Equivalence of generative systems and constraints:

finite state automata and monadic second-order logic over strings

JACOB LOUIS HOOVER

2018-11-15

Outline

- Two basic approaches
  - Minimalist Grammar - a generative system
  - Binding theory - a system of constraints
  - Both approaches

- FSA and MSOL[S] over strings
  - Finite state automata
  - Strings as models
  - Strings and logic
  - The equivalence of FSAs and MSOL[S]
    - proof ($\rightarrow$)
    - proof ($\leftarrow$)
  - And so forth
  - And so on

1. Two basic approaches

1.1 Minimalist Grammar - a generative system

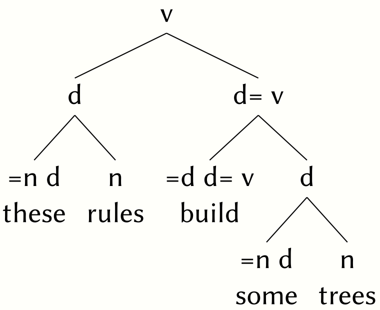
Given some lexical items:

- build :: =d d= v
- rules :: n
- trees :: n
- these :: =n d
- some :: =n d
- ∅ :: =n d

1.1 Minimalist Grammar - a generative system

Given some lexical items:

- build :: =d d= v
- rules :: n
- trees :: n
- these :: =n d
- some :: =n d
- ∅ :: =n d



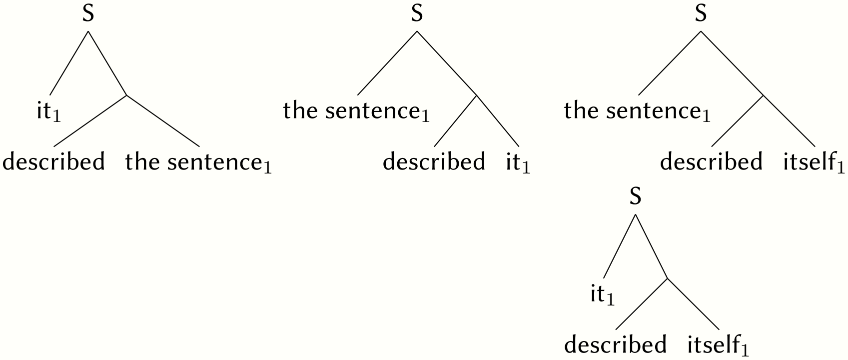
The rules are a formal system that describes a set of structures.

1.2 Binding theory - a system of constraints

- Principle A: An anaphor must be bound locally.
- Principle B: A pronoun must not be bound locally.
- Principle C: An R-expression must not be bound.

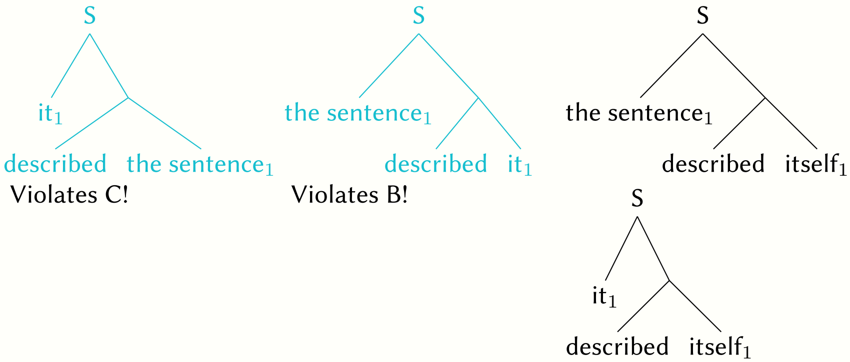
1.2 Binding theory - a system of constraints

- Principle A: An anaphor must be bound locally.
- Principle B: A pronoun must not be bound locally.
- Principle C: An R-expression must not be bound.



1.2 Binding theory - a system of constraints

- Principle A: An anaphor must be bound locally.
- Principle B: A pronoun must not be bound locally.
- Principle C: An R-expression must not be bound.



1.2 Binding theory - a system of constraints

Principle A: *An anaphor must be bound by an element in argument position within its governing category.*
This is a logical statement about trees.

1.2 Binding theory - a system of constraints

Principle A: *An anaphor must be bound by an element in argument position within its governing category.*
This is a logical statement about trees.
This can be formalized. E.g.:

$$(\forall x, \forall X)[+\text{anaphor}(x) \wedge \text{GC}(X, x)] \rightarrow (\exists y)[X(y) \wedge \text{A-position}(y) \wedge \text{c-command}(x, y) \wedge \text{coindexed}(x, y)]]$$

1.2 Binding theory - a system of constraints

Principle A: *An anaphor must be bound by an element in argument position within its governing category.*
This is a logical statement about trees.
This can be formalized. E.g.:

$$(\forall x, \forall X)[+\text{anaphor}(x) \land \text{GC}(X, x)] \rightarrow$$
$$(\exists y)[X(y) \land \text{A-position}(y) \land \text{c-command}(x, y) \land \text{coindexed}(x, y)]]$$

This is (more or less) from Jim Rogers translation of GB into logic.
I won't talk about this, but a similar, simpler version.

1.3 Both approaches

Define a set of structures.

1.3 Both approaches

Define a set of structures.

Linguistic theories are often formulated as a mix of both of these approaches.

1.3 Both approaches

Define a set of structures.

Linguistic theories are often formulated as a mix of both of these approaches.

1. Generative systems
   - Phrase Structure Grammar
   - Minimalism
   - ...
2. Constraint-based systems
   - Government and Binding
   - ...

1.3 Both approaches

Define a set of structures.

Linguistic theories are often formulated as a mix of both of these approaches.

1. Generative systems
   - Phrase Structure Grammar
   - Minimalism
   - ...
2. Constraint-based systems
   - Government and Binding
   - ...

Are these equivalent? Can one be translated into the other?

1.3 Both approaches

Define a set of structures.
Linguistic theories are often formulated as a mix of both of these approaches.

1. Generative systems
   - Phrase Structure Grammar
   - Minimalism
   - ...
2. Constraint-based systems
   - Government and Binding
   - ...

Are these equivalent? Can one be translated into the other? If yes... efficiently?

2. FSA and MSOL[S] over strings

2.1 Finite state automata

A finite state automaton is a simple generative system.

2.1 Finite state automata

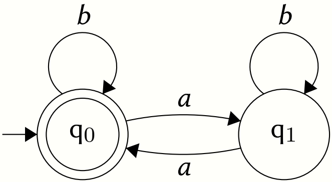
A finite state automaton is a simple generative system.
A minimalist grammar where all lexical items are of the following form (having at most one selectional feature only on the right):

- a :: (=y) x

is a finite state automaton.

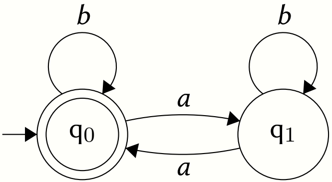
2.1 Finite state automata

For example, let $V = \{a, b\}$.
Take FSA $\mathcal{A}$:



2.1 Finite state automata

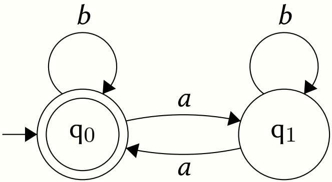
For example, let $V = \{a, b\}$.
Take FSA $\mathcal{A}$:



Accepts any string on V^* with an even number of *a*s.

2.1 Finite state automata

For example, let $V = \{a, b\}$.
Take FSA $\mathcal{A}$:



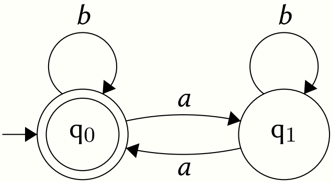
Accepts any string on V^* with an even number of *a*s.

Grammar for this language:

- a :: $=q_1$ q_0
- a :: $=q_0$ q_1
- b :: $=q_0$ q_0
- b :: $=q_1$ q_1
- ε :: q_0

2.1 Finite state automata

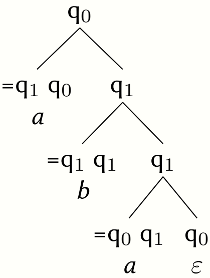
For example, let $V = \{a, b\}$.
Take FSA $\mathcal{A}$:



Accepts any string on V^* with an even number of as.

Grammar for this language:

- a :: =q_1 q_0
- a :: =q_0 q_1
- b :: =q_0 q_0
- b :: =q_1 q_1
- ε :: q_0



2.1 Finite state automata

The class of strings accepted by finite state automata can also be descibed in terms of constraints.

- “there are an even number of *a*’s in the string”

2.1 Finite state automata

The class of strings accepted by finite state automata can also be descibed in terms of constraints.

- "there are an even number of *a*'s in the string"

Question:

given a language (a set of strings) generated by a FSA, what kind of constraints are needed to define it?

2.1 Finite state automata

The class of strings accepted by finite state automata can also be descibed in terms of constraints.

- "there are an even number of *a*'s in the string"

Question:

given a language (a set of strings) generated by a FSA, what kind of constraints are needed to define it?

Answer:

If a language can be generated by a FSA, it can be defined using constraints written in monadic second-order logic (with a successor).

2.1 Finite state automata

In fact, this generative system (any FSA you can build) and this system of constraints (anything you can write in the logic MSOL[S]) are equivalent in expressive power: they both describe the same class of languages.

2.1 Finite state automata

In fact, this generative system (any FSA you can build) and this system of constraints (anything you can write in the logic MSOL[S]) are equivalent in expressive power: they both describe the same class of languages.
First, to formalize:

- FSA
- Strings as models
- MSOL

2.1 Finite state automata

A finite state automaton $\mathcal{A} = (V, Q, q_0, F, \Delta)$, consisting of

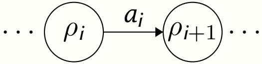
- the vocabulary V
- Q is a finite set of states $q_0, q_1, \ldots, q_k$
  - $q_0 \in Q$ is the *initial* state
  - $F \subseteq Q$ is set of *final* (or *accepting*) states
- $\Delta \subseteq Q \times V \times Q$ is the *transition relation*

2.1 Finite state automata

A finite state automaton $\mathcal{A} = (V, Q, q_0, F, \Delta)$, consisting of

- the vocabulary V
- Q is a finite set of states $q_0, q_1, \ldots, q_k$
  - $q_0 \in Q$ is the *initial* state
  - $F \subseteq Q$ is set of *final* (or *accepting*) states
- $\Delta \subseteq Q \times V \times Q$ is the *transition relation*

A string $\sigma = a_0 \ldots a_{n-1}$ is accepted by $\mathcal{A}$ if there is a successful run of $\mathcal{A}$ on σ, that is, a sequence $\rho = \rho_0 \ldots \rho_n$ of states, such that $\rho_0 = q_0$, $\rho_n \in F$, and $(\rho_i, a_i, \rho_{i+1}) \in \Delta$ for $i < n$.



A language is accepted by $\mathcal{A}$ if all its strings are accepted.

2.2 Strings as models

Let V be a finite vocabulary and $\sigma = a_0 \dots a_{n-1}$ be a string over V. In order to interpret logical statements about this string, create a *string model* of σ:

$$\underline{\sigma} = (\mathrm{pos}(\sigma), S^{\sigma}, (Q_a^{\sigma})_{a \in V}) \qquad (1)$$

Where $\mathrm{pos}(\sigma)$ is $\{0, 1, \dots, n-1\}$, the set of word positions in the string , S^{σ} is the natural successor relation defined on these integers, and Q_a^{σ} is a unary predicates collecting for $a \in V$ the positions in $\mathrm{pos}(\sigma)$ where a occurs.

2.2 Strings as models

Let V be a finite vocabulary and $\sigma = a_0 \dots a_{n-1}$ be a string over V. In order to interpret logical statements about this string, create a *string model* of σ:

$$\underline{\sigma} = (\mathrm{pos}(\sigma), S^{\sigma}, (Q_a^{\sigma})_{a \in V}) \tag{1}$$

Where $\mathrm{pos}(\sigma)$ is $\{0, 1, \dots, n-1\}$, the set of word positions in the string , S^{σ} is the natural successor relation defined on these integers, and Q_a^{σ} is a unary predicates collecting for $a \in V$ the positions in $\mathrm{pos}(\sigma)$ where a occurs.
E.g.

$$\underline{bab} = (\{0, 1, 2\}, S^{bab}, Q_a^{bab} = \{1\}, Q_b^{bab} = \{0, 2\})$$

2.3 Strings and logic

Given string models over V, statements about the strings can be formalized in logic.

First-order logic:

- variables
  - $x, y, x_i, \ldots$ (range over positions in the string models)
- atomic formulæ:
  - $x = y$ (equality)
  - $S(x, y)$ (the successor of x is y)
  - $Q_a(x)$, $a \in V$ (position x has label a)
- connectives: $\wedge, \vee, \neg, \rightarrow, \leftrightarrow$ and quantifiers: $\exists, \forall$

2.3 Strings and logic

Given string models over V, statements about the strings can be formalized in logic.

Monadic second-order logic:

- variables
  - $x, y, x_i, \ldots$ (range over positions in the string models)
  - $X, Y, X_i, \ldots$ (range over sets of positions)
- atomic formulæ:
  - $x = y$ (equality)
  - $S(x, y)$ (the successor of x is y)
  - $Q_a(x)$, $a \in V$ (position x has label a)
  - $X(x)$ (set X contains element x)
- connectives: $\wedge, \vee, \neg, \rightarrow, \leftrightarrow$ and quantifiers: $\exists, \forall$

For example: take the first-order sentence
$\phi = \exists x.[Q_a(x) \wedge \neg\exists y.S(x, y)]$ "the string ends in *a*"

2.3 Strings and logic

For example: take the first-order sentence

$\phi = \exists x.[Q_a(x) \land \neg \exists y.S(x, y)]$ "the string ends in *a*"

To evaluate this on string *ba*: take the model $\underline{ba}$, and see whether

$\exists x.[Q_a^{ba}(x) \land \neg \exists y.S^{ba}(x, y)]$ is true.

2.3 Strings and logic

For example: take the first-order sentence

$\phi = \exists x.[Q_a(x) \wedge \neg\exists y.S(x,y)]$ "the string ends in *a*"

To evaluate this on string *ba*: take the model $\underline{ba}$, and see whether $\exists x.[Q_a^{ba}(x) \wedge \neg\exists y.S^{ba}(x,y)]$ is true. ✓

2.3 Strings and logic

For example: take the first-order sentence

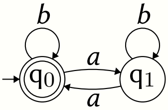
$\phi = \exists x.[Q_a(x) \wedge \neg \exists y.S(x, y)]$ "the string ends in *a*"

To evaluate this on string *ba*: take the model $\underline{ba}$, and see whether

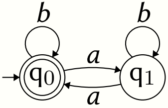
$\exists x.[Q_a^{ba}(x) \wedge \neg \exists y.S^{ba}(x, y)]$ is true. ✓

So, we say $\underline{ba} \models \phi$: *ba models* ϕ.

2.3 Strings and logic

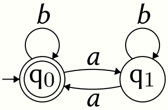


2.3 Strings and logic



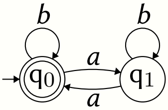
To capture the same language with a MSOL constraint:

2.3 Strings and logic



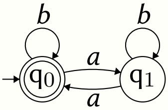
To capture the same language with a MSOL constraint:
First, define the ordering "$<$":

2.3 Strings and logic



To capture the same language with a MSOL constraint:
First, define the ordering "$<$":
$x < y := \neg(x = y) \land \forall X[X(x) \land \forall z \forall w(X(z) \land S(z, w) \rightarrow X(w)) \rightarrow X(y)]$

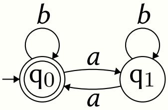
2.3 Strings and logic



To capture the same language with a MSOL constraint:
First, define the ordering "<":
$x < y := \neg(x = y) \wedge \forall X[X(x) \wedge \forall z \forall w(X(z) \wedge S(z, w) \rightarrow X(w)) \rightarrow X(y)]$ and
define some shorthand:

$$\phi(\mathrm{first}_a) := \exists x.\phi(x) \wedge Q_a(x) \wedge (\neg \exists z.z < x \wedge Q_a(z))$$
$$\phi(\mathrm{last}_a) := \exists x.\phi(x) \wedge Q_a(x) \wedge (\neg \exists z.x < z \wedge Q_a(z))$$
$$\mathrm{next}_a(x, y) := Q_a(x) \wedge Q_a(y) \wedge (\neg \exists z.x < z \wedge z < y \wedge Q_a(z))$$

2.3 Strings and logic



To capture the same language with a MSOL constraint:
First, define the ordering "<":
$x < y := \neg(x = y) \wedge \forall X[X(x) \wedge \forall z \forall w(X(z) \wedge S(z,w) \rightarrow X(w)) \rightarrow X(y)]$ and
define some shorthand:

$$\phi(\mathrm{first}_a) := \exists x.\phi(x) \wedge Q_a(x) \wedge (\neg\exists z.z < x \wedge Q_a(z))$$
$$\phi(\mathrm{last}_a) := \exists x.\phi(x) \wedge Q_a(x) \wedge (\neg\exists z.x < z \wedge Q_a(z))$$
$$\mathrm{next}_a(x,y) := Q_a(x) \wedge Q_a(y) \wedge (\neg\exists z.x < z \wedge z < y \wedge Q_a(z))$$

$$\forall x.Q_a(x) \rightarrow \exists X.X(\mathrm{first}_a) \wedge \forall y \forall z.[\mathrm{next}_a(y,z) \rightarrow (X(y) \leftrightarrow \neg X(z))] \wedge \neg X(\mathrm{last}_a)$$

2.4 The equivalence of FSAs and MSOL[S]

Theorem

A language L of finite-length strings is can be generated by a FSA iff L is definable in MSOL[S].

2.4 The equivalence of FSAs and MSOL[S]

Theorem

A language L of finite-length strings is can be generated by a FSA iff L is definable in MSOL[S].

The proof consists of showing that
($\rightarrow$) given an automaton, we can write a formula, and
($\leftarrow$) given a formula, we can construct an automaton.

2.4 The equivalence of FSAs and MSOL[S] proof ($\rightarrow$)

Given automaton $\mathcal{A} = (V, Q, q_0, F, \Delta)$, write $\phi_{\mathcal{A}}$, a MSOL[S]-sentence (a formula with no free variables) such that $\underline{\sigma} \models \phi_{\mathcal{A}}$ for any string σ that is accepted by $\mathcal{A}$.

2.4 The equivalence of FSAs and MSOL[S] proof ($\rightarrow$)

Given automaton $\mathcal{A} = (V, Q, q_0, F, \Delta)$, write $\phi_{\mathcal{A}}$, a MSOL[S]-sentence (a formula with no free variables) such that $\underline{\sigma} \models \phi_{\mathcal{A}}$ for any string σ that is accepted by $\mathcal{A}$.
That is, $\phi_{\mathcal{A}}$ must express in $\underline{\sigma}$ the existence of an accepting run of $\mathcal{A}$ on σ.

Given automaton $\mathcal{A} = (V, Q, q_0, F, \Delta)$, write $\phi_{\mathcal{A}}$, a MSOL[S]-sentence (a formula with no free variables) such that $\underline{\sigma} \models \phi_{\mathcal{A}}$ for any string σ that is accepted by $\mathcal{A}$.
That is, $\phi_{\mathcal{A}}$ must express in $\underline{\sigma}$ the existence of an accepting run of $\mathcal{A}$ on σ.
Strategy: Associate a set variable X_i with each state q_i in $\mathcal{A}$. This set variable denotes the positions on the string when $\mathcal{A}$ assumes state q_i.

2.4 The equivalence of FSAs and MSOL[S] proof ($\rightarrow$)

if $\mathcal{A}$ has k states, we can use set variables $X_0, \ldots, X$ to describe successful runs:

- the X_i's are pairwise disjoint sets over the positions (at any position in the string, the automaton is in exactly one state)
-
-
-

2.4 The equivalence of FSAs and MSOL[S] proof ($\rightarrow$)

if $\mathcal{A}$ has k states, we can use set variables $X_0, \ldots, X$ to describe successful runs:

- the X_i's are pairwise disjoint sets over the positions (at any position in the string, the automaton is in exactly one state)
- at the first position, the automaton is in initial state q_0
-
-

if $\mathcal{A}$ has k states, we can use set variables $X_0, \ldots, X$ to describe successful runs:

- the X_i's are pairwise disjoint sets over the positions (at any position in the string, the automaton is in exactly one state)
- at the first position, the automaton is in initial state q_0
- for any pair of successive positions in the string x, y, there is a valid transition given by the states at positions x and y, and the label at position x
-

2.4 The equivalence of FSAs and MSOL[S] proof ($\rightarrow$)

if $\mathcal{A}$ has k states, we can use set variables $X_0, \ldots, X$ to describe successful runs:

- the X_i's are pairwise disjoint sets over the positions (at any position in the string, the automaton is in exactly one state)
- at the first position, the automaton is in initial state q_0
- for any pair of successive positions in the string x, y, there is a valid transition given by the states at positions x and y, and the label at position x
- at the final position there is a valid transition into an accepting state

2.4 The equivalence of FSAs and MSOL[S] proof ($\rightarrow$)

$\exists X_0, \ldots, \exists X_k \Big[$

- the X_i's are pairwise disjoint sets over the positions (at any position in the string, the automaton is in exactly one state)
- at the first position, the automaton is in initial state q_0
- for any pair of successive positions in the string x, y, there is a valid transition given by the states at positions x and y, and the label at position x
- at the final position there is a valid transition into an accepting state

2.4 The equivalence of FSAs and MSOL[S] proof ($\to$)

$\exists X_0, \ldots, \exists X_k \Big[$

- $\bigwedge_{i \neq j} \forall x \neg(X_i(x) \wedge X_j(x))$
- at the first position, the automaton is in initial state q_0
- for any pair of successive positions in the string x, y, there is a valid transition given by the states at positions x and y, and the label at position x
- at the final position there is a valid transition into an accepting state

$\exists X_0, \ldots, \exists X_k \Big[$

- $\bigwedge_{i \neq j} \forall x \neg (X_i(x) \wedge X_j(x))$
- $\forall x (\neg \exists y S(y, x) \rightarrow X_0(x))$
- for any pair of successive positions in the string x, y, there is a valid transition given by the states at positions x and y, and the label at position x
- at the final position there is a valid transition into an accepting state

2.4 The equivalence of FSAs and MSOL[S] proof ($\rightarrow$)

$\exists X_0, \ldots, \exists X_k \Big[$

- $\bigwedge_{i \neq j} \forall x \neg (X_i(x) \wedge X_j(x))$
- $\forall x (\neg \exists y S(y, x) \rightarrow X_0(x))$
- $\forall x \forall y (S(x, y) \rightarrow \bigvee_{(q_i, a, q_j) \in \Delta} X_i(x) \wedge Q_a(x) \wedge X_j(y))$
- at the final position there is a valid transition into an accepting state

2.4 The equivalence of FSAs and MSOL[S] proof ($\rightarrow$)

$\exists X_0, \ldots, \exists X_k \Big[$

- $\bigwedge_{i \neq j} \forall x \neg (X_i(x) \wedge X_j(x))$
- $\forall x (\neg \exists y S(y, x) \rightarrow X_0(x))$
- $\forall x \forall y (S(x, y) \rightarrow \bigvee_{(q_i, a, q_j) \in \Delta} X_i(x) \wedge Q_a(x) \wedge X_j(y))$
- $\forall x (\neg \exists y S(x, y) \rightarrow \bigvee_{(q_i, a, q_j) \in \Delta, q_j \in F} X_i(x) \wedge Q_a(x)) \Big]$

2.4 The equivalence of FSAs and MSOL[S] proof ($\rightarrow$)

$\exists X_0, \ldots, \exists X_k \Big[$

- $\bigwedge_{i \neq j} \forall x \neg (X_i(x) \wedge X_j(x))$
- $\forall x (\neg \exists y S(y, x) \rightarrow X_0(x))$
- $\forall x \forall y (S(x, y) \rightarrow \bigvee_{(q_i, a, q_j) \in \Delta} X_i(x) \wedge Q_a(x) \wedge X_j(y))$
- $\forall x (\neg \exists y S(x, y) \rightarrow \bigvee_{(q_i, a, q_j) \in \Delta, q_j \in F} X_i(x) \wedge Q_a(x)) \Big]$

The form of this sentence: automata normal form.

2.4 The equivalence of FSAs and MSOL[S] proof ($\rightarrow$)

$\mathcal{A}$ accepts/generates a string σ iff

$$\underline{\sigma} \models \exists X_0 \ldots \exists X_k \quad \Big[\bigwedge_{i \neq j} \forall x \neg (X_i(x) \wedge X_j(x))$$

$$\wedge\, \forall x (\neg \exists y S(y, x) \rightarrow X_0(x))$$

$$\wedge\, \forall x \forall y (S(x, y) \rightarrow \bigvee_{(q_i, a, q_j) \in \Delta} X_i(x) \wedge Q_a(x) \wedge X_j(y))$$

$$\wedge\, \forall x (\neg \exists y S(x, y) \rightarrow \bigvee_{(q_i, a, q_j) \in \Delta, q_j \in F} X_i(x) \wedge Q_a(x)) \Big]$$

2.4 The equivalence of FSAs and MSOL[S] proof ($\leftarrow$)

In the other direction, show that given a formula ϕ of MSOL[S], we can construct an FSA $\mathcal{A}_\phi$, such that for any string which models the formula $\underline{\sigma} \models \phi$, this string is accepted by $\mathcal{A}_\phi$.

2.4 The equivalence of FSAs and MSOL[S] proof ($\leftarrow$)

In the other direction, show that given a formula ϕ of MSOL[S], we can construct an FSA $\mathcal{A}_\phi$, such that for any string which models the formula $\underline{\sigma} \models \phi$, this string is accepted by $\mathcal{A}_\phi$.
Strategy: A proof by induction on formulas. Show that for every atomic formula, there is a simple FSA that recognizes precisely the set of strings defined by that atomic formula.
For the inductive step show that this property is closed under connectives and quantification.

2.4 The equivalence of FSAs and MSOL[S] proof ($\leftarrow$)

Work with the expressively equivalent (but syntactically simpler) $\mathrm{MSOL}_0[S]$:

- has only set variables (first order variables x are converted to singleton set variables X).
- connectives $\neg$ and $\vee$, and quantification $\exists$.

This means the proof must just

- construct an FSA that is equivalent to each atomic formula of $\mathrm{MSOL}_0[S]$
- show that the class of recognizable languages is closed under complement and union and projection

2.5 And so forth

Consequences and corollaries:

- Any MSOL[S] formula can be written as one in ‘automata normal form’.

2.5 And so forth

Consequences and corollaries:

- Any MSOL[S] formula can be written as one in ‘automata normal form’.
- General translation of logic to automata: hard.

2.5 And so forth

Consequences and corollaries:

- Any MSOL[S] formula can be written as one in 'automata normal form'.
- General translation of logic to automata: hard. In general, it has been shown that time-complexity of *any algorithm* converting MSOL[S] $\rightarrow$ automata *cannot be bounded by any elementary function* (there is an unbounded family of formulæ the corresponding automaton's number of states grows with length of formula like $2 \uparrow\uparrow n = \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{n}$. That's a hefty lower bound.

2.6 And so on

So, MSOL is a nice way to describe languages that can be accepted by FSAs.

2.6 And so on

So, MSOL is a nice way to describe languages that can be accepted by FSAs. Translating in general is hard, but... most things that can be said in logic are not things we want to say.


Thanks!