A type system for calculating the maximum log memory used by transactional programs

Nguyen Ngoc Khai - K22, Software Engineering, Faculty of Information Technology, University of Engineering and Technology, Vietnam National University, Hanoi

Abstract
We developed a type system for estimating an upper bound of memory that multi-threaded and nested transactional programs may require for their transaction logs. In our previous works, we only estimated the maximum number of logs that can coexist by static type and effects systems. This work extends our previous language that allows one to specify also the size of logs in a transaction, and then develop a type system that can infer the memory bound required by the transaction logs.

Introduction
This work addresses the problem to determine the memory bound of a transactional program during the compile time to ensure that the program can run smoothly without memory overflow errors. To describe the problem more precisely, we use a core language based on [2]. Our language here focuses on features that allows the programmers to specify the size of transactional logs. We can formulate the problem as follows. Given the memory requirement of each transaction in the program, compute the maximal memory requirement for the whole program, and determine where in the execution of the program the maximal memory requirement is reached. The figure below represent a nested multi-threaded program.

Figure 1: A nested multi-threaded program

Abstract Language TM (Transactional Multi-threaded)

Syntax of the language TM

The global run-time environment is structured as a collection of local environments. Each local environment is a sequence of logs with their size. We formally define the local and global environments as follows.

Definition 1 (Local environment). A local environment $E$ is a finite sequence of log id's and their size $t_1, t_2, \ldots, t_n$. The environment with no element is called the empty environment and denoted by $\emptyset$.

Definition 2 (Global environment). A global environment $G$ is a sequence of thread id's and their local environments, $G = \{ [E_1, p_1 \mathbin{|} E_1], \ldots, [E_n, p_n \mathbin{|} E_n] \}$.

Dynamic semantics

The (global) run-time environment is structured as a collection of local environments. Each local environment is a sequence of logs with their size. We formally define the local and global environments as follows.

Concrete syntax

Function $\cdot$ is defined recursively as follows:

\begin{align*}
P & ::= \emptyset | [p \mid P] | P \cdot P \\
E & ::= \emptyset | [c_1 \cdot c_2] \\
\text{spawn}(s) & ::= \text{enroll}(s) \cdot \text{commit}
\end{align*}

Figure 2: TM syntax

Type system

The main purpose of our type system is to identify the maximum log memory that a TM program may require. The type of a term in our system is computed from what we call sequences of tagged numbers, which is an abstract representation of the term's transactional behavior w.r.t logs.

Refernces


Figure 3: Definition 3

Abstract Language TM (Transactional Multi-threaded)

Syntax of the language TM

The global run-time environment is structured as a collection of local environments. Each local environment is a sequence of logs with their size. We formally define the local and global environments as follows.

Definition 1 (Local environment). A local environment $E$ is a finite sequence of log id's and their size $t_1, t_2, \ldots, t_n$. The environment with no element is called the empty environment and denoted by $\emptyset$.

Definition 2 (Global environment). A global environment $G$ is a sequence of thread id's and their local environments, $G = \{ [E_1, p_1 \mathbin{|} E_1], \ldots, [E_n, p_n \mathbin{|} E_n] \}$.

Dynamic semantics

The (global) run-time environment is structured as a collection of local environments. Each local environment is a sequence of logs with their size. We formally define the local and global environments as follows.

Concrete syntax

Function $\cdot$ is defined recursively as follows:

\begin{align*}
P & ::= \emptyset | [p \mid P] | P \cdot P \\
E & ::= \emptyset | [c_1 \cdot c_2] \\
\text{spawn}(s) & ::= \text{enroll}(s) \cdot \text{commit}
\end{align*}

Figure 2: TM syntax

Type system

The main purpose of our type system is to identify the maximum log memory that a TM program may require. The type of a term in our system is computed from what we call sequences of tagged numbers, which is an abstract representation of the term's transactional behavior w.r.t logs.

Refernces


Figure 3: Definition 3

Abstract Language TM (Transactional Multi-threaded)

Syntax of the language TM

The global run-time environment is structured as a collection of local environments. Each local environment is a sequence of logs with their size. We formally define the local and global environments as follows.

Definition 1 (Local environment). A local environment $E$ is a finite sequence of log id's and their size $t_1, t_2, \ldots, t_n$. The environment with no element is called the empty environment and denoted by $\emptyset$.

Definition 2 (Global environment). A global environment $G$ is a sequence of thread id's and their local environments, $G = \{ [E_1, p_1 \mathbin{|} E_1], \ldots, [E_n, p_n \mathbin{|} E_n] \}$.

Dynamic semantics

The (global) run-time environment is structured as a collection of local environments. Each local environment is a sequence of logs with their size. We formally define the local and global environments as follows.

Concrete syntax

Function $\cdot$ is defined recursively as follows:

\begin{align*}
P & ::= \emptyset | [p \mid P] | P \cdot P \\
E & ::= \emptyset | [c_1 \cdot c_2] \\
\text{spawn}(s) & ::= \text{enroll}(s) \cdot \text{commit}
\end{align*}

Figure 2: TM syntax

Type system

The main purpose of our type system is to identify the maximum log memory that a TM program may require. The type of a term in our system is computed from what we call sequences of tagged numbers, which is an abstract representation of the term's transactional behavior w.r.t logs.