Specification in First Order Logic

COMP2600 / COMP6260

Dirk Pattinson
Australian National University

Semester 2, 2015

Why formal verification?

- So you have just written a program to do *something*
- But how do you know it works as expected?
- You could do a series of tests
- How do you know when you have run enough tests?

From testing to formal verification

> *Program testing can be used to show the presence of bugs, but never to show their absence!*

Edsger W. Dijkstra (1970), Dutch computer scientist and Turing Award recipient

- We want to formally *prove* that a program performs as expected
- Need to define what is *expected*
- Could describe it in a natural language specification document
- Natural language is ambiguous
- Use formal specification language

Ambiguity of natural language

- Consider the two natural language sentences:
  - Time flies like the wind
  - Fruit flies like a banana
- Whilst the first one is reasonably clear, the second one has two interpretations:

Ambiguity of natural language

- Consider the two natural language sentences:
  - Time flies like the wind
  - Fruit flies like a banana
- Whilst the first one is reasonably clear, the second one has two interpretations:
  - Subject = fruit. Verb = *to fly*. Fruit *flies* like a banana.

Ambiguity of natural language

- Consider the two natural language sentences:
  - Time flies like the wind
  - Fruit flies like a banana
- Whilst the first one is reasonably clear, the second one has two interpretations:
  - Subject = fruit. Verb = *to fly*. Fruit *flies* like a banana.
  - Subject = fruit flies (agriculture pests). Verb = *to like*. Fruit flies *like* a banana.

Ambiguity of natural language

- Consider the two natural language sentences:
  - Time flies like the wind
  - Fruit flies like a banana
- Whilst the first one is reasonably clear, the second one has two interpretations:
  - Subject = fruit. Verb = *to fly*. Fruit *flies* like a banana.
  - Subject = fruit flies (agriculture pests). Verb = *to like*. Fruit flies *like* a banana.
- Search for 'syntactic ambiguity' for more examples
- Natural language ambiguity the basis of many jokes and double entendres
- Fun, but not so useful for clear specifications!

Formal languages/frameworks

- Today: very basic specifications using First Order Logic
- More industrial strength approaches: Z method, petri nets, process calculi

A warmup example: if

- Translate the sentence *The bomb will explode if you cut the red wire*
- Let C denote *you CUT the red wire* and B denote *the BOMB will explode*

A warmup example: if

- Translate the sentence *The bomb will explode if you cut the red wire*
- Let C denote *you CUT the red wire* and B denote *the BOMB will explode*
- The sentence is an implication $C \rightarrow B$
- What does the sentence say about the case when C is false?

A warmup example: if

- Translate the sentence *The bomb will explode if you cut the red wire*
- Let C denote *you CUT the red wire* and B denote *the BOMB will explode*
- The sentence is an implication $C \rightarrow B$
- What does the sentence say about the case when C is false?
- The bomb may still explode (for other reasons).
- See truth table for $\rightarrow$ when the antecedent is false

A warmup example: only if

- Translate the sentence *The bomb will explode only if you cut the red wire*
- Let C denote *you CUT the red wire* and B denote *the BOMB will explode*

A warmup example: only if

- Translate the sentence *The bomb will explode only if you cut the red wire*
- Let C denote *you CUT the red wire* and B denote *the BOMB will explode*
- Paraphrase as *If you do not cut the red wire, the bomb will not explode*

A warmup example: only if

- Translate the sentence *The bomb will explode only if you cut the red wire*
- Let C denote *you CUT the red wire* and B denote *the BOMB will explode*
- Paraphrase as *If you do not cut the red wire, the bomb will not explode*
- The sentence is an implication $\neg C \rightarrow \neg B$
- What does the sentence say about the case when C is false?

A warmup example: only if

- Translate the sentence *The bomb will explode only if you cut the red wire*
- Let C denote *you CUT the red wire* and B denote *the BOMB will explode*
- Paraphrase as *If you do not cut the red wire, the bomb will not explode*
- The sentence is an implication $\neg C \rightarrow \neg B$
- What does the sentence say about the case when C is false?
- It says we are safe from explosion.

A warmup example: only if

- Translate the sentence *The bomb will explode only if you cut the red wire*
- Let C denote *you CUT the red wire* and B denote *the BOMB will explode*
- Paraphrase as *If you do not cut the red wire, the bomb will not explode*
- The sentence is an implication $\neg C \rightarrow \neg B$
- What does the sentence say about the case when C is false?
- It says we are safe from explosion.
- Now what about *The bomb will explode if and only if you cut the red wire*

A warmup example: only if

- Translate the sentence *The bomb will explode only if you cut the red wire*
- Let C denote *you CUT the red wire* and B denote *the BOMB will explode*
- Paraphrase as *If you do not cut the red wire, the bomb will not explode*
- The sentence is an implication $\neg C \rightarrow \neg B$
- What does the sentence say about the case when C is false?
- It says we are safe from explosion.
- Now what about *The bomb will explode if and only if you cut the red wire*
- The sentence is an equivalence (or bi-implication) $B \leftrightarrow C$
- It guarantees that:
  - We are safe *unless* the wire is cut and
  - Cutting the wire *will* cause an explosion

An example using conjunctions and disjunctions

- Translate *Bob owns a car and a bicycle*
  - Let *OC* denote *Bob owns a CAR*
  - Let *OB* denote *Bob owns a BICYCLE*

An example using conjunctions and disjunctions

- Translate *Bob owns a car and a bicycle*
  - Let *OC* denote *Bob owns a CAR*
  - Let *OB* denote *Bob owns a BICYCLE*
  - The sentence is a conjunction $OC \wedge OB$

An example using conjunctions and disjunctions

- Translate *Bob owns a car and a bicycle*
  - Let *OC* denote *Bob owns a CAR*
  - Let *OB* denote *Bob owns a BICYCLE*
  - The sentence is a conjunction $OC \wedge OB$
- Translate *Bob arrived at the lecture by car or bicyle*
  - Let *AC* denote *Bob arrived by CAR*
  - Let *AB* denote *Bob arrived by BICYCLE*

An example using conjunctions and disjunctions

- Translate *Bob owns a car and a bicycle*
  - Let OC denote *Bob owns a CAR*
  - Let OB denote *Bob owns a BICYCLE*
  - The sentence is a conjunction $OC \wedge OB$
- Translate *Bob arrived at the lecture by car or bicyle*
  - Let AC denote *Bob arrived by CAR*
  - Let AB denote *Bob arrived by BICYCLE*
  - The sentence is a disjunction $AC \vee AB$

Translation vs validity

- Is the following a tautology? *if the person owns a car and a bicycle, the person arrived by car or bicycle*

Translation vs validity

- Is the following a tautology? *if the person owns a car and a bicycle, the person arrived by car or bicycle*
- Step 1: translate the statement: *if the person owns a car and a bicycle, the person arrived by car or bicycle*

Translation vs validity

- Is the following a tautology? *if the person owns a car and a bicycle, the person arrived by car or bicycle*
- Step 1: translate the statement: *if the person owns a car and a bicycle, the person arrived by car or bicycle*
- $OC \wedge OB \rightarrow AC \vee AB$

Translation vs validity

- Is the following a tautology? *if the person owns a car and a bicycle, the person arrived by car or bicycle*
- Step 1: translate the statement: *if the person owns a car and a bicycle, the person arrived by car or bicycle*
- $OC \wedge OB \rightarrow AC \vee AB$
- Step 2: is $OC \wedge OB \rightarrow AC \vee AB$ valid (true for all values of OB, OC, AB, AC)?

Translation vs validity

- Is the following a tautology? *if the person owns a car and a bicycle, the person arrived by car or bicycle*
- Step 1: translate the statement: *if the person owns a car and a bicycle, the person arrived by car or bicycle*
- $OC \wedge OB \rightarrow AC \vee AB$
- Step 2: is $OC \wedge OB \rightarrow AC \vee AB$ valid (true for all values of OB, OC, AB, AC)?
- No. Perhaps the person walked or caught the bus? I.e. counterexample $OB = true$, $OC = true$ but $AB = false$, $AC = false$

Translation vs validity

- Is the following a tautology? *if the person owns a car and a bicycle, the person arrived by car or bicycle*
- Step 1: translate the statement: *if the person owns a car and a bicycle, the person arrived by car or bicycle*
- $OC \wedge OB \rightarrow AC \vee AB$
- Step 2: is $OC \wedge OB \rightarrow AC \vee AB$ valid (true for all values of OB, OC, AB, AC)?
- No. Perhaps the person walked or caught the bus? I.e. counterexample $OB = true$, $OC = true$ but $AB = false$, $AC = false$
- Moral of this contrived example: not everything that is expressed formally is a tautology.
- We will see proofs tomorrow. For now, focus on translations.

From bombs and bicycles to programs

- Suppose we have been asked to verify that a program called `Sort` correctly sorts a list of numbers
- The first step is to formally describe *correctly sorts a list of numbers*
- Let L be a list

From bombs and bicycles to programs

- Suppose we have been asked to verify that a program called `Sort` correctly sorts a list of numbers
- The first step is to formally describe *correctly sorts a list of numbers*
- Let L be a list
- Let $Sort(L)$ be the output of `Sort` given a list L

From bombs and bicycles to programs

- Suppose we have been asked to verify that a program called `Sort` correctly sorts a list of numbers
- The first step is to formally describe *correctly sorts a list of numbers*
- Let L be a list
- Let $Sort(L)$ be the output of `Sort` given a list L
- Let $pr(x_1, x_2, L)$ mean that x_1 precedes x_2 in L

From bombs and bicycles to programs

- Suppose we have been asked to verify that a program called `Sort` correctly sorts a list of numbers
- The first step is to formally describe *correctly sorts a list of numbers*
- Let L be a list
- Let $Sort(L)$ be the output of `Sort` given a list L
- Let $pr(x_1, x_2, L)$ mean that x_1 precedes x_2 in L
- Let $lt(x_1, x_2)$ mean that $x_1 < x_2$

From bombs and bicycles to programs

- Suppose we have been asked to verify that a program called `Sort` correctly sorts a list of numbers
- The first step is to formally describe *correctly sorts a list of numbers*
- Let L be a list
- Let $Sort(L)$ be the output of `Sort` given a list L
- Let $pr(x_1, x_2, L)$ mean that x_1 precedes x_2 in L
- Let $lt(x_1, x_2)$ mean that $x_1 < x_2$
- Here is a formal statement of `Sort` *correctly sorts a list of numbers*:

$$\forall x_1 \forall x_2 \in L . lt(x_1, x_2) \leftrightarrow pr(x_1, x_2, Sort(L))$$

- The second step is to verify that this holds for all lists.

Breaking down the translation

- `Sort` *correctly sorts a list of numbers*
- What does it mean for numbers to be sorted?
  - They occur in ascending order ($<$)
- How do we relate the members of the list?
  - We say that two numbers *precede* each other in the list ($pr(x_1, x_2, L)$)
- How do we say that the *entire* list is sorted?
  - We *quantify* over all members of the list

Which quantifier to choose

- $\forall$ - generalisation
  - *all*, *any*, *every*
  - can often be implicit
- $\exists$ - existence
  - *some*, *there is*, *a/an*
  - usually explicit

Quantifier scope

- Which subformula has the quantifier as the main connective?
- In linguistic terms - which part of the sentence does the quantifier affect?
- (*All* elements of the list L are greater than 0) and (*some* element is equal to 5)
- Quantifiers can be nested
- (*There is some* node of the tree T that is smaller than (*every* node of T))

Min(S)

- *The function* `Min` *calculates the smallest number in the set S*

Min(S)

- *The function* `Min` *calculates the smallest number in the set S*
- Step 1: elaborate / paraphrase

Min(S)

- *The function* `Min` *calculates the smallest number in the set S*
- Step 1: elaborate / paraphrase
  - *The output of the function* `Min` *is the smallest of all numbers in the set S*

Min(S)

- *The function* `Min` *calculates the smallest number in the set S*
- Step 1: elaborate / paraphrase
  - *The output of the function* `Min` *is the smallest of all numbers in the set S*

Min(S)

- *The function* `Min` *calculates the smallest number in the set S*
- Step 1: elaborate / paraphrase
  - *The output of the function* `Min` *is the smallest of all numbers in the set S*
- Step 2: Find the individual predicates

Min(S)

- *The function* `Min` *calculates the smallest number in the set S*
- Step 1: elaborate / paraphrase
  - *The output of the function* `Min` *is the smallest of all numbers in the set S*
- Step 2: Find the individual predicates
  - Let $Min(S)$ be the result of function *Min* over S
  - Let $MinResult(S, x)$ mean that $x = Min(S)$ (alternatively we could use FOL with equality)

Min(S)

- *The function* `Min` *calculates the smallest number in the set S*
- Step 1: elaborate / paraphrase
  - *The output of the function* `Min` *is the smallest of all numbers in the set S*
- Step 2: Find the individual predicates
  - Let $Min(S)$ be the result of function *Min* over S
  - Let $MinResult(S, x)$ mean that $x = Min(S)$ (alternatively we could use FOL with equality)
  - Let $lt(n_1, n_2)$ mean that $n_1 < n_2$

Min(S)

- *The function* `Min` *calculates the smallest number in the set S*
- Step 1: elaborate / paraphrase
  - *The output of the function* `Min` *is the smallest of all numbers in the set S*
- Step 2: Find the individual predicates
  - Let $Min(S)$ be the result of function *Min* over S
  - Let $MinResult(S, x)$ mean that $x = Min(S)$ (alternatively we could use FOL with equality)
  - Let $lt(n_1, n_2)$ mean that $n_1 < n_2$
- Step 3: find quantifiers and their scope

Min(S)

- *The function* `Min` *calculates the smallest number in the set S*
- Step 1: elaborate / paraphrase
  - *The output of the function* `Min` *is the smallest of all numbers in the set S*
- Step 2: Find the individual predicates
  - Let $Min(S)$ be the result of function *Min* over S
  - Let $MinResult(S, x)$ mean that $x = Min(S)$ (alternatively we could use FOL with equality)
  - Let $lt(n_1, n_2)$ mean that $n_1 < n_2$
- Step 3: find quantifiers and their scope
  - *all* numbers in the set S
  - note that this was implicit in the original sentence

Min(S)

- *The function `Min` calculates the smallest number in the set S*
- Step 1: elaborate / paraphrase
  - *The output of the function `Min` is the smallest of all numbers in the set S*
- Step 2: Find the individual predicates
  - Let $Min(S)$ be the result of function *Min* over S
  - Let $MinResult(S, x)$ mean that $x = Min(S)$ (alternatively we could use FOL with equality)
  - Let $lt(n_1, n_2)$ mean that $n_1 < n_2$
- Step 3: find quantifiers and their scope
  - *all* numbers in the set S
  - note that this was implicit in the original sentence
- Putting it all together:
  - $\forall n_1 \in S.MinResult(S, n_1) \leftrightarrow (\forall n_2 \in (S \setminus \{n_1\}).lt(n_1, n_2))$
  - Note: $\setminus$ stands for set subtraction (sometimes also denoted –)
  - Hence $S \setminus \{n_1\}$ means every element of S except n_1

File deletion

- *Any temporary file is deleted if no process is using it*

File deletion

- *Any temporary file is deleted if no process is using it*
- Step 1: elaborate / paraphrase

File deletion

- *Any temporary file is deleted if no process is using it*
- Step 1: elaborate / paraphrase
  - *For all temporary files, the file is deleted if no process is using the file*
  - *For all temporary files, if no process is using the file, then the file is deleted*

File deletion

- *Any temporary file is deleted if no process is using it*
- Step 1: elaborate / paraphrase
  - *For all temporary files, the file is deleted if no process is using the file*
  - *For all temporary files, if no process is using the file, then the file is deleted*
- Step 2: Find the individual predicates

File deletion

- *Any temporary file is deleted if no process is using it*
- Step 1: elaborate / paraphrase
  - *For all temporary files, the file is deleted if no process is using the file*
  - *For all temporary files, if no process is using the file, then the file is deleted*
- Step 2: Find the individual predicates
  - Let $U(x, y)$ mean process x is using file y
  - Let $D(x)$ mean file x is deleted

File deletion

- *Any temporary file is deleted if no process is using it*
- Step 1: elaborate / paraphrase
  - *For all temporary files, the file is deleted if no process is using the file*
  - *For all temporary files, if no process is using the file, then the file is deleted*
- Step 2: Find the individual predicates
  - Let $U(x, y)$ mean process x is using file y
  - Let $D(x)$ mean file x is deleted
- Step 3: find quantifiers and their scope

File deletion

- *Any temporary file is deleted if no process is using it*
- Step 1: elaborate / paraphrase
  - *For all temporary files, the file is deleted if no process is using the file*
  - *For all temporary files, if no process is using the file, then the file is deleted*
- Step 2: Find the individual predicates
  - Let $U(x, y)$ mean process x is using file y
  - Let $D(x)$ mean file x is deleted
- Step 3: find quantifiers and their scope
  - *all* files
  - note that this was implicit in the original sentence
- Putting it all together:

File deletion

- *Any temporary file is deleted if no process is using it*
- Step 1: elaborate / paraphrase
  - *For all temporary files, the file is deleted if no process is using the file*
  - *For all temporary files, if no process is using the file, then the file is deleted*
- Step 2: Find the individual predicates
  - Let $U(x, y)$ mean process x is using file y
  - Let $D(x)$ mean file x is deleted
- Step 3: find quantifiers and their scope
  - *all* files
  - note that this was implicit in the original sentence
- Putting it all together:
  - $\forall f.((\neg \exists p.\ U(p, f)) \rightarrow D(f))$

If vs *if and only if*

- We translated *Any temporary file is deleted if no process is using it* as:
  - $\forall f.((\neg \exists p.\ U(p,f)) \rightarrow D(f))$
- What does this tell us about the case when some process *is* using the file?

If vs *if and only if*

- We translated *Any temporary file is deleted if no process is using it* as:
  - $\forall f.((\neg \exists p.\ U(p,f)) \rightarrow D(f))$
- What does this tell us about the case when some process *is* using the file?
- Intuitively and informally, tempted to say *then file is not deleted*
- But!
- What is the truth value of $(\neg \exists p.\ U(p,f)) \rightarrow D(f)$ when the following hold:

If vs *if and only if*

- We translated *Any temporary file is deleted if no process is using it* as:
  - $\forall f.((\neg \exists p.\ U(p,f)) \rightarrow D(f))$
- What does this tell us about the case when some process *is* using the file?
- Intuitively and informally, tempted to say *then file is not deleted*
- But!
- What is the truth value of $(\neg \exists p.\ U(p,f)) \rightarrow D(f)$ when the following hold:
  - $\exists p_1.\ U(p_1,f)$ is *True* and
  - $D(f)$ is true?

If vs *if and only if*

- We translated *Any temporary file is deleted if no process is using it* as:
  - $\forall f.((\neg \exists p.\ U(p, f)) \rightarrow D(f))$
- What does this tell us about the case when some process *is* using the file?
- Intuitively and informally, tempted to say *then file is not deleted*
- But!
- What is the truth value of $(\neg \exists p.\ U(p, f)) \rightarrow D(f)$ when the following hold:
  - $\exists p_1.\ U(p_1, f)$ is *True* and
  - $D(f)$ is true?
- It is *False* $\rightarrow$ *True* ie *True*!

If vs *if and only if*

- We translated *Any temporary file is deleted if no process is using it* as:
  - $\forall f.((\neg \exists p.\ U(p, f)) \rightarrow D(f))$
- What does this tell us about the case when some process *is* using the file?
- Intuitively and informally, tempted to say *then file is not deleted*
- But!
- What is the truth value of $(\neg \exists p.\ U(p, f)) \rightarrow D(f)$ when the following hold:
  - $\exists p_1.\ U(p_1, f)$ is *True* and
  - $D(f)$ is true?
- It is *False* $\rightarrow$ *True* ie *True*!
- So $\forall f.((\neg \exists p.\ U(p, f)) \rightarrow D(f))$ is still true
  - provided no other file/process makes $(\neg \exists p.\ U(p, f)) \rightarrow D(f)$ false

Making a stronger statement

- *Any temporary file is deleted if no process is using it*
- What does this statement ensure?

Making a stronger statement

- *Any temporary file is deleted if no process is using it*
- What does this statement ensure?
- That a file is necessarily deleted in the case when no process is using it

Making a stronger statement

- *Any temporary file is deleted if no process is using it*
- What does this statement ensure?
- That a file is necessarily deleted in the case when no process is using it
- How to ensure that a file is *not* deleted when there *is* some process using it?

Making a stronger statement

- *Any temporary file is deleted if no process is using it*
- What does this statement ensure?
- That a file is necessarily deleted in the case when no process is using it
- How to ensure that a file is *not* deleted when there *is* some process using it?
  - *Any temporary file is deleted only if no process is using it*

Making a stronger statement

- *Any temporary file is deleted if no process is using it*
- What does this statement ensure?
- That a file is necessarily deleted in the case when no process is using it
- How to ensure that a file is *not* deleted when there *is* some process using it?
  - *Any temporary file is deleted only if no process is using it*
  - Equivalent to $\forall f.(D(f) \rightarrow (\neg \exists p.\ U(p, f)))$

Making a stronger statement

- *Any temporary file is deleted if no process is using it*
- What does this statement ensure?
- That a file is necessarily deleted in the case when no process is using it
- How to ensure that a file is *not* deleted when there *is* some process using it?
  - *Any temporary file is deleted only if no process is using it*
  - Equivalent to $\forall f.(D(f) \rightarrow (\neg \exists p.\ U(p, f)))$
  - Putting both requirements together:

Making a stronger statement

- *Any temporary file is deleted if no process is using it*
- What does this statement ensure?
- That a file is necessarily deleted in the case when no process is using it
- How to ensure that a file is *not* deleted when there *is* some process using it?
  - *Any temporary file is deleted only if no process is using it*
  - Equivalent to $\forall f.(D(f) \rightarrow (\neg \exists p.\ U(p, f)))$
  - Putting both requirements together:
  - *Any temporary file is deleted if and only if no process is using it*

Making a stronger statement

- *Any temporary file is deleted if no process is using it*
- What does this statement ensure?
- That a file is necessarily deleted in the case when no process is using it
- How to ensure that a file is *not* deleted when there *is* some process using it?
  - *Any temporary file is deleted only if no process is using it*
  - Equivalent to $\forall f.(D(f) \rightarrow (\neg \exists p.\ U(p,f)))$
  - Putting both requirements together:
  - *Any temporary file is deleted if and only if no process is using it*
  - $\forall f.((\neg \exists p.\ U(p,f)) \leftrightarrow D(f))$

Binary trees again

- Given:
  - $LT(z, w)$: $z < w$
  - $LC(z, w, T)$: z occurs in the left subtree of w in tree T
  - $RC(z, w, T)$: z occurs in the right subtree of w in tree T
- Translate the following into English:
- $\forall xy \in T.(LT(x, y) \rightarrow (LC(x, y) \vee RC(y, x))$

Binary trees again

- Given:
  - $LT(z, w)$: $z < w$
  - $LC(z, w, T)$: z occurs in the left subtree of w in tree T
  - $RC(z, w, T)$: z occurs in the right subtree of w in tree T
- Translate the following into English:
- $\forall xy \in T.(LT(x, y) \rightarrow (LC(x, y) \vee RC(y, x))$
- Direct translation into English
- For all nodes x, y of tree T, if node x is smaller than node y then x occurs in the left subtree of y or y occurs in the right subtree of x

Binary trees again

- Given:
  - $LT(z,w)$: $z < w$
  - $LC(z,w,T)$: z occurs in the left subtree of w in tree T
  - $RC(z,w,T)$: z occurs in the right subtree of w in tree T
- Translate the following into English:
- $\forall xy \in T.(LT(x,y) \rightarrow (LC(x,y) \vee RC(y,x))$
- Direct translation into English
- For all nodes x, y of tree T, if node x is smaller than node y then x occurs in the left subtree of y or y occurs in the right subtree of x
- Alternative / more readable version:

Binary trees again

- Given:
  - $LT(z,w)$: $z < w$
  - $LC(z,w,T)$: z occurs in the left subtree of w in tree T
  - $RC(z,w,T)$: z occurs in the right subtree of w in tree T
- Translate the following into English:
- $\forall xy \in T.(LT(x,y) \rightarrow (LC(x,y) \vee RC(y,x))$
- Direct translation into English
- For all nodes x, y of tree T, if node x is smaller than node y then x occurs in the left subtree of y or y occurs in the right subtree of x
- Alternative / more readable version:
- In tree T for every node a, the left subtree of a contains nodes smaller than a, and a is smaller than the nodes in the right subtree