

# Quantitative Analysis of Language in Multi-threaded Programs

Han Chen

February 15, 2007

Protecting the private information (confident data, sensitive data) by computing the systems is an increasingly significant problem in the security area. The program includes high variables and low variables which present confident variables and public variables. There is no assurance that current systems can protect confidentiality. People have great interests in computing the leakage in the programs. Previous works are not satisfactory in the precision. People in our project firstly present several rules for calculate the leakage for program. However, the analysis of secureness in the multi-thread program still leaves an open challenge. We try to analyse the quantitative information leakage in a multi-threaded language based on a probabilistic scheduler. We take into account the interference between threads and also probabilistic property rooted from the scheduler. We demonstrate an algorithm that can transfer a multi-threaded program into sequential program, which enables the precise computation of the quantitative information leakage. Further, we give proof of transformation equality by using bisimilarity. Finally, we identify some important future works.

Multiple threads sharing the same memory space in a process may cause a security problem. The interference will change the amount and rate of the looping constructs when a program is run with multiple threads in parallel rather than a single thread sequentially. An example is:

If that is to be run single-threaded ( $\tau_1; \tau_2$ ), the attacker will gain all information about the initial value of  $h$  by observing what output  $l$  is. When we use multi-threads to run this program, the threads would follow a time-slicing scheduler to preempt time pieces to run the partial code  $\tau_1 || \tau_2$  in parallel. As a result, we can know the initial value of  $h$  from  $l$  only if  $\tau_1$  is run in the first instance.

In this article we introduce the methodology of quantitative analysis of information flows into the study of a multi-threaded language. This is a significant task, since a multi-threaded language would subject to different types of attacks: not only from explicit and implicit flows as in a single-threaded language, but also from *timing channels* and *probabilistic timing channels*. Further, the se-

---

$\tau_1: l=h; \tau_2: \text{while } (h < 3) \text{ } h++;$

---

Table 1:

curity of a multi-threaded language could also represent the same problems for any multi-threaded systems.

A thread in a Multi-threaded language is on a higher level of the program hierarchy than the program control structures (such as the `if` statement) and looping constructs. Compared to the analysis of `if` statement and looping construct, the analysis of a multi-threaded language involves a number of challenges such as:

1. the interference between different threads and the timing probability would lead to different results if the same program is to be run multi-threaded rather than single-threaded;
2. the probabilistic property of a scheduler may result in non-deterministic observations in each run of the program.

As the security of multi-threaded computation is always tightly connected with *timing*- and *probability*-sensitive system security, many past works on security of multi-threaded language intended to develop a security specification for concurrent systems. However, based on *qualitative* type systems and *non-interference*, their findings are concerned to be over strict, and yet unapplicable to almost any practical programs. The reason for that is there is no formal definition of *strictness*, or practicability. However, in fact, the question of how to decide if some security policy is secure enough can be answered by a criterion based on quantitative analysis that describes how secure or how unsecure a system is.

The main contribution of our work is that we provide the first approach to quantitatively analyse a multi-threaded language, and in a technically feasible way. Instead of trying to analyse the covert channels, we transform a multi-threaded program into an *equivalent* single-threaded program, and we prove the transformation process retains the bisimilarity between the two programs. After this, the quantitative analysis of the information leakage could be easily carried out.