Journal of Mathematical Psychology 74 (2016) 46–57

Contents lists available at ScienceDirect

Journal of Mathematical Psychology

journal homepage: www.elsevier.com/locate/jmp

Syntax and operational semantics of a probabilistic programming language with scopes

Peter D. Bruza

Information Systems School, Queensland University of Technology (QUT), GPO Box 2434, Brisbane, Australia

HIGHLIGHTS

- Syntax and semantics of probabilistic programs (P-programs) which are motivated and adhere to first principle of Contextuality-by-Default.
- The issue of contextuality is related to generalized join operations in relational database theory.
- Operational semantics are provided in a denotational way using the well known database language SQL.

ARTICLE INFO

Article history:
Available online 14 July 2016

Keywords:
Probabilistic programming
Probabilistic modelling
Programming language semantics
Contextuality
Contextuality-by-Default

ABSTRACT

Dzhafarov and Kujala (2015) have introduced a contextual probability theory called Contextuality-by-Default (C-b-D) which is based on three principles. The first of these principles states that each random variable should be automatically labelled by all conditions under which it is recorded. The aim of this article is to relate this principle to block structured computer programming languages where variables are declared local to a construct called a "scope". Scopes are syntactic constructs which correspond to the notion of condition used by C-b-D. In this way a variable declared in two scopes can be safely overloaded meaning that they can have the same label but preserve two distinct identities without the need to label each variable in each condition as advocated by C-b-D. By means of examples, the notion of a probabilistic program, or P-program, is introduced which is based on scopes. The semantics of P-programs will be illustrated using the well known relational database language SQL which provides an efficient and understandable operational semantics. A core issue addressed is how to construct a single probabilistic model from the various interim probability distributions returned by each syntactic scope. For this purpose, a probabilistic variant of the natural join operator of relational algebra is used to "glue" together interim distributions into a single distribution. More generally, this article attempts to connect contextuality with probabilistic programming by means of relational database theory.

1. Introduction

Dzhafarov and Kujala (2015) have introduced a contextual probability theory called Contextuality-by-Default (C-b-D) which is based on three principles:

1. (Indexation by conditions): A random variable is identified (indexed, tagged) by all conditions under which its realizations are recorded.
2. (Unrelatedness): Two or more random variables recorded under mutually incompatible conditions are stochastically unrelated, i.e., they possess no joint distribution.
3. (Coupling) A set of pairwise stochastically unrelated random variables can be probabilistically coupled, i.e., imposed a joint distribution on; the choice of a coupling is generally non-unique.

It is curious with respect to the first principle that a similar line of thinking emerged in the field of computer programming languages in the nineteen sixties, particularly with the advent of a radical new language called ALGOL-60. Before ALGOL-60, programming languages such as FORTRAN featured variables that are always accessible from any part of the program. These variables were termed "global" variables because of this property. As programming languages evolved, global variables were seen as a cause of errors, or "bugs". For example, consider the pseudo-FORTRAN program depicted in Algorithm 1. It begins by declaring three variables over real numbers. Function A (demarcated by "Function definition") uses variable X as a parameter to compute the value of variable

E-mail address: p.bruza@qut.edu.au.

http://dx.doi.org/10.1016/j.jmp.2016.06.006
0022-2496/© 2016 Published by Elsevier Inc.

Y, i.e., $Y = A(X)$. The main program (demarcated by "Main program") starts with variable X which has been assigned the value x_1. Next the function A is invoked with this value. However, note that the line $X \leftarrow x_2$ in the function reflects an assignment of the value x_2 to variable X. Here the programmer has forgotten that the intended use of variable X is an input variable to function A. Instead the programmer is using X for some local calculations internal to the function. Observe how this assignment unwittingly erases the value of X that was supplied as input to the function when it is invoked. This error could flow into subsequent computations involving variable X which are used to calculate the value of the function returned via the variable Z.

In summary, the error arises because variable X is being used for functionally *different* purposes in each scope. Note how this error contravenes the first principle of C-b-D because variable X is being used in different conditions, and so two variables should be used, rather than one. A programmer could easily fix this problem by using two variables as advocated by C-b-D's first principle. (See the program fragment in Algorithm 2.) Note the use of two variables $X1$ and $X2$, where each is used for a unique purpose.

Algorithm 1 Example FORTRAN Program 1

```
REAL X, Y, Z
X = $x_1$         }  Main program
Y = A(X)          }                          ▷ Call function A
function A(X)     }
    :             }                          ▷ Initial calculations
    X = $x_2$     }                          ▷ Local value assignment
    :             }  Function                ▷ calculations involving Z and X
    A = Z         }  definition              ▷ return value
    return        }
end function      }
```

Algorithm 2 Example FORTRAN Program 2

```
REAL X1, X2, Y, Z
X1 = $x_1$        }  Main program
Y = A(X1)         }                          ▷ Call function A
function A(X1)    }
    :             }                          ▷ Initial calculations
    X2 = $x_2$    }                          ▷ Local value assignment
    :             }  Function                ▷ calculations involving Z and X2
    A = Z         }  definition              ▷ return value
    return        }
end function      }
```

In order to help counter programming errors like the one just described, so called block structured programming languages were developed. A block is sometimes referred to as a "lexical scope", which is a syntactically delineated fragment of a program. The term "scope" refers to the set of program constructs such as variable definitions that are valid within that particular syntactic fragment of program code. As a consequence, the same variable name can be used in two different scopes but preserve unique "identities", where the identity is defined by its particular functional use in a given scope. By way of illustration, consider the pseudo-program code depicted in Algorithm 3. The variable X is a local variable to both scopes 1 and 2 which are delineated syntactically via the use of the terms **begin** and **end**. Both scopes assign a value to variable Y which is defined globally and hence accessible to both scopes. In each case the variable X has a unique function within each scope, so the computations in the second scope may give a completely different value to X than in the first scope.

Algorithm 3 Example Program 3

```
Var: $Y$
begin                              }
    Var: $X$                       }
    calculations involving $X$     }  Scope 1
    $Y \leftarrow X$               }
end                                }
Print("Answer 1 ",Y)
begin                              }
    Var: $X$                       }
    calculations involving $X$     }  Scope 2
    $Y \leftarrow X$               }
end                                }
Print("Answer 2 ",Y)
```

The intuition we will develop in this article is that classical probability theory is in some ways similar to obsolete programming languages like FORTRAN where all variables are global. We contend that as a consequence modellers are unwittingly predisposed to the invalid assumption that a variable is "the same" across different conditions, which contravenes the first principle of C-b-D. This can lead to potential "bugs" in the development of probabilistic models.

For example, Fig. 1 depicts data taken from an experiment designed to determine whether decisions regarding the relevance of document summaries to a query scenario are subject to an order effect (Bruza & Chang, 2014). An order effect is said to occur when a decision made in a non-comparative context differs from the decision taken in a comparative context. The corresponding experiment comprises two conditions: In one condition the subject is first asked to rate whether a document summary is topically related to the query, followed by a rating of how well the subject understands the information in the summary. This condition is referred to as the "non-comparative" context for the decision on topicality because the decision regarding topicality is made first without reference to any other decisions. In the second condition, the order of the ratings is reversed, i.e., the understandability of the document summary is first rated followed by a rating on topical relatedness of the summary to the query. This second condition is referred to the "comparative" context for the decision on topicality because a decision on understandability is asked first and may influence the subsequent decision on topicality.

The presence of an order effect can in many cases be determined by a difference in marginal probabilities across the two conditions. Consider the contingency tables in Fig. 1(a) and (b). Fig. 1(a) represents the non-comparative condition in which topicality is rated first. In this condition, the marginal probability that the document is topically related to the query is summed across understandability:

$$P(T = y) = P(T = y, U = y) + P(T = y, U = n) \tag{1}$$
$$= 0.40 + 0.28 \tag{2}$$
$$= 0.68. \tag{3}$$

In the comparative condition topicality is rated after understandability (See Fig. 1(b)). Therefore in this condition, the marginal probability that the document is topically related to the query is calculated as follows:

$$P(T = y) = P(U = y, T = y) + P(T = n, T = y) \tag{4}$$
$$= 0.19 + 0.39 \tag{5}$$
$$= 0.58. \tag{6}$$

Note that the marginal probabilities are different (0.68 vs. 0.58). Therefore, variable T should *not* be assumed to be the same variable across the two experimental conditions. This fact is reflected by the pseudo-program code presented in Algorithm 4.