Locality as product

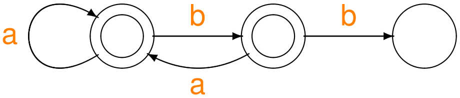
Kamal Lodaya and R. Ramanujam

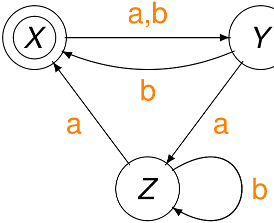
The Institute of Mathematical Sciences, Chennai

September 2011

Transition systems and automata

- Mathematical model.
- Widely used to study simple sequential programs.
- Computer scientists like them.
- Others —for example, people in the software industry— don't. (cf. Moshe Vardi's talk in ICLA '09 at Chennai.)





Rational expressions

Theorem (Kleene)

There is a syntax of rational expressions from which one can construct equivalent nondeterministic finite automata of polynomial size. Conversely from an automaton one can construct in a polynomial number of steps an expression which in the worst case can be of exponential size.

- Let A, a nonempty finite alphabet, consist of all atomic computations. The regular expressions are:

$$r ::= a \in A \mid r_1 ; r_2 \mid r_1 + r_2 \mid r_1^*$$

- The expressions define the following languages (sets of computations):

$$\begin{array}{rcl} Lang(a) & = & \{a\} \\ Lang(r_1 ; r_2) & = & \{w_1 w_2 \mid w_1 \in Lang(r_1), w_2 \in Lang(r_2)\} \\ Lang(r_1 + r_2) & = & Lang(r_1) \cup Lang(r_2) \\ Lang(r_1^*) & = & \{w_1 \dots w_n \mid \exists n, \forall 1 \leq i \leq n, w_i \in Lang(r_1)\} \end{array}$$

Language equivalence for rational expressions (Aanderaa 1965, Salomaa 1966)

(Monoid)	$(e+f)+g = e+(f+g)$; $e+0 = e$
(Comm)	$e+f = f+e$
(Idem)	$e+e = e$
(Monoid)	$(ef)g = e(fg)$; $e1 = 1e = e$
(Absorp)	$e0 = 0e = 0$
(Distr)	$(e+f)g = eg+fg$; $e(f+g) = ef+eg$
(Guard)	$e^* = (1+e)^*$
(Fixpt)	$e^* = 1+ee^*$; $e^* = 1+e^*e$
(GuardInd)	Let e have the NEWP. Then: $\dfrac{x = ex+f}{x = e^*f}$; $\dfrac{x = xe+f}{x = fe^*}$

Theorem (Salomaa; Meyer, Stockmeyer)

This axiomatization is sound and complete for language equivalence of rational expressions. Checking equivalence is complete for polynomial space.

Temporal logic

$\Phi ::= p, p \in AP \mid \neg\alpha \mid \alpha \vee \beta \mid \langle a\rangle\alpha \mid \alpha \mathbf{U} \beta$

$\Diamond\alpha \stackrel{\text{def}}{=} True\mathbf{U}\alpha$; $\Box\alpha \stackrel{\text{def}}{=} \neg\Diamond\neg\alpha$; $\bigcirc\alpha \stackrel{\text{def}}{=} \bigvee_{a\in\Sigma} \langle a\rangle\alpha$;

$\odot\alpha \stackrel{\text{def}}{=} \neg\bigcirc\neg\alpha$; $[a]\alpha \stackrel{\text{def}}{=} \neg\langle a\rangle\neg\alpha$.

Definition

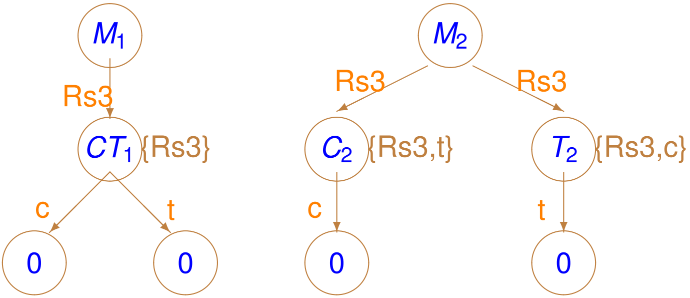
FRAME $F = (\mathcal{T}, \delta)$, where δ is a run (usually infinite) on the transition system $\mathcal{T}$.

MODEL $M = (F, V)$, $V : Q \to \wp(AP)$ the valuation function over Q, the states

- $M, k \models p$ iff $p \in V(\delta(k))$.
- $M, k \models \neg\alpha$ iff $M, k \not\models \alpha$.
- $M, k \models \alpha \vee \beta$ iff $M, k \models \alpha$ or $M, k \models \beta$.
- $M, k \models \langle a\rangle\alpha$ iff $\delta(k+1)$ exists, $\delta(k) \xrightarrow{a} \delta(k+1)$ and $M, k+1 \models \alpha$.
- $M, k \models \alpha\mathbf{U}\beta$ iff for some $m \geq k$ such that $M, m \models \beta$, and for all $l : k \leq l < m, M, l \models \alpha$.

Temporal logic

ϕ is SATISFIABLE if $M, 0 \models \phi$ for some model $M = ((\mathcal{T}, \delta), V)$.
ϕ is VALID if ϕ is satisfied in every model M.



$\langle Rs3\rangle(\langle c\rangle\alpha \wedge \langle t\rangle\beta), \langle Rs3\rangle\langle c\rangle\alpha \wedge \langle Rs3\rangle\langle t\rangle\beta$

Axiomatization

(A0)	All the substitutional instances of the tautologies of PC
(A1)	$[a](\alpha \implies \beta) \implies ([a]\alpha \implies [a]\beta)$
(A2)	$\langle a\rangle True \implies [b]False, \quad a \neq b$
(A3)	$\langle a\rangle\alpha \implies [a]\alpha$
(A4)	$\alpha\mathbf{U}\beta \implies (\beta \vee (\alpha \wedge \odot(\alpha\mathbf{U}\beta)))$
(MP)	$\dfrac{\alpha, \quad \alpha \implies \beta}{\beta}$
(TG)	$\dfrac{\alpha}{[a]\alpha}$

Theorem (Gabbay, Pnueli, Shelah, Stavi; Sistla, Clarke)

This axiomatization of temporal logic is sound and complete for infinite runs of a transition system. Satisfiability and validity are complete for polynomial space.

A different syntax

We also use right-linear grammars (or tail-recursive equations) to describe finite state systems.

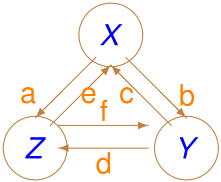
$x = az + by$

$y = cx + dz$

$z = ex + fy$

An equivalent program:

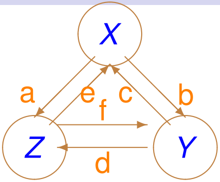
x: choose a; jump y
choose b; jump z; exit
y: choose c; jump x
choose d; jump z; exit
z: choose e; jump x
choose f; jump z; exit



Solving equations in rational expressions

$W = aX + bZ,\ X = aY + bW,\ Y = aZ + bX,\ Z = aZ + bZ$.
By right-distributivity and introducing star, $Z = (a+b)^*$.
Substituting:
$W = aX + b(a+b)^*,\ X = bW + a(a(a+b)^* + bX)$.
By left-distributivity and introduction of star:
$X = abX + bW + aa(a+b)^* = (ab)^*(bW + aa(a+b)^*)$.
Applying the same medicine again:
$W = a(ab)^*bW + a(ab)^*aa(a+b)^* + b(a+b)^*$ and
$W = (a(ab)^*b)^*(a(ab)^*aa(a+b)^* + b(a+b)^*)$.
This way of finding solutions is reminiscent of performing Gaussian elimination in linear arithmetic equations and was first used for regular languages by McNaughton and Yamada.

By Kleene's theorem . . .



Here is a rational expression:
$x = (a(fd)^*(e + fc) + b(df)^*(c + de))^*$

Or in program notation:

x: loop choose a;
loop f; d end loop ;
choose e; or f; c end choose
end choose ;
choose b;
loop d; f end loop ;
choose c; or d; e end choose
end choose
end loop

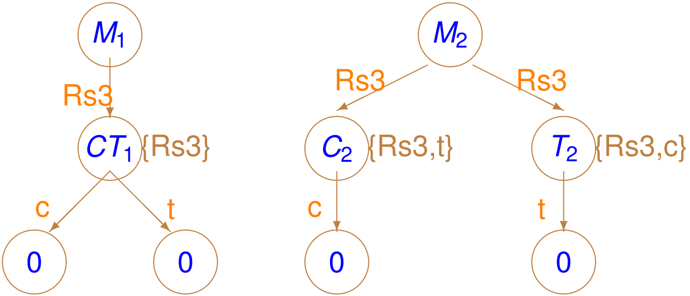
Structured programming

Theorem (Böhm and Jacopini)

Every flowchart program can be converted into an equivalent program using only assignments, sequencing, choice (if-then-else) and iteration (while-do) commands.

But iteration is not as powerful as tail-recursion for reactive behaviour, which we see in the next slide.

Reactive behaviour



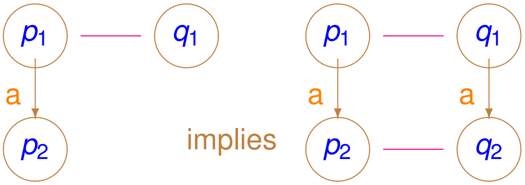
$Rs3 \cdot (c + t) \neq (Rs3 \cdot c) + (Rs3 \cdot t)$ Left-distributivity fails

$c \cdot 0 \neq 0$ Right-absorption fails

Definition (Brookes, Hoare and Roscoe)

Two machines are failure equivalent if one of them can perform a sequence of actions and then refuse to perform an action, so can the other.

Bisimulation



Definition (Park)

A bisimulation is a symmetric relation between the states of two transition graphs such that if p_1 is bisimilar to q_1 and p_1 can make an a-move to p_2, then there is a q_2 bisimilar to p_2 such that q_1 can make an a-move to q_2.

This is a recursive version of the definition of failure equivalence in the previous slide.

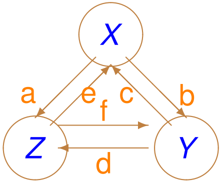
Right-linear grammars

$r ::= 0 \mid X \mid a \cdot r_1 \mid r_1 + r_2 \mid \mu X.r_1$

Theorem (Milner)

1. *For all finite state machines there is a linear size right-linear grammar which describes their behaviour upto bisimulation.*
2. *The behaviour upto bisimulation of the machine below cannot be described by a rational expression.*

$$X = aZ + bY$$
$$Y = cX + dZ$$
$$Z = eX + fY$$



Park bisimulation for finite systems (Milner 1984)

(Monoid)	$(e+f)+g = e+(f+g)$; $e+0 = e$
(Comm)	$e+f = f+e$
(Idem)	$e+e = e$
(Assoc)	$(ef)g = e(fg)$
(LeftAbs)	$0e = 0$
(RightDistr)	$(e+f)g = eg+fg$
(Guard)	$\mu X.e = \mu X.(X+e)$
(Fixpt)	$\mu X.e = e[\mu X.e/X]$
(GuardInd)	$\dfrac{f = e[f/X]}{f = \mu X.e}$ (provided X guarded in e)

Theorem (Milner; Kanellakis, Smolka)

This axiomatization is sound and complete for bisimulation of mu-expressions. Bisimulation can be checked in polynomial time.

Extensions of rational expressions

$$r ::= a \in A \mid r_1 ; r_2 \mid r_1 + r_2 \mid r_1^* \mid r_1 \cap r_2 \mid \overline{r_1}$$

$$\begin{aligned} Lang(r_1 \cap r_2) &= Lang(r_1) \cap Lang(r_2) \\ Lang(\overline{r_1}) &= \overline{Lang(r_1)} \end{aligned}$$

- For an automaton for the expression $r_1 \cap r_2$ we inductively assume automata M_1 for r_1 and M_2 for r_2 and perform a product construction. Its size will be $O(|M_1|) \times O(|M_2|)$. If there are many intersections, the size of the automaton constructed can be exponential in $|r_1 \cap r_2|$.
- An automaton for $\overline{r_1}$ requires a subset construction. Its size is exponential in the size of the automaton for r_1. If there are many negations, the size of the automaton constructed can be a tower of exponentials in $|\overline{r_1}|$.

Shuffle with synchronization (Campbell,Habermann 1974)

Concurrent composition of two automata can be thought of as a product on their common actions, and a shuffle of the other letters of the alphabet.

$$r \quad ::= \quad a \in A \mid r_1 ; r_2 \mid r_1 + r_2 \mid {r_1}^* \mid SYNC\ J\ IN(r_1, r_2), J \subseteq A$$

$$\begin{aligned} PAR(r_1, r_2) &= SYNC\ \emptyset\ IN(r_1, r_2) \\ Lang(SYNC\ J\ IN(r_1, r_2)) &= \{w \mid w \text{ is a shuffle of } w_1, w_2 \\ &\quad \text{except that } w_1 \lceil J = w_2 \lceil J; \\ &\quad w_1 \in Lang(r_1), w_2 \in Lang(r_2)\} \end{aligned}$$

Example

When *j* synchronizes *aaja aaja* and *na jaao na jaao*, you might get a word like *aa na jaao naaaa jaaao*.

Counting and state explosion

Example

$SYNC\ j\ IN((j;j)^*,(j;j;j)^*)$.

The first process does a loop of two j's.
The second process does a loop of three j's.
The j's synchronize.

Er . . . which j's ?

Suppose the automata for r_1 and r_2 have n_1 and n_2 states respectively. The automaton construction for $SYNC\ j\ IN(r_1, r_2)$ has complexity $O(n_1 n_2)$.

Theorem (State explosion)

From a syntax of parallel products of rational expressions one can construct equivalent product systems (products of nondeterministic finite automata) of exponential size.

Synchronization on common letters (Mazurkiewicz 1977)

LOCATIONS $Loc = \{1, \ldots, n\}$.

DISTRIBUTED ALPHABET $\Sigma \stackrel{\text{def}}{=} \Sigma_1 \cup \ldots \cup \Sigma_n$, each Σ_i finite nonempty set of actions of agent i. When an action a is in $\Sigma_i \cap \Sigma_j, i \neq j$, we think of it as a synchronization action between i and j. (There can be k-way synchronizations also.)

Let $loc(a) \stackrel{\text{def}}{=} \{i \mid a \in \Sigma_i\}$.

PRODUCT WORD $(w_1, \ldots, w_n) \in (\Sigma^*)^{Loc}$, such that for some $w \in \Sigma^*$, every $w_i = w\lceil\Sigma_i$.

Definition

PRODUCT SYSTEM $TS = (Q, \Rightarrow)$ over Σ, where

- $\widetilde{Q} \stackrel{\text{def}}{=} Q_1 \times \cdots \times Q_n$
- Global transition function $\Rightarrow \subseteq Q \times \Sigma \times Q$: $(q_1, \ldots, q_n) \stackrel{a}{\Rightarrow} (q'_1, \ldots, q'_n)$ iff for all $i \in loc(a), q_i \xrightarrow{a}_i q'_i$ and for all $j \notin loc(a), q_j = q'_j$.

Parallel product of regular expressions (Hoare 1981)

REGULAR EXPRESSIONS WITH PARALLEL

$r ::= 0 \mid 1 \mid a \mid r_1 \cdot r_2 \mid r_1 + r_2 \mid r_1^* \mid r_1 \parallel r_2 \mid r_1 + r_2$

MU-EXPRESSIONS WITH PARALLEL

$r ::= 0 \mid X \mid a \cdot r_1 \mid r_1 + r_2 \mid \mu X.r_1 \mid r_1 \parallel r_2 \mid r_1 + r_2$

(Distr)	$p \parallel (q + r) = (p \parallel q) + (p \parallel r)$
(Expansion)	$ap \parallel bq = a(p \parallel bq) + b(ap \parallel q) + (a \mid b)(p \parallel q)$
(Monoid)	$(e + f) + g = e + (f + g)$; $e + 0 = e$
(Comm)	$e + f = f + e$
(Idem)	$e + e = e$

Theorem (Milner; Bergstra and Klop)

Parallel product can be eliminated over term algebras to yield a sound and complete axiomatization.

This destroys locality.

Reasoning for products of rational expressions

For technical reasons (Zielonka 1989) we need a renaming operation.
By adding a few axioms for renaming (letter-to-letter substitution), we can characterize language and bisimulation behaviour of finite labelled product systems.

(Subst)	$a[\sigma] = \sigma(a)$
(Comp)	$e[\sigma_1][\sigma_2] = e[\sigma_1 \circ \sigma_2]$
(Distr)	$(p+q)[\sigma] = p[\sigma] + q[\sigma]$
(Distr)	$(pq)[\sigma] = p[\sigma]q[\sigma]$

Product temporal logic

$\Phi_i ::= p \mid \neg\alpha \mid \alpha \vee \beta \mid \langle a \rangle_i \alpha,\ a \in \Sigma_i \mid \alpha \mathbf{U}_i \beta$
$\Phi ::= \alpha@i, \alpha \in \Phi_i \mid \neg\phi \mid \phi_1 \vee \phi_2$
Let $\widehat{a} \stackrel{\text{def}}{=} \bigwedge_{i \in loc(a)} (\langle a \rangle_i True)@i$.
We now define the semantics of global formulas.
FRAME $F = (\mathcal{T}, \delta)$, where $\mathcal{T}$ is a product transition system
MODEL $M = (F, V)$, $V : Q \to \wp(AP)$, Q the set of all local states of the system. Thus, atomic propositions are evaluated at local states.

- $M \models \alpha@i$ iff $M_i, 0 \models \alpha$.
- $M \models \neg\phi$ iff $M \not\models \phi$.
- $M \models \phi_1 \vee \phi_2$ iff $M \models \phi_1$ or $M \models \phi_2$.

Product temporal induction

Usual temporal induction for reachability:

$$\frac{GI \implies \alpha \wedge \bigodot(GI)}{GI \implies \Box\alpha}$$

Product generalization is too weak:

$$\frac{\bigwedge_k LI@k \implies \alpha@i \wedge \bigodot_j LI@j}{\bigwedge_k LI@k \implies (\Box_i\alpha)@i}$$

Combination of LIs can specify global states which are not reachable.

Product induction:

$$\frac{\widehat{b} \wedge \bigwedge_k Pre@k \implies ([b]_j Post)@j,\ j \in loc(b)}{\bigwedge_{k \notin loc(b)} Pre@k \wedge \bigwedge_{j \in loc(b)} Post@j \implies GI}$$

Global axiomatization, part 1

(A0) $(\neg\alpha)@i \equiv \neg\alpha@i$

(A1) $(\alpha \vee \beta)@i \equiv (\alpha@i \vee \beta@i)$

(A2) $\bigvee_{a\in\Sigma} \widehat{a}$

(MP) $$\frac{\alpha, \quad \alpha \implies \beta}{\beta}$$

(GG) $$\frac{\vdash_i \alpha}{\alpha@i}$$

(GM) $$\frac{\bigwedge_{i\in loc(a)} \alpha_i@i \implies \bigvee_{j\notin loc(a)} \alpha_j@j}{\bigwedge_{i\in loc(a)} (\langle a\rangle_i \alpha_i)@i \implies \bigvee_{j\notin loc(a)} \alpha_j@j}$$

Global axiomatization, synchronization

Let $m > 0$ and $\alpha_1, \ldots, \alpha_m$ be formulas such that for all $l \in \{1, \ldots, m\}$, α_l is of the form $\bigwedge_{k \in Loc} \alpha_l(k)@k$. Let $\gamma \stackrel{\text{def}}{=} \bigvee_{l=1}^{m} \alpha_l$.

$$
\begin{array}{ll}
(\mathrm{Sy}_m) & \gamma \implies \neg\widehat{a} \\
& \bigwedge_{l \in \{1,\ldots,m\}} (\alpha_l \implies (\bigwedge_{b \notin \Sigma_i} (\widehat{b} \implies \bigwedge_{j \in loc(b)} ([b]_j \beta(l,b,j))@j))) \\
& \bigwedge_{l \in \{1,\ldots,m\}} \bigwedge_{b \notin \Sigma_i} ((\bigwedge_{k \notin loc(b)} \alpha_l(k)@k \wedge \bigwedge_{j \in loc(b)} \beta(l,b,j)@j) \implies \gamma) \\
\hline
& \gamma \implies ([a]_i False)@i, \text{ for } i \in loc(a)
\end{array}
$$

Gloabl axiomatization, the until operator

(Un$_m$)
$$\gamma \implies \neg\gamma_2 @ i$$
$$\bigwedge_{l \in \{1,\ldots,m\}} (\alpha_l \implies (\bigwedge_{b \in \Sigma} (\widehat{b} \implies \bigwedge_{j \in loc(b)} ([b]_j \beta(l,b,j)) @ j)))$$
$$\bigwedge_{l \in \{1,\ldots,m\}} \bigwedge_{b \in \Sigma} ((\bigwedge_{k \notin loc(b)} \alpha_l(k) @ k \wedge \bigwedge_{j \in loc(b)} \beta(l,b,j) @ j) \implies \gamma)$$

$$\gamma \implies \neg(\gamma_1 \mathbf{U}_i \gamma_2) @ i$$

Theorem

The local and global axiomatizations, put together, are sound and complete.

Equivalence of products of rational expressions

Consider $X||Y = (a+ba)^*||(ab)^*$.
By fixpoint expansion:
$X||Y = 1 + a(1 + (a+ba)(a+ba)^*) + ba(a+ba)^*||1 + ab(ab)^*$.
Distributing and eliminating useless actions:
$X||Y = 1 + aba(a+ba)^*||1 + ab(ab)^* = 1 + abaX||1 + abY$
By product induction and fixpoint expansion:
$X||Y = (aba)^*||(ab)^* = 1 + aba(aba)^*||1 + ab(ab)^*$.
Continuing in this way:
$W||Z = a(aba)^*||(ab)^* = ae||bf$, for some e, f.
Eliminating useless actions, $W||Z = 0||0$.
Hence $X||Y = 1||1$.

Questions open

A question from me:
Is there a sound and complete axiomatization of language equivalence of products of rational expressions which does not reduce parallel product to interleaving?