technology, Uppsala University
P.O. Box 337, SE-751 05 Uppsala, Sweden
{parosh,johannd,pritha,aletta}@it.uu.se

Abstract

We use downward closed languages for representing sets of states when performing forward reachability analysis on infinite-state systems. Downward closed languages are often more succinct than exact representations of the set of reachable states. We introduce a formalism for representing downward closed languages, called *downward closed language generators (dlgs)*. We show that standard set operations needed for performing symbolic reachability analysis are computable for dlgs. Using a class of hierarchically defined dlgs, we have implemented a prototype for analysing timed Petri nets and used it to analyze a parameterized version of Fischer's protocol. We also show how dlgs can be used for uniform representation of formalisms previously presented for models such as Petri nets and lossy channel systems.

1 Introduction

One of the most fundamental problems in model checking is that of *reachability*: is a set of *final (bad) states* reachable from a given set of *initial states*? The problem is usually solved by computing either *post-images* (forward analysis) or *pre-images* (backward analysis). While the two methods seem to be symmetric, they exhibit surprisingly different behaviours in many applications. For instance, for several classes of infinite-state systems, it turns out that backward analysis is feasible while forward analysis is not [AJ96, May00].

Nevertheless, forward analysis is still practically very attractive from the point of view of model checking [FRSB02, AAB$^+$99a, AAB99b]. Often, it turns out [ABLS01, AAB00, ABJ98] that it is possible to design semi-algorithms for forward analysis which terminate on almost all interesting examples. Furthermore, forward closure contains much more information about system behaviour than backward closure. This is due to the fact that forward closure characterizes the set of states which may arise during the execution of the system, while backward closure only describes the set from which the system can fail. More interestingly, forward analysis can often be used for constructing *symbolic graphs* which constitute finite abstractions of the systems, and on which standard techniques for finite state model checking can be applied [ABLS01, AAB$^+$99a, BLO98, LBBO01].

In the context of infinite-state systems, both methods can obviously only be implemented using a symbolic approach. The basic ingredient in any symbolic approach is to device a *symbolic representation* of the state space. The representation should be "adequate" in the sense that it allows computability of a number of basic operations present in implementations of all symbolic algorithms, such as union, intersection, checking entailment, post-images (or pre-images). Once algorithms have been designed for computing post-images, they can be used both to generate and verify invariants.

In [AČJYK00, AN00] we have developed a theory for designing symbolic representations, when performing backward reachability analysis, for a wide class of infinite-state systems. The symbolic representation relies on languages which are *upward closed* with respect to some given ordering on the state space.

The work of this paper complements that of [AČJYK00, AN00], and describes a formalism for performing forward analysis of infinite-state systems. We use *downward closed languages* as symbolic representation for sets of states. There are at least two reasons why downward closed languages are useful in forward reachability analysis. First, some systems, e.g. lossy channel systems [AJ96], have state spaces which are downward closed. More significantly, it is often the case that *bad states* are represented by upward closed sets (see e.g. [EFM99, DR00, AJ96, AČJYK00, AN00]). Therefore, if an invariant has an empty intersection with the set of bad states, then the downward closure of the invariant has also an empty intersection. This means that, in order to analyze safety properties formulated as upward closed sets, it is sufficient to consider downward closed invariants. The downward closure of an invariant has usually a much simpler structure and is therefore much easier to analyze.

While an upward closed language can simply be represented by the set of its minimal elements [DR00, AČJYK00, AN00], the situation is more complicated in the case of downward closed languages. In this paper, we introduce *downward closed language generators (dlgs)*, which characterize downward closed sets of words over a (potentially) infinite alphabet. We show that dlgs satisfy the general adequacy criteria mentioned above, i.e., they allow computability of union, intersection, and checking entailment, and describe how to compute post-images of dlgs for several computation models such as Petri nets, lossy channel systems, and timed Petri nets. A timed Petri net operates on a potentially unbounded number of real-valued clocks, and is therefore more powerful than classical timed automata.

Furthermore, we generalize our theory in two directions. We consider other data structures than words, and study languages which are sets of multisets, sets, vectors, etc. Such languages arise in the analysis of infinite-state systems (see e.g. [DEP99, EFM99, AČJYK00, AN01] for some examples). Also, in a similar manner to the case with upward closed languages [AČJYK00, AN00], we present a methodology for defining downward closed languages over complicated data structures in a hierarchical manner. We use the hierarchical definition to derive a novel formalism for representing downward closed sets of markings in timed Petri nets.

We have implemented a prototype based on our representation for timed Petri nets. We report the use of the prototype for proving an invariant for Fischer's protocol. The invariant is sufficiently strong for showing that the protocol satisfies mutual exclusion regardless of the number of processes inside the protocol.

Related Work This work is a generalization of that in [ABJ98, ABB01], where we present *simple regular expressions (SRE)* as representations for downward closed languages over a *finite* alphabet. The main difference compared to this paper, is that we allow infinite alphabets and also work with other data structures than words. This is crucial for several classes of infinite-state systems such as timed Petri nets which are beyond the capabilities of the theory of SREs.

Both [DR00] and [FRSB02] consider Petri nets and give symbolic representations for upward closed sets and downward closed sets of markings, respectively. [FIS00] gives a symbolic representation for FIFO automata. Since, these representations are designed for weaker models (Petri nets and FIFO automata), they are not sufficiently powerful for capturing the behaviour of timed Petri nets.

In [AN01, AN00, AJ98], we also consider timed Petri nets and represent upward closed sets of markings by *existentially quantified* predicates. Each predicate represents a *region* as defined in the classical theory of timed automata [AD90]. The dlgs we present in Section 7 are *region generators*. A region generator corresponds to the union of an infinite number of regions, and represents a downward closed set of markings. Both existential regions and region generators can model markings with unbounded numbers of clocks and are therefore strictly more powerful than regions.

Outline In the next section, we introduce dlgs and in the following sections we show how to compute intersection (Section 3) and how to check entailment between dlgs (Section 4). In Section 5 we show how to extend the theory to data structures such as multisets, sets and vectors. In Section 6 we consider hierarchical dlgs. In Section 7, we introduce region generators and show how they can be used for verification of timed Petri nets. In Section 8 we apply the theory for other classes of infinite-state systems such as Petri nets and lossy channels systems. We describe our implementation of region generators and report on some experimental results in Section 9. Finally, in Section 10, we give some conclusions and directions for future research.

2 Downward Closed Language Generators

In this section, we introduce *downward closed language generators (dlgs)*.

Assume a *quasi-ordering* $(A, \preceq)$, i.e., a reflexive and transitive binary relation $\preceq$ on a set A. For $B \subseteq A$, we use $A^{\not\succeq B}$ to denote the set $\{a \in A | \ \forall b \in B.\ b \not\preceq a\}$. For $a, b \in A$ we use $a \sqcup b$ to denote the set of least upper bounds of a and b, i.e., the set of all elements c such that $a, b \preceq c$, and there is no $c' \in A$ with $a, b \preceq c' \prec c$. For $B_1, B_2 \subseteq A$, we use $B_1 \sqcup B_2$ to denote the set $\bigcup_{a_1 \in B_1, a_2 \in B_2} (a_1 \sqcup a_2)$. For $B_1, B_2 \subseteq A$, we use $B_1 \sqsubseteq B_2$ to denote that for each $b_2 \in B_2$ there is a $b_1 \in B_1$ with $b_1 \preceq b_2$.

We assume that $\preceq$ is computable, i.e., for each $a, b \in A$, we can decide whether $a \preceq b$. We also assume that the set $a \sqcup b$ is finite and computable for each $a, b \in A$. The above implies that $B_1 \sqsubseteq B_2$ is computable and that $B_1 \sqcup B_2$ is finite and computable for finite sets B_1 and B_2.

A language L over A is a set of words over A, i.e., $L \subseteq A^*$. We use $\overline{L} = A^* - L$ to denote the *complement* of L. For a word $w \in A^*$ we use $|w|$ to denote the length of w, and $w(i)$ to denote the i^{th} element of w where $1 \leq i \leq |w|$. We use $w_1 \bullet w_2$ to denote the concatenation of the words w_1 and w_2.

We define the ordering $\preceq^*$ on A^* such that $w_1 \preceq^* w_2$ if there is monotonic injection $h : \{1, \ldots, |w_1|\} \rightarrow \{1, \ldots, |w_2|\}$ where $w_1(i) \preceq w_2(h(i))$ for $i : 1 \leq i \leq |w_1|$.

In the rest of this section we consider the case where L is downward closed with respect to $\preceq^*$, i.e., $w_1 \in L$ and $w_2 \preceq^* w_1$ imply $w_2 \in L$.

DLGs

We define *downward closed language generators (dlgs)* (over A) as follows. An *expression* e is either an *atomic expression* $\sim B$ or a *star expression* $\overset{*}{\sim} B$, where B is a finite subset of A. A *product* p is of the form $e_1 \bullet \cdots \bullet e_m$ where $e_1, \ldots, e_m$ are expressions. A dlg ϕ is a sum $p_1| \cdots |p_n$, where $p_1, \ldots, p_n$ are products. The dig ϕ defines a downward closed language $\mathcal{L}(\phi)$ in the following manner:

$$\mathcal{L}(\sim B) = A^{\not\succeq B} \cup \{\epsilon\} \tag{1}$$

$$\mathcal{L}(\overset{*}{\sim} B) = \{w_1 \bullet \cdots \bullet w_k |\; w_1, \ldots, w_k \in \mathcal{L}(\sim B)\} \tag{2}$$

$$\mathcal{L}(e_1 \bullet \cdots \bullet e_m) = \{w_1 \bullet \cdots \bullet w_m |\; w_1 \in \mathcal{L}(e_1), \ldots, w_m \in \mathcal{L}(e_m)\} \tag{3}$$

$$\mathcal{L}(p_1| \cdots |p_n) = \mathcal{L}(p_1) \cup \cdots \cup \mathcal{L}(p_n) \tag{4}$$

There are a number of special cases in the above definition. More precisely, if $B = \emptyset$ then $\mathcal{L}(\sim B) = \mathcal{L}(\sim \emptyset) = A \cup \{\epsilon\}$, and $\mathcal{L}(\overset{*}{\sim} B) = \mathcal{L}(\overset{*}{\sim} \emptyset) = A^*$. Furthermore, if $m = 0$ then $\mathcal{L}(e_1 \bullet \cdots \bullet e_m) = \{\epsilon\}$, and if $n = 0$ then $\mathcal{L}(p_1| \cdots |p_n) = \emptyset$.

For dlgs ϕ_1 and ϕ_2 we use $\phi_1 \bullet \phi_2$ to denote the dlg ϕ with $\mathcal{L}(\phi) = \{w_1 \bullet w_2 |\; w_1 \in \mathcal{L}(\phi_1) \text{ and } w_2 \in \mathcal{L}(\phi_2)\}$. Observe that ϕ always exists and is computable (by distributing $\bullet$ over $|$).

In the sequel, if the set A is known in the context, we only say a language (dlg) rather than a language (dlg) over A. Often, we identify dlgs with the languages they represent, so we write e.g. $\phi_1 \cap \phi_2$ instead of $\mathcal{L}(\phi_1) \cap \mathcal{L}(\phi_2)$. We observe that from the above definition it follows that it is trivial to compute the union of dlgs. For singleton sets, we often omit the set notation, so we write e.g., $\sim a$ instead of $\sim \{a\}$.

There are a number of simplification rules which we will always apply to dlgs as follows. For instance, we can assume that the set B in the definition of atomic and star expressions is canonical, i.e., there are no $a, b \in B$ with $a \neq b$ and $a \preceq b$. For each product p, there is another product $p' = e_1 \bullet \cdots \bullet e_m$ such that $\mathcal{L}(p') = \mathcal{L}(p)$ and for each $i : 1 \leq i < m$ it is the case that $\mathcal{L}(e_i \bullet e_{i+1}) \not\subseteq \mathcal{L}(e_i)$ and $\mathcal{L}(e_i \bullet e_{i+1}) \not\subseteq \mathcal{L}(e_{i+1})$. In fact, we can derive p' from p by deleting redundant expressions according to above, using the entailment algorithm described in Section 4. For each dlg ϕ, there is another dlg $\phi' = p_1| \cdots |p_n$ such that $\mathcal{L}(\phi') = \mathcal{L}(\phi)$ and there are no $i, j : 1 \leq i \neq j \leq n$ with $\mathcal{L}(p_i) \subseteq \mathcal{L}(p_j)$. Again, we can derive ϕ' from ϕ using the entailment algorithm.

Notice that when we write dlgs or languages we implicitly mean word dlgs or languages. In the later sections we will introduce dlgs and languages for multisets, sets, etc.

Example 1. *We consider dlgs over the finite set* $\{a,b,c,d\}$ *with the ordering being equality. Consider the dlgs* $\phi_1 = \sim a \bullet \overset{*}{\sim} \{b,c\}$ *and* $\phi_2 = \overset{*}{\sim} \{b,d\} \bullet (\sim c)$ *. Then e.g.* $bdddd \in \phi_1$ *and* $acaac \in \phi_2$ *while* $bb \notin \phi_1$ *and* $aabcc \notin \phi_2$*.*

The downward closed language consisting of all words which contain at most one a *and at most one* b *is characterized by the dlg*

$$\left(\overset{*}{\sim} \{a,b\} \bullet \sim a \bullet \overset{*}{\sim} \{a,b\} \bullet \sim b \bullet \overset{*}{\sim} \{a,b\}\right) \quad | \quad \left(\overset{*}{\sim} \{a,b\} \bullet \sim b \bullet \overset{*}{\sim} \{a,b\} \bullet \sim a \bullet \overset{*}{\sim} \{a,b\}\right)$$

Example 2. *We now consider dlgs over* $\mathbb{N}$ *with the ordering being* $\leq$*. Consider the dlgs* $\phi_1 = \overset{*}{\sim} 2 \bullet \sim 5$ *and* $\phi_2 = \overset{*}{\sim} 3 \bullet \sim 4$ *. Then e.g.* $101014 \in \phi_1$ *and* $20123 \in \phi_2$ *while* $010135 \notin \phi_1$ *and* $2034 \notin \phi_2$*.*

The downward closed language consisting of all words w*, such that the total number of elements greater than* 3 *is at most two and the total number of elements greater than* 5 *is at most one, is characterized by the dlg*

$$\left(\overset{*}{\sim} 3 \bullet \sim \emptyset \bullet \overset{*}{\sim} 3 \bullet \sim 5 \bullet \overset{*}{\sim} 3\right) \quad | \quad \left(\overset{*}{\sim} 3 \bullet \sim 5 \bullet \overset{*}{\sim} 3 \bullet \sim \emptyset \bullet \overset{*}{\sim} 3\right)$$

3 Intersection

In this section, we give an algorithm for computing the intersection $\phi = \phi_1 \cap \phi_2$ of two dlgs ϕ_1 and ϕ_2,

In case ϕ_1 and ϕ_2 are atomic expressions or star expressions, we can compute ϕ using the following equalities.

$$\begin{aligned} \sim B_1 \ \cap \sim B_2 &= \ \sim (B_1 \cup B_2) \\ \overset{*}{\sim} B_1 \ \cap \sim B_2 &= \ \sim (B_1 \cup B_2) \\ \overset{*}{\sim} B_1 \ \cap \overset{*}{\sim} B_2 &= \ \overset{*}{\sim} (B_1 \cup B_2) \end{aligned}$$

If ϕ_1 and ϕ_2 are products, then if either of them is empty, ϕ will also be empty. Otherwise, let $\phi_1 = e_1 \bullet p_1$ and $\phi_2 = e_2 \bullet p_2$. We have four cases depending on the form of e_1 and e_2.

1. e_1 and e_2 are atomic expressions,

   $\phi = (e_1 \cap e_2) \bullet (p_1 \cap p_2) \mid (\phi_1 \cap p_2) \mid (p_1 \cap \phi_2)$

2. e_1 is an atomic expression and e_2 is a star expression.

   $\phi = (e_1 \cap e_2) \bullet (p_1 \cap \phi_2) \mid (\phi_1 \cap p_2)$

3. e_1 is a star expression and e_2 is an atomic expression.

   $\phi = (e_1 \cap e_2) \bullet (\phi_1 \cap p_2) \mid (p_1 \cap \phi_2)$

4. e_1 and e_2 are star expressions.
   $\phi = (e_1 \cap e_2) \bullet (\phi_1 \cap p_2) \mid (e_1 \cap e_2) \bullet (p_1 \cap \phi_2)$

Finally, if $\phi_1 = p_1|\cdots|p_{n_1}$ and $\phi_2 = q_1|\cdots|q_{n_2}$ are sums, then $\phi = \phi_{11}|\cdots|\phi_{n_1 n_2}$, where $\phi_{ij} = p_i \cap q_j$ for $i : 1 \leq i \leq n_1$ and $j : 1 \leq j \leq n_2$.

In the appendix, we give a proof of correctness of the algorithm. From this we conclude that

Theorem 3.1. *Given dlgs ϕ_1 and ϕ_2, we can compute $\phi = \phi_1 \cap \phi_2$.*

A side effect of Theorem 3.1 is the following. We say that $\preceq$ is a *well quasi-ordering (wqo)* if for any infinite sequence $a_0, a_1, a_2, \ldots$ of elements in A there are i and j with $i < j$ and $a_i \preceq a_j$. Then, we have

Theorem 3.2. *For a wqo $\preceq$ and a language L which is downward closed with respect to $\preceq^*$, there is a dlg ϕ such that $L = \mathcal{L}(\phi)$.*

4 Entailment

In this section, we show how to check entailment between dlgs. For dlgs $\phi = p_1|\cdots|p_n$ and ϕ', it is clear that $\phi \subseteq \phi'$ iff $p_i \subseteq \phi'$ for each $i : 1 \leq i \leq n$. Therefore, in the rest of this section, we concentrate on the problem of checking entailments of the form $p \subseteq p_1|\cdots|p_n$.

First, we consider the special case, where all the products are expressions. We have

$$e \;\subseteq\; \left(\overset{*}{\sim} B_1\right) \mid \cdots \mid \left(\overset{*}{\sim} B_m\right) \mid (\sim B_{m+1}) \mid \cdots \mid (\sim B_n)$$

iff either

- e is an atomic expression $\sim B$, and $B \sqsubseteq (B_1 \sqcup \cdots \sqcup B_n)$; or
- e is a star expression $\overset{*}{\sim} B$, and there is an $i : 1 \leq 1 \leq m$ with $B \sqsubseteq B_i$.

From above and the assumption on computability of $\sqsubseteq$ and $\sqcup$ (Section 2), we get the following lemma.

Lemma 4.1. *For expressions $e, e_1, \ldots, e_n$, we can check whether $e \subseteq e_1 \mid \cdots \mid e_n$.*

Next, we consider the general case. For a product $p = e_1 \bullet \cdots \bullet e_n$, we define $\#p$ to be $\cup_{1 \leq i \leq n} \mathcal{L}(e_i)$.

Consider a dlg $\phi = p_1 \mid \cdots \mid p_n$. A *factorization* of ϕ with respect to e is a set $\{q_1, \ldots, q_m\}$ of (possibly empty) products, such that there is a partitioning Q_1, Q_2 of the set $\{p_1, \ldots, p_n\}$, where Q_1 is not empty. Furthermore, the set Q_1 is of the form $\{p'_1 \bullet e_1 \bullet p''_1, \ldots, p'_m \bullet e_m \bullet p''_m\}$, and one of the following conditions is satisfied

- e is an atomic expression and

$$\#e_1 \cap \cdots \cap \#e_m \cap \#e \quad \not\subseteq \quad \#p'_1 \cup \cdots \cup \#p'_m \cup \#Q_2$$

  Furthermore, either e_i is an atomic expression and $q_i = p''_i$, or e_i is a star expression and $q_i = e_i \bullet p''_i$, for each $i : 1 \leq i \leq m$.

- e is a star expression, $e \subseteq e_i$ for each $i : 1 \leq i \leq m$, and there is no expression e' in any of the products in the set $\{p'_1, \ldots, p'_m\} \cup Q_2$, with $\mathcal{L}(e) \subseteq \mathcal{L}(e')$. Furthermore, we have $q_i = e_i \bullet p''_i$ for each $i : 1 \leq i \leq m$.

Now we are ready to present the entailment algorithm. For products $p, p_1, \ldots, p_n$, where $p = e \bullet p'$, we have $p \subseteq p_1 \mid \cdots \mid p_n$ iff either p is empty or the following two conditions are satisfied:

1. $\#e \subseteq \#p_1 \cup \cdots \cup \#p_n$
2. For each factorization $\{q_1, \ldots, q_m\}$ of $\{p_1, \ldots, p_n\}$ with respect to e, it is the case that $p' \subseteq q_1 \mid \cdots \mid q_m$.

In fact the above characterization gives us an algorithm for checking entailment. This follows from Lemma 4.1, computability of intersection (Section 3),and the fact that there are finitely many possible factorizations (limited by the number of products and the number of expressions in each product).

In the appendix, we give a proof of correctness of the algorithm. From this we conclude that

Theorem 4.2. *For dlgs ϕ_1 and ϕ_2, we can check whether $\phi_1 \subseteq \phi_2$.*

5 Multisets, Sets, Vectors, etc

In this section, we show how to extend the theory presented in Section 2-4 to data structures other than words such as multisets, sets and vectors.

5.1 Multisets

Assume a quasi-ordering $(A, \preceq)$. A *multiset* or bag b over A of *size* n, is a mapping from $\{1, \ldots, n\}$ to A. For $a \in A$, we write b^a to denote the size of the set $\{i \mid b(i) = a\}$. In other words, b^a is the number of occurrences of a in b. We use $a \in b$ to denote that $b^a \geq 1$. Sometimes we write b as a list, where each element of the list is of the form a^n with $a \in A$ and $n \in \mathbb{N}$. For instance, (c^3, d^2) represents a multiset b where $b^c = 3$, $b^d = 2$, and $b^a = 0$ if $a \neq c, d$.

We let A^b denote the set of multisets over A. We define the ordering $\preceq^b$ on A^b as follows. For multisets b_1 with size n_1 and b_2 with size n_2, we have $b_1 \preceq^b b_2$ if there is a (not necessarily monotonic) injection h from $\{1, \ldots, n_1\}$ to $\{1, \ldots, n_2\}$ with $b_1(i) \preceq b_2(h(i))$ for every $i : 1 \leq i \leq n_1$.

Notice that $b_1 \preceq^b b_2$ means that for each element in b_1 there is a corresponding larger element (with respect to $\preceq$) in b_2.

For multisets b_1 and b_2, we define the *addition* $b_1 + b_2$ of b_1 and b_2 to be the multiset b where $b^a = {b_1}^a + {b_2}^a$. We also define the subtraction $b_1 - b_2$ to be the multiset b where $b^a = {b_1}^a - {b_2}^a$ if ${b_1}^a \geq {b_2}^a$ for all $a \in A$. Otherwise $b_1 - b_2$ is undefined.

A *multiset language* (over A) is a subset of A^b. The notions of downward and upward closedness (with respect to $\preceq^b$), complement, etc, are defined in a similar manner to words.

Now, we show how to use dlgs for characterizing downward closed multiset languages. We do that by introducing *multiset dlgs*, which are modifications of dlgs. The idea is to change the interpretation of the product operator, taking into consideration that it is commutative (in addition to being associative) in the case of multisets. More precisely, we take the new product operator, which we denote by $\odot$, to be multiset addition (rather than concatenation in the case of words). We also use $\circledast$ for multiset star expressions. This means that we change (2) and (3) in the definition of dlgs (Section 2) obtaining

$$\begin{aligned}
\mathcal{L}(\sim B) &= A^{\not\succeq B} \cup \{\epsilon\} && (5)\\
\mathcal{L}(\overset{\circledast}{\sim} B) &= \{b_1 + b_2 + \cdots + b_k |\ b_1, \ldots, b_k \in \mathcal{L}(\sim B)\} && (6)\\
\mathcal{L}(e_1 \odot \cdots \odot e_m) &= \{b_1 + \cdots + b_m |\ b_1 \in \mathcal{L}(e_1), \ldots, b_m \in \mathcal{L}(e_m)\} && (7)\\
\mathcal{L}(p_1 | \cdots | p_n) &= \mathcal{L}(p_1) \cup \cdots \cup \mathcal{L}(p_n) && (8)
\end{aligned}$$

Each multiset product p can be simplified into a product p' of the form

$$\left(\overset{\circledast}{\sim} B\right) \ \odot \ (\sim B_1) \ \odot \ \cdots \ \odot \ (\sim B_m)$$

such that $\mathcal{L}(p) = \mathcal{L}(p')$. This follows from the fact that all star expressions in p can be merged into one, using the relation

$$\left(\overset{\circledast}{\sim} B\right) \ \odot \ \left(\overset{\circledast}{\sim} B'\right) \ = \overset{\circledast}{\sim} (B \sqcup B')$$

In the sequel we will assume that all multiset dlgs are of this form.

Intersection

Computing intersection for multiset expressions and sums is done in the same manner as in the word case. Regarding the fact that multiset addition is commutative, we can derive the following algorithm for intersection of multiset dlg products. Suppose that we are given two products $p_1 = e_{11} \odot \cdots \odot e_{1k}$ and $p_2 = e_{21} \odot \cdots \odot e_{2m}$. Define p_{1i} to be the result of deleting the expression e_i from p_1. Define p_{2j} in a similar manner. Then, $p_1 \cap p_2$ is the union of all multiset dlgs ϕ_{ij}, for $i : 1 \leq i \leq k$ and $j : 1 \leq j \leq m$, computed according to one of the following four cases.

1. if e_{1i} and e_{2j} are atomic expressions.

   $\phi_{ij} \ = \ (e_{1i} \cap e_{2j}) \ \odot \ (p_{1i} \cap p_{2j})$.

2. if e_{1i} is an atomic expression and e_{2j} is a star expression.
   $\phi_{ij} \;=\; (e_{1i} \cap e_{2j}) \;\odot\; (p_{1i} \cap p_2)$.
3. if e_{1i} is a star expression and e_{2j} is an atomic expression.
   $\phi_{ij} \;=\; (e_{1i} \cap e_{2j}) \;\odot\; (p_1 \cap p_{2j})$.
4. if e_{1i} and e_{2j} are star expressions.
   $\phi_{ij} \;=\; (e_{1i} \cap e_{2j}) \;\odot\; (p_{1i} \cap p_2) \quad | \quad (e_{1i} \cap e_{2j}) \;\odot\; (p_1 \cap p_{2j})$

We give proof of correctness of the algorithm in the appendix. In a corresponding manner to words, we can show that each downward closed multiset language L over a well quasi-ordered set can be characterized by a multiset dlg.

Entailment

We derive the multiset entailment algorithm from the algorithm for words (Section 4), using the fact that multiset addition is commutative. Entailment for sums where all products are expressions is the same as for words. We consider the general case.

Consider a dlg $\phi = p_1 \mid \cdots \mid p_n$. A *factorization* of ϕ with respect to e is a set $\{q_1, \ldots, q_m\}$ of (possibly empty) products, such that there is a partitioning Q_1, Q_2 of the set $\{p_1, \ldots, p_n\}$, where Q_1 is not empty. Furthermore, the set Q_1 is of the form $\{p'_1 \odot e_1 \odot p''_1, \ldots, p'_m \odot e_m \odot p''_m\}$, and one of the following conditions is satisfied

- e is an atomic expression and
  $$\#e_1 \;\cap\; \cdots \;\cap\; \#e_m \;\cap\; \#e \quad \not\subseteq \quad \#Q_2$$
  Furthermore, either e_i is an atomic expression and $q_i = p'_i \odot p''_i$, or e_i is a star expression and $q_i = p'_i \odot e_i \odot p''_i$, for each $i : 1 \leq i \leq m$.
- e is a star expression, $e \subseteq e_i$ for each $i : 1 \leq i \leq m$, and there is no expression e' in any of the products in the set Q_2, with $\mathcal{L}(e) \subseteq \mathcal{L}(e')$. Furthermore, we have $q_i = p'_i \odot e_i \odot p''_i$ for each $i : 1 \leq i \leq m$.

Since the products are commutative, we can drop the requirement on prefixes, which was used in the algorithm for words, and we use the whole product (not only the postfix) in the next step of the analysis. When the requirement on prefixes is dropped, a subset of $\mathcal{L}(e)$ may cause more than one factorization. Given a set $F = \{f_1, \ldots, f_k\}$ of factorizations, we let the union $f_1 \cup \cdots \cup f_k$ be the set $\{q_1, \ldots, q_M\}$. The number of such sets is finite due to the fact that it is limited by the number of possible factorizations, which is in turn limited by the number and sizes of the products.

For a factorization f, we will use $cause(f)$ to denote the subset of $\mathcal{L}(e)$ which causes the factorization, i.e., the set $\#e_1 \;\cap\; \cdots \;\cap\; \#e_m \;\cap\; \#e$. We say that a set $F = \{f_1, \ldots, f_k\}$ of factorizations is caused by the set $cause(f_1) \cap \cdots \cap cause(f_k)$ denoted by $cause(F)$.

Now we are ready to present the entailment algorithm. For products $p, p_1, \ldots, p_n$, where $p = e \odot p'$, we have $p \subseteq p_1 \mid \cdots \mid p_n$ iff either p is empty or the following two conditions are satisfied:

1. $\#e \subseteq \#p_1 \cup \cdots \cup \#p_n$
2. For each set F of factorizations such that $cause(F)$ is nonempty and and there is no factorization f' with $cause(f') \subseteq cause(F)$, it is the case that $p' \subseteq q_1 \mid \cdots \mid q_M$.

In fact the above characterization gives us an algorithm for checking entailment. This follows from Lemma 4.1, computability of intersection (Section 3), and the fact that there are finitely many possible factorizations (limited by the number of products and the number of expressions in each product).

In the appendix, we give a proof of correctness of the algorithm. From this we conclude that

Theorem 5.1. *For multiset dlgs ϕ_1 and ϕ_2, we can check whether $\phi_1 \subseteq \phi_2$.*

5.2 Sets

A set is s special case of a multiset where each element occurs at most once in the set. We let A^S denote the set of sets over A. We define the ordering $\preceq^S$ in the same manner as we did for $\preceq^b$. We define set dlgs in a similar manner to multisets by taking the product operator to be set union. Notice that this operator is now also idempotent. We can compute the intersection and entailment of two set dlgs using similar algorithm to those for multisets.

5.3 Vectors

For a natural number k, we let A^k denote the set of vectors of size k over A. As usual, we write members of A^k as tuple $(a_1, \ldots, a_k)$, where $a_1, \ldots, a_k \in A$. We define the ordering $\preceq^k$ such that $(a_1, \ldots, a_k) \preceq^k (a'_1, \ldots, a'_k)$ iff $a_i \preceq a'_i$ for $i : 1 \leq i \leq k$. We define *vector dlgs* in a similar fashion to above. In fact, vector dlgs, can be viewed as special cases of word dlgs where each product is of the form $e_1 \odot \cdots \odot e_k$, where e_i is an atomic expression. This enables us to allow vector dlgs where the atomic expressions are dlgs of different types. An example is (e_1, e_2) where e_1 is a set dlg over a set A and e_2 is a word dlg over B.

6 Hierarchical DLGs

In this section, we present a methodology which enables us to derive dlgs in a hierarchical manner. Such dlgs are needed e.g. for analysis of TPNs (Section 7).

At the lowest level of the hierarchy, we consider dlgs of the form defined in Section 2 (possibly interpreted over multisets, sets, or vectors, rather than over words). Higher levels of the hierarchy are built by using dlgs at a certain level as atomic expressions at the next level.

Formally we define *hierarchical dlgs* as follows

- A *first-order* dlg over an ordered set A is of the same form as defined in Section 2. A dlg may be interpreted as a set of words, multisets, sets, vectors, etc.
- We build $k+1$-order dlgs, by taking the set of atomic expressions to be the set of k-order dlgs. We will allow mixing different kinds of dlgs (word, multiset, set, and vector dlgs) at the different levels. For instance a $k+1$-order multiset dlg over a set A^* is built by using atomic expressions which are (word) dlgs over the set A

If dlgs at a certain level allow computability of intersection and entailment, then the same operations can be computed also at the next level. This is due to the fact that the algorithms presented in Section 2, Section 3, and Section 4 depend only on the fact that the corresponding operations are computable for atomic expressions. This gives the following theorem

Theorem 6.1. *Intersection and entailment are computable for hierarchical dlgs.*

In [ABJ98] we introduce *simple regular expressions* which are first-order dlgs (Section 8). In Section 7 we introduce a class of third-order dlgs and use them for verification of timed Petri nets.

Example 3. *We show how to hierarchically build dlgs which generate words over multisets over a finite alphabet.*

We start with a finite set $A = \{a, b, c, d\}$ and take the ordering to be equality. Examples of first-order multiset dlgs over A are $\phi_1 = \left(\overset{*}{\sim} a\right) \odot (\sim \{b, c\})$ and $\phi_2 = \left(\overset{*}{\sim} \{a, b, c\}\right) \odot (\sim d)$. Multisets in the corresponding languages are $\left(b^2, d^3\right) \in \phi_1$, $\left(b^4, a\right) \in \phi_1$, $\left(d^4, b\right) \in \phi_2$ and $\left(d^6, c\right) \in \phi_2$.

An example of a (word) dlg over A^b is $\psi = \phi_1^* \bullet \phi_2$. The elements of $\mathcal{L}(\psi)$ are words over multisets over A, e.g. $\left(b^2, d^3\right) \bullet \left(b^4, a\right) \bullet \left(d^4, b\right) \in \psi$.

7 Timed Petri Nets and Region Generators

In this section we introduce *region generators*, third order dlgs which we use for verification of timed Petri nets.

A *timed Petri net (TPN)* [AN01] is an extension of a standard Petri net where each token is equipped with a real-valued clock representing the age of the token. Each arc of the net is labelled with an interval defined by natural numbers restricting the age of the tokens travelling the arc. Given a TPN, we use P to denote the set of places and *max* to denote the maximum integer occurring in the intervals on the arcs of the net. A *marking* assigns to each place in the net, a multiset of real numbers representing the ages of the tokens in that place. The transition relation for a TPN consists of two types of transitions, *discrete transitions* corresponding to firing transitions, and *timed transitions* corresponding to letting time pass. Since TPNs operate on unbounded numbers of tokens, they are a more powerful computation model than timed automata.

Regions are predicates commonly used in verification of timed automata [AD90]. We propose a formalism, called *region generators*, for defining downward closed sets of markings in TPNs. Regions cannot be used for characterizing such sets since they can only deal with a finite number of clocks. Intuitively, a region generator g corresponds to an infinite set of regions, and thus characterizes an infinite set of markings, namely all markings belonging to regions generated by g.

We recall some basic concepts of regions. A region [AD90] defines the integral parts of clock values up to max (the exact age of a token is irrelevant if it is greater than max), and also the ordering of the fractional parts among clock values. For verification of TPNs, we need to use a variant which also defines the place in which each token (clock) resides. Following Godskesen [God94] we represent a region in the following manner. Each token in a marking satisfying a region is represented by a pair (p, n) (for simplicity of notation, we write such a pair as $p(n)$), where p is the place in which the token resides and n is the integral part of the age of the token. The region is defined by a triple (b_0, w, b_{max}) where

- b_0 is a multiset of pairs (of the form $p(n)$) representing tokens whose ages have fractional parts equal to 0.
- w is a word over multisets of pairs of the above form where pairs in the same multiset represent tokens whose ages have equal fractional parts. The order of the multisets in w corresponds to the order of the fractional parts.
- b_{max} is a multiset over P representing tokens with ages strictly greater than max. Since the actual ages of these tokens are irrelevant, the integer representing the integral part of their ages is omitted.

We define an ordering on markings such that $M_1 \preceq M_2$ if there is a marking M_3, such that we can derive M_3 from M_2 be deleting tokens and such that M_1 and M_3 belong to the same region.

A *region generator* is a third-order dlg constructed as follows.

- We start from the set of pairs of the form $p(n)$. Notice that the set of such pairs is finite. At the first level, we build multiset dlgs over the set of pairs where the ordering is taken to be equality. Each such a multiset is used to characterize a number of tokens whose ages have equal fractional parts. Using such dlgs we can express conditions of the form "there may exist an unbounded number of tokens in place p_1 and p_2 with equal fractional parts".
- At the second level we build word dlgs over the dlgs in the first level. These words are used to capture the ordering of fractional parts in the same manner as in regions. We use these dlgs to express conditions of the form "there is an unbounded number of tokens with increasing fractional parts in places p_1 and p_2".
- Finally, at the third level we build vector dlgs of the form $(\psi_0, \phi, \psi_{max})$ where
  - ψ_0 is a multiset dlg over $P \times \{0, \ldots, max\}$ and corresponds to b_0 in a region.
  - ϕ is a second-order dlg of the form described above.

– ψ_{max} is a multiset dlg over P and corresponds to b_{max} where the ages of the tokens have been omitted.

Notice that a region generator characterizes a (potentially) infinite set $\mathcal{L}(g)$ of regions. Therefore, a region generator g characterizes also an infinite set $[\![g]\!]$ of markings, namely all markings belonging to regions in the language $\mathcal{L}(g)$. It is straightforward to show that $[\![g]\!]$ is downward closed with respect to $\preceq$.

7.1 Computing post-images

We define a function *Post* such that for a region generator g, $Post(g)$ is a set of region generators. $Post(g)$ characterizes the downward closure of the set of markings which can be reached from a marking in $[\![g]\!]$ by a single transition. We define $Post(g)$ to be the union of $Post_\delta(g)$ which corresponds to performing timed transitions, and $Post_D(g)$ which corresponds to performing discrete transitions. Notice that, from each marking, an infinite number of timed transitions are possible (namely a timed transition corresponding to time progress by any nonnegative real number). Our algorithm computes the post-image of an input dlg with respect to all such transitions, i.e., it gives a dlg characterizing all markings which are reachable through progress of time from the original dlg.

We use the notation introduced in Section 5 for multisets. Also, we use ϵ to denote the word (multiset) dlg ψ such that $\mathcal{L}(\psi)$ only contains the empty word (multiset). We recall that, in a region generator, each first-order dlg is a multiset dlg defined over a finite set A, namely the set $A = P \times \{0, \ldots, \max\}$, with the ordering being equality. Since we are dealing with a finite alphabet, we know that an atomic expression will have the form $\sim (A \setminus S)$ and that a star expression will have the form $\overset{\circledast}{\sim} (A \setminus S)$. In this section, we denote such expressions simply by S and $S^\circledast$, respectively. Furthermore, without loss of generality, we assume that all multiset products are of the form $S^\circledast \odot S_1 \odot \cdots \odot S_k$ such that $S \cap S_i = \epsilon$ and such that S_i is a singleton for all $i : 1 \leq i \leq k$. It is straightforward to show that, for each multiset dlg, we can construct an equivalent multiset dlg of the above form.

$\boldsymbol{Post_\delta}$

$Post_\delta(g)$ is computed through an iterative procedure which simulates successively increasing the age of the tokens in the markings in $[\![g]\!]$. Each time the aging makes the markings enter new regions, a set of region generators characterizing these new regions is added to $Post_\delta$. After a finite number of steps, the procedure will terminate. More precisely, the procedure terminates when we reach a point where the newly added region generators characterize markings where token ages are all strictly greater than max.

For an input region generator g_{inp}, we compute $Post_\delta(g_{inp})$ which characterizes the set of all markings reachable from g_{inp} through progress of time. We compute $Post_\delta(g_{inp})$ as a set of of region generators as follows. We maintain two sets O and W of region generators. Intuitively, the set O contains those region generators which will be part of the set $Post_\delta(g_{inp})$, while W is a waiting list of region generators which are yet to be analyzed. We start with $W = \{g_{inp}\}$ and $O = \emptyset$. We perform a number of iterations. During each iteration we pick a region generator $g \in W$, move it to O, and add a number of new region generators to W and O as follows:

1. If g is of the form $(\psi_0, \phi, \psi_{max})$, where $\psi_0 \neq \epsilon$. In other words, the set of tokens with zero fractional parts is not empty. Let us consider the tokens in ψ_0. The next thing which occurs when time advances is that the fractional parts of these tokens become positive, but smaller than the fractional parts of all other tokens. Furthermore, those tokens with integral parts equal to max, will now have ages strictly greater than max. Each product p in ψ_0 will add a new region generator $(\epsilon, young \bullet \phi, \psi_{max} \odot old)$ to W. Intuitively $young$ and old are dlgs which correspond to "splitting" the set of tokens corresponding to ψ_0 into two sets: those with integral parts strictly less than max ($young$), and those with integral parts equal to max (old). Since the fractional parts of the tokens in $young$ are smaller than all other tokens, we put $young$ first in the second component of the region generator. Also, the ages of the tokens in old are now strictly greater than max, so they are added to the third component of the region generator.

   Formally, let p be of the form $S^{\circledast} \odot S_1 \odot \cdots \odot S_k$. We define $young$ to be of the form $T^{\circledast} \odot T_1 \odot \cdots \odot T_k$, and define old to be of the form $R^{\circledast} \odot R_1 \odot \cdots \odot R_k$, where

   - $p(n) \in T$ if and only if $p(n) \in S$ and $n < max$.
   - $T_i = S_i$ if S_i is of the form $\{p(n)\}$ and $n < max$. Otherwise, $T_i = \epsilon$.
   - $p \in R$ if and only if $p(max) \in S$.
   - $R_i = \{p\}$ if S_i is of the form $\{p(max)\}$. Otherwise, $R_i = \epsilon$.

2. If g is of the form $(\epsilon, w \bullet \psi, \psi_{max})$. In other words, there are no tokens with zero fractional parts. Furthermore, the second component of the region generator is of the form $w \bullet \psi$. That is, the last expression ψ is an atomic expression (which is a multiset dlg). The tokens in ψ are those with highest fractional parts (among those tokens whose ages are not strictly greater than max). When time increases the tokens corresponding to ψ will change age, such that their integral parts will be increased by one and their fractional parts will become equal to zero. This means that we add the region generator (ψ', w, ψ_{max}) to W, where ψ' is derived from ψ by replacing each pair $p(n)$ in ψ by the pair $p(n+1)$.

3. If g is of the form $(\epsilon, w \bullet \psi^*, \psi_{max})$. In other words, there are no tokens with zero fractional parts. Furthermore, the second component of the region generator is of the form $w \bullet \psi^*$. That is, the last expression ψ^* is a star expression. This case is similar to the previous one. However, the tokens corresponding to ψ^* now form an unbounded sequence with strictly increasing fractional parts. When time is increased, the elements of the sequence will successively reach a point where their integral part is increased by one and their fractional parts become equal to zero (in a similar manner to case 2), and then they are split (in a similar manner to case 1). We add a region generator $\left(\psi', young^* \bullet w \bullet \psi^*, \psi_{max} \odot old^{\circledast}\right)$ to O, and a region generator $\left(\psi', young^* \bullet w, \psi_{max} \odot old^{\circledast}\right)$ to W, where

   - ψ' is derived from ψ in a similar manner to case 2. Let ψ' be of the form $c_1 | \cdots | c_m$.

- *young* is of the form $young_1|\cdots|young_m$, where $young_i$ is derived from c_i in a similar manner to case 1.
- $p \in old$ if and only if $p(max)$ occurs in some of the products $c_1, \ldots, c_m$ of ψ'.

Notice that we add one of the newly generated region generators directly to O (and its "successor" to W). This is done in order to guarantee reaching the termination condition.

4. If g is of the form $(\epsilon, \epsilon, \psi_{max})$, i.e., all tokens have ages which are strictly greater than max, then we do not add any elements to W.

The algorithm terminates when the set W becomes empty. In such a case, the set O will contain $Post_\delta(g_{inp})$. Termination is guaranteed since we will eventually reach a point where all added region generators will only characterize tokens with ages greater than max (i.e. will be of the form of case 4).

$\boldsymbol{Post_t}$

We define $Post_t(g)$ to be the post-image of g with respect to a transition t. $Post_D(g)$ is computed as the union of $Post_t(g)$ for all transitions t in the TPN. We compute $Post_t(g)$ by defining two operations on region generators, namely *addition* and *subtraction* (of tokens). The main challenge in defining these operations is posed by the hierarchical nature of region generators. This implies that an operation on one level of the hierarchy is defined in terms of the operation on the level below.

We define an *addition* and a *subtraction* operation for region generators. An addition (subtraction) corresponds to adding (removing) a token in a certain age interval. These operations have hierarchical definitions reflecting the hierarchical structure of region generators. An operation on one level is defined in terms of the operation on the level below.

We start by defining addition and subtraction for multiset dlgs, defined over a finite set $P \times \{0, \ldots, \mathrm{max}\}$, with the ordering being equality.

Given a multiset product $c = S^{\circledast} \odot S_1 \odot \cdots \odot S_k$ and a pair $p(n)$ where p is a place and n is natural number (value of the integral part of the age of a token in p), we define the *addition* $c \oplus p(n)$ to be the multiset product $c \odot p(n)$.

The subtraction $c \ominus p(n)$ is defined by the following three cases.

- If S contains the token we want to remove, i.e., $p(n) \in S$, then $c \ominus p(n) = c$. Intuitively, the region generator in which c appears describes markings with an unbounded number of tokens each with an integral part equal to n, and each residing in place p. Therefore, after removing one such a token, we will still be left with an unbounded number of them.
- If $p(n) \notin S$ and S_i is of the form $\{p(n)\}$ for some $i : 1 \leq i \leq k$ then $c \ominus p(n) = S^{\circledast} \odot S_1 \odot \cdots \odot S_{i-1} \odot S_{i+1} \odot \cdots \odot S_k$.
- Otherwise, the operation is undefined.

For a multiset dlg ψ we define $\psi \ominus p(n)$ to be the set of all products c' such that $c' = c \ominus p(n)$ for some product c in ψ.

Now, we extend the operations to second-order word dlgs defined over multiset dlgs of the above form.

The addition $w \oplus p(n)$ is a set of word products corresponding to the union of the following three sets.

- For each w_1, w_2, and ψ with $w = w_1 \bullet \psi \bullet w_2$, we have
  $w_1 \bullet (\psi \oplus p(n)) \bullet w_2 \ \in \ (w \oplus p(n))$.
- For each w_1, w_2, and ψ with $w = w_1 \bullet \psi^* \bullet w_2$, we have
  $w_1 \bullet \psi^* \bullet (\psi \oplus p(n)) \bullet \psi^* \bullet w_2 \ \in \ (w \oplus p(n))$.
- For each w_1 and w_2 with $w = w_1 \bullet w_2$, we have
  $w_1 \bullet p(n) \bullet w_2 \ \in \ (w \oplus p(n))$.

Intuitively, elements added according to the first two cases correspond to adding a token with a fractional part equal to that of some other token. In the third case the fractional part differs from all other tokens.

We define the subtraction $w \ominus p(n)$, where w is a word product, to be a set of word products, according to the following two cases.

- If there is a star expression containing the token we want to remove, i.e., if w is of the form $w_1 \bullet \psi^* \bullet w_2$, and $\psi \ominus p(n)$ is defined, then $w \ominus p(n) = \{w\}$.
- Otherwise, the set $w \ominus p(n)$ contains all word dlgs of the form $w_1 \bullet \psi' \bullet w_2$ such that w is of the form $w_1 \bullet \psi \bullet w_2$ and $\psi' \in (\psi \ominus p(n))$. Notice that this set may be empty.

For a word sum, the result of addition and subtraction is the union of the sets obtained by performing the same operations on the products of the sum (in a similar manner to multisets above).

Now we describe how to use the addition and subtraction operations for computing $Post_t$. Addition and subtraction of pairs of the form $p(n)$ can easily be extended to pairs of the form $p(N)$, $N \subseteq \{0, \ldots, max\}$, where $g \ominus p(N)$ characterizes all all markings which belong to $g \ominus p(n)$ for some $n \in N$. $g \oplus p(N)$ is defined in a similar manner.

We recall that, in a TPN, the effect of firing a transition is to remove tokens from the input places and add tokens to the output places. Furthermore, the tokens which are added or removed should have ages in the corresponding intervals. The effect of of firing transitions from the set of markings characterized by a region generator $g = (\psi_0, \phi, \psi_{max})$ can therefore be defined by the following operation. We assume an (open or closed) interval $\mathcal{I}$ of the form $[a : b]$. The subtraction $g \ominus (p, \mathcal{I})$ consists of the following region generators.

- $(\psi_0 \ominus (p, N), \phi, \psi_{max})$ where N is the subset of $\{0, \ldots, max\}$ which is in the interval $\mathcal{I}$. Intuitively, if the age of the token that is removed has a zero fractional part, then N contains the valid choices of integral part.
- $(\psi_0, \phi \ominus (p, N), \psi_{max})$ where N is the subset of $\{0, \ldots, max\}$ where each element n satisfies $a \le n < b$, i.e., each n is a valid choice of integral part for the age of the token if it has a non-zero fractional part.
- $(\psi_0, \phi, \psi_{max} \ominus p)$ if $\mathcal{I}$ is of the form $[a : \infty]$, i.e., the age of the token may be greater than max.

Addition is defined in a similar manner.

8 Other Models

In this section we will show how dlgs can be used to explain existing representations used in forward reachability analysis of classes of infinite-state systems other than TPNs.

Lossy Channel Systems

A *lossy channel system (LCS)* [AJ96] consists of a finite-state process which operates on a set of unbounded FIFO buffers which are unreliable in the sense that they can nondeterministically lose messages. A state of an LCS defines the local state of the process, and the contents of the channels which are described as words over a finite alphabet. In [ABJ98, ABB01]. we consider *Simple Regular Expressions (SREs)* and use them to characterize sets of reachable states in LCS. We also present algorithms for computing post-images of SREs. SREs are first-order word dlgs, where the underlying alphabet is finite and where the ordering on the alphabet is equality.

Petri Nets

A *Petri net* is a special case of a timed Petri net where the ages of the tokens are irrelevant. Sets of markings of a Petri net can be represented by multiset dlgs over a finite set (of places). This implies that the algorithms of Section 7 can also be applied to Petri nets. Dedicated and efficient symbolic representations for forward analysis of Petri nets are presented in [FRSB02].

9 Experimental Results

We have implemented a prototype which can be used to prove invariants of timed Petri nets. The tool is based on region generators and contains implementations of intersection, entailment checking, membership checking and *Post*. We have used our prototype to verify a parameterized version of Fischer's protocol [SBM92].

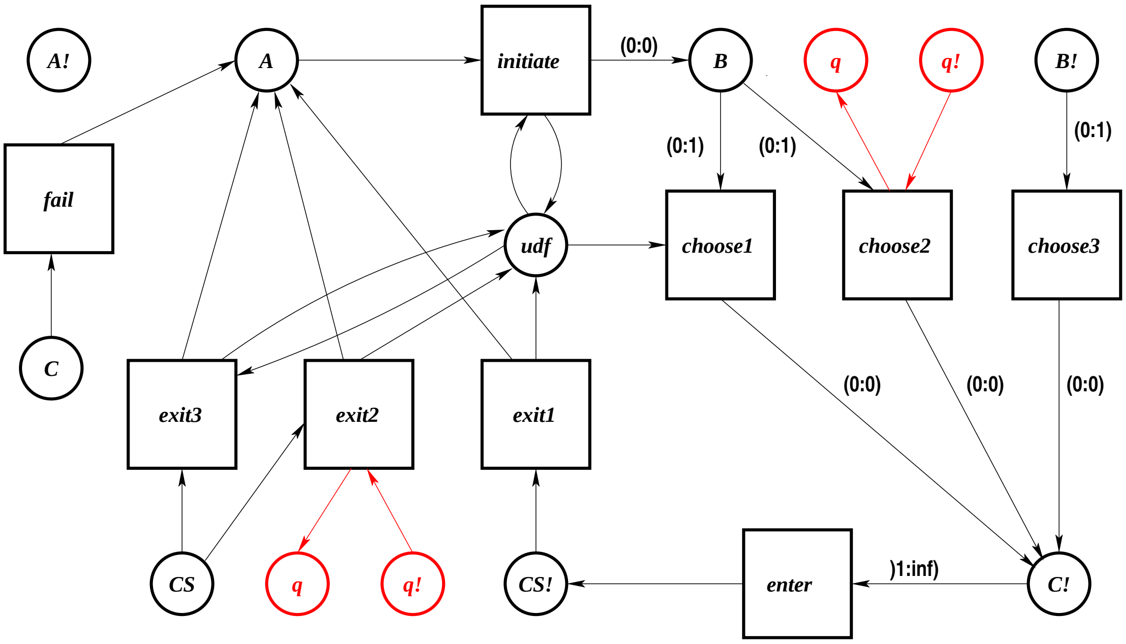

Figure 1: TPN model of Fischer's Protocol for Mutual Exclusion

Fischer's protocol is intended to guarantee mutual exclusion in a concurrent system consisting of an arbitrary number of processes. Figure 1 shows a timed Petri net model of the protocol. Tokens in the places A, B, C, CS, $A!$, $B!$, $C!$ and $CS!$ represent processes running the protocol. The critical section is modelled by the places CS and $CS!$, so mutual exclusion is satisfied when the number of tokens in those places is less than two. See [AN01] for a more detailed description of the protocol.

We prove that mutual exclusion is guaranteed by providing a set of region generators and then verifying that the set is an invariant. The mutual exclusion property is specified as an upward closed set U, where U is characterized by its set of minimal elements. Therefore, the emptiness of intersection of U with the invariant is equivalent to the membership of the set of minimal elements in the invariant.

Table 1 shows the invariant used in our analysis. The post-image of the invariant, computed by our tool, consists of the region generators in Table 2. The last column shows that all region generators generated in the analysis are subsumed by region generators in the invariant.

The implementation takes 1.01 s and 2 MB memory to compute the 21 region generators in the post-image of the invariant due to discrete transitions and the 9 region generators due to timed transitions, on an Intel Pentium III 1 Ghz processor with 256 MB RAM.

ϕ_1	$\left(\{A(0), A(1), B(0), B(1), C(0), C(1)\}^{\circledast} \odot C!(0),\ \left(\{A(0), B(0), C(0)\}^{\circledast}\right)^*,\ \{A, B, C\}^{\circledast}\right)$
ϕ_2	$(\{A(0), A(1), B(1), C(0), C(1)\}^{\circledast},$ $\left(\{A(0), C(0)\}^{\circledast}\right)^* \bullet (\{A(0), B(0), C(0)\}^{\circledast} \odot C!(0)) \bullet \left(\{A(0), B(0), C(0)\}^{\circledast}\right)^*,\ \{A, B, C\}^{\circledast}\Big)$
ϕ_3	$\left(\{A(0), A(1), B(1), C(0), C(1)\}^{\circledast} \odot C!(1),\ \left(\{A(0), C(0)\}^{\circledast}\right)^*,\ \{A, B, C\}^{\circledast}\right)$
ϕ_4	$\left(\{A(0), A(1), C(0), C(1)\}^{\circledast},\ \left(\{A(0), C(0)\}^{\circledast}\right)^*,\ \{A, B, C\}^{\circledast} \odot C!\right)$
ϕ_5	$\left(\{A(0), A(1), C(0), C(1)\}^{\circledast} \odot \{CS!(0), CS!(1)\}, \left(\{A(0), C(0)\}^{\circledast}\right)^*, \{A, B, C\}^{\circledast}\right)$
ϕ_6	$(\{A(0), A(1), C(0), C(1)\}^{\circledast},$ $\left(\{A(0), C(0)\}^{\circledast}\right)^* \bullet (\{A(0), C(0)\}^{\circledast} \odot CS!(0)) \bullet \left(\{A(0), C(0)\}^{\circledast}\right)^*, \{A, B, C\}^{\circledast}\Big)$
ϕ_7	$\left(\{A(0), A(1), C(0), C(1)\}^{\circledast}, \left(\{A(0), C(0)\}^{\circledast}\right)^*, \{A, B, C\}^{\circledast} \odot CS!\right)$
ϕ_8	$\left(\{A(0), A(1), B(0), B(1), C(0), C(1)\}^{\circledast} \odot \{U(0), U(1)\}, \left(\{A(0), B(0), C(0)\}^{\circledast}\right)^*, \{A, B, C\}^{\circledast}\right)$
ϕ_9	$(\{A(0), A(1), B(0), B(1), C(0), C(1)\}^{\circledast},$ $\left(\{A(0), B(0), C(0)\}^{\circledast}\right)^* \bullet (\{A(0), B(0), C(0)\}^{\circledast} \odot U(0)) \bullet \left(\{A(0), B(0), C(0)\}^{\circledast}\right)^*, \{A, B, C\}^{\circledast}\Big)$
ϕ_{10}	$\left(\{A(0), A(1), B(0), B(1), C(0), C(1)\}^{\circledast}, \left(\{A(0), B(0), C(0)\}^{\circledast}\right)^*, \{A, B, C\}^{\circledast} \odot U\right)$

Table 1: An invariant of Fischer's protocol.

10 Conclusions and Future Research

We have proposed a formalism, called *downward closed language generators (dlgs)* for representing infinite sets of states, which arise during forward reachability analysis of infinite-state systems. We show that dlgs allow computing the basic operations involved in the reachability algorithm. We have used dlgs to derive a novel representation for downward closed sets of markings of timed Petri nets. We have built a prototype based on the representation and used the prototype for verifying a a parameterized description of Fischer's protocol.

There are two interesting directions for future research. In this paper, we show how to compute post-images for dlgs with respect to single transitions in timed Petri nets. A remaining challenge is to invent (semi-)algorithms which automatically computes the (downward closure) of the set of markings which are reachable from a given initial marking. To achieve this we intend to develop a general theory, which gives sufficient conditions for computability of transitive closures of dlgs. Such a theory will be the "dual" of the one we have described in [AČJYK00, AN00] for upward closed languages. A main task is to define general acceleration techniques based on computing *meta-transitions* which correspond to iterating a given sequence of transition an arbitrary number of times (see e.g. [AAB00, ABJ98]).

Also, we intend to investigate designing data structures for efficient application of our methodology to concrete models. As a first step we intend to investigate using *zone generators* for analyzing behaviours of timed Petri nets. *Zones* are widely used in existing tools for verification of timed automata [LPY97, BDL+01, Yov97]. Each zone generator will correspond to a large number of region generators (Section 7), and therefore gives a much more compact representation (in a similar manner to the way *existential zones* are superior to *existential regions* [AN01, AN00]).

Reg. gen.	Trans.	*Post*	$\subseteq$
ϕ_1	choose2	ϕ_1	ϕ_1
ϕ_1	fail	ϕ_1	ϕ_1
ϕ_1	time	$\left(\{A(1),B(1),C(1)\}^{\circledast},(\{A(0),B(0),C(0)\}^{\circledast}\odot C!(0))\bullet\left(\{A(0),B(0),C(0)\}^{\circledast}\right)^{*},\ \{A,B,C\}^{\circledast}\right)$ $\left(\{A(1),B(1),C(1)\}^{\circledast}\odot C!(1),\varnothing,\ \{A,B,C\}^{\circledast}\right)$ $\left(\varnothing,\varnothing,\ \{A,B,C\}^{\circledast}\odot C!\right)$	ϕ_2 ϕ_3 ϕ_4
ϕ_2	choose2	ϕ_1	ϕ_1
ϕ_2	fail	ϕ_2	ϕ_2
ϕ_2	time	$\left(\{A(1),B(1),C(1)\}^{\circledast},\right.$ $\left.\left(\{A(0),C(0)\}^{\circledast}\right)^{*}\bullet(\{A(0),B(0),C(0)\}^{\circledast}\odot C!(0))\bullet\left(\{A(0),B(0),C(0)\}^{\circledast}\right)^{*},\{A,B,C\}^{\circledast}\right)$ $\left(\{A(1),B(1),C(1)\}^{\circledast}\odot C!(1),\left(\{A(0),C(0)\}^{\circledast}\right)^{*},\ \{A,B,C\}^{\circledast}\right)$ $\left(\{A(1),C(1)\}^{\circledast},\left(\{A(0),C(0)\}^{\circledast}\right)^{*},\ \{A,B,C\}^{\circledast}\odot C!\right)$	ϕ_2 ϕ_3 ϕ_4
ϕ_3	choose2	ϕ_1	ϕ_1
ϕ_3	fail	ϕ_3	ϕ_3
ϕ_3	time	$\left(\{A(1),C(1)\}^{\circledast},\left(\{A(0),C(0)\}^{\circledast}\right)^{*},\ \{A,B,C\}^{\circledast}\odot C!\right)$	ϕ_4
ϕ_4	enter	ϕ_5 ϕ_6 ϕ_7	ϕ_5 ϕ_6 ϕ_7
ϕ_4	fail	ϕ_4	ϕ_4
ϕ_4	time	$\left(\{A(1),C(1)\}^{\circledast},\left(\{A(0),C(0)\}^{\circledast}\right)^{*},\ \{A,B,C\}^{\circledast}\odot C!\right)$	ϕ_4
ϕ_5,ϕ_6,ϕ_7	exit1	$\left(\{A(0),A(1),C(0),C(1)\}^{\circledast}\odot\{U(0),U(1)\},\left(\{A(0),C(0)\}^{\circledast}\right)^{*},\ \{A,B,C\}^{\circledast}\right)$ $\left(\{A(0),A(1),C(0),C(1)\}^{\circledast},\right.$ $\left.\left(\{A(0),C(0)\}^{\circledast}\right)^{*}\bullet(\{A(0),C(0)\}^{\circledast}\odot U(0))\bullet\left(\{A(0),C(0)\}^{\circledast}\right)^{*},\ \{A,B,C\}^{\circledast}\right)$ $\left(\{A(0),A(1),C(0),C(1)\}^{\circledast},\left(\{A(0),C(0)\}^{\circledast}\right)^{*},\ \{A,B,C\}^{\circledast}\odot U\right)$	ϕ_8 ϕ_9 ϕ_{10}
ϕ_5	fail	ϕ_5	ϕ_5
ϕ_5	time	$\left(\{A(1),C(1)\}^{\circledast},(\{A(0),C(0)\}^{\circledast}\odot CS!(0))\bullet\left(\{A(0),C(0)\}^{\circledast}\right)^{*},\ \{A,B,C\}^{\circledast}\right)$ $\left(\{A(1),C(1)\}^{\circledast}\odot CS!(1),\varnothing,\ \{A,B,C\}^{\circledast}\right)$ $\left(\{A(1),C(1)\}^{\circledast},\left(\{A(0),C(0)\}^{\circledast}\right)^{*},\ \{A,B,C\}^{\circledast}\odot CS!\right)$	ϕ_6 ϕ_5 ϕ_7
ϕ_6	fail	ϕ_6	ϕ_6
ϕ_6	time	$\left(\{A(1),C(1)\}^{\circledast},\left(\{A(0),C(0)\}^{\circledast}\right)^{*}\bullet(\{A(0),C(0)\}^{\circledast}\odot CS!(0))\bullet\left(\{A(0),C(0)\}^{\circledast}\right)^{*},\{A,B,C\}^{\circledast}\right)$ $\left(\{A(1),C(1)\}^{\circledast}\odot CS!(1),\left(\{A(0),C(0)\}^{\circledast}\right)^{*},\ \{A,B,C\}^{\circledast}\right)$ $\left(\{A(1),C(1)\}^{\circledast},\left(\{A(0),C(0)\}^{\circledast}\right)^{*},\ \{A,B,C\}^{\circledast}\odot CS!\right)$	ϕ_6 ϕ_5 ϕ_7
ϕ_7	fail	ϕ_7	ϕ_7
ϕ_7	time	$\left(\{A(1),C(1)\}^{\circledast},\left(\{A(0),C(0)\}^{\circledast}\right)^{*},\ \{A,B,C\}^{\circledast}\odot CS!\right)$	ϕ_7
ϕ_8,ϕ_9,ϕ_{10}	initiate	ϕ_8 ϕ_9 ϕ_{10}	ϕ_8 ϕ_9 ϕ_{10}
ϕ_8,ϕ_9,ϕ_{10}	choose1	ϕ_1	ϕ_1
ϕ_8	fail	ϕ_8	ϕ_8
ϕ_8	time	$\left(\{A(1),B(1),C(1)\}^{\circledast},(\{A(0),B(0),C(0)\}^{\circledast}\odot U(0))\bullet\left(\{A(0),B(0),C(0)\}^{\circledast}\right)^{*},\ \{A,B,C\}^{\circledast}\right)$ $\left(\{A(1),B(1),C(1)\}^{\circledast},\left(\{A(0),B(0),C(0)\}^{\circledast}\right)^{*},\ \{A,B,C\}^{\circledast}\odot U\right)$ $\left(\{A(1),B(1),C(1)\}^{\circledast}\odot U(1),\varnothing,\ \{A,B,C\}^{\circledast}\right)$	ϕ_9 ϕ_{10} ϕ_8
ϕ_9	fail	ϕ_9	ϕ_9
ϕ_9	time	$\left(\{A(1),B(1),C(1)\}^{\circledast},\right.$ $\left(\{A(0),B(0),C(0))\}^{\circledast}\right)^{*}\bullet(\{A(0),B(0),C(0)\}^{\circledast}\odot U(0))\bullet\left(\{A(0),B(0),C(0)\}^{\circledast}\right)^{*},$ $\left.\{A,B,C\}^{\circledast}\right)$ $\left(\{A(1),B(1),C(1)\}^{\circledast}\odot U(1),\left(\{A(0),B(0),C(0)\}^{\circledast}\right)^{*},\ \{A,B,C\}^{\circledast}\right)$ $\left(\{A(1),B(1),C(1)\}^{\circledast},\left(\{A(0),B(0),C(0)\}^{\circledast}\right)^{*},\ \{A,B,C\}^{\circledast}\odot U\right)$	ϕ_9 ϕ_8 ϕ_{10}
ϕ_{10}	fail	ϕ_{10}	ϕ_{10}
ϕ_{10}	time	$\left(\{A(1),B(1),C(1)\}^{\circledast},\left(\{A(0),B(0),C(0)\}^{\circledast}\right)^{*},\ \{A,B,C\}^{\circledast}\odot U\right)$	ϕ_{10}

Table 2: Post-image of the invariant.

References

[AAB+99a] Parosh Aziz Abdulla, Aurore Annichini, Saddek Bensalem, Ahmed Bouajjani, Peter Habermehl, and Yassine Lakhnech. Verification of infinite-state systems by combining abstraction and reachability analysis. In *Proc.* 11^{th} *Int. Conf. on Computer Aided Verification*, volume 1633 of *Lecture Notes in Computer Science*, pages 146–160, 1999.

[AAB99b] Parosh Aziz Abdulla, Aurore Annichini, and Ahmed Bouajjani. Algorithmic verification of lossy channel systems: An application to the bounded retransmission protocol. In *Proc. TACAS '99,* 5^{th} *Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1579 of *Lecture Notes in Computer Science*, 1999.

[AAB00] A. Annichini, E. Asarin, and A. Bouajjani. Symbolic techniques for parametric reasoning about counter and clock systems. In *Proc.* 12^{th} *Int. Conf. on Computer Aided Verification*, volume 1855 of *Lecture Notes in Computer Science.* Springer Verlag, 2000.

[ABB01] Parosh Aziz Abdulla, Luc Boasson, and Ahmed Bouajjani. Effective lossy queue languages. In *Proc. ICALP '2001,* 28^{th} *International Colloquium on Automata, Lnaguages, and Programming*, volume 2076 of *Lecture Notes in Computer Science*, pages 639–651, 2001.

[ABJ98] Parosh Aziz Abdulla, Ahmed Bouajjani, and Bengt Jonsson. On-the-fly analysis of systems with unbounded, lossy fifo channels. In *Proc.* 10^{th} *Int. Conf. on Computer Aided Verification*, volume 1427 of *Lecture Notes in Computer Science*, pages 305–318, 1998.

[ABLS01] A. Annichini, A. Bouajjani, Y. Lakhnech, and M. Sighireanu. Analyzing fair parametric extended automata. In P. Cousot, editor, *Proceedings of the International Symposium on Static Analysis*, volume 2126 of *Lecture Notes in Computer Science*, 2001.

[AČJYK00] Parosh Aziz Abdulla, Karlis Čerāns, Bengt Jonsson, and Tsay Yih-Kuen. Algorithmic analysis of programs with well quasi-ordered domains. *Information and Computation*, 160:109–127, 2000.

[AD90] R. Alur and D. Dill. Automata for modelling real-time systems. In *Proc. ICALP '90*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335, 1990.

[AJ96] Parosh Aziz Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2):91–101, 1996.

[AJ98] Parosh Aziz Abdulla and Bengt Jonsson. Verifying networks of timed processes. In Bernhard Steffen, editor, *Proc. TACAS '98,* 4^{th} *Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1384 of *Lecture Notes in Computer Science*, pages 298–312, 1998.

[AN00] Parosh Aziz Abdulla and Aletta Nylén. Better is better than well: On efficient verification of infinite-state systems. In *Proc. LICS' 00* 16^{th} *IEEE Int. Symp. on Logic in Computer Science*, pages 132–140, 2000.

[AN01] Parosh Aziz Abdulla and Aletta Nylén. Timed Petri nets and BQOs. In *Proc.* 22^{th} *Int. Conf. on Application and Theory of Petri Nets*, volume 2075 of *Lecture Notes in Computer Science*, pages 53 –70, 2001.

[BDL+01] G. Behrmann, A. David, K. G. Larsen, O. Möller, Paul Pettersson, and Wang Yi. UPPAAL - present and future. In *Proc. CDC'2001, the 40th IEEE Conference on Decision and Control*, 2001.

[BLO98] S. Bensalem, Y. Lakhnech, and S. Owre. Computing abstractions of infinite state systems automatically and compositionally. In Alan J. Hu and Moshe Y. Vardi, editors, *Computer Aided Verification*, volume 1427 of *Lecture Notes in Computer Science*, pages 319–331. Springer-Verlag, 1998.

[DEP99] G. Delzanno, J. Esparza, and A. Podelski. Constraint-based analysis of broadcast protocols. In *Proc. CSL'99*, 1999.

[DR00] G. Delzanno and J. F. Raskin. Symbolic representation of upward-closed sets. In *Proc. TACAS '00,* 6^{th} *Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1785 of *Lecture Notes in Computer Science*, pages 426–440, 2000.

[EFM99] J. Esparza, A. Finkel, and R. Mayr. On the verification of broadcast protocols. In *Proc. LICS' 99* 14^{th} *IEEE Int. Symp. on Logic in Computer Science*, 1999.

[FIS00] A. Finkel, S. Purushothaman Iyer, and G. Sutre. Well-abstracted transition systems. In *Proc. CONCUR 2002,* 13^{th} *Int. Conf. on Concurrency Theory*, pages 566–580, 2000.

[FRSB02] A. Finkel, J.-F. Raskin, M. Samuelides, and L. Van Begin. Monotonic extensions of petri nets: Forward and backward search revisited. In *Proc. Infinity'02*, 2002.

[God94] J.C. Godskesen. *Timed Modal Specifications.* PhD thesis, Aalborg University, 1994.

[LBBO01] Y. Lakhnech, S. Bensalem, S. Berezin, and S. Owre. Symbolic techniques for parametric reasoning about counter and clock systems. In *Proc. TACAS '01,* 7^{th} *Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 2031 of *Lecture Notes in Computer Science.* Springer Verlag, 2001.

[LPY97] K.G. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *Software Tools for Technology Transfer*, 1(1-2), 1997.

[May00] R. Mayr. Undecidable problems in unreliable computations. In *Theoretical Informatics (LATIN'2000)*, number 1776 in Lecture Notes in Computer Science, 2000.

[SBM92] F. B. Schneider, B. Bloom, and K. Marzullo. Putting time into proof outlines. In de Bakker, Huizing, de Roever, and Rozenberg, editors, *Real-Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science*, 1992.

[Yov97] S. Yovine. Kronos: A verification tool for real-time systems. *Journal of Software Tools for Technology Transfer*, 1(1-2), 1997.

A Proofs

A.1 Correctness of the Intersection Algorithm

The case for expressions is obvious.

For products ϕ_1 and ϕ_2, we let $Alg(\phi_1, \phi_2)$ be the dlg computed by the algorithm (when given ϕ_1 and ϕ_2 as input). We show that $w \in Alg(\phi_1, \phi_2)$ if and only if $w \in \phi_1 \cap \phi_2$ for each word w.

Assume $w \in \phi_1 \cap \phi_2$. We show that $w \in Alg(\phi_1, \phi_2)$. We use induction on the number of expressions in p_1 and p_2. If either ϕ_1 or ϕ_2 is empty then the result follows trivially. Otherwise let $\phi_1 = e_1 \bullet p_1$ and $\phi_2 = e_2 \bullet p_2$ (as described by the algorithm). If $w \in p_1$ then $w \in p_1 \cap \phi_2$. By the induction hypothesis it follows that $w \in Alg(p_1, \phi_2)$. From definition of the algorithm and downward closedness it follows that $w \in Alg(\phi_1, \phi_2)$. The case when $w_2 \in p_2$ is similar. If $w \notin p_1$ and $w \notin p_2$, there are w_1, w_2, w_3, w_4 such that w_1 and w_3 are not empty, and $w = w_1 \bullet w_2 = w_3 \bullet w_4$, $w_1 \in e_1$, $w_2 \in p_1$, $w_3 \in e_2$, and $w_4 \in p_2$.

We consider four cases corresponding to those in the definition of the algorithm.

1. If e_1 and e_2 are atomic expressions. This means that $w_1 = w_3 = a$ for some a in the alphabet, and that $w_2 = w_4$. By induction hypothesis, it follows that $a \in Alg(e_1, e_2)$ and that $w_2 \in Alg(p_1, p_2)$. By definition of case 1 of the algorithm it follows that $w \in Alg(\phi_1, \phi_2)$.

2. If e_1 is an atomic expression and e_2 is a star expression. This means that $w_1 = a$, and that w_3 is of the form $a \bullet w_3'$. By downward closedness we know that $a \in e_2$ and that $w_3' \bullet w_4 = w_2 \in \phi_2$. By induction hypothesis, it follows that $a \in Alg(e_1, e_2)$ and that $w_2 \in Alg(p_1, \phi_2)$. By definition of case 2 of the algorithm it follows that $w \in Alg(\phi_1, \phi_2)$.

3. Similar to case 2 above.

4. If e_1 and e_2 are star expressions. There are two symmetric cases: If w_1 is a prefix of w_3, i.e., there is a w_3' with $w_3 = w_1 \bullet w_3'$ (and therefore $w_2 = w_3' \bullet w_4$). By downward closedness we know that $w_1 \in e_2$ and that $w_3' \bullet w_4 = w_2 \in \phi_2$. By induction hypothesis, it follows that $w_1 \in Alg(e_1, e_2)$ and that $w_2 \in Alg(p_1, \phi_2)$. By definition of case 4 of the algorithm it follows that $w \in Alg(\phi_1, \phi_2)$. If w_3 is a prefix of w_1,then the proof is similar.

Now, we consider the other direction. Suppose that $w \in Alg(\phi_1, \phi_2)$. Again, we use induction on the number of expressions in p_1 and p_2. The base case is similar to above. For the induction step, we have to prove the following:

1. $w \in \phi_1 \cap p_2 \Rightarrow w \in \phi_1 \cap \phi_2$.
2. $w \in p_1 \cap \phi_2 \Rightarrow w \in \phi_1 \cap \phi_2$.
3. $w \in (e_1 \cap e_2) \bullet (p_1 \cap p_2) \Rightarrow w \in \phi_1 \cap \phi_2$.
4. $w \in (e_1 \cap e_2) \bullet (\phi_1 \cap p_2) \Rightarrow w \in \phi_1 \cap \phi_2$ when e_1 is a star expression
5. $w \in (e_1 \cap e_2) \bullet (p_1 \cap \phi_2) \Rightarrow w \in \phi_1 \cap \phi_2$ when e_2 is a star expression

We show the proof for case 4. The other cases can be analyzed in a similar manner.

Assume $w \in (e_1 \cap e_2) \bullet (\phi_1 \cap p_2)$. Let w_1, w_2 be words such that $w = w_1 \bullet w_2$, $w_1 \in e_1 \cap e_2$ and $w_2 \in \phi_1 \cap p_2$. Since, $w_1 \in e_1$ and $w_2 \in \phi_1$, we have $w \in e_1 \bullet \phi_1$. Since e_1 is a star expression, we have $e_1 \bullet \phi_1 = \phi_1$. Also, $w_1 \in e_2$ and $w_2 \in p_2$ imply that $w \in \phi_2$. Therefore, $w \in \phi_1 \cap \phi_2$.

Correctness of the case for sums follows immediately from correctness for products.

A.2 Theorem 3.2

For a word, we let $\overline{w \uparrow}$ denote the set $\{w' |\ w \not\preceq^* w'\}$. Consider a word $w = a_1 a_2 \cdots a_k$. It is clear that $\overline{w \uparrow}$ is equal to

$$\left(A^{\not\succeq a_1}\right)^* \bullet\ (A \cup \{\epsilon\})\ \bullet\ \left(A^{\not\succeq a_2}\right)^* \bullet\ (A \cup \{\epsilon\})\ \bullet\ \cdots\ \bullet\ (A \cup \{\epsilon\})\ \bullet\ \left(A^{\not\succeq a_k}\right)^*$$

which, in turn is characterized by the dlg

$$\left(\overset{*}{\sim}\, a_1\right)\ \bullet\ (\sim \emptyset)\ \bullet\ \left(\overset{*}{\sim}\, a_2\right)\ \bullet\ (\sim \emptyset)\ \bullet\ \cdots\ \bullet\ (\sim \emptyset)\ \bullet\ \left(\overset{*}{\sim}\, a_k\right)$$

This gives the following

Lemma A.1. *For a w over A, there is a dlg ϕ such that $\mathcal{L}(\phi) = \overline{w \uparrow}$.*

Now, assume a downward closed language L. The complement $\overline{L}$ is upward closed and can be characterized by its minimal elements, i.e., $\overline{L} = \min(\overline{L}) \uparrow$. Let $\min(\overline{L})$ be the set $\{w_1, \ldots, w_n\}$. This set is finite by well quasi-ordering. It follows that $L = \overline{w_1 \uparrow} \cap \cdots \cap \overline{w_n \uparrow}$.

By Lemma A.1 we know that there are dlgs $\phi_1, \ldots, \phi_n$ such that $\overline{w_i \uparrow} = \phi_i$ and by Theorem 3.1 there is a dlg $\phi = \phi_1 \cap \cdots \cap \phi_n$.

A.3 Correctness of the Entailment Algorithm

For a product p and a dlg ϕ, we use $Alg(p, \phi)$ to indicate that the algorithm answers "yes" to the question whether $p \subseteq \phi$, and use $\neg Alg(p, \phi)$ to indicate that the algorithm answers "no".

Suppose that $Alg(p, \phi)$ holds. We show that $p \subseteq \phi$. We use induction on the number of atomic expressions in p.

Base Case If p is empty, the result is trivial.

Induction step Let ϕ be of the form $p_1 \mid \cdots \mid p_n$. Suppose that p is of the form $e \bullet p'$. There are two cases:

1. e is an atomic expression. Let $w \in p$. We show that $w \in \phi$.

   If $w \in p'$ then by definition of the algorithm, it follows that $Alg(p', \phi)$, so by the induction hypothesis, we have $w \in \phi$.

Otherwise, let w be of the form $a \bullet w_1$ where $a \in e$ and $w_1 \in p'$. By definition of the algorithm, we know that there is a factorization of ϕ with respect to e of the form shown in the algorithm such that $a \in e_i$ for each $i : 1 \leq i \leq m$, and such that $p' \subseteq q_1 \mid \cdots \mid q_m$. This means that $w_1 \in q_i$ for some $i : 1 \leq i \leq m$, and hence $w = a \bullet w_1 \in e_i \bullet q_i$. This implies $w \in \phi$.

2. e is a star expression. Let $w \in p$.

   If $w \in p'$ then the same argument as in case 1 applies.

   Otherwise, we know that $w = w_1 \bullet w_2$, where w_1 is not empty, $w_1 \in e$, and $w_2 \in p'$. By definition of the algorithm, we know that there is a factorization of ϕ with respect to e of the form shown in the algorithm such that $w_1 \in e_i$ for each $i : 1 \leq i \leq m$, and such that $p' \subseteq q_1 \mid \cdots \mid q_m$. This means that $w_2 \in q_i$ for some $i : 1 \leq i \leq m$, and hence $w = w_1 \bullet w_2 \in e_i \bullet q_i$. This implies $w \in \phi$.

Now, we show the other direction. Suppose $\neg Alg(p, \phi)$. We show that there is a $w \in p$ such that $w \notin \phi$. We use induction on the length of p.

Base Case If p is empty, then the result is trivial.

Induction Step Let p be of the form $e \bullet p'$. We distinguish between two cases.

1. e is an atomic expression. We consider two subcases.

   (a) If $\#e \not\subseteq \#p_1 \cup \cdots \cup \#p_n$ then there is an $a \in \#e$ with $a \notin \#p_1 \cup \cdots \cup \#p_n$. Take $w = a$.

   (b) There is a factorization of ϕ with respect e of the form shown in the algorithm, with $\neg Alg(p', q_1 \mid \cdots \mid q_m)$. By definition we know that there is $a \in e$ with $a \notin \#p'_1 \cup \cdots \cup \#p'_m \cup \#Q_2$. Also, by induction hypothesis we know that $p' \not\subseteq q_1 \mid \cdots \mid q_m$, so there is $w_1 \in p'$ with $w_1 \notin q_1 \mid \cdots \mid q_m$. This implies that $a \bullet w_1 \notin p_i$ for each $i : 1 \leq i \leq n$. Take $w = a \bullet w_1$.

2. e is a star expression. We consider two subcases.

   (a) $\#e \not\subseteq \#p_1 \cup \cdots \cup \#p_n$. There is a $w_1 \in \#e$ with $w_1 \notin \#p_1 \cup \cdots \cup \#p_n$. Take $w = w_1^k$ (i.e., w is the concatenation of k copies of w_1), where k is the largest number of expressions present in any p_i with $i : 1 \leq i \leq n$. It is clear that $w \in e$ and hence $w \in p$, and that $w \notin p_i$ for each $i : 1 \leq i \leq n$.

   (b) There is a factorization of ϕ with respect e of the form shown in the algorithm, with $\neg Alg(p', q_1 \mid \cdots \mid q_m)$. It follows that there is a $w_1 \in e$ with $w_1 \notin \#p'_1 \cup \cdots \cup \#p'_m \cup \#Q_2$. Define k as in the previous case. Also, by induction hypothesis, there is a $w_2 \in p'$ with $w_2 \notin q_1 \mid \cdots \mid q_m$. Take $w = w_1^k \bullet w_2$.

A.4 Correctness of the Multiset Intersection Algorithm

For products p_1 and p_2, we let $Alg(p_1, p_2)$ be the dlg computed by the algorithm (when given p_1 and p_2 as input). We show that $b \in Alg(p_1, p_2)$ if and only if $b \in p_1 \cap p_2$ for each multiset b.

Assume $b \in p_1 \cap p_2$. We show that $b \in Alg(p_1, p_2)$. We use induction on the number of expressions in p_1 and p_2. If either p_1 or p_2 is empty then the result follows trivially. Otherwise let $p_1 = e_{11} \odot \cdots \odot e_{1k}$ and $p_2 = e_{21} \odot \cdots \odot e_{2m}$ (as described by the algorithm).

Assume $b \in p_1 \cap p_2$. We show that $b \in Alg(p_1, p_2)$. Let b be of the form $b_1 + b_2$ where $b_1 \in e_{1i}$ and $b_2 \in p_{1i}$ for some $i : 1 \leq i \leq k$. There are two cases

1. e_{1i} is an atomic expression. Since $b \in p_2$ we know that there is j such that $b_1 \in e_{2j}$ and either e_{2j} is atomic and $b_2 \in p_{2j}$, or e_{2j} is a star expression and $b_2 \in p_2$.
   (a) If e_{2j} is atomic, then it follows that $b_1 \in e_{1i} \cap e_{2j}$ and $b_2 \in p_{1i} \cap p_{2j}$. By induction hypothesis, we have that $b_2 \in Alg(p_{1i}, p_{2j})$. It follows that $b \in \phi_{ij} = (e_{1i} \cap e_{2j}) \odot Alg(p_{1i}, p_{2j})$, i.e., $b \in Alg(p_1, p_2)$.
   (b) If e_{2j} is a star expression we obtain the result by following the same reasoning as in the above case.
2. e_{1i} is a star expression. Let $b = b_3 + b_4$ such that there is j with $b_3 \in e_{2j}$ and either e_{2j} is atomic and $b_4 \in p_{2j}$, or e_{2j} is a star expression and $b_4 \in p_2$. Furthermore, either $b_1 \leq b_3$[1] or $b_3 \leq b_1$.
   (a) If e_{2j} is atomic, then $b_1 = b_3 + b_1'$ and by downward closure we know that $b_3 \in e_{1i}$ and $b_1' + b_2 = b_4 \in p_1$. By induction hypothesis, we have that $b_4 \in Alg(p_1, p_{2j})$. It follows that $b \in Alg(p_1, p_2)$.
   (b) If e_{2j} is a star expression then either $b_3 \leq b_1$, in which case the reasoning in the previous case applies, or $b_1 \leq b_3$ which is symmetric.

Now, we consider the other direction. Suppose that $b \in Alg(p_1, p_2)$. Again, we use induction on the number of expressions in p_1 and p_2. The base case is similar to above. For the induction step, we have to prove the following:

1. $b \in (e_{1i} \cap e_{2j}) \odot (p_{1i} \cap p_{2j}) \Rightarrow b \in p_1 \cap p_2$.
2. $b \in (e_{1i} \cap e_{2j}) \odot (p_1 \cap p_{2j}) \Rightarrow b \in p_1 \cap p_2$ when e_1 is a star expression.
3. $b \in (e_{1i} \cap e_{2j}) \odot (p_{1i} \cap p_2) \Rightarrow b \in p_1 \cap p_2$ when e_2 is a star expression.

We show the proof for case 2. The other cases can be analyzed in a similar manner.

Assume $b \in (e_{1i} \cap e_{2j}) \odot (p_1 \cap p_{2j})$. Let b_1, b_2 be multisets such that $b = b_1 + b_2$, $b_1 \in e_{1i} \cap e_{2j}$ and $b_2 \in p_1 \cap p_{2j}$. Since, $b_1 \in e_{1i}$ and $b_2 \in p_1$, we have $b \in e_{1i} \odot p_1$. Since e_{1i} is a star expression, we have $e_{1i} \odot p_1 = p_1$. Also, $b_1 \in e_{2j}$ and $b_2 \in p_{2j}$ imply that $b \in p_2$. Therefore, $b \in \phi_1 \cap \phi_2$.

[1] The relation $\leq$ is defined by $b_1 \leq b_2$ iff $b_1{}^a \leq b_2{}^a$ for all a.

A.5 Correctness of the Multiset Entailment Algorithm

For a product p and a dlg ϕ, we use $Alg(p, \phi)$ to indicate that the algorithm answers "yes" to the question whether $p \subseteq \phi$, and use $\neg Alg(p, \phi)$ to indicate that the algorithm answers "no".

Suppose that $Alg(p, \phi)$ holds. We show that $p \subseteq \phi$. We use induction on the number of atomic expressions in p.

Base Case If p is empty, the result is trivial.

Induction step Let ϕ be of the form $p_1 \mid \cdots \mid p_n$. Suppose that p is of the form $e \odot p'$. There are two cases:

1. e is an atomic expression. Let $b \in p$. We show that $b \in \phi$.

   If $b \in p'$ then by definition of the algorithm, it follows that $Alg(p', \phi)$, so by the induction hypothesis, we have $b \in \phi$.

   Otherwise, let b be of the form $a + b_1$ where $a \in e$ and $b_1 \in p'$. By definition of the algorithm, we know that there is set F of factorizations of ϕ with respect to e of the form shown in the algorithm such that $a \in cause(F)$. Let the union $\bigcup F$ be the set $\{q_1, \ldots, q_M\}$. We have that $a \in e_i$ for each $i : 1 \leq i \leq M$. We also know that there $p' \subseteq q_1 \mid \cdots \mid q_M$. This means that $b_1 \in q_i$ for some $i : 1 \leq i \leq M$, and hence $b = a + b_1 \in e_i \odot q_i$. This implies $b \in \phi$.

2. e is a star expression. Let $b \in p$.

   If $b \in p'$ then the same argument as in case 1 applies.

   Otherwise, we know that $b = b_1 + b_2$, where b_1 is not empty, $b_1 \in e$ and $b_2 \in p'$. By definition of the algorithm, we know that there is a factorization of ϕ with respect to e of the form shown in the algorithm such that $b_1 \in e_i$ for each $i : 1 \leq i \leq m$, and such that $p' \subseteq q_1 \mid \cdots \mid q_m$. Since we assume that there is at most one star expression in each product, we know that there is only one such factorization. This means that $b_2 \in q_i$ for some $i : 1 \leq i \leq m$, and hence $b = b_1 + b_2 \in e_i \odot q_i$. This implies $b \in \phi$.

Now, we show the other direction. Suppose $\neg Alg(p, \phi)$. We show that there is a $b \in p$ such that $b \notin \phi$. We use induction on the length of p.

Base Case If p is empty, then the result is trivial.

Induction Step Let p be of the form $e \odot p'$. We distinguish between two cases.

1. e is an atomic expression. We consider two subcases.

   (a) If $\#e \not\subseteq \#p_1 \cup \cdots \cup \#p_n$ then there is an $a \in \#e$ with $a \notin \#p_1 \cup \cdots \cup \#p_n$. Take $b = (a)$.

   (b) There is a set F of factorizations of ϕ with respect to e of the form shown in the algorithm such that $\neg Alg(p', q_1 \mid \cdots \mid q_M)$ where $\bigcup F = \{q_1, \ldots, q_M\}$. By definition we know that there is $a \in e$ with $a \notin \#Q_2$. Also, by induction hypothesis we know that $p' \not\subseteq q_1 \mid \cdots \mid q_M$, so there is $b_1 \in p'$ with $b_1 \notin q_1 \mid \cdots \mid q_M$. This implies that $a + b_1 \notin p_i$ for each $i : 1 \leq i \leq n$. Take $b = a + b_1$.

2. e is a star expression. We consider two subcases.

   (a) $\#e \not\subseteq \#p_1 \cup \cdots \cup \#p_n$. There is a $b_1 \in \#e$ with $b_1 \notin \#p_1 \cup \cdots \cup \#p_n$. Take $b = b_1^k$ (i.e., b is the concatenation of k copies of b_1), where k is the largest number of expressions present in any p_i with $i : 1 \leq i \leq n$. It is clear that $b \in e$ and hence $b \in p$, and that $b \notin p_i$ for each $i : 1 \leq i \leq n$.

   (b) There is a factorization of ϕ with respect e of the form shown in the algorithm, with $\neg Alg(p', q_1 \mid \cdots \mid q_M)$. It follows that there is a $b_1 \in e$ with $b_1 \notin \#Q_2$. Define k as in the previous case. Also, by induction hypothesis, there is a $b_2 \in p'$ with $b_2 \notin q_1 \mid \cdots \mid q_M$. Take $b = b_1^k + b_2$.