MEDWIN PUBLISHERS
Committed to Create Value for Researchers

Advances in Robotic Technology
ISSN :2997-6197

# Exception Handling in Logic Programming

**Kwon K***

Department of Computer Engineering, Dong-A University, South Korea

***Corresponding author:** Keehang Kwon, Department of Computer Engineering, Dong-A University, South Korea, Tel: 01085387785; Email: khkwon@dau.ac.kr

**Short Communication**

**Volume 1 Issue 1**

**Received Date:** October 09, 2023

**Published Date:** October 31, 2023

DOI: 10.23880/art-16000104

## Abstract

One problem on logic programming is to express exception handling. We argue that this problem can be solved by adopting linear logic and prioritized-choice disjunctive goal formulas (PCD) of the form $G0 \oplus *G1$ where G0, G1 are goals. These goals have the following intended semantics: sequentially choose the first true goal GI and execute GI where i (= 0 or 1), discarding the rest if any.

## Introduction

One problem on logic programming is to treat the extra-logical primitive in a high-level way. The progress of logic programming has enriched the theory of Horn clauses with higher-order programming, mutual exclusion [1], etc. Never the less exception handling could not be dealt with elegantly.

In this paper, we propose a purely logical solution to this problem. It involves the direct employment of linear logic and game semantics (or computability logic [2,3] to allow for goals with exception handling capability. A prioritizedchoice disjunctive (PCD) goal – is of the form $G0 \oplus *G1$ where G0, G1 are goals. We assume here G0 has higher priority. Executing this goal with respect to a program $D - ex(D, G0 \oplus *G1)$ – has the following intended semantics:
Sequentially choose the first successful one between

$$ex(D, G0), ex(D, G1).$$

An illustration of this aspect is provided by the following definition of the relation sort(X, Y) which holds if Y is a sorted list of X:

$$Sort(X, Y) :- heap\ sort(X, Y) \oplus * quick\ sort(X, Y)$$

The body of the definition above contains a PCD goal. As a particular example, solving the query sort ([3, 100, 40, 2], Y) would result in selecting and executing the first goal heap sort ([3, 100, 40, 2], Y). If the heap sort module is available in the program, then the given goal will succeed, producing solutions for Y. If the execution fails, the machine tries the plan B, i.e., the quicksort module.

The operator $\oplus *$ is, in fact, indispensable to logic programming, as it is a logic-programming equivalent of the if-then-else in imperative languages. To see this, consider the following example:

$$Max(X, Y, Z) :- (X \geq Y \wedge Z = X) \oplus *(Z = Y).$$

Of course, we can specify PCD goals using cut in Prolog, but it is well-known that cuts complicates the declarative meaning of the program [4].

As seen from the example above, PCD goals of the form $A \oplus *B$ can be used to specify a task A, together with the failure-handling routine B.

The exact meaning of $A \oplus *B$ can be explained by translating it to

$$A \oplus (\neg A \wedge B)$$

Advances in Robotic Technology
ISSN :2997-6197

Here, dealing with ¬A, we rely on closed world assumption.

This paper proposes Prolog⊕∗, an extension of Prolog with PCD operators in goal formulas. The remainder of this paper is structured as follows. We describe Prolog⊕∗ in the next section. In Section 3, we present some examples of Prolog⊕∗.

## The Language

The language is a version of Horn clauses with PCD goals. It is described by G- and D-formulas given by the syntax rules below:

$$G := A \mid G \wedge G \mid \exists x\, G \mid G \oplus * G$$

$$D := A \mid G \supset A \mid \forall x\, D \mid D \wedge D$$

In the rules above, A represents an atomic formula. A D-formula is called a Horn clause with PCD goals.

We will present a proof procedure for this language as a set of rules. These rules in fact depend on the top-level constructor in the expression, a property known as uniform provability [5-8]. Note that proof search alternates between two phases: the goal-reduction phase and the back chaining phase. In the goal-reduction phase (denoted by pv(D, G)), the machine tries to solve a goal G from a clause D by simplifying G. The rule (6)-(8) are related to this phase. If G becomes an atom, the machine switches to the back chaining mode. This is encoded in the rule (5). In the back chaining mode (denoted by bc(D1, D, A)), the machine tries to solve an atomic goal A by first reducing a Horn clause D1 to simpler forms (via rule (3) and (4)) and then back chaining on the resulting clause (via rule (1) and (2)).

### Definition

Let G be a goal and let D be a program. Then the notion of proving $hD, Gi - pv(D, G)$ – is defined as follows:

- $bc(A, D, A)\%\, this\ is\ a\ success.$
- $bc(G0 \supset A), D, A)\, if\ pv(D, G0).$
- $bc(D1 \wedge D2, D, A)\, if\ bc(D1, \mathrm{D}, \mathrm{A})\, or\ bc(D2, D, A).$
- $bc(\forall x\, D1, D, A)\, if\ bc([t/x]\mathrm{D}1, \mathrm{D}, A).$
- $\mathrm{pv}(D, A)\,\mathrm{if}\ \mathrm{bc}(D, D, A).$
- $\mathrm{pv}(D, \mathrm{G}\,0 \wedge \mathrm{G}\,1)\,\mathrm{if}\ PV(D, \mathrm{G}\,0,)\,\mathrm{and}\,\mathrm{pv}(D, G1).$
- $\mathrm{pv}(D, \exists x\,\mathrm{G}\,0)\,\mathrm{if}\ PV(D, [t/x]G0).$
- $\mathrm{pv}(D, G0 \oplus * G1)$ If select the first successful disjunction between $pv(D, G0)$ and $pv(D, G1)$. % this goal behaves as a goal with exception handling.

In the above rules, only the rule (8) is a novel feature. To be specific, this goal first attempts to prove G0. If it succeeds, then do nothing (and do not leave any choice point for G1). If it fails, then G1 is attempted.

Unlike Prolog, execution consists of two phases in our setting: proof phase and execution phase. The proof phase discussed above provides a winning strategy, say S, for the machine, which is a hard part. The execution phase is a relatively easy one. In the execution phase, the machine just follows the winning strategy is to complete the computation. Such a strategy has been studied before [5] and can be directly used here after preprocessing every occurrence of.

$$A \oplus * B\, to\, A \oplus (\neg A \wedge B).$$

### Examples

As an example, let us consider the following database which contains the today's flight information for major airlines such as Panam and Delta airlines.
% panam (source, destination, dp time, are time)
% delta (source, destination, dp time, are time) panam (Paris, nice, 9: 40, 10: 50)
Panam(nice,london,9:45,10:10)
delta (Paris, nice, 8 : 40, 09 : 35)
Delta (Paris, London, 9: 24, 09: 50)

### Consider a goal

$$\exists dt\, \exists at\ panam(paris, london, dt, at) \oplus * \exists dt\, \exists at\ delta(paris, london, \mathrm{dt}, \mathrm{at})$$

This goal expresses the task of finding whether the user has a flight in Panam to fly from Paris to London today. Since there is no Panam flight, the system now tries Delta. Since Delta has a flight, the system produces the departure and arrival time of the flight of the Delta airline.

## References

1. Kwon K, Kang D (2023) Choice Disjunctive Queries in Logic Programming. IEICE Transactions on Information and Systems 106(3): 333-336.

2. Japaridze G (2003) Introduction to computability logic. Annals of Pure and Applied Logic 123(1-3): 1-99.

3. Japaridze G (2008) Sequential operators in computability logic. Information and Computation 206(12): 1443-1475.

4. Bratko I (2001) Prolog programming for Artificial Intelligence. 3rd (Edn.), Addison-Wesley, pp: 678.

5. Hodas JS, Miller D (1994) Logic Programming in a

Advances in Robotic Technology
ISSN :2997-6197

Fragment of Intuitionistic Linear Logic. Information and Computation 110(2): 327-365.

6. Komendantskaya E, Komendantsky V (2005) On uniform proof-theoretical operational semantics for logic programming, pp: 1-16.

7. Miller D (1989) A logical analysis of modules in logic programming. Journal of Logic Programming 6(1-2): 79-108.

8. Miller D, Nadathur G, Pfenning F, Scedrov A (1991) Uniform proofs as a foundation for logic programming. Annals of Pure and Applied Logic 51(1-2): 125-157.