Introduction to Computer Programming for Artificial Intelligence

June 29, 2015
Chapter 1

Introduction

What is Artificial Intelligence? While there are many answers to this question, in this course we will define it as using computers to solve problems in an intelligent manner.

As for Computer Science, it has three fundamental areas: Knowledge Representation, Algorithms, and Communication. In this course we will focus on Knowledge Representation.

Note that computers refer to computer hardware. They can do only very simple instructions automatically, e.g., moving data in the computer memory from one location to another. In order to solve problems intelligently, we need a way to communicate or interact with the computers. Therefore we need a common language by which we can explain or give information to the computer. The knowledge representation is to study how to represent the knowledge about a problem in this language so that this knowledge can be used by the computer to solve the problem.

There are two types of languages that can be used to communicate with computers. They are algorithmic and declarative languages.

Algorithmic languages: are used to represent a sequence of very simple instructions that are used by a computer to solve a problem.

Declarative languages: are used to represent precisely what a problem is. This representation is used by a computer program, usually called an inference engine or solver to solve the problem automatically.

In either language a representation of problem is called a program. In declarative languages, a representation can also be called a knowledge base.

In this course, we will be using a particular declarative programming language called SPARC to solve problems (using computers) that seem to need human intelligence. SPARC is a member of a well studied class of declarative programming languages – Answer Set Programming (ASP). ASPIDE is an environment that we will use to write SPARC programs and run or execute the programs.
Chapter 2

Basics of SPARC

2.1 Syntax of a SPARC Program

A SPARC program must have three sections: sorts, predicates, and rules.

Example of a SPARC program:

```sparc
sorts
#person = {john, bill}.

predicates
%father (X, Y) denotes that X is the father of Y.
father(#person, #person).

rules
%John (X) is the father of Bill (Y).
father(john, bill).
```

2.1.1 The sorts section

The sorts section consists a sequence of sort definitions. A sort definition is of the following form:

```
sortName = {objectName1, objectName2, ...}.
```

where sortName can be any name starting with #, and objectName can be any name starting with lower case.

Example.

```
#person = {john, bill}.
```

2.1.2 The predicates section

The predicates section consists a sequence of predicate declarations. A predicate declaration is of the following form:

```
relationName(sortName, sortName, ...).
```


where \( \text{sortName} \) is defined as before, and \( \text{relationNames} \) can be any name starting with lower case letter.

Example.
\[
\text{father(#person, #person)}.
\]

### 2.1.3 The rule section

The rule section consists a set of rules. For now, we consider only a simple type of rule, called fact. A fact is of the form:
\[
\text{relationName(object1, object2, ...}).
\]

A predicate declaration is of the following form:
\[
\text{relationName(sortName, sortName, ...}).
\]
where \( \text{sortName} \) is defined as before, and \( \text{relationNames} \) can be any name starting with lower case letter.

Example.
\[
\text{father(john, bill)}.
\]

### 2.1.4 Comments

To make our program easy to read, you are encouraged to use comments to explain the elements in your program. A comment should be start with `%.

Example.
\[
\% \text{John (X) is the father of Bill (Y).}
\]

Since the comments are just explanation of your program, the removal of them will not affect the correctness of your program. However, it affects how easily your program can be understood by you or other people.

Note. As you become more familiar with the materials, you are expected to revisit this section. Pay special attention on how we define the syntax of the SPARC program, an advanced but very useful skill. Note the distinction between sequence and set. The order in the definition of the syntax matters unless it is explicitly mentioned otherwise. Also pay attention to why we use different fonts in the text.

### 2.2 Problem Solving

An aim of this course is to solve problems that seem to need human intelligence. To achieve that we need

1. Problem description, and
2. A methodology to write SPARC program to solve the problem.
2.2. PROBLEM SOLVING

2.2.1 Problem Description

A problem description contains the background knowledge in the problem and questions we would like to answer.

Example (Family Problem).

1. Knowledge in the problem: There is a family. John is the father and Joan is the mother. The children are Jim, Bill, and Sam.

2. Questions: Is John the father of Bill?

2.2.2 Methodology

We have a two step methodology to write SPARC program to solve a problem (i.e., answer the questions in the problem):

1. Identification:

(a) the objects in the problem. E.g., John and Bill in the family problem.
(b) the relations in the problem, i.e., the relationship among objects. E.g., father is a relation between two persons.
(c) the knowledge in the problem. E.g., John is the father of Bill.

2. Translation/Coding:

(a) Group the objects into sorts (types or categories). For example, John and Bill forms a sort of “person”. We then represent the information above in the sorts section:

```
sorts
#person = {john, bill}.
```

(b) All relations will be declared in the predicates section.

```
predicates
father(#person, #person).
% father (X, Y) denotes that X is the father of Y.
```

(c) The knowledge will be represented as facts in the rules section.

```
rules
%John (X) is the father of Bill (Y).
father(john, bill).
```

Note. Remember to always add a period at the end of all sort definitions, predicate declarations, and rules. As you add on new sorts, predicates, rules, or any other information, be sure to check for errors. They will be displayed on the error console on the bottom left of the screen. The program will not run correctly when presented with errors.
2.3 Reasoning with Knowledge

The SPARC program (so far) for the family problem is

sorts
#person = {john, bill}.

predicates
%father (X, Y) denotes that X is the father of Y.
father(#person, #person).

rules
%John (X) is the father of Bill (Y).
father(john, bill).

Now let us look at how the knowledge in the program is used to answer the question of “is John the father of Bill.” When we are asked this question, we know it is about the relation father. So, we look for the father relation (with John and Bill in our mind too), and find the fact father(john, bill). Therefore, our answer is yes.

2.4 More Examples on Applying the Methodology

We discuss more examples to get familiar with the two steps of the methodology to write SPARC program (solve a problem).

Map Problem

Example (Map Problem). We have USA map (with four colors yellow, green, brown, purple)

Questions: is the color of Texas green? are New Mexico a neighbor of Oklahoma? do Texas and Oklahoma have the same color?
2.4. MORE EXAMPLES ON APPLYING THE METHODOLOGY

STEP 1 – Identification

Objects:

1. yellow, green, brown, purple;
2. newMexico, texas, oklahoma, ...

Relations:

1. color(X, Y) denotes that the color of state X is Y.
2. sameColor(X, Y) denotes that state X and state Y have the same color
3. neighbor(X, Y) denotes that the state X is a neighbor of Y.

Knowledge:

1. The color of Texas is green, ...... (each state has a color)
2. New Mexico is a neighbor of Texas, ...... (many such neighboring states)

STEP 2 – Translation

Objects:

1. yellow, green, brown, purple;
2. newMexico, texas, oklahoma, ...

SPARC code: we have two sorts: colors and states.

```sparc
sorts
#colors = {yellow, green, brown, purple}.
#states = {newMexico, texas, oklahoma}.
```

Relations:

1. color(X, Y) denotes that the color of state X is Y.
2. sameColor(X, Y) denotes that state X and state Y have the same color
3. neighbor(X, Y) denotes that the state X is a neighbor of Y.
CHAPTER 2. BASICS OF SPARC

SPARC code: we declare every relation – its name and which kind (sort) of objects are involved in this relation.

predicates
% color(X, Y) denotes that the color of state X is Y.
color(#states, #colors).
% sameColor(X, Y) denotes that state X and state Y have the same color
sameColor(#states, #states).
% neighbor(X, Y) denotes that the state X is a neighbor of Y.
neighbor(#states, #states).

Knowledge:
1. The color of Texas is green, ...... (each state has a color)
2. New Mexico is a neighbor of Texas, ...... (many such neighboring states)

SPARC code: we represent each piece of knowledge as facts.

rules
% The color of Texas is green
color(texas, green).
% New Mexico is a neighbor of Texas
neighbor(newMexico, texas).

Battle Ship Problem

Example (Battle Ship Problem). In the following grids, we have some marks with the following meaning:

****: air craft carrier
**: U-boat
**: submarine
There is a judge and a player. A part of the game is as follows. The player calls out a coordinate point. If that point is part of a ship, judge will say hit.

Questions: is (2,8) a hit? is (3,8) a hit?

STEP 1 – Identification

Objects:
1. aircraft carrier, u-boat, submarine
2. 0, 1, 2, ..., 10

Relations:
1. hit(X, Y) denotes that the coordinate (X, Y) is a hit.

Knowledge:
1. The coordinates (4,1), (5,1), (6,3), (7,3), (8,3), (9,3), (2,6), (2,7), (2,8) are hits.

STEP 2 – translation. We have the following SPARC program after translation.

```sparc
sorts
#numbers = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}.
#ships = {aircraftCarrier, uBoat, submarine}.
predicates
hit(#numbers, #numbers).
rules
% The coordinates (4,1), (5,1), (6,3), (7,3), (8,3),
% (9,3), (2,6), (2,7), (2,8) are hits
hit(4,1).
hit(5,1).
hit(6,3).
hit(7,3).
hit(8,3).
hit(9,3).
hit(2,6).
hit(2,7).
hit(2,8).
```
Chapter 3

Rules of SPARC

In this chapter we will discuss the rules which are very useful to represent knowledge that can not be represented as facts.

3.1 Write Rules for Knowledge

Example (Family Problem - dad)

1. Knowledge in the problem: There is a family. John is the father and Joan is the mother. The children are Jim, Bill, and Sam.

2. Questions: Is John the father of Bill? Is John the dad of Bill?

By applying the two step methodology we have

1. New relations: dad(X, Y): X is Y’s dad.
   SPARC code: add the following predicate declaration
   dad(#person, #person).

2. Knowledge: how do we add the knowledge?
   (a) Option 1: add the following knowledge derived from the problem description.
   dad(john, bill).
   In this approach, when we have more relations such as father(Justin, John) and father(brandon, joan), we have to “repeat” the relationship for dad: dad(Justin, John) and dad(brandon, joan)

   (b) Optional 2: we note that the father relation and dad is closely related. How would you describe this relationship between relations in English? One way may be “Dad is the same as father.” which can be refined as the following “For any two persons X and Y, X is the dad of Y if X is the father of Y.”
   dad(X, Y) :- father(X, Y).

Note about rules:
1. How to read the rule above? (same as the English description): :- read as “if”.

2. Variables are names starting with capital letter. It refers to some object. For example, X and Y in the rule above play a role of the word “somebody” (which refers to a person but we might not know who the person is).

3. We also say that the rule provides a definition of dad.

4. The left side of :- is called the head of the rule, and the right side is called the body of the rule. For example, the head of the rule above is \( \text{dad}(X, Y) \) and its body is \( \text{father}(X, Y) \).

### 3.2 Reasoning with Knowledge

Our SPARC program for the family problem so far is

**sorts**

\#person = \{john, joan, bill, jim, sam\}.

**predicates**

\%
father \((X, Y)\) denotes person \(X\) is father of \(Y\)
father(\#person, \#person).

\%
mother \((X, Y)\) denotes person \(X\) is mother of \(Y\)
mother(\#person, \#person).

**rules**

\%
John is the father of Bill, Jim and Sam
father(john, bill).
father(john, jim).
father(john, sam).

\%
Joan is the mother of Bill, Jim and Sam
mother(joan, bill).
mother(joan, jim).
mother(joan, sam).

\%
For any two persons \(X\) and \(Y\), \(X\) is the dad of \(Y\)
% if \(X\) is the father of \(Y\).

dad(X, Y) :- father(X, Y).

Let us now look at how we know that John is the dad of Bill. When we are asked is John the dad of Bill, we apply our knowledge (definition) on dad. To make the application, we replace \(X\) and \(Y\) in the head of the rule by john and bill respectively. At the same time, we also need to replace the other \(X\) and \(Y\) in the body of the rule in the same way. Now by the rule,
to know if John is the dad of Bill, we need to see if John is the father of Bill. Then from the our program we find the answer yes because of the fact father(john, bill). This type of reasoning is called top down reasoning.

Alternatively, we may start from the fact father(john, bill) in our program. Then by the rule defining dad, we conclude that John is dad of Bill. Here we need to replace the occurrences of X and Y by john and bill. This type of reasoning is called bottom up reasoning.

### 3.3 Write Rules for Knowledge with Disjunction in Condition

Now we further expand the family problem by a question: “is John a parent of Bill?”

To make our SPARC able to answer this question, we need

1. a new relation: parent(X, Y) which denotes person X is a parent of person Y, and we need

2. knowledge about parent: how parent is defined?

   %For any two persons X and Y, X is parent of Y
   % if X is the father of Y or X is the mother of Y.

   parent(X, Y) :- father(X, Y).
   parent(X, Y) :- mother(X, Y).

### 3.3.1 Reasoning with the Knowledge

Why do we get the answer that John is a parent of Bill? We know father(john, bill), and then we use the rule defining parent using father to get the positive answer. Or we start from the rule defining the parent. In fact, we can try the rule defining parent using mother, but quickly we realize that we can not conclude anything because the facts on mother does not contain information about john and bill. We then try the other definition of parent, and finally we get the “yes” answer.

Note. In this reasoning, we need to be careful to replace the variables by the right objects (john and bill).

How do we get the answer that Joan is a parent of Bill?

### 3.4 Default Negation and Classical Negation

You have learned negation before in your logic class. It is called classical negation.
Syntax of classical negation (\(-\)). As an example, \(-father(sam, bill)\) is read as Sam is not the father of Bill. How to represent that Sam is not the mother of Bill? It is by \(-mother(sam, bill)\).

Syntax of default negation \(\text{not}\). As an example, \(\text{not father}(sam, bill)\) is read as we don’t know or we don’t have a reason to believe joan is the father of bill.

We would like the computer to answer no when we ask if Sam is the father of Bill. What knowledge do we use, as human being, to get that answer? \(\text{Closed World Assumption}\) – I know everything about my world (e.g., family here), anything that I don’t know if false. In the example, we assume we know the complete father and mother information of this family. So,

\[
\text{% For any two person } X \text{ and } Y, \text{ } X \text{ is not the } \\
\text{father of } Y \text{ if we do not know that } X \text{ is the father of } Y \\
-father(X, Y) :- \text{not father}(X, Y).
\]

How to define \(-mother(X, Y)\) using the closed world assumption? How to define \(-parent(X, Y)\)?

3.4.1 Reasoning with Default Negation

Why does SPARC solver returns false when we ask \(\text{father}(joan, bill)\)? We examine every rule in the program, there is no way we can believe the information on \(\text{father}(joan, bill)\). So, by the rule defining \(-father(X, Y)\), we believe that \(-father(joan, bill)\).
3.5 Write Rules for Knowledge with Conjunction in Its Condition

(Please ignore section 3.5 in the previous handout.)

Now we further expand the problem by asking the question “Is Sam a brother of Bill”? By our two step methodology we need to do the following

1. New relation brother($X$, $Y$) denoting person $X$ is a brother of $Y$.
   Add the following predicate declaration to the SPARC program for family:
   
   brother(#person, #person).

2. New knowledge needed to define the new relation brother: “For any two person $X$ and $Y$, $X$ is a brother of $Y$ if $X$ and $Y$ have a same parent and $X$ is male.”

   To represent this knowledge, we apply the two step methodology here again. We need a new relation sameParent($X$, $Y$) which denotes that person $X$ and person $Y$ have a same parent. Now we are able to write the SPARC rule for brother:

   brother($X$, $Y$) :- sameParent($X$, $Y$), male($X$).

   We next have to declare sameParent in the predicates section:

   % add predicate declaration
   % sameParent(#person, #person).

   We have to identify the knowledge about sameParent and translate it to SPARC rule(s).

   %For any person $X$ and person $Y$, $X$ and $Y$ have a same parent
   % if there is a person $Z$ such that $Z$ is a parent of $X$
   % and $Z$ is a parent of $Y$.
   sameParent($X$, $Y$) :- parent($Z$, $X$), parent($Z$, $Y$).

   Note. The variable $Z$ does not appear in the head, but occur only in the body of the rule. It is understood as “there exists $Z$.”

   Comment. We missed the common sense knowledge of one can not be his own brother in our knowledge of brother. How will you revise your knowledge on brother? How will change the SPARC rule accordingly?

3.6 Iterative Refinement Methodology

To solve a problem (i.e., write SPARC program to solve a problem), we may need to apply the two step methodology iteratively to refine our identification and translation. For example, when to write SPARC rule(s) for brother, we simply introduce another relation sameParent to make our rule for brother easy to write. When then identify the knowledge needed for sameParent and write a SPARC rule for it.
Also note that in the translation, we may need to refine/rewrite the knowledge before we can write the SPARC rule(s) for it. For example, in Section ?? if an English description about parent is “Parent is the father and mother of a family,” it is not easy to write SPARC rules. We would like to make objects explicitly referred to in our English description so that writing SPARC rules is more straightforward. A refinement of the sentence above (aiming to make relations to involve objects explicitly):

For any two person X and Y, X is parent of Y if X is the father of Y or\(^1\) X is the mother of Y.

*On methodology of refining a piece of knowledge defining a relation:*

1. use relation name(s) explicitly and refer to objects involved in the relation explicitly, and

2. you can use both new relations (i.e., not defined yet) and/or old relations (i.e., defined already). In the case of using new relations, you have to define them in turn.

### 3.7 Recursive Rules

In this section we will introduce some interesting rules where the same relation name can occur in both the head and body of a rule. We will use the blocks world problem below as an example.

Problem Description (Blocks World Problem). We have some blocks on a table as shown in the table below

```
<table>
<thead>
<tr>
<th>b7</th>
</tr>
</thead>
<tbody>
<tr>
<td>b5</td>
</tr>
<tr>
<td>b3</td>
</tr>
<tr>
<td>b2</td>
</tr>
<tr>
<td>b1</td>
</tr>
<tr>
<td>b4</td>
</tr>
<tr>
<td>b6</td>
</tr>
</tbody>
</table>
```

Questions: is block b1 on the table? Which block is on b1? Is block b5 above b1?

STEP 1 – Identification.

1. Objects: b1, ..., b7, table

2. relations:

\(^1\)Note that the “and” in the original sentence really means “or.”
3.7. RECURSIVE RULES

(a) on(X, Y): denotes block X is on block Y
(b) above(X, Y): denotes block X is above Y

3. Knowledge:

(a) All knowledge about which block is on (touching) which other block or table. For example, b1 is on table, b4 is on b6, ... ...
(b) Knowledge about above. Do we have to list all above relationship one by one? Very tedious. We observe that the relation of above is dependent on the relation of on. We have the following definition (called recursive or inductive definition) of above:

For any block B1 and B2, B1 is above B2 if B1 is on B2.
For any block B1 and B2, B1 is above B2 if there is a block M such that B1 is on M and M is above B2.

STEP 2 – Translation. (The knowledge in English is in the comments of the SPARC program below.)

sorts
#blocks = {b1, b2, b3, b4, b5, b6, b7}.
#location = {b1, b2, b3, b4, b5, b6, b7, table}.
% we can write #location = #blocks + {table}.

predicates
% on(X, Y): denotes block X is on block Y
on(#blocks, #location).
% above(X, Y): denotes block X is above Y
above(#blocks, #locations).

rules
% knowledge about which block is on (touching) which other block.
on(b1, table).
on(b2, b1).
% .......
on(b6, table).
on(b4, b6).

% Knowledge about "above".

% For any block B1 and B2, B1 is above B2
% if B1 is on B2.
above(B1, B2) :- on(B1, B2).
% For any block B1 and B2, B1 is above B2
% if there is a block M such that B1 is on M
% and M is above B2.

above(B1, B2) :- on(B1, M), above(M, B2).

3.7.1 Reasoning with the Knowledge

Why do we get “yes” for the question above(b3, b1) in terms of your own knowledge? in terms of the knowledge in our SPARC program above?

We need to apply the rule of above several times. To see if above(b3, b1), by the rule defining above, we would like to see if there is a block B such that b5 is on B, i.e., on(b5, B) and B is above b1, i.e., above(B, b1). We do find a fact from the program that on(b5, b2). Then we let B be b2. Next we need to check if above(b2, b1). Applying the the first rule for above, we need to check on(b2, b1) from our program. We do find it as a fact. Therefore, our answer is yes.

3.7.2 More Knowledge for Blocks World

What knowledge do we need to answer questions we expect a negative answer, e.g., is b3 on b1? Is the closed world assumption proper here to answer these questions with negative answers?
Chapter 4

Solving Map Coloring Problems

In this chapter we will show how to solve some hard problems use SPARC. We first need to introduce the concept of answer sets of a SPARC program, and then will present how to write a SPARC program to solve the map coloring problem.

4.1 Answer Sets of A SPARC Program

We only give an intuitive understanding of what an answer set of a SPARC program means. Basically, an answer set of a SPARC program is all the relations among objects that can be derived from the program. Show an example of answer set(s) by using the family program.

4.2 Map Coloring Problem

Map coloring problem:

1. We have partial map of USA below
2. Question: can we use three colors green, yellow and brown to color the map such that no two neighboring states have the same color.

STEP 1 – Identification.

1. Objects:
   (a) tx, ok, ar, ks, mo
   (b) green, yellow, brown

2. relations:
   (a) color(X, Y) denotes that the color of state X is Y.
   (b) sameColor(X, Y) denotes that state X and state Y have the same color
   (c) neighbor(X, Y) denotes that the state X is a neighbor of Y.

3. Knowledge:
   (a) there is a color for each state.
   (b) the neighboring information such tx is a neighbor of ok, .......
   (c) no two neighboring states have the same color.

STEP 2 – Translation.

sorts
#states = {tx, ok, ar, ks, mo}.
#colors = {green, yellow, brown}.

predicates
% neighbor(X, Y) denotes state X is a neighbor of state Y.
neighbor(#states, #states).

% colorOf(S, C) denotes the color of state S is C.
colorOf(#states, #colors).

erules
% TX is a neighbor of OK and AR
neighbor(tx, ok).
neighbor(tx, ar).

% OK is a neighbor of TX, AR, KS, MO
neighbor(ok, tx).
neighbor(ok, ar).
......
% AR is a neighbor of OK and MO
.......% KS is a neighbor of OK and MO
.......% MO is a neighbor of KS, OK and AR
.......% There is a color for each state, i.e.,
% For any state S, it has a color of green, yellow or brown.
colorOf(S, green) | colorOf(S, yellow) | colorOf(S, brown).

% No two neighboring states have the same color, i.e.,
% For any two states S1 and S2, it is impossible that
% S1 and S2 are neighboring and
% S1 and S2 have the same color.
% We have to also make it explicit that S1 != S2
:- neighbor(S1, S2), colorOf(S1, C), colorOf(S2, C), S1 != S2.

4.3 Rule – Constraint/Denial

In the previous section we introduce a new type of rule: a rule without head. These rules are called \textit{constraints} or \textit{denials}.

:- neighbor(S1, S2), colorOf(S1, C), colorOf(S2, C), S1 != S2.

4.4 Auxiliary Relations

To express the knowledge that no two neighboring states have the same color, we can introduce a new relation (which is not necessary, but plays only an auxiliary role):
sameColor(S1, S2) which denotes state S1 and S2 have the same color.

Now, how to write the SPARC rules to express the knowledge? how to write the SPARC rule(s) for sameColor(S1, S2)?

4.5 Comments

1. Lets consider the knowledge that “no two neighboring states have the same color” again. Consider TX and OK. If we know the color of TX is green, what can we say
about the color of OK? Is it green? We are sure the color of OK is not green, according to the knowledge. Here is refinement/rewriting of the knowledge.

For any state $S_1$ and any state $S_2$ distinct from $S_1$, if the color of $S_1$ if C then the color of $S_2$ is not C.

The SPARC rule of the knowledge above is

$$\text{color}(S_2, C) \leftarrow \text{color}(S_1, C), \text{neighbor}(S_1, S_2), S_1 \neq S_2.$$ 

2. Symmetry. Here we list all neighboring states. However, observe that there is a symmetry on the neighbor relations: if state $S_1$ is a neighbor of state $S_2$, then $S_2$ is a neighbor of $S_1$. How will this observation will change your representation the knowledge of the neighboring states?
Chapter 5

Blocks World Problem and Planning

Problem description
  Two step methodology to solve the problem.
Chapter 6

Sudoku Problem

Problem description.
Two step methodology to solve the problem
Chapter 7

Reasoning with Defaults

Use the parent example as the note while penguin as an exercise?
Problem description.
Two step methodology to solve the problem.