Quantification definition

The **universal quantification** of predicate $P(x)$ over domain U is the statement "$P(x)$ for all values of x in the domain U" and is written $\forall x P(x)$ or $\forall x \in U\ P(x)$. When the domain is finite, universal quantification over the domain is equivalent to iterated *conjunction* (ands).

The **existential quantification** of predicate $P(x)$ over domain U is the statement "There exists an element x in the domain U such that $P(x)$" and is written $\exists x P(x)$ for $\exists x \in U\ P(x)$. When the domain is finite, existential quantification over the domain is equivalent to iterated *disjunction* (ors).

An element for which $P(x) = F$ is called a **counterexample** of $\forall x P(x)$.

An element for which $P(x) = T$ is called a **witness** of $\exists x P(x)$.

Quantification logical equivalence

Statements involving predicates and quantifiers are **logically equivalent** means they have the same truth value no matter which predicates (domains and functions) are substituted in.

Quantifier version of De Morgan's laws: $\boxed{\neg\forall x P(x) \ \equiv\ \exists x\,(\neg P(x))}$ $\boxed{\neg\exists x Q(x) \ \equiv\ \forall x\,(\neg Q(x))}$

Quantification examples finite domain

Examples of quantifications using $V(x), N(x), Mystery(x)$:

True or **False**: $\exists x\ (\ V(x) \wedge N(x)\ )$

True or **False**: $\forall x\ (\ V(x) \rightarrow N(x)\ )$

True or **False**: $\exists x\ (\ N(x) \leftrightarrow Mystery(x)\ )$

Rewrite $\neg\forall x\ (\ V(x) \oplus Mystery(x)\ )$ into a logical equivalent statement.

Notice that these are examples where the predicates have *finite* domain. How would we evaluate quantifications where the domain may be infinite?

Rna rnalen basecount definitions

Recall the definitions: The set of RNA strands S is defined (recursively) by:

Basis Step:	$\mathtt{A} \in S, \mathtt{C} \in S, \mathtt{U} \in S, \mathtt{G} \in S$
Recursive Step:	If $s \in S$ and $b \in B$, then $sb \in S$

where sb is string concatenation.

The function *rnalen* that computes the length of RNA strands in S is defined recursively by:

		$rnalen : S$	$\to \mathbb{Z}^+$
Basis Step:	If $b \in B$ then	$rnalen(b)$	$= 1$
Recursive Step:	If $s \in S$ and $b \in B$, then	$rnalen(sb)$	$= 1 + rnalen(s)$

The function *basecount* that computes the number of a given base b appearing in a RNA strand s is defined recursively by:

		$basecount : S \times B$	$\to \mathbb{N}$	
Basis Step:	If $b_1 \in B, b_2 \in B$	$basecount(\ (b_1, b_2)\ )$	$= \begin{cases} 1 & \text{when } b_1 = b_2 \\ 0 & \text{when } b_1 \neq b_2 \end{cases}$	
Recursive Step:	If $s \in S, b_1 \in B, b_2 \in B$	$basecount(\ (sb_1, b_2)\ )$	$= \begin{cases} 1 + basecount(\ (s, b_2)\ ) & \text{when } b_1 = b_2 \\ basecount(\ (s, b_2)\ ) & \text{when } b_1 \neq b_2 \end{cases}$	

Predicates example rnalen basecount

Using functions to define predicates:

L with domain $S \times \mathbb{Z}^+$ is defined by, for $s \in S$ and $n \in \mathbb{Z}^+$,

$$L(\ (s,n)\ ) = \begin{cases} T & \text{if } rnalen(s) = n \\ F & \text{otherwise} \end{cases}$$

In other words, $L(\ (s,n)\ )$ means $rnalen(s) = n$

BC with domain $S \times B \times \mathbb{N}$ is defined by, for $s \in S$ and $b \in B$ and $n \in \mathbb{N}$,

$$BC(\ (s,b,n)\ ) = \begin{cases} T & \text{if } basecount(\ (s,b)\ ) = n \\ F & \text{otherwise} \end{cases}$$

In other words, $BC(\ (s,b,n)\ )$ means $basecount(\ (s,b)\ ) = n$

Example where L evaluates to T: _____ Why?

Example where BC evaluates to T: _____ Why?

Example where L evaluates to F: _____ Why?

Example where BC evaluates to F: _____ Why?

$\exists t\ BC(t)$ $\exists (s,b,n) \in S \times B \times \mathbb{N}\ (basecount(\ (s,b)\ ) = n)$

In English:

Witness that proves this existential quantification is true:

$\forall t\ BC(t)$ $\forall (s,b,n) \in S \times B \times \mathbb{N}\ (basecount(\ (s,b)\ ) = n)$

In English:

Counterexample that proves this universal quantification is false:

Predicates projecting example rna basecount

New predicates from old

1. Define the **new** predicate with domain $S \times B$ and rule

   $$basecount(\ (s, b)\ ) = 3$$

   Example domain element where predicate is T:

2. Define the **new** predicate with domain $S \times \mathbb{N}$ and rule

   $$basecount(\ (s, \texttt{A})\ ) = n$$

   Example domain element where predicate is T:

3. Define the **new** predicate with domain $S \times B$ and rule

   $$\exists n \in \mathbb{N}\ (basecount(\ (s, b)\ ) = n)$$

   Example domain element where predicate is T:

4. Define the **new** predicate with domain S and rule

   $$\forall b \in B\ (basecount(\ (s, b)\ ) = 1)$$

   Example domain element where predicate is T:

Nested quantifiers

Nested quantifiers

$$\forall s \in S\ \forall b \in B\ \forall n \in \mathbb{N}\ (basecount(\ (s, b)\ ) = n)$$

In English:

Counterexample that proves this universal quantification is false:

$$\forall n \in \mathbb{N}\ \forall s \in S\ \forall b \in B\ (basecount(\ (s, b)\ ) = n)$$

In English:

Counterexample that proves this universal quantification is false:

Alternating quantifiers strategies rna examples

Alternating nested quantifiers

$$\forall s \in S\ \exists b \in B\ (\ basecount(\ (s, b)\ ) = 3\ )$$

In English: For each RNA strand there is a base that occurs 3 times in this strand.

Write the negation and use De Morgan's law to find a logically equivalent version where the negation is applied only to the BC predicate (not next to a quantifier).

Is the original statement **True** or **False**?

$$\exists s \in S\ \forall b \in B\ \exists n \in \mathbb{N}\ (\ basecount(\ (s, b)\ ) = n\ )$$

In English: There is an RNA strand so that for each base there is some nonnegative integer that counts the number of occurrences of that base in this strand.

Write the negation and use De Morgan's law to find a logically equivalent version where the negation is applied only to the BC predicate (not next to a quantifier).

Is the original statement **True** or **False**?

Tautology contradiction contingency examples

Label each of the following as a tautology, contradiction, or contingency.

$p \wedge p$

$p \oplus p$

$p \vee p$

$p \vee \neg p$

$p \wedge \neg p$

Why represent numbers

Modeling uses data-types that are encoded in a computer. The details of the encoding impact the efficiency of algorithms we use to understand the systems we are modeling and the impacts of these algorithms on the people using the systems. Case study: how to encode numbers?

Fixed width definition

Definition For b an integer greater than 1, w a positive integer, and n a nonnegative integer _____, the **base** b **fixed-width** w **expansion of** n is

$$(a_{w-1} \cdots a_1 a_0)_{b,w}$$

where $a_0, a_1, \ldots, a_{w-1}$ are nonnegative integers less than b and

$$n = \sum_{i=0}^{w-1} a_i b^i$$

Fixed width example

Decimal $b = 10$	Binary $b = 2$	Binary fixed-width 10 $b = 2$, $w = 10$	Binary fixed-width 7 $b = 2$, $w = 7$	Binary fixed-width 4 $b = 2$, $w = 4$
$(20)_{10}$				

Fixed width fractional definition

Definition For b an integer greater than 1, w a positive integer, w' a positive integer, and x a real number the **base b fixed-width expansion of x with integer part width w and fractional part width w'** is $(a_{w-1} \cdots a_1 a_0 . c_1 \cdots c_{w'})_{b,w,w'}$ where $a_0, a_1, \ldots, a_{w-1}, c_1, \ldots, c_{w'}$ are nonnegative integers less than b and

$$x \geq \sum_{i=0}^{w-1} a_i b^i + \sum_{j=1}^{w'} c_j b^{-j} \qquad \text{and} \qquad x < \sum_{i=0}^{w-1} a_i b^i + \sum_{j=1}^{w'} c_j b^{-j} + b^{-w'}$$

3.75 in fixed-width binary, integer part width 2, fractional part width 8	
0.1 in fixed-width binary, integer part width 2, fractional part width 8	

```
welcome $jshell
|  Welcome to JShell -- Version 10.0.1
|  For an introduction type: /help intro

jshell> 0.1
$1 ==>

jshell> 0.2
$2 ==>

jshell> 0.1 + 0.2
$3 ==>

jshell> Math.sqrt(2)
$4 ==>

jshell> Math.sqrt(2)*Math.sqrt(2)
$5 ==>

jshell>
```

Note: Java uses floating point, not fixed width representation, but similar rounding errors appear in both.

Negative int expansions

Representing negative integers in binary: Fix a positive integer width for the representation w, $w > 1$.

	To represent a positive integer n	To represent a negative integer $-n$
Sign-magnitude	$[0a_{w-2} \cdots a_0]_{s,w}$, where $n = (a_{w-2} \cdots a_0)_{2,w-1}$ Example $n = 17$, $w = 7$:	$[1a_{w-2} \cdots a_0]_{s,w}$, where $n = (a_{w-2} \cdots a_0)_{2,w-1}$ Example $-n = -17$, $w = 7$:
2s complement	$[0a_{w-2} \cdots a_0]_{2c,w}$, where $n = (a_{w-2} \cdots a_0)_{2,w-1}$ Example $n = 17$, $w = 7$:	$[1a_{w-2} \cdots a_0]_{2c,w}$, where $2^{w-1} - n = (a_{w-2} \cdots a_0)_{2,w-1}$ Example $-n = -17$, $w = 7$:

Calculating 2s complement

For positive integer n, to represent $-n$ in 2s complement with width w,

- Calculate $2^{w-1} - n$, convert result to binary fixed-width $w - 1$, pad with leading 1, or
- Express $-n$ as a sum of powers of 2, where the leftmost 2^{w-1} is negative weight, or
- Convert n to binary fixed-width w, flip bits, add 1 (ignore overflow)

Challenge: use definitions to explain why each of these approaches works.

Representing zero

Representing 0:

So far, we have representations for positive and negative integers. What about 0?

	To represent a **non-negative** integer n	To represent a **non-positive** integer $-n$
Sign-magnitude	$[0a_{w-2}\cdots a_0]_{s,w}$, where $n = (a_{w-2}\cdots a_0)_{2,w-1}$ Example $n = 0$, $w = 7$:	$[1a_{w-2}\cdots a_0]_{s,w}$, where $n = (a_{w-2}\cdots a_0)_{2,w-1}$ Example $-n = 0$, $w = 7$:
2s complement	$[0a_{w-2}\cdots a_0]_{2c,w}$, where $n = (a_{w-2}\cdots a_0)_{2,w-1}$ Example $n = 0$, $w = 7$:	$[1a_{w-2}\cdots a_0]_{2c,w}$, where $2^{w-1} - n = (a_{w-2}\cdots a_0)_{2,w-1}$ Example $-n = 0$, $w = 7$: