Adding Sequential Conjunctions to Prolog

Keehang Kwon and Sungwoo Hur
Dept. of Computer Engineering, DongA University
Busan 604-714, Korea
{khkwon, swhur}@dau.ac.kr

Abstract: Prolog have traditionally lacked devices for sequentiality. This limitation can be addressed by using a fragment of Japaridze’s computability logic. We propose Sprolog, an extension to Prolog with sequential-conjunctive goal formulas. Sprolog extends the LogicWeb by allowing goals of the form $G \cap G$ where $G$ is a goal. These goals allow us to specify sequential-conjunctive tasks.

keywords: Prolog, sequentiality, computability logic

1 Introduction

Sequentiality have played a key role in imperative languages. Despite its wide use, sequential tasks cannot be handled in logic programming based on classical logic or linear logic. For example, it is not possible to compute 5!, and then 6! in sequence in the theory of Prolog.

There have been many previous attempts to deal with sequentiality in Prolog, only by making the semantics much more complicated. One major limitation with these attempts is that the notion of truth is static, while the meaning of sequentiality requires a notion of action. This limitation can be overcome only by a new semantics for logical formulas, i.e., using Japaridze’s Computability Logic (CoL) [4]. CoL is a new semantic platform for reinterpreting logic as a theory of tasks. Formulas in CoL stand for instructions that can carry out some tasks.

This paper proposes Sprolog, an extension of Prolog with sequentiality in goal formulas. Traditionally, Prolog has the problem of expressing sequential tasks. The class of sequential goal formulas enables the programmer to express these sequential tasks.

To be specific, a sequential-conjunctive goal is of the form $G_1 \cap G_2$ where $G_1, G_2$ are goals. Executing this goal has the following intended semantics: execute both $G_1$ and $G_2$ sequentially. Both executions must succeed for executing $G_1 \cap G_2$ to succeed.

On the other hand, a parallel-conjunctive goal, denoted as a comma in Prolog, is of the form $G_1 \land G_2$ where $G_1, G_2$ are goals. Executing this goal has the following intended semantics: execute both $G_1$ and $G_2$ in parallel. Both executions must succeed for executing $G_1 \land G_2$ to succeed.

In this paper we present the syntax and semantics of this extended language, show some examples of its use and study the interactions among the newly added constructs.

The remainder of this paper is structured as follows. We describe Sprolog based on a first-order sequential Horn clauses in the next section. In Section 3, we present some examples of Sprolog. Section 4 concludes the paper.

2 The Language

The language we use is an expanded version of Horn clauses with sequential conjunctive goals. It is described by $G$- and $D$-formulas given by the syntax rules below:

$$G ::= A \mid G \land G \mid G \lor G \mid \exists x \, G \mid G \cap G$$

$$D ::= A \mid G \supset A \mid \forall x \, D \mid D \land D$$
In the rules above, $A$ represents an atomic formula. A $D$-formula is called a sequential Horn clause or an instruction.

In the transition system to be considered, $G$-formulas will function as queries and a set of $D$-formulas will constitute a set of available instructions. For this reason, we refer to a $G$-formula as a query, to a set of $D$-formulas as an instruction set. Our language is an extension to first-order Horn clauses with the main difference that new sequential constructs are added in $G$-formulas.

We will present an operational semantics for this language. These rules in fact depend on the top-level constructor in the expression and have the effect of producing a new program and a new available instruction set.

The rules for executing queries in our language are based on “goal-directness” in the sense that the next rule to be used depends on the top-level construct of the goal formula. This property is known as uniform provability[7]. This property is preserved by disallowing the disjunctive formulas and existential formulas in the instruction set. We further disallow the sequential conjunctive formulas as well for simplicity.

**Definition 2.1.** Let $G$ be a goal and let $P$ be a finite set of available instructions. Then the notion of executing $⟨P, G⟩$ – executing $G$ relative to $P$ – is defined as follows:

1. If $G$ is an atom and is identical to an instance of a program clause in $P$, then the current execution terminates with a success.

2. If $G$ is an atom and an instance of a program clause in $P$ is of the form $G_1 ⊃ G$, execute $⟨P, G_1⟩$. This execution must succeed for the current execution to succeed.

3. If $G$ is $G_1 \land G_2$, then execute both $⟨P, G_1⟩$ and $⟨P, G_2⟩$ in parallel. Both executions must succeed for the current execution to succeed.

4. If $G$ is $G_1 \lor G_2$, then execute both $⟨P, G_1⟩$ and $⟨P, G_2⟩$ in parallel. At least one of these two executions must succeed for the current execution to succeed.

5. If $G$ is $∃xG_1$, then execute $⟨P, [t/x]G_1⟩$ where $t$ is a term. This execution must succeed for the current execution to succeed.

6. If $G$ is $G_1 \cap G_2$, then execute both $⟨P, G_1⟩$ and $⟨P, G_2⟩$ in sequence. Both executions must succeed for the current execution to succeed.

In the above rules, the symbols $\cap$ provides sequential executions of instructions: they allow for the sequential conjunctive execution of the instructions. We conjecture the completeness of the above procedure following the techniques developed by Miller. The proof of this can be done using the induction on the degree of $P$ and $G$.

### 3 Examples

An example of the use of this construct is provided by the following “factorial” program which contains some basic factorial-handling instructions written in conventional Prolog. Here $+$ is used for the arithmetic addition, and $*$ for the multiplication.

\[
\text{fact}(0, 1), \% \text{ base case} \\
\text{fact} \left( X + 1, XY + Y \right) : - \text{fact}(X, Y).
\]

Our language in Section 2 permits sequential conjunction in goals. This allows for sequential interactions between the user and the system. An example of the use of this construct is provided by the page which does the following sequential tasks: read a number from the user, output the number, and then output its factorial number:

\[
\text{query1} : \\
\text{read}(N) \cap \\
\text{fact}(N, O) \cap
\]
For example, consider a goal \textit{query1}. Solving this goal has the effect of executing \textit{query1} with respect to the instructions in the factorial program.

Our language in Section 2 permits sequential conjunction of goal tasks. Another example of the use of this construct is provided by the page which does the following sequential tasks: read a number \(N\) from the user, output the number, computes \(N!\) and \((N+1)!\) in parallel and output the results:

\[
\begin{align*}
\text{query2} : & \quad \text{read}(N) \cap \\
& \quad \text{write}(N) \cap \\
& \quad \text{fact}(N, O1) \cap \text{fact}(N + 1, O2) \cap \\
& \quad \text{write}(\text{'fact :'}) \cap \\
& \quad \text{write}(O1) \cap \\
& \quad \text{write}(O2)
\end{align*}
\]

Again, consider a goal \textit{query2}. Solving this goal has the effect of executing \textit{query2} with respect to the instructions in the factorial program.

\section{Conclusion}

In this paper, we have considered an extension to Prolog with sequential Horn clauses. This extension allows goals of the form \(G \cap G\) where \(G\) is a goal. These goals are particularly useful for sequential executions of instructions. Our future interest is in a language which extends sequential hereditary Harrop formulas with these features[1, 2, 3].

\section{Acknowledgements}

This work was supported by Dong-A University Research Fund.

\section{References}


