

# CDM

## Automata and Logic

Klaus Sutner  
Carnegie Mellon University

Fall 2009

### Outline

- 1 The Logic Connection
- 2 Rational Relations
- 3 Deciding Rational Structures

### Approaches to Regularity

So far we have two different approaches to regular languages.

- Constant space: finite state machines.
- Kleene algebra: regular expressions.

But there is more:

- Logic: words as structures.

The logic angle leads to some very powerful algorithms that are used e.g. in model checking and formal verification.

### Quoi?

The idea is to think of a word as a structure (like a tree, a matrix, the natural numbers, ...) and to use logic to describe the properties of the structure.

This is a bit unusual, but bear with me. First, we need to fix an appropriate language for our logic. As always, we want at least propositional logic.

$\perp, \top$	constants false, true
$\neg$	not, negation
$\wedge$	and, conjunction
$\vee$	or, disjunction
$\rightarrow$	implication
$\leftrightarrow$	equivalence

### Variables and Atomic Formulae

We will have variables  $x, y, z, \dots$  that range over positions in a word.

We allow the following basic predicates between variables:

$$x < y \quad x = y$$

Of course, we can get, say,  $x \geq y$  by Boolean operations.

We write

$$Q_a(x)$$

for "there is a letter  $a$  in position  $x$ ."

### Higher Order Variables and Quantifiers

We also have second order variables  $X, Y, Z, \dots$  that range over sets of positions in a word. We write  $X(x)$  to indicate that  $x \in X$ .

Sets of positions are all there is; we do not have variables in our language for, say, binary relations on positions. This type of system is often called a monadic logic.

Lastly, we allow quantification for position variables and for sets of positions.

$$\exists x \varphi \quad \forall x \varphi \quad \exists X \varphi \quad \forall X \varphi$$

#### Definition

This system is called **monadic second order logic (with less-than)**,  $\text{MSO}[\<]$ .

## Semantics

We need some notion of satisfaction

$$w \models \varphi$$

where  $w$  is a word and  $\varphi$  a sentence.

We won't give a formal definition, but the basic idea is simple: Let  $|w| = n$ . Then the first order variables range over  $[n] = \{1, 2, \dots, n\}$  and second order variables range over  $\mathfrak{P}([n])$ .

The basic relations  $x < y$  and  $x = y$  are clear. For the  $Q_a(x)$  predicate we have

$$Q_a(x) \iff w_x = a$$

## Examples

$$aaacbbb \models \forall x (Q_a(x) \vee Q_b(x) \vee Q_c(x))$$

$$aaabbb \models \exists x, y (x < y \wedge Q_a(x) \wedge Q_b(y))$$

$$bbbaaa \not\models \exists x, y (x < y \wedge Q_a(x) \wedge Q_b(y))$$

$$aaabbb \models \exists x, y (x < y \wedge \neg \exists z (x < z \wedge z < y) \wedge Q_a(x) \wedge Q_b(y))$$

$$aaacbbb \not\models \exists x, y (x < y \wedge \neg \exists z (x < z \wedge z < y) \wedge Q_a(x) \wedge Q_b(y))$$

$$aaacbbb \models \exists x (Q_c(x) \rightarrow \forall y (x < y \rightarrow Q_b(y)))$$

## All Satisfying Words

## Example

$$\varphi = \exists x \forall y, z ((y \leq x \rightarrow Q_a(y)) \wedge (z > x \rightarrow Q_b(z)))$$

Then  $w \models \varphi$  iff  $w \in a^*b^*$ .

## Example

$$\varphi = \exists x, y (\text{first}(x) \wedge Q_a(x) \wedge \text{last}(y) \wedge Q_b(y))$$

Here  $\text{first}(x)$  is shorthand for  $\forall z (x \leq z)$ , and  $\text{last}(x)$  is shorthand for  $\forall z (x \geq z)$ .

Then  $w \models \varphi$  iff  $w \in a\Sigma^*b$ .

## The Language of a Sentence

The examples suggest that, for any sentence  $\varphi$ , we should consider the collection of all words that satisfy  $\varphi$ :

$$\mathcal{L}(\varphi) = \{w \in \Sigma^* \mid w \models \varphi\}.$$

One cannot fail to notice that in the examples so far  $\mathcal{L}(\varphi)$  is always regular. Of course, this might be coincidence.

## Even/Even

## Example

$$\varphi = \exists X (\forall x (Q_a(x) \leftrightarrow X(x)) \wedge \text{even}(X))$$

Here  $\text{even}(X)$  means that  $X$  has even cardinality. Then  $w \models \varphi$  iff the number of  $a$ 's in  $w$  is even.

This is really cheating, we need to show that even is definable in our setting. This is tedious but not really hard:

$$\text{even}(X) \iff \exists Y, Z (X = Y \cup Z \wedge \emptyset = Y \cap Z \wedge \text{alt}(Y, Z))$$

Here  $\text{alt}(Y, Z)$  is supposed to express that the elements of  $Y$  and  $Z$  strictly alternate as in

$$y_1 < z_1 < y_2 < z_2 < \dots < y_k < z_k$$

## Missing Pieces

$$X = Y \cup Z \iff \forall u (X(u) \leftrightarrow Y(u) \vee Z(u))$$

$$\emptyset = Y \cap Z \iff \neg \exists u (Y(u) \wedge Z(u))$$

$$\text{alt}(Y, Z) \iff \exists y \in Y \forall x < y (\neg Z(x)) \wedge$$

$$\exists z \in Z \forall x > z (\neg Y(x)) \wedge$$

$$\forall y \in Y \exists z \in Z (y < z \wedge \forall x (y < x < z \rightarrow \neg Y(x) \wedge \neg Z(x)))$$

$$\forall z \in Z \exists y \in Y (y < z \wedge \forall x (y < x < z \rightarrow \neg Y(x) \wedge \neg Z(x)))$$

## Exercise

Show that one can check if the number of  $a$ 's is a multiple of  $k$ , for any fixed  $k$ .

## Less-Than or Successor

In applications  $x < y$  is slightly more useful than  $y = x + 1$ , but either one would have the same expressiveness.

On the one hand

$$y = x + 1 \iff x < y \wedge \forall z (x < z \rightarrow y \leq z)$$

On the other hand

$$x < y \iff x \neq y \wedge \forall X (\forall z (X(z) \rightarrow X(z+1)) \rightarrow X(y))$$

This is sometimes written as  $\text{MSO}[<] = \text{MSO}[+1]$ .

## The Link

## Definition

A language  $L$  is **MSO[<] definable** (or simply **MSO[<]**) if there is some sentence  $\varphi$  such that

$$L = \mathcal{L}(\varphi) = \{ w \in \Sigma^* \mid w \models \varphi \}.$$

The examples suggest that  $\text{MSO}[<]$  is closely related to being regular.

## Theorem (Büchi 1960)

A language is regular if, and only if, it is  $\text{MSO}[<]$  definable.

The theorem connects complexity with definability: we can recognize a set of strings in constant space if, and only if, the set can be described by a formula in our logic.

## Formula to Regular (Sketch)

Obviously, the proof comes in two parts:

- 1 For every regular language  $L$  there is a sentence  $\varphi$  such that  $L = \mathcal{L}(\varphi)$ .
- 2 For every sentence  $\varphi$  the language  $\mathcal{L}(\varphi)$  is regular.

Part (2) is by induction on  $\varphi$  but there is the usual technical twist: we need to deal not just with sentences but also with free variables.

We will postpone this discussion for the moment, see the next section

## Regular to Formula (Sketch)

We may safely assume that the regular language  $L$  is given by a DFA  $M = \langle Q, \Sigma, \delta; q_0, F \rangle$ .

For simplicity assume  $Q = [n]$  and  $q_0 = 1$ .

We have to construct a formula  $\varphi$  such that  $w \models \varphi$  iff  $M$  accepts  $w$ .

Consider a trace of  $M$  on input  $w$

$$q_0 w_1 q_1 w_2 q_2 \dots q_{m-1} w_m q_m.$$

Here  $m$  can be arbitrarily large.

The main idea is to partition  $[m]$  into  $n$  blocks  $X_1, X_2, \dots, X_n$  such that

$$X_p(x) \iff \delta(q_0, w_1 \dots w_x) = p$$

Of course, some of these blocks may be empty.

## Expressing Transitions

Now suppose  $\delta(p, a) = q$ .

Then we can assert

$$\forall x (X_p(x) \wedge Q_a(x+1) \rightarrow X_q(x+1))$$

By writing a big conjunction of such assertions, plus formulae that pin down the first and last state we arrive at a formula of the form

$$\varphi = \exists X_1, \dots, X_n \psi$$

that does the job (here  $\psi$  contains no second order quantifiers).

## Exercise

Fill in all the details in the last proof.

## Upcoming Attractions

Büchi's result may seem a bit odd and of no practical values.

However, it also holds for **infinite words** and in that version it is extremely useful.

Variants of Büchi's theorem are also important in complexity theory.

- The Logic Connection
- Rational Relations
- Deciding Rational Structures

## More Logic

We already know that regular languages are closed under union, intersection and complement.

In the context of logic, these operations correspond to "or," "and" and "not."

Moreover, the closure is effective: we can actually compute the corresponding machines (and fairly easily).

**Innocent Question:** Is there anything corresponding to quantification? Can we model "there exists  $x$  such that ..." in terms of finite state machines?

This time we want to quantify over words, not positions, so the framework is slightly different.

So far, our machines only describe languages, which we can think of as monadic (unary) relations on words. How about binary relations?

## Relations versus Machines

How does a FSM check a binary relation  $\rho \subseteq \Sigma^* \times \Sigma^*$ ?

If only words of the same length are related then we can think of the input as being specified by pairs of letters: to test  $x \rho y$  we feed pairs  $(x_i, y_i) \in \Sigma \times \Sigma$  to the machine.

Alas, this is not enough, we need to be able to deal with words of different length. To this end, define a new alphabet

$$\Gamma = (\Sigma \cup \{\#\}) \times (\Sigma \cup \{\#\})$$

where  $\# \notin \Sigma$  is a new **padding symbol**.

## Padding Words

We can think of a word over  $\Gamma$  as split into two tracks, each containing a word over  $\Sigma \cup \{\#\}$ .

Given two words  $x, y \in \Sigma^*$  we define a word  $x:y$  over  $\Gamma$  by padding the shorter word with #'s on the right, if necessary.

$$x:y = \begin{array}{ccccccc} x_1 & x_2 & \dots & x_n & \# & \dots & \# \\ y_1 & y_2 & \dots & y_i & y_{i+1} & \dots & y_m \end{array}$$

This is sometimes called the **convolution** of  $x$  and  $y$ .

Note that no padding is needed if  $x$  and  $y$  have the same length.

## Rational Relations

### Definition

A relation  $\rho \subseteq \Sigma^* \times \Sigma^*$  is **rational** if there is finite state machine  $M$  over  $\Gamma$  such that

$$\mathcal{L}(M) = \{x:y \mid \rho(x, y)\}$$

$k$ -ary relations are treated similarly.

So in a sense rational relations are the most computationally simple relations (other than entirely trivial ones such as "always true").

A cheap example is identity:  $x = y$  is a rational relation.

A moment's thought reveals that  $x \neq y$  is also rational.

## Lexicographic Order is Rational

Given an ordered alphabet  $\Sigma$  consider the binary relation  $\leq_\ell$  on  $\Sigma^*$  defined by

$$x \leq_\ell y \iff x \text{ is a prefix of } y \vee \exists a, b \in \Sigma, u, v, w \in \Sigma^* (x = uav \wedge y = ubw \wedge a < b)$$

### Proposition

*Lexicographic order is rational.*

*Proof.* Well ... □

The length-lex order is likewise rational.

But the relations "x is a factor of y" and "x is a subword of y" are not rational.

## Addition is Rational

Consider the ternary relation  $\alpha$  on  $2$  defined by

$$\alpha(x, y, z) \iff \text{bin}(x) + \text{bin}(y) = \text{bin}(z)$$

where  $\text{bin}(x)$  is the numerical value of  $x$  assuming the LSD is first (reverse binary).

### Proposition

*Binary addition in reverse binary is rational.*

*Proof.* The kindergarten algorithm for addition shows that  $\alpha$  is rational.  $\square$

But note: there is no analogous result for multiplication.

## Boolean Operations

### Claim

Given two  $k$ -ary rational relations  $\rho$  and  $\sigma$  on  $\Sigma^*$  the following relations are also rational:

$$\rho \cup \sigma, \rho \cap \sigma, \rho - \sigma,$$

The proof is very similar to the argument for regular languages: one can effectively construct the corresponding automata.

So we can handle disjunctions, conjunctions and negation of rational relations.

So far, this is not exactly overwhelming.

## Relational Composition

Given two binary relations  $\rho$  and  $\sigma$  on  $\Sigma^*$  define their composition to be the binary relation

$$x (\rho \circ \sigma) y \iff \exists z (x \rho z \wedge z \sigma y)$$

### Claim

*If both  $\rho$  and  $\sigma$  are rational then so is their composition  $\rho \circ \sigma$ .*

*Proof.* Clearly the ternary relation  $x \rho z \wedge z \sigma y$  is rational: we can run the machine for  $\rho$  the first two tracks and the machine for  $\sigma$  on the last two.

We can make this product machine "guess" what the proper values for  $z_i$  is: just use nondeterminism. Note that one needs to cope with the problem is that  $z$  may be longer than  $x$  and  $y$ .  $\square$

The same holds for  $k$ -ary  $\rho$  and  $\ell$ -ary  $\sigma$ .

## Iteration

One might wonder what happens to a rational relation  $\rho$  if we move to the transitive reflexive closure  $\rho^*$ .

The answer in general is: nothing good. Rationality is lost in general, and things tend to become undecidable.

To see why, recall that configurations of Turing machines are just words  $u p v \in \Sigma^* Q \Sigma^*$ . It is easy to check that the one-step relation for Turing machines is rational.

But then  $\frac{|}{M}$ , the transitive reflexive closure of this relation, is undecidable (though semi-decidable).

## Existential Quantifiers

Suppose  $\rho$  is a  $k$ -ary relation on  $\Sigma^*$ . Define the **projection** of  $\rho$  to be

$$\rho'(x_2, \dots, x_k) \iff \exists z \rho(z, x_2, \dots, x_k)$$

### Claim

*Whenever  $\rho$  is rational, so is its projection  $\rho'$ .*

*Proof.*

Erase the first track in the  $k$ -track alphabet:

$$p \xrightarrow{a_1 a_2 \dots a_k} q \Rightarrow p \xrightarrow{a_2 \dots a_k} q$$

That's it! Of course, the new machine will be nondeterministic in general.  $\square$

Universal quantifiers can be handled via  $\forall = \neg \exists \neg$ .

### ■ The Logic Connection

### ■ Rational Relations

### ③ Deciding Rational Structures

## Rational Structures

Suppose we have some space whose points are given by words over some alphabet: just think about binary words, representing a fixed number of bits.

Suppose further that the system evolves according to some evolution relation  $\rightarrow$  (where  $x \rightarrow y$  means that  $x$  evolves to  $y$  in one step).

So we have a structure

$$\mathcal{C} = \langle \Sigma^*, \rightarrow \rangle$$

where  $\rightarrow$  may or may not be deterministic (functional).

**Innocent Question:** Assuming that  $\rightarrow$  is rational, what kind of properties of such a system can we check automatically?

## Some Typical Questions

- Is the system reversible?
- Does every state have a predecessor?
- Does every state have exactly  $k$  successors?
- Does every state have exactly  $k$  predecessors?
- Do all orbits end in a fixed point?
- Is there a limit cycle of length  $k$ ?
- Can a set of states  $B$  ( $B$  as in bad) be reached from some state  $p$ ?

## A Good Framework

Some of these questions turn out to be undecidable, even if  $\rightarrow$  is rational. But others can be dealt with nicely.

The key trick is to choose the right logic: expressive enough to make interesting assertions, but easy enough so that truth remains decidable.

In this case, plain first order logic is a good first choice. For example

$$\forall y \exists x (x \rightarrow y)$$

means that every state has a predecessor state. And

$$\forall x, y, z (x \rightarrow y \wedge x \rightarrow z \Rightarrow y = z)$$

means that the system is reversible (deterministic).

$$\exists x, y, z (x \rightarrow y \wedge y \rightarrow z \wedge z \rightarrow x \wedge x \neq y)$$

means that there is a 3-cycle somewhere.

## Truth

So we are given  $\mathcal{C} = \langle \Sigma^*, \rightarrow \rangle$ : the alphabet is just a table and we have a finite state machine for  $\rightarrow$ .

Also, we have a sentence  $\varphi$  in first order logic  $\text{FOL}[\rightarrow]$  and we would like to know if  $\varphi$  is true over  $\mathcal{C}$ .

More technically, we want to solve the following decision problem:

**Problem:** **FO Validity**  
**Instance:** A rational structure  $\mathcal{C}$  and a FOL sentence  $\varphi$ ?  
**Question:** Is  $\varphi$  valid over  $\mathcal{C}$ ?

And, as always, we want good algorithms for this.

## The Key Idea: Using Automata

Consider first a formula without quantifiers  $\varphi(x_1, \dots, x_k)$ .

In the easiest case  $\varphi \equiv x_1 = x_2$  or  $\varphi \equiv x_1 \rightarrow x_2$ .

We can construct an automaton  $\mathcal{A}_\varphi$  that accepts precisely those  $k$ -track words that satisfy the formula:

$$\mathcal{L}(\mathcal{A}_\varphi) = \{ u_1 u_2 \dots u_k \in \Gamma^* \mid \mathcal{C} \models \varphi(u_1, u_2, \dots, u_k) \}$$

For basic formulae this follows from the fact that  $\rightarrow$  is rational (and equality, of course).

For Boolean combinations we can construct the appropriate machines using closure properties.

## Quantifiers

It is well-known that every FO formula can be written in prenex normal-form: all quantifiers are up front, something like

$$\exists x_1 \exists x_2 \forall x_3 \exists x_4 \forall x_5 \varphi(x_1, \dots, x_k)$$

There is a simple algorithm to convert an arbitrary formula to PNF.

Since we can eliminate universal quantifiers via  $\forall = \neg \exists \neg$  we only need to contend with

$$\exists z \varphi(z, x_1, \dots, x_k)$$

We already know how to do this: Just project  $\mathcal{L}(\varphi(z, x_1, \dots, x_k))$ : there is an accepting computation in the projected automaton if, and only if, there is some  $z$ .

## Efficiency

- $\forall$  and  $\exists$  are linear if we allow nondeterminism.
- $\wedge$  is quadratic via a product machine construction.
- But:  $\neg$  is potentially exponential since we need to determinize first. Note that universal quantifiers produce two negations.

So this is a bit disappointing: we may run out of computational steam even when the formula is not terribly large.

A huge amount of work has gone into streamlining this and similar algorithms to deal with instances that are of practical relevance.

## 3-Cycles

As an example how this might work, consider the question of whether the structure  $\mathcal{C}$  contains a 3-cycle.

To keep things simple, suppose that  $\rightarrow$  is length-preserving:  $x \rightarrow y$  implies  $|x| = |y|$ .

We need to check

$$\exists x, y, z (x \rightarrow y \wedge y \rightarrow z \wedge z \rightarrow x \wedge x \neq y)$$

So we use the 3-track alphabet  $\Gamma = \Sigma \times \Sigma \times \Sigma$ .

Given a machine  $M$  for  $\rightarrow$  we can concoct a machine

$$C_3 = M_{1,2} \times M_{2,3} \times M_{3,1} \times U_{1,2}$$

where  $M_{i,j}$  tests if the word in track  $i$  evolves to the word in track  $j$ .

Machine  $U_{i,j}$  tests if the word in track  $i$  is different from the word in track  $j$ .

## Projecting

So we get a machine  $C_3$  that is roughly cubic in the size of the original machine for  $\rightarrow$  (disregarding possible savings for accessibility).

Once  $A$  is built, we erase all the labels and are left with a digraph.

That digraph has a path from an initial state to a final state if, and only if, there is a 3-cycle in  $\mathcal{C}$ .

Note, though, how the machines grow if we want to test for longer cycles: the size of  $C_k$  is roughly  $n^k$  where  $n$  is the size of the original machine – this will not work for long cycles.

## Summary

- There is a deep connection between regular languages and certain weak logics, notably monadic second order logic.
- Rational relations can be checked by a finite state machine and are in a sense the simplest non-trivial relations.
- Systems built from rational relations have decidable first order properties; the decision algorithm uses automata to express logic.
- Alas, efficiency constraints make the algorithm applicable only in a fairly narrow setting.
- Similar methods are very important in model checking and formal verification.