Algebraic Models for Concurrent Programs and Bunched Implication Algebras

Peter Jipsen and M. Andrew Moshier

School of Computational Sciences and
Center of Excellence in Computation, Algebra and Topology (CECAT)
Chapman University, Orange, California

Jan 6, 2015

Outline

- **Introduction**
- **Series-parallel bimonoids** and N-free **pomsets**
- **Deterministic flowcharts** and **concurrent flowcharts**
- **Trace semantics** for **concurrent** programs
- **Generalized bunched implication algebras**

Introduction

Most computers today contain 4 or more processors

Most software is still written to run on only one of them

All supercomputers have many thousands of processors

What is a good approach to designing concurrent programs?

For sequential programs there are good abstract models

E.g. flowcharts, automata, coalgebras, trace models

But what does it mean to run programs in parallel: $P||Q$?

When is this allowed? What happens when they halt?

Bimonoids

A ***series-parallel bimonoid*** is an algebra $(A, \cdot, ||, \varepsilon)$ such that

$\cdot$ and $||$ are **associative binary operations** that have

the **same identity element** ε

and $||$ is **commutative**

Let Σ be a **set** (of **generators**)

$(\Sigma^*, \cdot, \varepsilon)$ is the **free monoid** over Σ

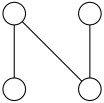
E.g. $\{p, q\}^* = \{\varepsilon, p, q, pp, pq, qp, qq, ppp, ppq, pqp, \ldots\}$

Free series-parallel bimonoids

Σ^{sp} is the **free series-parallel bimonoid**

The elements of Σ^{sp} are represented by finite N-free **po-multisets**

= finite posets labelled by elements of Σ such that there is

no induced subposet isomorphic to the 4-element poset 

E.g. $\{p, q\}^{sp} =$

$\{\varepsilon, p, q, p||p, p||q, q||q, pp, pq, qp, qq, p||pp, p||pq, \ldots, p||p||p, \ldots\}$

Note that $p||q = q||p$, and $\cdot$ has priority over $||$

Partially ordered multisets (pomsets)

Pomsets were introduced by Vaughn Pratt [1986]

to model concurrency

They give a **normal form** for sp-strings:

Two sp-strings are **equal** in Σ^{sp} iff they produce **isomorphic** pomsets

$$pp = \begin{tikzcd} p \arrow[d, dash] \\ p \end{tikzcd} \qquad p||pq = \begin{tikzcd} & p \arrow[d, dash] \\ p & q \end{tikzcd} \qquad (p||p)q = \begin{tikzcd} p \arrow[dr, dash] & & p \arrow[dl, dash] \\ & q & \end{tikzcd}$$

$||$ is **disjoint union** (position pomsets next to each other)

$\cdot$ is **ordinal sum** (position first pomset above the second)

Strings and series-parallel strings

The elements of Σ^* are **strings**

Thought of as traces of actions (or states or both)

Model the execution path of a **sequential** program

Concatenation is **sequential** composition

The elements of Σ^{sp} are **series-parallel strings**

Thought of as (possibly) **concurrent** actions/states

Model the execution path of a **concurrent** program

Horizontal union is **parallel** composition

Flowcharts

To make this concrete, consider a simple flowchart language

Flowcharts are defined over a standard first-order signature

Function symbols $f, g, \ldots$ and predicate symbols $P, Q, \ldots$, each with a fixed arity

Terms are built from variables $\{x_i, y_i, z_i : i = 1, 2, \ldots\}$ and function symbols

An expression $t(\bar{x}, \bar{y})$ indicates that the term t uses (some) variables from the sequences $\bar{x} = x_1, \ldots, x_m$ and $\bar{y} = y_1, \ldots, y_n$

Flowcharts

A deterministic flowchart is a finite directed graph with nodes labeled by statements

A **start statement** with one outgoing edge

assignment statements $\bar{y} := \bar{t}(\bar{x}, \bar{y})$ with one outgoing edge

test statements $P(\bar{t}(\bar{y}))$ with two outgoing edges labeled T and F

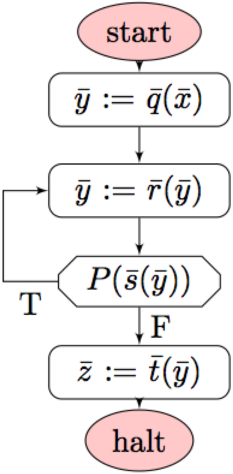
halt statements with no outgoing edges

A start statement has **no** incoming edges

All others have a **finite non-zero** number of incoming edges

$y_1, \dots, y_n := t_1, \dots, t_n$ means the terms t_i on the right are first all evaluated and then assigned to their corresponding variable on the left

Flowchart schemes



start statement

initialization statement

assignment statement

test statement

finalization statement

halt statement

Figure 1: A flowchart scheme showing the different types of statements

Flowcharts

A **deterministic** flowchart computes a (partial) function that maps

values of the **input variables** $\bar{x} = x_1, \ldots, x_m$ to

values of the **output variables** $\bar{z} = z_1, \ldots, z_n$

but the algorithm may **not halt**

Other variables like $y_1, y_2, y_3, \ldots$ are called **work variables**

Concurrent flowcharts

*A **concurrent deterministic flowchart*** is defined with two more statement types: *fork* and *join*

Each **fork statement** has $k > 1$ outgoing edges followed directly by initialization statements $\overline{y_i} := \overline{r_i}(\bar{x}, \bar{y})$ for $i = 1, \ldots, k$

Here $\overline{y_i} = y_{i1}, \ldots, y_{in_i}$ is a sequence of **work variables** distinct from all other variables

When a fork is processed, the **current process is suspended**

the **initialization statements** of the k new processes are evaluated

and then these processes continue **concurrently**

The work variables of the suspended process can be accessed by the new processes

but this can lead to **race conditions** where two concurrent processes modify/read the same variable, resulting in potential nondeterminism

A simple concurrent algorithm

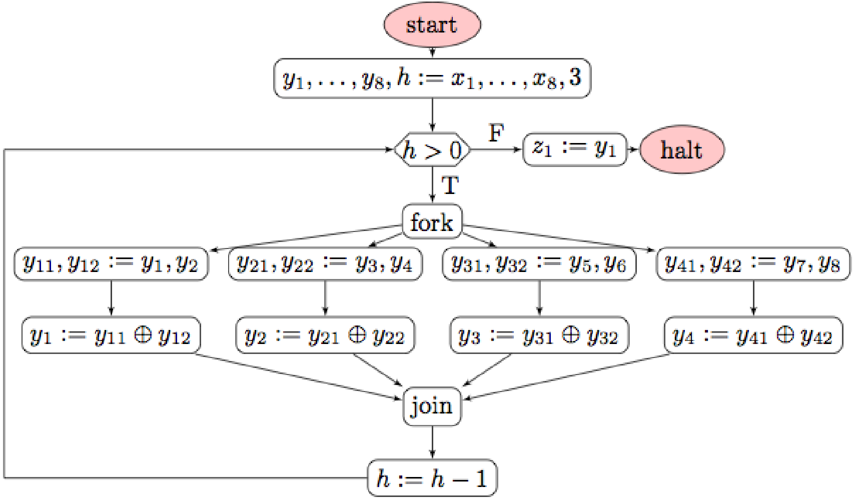


Figure 2: A concurrent scheme for calculating $x_1 \oplus x_2 \oplus \cdots \oplus x_8$

Implementing a parallel for-loop

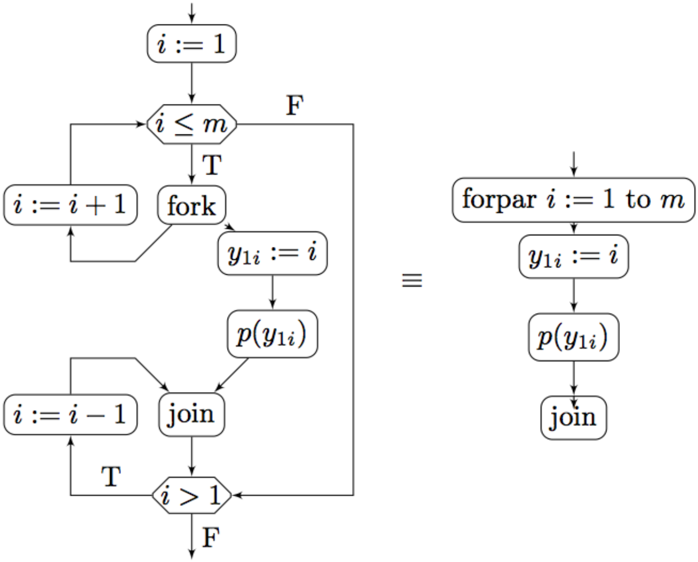


Figure 3: Implementing **forpar** using binary fork and join

Simple algorithm redone using forpar

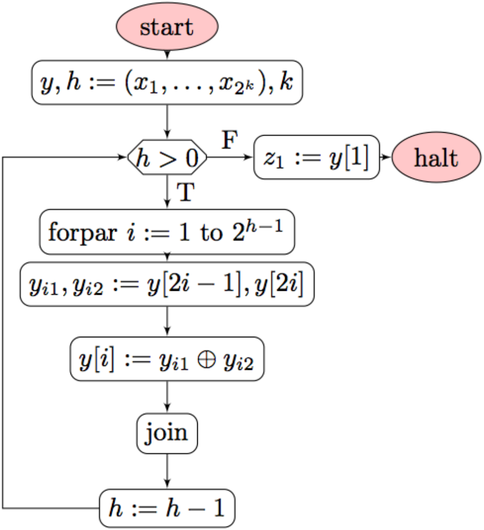


Figure 4: Using **forpar** to calculate $x_1 \oplus x_2 \oplus \cdots \oplus x_{2^k}$

From flowcharts to guarded automata

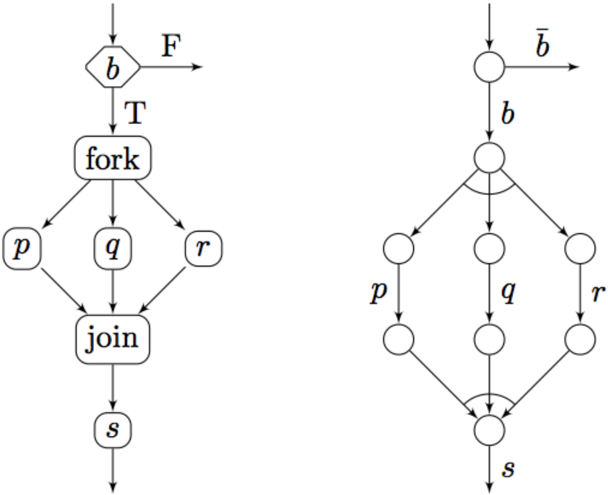


Figure 5: Correspondence between flowcharts and automata

From guarded automata to algebra

Now we define an algebra that provides **trace semantics** for **concurrent** flowcharts

Let $N = \{x_i, y_i, z_i : i = 1, 2, 3, \ldots\}$ be a set of **variables**

Let V be a set of **values** (e. g. $V = \mathbb{Z}$)

The set of **states** is $X = \bigcup\{V^D : D \subseteq N \text{ and } D \text{ is finite }\}$

A state $s \in X$ specifies the values for a finite set $D = \mathrm{dom}(s)$ of variables

States r, s are **separated** if $\mathrm{dom}(r) \cap \mathrm{dom}(s) = \emptyset$, denoted $r \perp s$

X^{sp} is the set of all sp-strings over the set X

Series-parallel traces

An sp-string is called an ***sp-trace*** if

1. its underlying poset has a **largest and a smallest element**, and these two elements have the **same domain**,
2. any two **incomparable** states are **separated**,
3. if $s_1, s_2, \ldots, s_k$ are all the **covers** or all the **lower covers** of state r then $\mathrm{dom}(r) = \mathrm{dom}(s_1) \cup \cdots \cup \mathrm{dom}(s_k)$, and
4. if r is the **only cover** of s, and s is the **only lower cover** of r then $\mathrm{dom}(r) = \mathrm{dom}(s)$

It follows from 1. that the poset of an sp-trace is a **planar lattice**

A sequence $(s_1, \ldots, s_n)$ of terms is also written simply as $s_1 s_2 \ldots s_n$

So an sp-trace is of the form s or svs' where $s, s' \in X$ and $v \in X^{sp}$

Trace semantics

Let p be a (concurrent) flowchart

The ***trace semantics*** of p is the set $[p]$ of all **sp-traces** that are finite execution traces of the flowchart

$[p]$ can be **calculated** in the following way:

For an **assignment** such as $y := t(x_1, \ldots, x_n)$, the semantics are

$$[y := t(\bar{x})] = \{(s, s') \in X^2 : \bar{x} \in \mathrm{dom}(s) = \mathrm{dom}(s') \text{ and } s' = s[y \mapsto t(s(x_1), \ldots, s(x_n))]\}$$

For a **test** $P(y_1, \ldots, y_n)$, the semantics are a set of **length-one sequences**

$$[P(\bar{y})] = \{(s) \in X^1 : y_1, \ldots, y_n \in \mathrm{dom}(s) \text{ and } P(s(y_1), \ldots, s(y_n))\}$$

Sequential and concurrent composition of sp-traces

Sequential composition of sp-traces uses the **coalesced product** $\diamond$

$$rur' \diamond svs' = \begin{cases} rusvs' & \text{if } r' = s \\ \text{undefined} & \text{otherwise} \end{cases}$$

The concurrent composition is based on a ***separated*** **product**:

$$rur' \mid svs' = \begin{cases} (r \cup s)(u||v)(r' \cup s') & \text{if } r \perp s \\ \text{undefined} & \text{otherwise} \end{cases}$$

Note that here $||$ is the parallel composition of sp-strings

$r \diamond s = r$ if $r = s$, else undefined

$rr' \mid svs' = (r \cup s)v(r' \cup s')$ if $r \perp s$

$r \mid svs' = rr \mid svs'$ and $r|s = r \cup s$ if $r \perp s$

The associativity and commutativity of the operation $|$ is easily checked.

Extending to sets of sp-traces

Let X^{spt} be the set of all sp-traces

The semantics of a program (flowchart) p is a **set** of sp-traces

So extend the above two operations to subsets by

- $R \cdot S = \{v \diamond w : v \in R, w \in S \text{ and } v \diamond w \text{ is defined}\}$
- $R||S = \{v \mid w : v \in R, w \in S \text{ and } v \mid w \text{ is defined}\}$
- $R + S = R \cup S$
- $0 = \emptyset, \quad 1 = X^1, \quad \overline{B} = X^1 \setminus B$ and
- $R^* = \bigcup_{n<\omega} R^n$

Mapping from sp-traces to input/output pairs

Theorem: The algebra $\mathbf{A}_{N,V} = (\mathcal{P}(X^{spt}), +, \cdot, ||, 0, 1, ^*, \bar{\ })$ is a **concurrent Kleene algebra with tests** (CKAT)

Each subset R of X^{spt} determines a **binary relation**

$R' = \{(s, s') \in X^2 : svs' \in R$ for some $v \in X^{sp}$ or $s = s' \in R\}$

The map $R \mapsto R'$ is a homomorphism from a CKAT to an algebra of **input/output relations**

Start with a **sequential program** and modify it to run **concurrently** on a **multicore processor** or a **distributed system**

This homomorphism is useful for checking that the **concurrent version** and the **sequential version** still satisfy **the same input/output relation**.

A subalgebra of concurrent composition

The algebra $\mathbf{B}_{N,V} = (\mathcal{P}(X), +, \cdot, ||, 0, 1, \bar{\ })$ is a Boolean subalgebra of $\mathbf{A}_{N,V}$ with a commutative associative operator

So it is an associative r-algebra in the terminology of Jónsson-Tsinakis [1993]

A **generalized effect algebra** is a partial commutative cancellative monoid $(M, |, e)$ such that $x|y = e$ implies $x = y = e$

$(X, |, \emptyset)$ is a generalized effect algebra (since $s|t = s \cup t$ if $s \perp t$)

$\mathbf{B}_{N,V}$ is the **complex algebra** of $(X, |)$

Problem: axiomatize the variety generated by $\{\mathbf{B_{N,V}} : N, V$ are sets$\}$

Generalized bunched implication algebras

A **generalized bunched implication algebra** (GBI-algebra) is of the form $(A, \vee, \wedge, \to, \top, \bot, \cdot, \backslash, /, 1)$ where

$(A, \vee, \wedge, \to, \top, \bot)$ is a **Heyting algebra** (i.e. a bounded distributive lattice with $x \wedge y \leq z$ iff $y \leq x \to z$) and

$(A, \vee, \wedge, \cdot, \backslash, /, 1)$ is a **residuated lattice** (i.e. a monoid with $x \cdot y \leq z$ iff $y \leq x \backslash z$ iff $x \leq z/y$)

If $\cdot$ is **commutative** we get **BI-algebras**

If $(x \to \bot) \to \bot = x$ we get **classical** GBI-algebras

CGBI-algebras = residuated Boolean monoids = *rm*-algebras of Jónsson-Tsinakis

BI-algebras come from **Separation Logic**, a **Hoare programming logic** for reasoning about pointers and concurrent resources

A bunched implication algebra of relations

Relation algebras are **C**GBI-algebras

What are **natural examples** of GBI-algebras?

Let $(X, \sqsubseteq)$ be a poset (or preorder) and define

$Rel(X, \sqsubseteq) = \{R \subseteq X^2 : \sqsubseteq \circ R \circ \sqsubseteq = R\}$

R is called a **weakening relation** if $\sqsubseteq \circ R \circ \sqsubseteq = R$

$Rel(X, \sqsubseteq)$ is a **complete** $\bigcup, \bigcap$-sublattice of $\mathcal{P}(X^2)$ **closed** under $\circ$

$\sqsubseteq, \emptyset, X^2 \in Rel(X, \sqsubseteq)$ and $\sqsubseteq$ is an identity for $\circ$

By completeness, $\cap$ and $\circ$ are **residuated**

Hence $Rel(X, \sqsubseteq)$ is a **GBI-algebra**

Computing with weakening relations

Lemma: $\sqsubseteq \circ R \circ \sqsubseteq = R$ iff $\sqsupseteq \circ \neg R \circ \sqsupseteq = \neg R$

Proof. Assume $\sqsubseteq \circ R \circ \sqsubseteq = R$ and $(x, y) \in \sqsupseteq \circ \neg R \circ \sqsupseteq$

There exist u, v such that $x \sqsupseteq u$, $(u, v) \notin R$ and $v \sqsupseteq y$

If $(x, y) \in R$ then $u \sqsubseteq x$ and $y \sqsubseteq v$ imply $(u, v) \in R$, contradiction

Hence $(x, y) \in R$ and therefore $\sqsupseteq \circ \neg R \circ \sqsupseteq = \neg R$. $\square$

With this result it is easy to prove the following formulas

$R \to S = \neg(\sqsupseteq \circ (R \cap \neg S) \circ \sqsupseteq)$,

$R \backslash S = \neg(\sqsupseteq \circ (R^{\smile} \circ \neg S) \circ \sqsupseteq)$ and $R/S = \neg(\sqsupseteq \circ (\neg R \circ S^{\smile}) \circ \sqsupseteq)$

Kripke frame for $Rel(X, \sqsubseteq)$

Theorem: Let $X^\partial = (X, \sqsupseteq)$. Then the Kripke frame for $Rel(X, \sqsubseteq)$ is $X^\partial \times X$ with ternary accessibility relation

$\circ((u, v), (x, y), (z, w))$ iff $y = z$, $u = x$, and $v = y$.

Hence $Rel(X, \sqsubseteq) = \mathrm{Up}(X^\partial \times X)$

In **contrast** to abstract relation algebras we have:

Theorem [Galatos - J.]: The equational theory of GBI-algebras is decidable

The proof uses **distributive residuated frames** and shows that there is a **cut-free Gentzen system** for GBI

Representable GBI-algebras are not finitely based

Let $\sim x = x \backslash 0$, where 0 is a constant.

A GBI-algebra is **cyclic involutive** if $\sim x = 0/x$ and $\sim\sim x = x$

Lemma: $Rel(X, \sqsubseteq)$ is **cyclic involutive** if we define $0 = \neg \sqsupseteq$

Proof. $\sim R = R \backslash 0 = \neg(\sqsupseteq \circ (R^{\smile} \circ \sqsupseteq) \circ \sqsupseteq) = \neg(\sqsubseteq \circ R \circ \sqsubseteq)^{\smile} = \neg R^{\smile}$

Hence $\sim\sim R = R$ and $0/R = \sim R$

Representable GBI-algs = $\mathrm{Var}\{Rel(X, \sqsubseteq) : (X, \sqsubseteq) \text{ is a poset}\}$

Theorem: The variety of representable GBI-algebras is not finitely based.

Proof. Adding the axiom $(x \to \bot) \to \bot = x$ defines the nonfinitely based variety of representable relation algebras.

Thank you