Creating a Case-Base using CASL

1. Introduction

CASL is a language used for Case-Based Reasoning. The contents of a case-base are described in a file known as a case file, using the language CASL. The program Caspian uses this case file to create a case-base in the computer's memory, which can then be accessed and modified in order to solve problems, give a diagnosis etc. using Case-Based Reasoning techniques. When new cases are added to the case-base in the computer's memory, they are also appended to the end of the case file.

The process of Case-Based Reasoning (CBR) is to find a case that is similar to the current situation, modify the solution to fit the current situation and then to store the case in the case-base.

These processes can be carried out using the program Caspian.

The purpose of this section is to describe the language used in the case file for specifying cases (CASL), and to describe how Caspian uses this information to create the case-base.

You create a case base in CASL using your favourite editor and then load it into Caspian. Caspian checks that it is a legal CASL program as it loads it in (see the Caspian document for more details of error messages).

2. General Structure of a Case

In CASL, a case is similar to a record in a database. The basic unit is a field which may contain a string, a number, an enumeration symbol or a list. A list item may be any of the four basic types.

There are a number of differences between a CASL case and an ordinary database record:

1) There are two groups of fields in a case. The first group describes the situation and the nature of the problem. The second group describes the solution to the problem.

2) It is possible for a field to be omitted. This is only true for those fields which are not used for indexing (see below). Both field lists must contain at least one field. The problem section must use at least one field as an index.

3) Each case has a value known as the result. This can be the value SUCCESS, or the value FAILURE, or a value between -5 and 5.

4) A case has a name associated with it.

Enumeration fields defined in the problem section of a case may be used as indexes.

General Syntax of the case file

The case file consists of a number of blocks of code. The overall syntax is:

Introduction
Case Definition
Index Definition
[Modification Definition]
[Repair Rule Definition]
Case Instance {Case Instance}

The Introduction block contains introductory text which gets displayed when the program Caspian has finished checking the case file.

The Case Definition defines the types and the weights of the fields that may appear in a case. The information in the Case Definition is used for typechecking input cases while the weights are used to aid the case-matching process (described below).

The Index Definition defines the fields used as indexes when searching for a matching case. A case base should have at least one field used as an index. The type of an index field must be an enumerated type.

The Modification definition defines the modification rules. The purpose of the modification definition is twofold:

1) It provides a means of specifying that certain symbols or numbers are similar, for matching purposes.

2) It provides a means of specifying symbols as abstractions of others. This is useful for making the search more general or for defining generalised cases.

The Repair Rule definition contains the repair rules. The repair rules are used to modify the solution retrieved from the case-base, to make it more suitable for the current situation.

Both the Modification definition and the Repair rule definition may be omitted, but a completed system should contain both, if the system is to be a true CBR system.

The last set of blocks are the case instances. These are the cases that make up the case base. The case file must contain at least one case instance and will need to be seeded with many cases initially to be of any use.

A description of the case matching process is given in section 9.

3. The Introduction Section

The Introduction block defines introductory text which is displayed when Caspian is run. The purpose of the text is to help the user understand the purpose of the casebase. The syntax definition is:

Introduction = INTRODUCTION IS intro-text END ';'
intro-text = string { , string }

Within the strings themselves, **\n** is used to represent the newline character and **\t** is used to represent tab. This is not true of strings used elsewhere in the case file.

4. The Case Definition

This is the definition of the syntax of the case definition block.
Keywords are given in upper case.

```
Case-Definition = CASE DEFINITION IS field-definitions
                        SOLUTION DEFINITION IS field-definitions END ';'
field-definitions = fielddef {fielddef}
fielddef = FIELD fieldname TYPE IS fieldtype [WEIGHT IS positiveno] ';'
fieldtype = STRING | NUMBER | LIST | enumeration
enumeration = '(' symbol { , symbol } ')'
```

The symbols 'fieldname' and 'symbol' are identifiers. The symbol positiveno is a positive number.

The purpose of the case definition block is to define the fields contained in a case. It consists of a series of field definitions each of which defines the name, type and optionally the weight of each field.

The fields defined in the first field definition list are known as the problem fields. The fields defined in the section field definition list are known as the solution fields. In a case-instance a field may only appear in the section for which it was defined.

The weight value is used in matching cases. The larger the weight, the more important the field is. If the weight is not included in the field definition, the default value of one is assumed as the weight for the field.

The case definition is used to perform type-checking on the cases and user input to Caspian.

5. The Index Definition

The syntax of the index definition block is as follows:

```
Index-Definition = INDEX DEFINITION IS indexdefs END ';'
indexdefs = indexdef {indexdef}
indexdef = INDEX ON fieldname ';'
```

The 'fieldname' symbol is an identifier denoting the name of a field which must be an enumerated type.

The purpose of the index definition is to define which fields are to be used as indexes. This information is used by Caspian to generate an index structure to improve the search. Further information on indexes is given in the section on the modification rules. A field should be indexed only if the feature it represents is very important. Indexing on an appropriate field helps to ensure that the retrieved cases are in the right ball-park. There should be no need to use a weight on an indexed field.

Experience of using indexes shows that they are rather less useful than had been expected. They are intended to prune the matching process so that many cases are rejected early and a lot of work is saved. Unless you have a very large case base, then you might as well match on all fields. Don’t have more than one or two indexes.

6. The Modification Definition

The modification definition consists of a list of definitions known as the modification rules. The term modification rule is used slightly differently here in that the concern is with defining certain values to be similar, thus guiding the matching process, rather than performing modifications to the retrieved solution.

The syntax of the Modification Definition is as follows:

```
Modification-Definition = MODIFICATION DEFINITION IS moddefs END ';'
moddefs = moddef {moddef}
moddef = enummodrule | numbermodrule
enummodrule = FIELD fieldname ABSTRACTION symbol IS enumeration ';'
numbermodrule = FIELD fieldname SIMILAR RANGE number TO number ';'
```

There are two types of modification rule:

1) Rules on fields which are enumerated types. These have a dual function. First, the symbol after the keyword ABSTRACTION is considered to be an abstraction of the symbols in the following enumeration list. This serves two purposes:

a) When matching cases, if the user enters an abstraction symbol, the case matcher will match with the abstraction symbol or any of the symbols in the following enumeration list.

b) In a similar fashion, when examining an enumeration value to activate a repair rule, using an abstraction will cause a match with the abstraction symbol or any of the symbols in the enumeration list.

Second, when cases are being matched using the weights (see case matching, below) and two enumeration values are being compared, if neither symbol is used elsewhere as an abstraction and both symbols are in the the same enumeration list within a modification rule, then the symbols are defined to be "similar".

The result of comparing two symbols and finding them to be similar will cause the case matcher to return a weight for that field equal to three-quarters of the field's full weight.

2) Rules on fields which are numbers. The purpose of these rules is to define ranges over which numbers are considered similar.

If two number fields have values which are both contained in a range defined by a modification rule, then the numbers are "similar" and this will cause the case matcher to return a weight value equal to three-quarters of the field's full weight.

7. The Repair Rule Definition

The syntax of the repair rule block is as follows:

```
Repair-Definition = REPAIR RULE DEFINITION IS repseq END ';'
repseq = reprule {reprule}
reprule = REPAIR RULE rulename IS comblist THEN replist END ';'
comblist = comblistitem {AND comblistitem}
comblistitem = WHEN IDENTIFIER IS repvalue
repvalue = symbol |
        number |
        RANGE number TO number |
        string |
        list |
        UNDEFINED
```



```
replist = replistitem {replistitem}
replistitem = EVALUATE IDENTIFIER TO expression ';' |
        CHANGE IDENTIFIER TO newvalue ';'
newvalue = symbol | string | list | UNDEFINED
expression = expression + expression |
        expression - expression      |
        expression * expression      |
        expression / expression      |
        expression ^ expression      |
        '(' expression ')'           |
        - expression                 |
        number                       |
        fieldname
```

When the retrieved case is to be repaired, two things happen. First, the original values in the problem section are replaced by new values which have been entered by the user. Thus the problem section now represents the current situation.

Then the repair rules are applied. These examine the problem fields and the solution fields of the retrieved case for certain combinations of values that would cause problems. If this is the case then the repair rule fires and changes are made to the solution part of the retrieved case.

When Caspian applies the repair rules, it cycles through each of them in order until no more repair rules fire. A rule is only allowed to fire once. Since it is possible for one set of changes to cause another to fire, it is possible to decompose the repair rules, so that one rule can tweak the changes made by a previous rule activation.

The occurrence of "^" in an arithmetical expression denotes the power operator.

When a fieldname occurs in an arithmetical expression and the field is not used in the case, the field used to store the result becomes undefined. Changing a field to UNDEFINED deletes the field from the solution part of the retrieved case.

8. Defining a Case

The syntax of a case instance is as follows:

```
Case-Instance = CASE INSTANCE casename IS fieldlist SOLUTION IS fieldlist
        RESULT IS resulttype ';' END ';'
fieldlist = field {field}
field = fieldname '=' value ';'
value = symbol | number | string | list
resulttype = SUCCESS | FAILURE | number

list = '[' listitems ']' | '[' ']'
listitems = listitem {',' listitem}
listitem = number | string | symbol | list
```

The case holds two lists of fields, the first represents the current situation or problem, the second represents the solution, diagnosis or whatever. The field also has a result value which is a number between -5 and 5. The value SUCCESS is equivalent to 5 and FAILURE is equivalent to -5. A case also has a name associated with it.

9. Using Caspian to retrieve a case

The case retrieval is carried out in two stages:

1) The index values, which are either taken from the user case or specified separately by the user, are used to perform an index search. This retrieves a subset of cases from the case-base which match all the index values exactly (except when abstraction symbols are specified as index values, see the section on modification rules).

2) Once this list of cases has been retrieved, the user can do one of two things:

a) Allow the program to automatically select a case, based on weight-matching. For each case in the subset, the case-matcher find a weight which is obtained by totalling the weights of all the fields that matched. Fields which do not match exactly but are defined to be similar by the modification rules return a value which is three-quarters of the field's normal weight. The case-matcher selects the case with the highest total weight.

b) The user can browse through the selected cases and select a case manually.

After the case has been selected, the repair rules (see the repair rule definition section) are carried out. The user then has the option of adding the repaired case to the casebase.

If the user decides to add the repaired case to the case base, the values for the name and result of the case are entered by the user; the case is then appended to the case file and added to the casebase in the memory.

10. Lexical Niceties

Identifiers are made up of letters, digits and underscore.

Identifiers always begin with a letter. Strings are an arbitrary sequence of characters enclosed in double quotes.

Identifiers and strings have a maximum length of 29 characters.

Numbers may or may not have a decimal point. If a number has a decimal point, it must have a digit before it. It may be syntactically incorrect to use a negative number where only a positive value is appropriate e.g. field weights. Numbers may not use scientific notation.

The occurrence of "^" in an arithmetical expression denotes the power operator.

CASL is case-sensitive.

Comments begin with the tilde "~" character and are terminated by the end of line (or end of file).