Models for Concurrent Behaviors

January 7, 2002, 17:43

Uwe Nestmann

EPFL-DI-LAMP

Repetition of Algebraic Notions

relation

- binary, ternary, . . .
- function
- partial/total
- injective
- surjective
- converse/inverse

Automata

An **automaton** $A = (Q, q_0, F, T)$
over an **action alphabet** ***Act***:

- a set $Q = \{q_0, q_1 \ldots\}$: the **states**
- a state $q_0 \in Q$: the **start state**
- a subset $F \subseteq Q$: the **accepting states**
- a subset $T \subseteq (Q \times \mathit{Act} \times Q)$: the **transitions**

A transition $(q, \alpha, q') \in T$ is also written $q \xrightarrow{\alpha} q'$.
Transition Graphs are useful . . .

Example Automaton

Let *Act* be $\{a, b, c\}$.
Let A be defined as

$$
\begin{array}{ll}
(& \{q_0, q_1, q_2, q_3\}, q_0, \{q_1\}, \\
& \{ \quad (q_0, b, q_3), (q_0, c, q_3), (q_0, a, q_1), \\
& \qquad (q_1, c, q_0), (q_1, a, q_3), (q_1, b, q_2), \\
& \qquad (q_2, c, q_0), (q_2, a, q_3), (q_2, b, q_3), \\
& \qquad (q_3, c, q_3), (q_3, a, q_3), (q_3, b, q_3), \\
& \} \\
) &
\end{array}
$$

Automata (II)

An automaton A is

- **finite-state**, if Q is finite, and
- **deterministic** if for each pair $(q, \alpha) \in Q \times \mathit{Act}$ there is **exactly one** transition $q \xrightarrow{\alpha} q'$.

Question: Would the formulation "**at most one**" transition yield less deterministic automata?

Note: "Complete" an automaton?

Behavior: Language of an Automaton

Let A be an automaton over *Act*.
Let $s = \alpha_1 \ldots \alpha_n$ be a string over *Act*. Then:

- A is said to **accept** s, if there is a path in A — from q_0 to some accepting state — whose arcs are labeled successively $\alpha_1 \ldots \alpha_n$.
- The **language** of A, denoted by $\widehat{A}$, is the set of strings accepted by A.

ϵ denotes the empty string.

Regular Sets

Definition: A set of strings over *Act* is **regular** if it can be built from

- the **empty set** $\emptyset$ and the **singleton** sets $\{\alpha\}$ (for each $\alpha \in$ *Act*),
- using the operations of **union** ($\cup$), **concatenation** ($\cdot$), and **iteration** (*).

$$\begin{aligned} S_1 \cdot S_2 &\stackrel{\mathrm{def}}{=} \ldots \\ S^* &\stackrel{\mathrm{def}}{=} \ldots \end{aligned}$$

In regular sets, we write α for $\{\alpha\}$ and ϵ for $\{\epsilon\}$.

Regular Expressions

<u>Definition:</u> The set of **regular expressions** over *Act* is generated by the following grammar:

$$e \quad ::= \quad \epsilon \quad | \quad \alpha \quad | \quad e + e \quad | \quad e \cdot e \quad | \quad e^*$$

where $\alpha \in$ *Act*.

In regular expressions, we write $\alpha\beta$ for $\alpha \cdot \beta$ ...

regular expressions	*vs regular sets*
$(a + b)c$	$\{ac, bc\}$
$a + bc$	$\{a, bc\}$

Regular Expressions (II)

$$
\begin{aligned}
(S_1 \cdot S_2) \cdot S_3 &= S_1 \cdot (S_2 \cdot S_3) \\
S \cdot \epsilon &= S \\
S \cdot \emptyset &= \emptyset
\end{aligned}
$$

$$
\begin{aligned}
(S_1 + S_2) \cdot T &= S_1 \cdot T + S_2 \cdot T \\
T \cdot (S_1 + S_2) &= T \cdot S_1 + T \cdot S_2
\end{aligned}
$$

$$
S \cdot (T \cdot S)^* = (S \cdot T)^* \cdot S
$$

Note:
The regular set $\emptyset$ means “no path”. But:
The regular expression ϵ means “empty path”.

$$\emptyset \neq \{\epsilon\}$$

As an example, compare $\{\alpha\beta\} \cdot \{\epsilon\}$ with $\{\alpha\beta\} \cdot \emptyset$.

Arden's rule

Theorem:
For any sets of strings S and T, the equation

$$X = S \cdot X + T \qquad \text{has} \qquad X = S^* \cdot T$$

as a **solution**.
Moreover, this solution is unique if $\epsilon \notin S$.

Fact: The language $\widehat{A}$ of any finite-state automaton A is regular.

Example Automaton

Determine the language of the previous automaton as the regular expression describing the strings accepted in the initial state.

Write down a set of equations,
one equation for each state.

Solve the set of equations . . .

Determinism / Nondeterminism

Analyze two automata . . .
Exercise in § 2.4 of [Mil99]

Message: **Language equivalence is blind for nondeterminism.**

In fact, every nondeterministic automaton can be converted into a determinstic that accepts the same language.

The Zen of Black Boxes

- automata as black boxes
- actions as buttons (for interaction)

“What matters of a string s (a sequence of actions) is not whether it drives the automaton into an accepting state (since we cannot detect this by interaction) but whether the automaton is able to perform the sequence s interactively.” [Mil99]

Attempt: Consider *every state accepting* ...

Prefixes

If a string s can be expressed in the form $s_1 s_2$, then s_1 is a **prefix** of s.

A language S is **prefix-closed** if, whenever $s_1 s_2 \in S$, then also $s_1 \in S$.

The **prefix-closure** of a language S is the *larger* language *Pref*(S) that contains *all the prefixes of every string* in S. *Pref*(S) is the smallest prefix-closed language that includes S.

Tea / Coffee

Reinterpret the previous automata as tea/coffee machines.

Observe the mismatch between black-box-interaction and accepted languages.

Observe the representation of the mismatch in the equations satisfied by regular languages.

Attempt: Consider *any state starting* . . .

Labeled Transition Systems

Definition:
An **LTS** $L = (Q, T)$ over an **action alphabet** ***Act***:

- a set of **states** $Q = \{q_0, q_1 \ldots\}$
- a ternary **transition relation** $T \subseteq (Q \times \mathit{Act} \times Q)$

A transition $(q, \alpha, q') \in T$ is also written $q \xrightarrow{\alpha} q'$.

If $q \xrightarrow{\alpha_1} q_1 \cdots \xrightarrow{\alpha_n} q_n$ we call q_n a **derivative** of q.

Transition Graphs are useful ...

Equivalence on LTS

Recall why we are interested in equivalence relations on our model.

When should two LTS (or two states within an LTS) be considered equivalent?

- language equivalence ?
- isomorphism ?

Try to **interact** with them . . .
. . . and observe any possible difference.

Equivalence on LTS (II)

Example: Compare p_0 and q_0 in

$$\begin{aligned} \{ \quad & (p_0, a, p_1), (p_1, b, p_2), (p_1, c, p_3), \\ & (q_0, a, q_1), (q_0, a, q_1'), (q_1, b, q_2), (q_1', c, q_3) \quad \} \end{aligned}$$

Motivate simulation !

Induce simulation of paths
through step-by-step simulation of actions ...

Strong Simulation on LTS

Definition: **(learn it by heart!)**
Let (Q, T) be an LTS.
Let $\mathcal{S}$ be a binary relation over Q.

Then $\mathcal{S}$ is a **strong simulation** over (Q, T) if
whenever $p\mathcal{S}q$: if $p \xrightarrow{\alpha} p'$
then there is $q' \in Q$ such that $q \xrightarrow{\alpha} q'$ and $p'\mathcal{S}q'$.

q **strongly simulates** p
if there is a strong simulation $\mathcal{S}$ such that $p\mathcal{S}q$.

Working with Simulation

- exhibiting a simulation
- checking a simulation
- “generating” a simulation

What changes if we define simulation between two different LTSs instead of on a single one?

Working with Simulation (II)

Example: Find all non-trivial simulations in . . .