CS 6110, Software Correctness Analysis, Spring 2023
Lecture 9 : Declarative notations, First Order Logic

Ganesh Gopalakrishnan
School of Computing
University of Utah
Salt Lake City, UT 84112

Excerpts from Bradley and Manna's book

- Let's go through the basics of First Order Logic's (FOL) syntax and semantics

Excerpts from Bradley and Manna's book

One task we logicians are interested in is that of analyzing the notion of "proof" — to make it as rigorous as any other notion in mathematics.

— Raymond Smullyan
The Lady or the Tiger?, 1982

Excerpts from Bradley and Manna's book

2.1 Syntax

All formulae of PL evaluate to true or false. FOL is not so simple. In FOL, **terms** evaluate to values other than truth values such as integers, people, or cards of a deck. However, we are getting ahead of ourselves: just as in PL,

Excerpts from Bradley and Manna's book

the syntax of FOL is independent of its meaning. The most basic terms are **variables** x, y, z, x_1, x_2, ... and **constants** a, b, c, a_1, a_2, ...

More complicated terms are constructed using **functions**. An n-ary function f takes n terms as arguments. Notationally, we represent generic FOL functions by symbols f, g, h, f_1, f_2, ... A constant can also be viewed as a 0-ary function.

Example 2.1. The following are all terms:

- a, a constant (or 0-ary function);
- x, a variable;
- $f(a)$, a unary function f applied to a constant;
- $g(x, b)$, a binary function g applied to a variable x and a constant b;
- $f(g(x, f(b)))$.

Excerpts from Bradley and Manna's book

The propositional variables of PL are generalized to **predicates**, denoted p, q, r, p_1, p_2, ... An n-ary predicate takes n terms as arguments. A **FOL propositional variable** is a 0-ary predicate, which we write P, Q, R, P_1, P_2, ...

Countably infinitely many constant, function, and predicate symbols are available.

An **atom** is $\top$, $\bot$, or an n-ary predicate applied to n terms. A **literal** is an atom or its negation.

Example 2.2. The following are all literals:

- P, a propositional variable (or 0-ary predicate);
- $p(f(x), g(x, f(x)))$, a binary predicate applied to two terms;
- $\neg p(f(x), g(x, f(x)))$.

The first two literals are also atoms. ■

Excerpts from Bradley and Manna's book

A **FOL formula** is a literal, the application of a logical connective $\neg$, $\wedge$, $\vee$, $\rightarrow$, or $\leftrightarrow$ to a formula or formulae, or the application of a **quantifier** to a formula. There are two FOL quantifiers:

- the **existential quantifier** $\exists x.\ F[x]$, read "there exists an x such that $F[x]$";
- and the **universal quantifier** $\forall x.\ F[x]$, read "for all x, $F[x]$".

In $\forall x.\ F[x]$, x is the **quantified variable**, and $F[x]$ is the **scope** of the quantifier $\forall x$. For convenience, we sometimes refer informally to the scope of the quantifier $\forall x$ as the scope of the quantified variable x itself. The case is similar for $\exists x.\ F[x]$. Also, x in $F[x]$ is **bound** (by the quantifier). By convention, the period in ". $F[x]$" indicates that the scope of the quantified variable x extends as far as possible. We often abbreviate $\forall x.\ \forall y.\ F[x, y]$ by $\forall x, y.\ F[x, y]$.

Excerpts from Bradley and Manna's book

Example 2.3. In

$$\underbrace{\forall x.\ p(f(x),x)\ \rightarrow\ (\exists y.\ \underbrace{p(f(g(x,y)),g(x,y))}_{G})\ \wedge\ q(x,f(x))}_{F}\ ,$$

the scope of x is F, and the scope of y is G. This formula is read: "for all x, if $p(f(x),x)$ then there exists a y such that $p(f(g(x,y)),g(x,y))$ and $q(x,f(x))$".

■

Excerpts from Bradley and Manna's book

Example 2.4. In

$$F: \ \forall x.\ p(f(x), y) \ \rightarrow \ \forall y.\ p(f(x), y) \ ,$$

x only occurs bound, while y appears both free (in the antecedent) and bound (in the consequent). Thus, $\mathsf{free}(F) = \{y\}$ and $\mathsf{bound}(F) = \{x, y\}$. ■

If $\mathsf{free}(F) = \{x_1, \ldots, x_n\}$, then its **universal closure** is

$$\forall x_1. \ \ldots \forall x_n.\ F \ ,$$

and its **existential closure** is

$$\exists x_1. \ \ldots \exists x_n.\ F \ .$$

A formula is valid IFF its universal closure is valid

A formula is satisfiable IFF its existential closure is satisfiable

Excerpts from Bradley and Manna's book

The **strict subterms** of a term excludes the term itself.

Example 2.5. In

$$F : \ \forall x.\ p(f(x), y) \ \rightarrow \ \forall y.\ p(f(x), y) \ ,$$

the subformulae of F are

$$F \ , \ p(f(x), y) \ \rightarrow \ \forall y.\ p(f(x), y) \ , \ \forall y.\ p(f(x), y) \ , \ p(f(x), y) \ .$$

The subterms of $g(f(x), f(h(f(x))))$ are

$$g(f(x), f(h(f(x)))) \ , \ f(x) \ , \ f(h(f(x))) \ , \ h(f(x)) \ , \ x \ .$$

$f(x)$ occurs twice in $g(f(x), f(h(f(x))))$. ■

Excerpts from Bradley and Manna's book

Example 2.6. Before discussing the formal semantics for FOL, we suggest translations of English sentences into FOL. The names of the constants, functions, and predicates are chosen to provide some intuition for the meaning of the FOL formulae.

- Every dog has its day.

$$\forall x.\ dog(x) \rightarrow \exists y.\ day(y) \wedge itsDay(x, y)$$

- Some dogs have more days than others.

$$\exists x, y.\ dog(x) \wedge dog(y) \wedge \#days(x) > \#days(y)$$

- All cats have more days than dogs.

$$\forall x, y.\ dog(x) \wedge cat(y) \rightarrow \#days(y) > \#days(x)$$

- Fido is a dog. Furrball is a cat. Fido has fewer days than does Furrball.

$$dog(Fido) \wedge cat(Furrball) \wedge \#days(Fido) < \#days(Furrball)$$

- The length of one side of a triangle is less than the sum of the lengths of the other two sides.

$$\forall x, y, z.\ triangle(x, y, z) \rightarrow length(x) < length(y) + length(z)$$

- Fermat's Last Theorem.

$$\begin{array}{l} \forall n.\ integer(n) \wedge n > 2 \\ \quad \rightarrow \forall x, y, z. \\ \quad\quad integer(x) \wedge integer(y) \wedge integer(z) \wedge x > 0 \wedge y > 0 \wedge z > 0 \\ \quad\quad\quad \rightarrow x^n + y^n \neq z^n \end{array}$$

Excerpts from Bradley and Manna's book

2.2 Semantics

Having defined the syntax of FOL, we now define its **semantics**. Formulae of FOL evaluate to the truth values true and false as in PL. However, terms of FOL formulae evaluate to values from a specified domain. We extend the concept of interpretations to this more complex setting and then define the semantics of FOL in terms of interpretations.

First, we define a FOL **interpretation** I. The **domain** D_I of an interpretation I is a nonempty set of values or objects, such as integers, real numbers, dogs, people, or merely abstract objects. $|D_I|$ denotes the **cardinality**, or size, of D_I. Domains can be finite, such as the 52 cards of a deck of cards; countably infinite, such as the integers; or uncountably infinite, such as the reals. But all domains are nonempty.

The **assignment** α_I of interpretation I maps constant, function, and predicate symbols to elements, functions, and predicates over D_I. It also maps variables to elements of D_I:

- each variable symbol x is assigned a value x_I from D_I;
- each n-ary function symbol f is assigned an n-ary function

  $$f_I : D_I^n \to D_I$$

  that maps n elements of D_I to an element of D_I;
- each n-ary predicate symbol p is assigned an n-ary predicate

  $$p_I : D_I^n \to \{\mathsf{true},\ \mathsf{false}\}$$

  that maps n elements of D_I to a truth value.

An interpretation $I : (D_I, \alpha_I)$ is thus a pair consisting of a domain and an assignment.

A "model" as generated by Z3
Or
An "instance" as generated by Alloy

Are examples of "Interpretations"

- Domains of interpretation
- Assignments

Excerpts from Bradley and Manna's book

Example 2.8. Recall the formula

$$F: \; x + y > z \;\rightarrow\; y > z - x$$

of Example 2.7 and the interpretation $I : (\mathbb{Z}, \alpha_I)$, where

$$\alpha_I : \; \{+ \mapsto +_{\mathbb{Z}},\; - \mapsto -_{\mathbb{Z}},\; > \mapsto >_{\mathbb{Z}},\; x \mapsto 13_{\mathbb{Z}},\; y \mapsto 42_{\mathbb{Z}},\; z \mapsto 1_{\mathbb{Z}}\} \; .$$

Compute the truth value of F under I as follows:

1. $I \models x + y > z$ since $\alpha_I[x + y > z] = 13_{\mathbb{Z}} +_{\mathbb{Z}} 42 >_{\mathbb{Z}} 1_{\mathbb{Z}}$
2. $I \models y > z - x$ since $\alpha_I[y > z - x] = 42_{\mathbb{Z}} >_{\mathbb{Z}} 1_{\mathbb{Z}} -_{\mathbb{Z}} 13_{\mathbb{Z}}$
3. $I \models F$ by 1, 2, and the semantics of $\rightarrow$

■

Excerpts from Bradley and Manna's book

Example 2.9. Consider the formula

$$F: \ \exists x.\ f(x) = g(x)$$

and the interpretation $I : (D : \{\circ, \bullet\},\ \alpha_I)$ in which

$$\alpha_I : \ \{f(\circ) \mapsto \circ,\ f(\bullet) \mapsto \bullet,\ g(\circ) \mapsto \bullet,\ g(\bullet) \mapsto \circ\}\ .$$

Compute the truth value of F under I as follows:

1.	$I \triangleleft \{x \mapsto \mathsf{v}\}$	$\not\models$	$f(x) = g(x)$	for $\mathsf{v} \in D$
2.	I	$\not\models$	$\exists x.\ f(x) = g(x)$	since $\mathsf{v} \in D$ is arbitrary

In the first line, basic reasoning about the interpretation I reveals that f and g always disagree. The second line follows from the first by the semantics of existential quantification. ■

Examples from Manna's original book about interpretation and validity

Example 1

Consider the formula

$$Fmla1 = \exists F.F(a) = b \\ \wedge(\forall x).[p(x) \Rightarrow F(x) = g(x, F(f(x)))]$$

We will now provide *three distinct* interpretations for it.

Interpretation 1.

$\mathrm{D} = \mathrm{Nat}$
$\mathrm{a} = 0$
$\mathrm{b} = 1$
$\mathrm{f} = \lambda x.(x = 0 \rightarrow 0, x - 1)$
$\mathrm{g} = *$
$\mathrm{p} = \lambda x.x > 0$

Interpretation 2.

$\mathrm{D} = \Sigma^*$
$\mathrm{a} = \varepsilon$
$\mathrm{b} = \varepsilon$
$\mathrm{f} = \lambda x.(tail(x))$
$\mathrm{g(x,y)} = \mathrm{concat(y,head(x))}$
$\mathrm{p} = \lambda x.x \neq \varepsilon$

Interpretation 3.

$\mathrm{D} = \mathrm{Nat}$
$\mathrm{a} = 0$
$\mathrm{b} = 1$
$\mathrm{f} = \lambda x.x$
$\mathrm{g(x,y)} = \mathrm{y+1}$
$\mathrm{p} = \lambda x.x > 0$

It is clear that under Interpretation 1, $Fmla1$ is true, because there indeed exists a function F, namely the factorial function, that makes the assertion true. It is also true under Interpretation 2, while it is false under Interpretation 3 (Exercise 18.4 asks for proofs). Hence, this formula is *not* valid — because it is not true under all interpretations.

Excerpts from Bradley and Manna's book

2.3 Satisfiability and Validity

A formula F is said to be **satisfiable** iff there exists an interpretation I such that $I \models F$. A formula F is said to be **valid** iff for all interpretations I, $I \models F$. Determining satisfiability and validity of formulae are important tasks in FOL. Recall that satisfiability and validity are dual: F is valid iff $\neg F$ is unsatisfiable.

CVC used to be called SVC

Anyway , "V" = validity

But they always checked by negating and checking for unsat

Thus it really is a SAT (or SMT) tool … used to check validity via the above definition

Excerpts from Bradley and Manna's book

Excerpts from Bradley and Manna's book

Alloy checks asserts for validity
by negating them
and checking whether they are sat or not
If Sat then … ?
If Unsat then … ?

Remember that any Boolean formula is
* Valid
* A contradiction (its negation is valid)
* Neither – it and its negation are satisfiable

Alloy checks asserts for validity
by negating them
and checking whether they are sat or not
If Sat then the original assertion is invalid
If Unsat then the original assertion is valid
Just to be thorough, we will (later) do this:
* Write an assertion Assn
* Write negAssn = !Assn
* We will check both
* If Assn's "check" says "no counterexample, "Assn may be valid",
then ensure that !Assn is satisfiable (there is a counterexample for it)

BUT if both Assn and !Assn generate counterexamples, then they are
merely satisfiable but not valid AND also not contradictions !!

Encoding FOL zero-ary predicates (Bools)

```
-- How to introduce FOL zero-ary functions or constants
one sig a,b extends U { } -- constants 'a' in U - guaranteed within U

some sig U {
 p1 : set U, -- p1 is an arbitrary unary predicate
 q1 : set U, -- q1 is an arbitrary unary predicate
 r0  : lone U, -- r0 is for the simulation of a zero-ary pred -- BOOL
 r1  : lone U, -- r1 is for .. another zero-ary pred -- BOOL
}

pred R0 { some a.r0 } -- Got one Bool Var! True if r0 is suitably assigned!
//-- sanity assert r0true { some a.r0 }
//-- may not be true - depends on how r0 is assigned!

pred R1 { some a.r1 } -- Got another Bool var!

assert r0a { R0 => (R1 => R0) } -- Now we can prove Boolean facts!
check r0a
```

Encoding higher arity predicates

```
some sig U {}
sig S1 in U {}
sig S2 in U {}
pred p1[x:U]   { x in S1 }              -- general-enough 1-ary pred
pred p2[x,y:U] { x in S1 and y in S2 } -- general-enough 2-ary pred

-- This is general-enough because
-- S1, S2 can grow from {} to the full U, and it allows x,y to be
-- either within S1 or S2
-- SANITY run { #U = 2 }

assert EA { some y:U | all x:U | p2[x,y]
            =>
            all  x:U | some y:U | p2[x,y]
          }
assert nEA { !(some y:U | all x:U | p2[x,y]
            =>
            all  x:U | some y:U | p2[x,y])
          }
check EA
check nEA
```

Functions encoded now

```
-- Now to simulate functions in Pred Logic (of arity higher)
some sig U {
  f1 : U,        -- one-ary fn
  f2 : U -> U,   -- two-ary fn
}
sig S1 in U {}
sig S2 in U {}
pred p1[x:U]   { x in S1 } -- general-enough 1-ary pred
pred p2[x,y:U] { x in S1 and y in S2 } -- general-enough 2-ary pred

assert EA { some y:U | all x:U | p2[x,f1[y] ]
            =>
            all  x:U | some y:U | p2[x,f1[y] ]
          }
assert nEA { !(some y:U | all x:U | p2[x, f2[y,y] ]
            =>
            all  x:U | some y:U | p2[x, f2[y,y]  ] )
          }
check EA
check nEA
```

Excerpts from Bradley and Manna's book

To show that a formula F is invalid, it suffices to find an interpretation I such that $I \models \neg F$.

Example 2.13. Consider the formula

$$F: \ (\forall x.\ p(x,x)) \ \rightarrow \ (\exists x.\ \forall y.\ p(x,y)) \ .$$

To show that it is invalid, we find an interpretation I such that

$$I \ \models \ \neg((\forall x.\ p(x,x)) \ \rightarrow \ (\exists x.\ \forall y.\ p(x,y))) \ ,$$

or, according to the semantics of $\rightarrow$,

$$I \ \models \ (\forall x.\ p(x,x)) \ \wedge \ \neg(\exists x.\ \forall y.\ p(x,y)) \ .$$

Choose

$$D_I = \{0,1\}$$

and

$$p_I = \{(0,0),\ (1,1)\} \ .$$

We use a common notation for defining relations: $p_I(a,b)$ is **true** iff $(a,b) \in p_I$. Here, $p_I(0,0)$ is **true**, and $p_I(1,0)$ is **false**.

Both $\forall x.\ p(x,x)$ and $\neg(\exists x.\ \forall y.\ p(x,y))$ evaluate to **true** under I, so

$$I \ \models \ (\forall x.\ p(x,x)) \ \wedge \ \neg(\exists x.\ \forall y.\ p(x,y)) \ ,$$

which shows that F is invalid. Interpretation I is a falsifying interpretation of F. ■

Excerpts from Bradley and Manna's book

```
-- From Bradley and Manna, Example 2.13

-- a non-empty set U
some sig U {}

-- S is a subset of U
sig S in U {}

-- A general 2-ary predicate
pred p[x,y:U] { x in S and y in S } -- general-enough 2-ary pred

-- The Bradley/Manna assertion
assert EA { (all x:U | p[x,x]
             =>
             some x:U | all y:U | p[x,y]
           )
         }

-- Negation of the Bradley/Manna assertion
assert nEA {!(all x:U | p[x,x]
             =>
             some x:U | all y:U | p[x,y]
           )
          }
check EA
check nEA
```

Excerpts from Bradley and Manna's book

2.2 (FOL validity & satisfiability). For each of the following FOL formulae, identify whether it is valid or not. If it is valid, prove it with a semantic argument; otherwise, identify a falsifying interpretation.

(a) $(\forall x, y.\ p(x,y) \rightarrow p(y,x)) \rightarrow \forall z.\ p(z,z)$
(b) $\forall x, y.\ p(x,y) \rightarrow p(y,x) \rightarrow \forall z.\ p(z,z)$
(c) $(\exists x.\ p(x)) \rightarrow \forall y.\ p(y)$
(d) $(\forall x.\ p(x)) \rightarrow \exists y.\ p(y)$
(e) $\exists x, y.\ (p(x,y) \rightarrow (p(y,x) \rightarrow \forall z.\ p(z,z)))$

Seeding your FOL experience via a problem

- See the solution to Sudoku in https://ericpony.github.io/z3py-tutorial/guide-examples.htm
- Modify it to solve Kenken
  - I'll give you the cage encoding in Python
    - See this folder – template Python file kenken-template.py
  - You should write the constraints and produce a Kenken solution

Finishing up our study of FOL

- Review of notions about interpretations
  - See discussions about Interpretations in the Bradley/Manna book
- Soundness and Completeness
- Decidability

How do we systematically prove in FOL?

- One needs proof systems and proof rules
- FOL has a sound proof system (many, actually)
  - One such proof system is Natural deduction (others are Hibert-style, …)
  - Sound means if a theorem is proven using proof-rules, it is true
    - in the sematic model, or in the space of interpretations
      - More on it soon
  - Sound proof systems can be rather simple: have no proof rules!
    - No proofs means no unsoundness!
- What more do we need for proof systems?

How do we systematically prove in FOL?

- One needs proof systems and proof rules
- FOL has a sound proof system (many, actually)
    - One such proof system is Natural deduction (others are Hibert-style, …)
  - Sound means if a theorem is proven using proof-rules, it is true
    - in the sematic model, or in the space of interpretations
      - More on it soon
  - Sound proof systems can be rather simple: have no proof rules!
    - No proofs means no unsoundness!
- FOL has a complete proof system
  - Complete means ”if it is true, then it is provable”
- Thus, valid sentences in FOL are recursively enumerable
  - What is that? (next slide)

How do we systematically prove in FOL?

- FOL has a complete proof system
  - Complete means ”if it is true, then it is provable”
  - Thus there is a proof
    - a sequence of proof-steps that chain together, some implying later ones, ending in the theorem of interest
- Thus, valid sentences in FOL are recursively enumerable
  - That means, if true, we can find a proof
    - How?
    - Keep enumerating all proofs of all lengths, using a sound and complete proof system
    - If indeed true, there is a proof that will be discovered in a finite amount of time!
  - What if the sentence (FOL formula without free variables) is not valid?

How do we systematically prove in FOL?

- What if the sentence (formula w/o free variables) is not valid?
  - Then the previously mentioned enumeration process will never terminate!

- In other words, there is a procedure to check for FOL validity

- Is there an algorithm to check for FOL validity?

No! Reduction from PCP

- An undecidable problem involving arrangement of tiles
- Undecidable problem
  - There is no TM to solve a PCP instance
- We build a FOL sentence from a PCP instance such that the FOL sentence is valid IFF the PCP system has a solution!

- https://colab.research.google.com/github/ganeshutah/Jove/blob/master/For_CS3100_Fall2020/19_Algo_Proc_PCP/CH15.ipynb

PCP

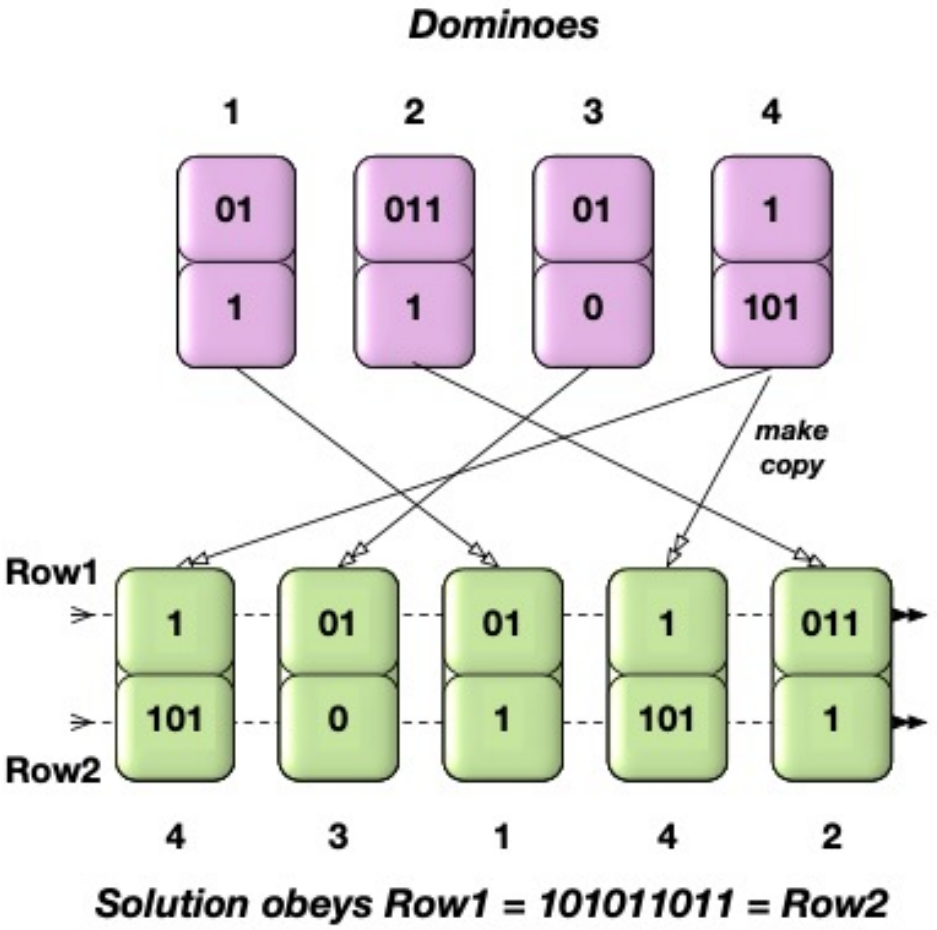

Figure 15.1: A PCP instance and *one* of its solutions. PCP helps show the undecidability of important problems such as grammar ambiguity, validity of first order logic sentences, and context-sensitive data dependence analysis.

PCP

```
1  pcp_solve([('110', '1'), ('1', '0'), ('0', '110')], OWN_INSTALL)
```

```
Detected platform linux
 Running the command ... :  ['./Jove/jove/pcp', '-i', 'temp.txt']
Solution(s) to PCP instance are below. Note: the tiles may be reversed,
  as the solver may sometimes present the solution in reverse.:

Solution 1
[3, 2, 2, 3, 3, 1, 3, 2, 2, 2, 1, 3, 3, 3, 2, 1, 3, 2, 2, 1, 3, 2, 1, 3, 3
0   1   1   0   0   011 0   1   1   1   011 0   0   0   1
011 0   0   011 011 1   011 0   0   0   1   011 011 011 0

011 0   1   1   011 0   1   011 0   0   1   011 0   1   0
1   011 0   0   1   011 0   1   011 011 0   1   011 0   011

1   1   011 0   1   011 0   0   1   1   0   0   1   011 0
0   0   1   011 0   1   011 011 0   0   011 011 0   1   011

1   011 011 0   0   011 011 0   1   011 011 011 011 1   1
0   1   1   011 011 1   1   011 0   1   1   1   1   0   0

011 011 1   1   0   011 0   0   011 1   011 011 1   011 011
1   1   0   0   011 1   011 011 1   0   1   1   0   1   1
```

PCP : Run from here

https://github.com/ganeshutah/Jove/blob/master/For_CS3100_Fall2020/19_Algo_Proc_PCP/CH15.ipynb

Undecidability of Validity

Full proof on Monday

PLEASE SEE https://www.overleaf.com/read/sqgffsctwtsy for a FULL proof that you can check off!

Read Book-1's proof (taken from Manna who credits Floyd's elegant proof)

It is a mapping reduction frop PCP to FOL-validity!

(all this in my version of CS 3100 – I tend to cover PCP always!)

FOL Validity is undecidable

- Proof: Reduction from PCP
- Given a PCP instance, build a specific FOL formula
- Argue that the
  - FOL formula obtained via the translation (mapping reduction) from a given PCP instance { (alpha_1, beta_1), (alpha_2, beta_2), … }
    - is valid
  - IFF the PCP instance is solvable

FOL Validity (from Book-1, Manna, Floyd)

Validity stems from the innate structure of the formula, as it must remain true under *every conceivable* interpretation. We will now summarize Floyd's proof (given in [78]) that the validity problem for first-order logic is undecidable. First, an abbreviation: for $\sigma_i \in \{0, 1\}$, use the abbreviation

$$f_{\sigma_1,\sigma_2,\ldots,\sigma_n}(a) = f_{\sigma_n}(f_{\sigma_{n-1}}(\ldots f_{\sigma_1}(a))\ldots).$$

Keep these definitions handy!
Answer by Zoom Text!

- Keep the definitions of alpha_i and beta_i handy
  - They are bit strings that describe the contents of the dominoes
  - E.g. given the dominoes [(110,1), (1,0), (0, 110)]
    - Draw the dominoes below

  - Alpha_1 is 110 and Beta_1 is 1: What are
    - Alpha_2 ?
    - Beta_2 ?
    - Alpha_3 ?
    - Beta_3 ?

  - What is f_{alpha_1} (a) ?

  - What is f_{beta_3} (b) ?

FOL Validity (from Book-1, Manna, Floyd)

Given a Post system $S = \{(\alpha_1, \beta_1), (\alpha_2, \beta_2), \ldots, (\alpha_n, \beta_n),\ n \geq 1$ over $\Sigma = \{0, 1\}$, construct the wff W_S (we will refer to the two antecedents of W_S as A1 and A2, and its consequent as C1):

$$\wedge_{i=1}^{n}\, p(f_{\alpha_i}(a), f_{\beta_i}(a)) \qquad \text{(A1)}$$

$$\wedge \quad \forall x\, \forall y\, [p(x, y) \Rightarrow \wedge_{i=1}^{n}\, p(f_{\alpha_i}(x), f_{\beta_i}(y))] \qquad \text{(A2)}$$

$$\Rightarrow \exists z\, p(z, z) \qquad \text{(C1)}$$

Keep these aside! We will need ‘em later

- A1
- A2
- C1

FOL Validity (from Book-1, Manna, Floyd)

We now prove that S has a solution iff W_S is valid.

Part 1. (W_S valid) $\Rightarrow$ (S has a solution).

If valid, it is true for all interpretations. Pick the following interpretation:

$a = \varepsilon$

$f_0(x) = x0$ (string 'x' and string '0' concatenated)

$f_1(x) = x1$ (similar to the above)

$p(x, y) =$ There exists a non-empty sequence $i_1 i_2 \ldots i_m$ such that

$$x = \alpha_{i_1} \alpha_{i_2} \ldots \alpha_{i_m} \text{ and } y = \beta_{i_1} \beta_{i_2} \ldots \beta_{i_m}$$

Under this interpretation, parts A1 and A2 of W_S are true. Here is

Details

- We are assuming that W_S, the FOL formula obtained by translating the PCP instance S is valid

- Thus S is true under all interpretations
- Thus we can choose any interpretation we like (to finish our proof)

- Recall (from a previous lecture) that given a formula, we can choose the domain of interpretation D to be Nat or Sigma* and choose the function and predicate symbols' meanings to be anything at all over that domain (those slides are provided next, to jog your memory)

True under this interpretation? Your answer?

Consider the formula

$$Fmla1 = \exists F.F(a) = b \\ \wedge(\forall x).[p(x) \Rightarrow F(x) = g(x, F(f(x)))]$$

For this formula, it was true
Under two interpretations
And false under the third

Which one?

We will now provide *three distinct* interpretations for it.

Interpretation 1.

D = Nat
a = 0
b = 1
f = $\lambda x.(x = 0 \rightarrow 0, x - 1)$
g = *
p = $\lambda x.x > 0$

Interpretation 2.

D = Σ^*
a = ε
b = ε
f = $\lambda x.(tail(x))$
g(x,y) = concat(y,head(x))
p = $\lambda x.x \neq \varepsilon$

Interpretation 3.

D = Nat
a = 0
b = 1
f = $\lambda x.x$
g(x,y) = y+1
p = $\lambda x.x > 0$

But we are now working on W_S being valid

Given a Post system $\mathrm{S} = \{(\alpha_1, \beta_1), (\alpha_2, \beta_2), \ldots, (\alpha_n, \beta_n),\ n \geq 1$ over $\Sigma = \{0, 1\}$, construct the wff W_S (we will refer to the two antecedents of W_S as A1 and A2, and its consequent as C1):

$$\wedge_{i=1}^{n}\, p(f_{\alpha_i}(a), f_{\beta_i}(a)) \qquad \text{(A1)}$$

$$\wedge \quad \forall x\, \forall y\, [p(x, y) \Rightarrow \wedge_{i=1}^{n}\, p(f_{\alpha_i}(x), f_{\beta_i}(y))] \qquad \text{(A2)}$$

$$\Rightarrow \exists z\, p(z, z) \qquad \text{(C1)}$$

That means, under ANY interpretation, it is the case that

A1 /\ A2 => C1 is true

Proof setup for W_S is valid ➔ S is solvable

Thus, we are going to interpret A1 /\ A2 => C1 below, where…

$$\wedge_{i=1}^{n}\, p(f_{\alpha_i}(a), f_{\beta_i}(a)) \qquad \text{(A1)}$$

$$\wedge \quad \forall x\, \forall y\, [p(x,y) \Rightarrow \wedge_{i=1}^{n}\, p(f_{\alpha_i}(x), f_{\beta_i}(y))] \qquad \text{(A2)}$$

$$\Rightarrow \exists z\, p(z,z) \qquad \text{(C1)}$$

Under THIS interpretation --- call it PI (for Post Interpretation!!)

$a = \varepsilon$

$f_0(x) = x0$ (string 'x' and string '0' concatenated)

$f_1(x) = x1$ (similar to the above)

$p(x,y) =$ There exists a non-empty sequence $i_1 i_2 \ldots i_m$ such that

$$x = \alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_m} \text{ and } y = \beta_{i_1}\beta_{i_2}\ldots\beta_{i_m}$$

Why is A1 true under PI? Let us break this down!

Thus, we are going to interpret A1 /\ A2 => C1 below, where…

$$\wedge_{i=1}^{n}\, p(f_{\alpha_i}(a), f_{\beta_i}(a)) \qquad \text{(A1)}$$

$$\wedge \quad \forall x\, \forall y\, [p(x,y) \Rightarrow \wedge_{i=1}^{n}\, p(f_{\alpha_i}(x), f_{\beta_i}(y))] \qquad \text{(A2)}$$

$$\Rightarrow \exists z\, p(z,z) \qquad \text{(C1)}$$

Under THIS interpretation!

$a = \varepsilon$

$f_0(x) = x0$ (string 'x' and string '0' concatenated)

$f_1(x) = x1$ (similar to the above)

$p(x,y) =$ There exists a non-empty sequence $i_1 i_2 \ldots i_m$ such that

$$x = \alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_m} \text{ and } y = \beta_{i_1}\beta_{i_2}\ldots\beta_{i_m}$$

Why is p (f_{alpha_1}(a), f_{beta_1}(a)) true?

Thus, we are going to interpret A1 /\ A2 => C1 below, where…

$$\wedge_{i=1}^{n}\, p(f_{\alpha_i}(a), f_{\beta_i}(a)) \qquad \text{(A1)}$$

$$\wedge \quad \forall x\, \forall y\, [p(x,y) \Rightarrow \wedge_{i=1}^{n}\, p(f_{\alpha_i}(x), f_{\beta_i}(y))] \qquad \text{(A2)}$$

$$\Rightarrow \exists z\, p(z,z) \qquad \text{(C1)}$$

Under THIS interpretation!

$a = \varepsilon$

$f_0(x) = x0$ (string 'x' and string '0' concatenated)

$f_1(x) = x1$ (similar to the above)

$p(x,y) =$ There exists a non-empty sequence $i_1 i_2 \ldots i_m$ such that

$$x = \alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_m} \text{ and } y = \beta_{i_1}\beta_{i_2}\ldots\beta_{i_m}$$

Why is p (f_{alpha_1}(a), f_{beta_1}(a)) true? Ans: Because it reduces to p (alpha_1, beta_1) !!!

Thus, we are going to interpret A1 /\ A2 => C1 below, where…

$$\wedge_{i=1}^{n}\, p(f_{\alpha_i}(a), f_{\beta_i}(a)) \qquad \text{(A1)}$$

$$\wedge \quad \forall x\, \forall y\, [p(x,y) \Rightarrow \wedge_{i=1}^{n}\, p(f_{\alpha_i}(x), f_{\beta_i}(y))] \qquad \text{(A2)}$$

$$\Rightarrow \exists z\, p(z,z) \qquad \text{(C1)}$$

Under THIS interpretation!

$a = \varepsilon$

$f_0(x) = x0$ (string 'x' and string '0' concatenated)

$f_1(x) = x1$ (similar to the above)

$p(x,y) =$ There exists a non-empty sequence $i_1 i_2 \ldots i_m$ such that

$$x = \alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_m} \text{ and } y = \beta_{i_1}\beta_{i_2}\ldots\beta_{i_m}$$

Why is p (f_{alpha_2}(a), f_{beta_2}(a)) true? Ans: YOU PLEASE TYPE IN !!

Thus, we are going to interpret A1 /\ A2 => C1 below, where…

$$\wedge_{i=1}^{n}\, p(f_{\alpha_i}(a), f_{\beta_i}(a)) \qquad \text{(A1)}$$

$$\wedge \quad \forall x\, \forall y\, [p(x,y) \Rightarrow \wedge_{i=1}^{n}\, p(f_{\alpha_i}(x), f_{\beta_i}(y))] \qquad \text{(A2)}$$

$$\Rightarrow \exists z\, p(z,z) \qquad \text{(C1)}$$

Under THIS interpretation!

$a = \varepsilon$

$f_0(x) = x0$ (string 'x' and string '0' concatenated)

$f_1(x) = x1$ (similar to the above)

$p(x,y) =$ There exists a non-empty sequence $i_1 i_2 \ldots i_m$ such that

$$x = \alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_m} \text{ and } y = \beta_{i_1}\beta_{i_2}\ldots\beta_{i_m}$$

The conjunction of all these primitive applications of p are true! Thus A1 is true!

Thus, we are going to interpret A1 /\ A2 => C1 below, where…

$$\wedge_{i=1}^{n}\, p(f_{\alpha_i}(a), f_{\beta_i}(a)) \qquad \text{(A1)}$$

$$\wedge \quad \forall x\, \forall y\, [p(x,y) \Rightarrow \wedge_{i=1}^{n}\, p(f_{\alpha_i}(x), f_{\beta_i}(y))] \qquad \text{(A2)}$$

$$\Rightarrow \exists z\, p(z,z) \qquad \text{(C1)}$$

Under THIS interpretation!

$a = \varepsilon$

$f_0(x) = x0$ (string 'x' and string '0' concatenated)

$f_1(x) = x1$ (similar to the above)

$p(x,y) =$ There exists a non-empty sequence $i_1 i_2 \ldots i_m$ such that

$$x = \alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_m} \text{ and } y = \beta_{i_1}\beta_{i_2}\ldots\beta_{i_m}$$

Now why is one p(x,y) => p(f_{alpha_i}(x), f_{beta_i}(x)) true?

Thus, we are going to interpret A1 /\ A2 => C1 below, where…

$$\wedge_{i=1}^{n}\, p(f_{\alpha_i}(a), f_{\beta_i}(a)) \qquad \text{(A1)}$$

$$\wedge \quad \forall x\, \forall y\, [p(x,y) \Rightarrow \wedge_{i=1}^{n}\, p(f_{\alpha_i}(x), f_{\beta_i}(y))] \qquad \text{(A2)}$$

$$\Rightarrow \exists z\, p(z,z) \qquad \text{(C1)}$$

Under THIS interpretation!

$a = \varepsilon$

$f_0(x) = x0$ (string 'x' and string '0' concatenated)

$f_1(x) = x1$ (similar to the above)

$p(x,y) =$ There exists a non-empty sequence $i_1 i_2 \ldots i_m$ such that

$$x = \alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_m} \text{ and } y = \beta_{i_1}\beta_{i_2}\ldots\beta_{i_m}$$

Now why is one p(x,y) => p(f_{alpha_i}(x), f_{beta_i}(x)) true? ANS: see what “f” and “p” mean !!

Thus, we are going to interpret A1 /\ A2 => C1 below, where…

$$\wedge_{i=1}^{n}\, p(f_{\alpha_i}(a), f_{\beta_i}(a)) \qquad \text{(A1)}$$

$$\wedge \quad \forall x\, \forall y\, [p(x,y) \Rightarrow \wedge_{i=1}^{n}\, p(f_{\alpha_i}(x), f_{\beta_i}(y))] \qquad \text{(A2)}$$

$$\Rightarrow \exists z\, p(z,z) \qquad \text{(C1)}$$

Under THIS interpretation!

$a = \varepsilon$

$f_0(x) = x0$ (string ‘x’ and string ‘0’ concatenated)

$f_1(x) = x1$ (similar to the above)

$p(x,y) =$ There exists a non-empty sequence $i_1 i_2 \ldots i_m$ such that

$$x = \alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_m} \text{ and } y = \beta_{i_1}\beta_{i_2}\ldots\beta_{i_m}$$

Now why is one p(x,y) => p(f_{alpha_i}(x), f_{beta_i}(x)) true? ANS: Basically, A2 keeps extending the alpha/beta sequences and they are in the matching order! Thus the whole nested A2 conjunction is true!

Thus, we are going to interpret A1 /\ A2 => C1 below, where…

$$\wedge_{i=1}^{n}\, p(f_{\alpha_i}(a), f_{\beta_i}(a)) \qquad \text{(A1)}$$

$$\wedge \quad \forall x\, \forall y\, [p(x,y) \Rightarrow \wedge_{i=1}^{n}\, p(f_{\alpha_i}(x), f_{\beta_i}(y))] \qquad \text{(A2)}$$

$$\Rightarrow \exists z\, p(z,z) \qquad \text{(C1)}$$

Under THIS interpretation!

$a = \varepsilon$

$f_0(x) = x0$ (string 'x' and string '0' concatenated)

$f_1(x) = x1$ (similar to the above)

$p(x,y) =$ There exists a non-empty sequence $i_1 i_2 \ldots i_m$ such that

$$x = \alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_m} \text{ and } y = \beta_{i_1}\beta_{i_2}\ldots\beta_{i_m}$$

This means that A2 is true! We know that A1, A2 are true and A1, A2 => C1 is true [[why ??]] Thus what about C1 ??

Thus, we are going to interpret A1 /\ A2 => C1 below, where…

$$\wedge_{i=1}^{n}\, p(f_{\alpha_i}(a), f_{\beta_i}(a)) \qquad \text{(A1)}$$

$$\wedge \quad \forall x\, \forall y\, [p(x,y) \Rightarrow \wedge_{i=1}^{n}\, p(f_{\alpha_i}(x), f_{\beta_i}(y))] \qquad \text{(A2)}$$

$$\Rightarrow \exists z\, p(z,z) \qquad \text{(C1)}$$

Under THIS interpretation!

$a = \varepsilon$

$f_0(x) = x0$ (string 'x' and string '0' concatenated)

$f_1(x) = x1$ (similar to the above)

$p(x,y) =$ There exists a non-empty sequence $i_1 i_2 \ldots i_m$ such that

$$x = \alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_m} \text{ and } y = \beta_{i_1}\beta_{i_2}\ldots\beta_{i_m}$$

This means that A2 is true! We know that A1, A2 are true and A1, A2 => C1 is true [[why? W_S is valid!]] Thus what about C1 ?? True!

Thus, we are going to interpret A1 /\ A2 => C1 below, where…

$$\wedge_{i=1}^{n}\, p(f_{\alpha_i}(a), f_{\beta_i}(a)) \qquad \text{(A1)}$$

$$\wedge \quad \forall x\, \forall y\, [p(x,y) \Rightarrow \wedge_{i=1}^{n}\, p(f_{\alpha_i}(x), f_{\beta_i}(y))] \qquad \text{(A2)}$$

$$\Rightarrow \exists z\, p(z,z) \qquad \text{(C1)}$$

Under THIS interpretation!

$a = \varepsilon$

$f_0(x) = x0$ (string 'x' and string '0' concatenated)

$f_1(x) = x1$ (similar to the above)

$p(x,y) =$ There exists a non-empty sequence $i_1 i_2 \ldots i_m$ such that

$$x = \alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_m} \text{ and } y = \beta_{i_1}\beta_{i_2}\ldots\beta_{i_m}$$

What does C1 being true mean ?

Thus, we are going to interpret A1 /\ A2 => C1 below, where…

$$\wedge_{i=1}^{n}\, p(f_{\alpha_i}(a), f_{\beta_i}(a)) \qquad \text{(A1)}$$

$$\wedge \quad \forall x\, \forall y\, [p(x,y) \Rightarrow \wedge_{i=1}^{n}\, p(f_{\alpha_i}(x), f_{\beta_i}(y))] \qquad \text{(A2)}$$

$$\Rightarrow \exists z\, p(z,z) \qquad \text{(C1)}$$

Under THIS interpretation!

$a = \varepsilon$

$f_0(x) = x0$ (string 'x' and string '0' concatenated)

$f_1(x) = x1$ (similar to the above)

$p(x,y) =$ There exists a non-empty sequence $i_1 i_2 \ldots i_m$ such that

$$x = \alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_m} \text{ and } y = \beta_{i_1}\beta_{i_2}\ldots\beta_{i_m}$$

What does C1 being true mean ? Means that the PCP system S has a solution, namely z !!

Thus, we are going to interpret A1 /\ A2 => C1 below, where…

$$\wedge_{i=1}^{n}\, p(f_{\alpha_i}(a), f_{\beta_i}(a)) \qquad \text{(A1)}$$

$$\wedge \quad \forall x\, \forall y\, [p(x,y) \Rightarrow \wedge_{i=1}^{n}\, p(f_{\alpha_i}(x), f_{\beta_i}(y))] \qquad \text{(A2)}$$

$$\Rightarrow \exists z\, p(z,z) \qquad \text{(C1)}$$

Under THIS interpretation!

$a = \varepsilon$

$f_0(x) = x0$ (string 'x' and string '0' concatenated)

$f_1(x) = x1$ (similar to the above)

$p(x,y) =$ There exists a non-empty sequence $i_1 i_2 \ldots i_m$ such that

$$x = \alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_m} \text{ and } y = \beta_{i_1}\beta_{i_2}\ldots\beta_{i_m}$$

FOL Validity (from Book-1, Manna, Floyd)

- Under the above interpretation, $f_{\alpha_i}(a) = \varepsilon\alpha_i = \alpha_i$ and similarly $f_{\beta_i}(a) = \beta_i$.
- Thus A1 becomes $\wedge_{i=1}^{n}\, p(\alpha_i, \beta_i)$. Each conjunct in this formula is true by p's interpretation; hence A1 is true.
- The part $[p(x, y) \Rightarrow \wedge_{i=1}^{n}\, p(f_{\alpha_i}(x), f_{\beta_i}(y))]$ reduces to the following claim: $p(x, y)$ is true means that x and y can be written in the form $x = \alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_m}$ and $y = \beta_{i_1}\beta_{i_2}\ldots\beta_{i_m}$; the consequent of this implication then says that we can append some α_i and the corresponding β_i to x and y, respectively. The consequent is also true by p's interpretation. Thus A2 is also true.

FOL Validity (from Book-1, Manna, Floyd)

- Since W_S is valid (true), C1 must also be true. C1 asserts that the Post system S has a solution, namely some string z that lends itself to being interpreted as some sequence $\alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_m}$ as well as $\beta_{i_1}\beta_{i_2}\ldots\beta_{i_m}$. That is,

$$\alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_m} = z = \beta_{i_1}\beta_{i_2}\ldots\beta_{i_m}.$$

FOL Validity (from Book-1, Manna, Floyd)

Part 2. (W_S valid) $\Leftarrow$ (S has a solution).

If S has a solution, let it be the sequence $i_1 i_2 \ldots i_m$. In other words, $\alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_m} = \beta_{i_1}\beta_{i_2}\ldots\beta_{i_m} = Soln$. Now, in order to show that W_S is valid, we must show that for *every* interpretation it is true. We approach this goal by showing that under every interpretation where the antecedents of W_S, namely A1 and A2, are true, the consequent, namely C1, is also true (if any antecedent is false, W_S is true, so this case is not considered).

From A1, we conclude that

$$p(f_{\alpha_{i_1}}(a), f_{\beta_{i_1}}(a))$$

Notes

- Here we have to show that W_S is valid, given that S has a solution
- If S has a solution, there exists a sequence
  - Alpha_{i1} Alpha_{i2} … Alpha_{iN} that equals a sequence
  - Beta_{i1} Beta_{i2} … Beta_{iN}
- Thus, we proceed as follows

Notes

- W_S being valid involves only one really interesting case
  - making A1 and A2 true
  - Why?

Notes

- W_S being valid involves only one really interesting case
  - making A1 and A2 true
  - Why?

- Because A1 and A2 being false renders W_S true ☺

Notes

- But if A1 and A2 are true, we can use A1 as “axioms” and A2 as a whole bunch of implications
- We can apply modus ponens using A1 and A2 instances
- Then we can pump up the “f” nests
- Pump it up to match the PCP solution given to us !!!

FOL Validity (from Book-1, Manna, Floyd)

is true. Now using A2 as a rule of inference, we can conclude through Modus ponens, that

$$p(f_{\alpha_{i_2}}(f_{\alpha_{i_1}}(a)), f_{\beta_{i_2}}(f_{\beta_{i_1}}(a))$$

is true. In other words,

$$p(f_{\alpha_{i_1}\alpha_{i_2}}(a), f_{\beta_{i_1}\beta_{i_2}}(a))$$

FOL Validity (from Book-1, Manna, Floyd)

is true. We continue this way, applying the functions in the order dictated by the assumed solution for S; in other words, we arrive at the assertion that the following is true (notice that the subscripts of f describe the order in which the solution to S considers the α's and β's):

$$p(f_{\alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_m}}(a), f_{\beta_{i_1}\beta_{i_2}\ldots\beta_{i_m}}(a)).$$

Which z shall we pick?

FOL Validity (from Book-1, Manna, Floyd)

However, since

$$\alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_m} = \beta_{i_1}\beta_{i_2}\ldots\beta_{i_m} = Soln,$$

we have essentially shown that

$$p(f_{Soln}(a), f_{Soln}(a)).$$

Now, $p(f_{Soln}(a), f_{Soln}(a))$ means that there exists a z such that $p(z, z)$, namely $z = f_{Soln}(a)$. □

Which z shall we pick?

- The z picked is the nest of function applications we built up!

open *FOL

Alloy syntax details

3.4.3.3 Box Join

The box operator *[]* is semantically identical to join, but takes its arguments in a different order, and has different precedence. The expression

```
e1 [e2]
```

has the same meaning as

```
e2.e1
```

Example. Given a relation *address* from names to addresses, and a scalar *n* representing a name, the expression *address[n]* is equivalent to *n.address*, and denotes the set of addresses that *n* is mapped to.

Dot binds more tightly than box, however, so

Alloy syntax details

a.b.c [d]

is short for

d.(a.b.c)

3.5.1 Logical Operators

There are two forms of each logical operator: a shorthand and a verbose form (similar to the operators used in boolean expressions in programming languages):

· **not**	!	negation
· **and**	&&	conjunction
· **or**	||	disjunction
· **implies**	=>	implication
· **else**	,	alternative
· **iff**	<=>	bi-implication

Alloy syntax details

3.5.2 Quantification

A quantified constraint takes the form

```
Q x: e | F
```

where *F* is a constraint that contains the variable *x*, *e* is an expression bounding *x*, and *Q* is a quantifier.

The forms of quantification in Alloy are

- **all** *x: e | F* — *F* holds for every *x* in *e*;
- **some** *x: e | F* — *F* holds for some *x* in *e*;
- **no** *x: e | F* — *F* holds for no *x* in *e*;
- **lone** *x: e | F* — *F* holds for at most one *x* in *e*;
- **one** *x: e | F* — *F* holds for exactly one *x* in *e*.

To remember what *lone* means, it might help to think of it as being short for "less than or equal to one."

Alloy syntax details

Examples. Given a set *Address* of email addresses, *Name* of names, and a relation *address* representing a multilevel address book mapping names to names and addresses,

- **some** n: Name, a: Address | a **in** n.address
  says that some name maps to some address (that is, the address book is not empty);
- **no** n: Name | n **in** n.^address
  says that no name can be reached by lookups from itself (that is, there are no cycles in the address book);
- **all** n: Name | **lone** d: Address | d **in** n.address
  says that every name maps to at most one address;
- **all** n: Name | **no disj** d, d': Address | d + d' **in** n.address
  says the same thing, but slightly differently: that for every name, there is no pair of distinct addresses that are among the results obtained by looking up the name.

Quantifiers can be applied to expressions too:

- **some** *e* — *e* has some tuples;
- **no** *e* — *e* has no tuples;
- **lone** *e* — *e* has at most one tuple;
- **one** *e* — *e* has exactly one tuple.

Alloy syntax details

Examples. Using the sets and relation from the previous example,

- **some** Name
  says that the set of names is not empty;
- **some** address
  says that the address book is not empty: there is some pair mapping a name to an address;
- **no** (address.Addr - Name)
  says that nothing is mapped to addresses except for names;
- **all** n: Name | **lone** n.address
  says that every name maps to at most one address (more succinctly than in the previous example);
- **all** n: Name | **one** n.address **or no** n.address
  says the same thing.

Alloy syntax details

3.6 Declarations and Multiplicity Constraints

A declaration introduces a relation name. We've just seen how declarations are used in quantified constraints and comprehensions. Free-standing declarations of relation names make sense too, although we'll see in chapter 4 how, in the full Alloy language, these would instead be declared within "signatures."

Alloy syntax

3.6.2 Set Multiplicities

In the last subsection, I said that the meaning of a declaration

x: e

was almost the same as the meaning of a subset constraint

x **in** e

Now the caveat: the declaration can include *multiplicity constraints*, which are sometimes implicit. Multiplicities are expressed with the *multiplicity keywords*:

- **set** any number
- **one** exactly one
- **lone** zero or one
- **some** one or more

Note that *one*, *lone*, and *some* are the same keywords used for quantification.

The meaning of a declaration depends on the arity of the bounding expression. If it denotes a set (that is, is unary), it can be prefixed by a multiplicity keyword like this

x: *m* e

which constrains the size of the set *x* according to *m*. For a set-valued bounding expression, omitting the keyword is the same as writing *one*. So if no keyword appears, the declaration makes the variable a scalar.

Examples

- RecentlyUsed: **set** Name
  says that *RecentlyUsed* is a subset of the set *Name*;
- senderAddress: Addr
  says that *senderAddress* is a scalar in the set *Addr*;
- senderName: **lone** Name
  says that *senderName* is an option: either a scalar in the set *Name*, or empty;

Alloy syn

3.6.3 Relation Multiplicities

When the bounding expression is a relation (that is, a relation with arity greater than one), it may not be preceded by a multiplicity keyword. But if the bounding expression is constructed with the arrow operator, multiplicities can appear inside it. Suppose the declaration looks like this:

r: A *m* -> *n* B

where *m* and *n* are multiplicity keywords (and where *A* and *B* are, for now, sets). Then the relation *r* is constrained to map each member of *A* to *n* members of *B*, and to map *m* members of *A* to each member of *B*.

Such a declaration can indicate the domain and range of the relation (see subsection 3.2.5), and whether or not it is functional or injective (see subsection 3.2.4):

- r: A -> **one** B
  a function whose domain is *A*;
- r: A **one** -> B
  an injective relation whose range is *B*;
- r: A -> **lone** B
  a function that is partial over the domain *A*;
- r: A **one** -> **one** B
  an injective function with domain *A* and range *B*, also called a bijection from *A* to *B*;
- r: A **some** -> **some** B
  a relation with domain *A* and range *B*.

Alloy syntax details

Alloy syntax details

Alloy syntax details

Alloy syntax details