SJSU Students
11/18/2022

**HW 4**

**1. Express each of the following as first order logic formula in the language with constant 0, function symbols MakeTree, LeftTree, RightTree and predicates Equals, and BinaryTree:**

**(a) 0 is a binary tree**

**(b) if x, y are binary trees, then so is MakeTree(x,y),**

**(c) z equals itself**

**(d) 0 is not equal to MakeTree(x,y),**

**(e) if z equals MakeTree(x,y) then LeftTree(z) equals x and RightTree(z) equals y,**

**(f) if x is a binary tree and x is not equal to 0, then LeftTree(x) and RightTree(x) are binary trees.**

a) BinaryTree(0)
b) ∀x,y (BinaryTree(x)∧ BinaryTree(y)) ⇒ BinaryTree(MakeTree(x,y)))
c) Equals(z, z)
d) ¬Equals(0, MakeTree(x,y))
e) ∀z Equals(z, MakeTree(x,y)) ⇒ (Equals(LeftTree(z), x)∧Equals(RightTree(z), y))
f) ∀x (BinaryTree(x)∧¬Equals(x,0)) ⇒ (BinaryTree(LeftTree(x))∧BinaryTree(RightTree(x)))

**2. Using our Natural Deduction system extended by rules for First-order logic, assuming (a)-(f) of problem 1 as our knowledge base, give a formal proof of the formula α := there exists an x such that x is a binary tree and LeftTree(LeftTree(RightTree(x))) equals 0.**

**Prove:** α := ∃x (BinaryTree(x) ∧ Equals(LeftTree(LeftTree(RightTree(x))),0))

| | | |
|---|---|---|
| R1 | BinaryTree(0) | KB |
| R2 | ∀x,y (BinaryTree(x)∧ BinaryTree(y)) ⇒ BinaryTree(MakeTree(x,y))) | KB |
| R3 | Equals(z, z) | KB |
| R4 | ¬Equals(0, MakeTree(x,y)) | KB |

| | | |
|---|---|---|
| R5 | ∀z Equals(z, MakeTree(x,y)) ⇒ (Equals(LeftTree(z), x)ΛEquals(RightTree(z), y)) | KB |
| R6 | ∀x (BinaryTree(x) Λ ¬Equals(x,0)) ⇒ (BinaryTree(LeftTree(x)) Λ BinaryTree(RightTree(x))) | KB |
| R7 | Equals(MakeTree(0,0), MakeTree(x,y)) ⇒ (Equals(LeftTree(MakeTree(0,0)), x)ΛEquals(RightTree(MakeTree(0,0)), y)) | For All Elimination to R5 |
| R8 | (Equals(LeftTree(MakeTree(0,0)), x)ΛEquals(RightTree(MakeTree(0,0)), y)) | Modus Ponen R3 and R7 |
| R9 | Equals(RightTree(MakeTree(0,0)), x) | And Elimination on R8 |
| R10 | Equals(RightTree(MakeTree(0,0)), MakeTree(x,y)) ⇒ (Equals(LeftTree(RightTree(MakeTree(0,0))), x)ΛEquals(RightTree(RightTree(MakeTree(0,0))), y)) | For All Elimination to R5 |
| R11 | Equals(LeftTree(RightTree(MakeTree(0,0))), x)ΛEquals(RightTree(RightTree(MakeTree(0,0))), y) | Modus Ponen R9 and R10 |
| R12 | Equals(LeftTree(RightTree(MakeTree(0,0))), x) | And elimination on R11 |
| R13 | Equals(LeftTree(RightTree(MakeTree(0,0))), MakeTree(x,y)) ⇒ (Equals(LeftTree(LeftTree(RightTree(MakeTree(0,0)))), x)ΛEquals(RightTree(LeftTree(RightTree(MakeTree(0,0)))), y)) | For All Elimination to R5 |
| R14 | Equals(LeftTree(LeftTree(RightTree(MakeTree(0,0)))), x) | Modus Ponen R12 and R13 |
| R15 | (BinaryTree(0) Λ BinaryTree(0)) ⇒ BinaryTree(MakeTree(0,0))) | For All Elimination to R2 |
| R16 | BinaryTree(MakeTree(0,0)) | Modus Ponen to R1 and R7 |
| R17 | BinaryTree(MakeTree(0,0)) Λ Equals(LeftTree(LeftTree(RightTree(MakeTree(0,0)))), x) | And Introduction of R14 and R16 |

| | | |
|---|---|---|
| R18 | $\exists$x (BinaryTree(x) $\wedge$ Equals(LeftTree(LeftTree(RightTree(x))), 0)) | Apply Exist introduction to R17 |

**3. Let the formulas of Problem 1 be our KB and α be as in Problem 2. Skolemize the formulas in KB and ¬α, convert the result to CNF, and then clauses. Finally, find a resolution refutation. For at least one place where you needed to do unification carefully show the steps the algorithm from class would use.**

### Skolemize each formula in KB and ¬α:

a) BinaryTree(0)
b) ∀x,y (BinaryTree(x)∧BinaryTree(y)) ⇒ BinaryTree(MakeTree(x,y)))
c) Equals(z, z)
d) ¬Equals(0, MakeTree(x,y))
e) ∀z Equals(z, MakeTree(x,y)) ⇒ (Equals(LeftTree(z), x)∧Equals(RightTree(z), y))
f) ∀x (BinaryTree(x)∧¬Equals(x,0)) ⇒ (BinaryTree(LeftTree(x))∧BinaryTree(RightTree(x)))
¬α) ∀x (¬BinaryTree(x)∨¬Equals(LeftTree(LeftTree(RightTree(x))),0))

### Convert the result to CNF

a) BinaryTree(0)
b) ¬BinaryTree(x)∨¬ BinaryTree(y)∨BinaryTree(MakeTree(x,y))
c) Equals(z, z)
d) ¬Equals(0, MakeTree(x,y))
e1) ¬Equals(z, MakeTree(x,y))∨Equals(LeftTree(z), x)
e2) ¬Equals(z, MakeTree(x,y))∨Equals(RightTree(z), y)
f1) ¬BinaryTree(x)∨Equals(x,0)∨BinaryTree(LeftTree(x)
f2) ¬BinaryTree(x)∨Equals(x,0)∨BinaryTree(RightTree(x))
¬α) ¬BinaryTree(x)∨¬Equals(LeftTree(LeftTree(RightTree(x))),0)

### Convert to clauses

a) {BinaryTree(0)}
b) {¬BinaryTree(x), ¬BinaryTree(y), BinaryTree(MakeTree(x,y))}
c) {Equals(z,z)}
d) {¬Equals(0, MakeTree(x,y))}
e1) {¬Equals(z, MakeTree(x,y)), Equals(LeftTree(z), x) }
e2) {¬Equals(z, MakeTree(x,y)), Equals(RightTree(z), y)}

f1) {¬BinaryTree(x) , Equals(x,0), BinaryTree(LeftTree(x))}
f2) {¬BinaryTree(x) , Equals(x,0), BinaryTree(RightTree(x))}
¬α) {¬BinaryTree(x) , ¬Equals(LeftTree(LeftTree(RightTree(x))), 0)}

**Resolution Refutation**

| | | |
|---|---|---|
| R1 | {BinaryTree(0)} | KB |
| R2 | {¬BinaryTree(x), ¬BinaryTree(y), BinaryTree(MakeTree(x,y))} | KB |
| R3 | {Equals(z,z)} | KB |
| R4 | {¬Equals(0, MakeTree(x,y))} | KB |
| R5 | {¬Equals(z, MakeTree(x,y)), Equals(LeftTree(z), x)} | KB |
| R6 | {¬Equals(z, MakeTree(x,y)), Equals(RightTree(z), y)} | KB |
| R7 | {¬BinaryTree(x), Equals(x,0), BinaryTree(LeftTree(x))} | KB |
| R8 | {¬BinaryTree(x) , Equals(x,0), BinaryTree(RightTree(x))} | KB |
| R9 | {¬BinaryTree(x), ¬Equals(LeftTree(LeftTree(RightTree(x))), 0)} | KB |
| R10 | {¬BinaryTree(y), BinaryTree(MakeTree(0,y))} | Resolve from R1 and R2, x->0 |
| R11 | {¬BinaryTree(x), BinaryTree(MakeTree(x,0))} | Resolve from R1 and R2, y->0 |
| R12 | {¬BinaryTree(MakeTree(x,0)), BinaryTree(MakeTree(0,MakeTree(x,0)))} | Resolve from R10 and R11, y->MakeTree(x,0) |
| R13 | {BinaryTree(MakeTree(0,0))} | Resolve from R1 and R11, x->0 |
| R14 | {BinaryTree(MakeTree(0,MakeTree(MakeTree(0,0),0))) | Resolve from R12 and R13, x->MakeTree(0,0) |
| R15 | {¬Equals(LeftTree(LeftTree(RightTree(MakeTree(0,MakeTree(MakeTree(0,0),0))))), 0)} | Resolve from R9 and R14, x->MakeTree(0,MakeTree(MakeTree(0,0),0)) |
| R16 | {¬Equals(LeftTree(RightTree(MakeTree(0,MakeTree(MakeTree(0,0),0)))), MakeTree(0,y))} | Resolve from R5 and R15, z->LeftTree(RightTree(MakeTree(0,MakeTree(MakeTree(0,0),0)))), x->0 |

| | | |
|---|---|---|
| R17 | {¬Equals(RightTree(MakeTree(0,MakeTree(MakeTree(0,0),0))), MakeTree(MakeTree(0,y),y)) } | Resolve from R5 and R16, z->RightTree(MakeTree(0, MakeTree(MakeTree(0,0), 0))), x->MakeTree(0,y) |
| R18 | {¬Equals(MakeTree(0,MakeTree(MakeTree(0,0),0)), MakeTree(x,MakeTree(MakeTree(0,0), 0))) } | Resolve from R6 and R17, z->MakeTree(0,MakeTree( MakeTree(0,0),0)), y->MakeTree(MakeTree(0, 0), 0) |
| R19 | { } | Resolve from R3 and R18, z->MakeTree(x,MakeTree( MakeTree(0,0), 0)), x->0 |

**Sample of using Unification Algorithm:**
**For the resolved clause R13**

1. Call Unified(BinaryTree(x), BinaryTree(0), {})
2. Both BinaryTree(x), BinaryTree(0) are terms
   a. Call Unify(args(BinaryTree(x)), args(BinaryTree(0)), Unify(op(BinaryTree(x)), op(BinaryTree(0)), {}))
   b. Calculate arguments:
      i. args(BinaryTree(x)) returns x
      ii. args (BinaryTree(0)) return 0
      iii. op(BinaryTree(x)) returns BinaryTree(a)
      iv. op(BinaryTree(0)) returns BinaryTree(a)
      v. op(BinaryString(0) == BinaryString(a), Unify(op(BinaryTree(x)), op(BinaryTree(0)), {}) return {}
   c. Function call in a) becomes Unify(x, 0, {})
   d. X is a variable
      i. Unify-var(x, 0)
      ii. S is empty, and Occur-Check(x, 0) == False, return (x |-> 0)
   e. Return (x |-> 0)

**4. Pretend your parents want you to change the sheets on your king size bed with two pillows. Imagine all the different things you might need to choose between, put on, or remove from your bed to accomplish this daunting task. Model this as a PDDL problem. Then use the GraphPlan algorithm to find a solution.**

**PDDL Problem:**

**Init**( On(oldSheets, bed) ∧ On(pillow1, oldSheets) ∧ On(pillow2, oldSheets)
∧ On(newSheets, ground) )

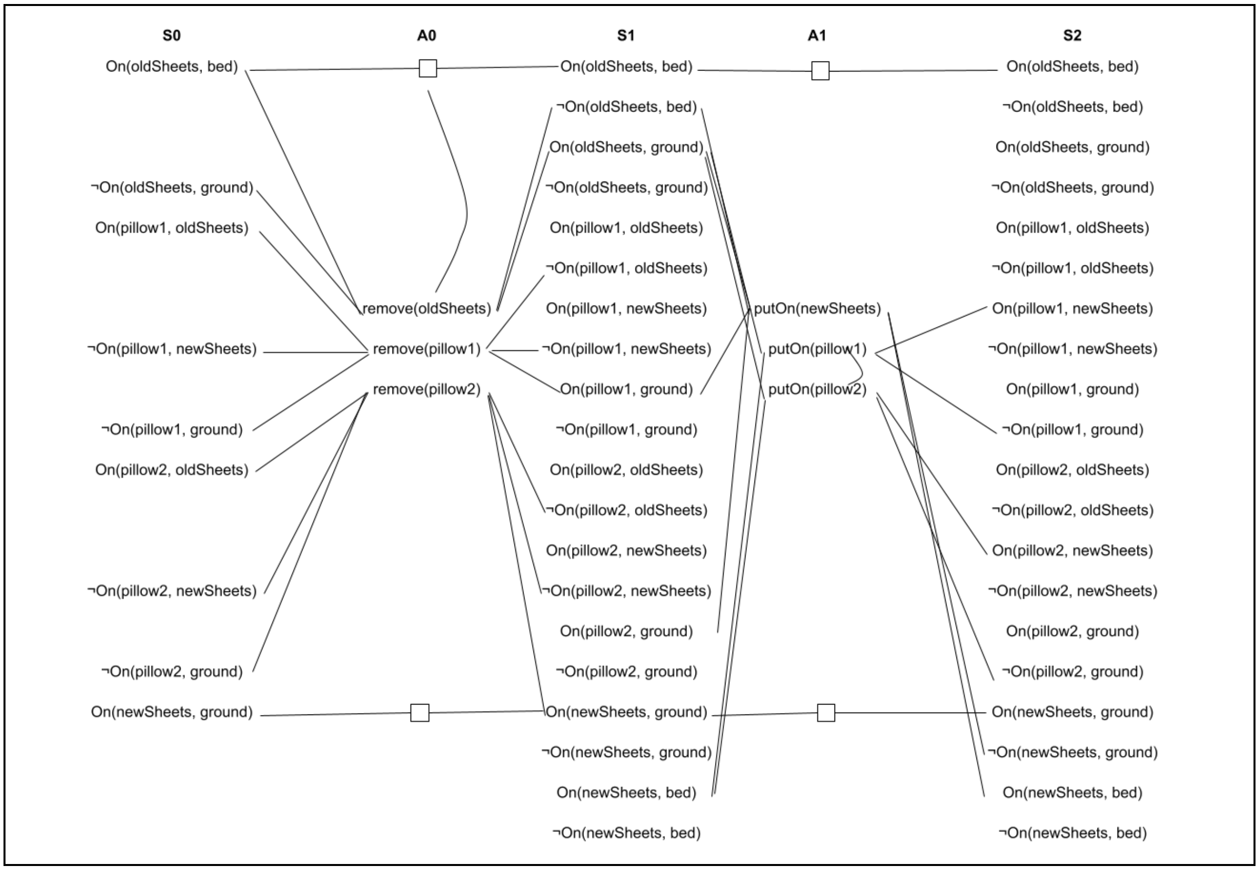
**Goal**( On(newSheets, bed) ∧ On(pillow1, newSheets) ∧ On(pillow2, newSheets)
∧ On(oldSheets, ground) )

**Action**( **remove**(x),
PRECOND: Pillow(x) ∨ Sheets(x)
EFFECT: On(x, ground) )

**Action**( **putOn**(x),
PRECOND: (Pillow(x) ∨ Sheets(x)) ∧ On(x, ground)
EFFECT: On(x, bed) )

**GraphPlan algorithm:**

- Start GraphPlan algorithm
- Start with S0, initial state shown in diagram
- 3 available action in A0: remove(oldSheets), remove(pillow1), remove(pillow2)
- After applying the actions A0, we get S1
- Goal not reached, so go to A1
- 3 available action in A1: putOn(newSheets), putOn(pillow1), putOn(pillow2)
- After applying the actions A1, we get S2
- We reached the goal at S2:
    - On(newSheets, bed) ∧ On(pillow1, newSheets) ∧ On(pillow2, newSheets) ∧ On(oldSheets, ground) are all present
- Run Extract-Solution




**5. Express the Yale Shooting Problem in PDDL and show your solution does not suffer from the frame problem.**

**Init**( Alive(Fred) ∧  ¬Loaded(gun) )

**Goal**( ¬Alive(Fred) ∧  ¬Loaded(gun) )

**Action**( Load(gun),
PRECOND: ¬Loaded(gun)
EFFECTS: Loaded(gun) )

**Action**( Shoot(Fred),
PRECOND: Loaded(gun)
EFFECTS: ¬Alive(Fred) ∧ ¬Loaded(gun) )

**Solution:** [Load(gun), Shoot(Fred)]

Our solution to the Yale shooting problem represented in PDDL does not suffer from the frame problem because it clearly defines what changes or stays the same as a result of an action. In our solution, the action Load(gun) has the effect of loading the gun, and the effect of the Shoot(Fred) action clearly defines that Fred is dead and the gun becomes unloaded. There is no confusion that the goal state could be reached by the action Load(gun) and Shoot(Fred).