Concurrency Algebra and Petri Nets

Young-il Choo

Computer Science
California Institute of Technology

5190:TR:85

Concurrency Algebra and Petri Nets

Young-il Choo

Computer Science Department
California Institute of Technology

5190:TR:85

June 7, 1985

The research described in this paper was sponsored by the Defense Advanced Research Projects Agency, ARPA Order No. 3771, and monitored by the Office of Naval Research under contract number N00014-79-C-0597.

Concurrency Algebra and Petri Nets

Young-il Choo
Caltech Computer Science
June 7, 1985

Abstract

Concurrency Algebra is an algebraic framework for reasoning about the dynamic behavior of Petri nets. Based on a simple algebra containing concatenation, choice, and shuffle of strings, concurrency algebra uses a substitution schema determined by the structure of the given Petri net to generate new terms that denote its behavior. For non-terminating Petri nets, the behavior becomes the smallest set of strings satisfying a recursive equation over sets of strings.

1 Introduction

Petri nets are mathematical constructs for modeling concurrent activity. Their structure is given by a bipartite directed graph and their behavior is determined by the movement of tokens around the graph. The behavior is usually represented as the set of strings generated by the firings of the transitions.

Modeling a concurrent system using a Petri net is not too difficult. Once the Petri net is obtained, however, analyzing its behavior quickly becomes unmanageable because of the combinatorial nature of the set of possible strings of behaviors. One way to deal with this complexity is by restricting the structure of Petri nets. The restrictions traditionally imposed tend to be syntactic ones that do not provide much help in analysis.

Another approach is to provide rules for generating new Petri nets that already have the desired behavior such as liveness and safeness. In Choo [82] an hierarchical approach was taken to construct nets that are safe and easily tested for liveness. Even with these issues solved, specifying the actual behavior of a Petri net still remains.

What we would like is to reduce the complexity by representing sets of behaviors as expressions in some algebraic framework and also be able to manipulate them to determine their behavior.

In this paper we define a framework called *Concurrency Algebra*, and show how it can be used to reason about concurrent systems represented as Petri nets. Concurrency Algebra consists of an algebra whose expressions denote sets of strings, and a system of substitutions that allows new expressions to be generated from old.

A work somewhat similar to this is Crespi-Reghizzi and Mandrioli [76]. They use a formal language approach for defining Petri net behavior, though they do not use an algebra with explicit use of operators and identities.

Familiarity with the basic concepts of Petri nets is assumed. An excellent book with lots of references into the literature is Peterson [81].

2 Elements of Concurrency Algebra

The ingredients of concurrency algebra expressions are motivated by the behavior of Petri nets. There is the basic behavior of transitions firing in a sequence; the choice among two or more transitions where only one can fire; and the behavior where several transitions can fire in parallel without any constraint in their order. These ideas are captured in the three operators defined below.

Definition

An *expression* in Concurrency Algebra is made of letters (a, b, c, etc.) from a given alphabet Σ, and three binary operators: concatenation (represented as juxtaposition), choice ($|$) and shuffle ($\|$). □

We adopt the convention that concatenation binds tighter than shuffle, which in turn binds tighter than choice. So, $ab \,\|\, c \mid d \stackrel{\text{def}}{=} ((ab) \,\|\, c) \mid d$. Then, due to associativity of the operators, $abcdef$, $ab \mid cde$, $abc \,\|\, de$, and $a(b \mid c) \,\|\, def$ are all valid expressions. The variables x, y, z, etc. will be used to range over expressions.

The expressions satisfy the following identities:

Identities

Associativity:

$$x(yz) = (xy)z$$
$$x \mid (y \mid z) = (x \mid y) \mid z$$
$$x \,\|\, (y \,\|\, z) = (x \,\|\, y) \,\|\, z$$

Commutativity:

$$x \mid y = y \mid x$$
$$x \,\|\, y = y \,\|\, x$$

Distributivity:

$$x(y \mid z) = xy \mid xz$$
$$(y \mid z)x = yx \mid zx$$
$$x \,\|\, (y \mid z) = x \,\|\, y \mid x \,\|\, z$$
$$(y \mid z) \,\|\, x = y \,\|\, x \mid z \,\|\, x$$

Idempotency:

$$x \mid x = x$$

Peeling:

$$ax \,\|\, by = a(x \,\|\, by) \mid b(ax \,\|\, y) \quad \text{where } a \text{ and } b \text{ are letters in } \Sigma.$$ □

Intuitively, concatenation represents sequential activity, choice represents non-deterministic branching, and shuffle represents concurrent activity. Peeling is essentially the inductive definition of the shuffle, where we represent concurrent activity as the set of all possible sequential activity

that is consistent with the concurrent activity. Intuitively, if x and y are considered as decks of cards, the shuffle operator produces all possible shuffles.

For more detail on the shuffle, good references are Ginsburg [66], and Ogden et. al.[78] where the complexity of languages with the shuffle are analyzed.

In a formulation of Petri nets that allows the same name to be given to two or more transitions, the meaning of $a \| b$ and $ab \mid ba$ may not be the same. The latter may denote a nondeterministic choice followed by two sequential activities, while the former denotes true concurrency. For us, we will avoid this issue by assuming that all transitions and places of a Petri net have unique names.

3 The Meaning of Expressions

In this section we define the meaning of concurrency algebra expressions as sets of strings over the alphabet, i.e., a language. The language denoted by an expression is defined inductively on the structure of the expression.

First, to aid the definition of the shuffle, we extend concatenation to sets of strings by elementwise concatenation.

Definition

If A and B are sets of strings, then the concatenation is defined by:

$$AB \stackrel{\text{def}}{=} \{uv \mid u \in A \text{ and } v \in B\}.$$

□

Definition

For an expression of the algebra x, the language generated will be denoted by $[\![x]\!]$ and is defined inductively:

Base case:

$$[\![a]\!] \stackrel{\text{def}}{=} \{a\} \quad \text{for } a \text{ in } \Sigma$$

Concatenation:

$$[\![xy]\!] \stackrel{\text{def}}{=} [\![x]\!][\![y]\!]$$

Choice:

$$[\![x \mid y]\!] \stackrel{\text{def}}{=} [\![x]\!] \cup [\![y]\!]$$

Shuffle:

$$[\![ax \| by]\!] \stackrel{\text{def}}{=} \{a\}[\![x \| by]\!] \cup \{b\}[\![ax \| y]\!]$$

□

If we consider the sets of strings as an algebra with concatenation and union as the two operators, then what we have done is defined $[\![\cdot]\!]$ to be an algebra homomorphism. Therefore, the definition for the shuffle is implied by the peeling rule, although we include it here for sake of completeness.

4 Representing the Structure of Petri Nets

The first step in representing Petri nets in concurrency algebra is to reduce the structure of Petri nets to expressions in the algebra. The key step is in decomposing a Petri net as a union of small

units called atomic nets. Each atomic net contains all the information on the firability of one transition and its effect on the marking. From the atomic nets we generate a set of substitutions, one for each atomic net.

Definition

An *atomic net* is a Petri net consisting of one transition along with its set of input and output places. We denote it as a triple, $\langle \cdot t, t, t \cdot \rangle$, where t is the name of a transition and $\cdot t$ and $t \cdot$ represent, respectively, its set of input places and output places. □

Clearly, a Petri net can be considered to be the union of atomic nets where the places with the same name are identified. Now, let P be the set of names of places and T the set of names of transitions. From here on we let Σ be the union of P and T.

Definitions

A *marking* is a special type of expression whose letters are from P and the only operator used is the shuffle ($\|$). For example, $a \| b \| c$ is a marking. If s is a set of places, then the shuffle of all the elements of s will be denoted by $\| s$. A *labeled marking* is an expression formed by concatenating a transition with a marking. For example, if t is a transition, then $t(a \| b \| c)$ is a labeled marking. □

A marking in Concurrency Algebra is an expression that represents the marking of a Petri net. A labeled marking represents the result after the transition has been fired with the trace of the transition remaining as part of the expression.

Definition

A *substitution* is an ordered pair of a marking and a choice ($|$) of labeled markings. A set of substitutions is *canonical* if no two substitutions have the same left hand sides. □

For example, if m and m' are markings and t is a transition, then a substitution will be written as $m \rightarrow tm'$, and we say that m is the left hand side, and tm' is the right hand side of the substitution. A substitution indicates that from the marking m we can fire the transition t to get a new marking m'.

Generating a Canonical Set of Substitutions

First, decompose the net into its constituent atomic nets. For each atomic net $\langle \cdot t, t, t \cdot \rangle$ construct a substitution $\| \cdot t \rightarrow t(\| t \cdot)$. If there is more than one substitution with the same expression on the left hand side, then combine them into one substitution with the same left hand side but the right hand side is the choice ($|$) of each of the right hand sides. For example, $p \rightarrow t_1$ and $p \rightarrow t_2$ are combined to give $p \rightarrow t_1 \,|\, t_2$.

These substitutions govern when one expression may be replaced by another, and will be used to generate the global behavior.

Definitions

An expression is *expandable* if there is a substitution whose left hand side matches a subexpression of the expression. Given a set of substitutions S and an expandable expression x, we

can generate a new expression y by *applying* a substitution that matches x, written as $x \Rightarrow y$, by replacing the left hand side of the substitution that occurs in the expression with the right hand side. □

For example, we can apply the substitution $p \rightarrow t_1 \,|\, t_2$ to $p \,\|\, q$ to get $(t_1 \,|\, t_2) \,\|\, q$.

The generation of a sequence of expressions from the initial marking determined by the initial marking of the Petri net generates the behavior.

5 Special Rules

Substitutions have only markings on their left hand side. After a few substitutions have been applied, an expression may become unexpandable without some modification. Also, there is the possibility that more than one substitution is applicable at the same time. To handle these cases, we introduce two special rules that link application of substitutions with manipulation of expressions.

Definition

The escape rule:

If m is not expandable, then $(m \,\|\, tq) \Rightarrow t(m \,\|\, q)$.

The branch rule:

If $\sigma_1, \ldots, \sigma_n$ are all the substitutions that match a given expression x, then we have $x \Rightarrow x_1 \,|\, \ldots \,|\, x_n$ where x_i is derived using from x by applying σ_i. □

The branch rule is where the combinatorial nature of the behavior occurs. For structured systems, however, the idempotency and distributivity laws should keep the complexity manageable.

Example

Let p_1, p_2, p_3 be names of places, and let the set of substitutions be $\{p_1 \,\|\, p_2 \rightarrow t_1, p_2 \,\|\, p_3 \rightarrow t_2\}$, then $p_1 \,\|\, p_2 \,\|\, p_3 \Rightarrow t_1 \,\|\, p_3 \mid t_2 \,\|\, p_1$.

Proposition

If p and q are not expandable, then $(ap \,\|\, bq) \Rightarrow (a \,\|\, b)(p \,\|\, q)$.

Proof

$$
\begin{aligned}
(ap \,\|\, bq) &\Rightarrow a(p \,\|\, bq) \mid b(ap \,\|\, q) && \text{(peeling)} \\
&\Rightarrow ab(p \,\|\, q) \mid ba(p \,\|\, q) && \text{(escape)} \\
&\Rightarrow (ab \mid ba)(p \,\|\, q) && \text{(distributivity)} \\
&\Rightarrow (a \,\|\, b)(p \,\|\, q) && \text{(peeling)}
\end{aligned}
$$

□

The proposition indicates when two concurrent events may be factored to the front of an expression. This is useful for rewriting expressions so that there are markings that have matching substitutions.

6 Worked Examples

In this section we illustrate Concurrency Algebra by looking at couple of examples in detail.

Example 1

Consider the Petri net of Figure A with the following structure intended to model mutual exclusion:

(1) $m \| p_1 \rightarrow t_1 p_2$

(2) $p_2 \rightarrow t_2(m \| p_1)$

(3) $m \| p_3 \rightarrow t_3 p_4$

(4) $p_4 \rightarrow t_4(m \| p_3)$.

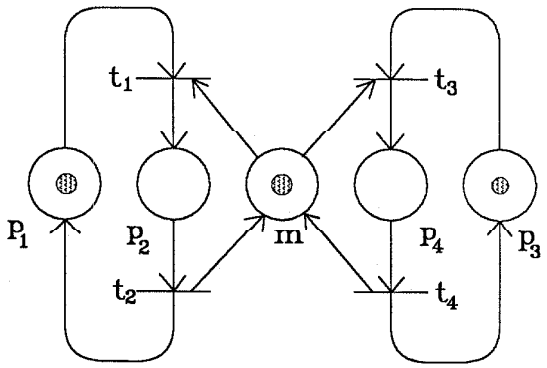

Figure A

Where t_1 and t_2 indicate entry and exit respectively from the critical section for process 1, t_3 and t_4 are the same for process 2, and m is the semaphore controlling access. Then, with initial marking $p_1 \| m \| p_3$, we have the following derivation of the behavior:

$$
\begin{aligned}
p_1 \| m \| p_3 &\Rightarrow t_1 p_2 \| p_3 \mid t_3 p_4 \| p_1 && \text{(branching) (1) (2)} \\
&\Rightarrow t_1 t_2 (p_1 \| m) \| p_3 \mid t_3 t_4 (p_3 \| m) \| p_1 && \text{(2) (4)} \\
&\Rightarrow t_1 t_2 (p_1 \| m \| p_3) \mid t_3 t_4 (p_3 \| m \| p_1) && \text{(escape on } p_3, p_1) \\
&\Rightarrow (t_1 t_2 \mid t_3 t_4)(p_1 \| m \| p_3) && \text{(distributivity)} \\
&\Rightarrow (t_1 t_2 \mid t_3 t_4)(t_1 t_2 \mid t_3 t_4)(p_1 \| m \| p_3) \\
&\Rightarrow \ldots
\end{aligned}
$$

This shows that we have mutual exclusion between the critical sections $t_1 t_2$ and $t_3 t_4$, and the system is live since the strings generated are infinite.

Example 2

Consider a Petri net with the following structure:

(1) $p_1 \rightarrow t_2(p_2 \| p_3)$

(2) $p_2 \| p_3 \rightarrow t_3 p_4$

(3) $p_4 \rightarrow t_4 \mid t_1 p_1$.

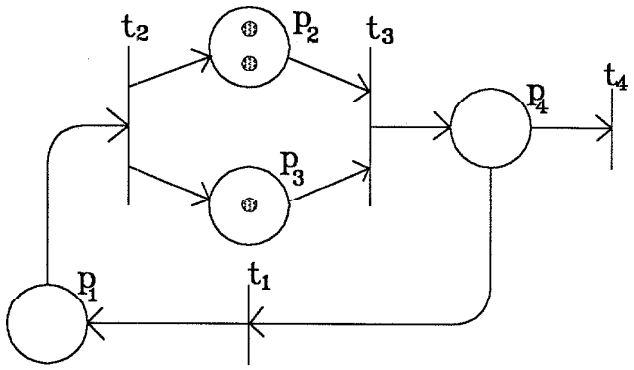

Figure B

With $p_2 \| p_2 \| p_3$ as the initial marking, the derivation is:

$$
\begin{aligned}
p_2 \| p_2 \| p_3 &\Rightarrow p_2 \| t_3 p_4 && (2)\\
&\Rightarrow p_2 \| t_3(t_4 \mid t_1 p_1) && (3)\\
&\Rightarrow p_2 \| (t_3 t_4 \mid t_3 t_1 p_1) && \text{(distributivity)}\\
&\Rightarrow p_2 \| t_3 t_4 \mid p_2 \| t_3 t_1 p_1 && \text{(distributivity)}\\
&\Rightarrow t_3 t_4 p_2 \mid t_3 t_1(p_1 \| p_2) && \text{(escape)}\\
&\Rightarrow t_3 t_4 p_2 \mid t_3 t_1(t_2(p_2 \| p_3) \| p_2) && (1)\\
&\Rightarrow t_3 t_4 p_2 \mid t_3 t_1 t_2(p_2 \| p_3 \| p_2) && \text{(escape)}\\
&\Rightarrow t_3 t_4 p_2 \mid t_3 t_1 t_2 t_3 t_4 p_2 \mid t_3 t_1 t_2 t_3 t_4(p_2 \| p_3 \| p_2)\\
&\Rightarrow \ldots
\end{aligned}
$$

So the set of strings generated includes $t_3 t_4$ and $t_3 t_1 t_2 t_3 t_4$.

7 Some Theoretical Issues

In the previous examples the derivation sequences do not terminate as the derived expressions contain the original expressions again. We have a sort of recursive definition that generates infinite strings. In general, if x is an expression and τ any function on expressions that is formed using choice, shuffle and concatenation, we have $X \Rightarrow \tau[X]$, and what we mean by them is the

smallest set of strings X that is closed under the right hand side, i.e. the smallest set X such that $X = r[X]$.

From the previous examples, we have

$$X \Rightarrow (t_1 t_2 \,|\, t_2 t_1) X \quad \text{and} \quad X \Rightarrow t_3 t_4 p_2 \,|\, t_3 t_1 t_2 X$$

respectively.

To make all this precise, we need to introduce a notion of finite strings both partial and fully defined with an ordering between them, and then extend the ordering to sets of strings. Once the domain of sets of strings is given, the equation $x = r[x]$ can be solved for the least fixed point. This fixed point becomes the meaning of derivations that are recursive. For an example of constructing a domain of infinite strings, see Choo [85].

Another theoretical issue is whether the sets of strings generated as the behavior of a Petri net is complete, i.e. whether all possible behavior are included in the derivation. For finite behaviors, we can show by induction on the length of the behavior that it is complete. For non-terminating Petri nets, the fair firing rule for Petri nets creates subtle issues related to infinite fair shuffle that makes a proof difficult. For infinite behavior, we need to have an independent definition of the shuffle, since the peeling rule is not sufficient to characterize it.

8 Classification of Petri Nets

By using the set of substitutions to represent the structure of Petri nets, we have a natural way of characterizing subclasses by considering the complexity of the expressions that occur on the left and right hand sides of the substitutions.

Definition of Syntactic Subclasses

(a) No choice (|) occurs on the right hand side: **Marked Graphs.**
(b) No shuffle (||) occurs on either side: **State Machines.**
(c) If the left hand side contains the shuffle (||), then no choice (|) occurs on the right hand side, and each place occurs in at most one left hand side: **Free Choice Nets.**
(d) Each place occurs in at most one left hand side: **Multiple Free Choice Nets.** □

The Multiple Free Choice Nets are a natural generalization of Free Choice nets that are no more complex than Free Choice Nets to analyze.

9 Conclusion

The main difficulty in reasoning about concurrent systems is the combinatorial explosion that occurs when several sequential processes are merged together. In Concurrency Algebra we try to manage the complexity by representing sets of strings or behaviors as expressions in our algebra. Then the global behavior can be determined as the solution to a recursion equation. For live and safe systems, this is quite sufficient to characterize the full behavior since there are only a finite number of states.

I would like to thank Prof. James Kajiya for the discussions that led to these ideas.

References

Young-il Choo,

[82] Hierarchical Nets: A Structured Petri Net Approach to Concurrency, California Institute of Technology Computer Science Technical Report 5044:TR:82, 1982.

[85] An Inverse Limit Construction of a Domain of Infinite Lists, California Institute of Technology Computer Science Technical Report 5188:TR:85, 1985.

S. Crespi-Reghizzi and D. Mandrioli,

[76] Some algebraic properties of Petri nets, *Alta Frequenza*, N. 2 vol XLV, 1976.

Seymore Ginsburg,

[66] *Mathematical Theory of Context Free Languages*, McGraw-Hill, New York, 1966.

W.F. Ogden, W.E. Riddle, and W.C. Rounds,

[78] Complexity of expressions allowing concurrency, *Fifth Annual Principles of Programming Languages*, pp. 185–194, 1978.

James L. Peterson,

[81] *Petri net theory and the modeling of systems*, Prentice-Hall, Englewood Cliffs, New Jersey, 1981.