Logic: Unambiguous Description of Problems

Nicholas Mainardi$^1$

Dipartimento di Elettronica e Informazione
Politecnico di Milano

nicholas.mainardi@polimi.it

April 18, 2018

$^1$Partly Based on Alessandro Barenghi’s material, enriched with some additional exercises
Logic as an unambiguous description

Formalization and logic

- It is useful to provide an *unambiguous* description of a problem
- The oldest, and most widely understood method is to employ the language of logic
- Formal mathematical logic is a way of describing properties holding for a set of objects
- We will understand the basic building blocks of logic and we will see how to formalise facts
Monadic First Order Logic (MFO) is used to describe languages.

Syntax of a formula

- The variables are defined on a finite subset of \( \mathbb{N} \), and they represent indexes on the string being described, starting from 0.
- A generic formula \( \varphi \) may be:
  - \( \neg \varphi \)
  - \( \varphi_1 \land \varphi_2 \)
  - \( \forall x(\varphi) \)
  - \( a(x) \), a predicate which exists for each letter of the alphabet of the language, which is true if the \( x \)th letter is \( a \)
  - \( x < y \), that is the variable \( x \) is strictly less than the variable \( y \)
Derivations and Helper Predicates

Derivations
- $\varphi_1 \lor \varphi_2 \equiv \neg(\neg \varphi_1 \land \neg \varphi_2)$
- $\varphi_1 \implies \varphi_2 \equiv \neg \varphi_1 \lor \varphi_2$
- $\exists x(\varphi) \equiv \neg \forall x(\neg \varphi)$

Helper Predicates
- $x = y \equiv \neg(x < y) \land \neg(y < x)$
- $x \leq y \equiv \neg(y < x)$
- zero constant: $x = 0 \equiv \forall y(x \leq y)$
- successor predicate: $y = S(x) \equiv x < y \land \neg \exists z(x < z \land z < y)$
- numerical constants: $1 = S(0), 2 = S(1)\ldots$
- $\text{last}(x) \equiv \neg \exists y(x < y)$
Monadic Logic

First Order Logic

Program Specifications

---

**Notation and Warm-up Examples**

**Concise Notation**

- We can write \( y = x + 1 \) as replacement for \( y = S(x) \).
- Similarly, \( y = x + k \) as a replacement for \( y = a^k \cdot b^k \), with \( a = S( \text{ and } b =) \).
- Conversely, \( y = x - 1 \) as replacement for \( x = S(y) \).
- \( y = x - k \) as a replacement for \( x = y + k \).

**Warm-up**

- Words starting by \( a \) with at least three \( b \):
  \[
  a(0) \land \exists x_1, x_2, x_3 (\neg (x_1 = x_2) \land \neg (x_2 = x_3) \land \neg (x_1 = x_3) \land b(x_1) \land b(x_2) \land b(x_3))
  \]
- The language \( L = a^+ b^+ \):
  \[
  \exists l (\text{last}(l) \land b(l) \land \exists x (\neg (x = 0) \land b(x) \land \\
  \forall y (y < x \Rightarrow a(y)) \land \forall z ((x < z \land z < l) \Rightarrow b(z))))
  \]
Let’s consider again the Vasco’s lyrics: $L = (e^+ h | la)^+$. Can we specify it with a MFO formula?

- We may think it is not possible since MFO is not close w.r.t. Kleene $\ast$ operator.
- However, being not closed does not mean that every star-language cannot be specified.

Idea: We can write a formula by specifying possible successors for each possible letter in the alphabet. Then, if we specify the first and last characters, these constraints will be satisfied only by strings in $L$. 
Languages For MFO

An MFO formula for \( L = (e^+ h | la)^+ \)

- \( \forall x(e(x) \Rightarrow \exists y(y = x + 1 \land (e(y) \lor h(y)))) \). Each \( e \) is followed by \( e \) or \( h \)
- \( \forall x(l(x) \Rightarrow \exists y(y = x + 1 \land a(y))) \). An \( l \) can be followed only by \( a \)
- \( \forall x(h(x) \land \neg(last(x)) \Rightarrow \exists y(y = x + 1 \land (e(y) \lor l(y)))) \). An \( h \) can be followed by \( e \) or \( l \), if it is not the last character
- \( \forall x(a(x) \land \neg(last(x)) \Rightarrow \exists y(y = x + 1 \land (e(y) \lor l(y)))) \). An \( a \) can be followed by \( e \) or \( l \), if it is not the last character
- \( e(0) \lor l(0) \). Specifying that the first character cannot be \( h \) or \( a \)
- \( \exists x(last(x) \land (h(x) \lor a(x))) \). Specifying the last character

As acknowledged during lecture, the last rule is not strictly required.
Monadic Second Order Logic

**MFO ⊂ REG**

Since MFO is not closed w.r.t. Kleene $\ast$ operator, not all the regular languages can be expressed with an MFO formula.

- The problem is that we cannot specify structural properties which are independent from finite positions or successors!
- Example: The language $L = \{a^{2k+1} \mid k \geq 0\}$

**MSO**

We can describe all regular languages with a second order logic, that is allowing quantifiers on predicates:

$$\exists P(0) \land \forall x ((\neg P(x) \iff P(x+1)) \land a(x) \land (last(x) \Rightarrow P(x))))$$
\[ L = \{ a^{3k} \mid k \geq 0 \} \]

- A First order or second order formula?
- It seems a second order formula is needed, since we cannot describe an index being multiple of 3 without defining a predicate

**Idea:** Recursive definition!

\[ \exists P(P(2) \land \exists k(last(k) \land P(k)) \land \forall x (\)

\[ (P(x) \land \neg last(x)) \Rightarrow P(x + 3) \land \forall y ((x < y \land y < x + 3) \Rightarrow \neg P(y)) \]

\[ ) \land (x < 2 \Rightarrow \neg P(x)) \land a(x))) \]

- Issues?
- The languages includes also the empty string \( \epsilon \), but the formula is not satisfied by this string!
The Empty String

Consider a language over the alphabet \( I \). The empty string can be defined with a MFO formula as:

- \( \forall x ( \bigwedge_{i \in I} \neg i(x)) \)
- That is, there are no characters in the string

If a language includes the empty string, we can add it as an alternative to the formula defining the language:

\[
\forall x (\neg a(x)) \lor \exists P(P(2) \land \exists k(last(k) \land P(k)) \land \forall x( (P(x) \land \neg last(x)) \Rightarrow P(x + 3) \land \\
\forall y((x < y \land y < x + 3) \Rightarrow \neg P(y))) \land (x < 2 \Rightarrow \neg P(x)) \land a(x)))
\]

- \( \forall x (\neg a(x)) \) is True iff there are no characters in the string, that is in case the string is \( \epsilon \)
- it is false, and thus irrelevant, otherwise
It is always possible to move from a FSA to an MSO formula:

- Consider a FSA with $Q = \{q_0, q_1, \ldots, q_K\}$ and an input alphabet $I$.
- There exists a predicate $i(x), \forall i \in I$.
- We can write an MSO formula with this structure:

$$
\exists Q_0, Q_1, \ldots, Q_K (\forall z (\bigwedge_{i \neq j} (\neg Q_i(z) \land Q_j(z))) \land Q_0(0) \land \\
\forall x (\neg (\text{last}(x)) \Rightarrow (\bigvee_{\delta(q_i,i)=q_j} Q_i(x) \land i(x) \land Q_j(x + 1)) \land \\
\text{last}(x) \Rightarrow (\bigvee_{\delta(q_i,i)=q_j, q_j \in F} Q_i(x) \land i(x))))
$$
FSA to MSO

**MSO**

\[ \exists Q_0, Q_1, Q_2, Q_3 (\forall z (\neg (Q_0(z) \land Q_1(z)) \land \\
(\neg (Q_0(z) \land Q_2(z)) \land (\neg (Q_1(z) \land Q_3(z)) \land \\
(\neg (Q_2(z) \land Q_3(z))) \land Q_0(0) \land (\forall x (\neg (last(x)) \Rightarrow \\
(Q_0(x) \land a(x) \land Q_1(x + 1)) \lor \\
Q_0(x) \land a(x) \land Q_1(x + 1)) \lor \\
Q_1(x) \land a(x) \land Q_1(x + 1)) \lor \\
Q_1(x) \land b(x) \land Q_2(x + 1)) \lor \\
Q_2(x) \land b(x) \land Q_2(x + 1)) \lor \\
Q_2(x) \land c(x) \land Q_3(x + 1)) \lor \\
Q_2(x) \land c(x) \land Q_3(x + 1)) \land \\
last(x) \Rightarrow (Q_1(x) \land b(x)) \lor \\
Q_2(x) \land b(x) \lor \\
Q_2(x) \land c(x) \lor \\
Q_3(x) \land c(x))) ) ]

**FSA**

- Start state: \( q_0 \)
- States: \( q_0, q_1, q_2, q_3 \)
- Transitions:
  - \( q_0 \) to \( q_1 \) on \( a \)
  - \( q_1 \) to \( q_0 \) on \( b \)
  - \( q_2 \) to \( q_3 \) on \( c \)
  - \( q_3 \) to \( q_2 \) on \( b \)
Length Fields

Describe the set of strings on the alphabet $I = \{a, b\}$ which are a sequence of substrings, each prefixed by a digit stating its length.

- $2ab3aaa6abbaba, 3aba5bbbb0, 4abaa05abba$ are valid strings.
- $2a3aba, 3abab, 1aba$ are not valid strings.
- First order or second order?
Length Fields

Describe the set of strings on the alphabet $\mathbf{I} = \{a, b\}$ which are a sequence of substrings, each prefixed by a digit stating its length.

- $2ab3aaa6abbaba, 3aba5bbbbb0, 4abaa05abbba$ are valid strings.
- $2a3aba, 3abab, 1aba$ are not valid strings.

First order or second order?

The constraint is that if there is a digit $i$, then the subsequent $i$ characters have to be $a$ or $b$, with the $i + 1$ one being another digit (unless the string is finished).
Length Fields

Describe the set of strings on the alphabet \( I = \{a, b\} \) which are a sequence of substrings, each prefixed by a digit stating its length

- \( 2ab3aaa6abbaba, 3aba5bbbb0, 4abaa05abbba \) are valid strings
- \( 2a3aba, 3abab, 1aba \) are not valid strings

First order or second order?

The constraint is that if there is a digit \( i \), then the subsequent \( i \) characters have to be \( a \) or \( b \), with the \( i + 1 \) one being another digit (unless the string is finished)

\( \rightarrow \) We can express it with successors, thus first order seems to be sufficient!
Length Fields

The structure of the formula: \( \forall x (\bigwedge_{k=0}^{9} k(x) \Rightarrow (\forall y (x < y \leq x + k \Rightarrow a(y) \lor b(y)) \land \neg a(x + k + 1) \land \neg b(x + k + 1))) \)

The actual formula with some cases explicitly stated is \( \forall x : \)

- \( 0(x) \Rightarrow (\forall y (x < y \leq x \Rightarrow a(y) \lor b(y)) \land \neg a(x + 1) \land \neg b(x + 1)) \)
- \( 1(x) \Rightarrow (\forall y (x < y \leq x + 1 \Rightarrow a(y) \lor b(y)) \land \neg a(x + 2) \land \neg b(x + 2)) \)
- \( 2(x) \Rightarrow (\forall y (x < y \leq x + 2 \Rightarrow a(y) \lor b(y)) \land \neg a(x + 3) \land \neg b(x + 3)) \)
- \( 3(x) \Rightarrow (\forall y (x < y \leq x + 3 \Rightarrow a(y) \lor b(y)) \land \neg a(x + 4) \land \neg b(x + 4)) \)

- \( \ldots \)

- \( 9(x) \Rightarrow (\forall y (x < y \leq x + 9 \Rightarrow a(y) \lor b(y)) \land \neg a(x + 10) \land \neg b(x + 10)) \)
\[ L = a^* b^* c^* \]

We can describe this language by specifying possible successors for each character of the alphabet:

- \( \forall x (b(x) \land \neg last(x) \Rightarrow b(x + 1) \lor c(x + 1)) \). If a \( b \) is not the last character, it can be followed by a \( b \) or a \( c \)
- \( \forall x (c(x) \land \neg last(x) \Rightarrow c(x + 1)) \). If a \( c \) is not the last character, it can be followed only by a \( c \)
- We do not need to specify first and last characters, since they can be all possible characters of the alphabet
- \( a \) can be followed by any character in the alphabet, thus we do not need to specify a constraint on its possible successors
- The language accepts also the empty string, thus we need to add it as an alternative. Complete formula:

\[
\forall x ((\neg a(x) \land \neg b(x) \land \neg c(x)) \lor ((b(x) \land \neg last(x) \Rightarrow b(x + 1) \lor c(x + 1)) \land (c(x) \land \neg last(x) \Rightarrow c(x + 1))))
\]
\[ L = \{ y \in \{a, b\}^* \mid \#a = 2k \land \#b = 2h + 1 \land h, k \geq 0 \} \]

- We need two predicates, one for the parity of \(a\), call it \(A\), and one for the parity of \(b\), call it \(B\).
- As for FSA to MSO algorithm, \(A(y)\) is the truth value of the predicate \(A\) after \(y\) characters being read.
- We consider all the possible combinations of predicates truth values and input characters and specify how the truth value of the predicate changes. \(\exists A, B(\forall x \neg \text{last}(x) \Rightarrow (:\)
  - \(A(x) \land B(x) \land a(x) \Rightarrow \neg A(x + 1) \land B(x + 1) \land\)
  - \(A(x) \land B(x) \land b(x) \Rightarrow A(x + 1) \land \neg B(x + 1) \land\)
  - \(A(x) \land \neg B(x) \land a(x) \Rightarrow \neg A(x + 1) \land \neg B(x + 1) \land\)
  - \(A(x) \land \neg B(x) \land b(x) \Rightarrow A(x + 1) \land B(x + 1) \land\)
  - \(\neg A(x) \land B(x) \land a(x) \Rightarrow A(x + 1) \land B(x + 1) \land\)
  - \(\neg A(x) \land B(x) \land b(x) \Rightarrow \neg A(x + 1) \land \neg B(x + 1) \land\)
  - \(\neg A(x) \land \neg B(x) \land a(x) \Rightarrow A(x + 1) \land \neg B(x + 1) \land\)
  - \(\neg A(x) \land \neg B(x) \land b(x) \Rightarrow \neg A(x + 1) \land B(x + 1) \land\)
- Lastly, \((a(0) \land \neg A(0) \land \neg B(0) \lor b(0) \land A(0) \land B(0)) \land \exists y (\text{last}(y) \land A(y) \land B(y)))\)
A First order or second order formula?

Can we specify it by using successor relationships only?

Idea: Let’s handle the 3 cases separately:

1. \((a(0) \land b(1) \land \forall x (b(x) \land \neg \text{last}(x) \Rightarrow a(x + 1)) \land \forall y (a(y) \Rightarrow b(y + 1)))\)

2. \((a(0) \land a(1) \land b(2) \land \forall x (a(x) \land b(x + 1) \Rightarrow (b(x + 2) \land (\neg \text{last}(x + 2) \Rightarrow (a(x + 3) \land a(x + 4) \land b(x + 5))))))\)

3. \((a(0) \land a(1) \land a(2) \land b(3) \land \forall x (a(x) \land b(x + 1) \Rightarrow (b(x + 2) \land b(x + 3) \land \neg \text{last}(x + 3) \Rightarrow (a(x + 4) \land a(x + 5) \land a(x + 6) \land b(x + 7))))))\)
First order logic

Basics

- **term**: the basic atom of a logic proposition (i.e. a variable)
  - A term may be a variable, whose domain has to be specified, a constant or a function
  - Generally, the set of functions which may be used is provided
- **predicate**: a symbol which represents a relationship among its terms. If such a relation holds, the truth value of the symbol is **True**
- **formula**: a well-formed assembly of terms, predicates and logic signs
- **sign**: a logic operator with a specific truth table
First order logic

**Signs**

- $\land$ (AND): the formula $\text{term} \land \text{term}$ is true only if both terms are
- $\lor$ (OR): the formula $\text{term} \land \text{term}$ is true if at least a term is
- $\neg$ (negation): the formula $\neg \text{term}$ is true if term is not
- $\forall$ or $\oplus$: the formula $\text{term} \forall \text{term}$ (or $\text{term} \oplus \text{term}$) is true if exactly one term is
- $\forall$ : $\forall \text{term(formula)}$ indicates that the formula is true only if it is true for all the possible instances of term
- $\exists$ : $\exists \text{term(formula)}$ indicates that the formula is true if it is true for at least an instance of term
First order logic

More Basics

- \( \Rightarrow : t_1 \Rightarrow t_2 \) is just \( \neg t_1 \lor t_2 \) : the implication is true if the premise is false or both premise and consequence are true.

- \( \Leftarrow : \) by contrast \( t_1 \Leftarrow t_2 \) is simply \( t_1 \lor \neg t_2 \).

- \( \iff : \) \( t_1 \iff t_2 \) is a shortcut for both of the above.

- \( \forall x(\neg t_1(x)) \equiv \neg \exists x(t_1(x)) \) : Effects of negation on quantifiers.

Common sets are:

- \( \mathbb{N} \) : the set of all integers \( \geq 0 \)
- \( \mathbb{Z} \) : the set of all integers
- \( \mathbb{Q} \) : the set of rationals (i.e. \( q \in \mathbb{Q} \iff q = \frac{z_1}{z_2}, z_1, z_2 \in \mathbb{Z} \))
- \( \mathbb{R} \) : the set of real numbers
- \( \emptyset \) : the empty set

- Is \( \forall i \in \emptyset(i = i + 1) \) true or false? **True:** although the consequence is false, there is no \( i \) belonging to the empty set.
To brush the dust away from first order logic, we will now prove that $x = x$ holds.

We do remember that $x = x$ can be defined as $\forall x (x \iff x)$.

Expanding the $\iff$ sign we obtain: $\forall x ((x \implies x) \land (x \impliedby x))$.

Further expanding the implication yields: $\forall x ((\neg x \lor x) \land (x \lor \neg x))$.

Knowing that $(x \lor \neg x) = \top^a$ we can simplify the previous to $\forall x, \top \land \top$.

Which in turn is $\forall x, \top$, i.e. the formula is true for any $x$

---

$^a$Hamlet, Act III, Scene I
Define a first order logic formula specifying the structure of strings in the language $L$, which are arbitrary strings over $\{a, b, c\}^*$, whose last character is $f$. Moreover:

- If the string $x$ contains at least an $a$, then $\#b(x) = \#c(x)$
- If the string $x$ has no character $a$, then $\#b(x) = \#c(x) + 10$

**Logic Formula: Helper Functions**

We define a function $\#b(x)$ for a string $x$, which counts the occurrences of $b$ in $x$:

$$\forall x, y (x = \epsilon \Rightarrow \#b(x) = 0 \land (x = y.a \lor x = y.c \Rightarrow \#b(x) = \#b(y)) \land (x = y.b \Rightarrow \#b(x) = \#b(y) + 1))$$

Similarly, we define a function $\#c(x)$ for a string $x$, counting the occurrences of $c$ in $x$. 
Defining a Language

Logic Formula for $L$

- First, we define the language $L_1$, which is $\{a, b, c\}^*$:
  $$\forall x (x \in L_1 \iff x = \epsilon \lor \exists y (y \in L_1 \land (x = a.y \lor x = b.y \lor x = c.y)))$$

- Then, we define strings of $L$ by using $L_1$ and the 2 helper functions previously defined:
  $$\forall x (x \in L \iff \exists y (y \in L_1 \land x = y.f) \land (\exists z, w (x = z.a.w.f \land z \in L_1 \land w \in L_1) \Rightarrow \#b(x) = \#c(x)) \land (\neg \exists z, w (x = z.a.w.f \land z \in L_1 \land w \in L_1) \Rightarrow \#b(x) = \#c(x) + 10))$$
First order logic

Defining basic properties

- We will define “p is a prime”
- First of all, we need to define the domain over which the property holds: $p \in \mathbb{N} \setminus \{0\}$
- Now we want to formalise the classic definition “a number $p \in \mathbb{N}, p > 1$ is prime if it is divisible only by 1 and itself”:
  - $\forall p \in \mathbb{N} (Prime(p) \iff (\neg \exists f_1 \in \mathbb{N} \setminus 1, p(f_1 | p)))$
- Willing to avoid the $|$ (divides) symbol, the same concept can be expressed as:
  - $\forall p \in \mathbb{N} (Prime(p) \iff (\neg \exists f_1, f_2 \in \mathbb{N} \setminus 1, p(f_1 f_2 = p)))$
Preconditions and Post-conditions of a program

What are Pre-s and Post-s

- Preconditions of an algorithm (a program) are the prerequisites on the inputs in order for the algorithm to work properly.

- Postconditions of an algorithm (a program) are the properties on the output which are warranted to hold by the execution of the algorithm.

- Advantages: it is possible to daisy chain properties of algorithms executed in a sequence to prove that the output of a complex system is exactly what we want.
A first example

- Let’s start simple: An algorithm $A(n, m)$ outputs $z = 1$ if $n$ and $m$ are co-prime, 0 otherwise.
- 2 integers $n, m$ are coprime if the only positive integer factor they share is 1, or equivalently, if $gcd(n, m) = 1$.
- Preconditions: $n, m \in \mathbb{N}$ (really needed?)
- Before hitting the Post-conditions, let’s define a shortcut: $\text{coprime}(n, m) \iff \neg \exists k, f_1, f_2 (k \in \mathbb{N} \setminus 0, 1 \land f_1 \in \mathbb{N} \land f_2 \in \mathbb{N} \land kf_1 = n \land kf_2 = m)$
- Instinctively, Postcondition: $\text{coprime}(n, m) \rightarrow z = 1$
- but this does not bind $z \neq 1$ if $n$ and $m$ are not coprime!
Specifying Pre-s and Post-s

A first example

- **Second try**: Postcondition: \( \text{coprime}(n, m) \iff z = 1 \)
  - Ok, now \( z \) is not 1, but it may not be 0 too.

- **Third try**: Postcondition:
  \( \text{coprime}(n, m) \iff z = 1 \lor z = 0 \)
  - but goto end; return 0; satisfies this!

- **Getting it right**: Postcondition:
  \( (\text{coprime}(n, m) \land z = 1) \lor (\neg \text{coprime}(n, m) \land z = 0) \)

- alternatively:
  \( (\text{coprime}(n, m) \iff z = 1) \land (\neg z = 1 \Rightarrow z = 0) \)
Specifying Pre-s and Post-s

Slightly more difficult

- A slightly different version: \( A(n, o, m) \) outputs \( z = 1 \) if a number is a product of the other two, 0 otherwise
- Preconditions: True
- Postconditions: \( (z = 1 \land (m = no \lor n = mo \lor o = mn)) \lor (z = 0 \land (m \neq no \land n \neq mo \land o \neq mn)) \)
- Basically, the postconditions are represented by ( output_value \( \land \) property) for all outputs.
- This is reasonably convenient if outputs are finite
Specifying Pre-s and Post-s

Infinite output domain

- An example without a finite codomain: \( A(x, y) \) outputs 
  \[ z = \lfloor \log_x(y) \rfloor \]
- Preconditions: \( True \)
- First try: Post: \( (x^z = y) \lor (x^z < y \land x^{z+1} > y) \)
- A bit redundant, and misses the fact that \( z \) must be an integer
- \( z \in \mathbb{N} \land (x^z \leq y \land x^{z+1} > y) \)
Specifying Pre-s and Post-s

Least Common Multiple

\[ A(x, y) = z, \text{ where } z \text{ is } \text{lcm}(x, y) \]

- Preconditions: \( x, y \neq 0 \), since \( \text{lcm}(0, a) \) is generally undefined
  - Alternatively, the precondition can be True but in the postcondition we need to specify that \( x = 0 \lor y = 0 \Rightarrow z = \bot \)
- Let’s define an helper predicate: defining the common multiple of 2 numbers
  - \( \text{CommonMultiple}(x, y, z) \iff \exists k_1, k_2 \in \mathbb{N}(z = k_1x \land z = k_2y) \)
- We can now define the \( \text{lcm}(x, y) = z \) to be the minimum among the common multiples:
  - Postconditions: \( \text{CommonMultiple}(x, y, z) \land \neg \exists h \in \mathbb{N}(h < z \land \text{CommonMultiple}(x, y, h)) \)
**Specifying Pre-s and Post-s**

**Question:** Is the algorithm always returning 0 compliant to our specification?

- CommonMultiple\((x, y, 0)\) is true for every possible pair \(x, y\)
  
  \[ \text{Just sufficient to choose } k_1 = 0 \text{ and } k_2 = 0 \]

- Therefore, 0 is always the least common multiple, thus the algorithm always returning 0 is the only one which is compliant to our specification

**How to fix this issue?** Change the definition of CommonMultiple\((x, y, z)\) predicate:

- CommonMultiple\((x, y, z) \iff \exists k_1, k_2 \in \mathbb{N} \setminus 0 (z = k_1x \land z = k_2y)\)

- Now, \(z\) cannot be 0 unless \(x = 0 \lor y = 0\), which is however impossible because of the preconditions
A whole system

System structure

- We will now specify the state of accessibility of a medieval castle during a siege:
  - Two separate constraints are keeping the castle inaccessible: a portcullis and drawbridge
  - To render the castle accessible, two besiegers should operate simultaneously the two cranes to open the portcullis and lower the drawbridge within a time interval $\delta$
  - Once the castle is accessible, the besiegers are killed by the besieged after a finite time $K$
  - The operation on the portcullis and drawbridge are assumed instantaneous
  - The time domain is assumed to be continuous
Step 1: identify the events

- The first step is considering over which domain we will define the logic formulas
- We only need a continuous time axis, \( t \in \mathbb{R}^+ \) will suffice
- The first step in modelling the system is identifying the events and states (both defined as logic predicates)
  - \( \text{RaisePC}(t) : \) the portcullis is risen at time \( t \) by a besieger
  - \( \text{LowerDB}(t) : \) the drawbridge is lowered at time \( t \) by a besieger
  - \( \text{SiegeBroken}(t) : \) the castle is accessible at time \( t \)
  - \( \text{CastleSafe}(t) : \) the castle is inaccessible at time \( t \)
  - \( \text{BreakSiege}(t) : \) The castle is made accessible at time \( t \)
  - \( \text{RetakeCastle}(t) : \) The castle is made inaccessible at time \( t \)
- Since we are modelling a generic behaviour, an implicit \( \forall t \) quantifier will be binding all formulas
A whole system

Step 2: a first rough description

- A first tentative description of the system is \((t_1, t_2 \in \mathbb{R}^+\) for the rest of the slides):
  - \(\exists t_1 \ (t_1 < t \land \text{BreakSiege}(t_1) \land (\forall t_2 \ (t_1 < t_2 < t \Rightarrow \neg \text{RetakeCastle}(t_2))) \Rightarrow \text{SiegeBroken}(t)\) : if the castle has been taken in \(t_1\) and no-one made it safe again between \(t_1\) and \(t\), the siege is broken at \(t\)
  - \(\exists t_1 \ (t_1 < t \land \text{RetakeCastle}(t_1) \land (\forall t_2 \ (t_1 < t_2 < t \Rightarrow \neg \text{BreakSiege}(t_2))) \Rightarrow \text{CastleSafe}(t)\) : if the castle was safe in \(t_1\) and no-one broke the siege between \(t_1\) and \(t\) the castle is still safe
  - \(\text{RaisePC}(t) \land \exists t_1 \ (\text{LowerDB}(t_1) \land (t_1 \leq t) \land (t - t_1 \leq \delta)) \Rightarrow \text{BreakSiege}(t)\) : if the portcullis is risen within \(\delta\) from when the drawbridge is lowered, the siege is broken
  - \(\forall t_2 \ (t - K < t_2 \leq t \Rightarrow \text{SiegeBroken}(t_2)) \Rightarrow \text{RetakeCastle}(t)\)
A whole system

Issues

- The castle may be safe and non-safe in the same time instant $t$
- Nothing is forbidding the siege to self-break
- Or the castle to self repel the siege, for that matter
- The portcullis must be risen after the drawbridge is lowered
A whole system

Step 3: Fix the issues

- \( CastleSafe(t) \iff \neg SiegeBroken(t) \): Fixes Schroedinger’s castle issue
- \( \exists t_1 \ (t_1 < t \land BreakSiege(t_1) \land (\forall t_2 (t_1 < t_2 < t \Rightarrow \neg RetakeCastle(t_2))) \iff SiegeBroken(t) \): fixes the self-siege break issue
- \( \exists t_1 (t_1 < t \land RetakeCastle(t_1) \land (\forall t_2 (t_1 < t_2 < t \Rightarrow \neg BreakSiege(t_2))) \iff CastleSafe(t) \): fixes the self-heal issue item
- \( (RaisePC(t) \land \exists t_1 (LowerDB(t_1) \land t_1 \leq t \land (t - t_1 \leq \delta))) \lor (LowerDB(t) \land \exists t_1 (RaisePC(t_1) \land t_1 \leq t \land (t - t_1 \leq \delta))) \iff BreakSiege(t) \): copes with the forced ordering of the besieging actions
- \( \forall t_2 (t - K < t_2 \leq t \Rightarrow SiegeBroken(t_2)) \iff RetakeCastle(t) \): fixes the last of self-healing properties
A whole system

More Issues

- What if $\delta \gg K$? Consider this sequence of events:
  - The drawbridge is lowered at $t_1$
  - The portcullis is risen at $t_2$, with $t_1 < t_2 < t_1 + \delta$, thus the siege is broken at $t_2$
  - The castle is retaken at $t_2 + K$
  - The portcullis is risen at $t_3 > t_2 + K$
  - If $t_3 - t_1 < \delta$, then the siege is broken again, even if the drawbridge has not been lowered twice
  - But the drawbridge is presumably risen during retaking, thus it does not make sense that the castle is sieged without lowering again the drawbridge

- What about initial conditions? Presumably the castle is safe at $t = 0$, but from our specification it implies that it has been retaken in $t_1 < t$, which cannot happen
A whole system

Step 4: Fix more issues

- $\text{CastleSafe}(t) \Leftrightarrow \neg \text{SiegeBroken}(t)$
- $\exists t_1 \ (t_1 < t \land \text{BreakSiege}(t_1) \land (\forall t_2 (t_1 < t_2 < t \Rightarrow \neg \text{RetakeCastle}(t_2))) \Leftrightarrow \text{SiegeBroken}(t)$
- $(\exists t_1 (t_1 < t \land \text{RetakeCastle}(t_1) \land (\forall t_2 (t_1 < t_2 < t \Rightarrow \neg \text{BreakSiege}(t_2)))) \lor \neg \exists t_1 (t_1 < t \land \text{BreakSiege}(t_1)) \Leftrightarrow \text{CastleSafe}(t)$: the castle is safe also if the siege has never been broken
- $((\text{RaisePC}(t) \land \exists t_1 (\text{LowerDB}(t_1) \land t_1 \leq t \land t - t_1 \leq \delta)) \land \forall t_2 (t_1 \leq t_2 \leq t \Rightarrow \text{CastleSafe}(t_2))) \lor (\text{LowerDB}(t) \land \exists t_1 (\text{RaisePC}(t_1) \land t_1 \leq t \land t - t_1 \leq \delta) \land \forall t_2 (t_1 \leq t_2 \leq t \Rightarrow \text{CastleSafe}(t_2))) \Leftrightarrow \text{BreakSiege}(t)$: fixes siege double breaks
- $\forall t_2 \ (t - K < t_2 \leq t \Rightarrow \text{SiegeBroken}(t_2)) \Leftrightarrow \text{RetakeCastle}(t)$
A Transducer

We want to describe the behavior of the following transducer:

The transducer receives a sequence of (possibly repeated) characters which is terminated by the $ symbol, translating it into a $-terminated sequence with no repetitions which contains all and only symbols found in the input sequence

- Example: abcdheeefttdafscbs$ \land abcdhefts$, or dhfacsbte$

- Suppose there exists two functions $in(i)$, $out(i)$, which returns the $i$ – $th$ character of the, respectively, input and output string

First, let’s define two helper predicates:

- $InputLen(k) \leftrightarrow \exists k(in(k) = $ \land \forall i(i \in \mathbb{N} \land i < k \Rightarrow in(i) \neq $)$)
- $OutLen(k) \leftrightarrow \exists k(out(k) = $ \land \forall i(i \in \mathbb{N} \land i < k \Rightarrow out(i) \neq $))$
A Transducer

- First trial:
  - $\exists h, k (\text{InputLen}(h) \land \text{OutLen}(k) \land$
    - $\forall i ((i \in \mathbb{N} \land i < k) \Rightarrow \exists j (j \in \mathbb{N} \land j < h \land \text{out}(i) = \text{in}(j))))$
A Transducer

First trial:
- $\exists h, k (InputLen(h) \land OutLen(k) \land \forall i ((i \in \mathbb{N} \land i < k) \Rightarrow \exists j (j \in \mathbb{N} \land j < h \land out(i) = in(j))))$

Issues?
A Transducer

- First trial:
  - $\exists h, k (\text{InputLen}(h) \land \text{OutLen}(k) \land \forall i ((i \in \mathbb{N} \land i < k) \Rightarrow \exists j (j \in \mathbb{N} \land j < h \land \text{out}(i) = \text{in}(j))))$

- Issues?

- Characters in the output string may be repeated!
A Transducer

- First trial:
  - $\exists h, k (\text{InputLen}(h) \land \text{OutLen}(k) \land$
    $\forall i ((i \in \mathbb{N} \land i < k) \Rightarrow \exists j (j \in \mathbb{N} \land j < h \land \text{out}(i) = \text{in}(j))))$

- Issues?

- Characters in the output string may be repeated!

- Second Trial:
  - $\exists h, k (\text{InputLen}(h) \land \text{OutLen}(k) \land$
    $\forall i ((i \in \mathbb{N} \land i < k) \Rightarrow \exists j (j \in \mathbb{N} \land j < h \land \text{out}(i) = \text{in}(j))))$
  - $\exists k (\text{OutLen}(k) \land \forall i, j ((i \in \mathbb{N} \land j \in \mathbb{N} \land i < k \land j < k \land i \neq j) \Rightarrow \text{out}(i) \neq \text{out}(j)))$
A Transducer

- First trial:
  - $\exists h, k (\text{InputLen}(h) \land \text{OutLen}(k) \land$
  - $\forall i ((i \in \mathbb{N} \land i < k) \Rightarrow \exists j (j \in \mathbb{N} \land j < h \land \text{out}(i) = \text{in}(j))))$

- Issues?

- Characters in the output string may be repeated!

- Second Trial:
  - $\exists h, k (\text{InputLen}(h) \land \text{OutLen}(k) \land$
  - $\forall i ((i \in \mathbb{N} \land i < k) \Rightarrow \exists j (j \in \mathbb{N} \land j < h \land \text{out}(i) = \text{in}(j))))$
  - $\exists k (\text{OutLen}(k) \land \forall i, j ((i \in \mathbb{N} \land j \in \mathbb{N} \land i < k \land j < k \land i \neq j) \Rightarrow \text{out}(i) \neq \text{out}(j)))$

- More issues?
A Transducer

- First trial:
  - \( \exists h, k (\text{InputLen}(h) \land \text{OutLen}(k) \land \forall i ((i \in \mathbb{N} \land i < k) \Rightarrow \exists j (j \in \mathbb{N} \land j < h \land \text{out}(i) = \text{in}(j)))) \)

- Issues?
- Characters in the output string may be repeated!

- Second Trial:
  - \( \exists h, k (\text{InputLen}(h) \land \text{OutLen}(k) \land \forall i ((i \in \mathbb{N} \land i < k) \Rightarrow \exists j (j \in \mathbb{N} \land j < h \land \text{out}(i) = \text{in}(j)))) \)
  - \( \exists k (\text{OutLen}(k) \land \forall i, j ((i \in \mathbb{N} \land j \in \mathbb{N} \land i < k \land j < k \land i \neq j) \Rightarrow \text{out}(i) \neq \text{out}(j))) \)

- More issues?
- The first formula specifies that symbols of the output string must be found in the input string …
A Transducer

- First trial:
  - $\exists h, k (\text{InputLen}(h) \land \text{OutLen}(k) \land$
    $\forall i ((i \in \mathbb{N} \land i < k) \Rightarrow \exists j (j \in \mathbb{N} \land j < h \land \text{out}(i) = \text{in}(j))))$

- Issues?

- Characters in the output string may be repeated!

- Second Trial:
  - $\exists h, k (\text{InputLen}(h) \land \text{OutLen}(k) \land$
    $\forall i ((i \in \mathbb{N} \land i < k) \Rightarrow \exists j (j \in \mathbb{N} \land j < h \land \text{out}(i) = \text{in}(j))))$
  - $\exists k (\text{OutLen}(k) \land \forall i, j ((i \in \mathbb{N} \land j \in \mathbb{N} \land i < k \land j < k \land i \neq j) \Rightarrow \text{out}(i) \neq \text{out}(j)))$

- More issues?

- The first formula specifies that symbols of the output string must be found in the input string ...

- But there is no formula specifying that all the symbols in the input string must be found in the output one!
A transducer

Final Specification

- \( \exists h, k (\text{InputLen}(h) \land \text{OutLen}(k)) \land \)
  \( \forall i ((i \in \mathbb{N} \land i < k) \Rightarrow \exists j (j \in \mathbb{N} \land j < h \land \text{out}(i) = \text{in}(j))) \land \)
  \( \forall j ((j \in \mathbb{N} \land j < h) \Rightarrow \exists i (i \in \mathbb{N} \land i < h \land \text{out}(i) = \text{in}(j))) \)
- \( \exists k (\text{OutLen}(k) \land \forall i, j ((i \in \mathbb{N} \land j \in \mathbb{N} \land i < k \land j < k \land i \neq j) \Rightarrow \text{out}(i) \neq \text{out}(j))) \)

A Slight Addition

We may require that there are no more characters after $ in the input and output string (NB not a necessary requirement):

- \( \text{InputLen}(k) \iff \exists k (\text{in}(k) = \$ \land \forall i (i \in \mathbb{N} \land i < k \Rightarrow (\text{in}(i) \neq \$ \land \text{in}(i) \neq \epsilon) \land (i \in \mathbb{N} \land i > k \Rightarrow \text{in}(i) = \epsilon))) \)
- \( \text{OutLen}(k) \iff \exists k (\text{out}(k) = \$ \land \forall i (i \in \mathbb{N} \land i < k \Rightarrow (\text{out}(i) \neq \$ \land \text{out}(i) \neq \epsilon) \land (i \in \mathbb{N} \land i > k \Rightarrow \text{out}(i) = \epsilon))) \)
A Rail Crossing

We want to describe the train behavior and the control system design for a railway crossing

Simplifying Assumptions

- The bars raise and lower with no latencies
- One rail and one direction
- A train cannot enter the rail crossing if a previous train has not exited yet

Description of the System

- There is a photocell placed at a distance $d$ from the beginning of the rail crossing
- The distance between the beginning and the end of the rail crossing, that is the length of the bars, is $l$
- A train may have a speed between $[V_{min}, V_{max}]$
A Rail Crossing

Defining events:

- $Ar(t)$: The photocell signals a train is arriving
- $En(t)$: The train reaches the beginning of the rail crossing
- $Ex(t)$: The train reaches the end of the rail crossing

Defining states:

- $Down(t)$: The bars are down

Defining the following constants:

- $\tau^d_{\text{min}} = \frac{d}{V_{\text{max}}}$
- $\tau^d_{\text{max}} = \frac{d}{V_{\text{min}}}$
- $\tau^l_{\text{min}} = \frac{l}{V_{\text{max}}}$
- $\tau^l_{\text{max}} = \frac{l}{V_{\text{min}}}$
- $\tau^{d+l}_{\text{min}} = \frac{d+l}{V_{\text{max}}}$
- $\tau^{d+l}_{\text{max}} = \frac{d+l}{v_{\text{min}}}$
A Rail Crossing

Description of the train

- \( Ar(t) \Rightarrow (\exists t_1(t + \tau^d_{min} \leq t_1 \leq t + \tau^d_{max} \land En(t_1) \land \exists t_2(t_1 + \tau^l_{min} \leq t_2 \leq t_1 + \tau^l_{max} \land Ex(t_2))) ) \)
- \( En(t) \Rightarrow (\exists t_1(t - \tau^d_{max} \leq t_1 \leq t - \tau^d_{min} \land Ar(t_1) \land \exists t_2(t + \tau^l_{min} \leq t_2 \leq t + \tau^l_{max} \land Ex(t_2))) ) \)
- \( Ex(t) \Rightarrow (\exists t_1(t - \tau^l_{max} \leq t_1 \leq t - \tau^l_{min} \land En(t_1) \land \exists t_2(t_1 - \tau^d_{max} \leq t_2 \leq t_1 - \tau^d_{min} \land Ar(t_2))) ) \)

Rail Crossing Design

- \( Down(t) \Rightarrow \exists t_1(t - \tau^d_{max} \leq t_1 \leq t - \tau^d_{min} \land Ar(t_1)) \)
- \( Ar(t) \Rightarrow \forall t_1(t + \tau^d_{min} \leq t_1 \leq t + \tau^d_{max} \Rightarrow Down(t_1)) \)

Security Requirement

- \( \forall t(\exists t_1(t_1 \leq t \land En(t_1)) \land \neg \exists t_2(t_1 < t_2 \land Ex(t_2)) \Rightarrow Down(t) ) \)