A Formal Model and Interactive Visualization of miniKanren Search Semantics

Brysen Pfingsten Jason Hemann

Abstract

Mechanized executable semantics are a valuable tool for implementers and users alike to better understand their programs' behavior. The need is particulary acute in the context of novel logic programming languages with complex search strategies. Few such tools exist for logic-based programming languages however, and existing work only models languages' search behavior at a somewhat coarse-grained level. **In this work, we present the first small-step semantics for a logic language that models interleaving search at the level of individual goal execution.** The model is implemented in PLT Redex and allows users to step through the language's execution at a sub-interleave level.

We also present a visualization tool implemented through interactive JS in the browser with our miniKanren Redex model at its core. This visualizer allows users to input their own miniKanren programs as Racket source code; these programs are transpiled to the language of our model. At each step, users can easily see the evaluation context of their programs, trace goals both to and from the source code and the search tree visualization, and readily understand the history of the computation that led parts of the current state to arise. This not only enables better performance analysis and a unique, visual way of debugging but also allows novice miniKanren programmers to more readily understand the semantics and structure underlying the miniKanren search evolution.

Grammar

$$\begin{array}{lcl}
p & ::= & (\texttt{prog}\ \Gamma\ e) \\
\Gamma & ::= & ((r_!\ d\ g)\ \ldots) \\
d & ::= & (x_!\ \ldots) \\
s & ::= & () \mid (g\ \sigma) \mid (s \rightarrow s) \mid (s \leftarrow s) \\
 & \mid & ((\top\ \sigma)\ \texttt{+}\ s) \mid (s\ \times\ g) \\
 & \mid & (\texttt{proceed}\ ((r\ t\ \ldots)\ \sigma) \\
 & \mid & (\texttt{delay}\ s) \\
g & ::= & \top \mid (t\ \texttt{=?}\ t) \mid (r\ t\ldots) \\
 & \mid & (g \vee g) \mid (g \wedge g) \mid (\exists\ d\ g) \\
c & ::= & \textbf{natural} \\
x & ::= & \text{(variable-prefix x:)} \\
r & ::= & \text{(variable-prefix r:)} \\
t & ::= & c \mid \textbf{boolean} \mid \textbf{string} \mid \textbf{symbol} \\
 & \mid & x \mid \texttt{empty} \mid (t\ :\ t) \\
\sigma & ::= & (\texttt{state}\ sub\ c) \\
sub & ::= & ((c\ t)\ \ldots)
\end{array}$$

Figure 1. Abridged grammar of our language.

Interleaving

At the core of miniKanren search is the idea of the interleave. In order to ensure a more fair search, each relation call is delayed until it reaches the top of our evaluation context. Traditionally, this operation is done by flipping the tree such that the newly delayed relation call must wait for the other side of the tree to complete its computation or encounter another relation call to be invoked. Visually, however, this transformation is confusing as it requires the programmer to recalibrate themselves with the new shape of the tree. As such, we implement a *railway model* with directed disjunctions that preserves the shape of the tree while allowing for interleave behavior.

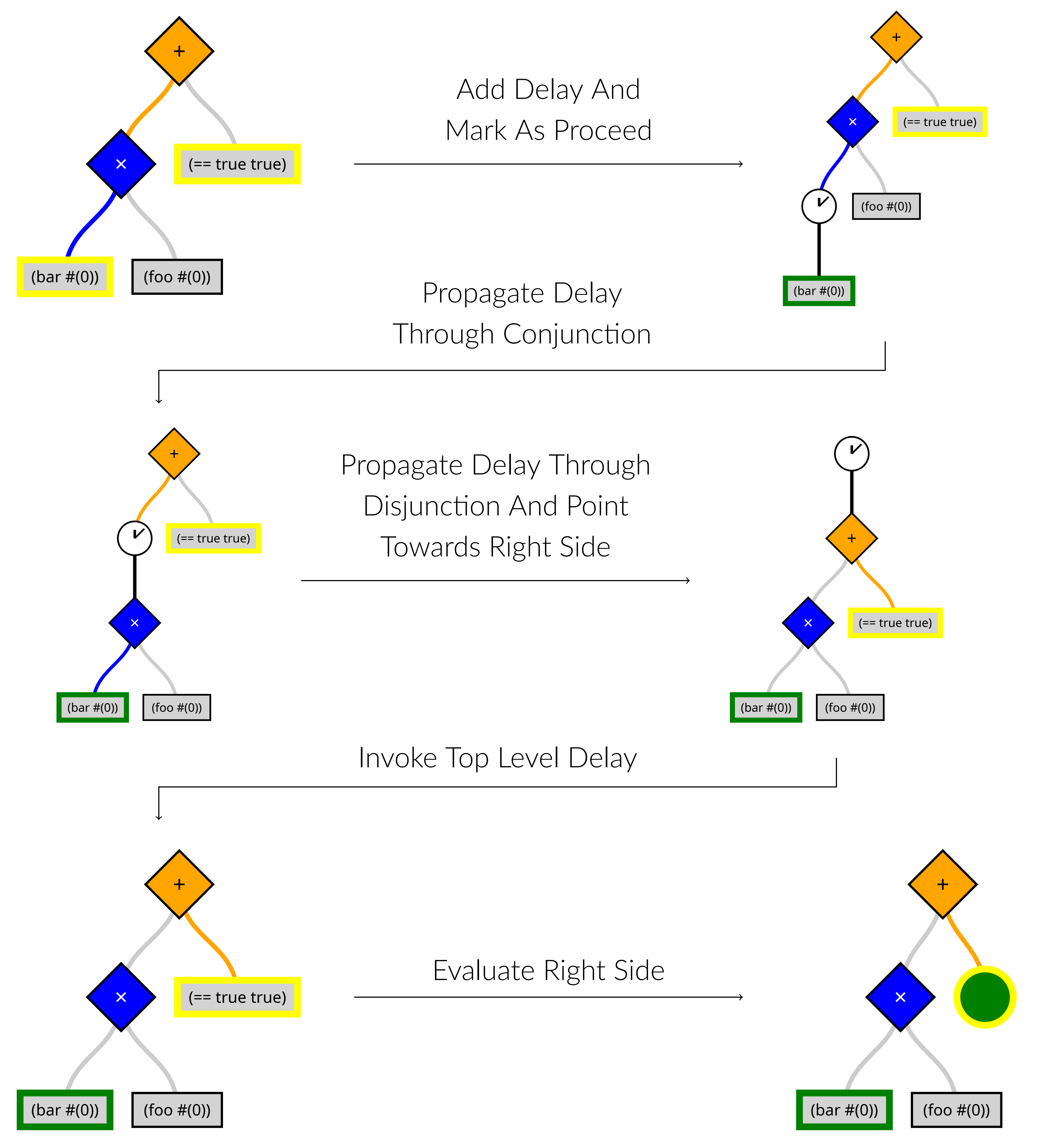
Add Delay And Mark As Proceed
Propagate Delay Through Conjunction
Propagate Delay Through Disjunction And Point Towards Right Side
Invoke Top Level Delay
Evaluate Right Side
(== true true)
(bar #(0))
(foo #(0))

Figure 2. Flow chart demonstrating interleave and railway behavior.

Reduction Steps

Distribute State Across Goals to Create Trees

$$\begin{aligned}
\texttt{Ex}[\![((g_1\ \vee\ g_2)\ \sigma)]\!] &\longrightarrow \texttt{Ex}[\![((g_1\ \sigma)\ \leftarrow\ (g_2\ \sigma))]\!] \\
\texttt{Ex}[\![((g_1\ \wedge\ g_2)\ \sigma)]\!] &\longrightarrow \texttt{Ex}[\![((g_1\ \sigma)\ \times\ g_2)]\!]
\end{aligned}$$

Railway Model

Note: We show here those reduction steps dealing with left-facing disjunctions and omit those dealing with right-facing disjunctions.

$$\begin{aligned}
\texttt{Ex}[\![((\texttt{delay}\ s_1)\ \leftarrow\ s_2)]\!] &\longrightarrow \texttt{Ex}[\![(\texttt{delay}\ (s_1\ \rightarrow\ s_2))]\!] \\
\texttt{Ex}[\![((\top\ \sigma)\ \leftarrow\ s)]\!] &\longrightarrow \texttt{Ex}[\![((\top\ \sigma)\ \texttt{+}\ s)]\!] \\
\texttt{Ex}[\![(()\ \leftarrow\ s)]\!] &\longrightarrow \texttt{Ex}[\![s]\!] \\
\texttt{Ex}[\![(((\top\ \sigma)\ \leftarrow\ s_1)\ \leftarrow\ s_2)]\!] &\longrightarrow \texttt{Ex}[\![((\top\ \sigma)\ \leftarrow\ (s_1\ \leftarrow\ s_2))]\!] \\
\texttt{Ex}[\![((s_1\ \rightarrow\ (\top\ \sigma))\ \leftarrow\ s_2)]\!] &\longrightarrow \texttt{Ex}[\![((s_1\ \rightarrow\ s_2)\ \rightarrow\ (\top\ \sigma))]\!]
\end{aligned}$$

Delays and Proceeds

$$\begin{aligned}
Ex[\![((r_1\ t\ \ldots)\ \sigma)]\!] &\longrightarrow Ex[\![(\texttt{delay}\ (\texttt{proceed}\ ((r_1\ t\ \ldots)\ \sigma)))]\!] \\
(\texttt{prog}\ \Gamma\ Ev[\![(\texttt{delay}\ s)]\!] &\longrightarrow (\texttt{prog}\ \Gamma\ Ev[\![s]\!])
\end{aligned}$$

Nullam vel erat at velit convallis laoreet

Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Phasellus libero enim, gravida sed erat sit amet, scelerisque congue diam. Fusce dapibus dui ut augue pulvinar iaculis.

First column	Second column	Third column	Fourth
Foo	13.37	384,394	α
Bar	2.17	1,392	β
Baz	3.14	83,742	δ
Qux	7.59	974	γ

Table 1. A table caption.

Donec quis posuere ligula. Nunc feugiat elit a mi malesuada consequat. Sed imperdiet augue ac nibh aliquet tristique. Aenean eu tortor vulputate, eleifend lorem in, dictum urna. Proin auctor ante in augue tincidunt tempor. Proin pellentesque vulputate odio, ac gravida nulla posuere efficitur. Aenean at velit vel dolor blandit molestie. Mauris laoreet commodo quam, non luctus nibh ullamcorper in. Class aptent taciti sociosqu ad litora