Artificial Intelligence

Inference Engine Implementation

(rule based system using the forward chaining strategy)

Table of Contents

Part 1 – Requirement

Implement an inference engine for a rule-based system using a propositional representation language that operates according to the forward chaining strategy, using two data structures: a rule base and a working memory.

The inference engine should filter all working memory elements through the rule base according to the following three steps: match, select and act. The match procedure should try to find a rule from the rule base whose antecedents are all matched by facts from the working memory. The first discovered rule with such a characteristic should be taken, and its consequents should be added to the working memory. The inference engine mechanism should be re-invoked recursively until no rule produces a new assertion, or until proving a certain goal.

Implement a prototype of the inference engine, operating with the depth-first search algorithm, using an imperative language. Demonstrate its performance using the example rules, facts and working memory given below.

Rule base:

```
(RULE 1   (IF    ( f h ) & ( a c ))
          (THEN ( b g )))

(RULE 2   (IF    ( n s ))
          (THEN ( e m )))

(RULE 3   (IF    ( r t ))
          (THEN ( p q )))

(RULE 4   (IF    ( dj ) & ( e m ) & ( k i ))
          (THEN ( a c )))

(RULE 5   (IF    ( p q ))
          (THEN ( n s )))

(RULE 6   (IF    ( u v ))
          (THEN ( k i )))
```

Working memory:

$$WM = (\,(\ f\ h\ )\ (\ d\ j\ )(\ u\ v\ )\ (\ r\ t\ )\,)$$

Part 2 – Algorithm

```
package AI;
import java.util.*;

public class InferenceEngineWrapper {

	private static void print(String message)
	{
		System.out.println(message);
	}

	@SuppressWarnings("serial")
	public static void main(String[] args) {

		try {
			InferenceEngineWrapper infEngWrapper = new InferenceEngineWrapper();

			print("***Create the item list");
			final Item item1 = infEngWrapper.new Item(1,"f");
			final Item item2 = infEngWrapper.new Item(2,"h");
			final Item item3 = infEngWrapper.new Item(3,"a");
			final Item item4 = infEngWrapper.new Item(4,"c");
			final Item item5 = infEngWrapper.new Item(5,"b");
			final Item item6 = infEngWrapper.new Item(6,"g");
			final Item item7 = infEngWrapper.new Item(7,"n");
			final Item item8 = infEngWrapper.new Item(8,"s");
			final Item item9 = infEngWrapper.new Item(9,"e");
			final Item item10 = infEngWrapper.new Item(10,"m");
			final Item item11 = infEngWrapper.new Item(11,"r");
			final Item item12 = infEngWrapper.new Item(12,"t");
			final Item item13 = infEngWrapper.new Item(13,"d");
			final Item item14 = infEngWrapper.new Item(14,"j");
			final Item item15 = infEngWrapper.new Item(15,"k");
			final Item item16 = infEngWrapper.new Item(16,"i");
			final Item item17 = infEngWrapper.new Item(17,"p");
			final Item item18 = infEngWrapper.new Item(18,"q");
			final Item item19 = infEngWrapper.new Item(19,"u");
			final Item item20 = infEngWrapper.new Item(20,"v");
			//final Item item21 = infEngWrapper.new Item(21,"z");

			print("");
			print("***Create the fact list");
			final Fact fact1 = new Fact(1, new
				ArrayList<Item>(){{add(item1);add(item2);/*add(item21);*/}}); //f
			h
			final Fact fact2 = new Fact(2, new
				ArrayList<Item>(){{add(item3);add(item4);}}); // a c
			final Fact fact3 = new Fact(3, new
				ArrayList<Item>(){{add(item7);add(item8);}}); // n s
			final Fact fact4 = new Fact(4, new
				ArrayList<Item>(){{add(item11);add(item12);}}); // r t
			final Fact fact5 = new Fact(5, new
				ArrayList<Item>(){{add(item13);add(item14);}}); // d j
			final Fact fact6 = new Fact(6, new
				ArrayList<Item>(){{add(item9);add(item10);}}); // e m
			final Fact fact7 = new Fact(7, new
				ArrayList<Item>(){{add(item15);add(item16);}}); // k i
			final Fact fact8 = new Fact(8, new
				ArrayList<Item>(){{add(item17);add(item18);}}); // p q
			final Fact fact9 = new Fact(9, new
				ArrayList<Item>(){{add(item19);add(item20);}}); // u v
```

```
                print("");
                print("***Create the consequent list");
                final Consequent consequent1 = infEngWrapper.new Consequent(100, new
                      ArrayList<Item>(){{add(item5);add(item6);}}); //b g
                final Consequent consequent2 = infEngWrapper.new Consequent(6, new
                      ArrayList<Item>(){{add(item9);add(item10);}}); //e m
                final Consequent consequent3 = infEngWrapper.new Consequent(8, new
                      ArrayList<Item>(){{add(item17);add(item18);}}); // p q
                final Consequent consequent4 = infEngWrapper.new Consequent(2, new
                      ArrayList<Item>(){{add(item3);add(item4);}}); //a c
                final Consequent consequent5 = infEngWrapper.new Consequent(3, new
                      ArrayList<Item>(){{add(item7);add(item8);}}); // n s
                final Consequent consequent6 = infEngWrapper.new Consequent(7, new
                      ArrayList<Item>(){{add(item15);add(item16);}}); // k i

                print("");
                print("***Create the goal");
                final Goal goal = infEngWrapper.new Goal(consequent1.getId(), new
                      ArrayList<Item>(){{add(item5);add(item6);}}); // b g
                print("");
                print("***Create the RuleBase data structure and add rules");
                RuleBase ruleBase = infEngWrapper.new RuleBase();
                ruleBase.addRule(infEngWrapper.new Rule(1, new
                      ArrayList<Fact>(){{add(fact1);add(fact2);}}, consequent1));
                ruleBase.addRule(infEngWrapper.new Rule(2, new
                      ArrayList<Fact>(){{add(fact3);}}, consequent2));
                ruleBase.addRule(infEngWrapper.new Rule(3, new
                      ArrayList<Fact>(){{add(fact4);}}, consequent3));
                ruleBase.addRule(infEngWrapper.new Rule(4, new
                      ArrayList<Fact>(){{add(fact5);add(fact6);add(fact7);}},
                consequent4));
                ruleBase.addRule(infEngWrapper.new Rule(5, new
                      ArrayList<Fact>(){{add(fact8);}}, consequent5));
                ruleBase.addRule(infEngWrapper.new Rule(6, new
                      ArrayList<Fact>(){{add(fact9);}}, consequent6));

                print("");
                print("***Create the WorkingMemory and add facts");
                WorkingMemory workingMemory = infEngWrapper.new WorkingMemory();
                workingMemory.addFact(new Fact(1, new
                      ArrayList<Item>(){{add(item1);add(item2);/*add(item21);*/}})); //
                f h
                workingMemory.addFact(new Fact(5, new
                      ArrayList<Item>(){{add(item13);add(item14);}})); // d j
                workingMemory.addFact(new Fact(9, new
                      ArrayList<Item>(){{add(item19);add(item20);}})); // u v
                workingMemory.addFact(new Fact(4, new
                      ArrayList<Item>(){{add(item11);add(item12);}})); // r t

                print("");
                print("***Create InferenceEngine and start the inference process");
                InferenceEngine inferenceEngine =
                      infEngWrapper.new InferenceEngine(goal, ruleBase,workingMemory );
                inferenceEngine.execute();

          } catch (Exception e) {
                e.printStackTrace();
          }
    }
```

```
public class InferenceEngine {

	private RuleBase ruleBase;
	private WorkingMemory workingMemory;
	private Goal goal;
	private Goal initialGoal;

	public InferenceEngine(  Goal goal, RuleBase ruleBase,
	                         WorkingMemory workingMemory ) throws Exception{
		this.ruleBase = ruleBase;
		this.workingMemory = workingMemory;
		this.goal = goal;
		this.initialGoal = goal;

		if (this.ruleBase == null) {throw new Exception ("InferenceEngine -> The
		                                    ruleBase cannot be null" );}
		if (this.ruleBase == null) {throw new Exception ("InferenceEngine -> The
		                                    ruleBase cannot be null" );}
		if (this.goal == null){throw new Exception ("InferenceEngine -> The Goal
		                                cannot be null" );}
	}

	//Starts the engine
	public void execute()
	{
		try {

			print("***The 'infer' method receives the goal: "+
			goal.getLabels());
			infer(goal);

		} catch (Exception e) {
			e.printStackTrace();
		}
	}
```

```
//Infer does the match, select and act recursively
private void infer(Goal goal) throws Exception {

	print("");

	//Goal is in WM... inference proven
	if (this.workingMemory.containsFact(goal)){
		print("SUCCESS / THE GOAL IS IN THE WORKING MEMORY -> The
			inference could be proven with the goal id: " + goal.getId()
		+ " / " + goal.getLabels());
		return;
	}

	print("***MATCH (v1) goal hypothesis (i.e. rule with consequent):" +
	goal.getLabels() );
	int ruleId =  matchAGoalWithConsequent(goal);

	print("***SELECT a rule");
	Rule rule = select (ruleId);
	print("Rule selected: " + rule.id );

	print("***MATCH (v2) goal with antecedents for rule: " + rule.getId());
	List<Fact> antecedents = matchASubGoalWithAntecedents(rule);

	for (int i = 0; i< antecedents.size(); i++ )
	{
		Fact antecedent = antecedents.get(i);
		if (this.workingMemory.containsFact(antecedent)){
			print("Fact Already present in Working Memory -> Fact Key: "
			+ antecedent.getId() +
			" - Fact: " + antecedent.getLabels());
		}
		else{
			print ("New hypothesis to prove: Fact id " +
				antecedent.getId() + " - Label: " +
				antecedent.getLabels());

			infer(new Goal (antecedent.getId(), antecedent.getItems()));
		}
	}

	print ("***ACT - Attempt to add Fact to working memory");
	act(goal);

	//All rules have fired... inference proven
	if (this.workingMemory.containsFact(this.initialGoal)){
		print("");
		print("SUCCESS / ALL RULES HAVE BEEN FIRED -> The inference could
			be proven with the goal id: " + goal.getId() + " / " +
			goal.getLabels());
		return;
	}
}
```

```
	public int matchAGoalWithConsequent(Goal goal) throws Exception{
		int ruleId = this.ruleBase.getRuleId(goal);
		if (ruleId <0){
			throw new Exception ("There is no rule for goal id: " +
				goal.getId() );
		}
		return ruleId;
	}

	public List<Fact> matchASubGoalWithAntecedents(Rule rule) throws Exception{
		List<Fact> antecedents = rule.getAntecedents();
		if (rule.getAntecedents() ==null){
			throw new Exception ("There is no antecedant for rule id: " +
					rule.getId() );
		}
		return antecedents;
	}

	public Rule select(int ruleId) throws Exception{
		Rule rule = this.ruleBase.getRule(ruleId);
		return rule;
	}

	public void act(Fact fact) throws Exception{
		this.workingMemory.addFact(fact);
	}
}

public class WorkingMemory{
	private ArrayList<Fact> facts;

	public WorkingMemory (){
		facts = new ArrayList<Fact>();
	}

	public void addFact(Fact fact) throws Exception{
		if (this.facts.contains(fact.id)){
			throw new Exception ("WorkingMemory -> addFact - key: " + fact.id
					+" already contained in hashtable.");
		}

		facts.add(fact);
		print("WorkingMemory -> addFact - key: " + fact.id + " - Fact:" +
				fact.getLabels());
		display();
	}

	public Boolean containsFact(Fact fact){
		for (int i = 0; i< facts.size(); i++){
			if (Fact.areEqual(facts.get(i), fact)){
				return true;
			}
		}
		return false;
	}

	private void display(){
		StringBuilder sb = new StringBuilder();
		String factLabel = null;
		sb.append("Working Memory State: ( ");
		for (int i = 0; i< facts.size(); i++){
			factLabel = facts.get(i).getLabels();
			sb.append(factLabel);
		}
		print(sb.toString() + " )");
	}
}
```

```
public class RuleBase{

	private Hashtable<Integer, Rule> rules;

	public RuleBase ()
	{
		rules = new Hashtable<Integer, Rule>();
	}

	public void addRule(Rule rule ) throws Exception{
		if (this.rules.containsKey(rule.getId())){
			throw new Exception ("RuleBase -> addRules - key: " + rule.getId()
					+" already contained in hashtable.");
		}

		rules.put(rule.getId(), rule);
		print("RuleBase -> addRules - key: " + rule.getId());
	}

	public Rule getRule(int ruleId){
		return rules.get(ruleId);
	}

	public int getRuleId(Goal goal){
		for ( int i =1; i <= this.rules.size(); i++){
			Rule rule = this.rules.get(i);
			if (Fact.areEqual(rule.getConsequent(), goal)){
				return rule.getId();
			}
		}
		//should never get there...
		return -1;
	}
}

public class Rule{
	private Integer id;
	private List<Fact> antecedents;
	private Consequent consequent;

	public Rule (Integer id, List<Fact> antecedents, Consequent consequent){
		this.id = id;
		this.antecedents = antecedents;
		this.consequent = consequent;
	}

	public Integer getId() {return this.id;}
	public List<Fact> getAntecedents() {return this.antecedents;}
	public Consequent getConsequent() {return this.consequent;}
}
```

```
public static class Fact{
	private Integer id;
	private List<Item> Items;

	public static Boolean areEqual(Fact fact1, Fact fact2){
		List<Item> fact1Items = fact1.getItems();
		int fact1Size = fact1Items.size();
		//the list of items is different
		if (fact1Size != fact2.Items.size()){
			return false;
		}
		else{
			//check the items have the same id
			for (int k = 0; k<fact1Size;k++){
				//at least one item has a different id
				if (fact1Items.get(k).getId() !=
					fact2.getItems().get(k).getId()){
					return false;
				}
			}
		}
		//the item list is the same
		return true;
	}

	public Fact (Integer id, List<Item> Items){
		this(id, Items, false);
	}

	public Fact (Integer id, List<Item> Items, Boolean proven){
		this.id = id;
		this.Items = Items;
		display();
	}

	public Integer getId() {return this.id;}
	public List<Item> getItems() {return this.Items;}

	public String getLabels()
	{
		StringBuilder sb = new StringBuilder();
		sb.append("(");
		for (int i=0; i < this.Items.size() ;i++)
		{
			sb.append(" ");
			sb.append(Items.get(i).getLabel());
			sb.append(" ");
		}
		sb.append(") ");

		return sb.toString();
	}

	protected void display() {
		print("Fact - key: " + id +" - label: " + this.getLabels());
	}
}
```

```
	public class Consequent extends Fact {

		public Consequent (Integer id, List<Item> Items){
			super(id, Items, false);
		}

		protected void display() {
			print("Consequent - key: " + super.getId() +" - label: " +
				super.getLabels());
		}
	}

	public class Goal extends Consequent{

		public Goal (Integer id, List<Item> Items){
			super(id, Items);
		}

		protected void display() {
			print("Goal (or Sub Goal) - key: " + super.getId() +" - label: " +
				super.getLabels());
		}
	}

	public class Item{

		private Integer id;
		private String factLabel;

		public Item (Integer id, String factLabel){

			this.id = id;
			this.factLabel = factLabel;
		}

		public Integer getId() {return this.id;}
		public String getLabel(){return this.factLabel;}

	}


}
```

Part 3 – Detailed Output

```
***Create the item list

***Create the fact list
Fact - key: 1 - label: ( f  h )
Fact - key: 2 - label: ( a  c )
Fact - key: 3 - label: ( n  s )
Fact - key: 4 - label: ( r  t )
Fact - key: 5 - label: ( d  j )
Fact - key: 6 - label: ( e  m )
Fact - key: 7 - label: ( k  i )
Fact - key: 8 - label: ( p  q )
Fact - key: 9 - label: ( u  v )

***Create the consequent list
Consequent - key: 100 - label: ( b  g )
Consequent - key: 6 - label: ( e  m )
Consequent - key: 8 - label: ( p  q )
Consequent - key: 2 - label: ( a  c )
Consequent - key: 3 - label: ( n  s )
Consequent - key: 7 - label: ( k  i )

***Create the goal
Goal (or Sub Goal) - key: 100 - label: ( b  g )

***Create the RuleBase data structure and add rules
RuleBase -> addRules - key: 1
RuleBase -> addRules - key: 2
RuleBase -> addRules - key: 3
RuleBase -> addRules - key: 4
RuleBase -> addRules - key: 5
RuleBase -> addRules - key: 6

***Create the WorkingMemory and add facts
Fact - key: 1 - label: ( f  h )
WorkingMemory -> addFact - key: 1 - Fact:( f  h )
Working Memory State: ( ( f  h )  )
Fact - key: 5 - label: ( d  j )
WorkingMemory -> addFact - key: 5 - Fact:( d  j )
Working Memory State: ( ( f  h ) ( d  j )  )
Fact - key: 9 - label: ( u  v )
WorkingMemory -> addFact - key: 9 - Fact:( u  v )
Working Memory State: ( ( f  h ) ( d  j ) ( u  v )  )
Fact - key: 4 - label: ( r  t )
WorkingMemory -> addFact - key: 4 - Fact:( r  t )
Working Memory State: ( ( f  h ) ( d  j ) ( u  v ) ( r  t )  )

***Create InferenceEngine and start the inference process
***The 'infer' method receives the goal: ( b  g )

***MATCH (v1) goal hypothesis (i.e. rule with consequent):( b  g )
***SELECT a rule
Rule selected: 1
***MATCH (v2) goal with antecedents for rule: 1
Fact Already present in Working Memory -> Fact Key: 1 - Fact: ( f  h )
New hypothesis to prove: Fact id 2 - Label: ( a  c )
Goal (or Sub Goal) - key: 2 - label: ( a  c )

***MATCH (v1) goal hypothesis (i.e. rule with consequent):( a  c )
***SELECT a rule
Rule selected: 4
***MATCH (v2) goal with antecedents for rule: 4
Fact Already present in Working Memory -> Fact Key: 5 - Fact: ( d  j )
New hypothesis to prove: Fact id 6 - Label: ( e  m )
Goal (or Sub Goal) - key: 6 - label: ( e  m )
```

```
***MATCH (v1) goal hypothesis (i.e. rule with consequent):( e  m )
***SELECT a rule
Rule selected: 2
***MATCH (v2) goal with antecedents for rule: 2
New hypothesis to prove: Fact id 3 - Label: ( n  s )
Goal (or Sub Goal) - key: 3 - label: ( n  s )

***MATCH (v1) goal hypothesis (i.e. rule with consequent):( n  s )
***SELECT a rule
Rule selected: 5
***MATCH (v2) goal with antecedents for rule: 5
New hypothesis to prove: Fact id 8 - Label: ( p  q )
Goal (or Sub Goal) - key: 8 - label: ( p  q )

***MATCH (v1) goal hypothesis (i.e. rule with consequent):( p  q )
***SELECT a rule
Rule selected: 3
***MATCH (v2) goal with antecedents for rule: 3
Fact Already present in Working Memory -> Fact Key: 4 - Fact: ( r  t )
***ACT - Attempt to add Fact to working memory
WorkingMemory -> addFact - key: 8 - Fact:( p  q )
Working Memory State: ( ( f  h ) ( d  j ) ( u  v ) ( r  t ) ( p  q )  )
***ACT - Attempt to add Fact to working memory
WorkingMemory -> addFact - key: 3 - Fact:( n  s )
Working Memory State: ( ( f  h ) ( d  j ) ( u  v ) ( r  t ) ( p  q ) ( n  s )  )
***ACT - Attempt to add Fact to working memory
WorkingMemory -> addFact - key: 6 - Fact:( e  m )
Working Memory State: ( ( f  h ) ( d  j ) ( u  v ) ( r  t ) ( p  q ) ( n  s ) ( e  m )  )
New hypothesis to prove: Fact id 7 - Label: ( k  i )
Goal (or Sub Goal) - key: 7 - label: ( k  i )

***MATCH (v1) goal hypothesis (i.e. rule with consequent):( k  i )
***SELECT a rule
Rule selected: 6
***MATCH (v2) goal with antecedents for rule: 6
Fact Already present in Working Memory -> Fact Key: 9 - Fact: ( u  v )
***ACT - Attempt to add Fact to working memory
WorkingMemory -> addFact - key: 7 - Fact:( k  i )
Working Memory State: ( ( f  h ) ( d  j ) ( u  v ) ( r  t ) ( p  q ) ( n  s ) ( e  m ) ( k
i )  )
***ACT - Attempt to add Fact to working memory
WorkingMemory -> addFact - key: 2 - Fact:( a  c )
Working Memory State: ( ( f  h ) ( d  j ) ( u  v ) ( r  t ) ( p  q ) ( n  s ) ( e  m ) ( k
i ) ( a  c )  )
***ACT - Attempt to add Fact to working memory
WorkingMemory -> addFact - key: 100 - Fact:( b  g )
Working Memory State: ( ( f  h ) ( d  j ) ( u  v ) ( r  t ) ( p  q ) ( n  s ) ( e  m ) ( k
i ) ( a  c ) ( b  g )  )
```

SUCCESS / ALL RULES HAVE BEEN FIRED -> The inference could be proven with the goal id: 100 / (b g)