Multithreaded Pointer Analysis Based on Petri Net

Fei Liu, Bixin Li

School of Computer Science and Engineering, Southeast University, Nanjing, China
Key Lab of Computer Network and Information Integration (Southeast University), Ministry of Education
Email: {fei_liu, bx.li}@seu.edu.cn

Abstract

This paper gives a novel method of investigating flow-sensitive pointer analysis for multithreaded program based on petri net. The method mainly borrows causal dataflow analysis idea from Azadeh Farzan. Petri net is used to describe control flow structure of multithreaded program. And pointer points-to information is propagated along causal dependencies of events in the partial order execution of petri net. The problem of pointer analysis is reduced to the coverability problem on the petri net.

Keywords—pointer analysis; multithreaded program; flow-sensitive; petri net

I. Introduction

With development of multi-core technique and extensive use of program language supporting threads, multithreaded programs are becoming more and more common. However, many mature and traditional program analysis techniques mainly aim at sequential programs. Because of non-determination of multithreaded program semantics, analysis process of multithreaded program is more difficult, compared to that of sequential program. Interactions among multiple threads make it difficult to extend traditional program analysis techniques that are developed for sequential programs to multithreaded programs[5]. Research on analysis of multithreaded program is still immature.

Pointer analysis[4] is a fundamental static program analysis technique. With many years’ research, there exist many publications about pointer analysis of sequential programs[1, 3]. Pointer analysis of multithreaded programs makes slow progress, compared to that of sequential programs. But there were still some work on multithreaded pointer analysis[5]. As far as flow-sensitive pointer analysis for multithreaded program is concerned, the main difficulty is the potential interferences among concurrent threads. Two threads interfere when a thread assigns to the shared pointer variable that another thread accesses (i.e., dereference, read or assign). Therefore, the key problem for designing flow-sensitive multithreaded pointer analysis algorithms is how to characterize inter-thread interference. This paper presents a novel method of investigating flow-sensitive pointer analysis for multithreaded program based on petri net. The method borrows causal dataflow analysis idea from[2]. It captures control flow structure using 1-safe petri net and explores partially ordered execution of petri net. The points-to information of pointer is propagated along causal dependency of events. Pointer analysis problem of multithreaded program is converted to the marking coverability problem of petri net.

The rest of paper is organized as follows. Section II introduces some preliminaries containing 1-safe Petri net and Mazurkiewicz trace. Section III presents multithreaded pointer analysis based on petri net. Section IV shows a case study and Section V gives some related work analysis. Section VI concludes.

II. Preliminaries

A petri net is a triple N=(P,T,F), where every element in set P is called a place, tokens in every place represents resources that each place owns, each element in set T is called a transition, set P and T satisfying $P \cap T = \emptyset$, and the binary relation F is called flow relation of net N, satisfying $F \subseteq (P \times T) \cup (T \times P)$. Net N is called 1-safe petri net if each place contains at most one token at any time. A trace alphabet is a pair $(\Sigma, I)$ where $\Sigma$ is a finite set
of events, every element representing action and relation 
\( I \subseteq \Sigma \times \Sigma \) is called the independence relation, it being

irreflexive and symmetric. \( D = (\Sigma \times \Sigma) - I \) is called the 
dependence relation. A Mazurkiewicz trace is a behavior 
that describes a partially-ordered execution of events in \( \Sigma \).

III. Multithreaded Pointer Analysis Based on 
Petri Net

This section mainly contains three parts. In the first place, the definition of multithreaded language 
PML(Pointer-included simple Multithreaded Language) is introduced. In the second place, the construction of petri 
net model of PML program is illustrated. In the last 
place, multithreaded pointer analysis based on petri net 
is proposed. For an arbitrary given PML multithreaded 
program P, the aim of pointer analysis is to determine 
the points-to set of pointer variable at some program 
point. The outline of the method of investigating flow-
sensitive pointer analysis for multithreaded program based 
on petri net is as follows. Firstly, program P is transformed 
to P_transformed, which is semantically equivalent to 
program P. Secondly, petri net representation N of program 
P_transformed is constructed according to petri net repre-
sentation of basic statement. Thirdly, CCD framework in-
stance for multithreaded pointer analysis based on petri net 
is given. Fourthly, MOT(Meet Over All Traces solution) 
solution to reaching definition analysis of pointer variable 
is reduced to a coverability problem on the petri net 
which is solved by PEP(Programming Environment based 
on Petri Nets) tool based on partially ordered unfolding 
techniques.

A. Multithreaded language definition

The analyzed multithreaded program in this paper is 
based on simple multithreaded language PML(Pointer-
included simple Multithreaded Language). The syntax of 
PML is shown in figure 1. Pointer_first_level in figure 1 
is a set of pointer variable whose type is declared as int *. 
And Pointer_two_level is a set of pointer variable whose 
type is declared as int **.

B. Petri net model of PML program

For a given PML program P, this paper utilizes petri 
net to represent program control flow structure. In order to 
construct petri net model for PML program, there are two 
steps: (a)source code transformation of multithreaded pro-
gram P keeping semantically equivalent, let transformed 
program be P_transformed; (b)model control flow structure 
of P_transformed using petri net.

Fig. 1. Syntax of PML

1) Semantically equivalent source code transformation: 
For multithreaded program P, firstly, find all assignments 
statements in relation to dereference to two-level pointer 
variable, such as p=*t or *t=q statements. Secondly, do 
with the foregoing statements as follows: (using p=*t as 
an example to illustrate)

- scan source code of program P, for two-level pointer 
variable t, find all assignment statements where t is 
assigned. In the simple multithreaded language PML, 
pointer variable is one-level or two-level variable and 
PML currently doesn’t consider type cast. So the 
value of two-level pointer variable is only changed by 
assignment statements such as \( s = &t \), or \( s=t \). When 
the found assignments in the thread containing state-
ment p=*t, the last assignment statement to t is se-
lected. Otherwise, all assignment statements in other 
threads or some relevant assignment statements ac-
cording to synchronization structure of multithreaded 
program are selected.

- convert statement p=*t to statement section(*) ac-
cording to all selected assignment statements (t=&w, 
t=&v, \cdots; t=&q; let the number be N), as illustrated in 
figure 2. For statement *t=q, there exists correspond-
ing statement section(**) similar to(*).

- replace all assignment statements in relation to deref-
ereence to two-level pointer variable t with statement 
section(*) or (**) firstly, then delete all assignment 
statements referring to variable t. The remainder is 
program P_transformed, which is semantically equiva-
 lent to program P. The pointer variables in program 
P_transformed are all single-level pointers. The as-
signment statements of pointer variable only contain 
following type: p=&X and p=q.

2) Petri net representation of program control struc-
ture: This paper models control flow structure of 
P_transformed using petri net. The transition corresponds 
to program statements, and the place is used to represent 
 intra-thread control flow relation, inter-thread dependency 
relation and synchronization relation. Petri net representa-
tions of basic statements are illustrated in Figure 3.
C. Multithreaded pointer analysis based on petri net

The method of investigating flow-sensitive pointer analysis for multithreaded program based on petri net in this paper mainly borrows causal dataflow analysis idea from [2]. PML Program P is transformed to P_transformed, which is semantically equivalent to program P. Thus pointer analysis of program P is equivalent to that of program P_transformed. P_transformed doesn’t contain procedure call. The statements that could change pointer variable points-to information can only be basic statements such as p=q and p=&X. Under this circumstance, pointer analysis for program P_transformed, that is, determining points-to set of pointer variable p at program point n, is equivalent to determining which assignment statements referring to variable p in program P_transformed could reach program point n, namely reaching definition analysis of variable p. Then reaching definition analysis of pointer variable p is solved using a concrete framework instance based on CCD(Concurrent Causal Dataflow analysis) framework proposed in [2].

1) CCD framework instance for multithreaded pointer analysis based on petri net: The CCD framework instance used for multithreaded pointer analysis is a quintuple \((N, S, T, \mathcal{F}, \mathcal{D}, \mathcal{D}^*)\), including petri net model of P_transformed \(N=(P,T,F)\), property space \(S = (P(D), \subseteq, \cup, \emptyset)\), finite pointer points-to set \(\mathcal{D}\), powerset\(P(D)\), \(\emptyset\) representing initial pointer points-to set, \(\cup\) determining how we will combine points-to information along program control structure reaching the same control point in a program. For program P_transformed, let pointers={p | there exists assignment statement to pointer variable p in P_transformed}. Scanning program P_transformed and finding all pointer assignment statements similar to \(p = &X\), for example \(p =&X, q = &Y, p =&A, s = &C\), and denoting p-to=\{X, A, \cdots\}, q-to=\{Y, \cdots\}, s-to=\{C, \cdots\}, \cdots. Finally merging foregoing \((*)\)-to set as set pointers-to. Thus, \(\mathcal{D}\)=pointers \(\times\) pointers-to = \{(p,X) | p \in\) pointers \(\land\) X \(\in\) pointers-to\}. For any transition \(t\), set \(\mathcal{D}_t\) and \(\mathcal{D}_t^*\) are defined. \(\mathcal{D}_t^*\) represents points-to information relative to transition \(t\), and \(\mathcal{D}_t\) expresses points-to information that may modify when it executes. \(\mathcal{D}_t\) and \(\mathcal{D}_t^*\) satisfy condition \(\mathcal{D}_t \subseteq \mathcal{D}_t^* \subseteq \mathcal{D}\). And \(\mathcal{D} = \{\mathcal{D}_t | t \in T\}\) and \(\mathcal{D}^* = \{\mathcal{D}_t^* | t \in T\}\). For each transition \(t\) of petri net, function \(f_t\) is defined in order to represent how points-to information of pointer variable changes when \(t\) is executed.

2) MOT solution to multithreaded pointer analysis based on petri net: It can be verified that CCD framework instance for multithreaded pointer analysis based on petri net proposed in this paper is a distributive instance. The multithreaded pointer analysis problem can be solved by utilizing algorithm proposed in [2].

IV. Case Study

This section gives a simple PML program to illustrate the method of multithreaded program pointer analysis proposed in this paper.

- There are three threads in the example program, as given in figure 4. Some statements in three threads are synchronized by lock variable. The dealt problem is to determining the points-to information of pointer variable p in statement t14 and statement t32.
petri net. And the problem of computing MOT(t) solution is reduced to the problem of marking reachability on petri net.

From the experiment, we can acquire the conclusion that points-to information of pointer p holding before the execution of t14(*p=1) is \( d_1, d_2, d_3 \) (let \( d_1 = (p, x), d_2 = (p, y), d_3 = (p, z) \)), and that of pointer p holding before the execution of t32(a=*p) is \( d_1, d_3 \). Some related result are shown as figures 6,7.

V. Related Work Analysis

Pointer analysis of multithreaded programs makes slow progress, compared to that of sequential programs. There were relatively little work on multithreaded pointer analysis. The first flow-sensitive pointer analysis algorithm for multithreaded program is proposed in[5]. It uses a parallel flow graph to represent multithreaded program. And no edges between concurrent threads are supposed. Similar to traditional sequential program flow-sensitive pointer analysis, dataflow equation systems for four basic pointer assignments and concurrent structure are established and are solved by fixed-point iteration algorithm. It handles many constructors and has many advantages, but it only analyzes programs with structured concurrent constructs such as fork-join, and it ignores synchronization constructs such as lock. Flow-sensitive pointer analysis for multithreaded program introduced in this paper is based on petri net. Petri net commendably represents synchronization constructs and unstructured multithreaded program. And petri net based pointer analysis provides a novel perspective to consider pointer analysis for multithreaded program. Multithreaded pointer analysis can be reduced to coverability problem on the petri net which is solved by PEP tool based on partially ordered unfolding techniques.

VI. Conclusion

This paper proposes a novel method of investigating flow-sensitive pointer analysis for multithreaded program based on petri net. The method has been applied to several simple designed programs and the experiment results show its effectiveness. It is motivated by causal dataflow analysis idea from[2]. It models multithreaded pointer program using 1-safe petri net, introduces CCD instance for multithreaded pointer analysis based on petri net, and solves the instance using petri net reachability analysis.

References


