Rustxi - A transaction-based interactive prompt for Rust

Arvind Easwaran, Jason E. Aten, Do Nhat Minh

October 8, 2013

Abstract

This is the abstract

1 Introduction

The Rust programming language, led by Mozilla Research since 2010, aims to provide safety, concurrency and efficiency.

2 Architecture

The architecture of Rustxi is multi-process and transaction-based.

2.1 Multi-process

There are three OS-level processes which together constitute Rustxi.

Visor
: Visor is the long-running process which maintains input history, accepts user input from standard input and pipes input to Try for compilation and execution. On user exit, Visor kills both Cur and Try.

Cur
: Cur maintains the state of the interactive prompt (i.e. the compiled code for functions, structures and data defined at the prompt by the user) and respawns Try whenever Try fails.

Try
: Try compiles and executes user input. In case of success, Try becomes the new Cur by killing current Cur and spawning another Try.

2.2 Transaction-based

By default, memory is not shared between processes. In addition, virtual memory and Copy-on-Write support in the kernel ensures Rustxi's transactional requirements.

3 Call Graph

The call graph is a graph data structure whose nodes are the top-level bindings in the current session of Rustxi and whose edges are the calling (or usage) dependencies between the nodes. Through the help of call graph, Rustxi can decide which functions and types need recompilation whenever one node in the call graph is modified.

3.1 Implementation

The basic operations of call graphs include the following.

add
: adding a new node into the call graph

update
: updating the topology of the call graph

delete
: deleting a node from the call graph

3.1.1 Add

3.1.2 Update

3.1.3 Delete