The Optimum Leakage Principle for Analyzing Multi-threaded Programs

Han Chen and Pasquale Malacaria

School of Electronic Engineering and Computer Science,
Queen Mary University of London
hanchen@dcs.qmul.ac.uk,
pm@dcs.qmul.ac.uk

Abstract. Bellman's optimality principle is a method for solving problems where one needs to find best decisions one after another. The principle can be extended to assess the information leakage in multi-threaded programs, and is formalized into the optimum leakage principle hereby proposed in this paper. By modeling the state transitions in multi-threaded programs, the principle is combined with information theory to assess the leakage in multi-threaded programs, as the result of an optimal policy. This offers a new perspective to measure the information leakage and enables to track the leakage at run-time. Examples are given to demonstrate the analysis process. Finally, efficient implementation of this methodology is also briefly discussed.

1 Introduction and Background

The quantitative analysis of multi-threaded programs and concurrent systems is recognized as an important challenge. A multi-threaded program may have more vulnerabilities when compared to a single-threaded one: not only from explicit and implicit information flows but also from the *timing channels* and *probabilistic timing channels* [26]. It is also a difficult problem because the leakage in the same program may vary due to the additional uncertainty in scheduling. For example, consider the following program:

```
        l=h; | h=h & 0x07h;
```

Suppose the attacker observes the value of l in every single step of execution [22]. If the second statement is run at first then 3 bits of h is leaked, otherwise every bit of h is leaked. In this case the channel capacity is $size(h)$ bits, which is achieved by running the first statement at first.

An early quantitative assessment of leakage in multi-threaded programs has been using the mutual information between the input and the output [13]. Further proposals using algebraic or approximation methods to derive the channel capacity as an leakage upper-bound include [12] and [29]. Recently, Smith [28] proposed to use minimum entropy to evaluate the leakage. However, these approaches have remained preliminary; also, all of them are static, unable to track

K. Kurosawa (Ed.): ICITS 2009, LNCS 5973, pp. 177–193, 2010.

the actual amount of information leaked when a program is run. Until today, there is not yet a feasible solution for dynamically tracking the (quantitative) information leakage of multi-threaded programs at run-time.

Now, by combining the Bellman optimality principle with recent progress on the quantitative information flow, a method is proposed in this paper to provide a more sensible analysis of the leakage (or the confidentiality) of programs as well as to allow the tracking of leakage dynamically.

To apply this method, firstly the target multi-threaded program is modeled by a state-transition automata. We consider a probabilistic scheduler (the Lottery scheduler) which represents a general case for a range of modern schedulers. The execution of the program can be seen as a Markov process and the state-transition can be represented as a tree, where each possible state of the execution is a node in the tree with non-negative values on the edges. We assume the attacker can observe each single step of the execution. Then by applying the Bellman equation, the optimal or the pessimal leakage, which represents the leakage generated using an optimal policy or a worst policy in the program execution, can be derived. These can be derived either from the start of the program, or from any point of execution.

The method has several unique qualities:

- general: it is generally applicable to analyze multi-threaded programs run by a probabilistic scheduler, as well as similar probabilistic state-transition systems;
- sensible: the Bellman equation gives the accurate optimal leakage bounds;
- flexible: it is able to track the current leakage bound at any point of the execution tree;
- simple: a simplification algorithm can be applied prior to the Bellman algorithm, such that only the state-transitions with interference between high and low variables need to be considered.

In the longer term, this is aiming to build a policy which quantitatively restrict and control the leakage. By applying such bounds decision can be made either to accept or to reject a program, while dynamic measurement can reassure that an attacker can not acquire a substantial quantity of information. Also, in a broader sense, we believe the method can also be a template for tracking information leaks in state transition systems.

The paper is organized as follows: the next subsection reviews existing literature and the background. Section 2 provides a short tutorial of the Bellman equation and the optimality principle, and Section 3 presents the definition of the information leakage in multi-threaded programs. In Section 4 we show how multi-threaded programs are modeled and we develop the theorems and propositions of optimal leakage analysis. Then we present an analysis of two sample programs. Finally, we investigate the complexity in the process and propose a simplification algorithm to accelerate the solution process. Section 5 concludes the paper and identifies our future work.

1.1 Related Work

Learning theory, statistics and information flow analysis are naturally tied together by Shannon's information theory [27]. A few pioneers have brought Bayesian methods into the field of quantitative information flow, such as [6,5]. In this paper, besides the application of the Bellman's optimality principle to this field, we hope to provoke discussions on identifying more interesting connections between quantitative information flow and the learning theory.

The Bellman equation is regarded as one of the most fundamental theories in reinforcement learning. It gives an accurate model of gaining information in a state-transition system and underpin a vast extension of optimality algorithms in various specific directions.

The other end of the connection is the quantification of information leakage. The use of conditional mutual information in the context of information leakage has been pioneered by Gray [11]. However his definition is not aimed to measure leakage but to define it. Other pioneers on the use of information theory in the context of security are Dennings, McLean and Millen [9,8,20,21]. In recent years, a theoretical framework has been established based on Shannon's information theory to allow static, quantitative program analysis that provides an expectation of leakage in programs [15,16,17,23]. The theory is preliminarily extended to multi-threaded programs [13]. Recently an automatic method for information flow analysis is developed in [18]. Lowe's work [19] defined quantitative channel capacity in the context of CSP. Further, the channel capacity of a leakage channel under constraints was worked out by using Lagrange multiplier methodology and Karush–Kuhn–Tucker conditions, which was also applied in programs and anonymity protocols [22,12,14].

Besides, various other different, albeit inherently relevant definitions and methods have been proposed to quantify the information leakage. Among them, Di Pierro et al. used the norm of a transition matrix as a measure of probabilistic confinement [10]. Recently, Smith et al. proposed the use of minimum entropy, and argued that it can better describe the risk of leakage in [28]. Moreover, the idea of quantitative leakage in the context of protocols has been investigated in [3]. A discussion of the relationship between min entropy and Shannon entropy relevant to the context of this work can be found in [24].

In comparison, what our results represent is based on adopting the Bellman's optimality principle as the rule-of-thumb: it is not representing the very worst case which may happen with a very rare chance, but instead representing the expectation from an optimal strategy (or a most dangerous one) with which a multi-threaded program can be set to run.

2 Bellman's Optimality Equation and Optimality Principle

2.1 Bellman's Optimality Equation

In reinforcement learning, a Bellman equation refers to a recursion for expected rewards. The expected reward in a particular state s using a certain policy π follows the Bellman equation:

$$V^{\pi}(s) = R(s) + \gamma \sum_{s' \in S} P(s'|s, \pi(s))V^{\pi}(s')$$

where:

1. S is the set of states.
2. s, s' are states and $s, s' \in S$.
3. R is the one-period return function (e.g., a utility function).
4. π is a policy which maps from S to A which is the set of actions. A policy is hence a way to choose an action given a particular state of the system.
5. $P(s'|s, \pi(s))$ is a probability which describes the transition probability from the state s to s' with the action $a \in A$ following a policy π. In deterministic case, for each state and action, we specify a new state $S \times A \rightarrow S$ while in probabilistic case $S \times A \rightarrow P(S)$. For each state and action we specify a probability distribution $P(s|s, a)$ over next states.
6. V^{π} is the value function representing the expected objective value obtained by following a policy π from each state in S.
7. γ is a weight value, we can take $\gamma = 1$ for simplicity.

This equation describes the expected reward for taking the action prescribed by a given policy π. It is used to show how to use a model of the environment to convert immediate rewards into values.

Value functions partially order the policies, but at least one optimal policy π^* exists, and all optimal policies have the same value function V^*, which is solvable by Bellman optimality equations.

The equation for the optimal policy is referred to as the Bellman optimality equation:

$$V^*(s) = R(s) + \max_{a} \gamma \sum_{s' \in S} P(s'|s, a)V^*(s')$$

and

$$\pi^* = \arg\max_{a} \gamma \sum_{s' \in S} P(s'|s, a)V^*(s')$$

the optimality of π^* can be proved via negation: if a policy π selected an action a does not give out the maximal value of

$$\gamma \sum_{s' \in S} P(s'|s, a)V^*(s')$$

then there exists another policy π', which is the same as π everywhere except at state s. At state s, π' chooses the action a' which maximize the above expression. Thus, π can not be optimal and can not be chosen. Inversely, every optimal policy must choose actions to maximize the above one.

2.2 Bellman Optimality Principle

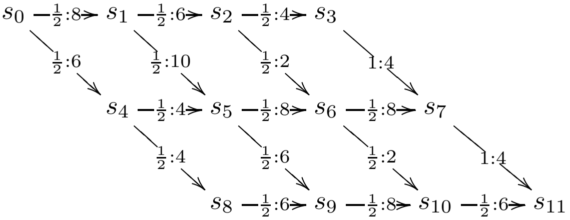
The Bellman optimality equation is central throughout the theory of Markov decision processes [25] (MDPs) and reflects the principle of optimality. The principle states:

> **"Regardless of the decision taken to enter a particular state in a particular stage, the remaining decision made for leaving that stage must constitute an optimal policy" [2].**

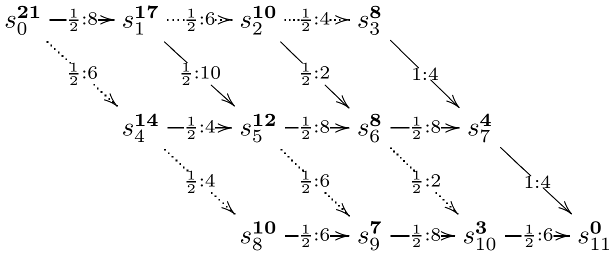
There is another way of saying that: an optimal policy always achieves optimal value for every start state, or, in each state the optimal policy will always select the same action as an optimal policy for which the state is the start state.

Therefore, it means if we entered the terminal state of an optimal policy we can trace it back. The equation reflects the principle: in the solution process, the Bellman equation is written forwards from the initial state but can be solved backwards from terminal state. The following is a small example to show how this principle is used.

Example of Bellman optimality principle. Consider the following probabilistic state transition system. In this transition system we assume s_0 is the initial state and s_{11} is the terminal state. We mark the probability and value of the transition in the path. Here the value of the transition is computed by the value function V as mentioned in the Bellman equation. We are going to use Bellman's optimality principle to find the policies for both maximal and minimal profit for this transition system.


s_0 —$\frac{1}{2}$:8→ s_1 —$\frac{1}{2}$:6→ s_2 —$\frac{1}{2}$:4→ s_3
$\frac{1}{2}$:6
$\frac{1}{2}$:10
$\frac{1}{2}$:2
1:4
s_4 —$\frac{1}{2}$:4→ s_5 —$\frac{1}{2}$:8→ s_6 —$\frac{1}{2}$:8→ s_7
$\frac{1}{2}$:4
$\frac{1}{2}$:6
$\frac{1}{2}$:2
1:4
s_8 —$\frac{1}{2}$:6→ s_9 —$\frac{1}{2}$:8→ s_{10} —$\frac{1}{2}$:6→ s_{11}

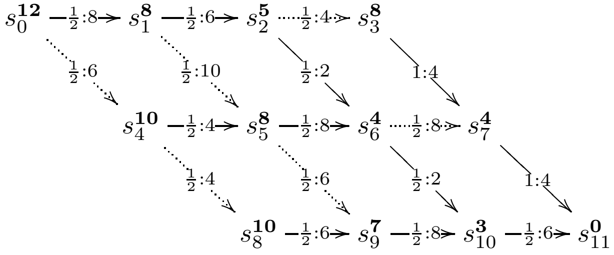
According to Bellman's optimality principle we start from the terminal state s_{11} and mark it as 0. We can reach this terminal node from nodes s_7 and s_{10}. If we are at node s_7 the value at transition is 4 and it is the only possibility transition from s_7, so we write s_7 of "$1 \times 4 = 4$" using "$P(s'|s,a)V^*(s')$ " where here we assume the factor $\gamma = 1$. Similarly the value of only transition from s_3 to s_7 is "$1 \times 4 = 4$" and we write s_3 of 4 as well. Likewise s_{10} is marked by "3" because the only transition from $s_{10} \rightarrow s_{11}$ has the value "$\frac{1}{2} \times 6 = 3$". Not all node only has one possibility, some states in the system have two possibilities, for example s_6 there are two transitions: one is to s_7 with the value "$\frac{1}{2} \times 8 + 4 = 8$" where in the equation "4" is the old value of s_7 and "$\frac{1}{2} \times 8$" comes from the transition; the other is to s_{10} with the value "$\frac{1}{2} \times 2 + 3 = 4$". Because $8 > 4$ we choose the transition to s_7 and write s_6 of 8. We leave the transition chosen as solid arrow and the transitions not chosen are marked with a dot arrow. Next we consider the previous node to s_6 which also has two possible transitions which are: one is to s_3 with the value "$\frac{1}{2} \times 4 + 4 = 6$"; the other is to s_6 with the value "$\frac{1}{2} \times 2 + 8 = 9$". At node s_2 we choose the transition to s_6 because $9 > 6$. We continue this procedure back to state s_0 with a value 21 which is the sought maximal profit.



The maximal profit is achieved by the path:

$$s_0 \rightarrow s_1 \rightarrow s_5 \rightarrow s_6 \rightarrow s_7 \rightarrow s_{11}$$

Using the same principle and oppositely, if we choose minimal value at each stage, when there are more than one choices, we can find the solution which results in a minimal profit of the transition system. The solution is 12 where the details are showing below:



and the selected path is

$$s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_6 \rightarrow s_{10} \rightarrow s_{11}$$

3 Information Leakage of Multi-threaded Programs

Information theory can be used to quantify the leakage in programs [15,16,17,13]. Generally speaking, the leakage of a system is the difference between the amount of original confidential information and the amount of remaining confidential information after observations. In information theory, this difference is formulated by mutual information:

$$I(h; l) = H(h) - H(h|l)$$

where

1. h is the high (confidential) information and l is the low (public) one.
2. $H(h)$ is the Shannon's entropy defined as $H(X) = -\sum_{x \in X} \mu(x) \log \mu(x)$ in which $X = \{x_1, \ldots, x_n\}$ with probabilities $\mu(x_1), \ldots, \mu(x_n)$.
3. $H(h|l)$ is the conditional entropy defined as $H(X|Y) = -\Sigma_{Y=y}\mu(Y = y)\Sigma_{X=x}\mu(X = x|Y = y)\text{log}(\mu(X = x|Y = y))$, where $\mu(X = x|Y = y)$ is the conditional probability of $X = x$ when $Y = y$.

Intuitively, mutual information $I(h;l)$ measures the information shared between h and l. In other words, it measures how much uncertainty of a variable is reduced by knowing the other. An extreme case is if h and l are independent, then $I(h;l) = 0$.

Further, conditional mutual information, a form of ternary interaction will be used to quantify *interference*. Conditional mutual information measures the correlation between two random variables conditioned on a third random variable, which is defined as:

$$I(h;l|Z) = H(h|Z) - H(h|l, Z) = H(l|Z) - H(l|h, Z)$$

Given the leakage formula defined from mutual information and conditional mutual information, we can compute the leakage of the high variable h coming from the observation of low variable l in a program.

Now, we consider the multi-threaded programs with probabilistic scheduling, as in [13]. We assume the attacker has the ability to observe the value of l in each single step; this represents the most conservative observational model in [22] and can be easily adapted to the other models such as the widely-used input-output model as in [16,23].

For example:

```
h=random(0, n); | l=h;
```

There are two threads and we assume each thread has probability $\frac{1}{2}$ to be chosen first and h is a k bit integer variable ($n = 2^k - 1$). The statement `h=random(0, n)` assigns a random number to h, while the other `l=h` leaks everything about h, which is k bits. Due to different scheduling there are two possible kinds of observations with equal probabilities of $\frac{1}{2}$, which will lead to either 0 bit or k bits of leakage. Then the expected leakage (as in [23]) would be

$$\frac{1}{2} \times k + \frac{1}{2} \times 0 = \frac{k}{2}$$

while the upper bound is k and lower bound is 0. For more complex multi-threaded programs, the computation of leakage could refer to the method in [13] and [22].

In comparison, we propose the optimal leakage principle below. We assume the attacker can make decision about the scheduling in the run time of multi-threaded programs and we give a methodology to evaluate the optimistic decision. The modeling of multi-threaded programs is described below, followed by theorems and propositions and then demonstrated by two program examples.

4 The Optimal Leakage Principle for Multi-threaded Programs

4.1 Modeling Multi-threaded Programs

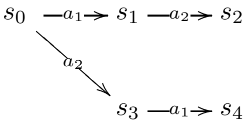
Here we model a multi-threaded program using a probabilistic state-space transition system:

$$\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{L} \rangle$$

where

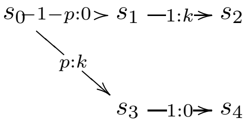
1. $\mathcal{S}$ is a set of possible states in the system; we note the initial state as s_0.
2. $\mathcal{A}$ is a set of actions which are statements in multi-threaded programs and we write them as a_i.
3. $\mathcal{P}$ is a set of probabilities associated to $\mathcal{S}$, and we note the probability from s_i to s_j as p_{ij}. We assume determinacy, i.e. given s_i and an action a there is at most one s_j s.t. $p_{ij} > 0$.
4. $\mathcal{L}$ is a set of values associated to $\mathcal{S}$, and we note the value from s_i to s_j as L_{ij}, where L_{ij} is the information leaked in the state transition s_i to s_j.

To this structure we can associate a state transition graph: we start from the initial state and select the statement from the program to reach a new state. We continue with this procedure until the last statement of the program. For example we first write the state transition of above example as



where a_1="l=random(0,n)" and a_2="l=h". and we also have the $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{L} \rangle$ where $\mathcal{S} = \{s_0, s_1, s_2, s_3, s_4\}$;
$\mathcal{A} = \{a_1, a_2\}$;
$\mathcal{P} = \{p_{01} = 1 - p, p_{12} = 1, p_{03} = p, p_{34} = 1\}$;
$\mathcal{L} = \{L_{01} = 0, L_{12} = k, L_{03} = k, L_{34} = 0\}$.

It is often easier to write the probabilities and values instead of actions in the transition system. Thus, the above state transition can be written as



A Note on Scheduler Sequence. There are many well-known schedulers that provide a deterministic execution order, for example Round Robin and Shortest Time First, however the execution sequences of multi-threaded programs in most of today's computing systems are non-deterministic.

In this paper we specifically analyze probabilistic schedulers, also known as the Lottery scheduler. Since a probabilistic scheduler represents a probabilistic policy of choosing threads, almost all other simple schedulers can be seen as specific examples of that. The only difference between different schedulers is in the choice of statements in the execution sequence due to the different scheduling policies.

We use the scheduler sequence to denote the execution order of a multi-threaded program. After choosing a statement in each small step in the run time, there is only one execution sequence chosen from all possible scheduling sequences following a certain probability distribution. We assume there are n threads and the scheduler sequence would be: $ijk...$ which means the i^{th} thread is chosen first, followed by the j^{th} thread, then the k^{th} thread, where $0 \leq i, j, k \leq n-1$.

Different outputs may come from different scheduler sequences, but one scheduler sequence can only produce one output. In the transition system, one path from the initial state to the terminal state represents a scheduler sequence.

We can now state an optimal leakage theorem.

4.2 Optimal Leakage Theorem

Theorem 1. *Optimal Leakage Theorem*
In a transition system, the upper bound of leakage L starting from a state s is given by the optimality equation:

$$L^*(s) = L(s) + \max_a \sum_{s'} P(s'|s,a) L^*(s')$$

and the corresponding scheduler for achieving this upper bound is

$$\mathcal{S}^* = arg\max_a \sum_{s'} P(s'|s,a) L^*(s')$$

where

1. *L is the leakage function, i.e.* $\max_j L_{s,s_j}$ *and*
2. *$P(s'|s,a)$ is the unique probability $p_{s,s'}$ given the action a*

Proof:
Proof by contradiction: if a scheduler sequence S^* selected a statement a which does not give out the maximal value of

$$\sum_{s'} P(s'|s,a) L^*(s')$$

then we can find another scheduler sequence S', which is the same as S^* everywhere except at state s. At state s, S' chooses the action a' which maximize the above expression.

Thus, S^* can not be optimal and can not be chosen. Inversely, every optimal policy must choose actions to maximize the above one.

The proof completes.

Similarly we can have the following proposition to get the lower bound.

Proposition 1. *Pessimal Leakage Theorem*
In a transition system, the lower bound of leakage L is given by the optimality equation:

$$L^*(s) = L(s) + \min_a \sum_{s'} P(s'|s,a)L^*(s')$$

where

$$\mathcal{S}^* = arg\min_a \sum_{s'} P(s'|s,a)L^*(s')$$

We can also easily prove this proposition via negation. The proof is omitted due to space limitation.

4.3 The Optimal Leakage Principle

Like the Bellman equation which reflects the optimal principle, Theorem 1 and Proposition 1 reflect the principle of information leakage under optimal exploit strategies. To build a transition system, we need to simulate all possible transitions for possible executions. As previously mentioned, in multi-threaded programs, different probabilistic scheduler may produce different outputs. Thus, there will be a set of terminal states, rather than one terminal state, in the transition system for a multi-threaded program. Suppose the set of terminal states is $\mathcal{T}$, each item in $\mathcal{T}$ is noted as t_i where $t_i \in \mathcal{S}$ as well.

To find the optimal and pessimal leakage, every element in $\mathcal{T}$ needs to be accessed, then traced back to the initial state. Formally, we have the proposition below:

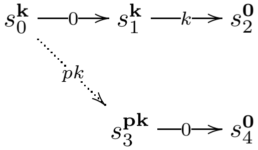
Proposition 2. *Optimal Leakage Principle*

1. *Firstly we start from the elements in $\mathcal{T}$. As these are terminal states, we mark them as* 0*.*
2. *Now trace back one level to look for previous nodes $s_i, s_j, ...$ adjacent to each element in $\mathcal{T}$. For each state, use Theorem 1 and Proposition 1 to compute the leakage at this stage and make the optimal or pessimal choice.*
3. *Repeat this process. At each stage compute the new value using Theorem 1 and Proposition 1 to make the optimal or pessimal choice. Trace backwards until arriving at the initial state, then we can achieve the optimal or pessimal leakage for the transition system.*
4. *Finally, the reverse path that starts from the initial state and constitutes of the chosen decisions above forms an optimal path.*

Example I. In the previous example:

```
l=rand(n); | l=h;
```

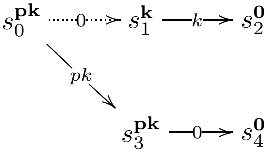
Here we use p to represent the probability of choosing "l=h" first and we assume $p < 1$. With the transition system previously established in Section 4.1, we use Proposition 2 to solve the leakage bounds recursively. There are two terminal states in this automata s_2 and s_4 so we mark them as 0. Then we look for the previous level and find s_1 and s_3. We start from s_1, the only reachable state is s_2 and the only transition has a value of leakage k with probability 1 so $L^{s_1}_{new} = k$; thus we mark s_1 to be k. For s_3, the only transition is $s_3 \to s_4$ which has a value of leakage of 0 with probability 1 so we mark s_3 as 0. We continue tracking back to s_0. s_0 has two possible choices: $s_0 \to s_1$ and $s_0 \to s_3$. $s_0 \to s_1$ has a leakage value of $k + 0 = k$ where k is the previous leakage coming from s_1 while $s_0 \to s_3$ has $0 + p \times k = pk$. Since $pk \le k$ we choose the transition $s_0 \to s_1$ and we mark s_0 as k. We mark the unchosen edge as dotted line.



The optimal leakage is achieved by the path

$$s_0 \to s_1 \to s_2$$

Also, we can easily get the pessimal path in the transition system



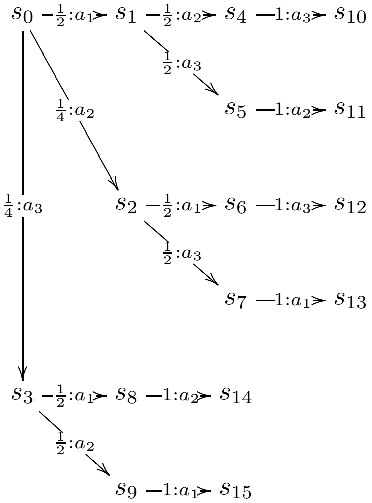
and the pessimal leakage pk is achieved by the path

$$s_0 \to s_3 \to s_4$$

Example II. Let us consider another example from [26]. This is a nested multi-threaded program. In the outer two threads, we use p as probability operator. There are two nested threads in one of them, reflected by the introduction of an additional probability operator q. Also, we assume that h is k bits long.

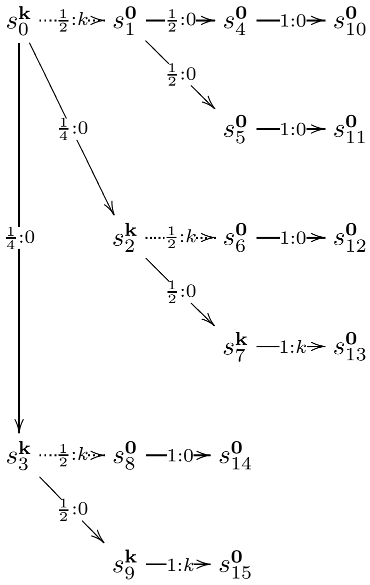
$$\text{l=h}|_p(\text{ l=0}|_q\text{l=1})$$

Here we assume $p = q = \frac{1}{2}$ which is a coin-flip choice operator. Using the modeling method in Section 4.1 we can got the transition system:



where a_1= "l=h"; a_2 = "l=0"; a_3= "l=1". We can see from the statements that a_1 leaks k bits while others do not leak. From this nested threads example, we also note that if the program has dynamic thread creation, then its transition system may similarly be constructed by reserving states and choices for the upcoming threads.

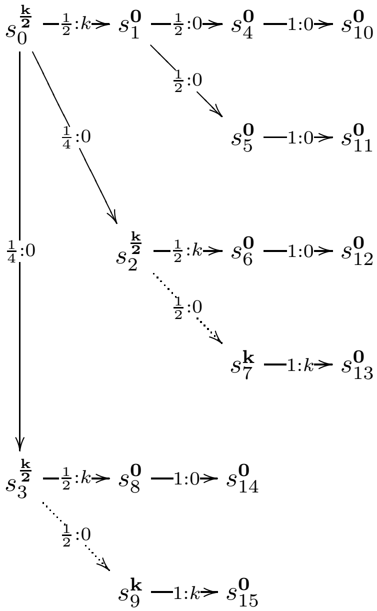
We are going to use Proposition 2 to solve the bounds of the leakage for this transition system. Firstly we consider the optimal leakage. At each stage we use Theorem 1 to achieve the optimal choice. We start from six possible terminal states $s_{10} \dots s_{15}$ and we mark them to be 0. We track back one level to find the states $s_4 \dots s_9$. In these states we first consider the node s_4, there is only one reachable state from s_4 which is s_{10} and the leakage in this transition is 0 with a probability of 1 so we mark s_4 to be $1 \times 0 = 0$. Also we can easily find that $s_{11}, s_{12}, s_{13}, s_{14}, s_{15}$ can only be reached by s_5, s_6, s_7, s_8, s_9. The leakage values for these transitions are $0, 0, k, 0, k$ with the probability 1, because in the transitions $s_7 \rightarrow s_{13}$ and $s_9 \rightarrow s_{15}$, a_1 has k bits leakage while in the other transitions, a_2 and a_3 has 0 leakage. So we mark s_5, s_6, s_7, s_8, s_9 with $0, 0, k, 0, k$ accordingly. We continue tracking back one level and find the states s_1, s_2 and s_3. s_1 can be reached by s_4 and s_5 where the leakage from the two transitions are both 0 so we mark s_1 as 0. Then we consider s_2, which can be reached by s_6 and s_7. The leakage in transition $s_2 \rightarrow s_6$ is k with probability $\frac{1}{2}$ and in transition $s_2 \rightarrow s_7$ is 0. Considering the leakage previously we get $0 + \frac{1}{2} \times k < k + \frac{1}{2} \times 0$, thus at this stage we choose $s_2 \rightarrow s_7$ and we put $s_2 \rightarrow s_6$ as dotted line. Similarly we know that for s_3, the optimal choice is $s_3 \rightarrow s_9$ with the leakage k. We mark it as k and put $s_3 \rightarrow s_8$ as dotted line. Then we arrive at the initial state s_0. There are three reachable states s_1, s_2, s_3 from s_0, the leakage for $s_0 \rightarrow s_1$ is $0 + \frac{1}{2} \times k$ while for the other two transitions is $k + \frac{1}{4} \times 0$, so we could choose either $s_0 \rightarrow s_2$ or $s_0 \rightarrow s_3$. The solution is showing in the following graph.



The optimal leakage k is achieved by:

$$s_0 \rightarrow s_2 \rightarrow s_7 \rightarrow s_{13} \quad (\{a_2,\, a_3,\, a_1\}), \text{ or } \quad s_0 \rightarrow s_3 \rightarrow s_9 \rightarrow s_{15} \quad (\{a_3,\, a_2,\, a_1\})$$

Alternatively, using Proposition 2, at each stage we can choose the minimal value to get the pessimal leakage:



and the pessimal leakage $\frac{k}{2}$ is achieved by one of the following paths:

$$\begin{aligned}
s_0 \rightarrow s_1 \rightarrow s_4 \rightarrow s_{10} \quad (\{a_1,\, a_2,\, a_3\}) \\
s_0 \rightarrow s_1 \rightarrow s_5 \rightarrow s_{11} \quad (\{a_1,\, a_3,\, a_2\}) \\
s_0 \rightarrow s_2 \rightarrow s_6 \rightarrow s_{12} \quad (\{a_2,\, a_1,\, a_3\}) \\
s_0 \rightarrow s_3 \rightarrow s_8 \rightarrow s_{14} \quad (\{a_3,\, a_1,\, a_2\})
\end{aligned}$$

4.4 Complexity

Computational complexity is a very important factor for implementation and is considered a practical issue for the use of Bellman equation. We denote the computational complexity as R here. Since the execution trees in our state transition systems are acyclic and strictly nondecreasing backwards, the computational complexity of the optimality leakage principle (Proposition 2) is bounded by the number of vertexes (nodes) or edges in the tree, which can be bounded by two factors: the number of choices at each stage and the other is the number of stages.

If the state transition system has n stages[1], with two decisions taken at every stage, this requires $R = O(2^n)$ arithmetical operations. In the general case, if there are n stages in the transition system and at each stage there are m decisions, the complexity for the implementation is of the order of $O(m^n)$ arithmetical operations. The computational complexity will increase significantly with the decisions at every stage. For example, if there are 20 stages and 3 decisions at every stage, we will get 3 486 784 401 operations; in a computer with a speed of 1 million arithmetical operations per second, it will take 3487 seconds i.e 0.97 hour to finish this computation. For this reason there is a strong motivation to simplify the computation otherwise the method would be rarely applicable. Then we have to consider the method to simplify the complexity.

Here we only consider the transition system without considering any transition probabilities. Firstly a transition system can be written as a set of transitions $\mathcal{T}$, in which an element t_{ij} can be written as a triple

$$\langle s_i,\ a_k,\ s_j \rangle$$

where s_i is a starting state and a_k is an action on s_i which transit s_i to the state s_j.

We consider two kinds of improvements. Firstly, since the graph is a tree, there are existing standard algorithms which are much more computationally efficient than $O(m^n)$ for tree-search. Secondly, in the process of using Proposition 2 to solve the leakage bounds, if $L^*(s') = 0$ whatever $P(s'|s, a)$ is, the edge will not contribute to the new value of leakage. In the following algorithm, we are removing these edges whose weight is 0, where there is no interference between h and l.

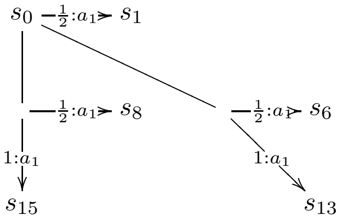
[1] In the examples, each line is seen as a stage. In reality however, instead of tracking every line of program, it is rational to only track the lines which has something to do with the high variable(s).

Table 1. Algorithm to simplify the transition system

Algorithm 1. Simplification algorithm for transition system

```
Require: $\mathcal{T}$ a set of transitions
Ensure: $\mathcal{T} \neq \phi$?
Visited $=\phi$
Waiting $= \mathcal{T}$
repeat
  Get $t_{ij}$ from Waiting
  Visited =Visted $\cup \{t_{ij}\}$
 if $L(t_{ij}) == 0$
   Remove $t_{ij}$ from Waiting
   Modify $t_{j*} \in Waiting$ as $t_{i*}$
 endif
until Visited $= \mathcal{T}$
Return Waiting;
```

For example, in the example 4.3, there are 15 transitions (edges). Now if we use the above algorithm to cut some 0 weight edges, we can then simplify the transition system to be



There are only 5 remaining edges after simplification and the number of edges has reduced by 67%.

4.5 Further Remarks

1. The optimality principle allows for an interesting characterization of leakage in multi-threaded programs, based on what can be leaked from an optimal or pessimal policy. In comparison, previous quantitative result for multi-threaded programs is an overall expectation [13].
2. Since state-transition forms a tree graph, in the program run-time the tree will continuously evolve into subtrees. This allows to track run-time leakage at each time spot, by finding the optimal leakage in the subtree with the knowledge of which previous steps have been taken. Further, this can hopefully allow automatic run-time leakage tracking of programs by attaching such a builtin state-transition tree into the program code segment.
3. Furthermore, we should repeat that we have assumed the attacker can observe the low-variable in every single step of execution, and we have modeled

the state transitions based on that. The stages thus can be seen as a super-set [22]. If other kinds of observational assumptions are desirable, our leakage optimality principle can also be easily adapted to those assumptions by considering a subset of the stages, which would lead to a somewhat simpler state transition graph.

4. Finally, the work remains preliminary with respect to real implementation. For programs following a Turing-complete language (with imperative statements, if statements and for loops) we can hopefully borrow experiences from previous works [16,23,13], although several problems have to be solved, for example, how to cope with non-terminating loops and breaks. This would be an open problem for the next step.

5 Conclusions and Future Work

By extending the Bellman's optimality principle into quantitative information flow, we propose a novel principle for characterizing information leakage and tracking the run-time leakage in multi-threaded programs.

This may create lots of exciting opportunities: according to the static results further actions can be made either to accept or to reject a program, while dynamic measurement can be used for alert or guarantee that an attacker can not acquire a certain quantity of information at run-time. Such a method can also serve as a template for tracking information leaks in state transition systems. Finally, we believe this work demonstrates an interesting perspective by connecting the field of information security with the theory of machine learning.

References

1. Bhargava, M., Palamidessi, C.: Probabilistic Anonymity. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 171–185. Springer, Heidelberg (2005)
2. Bellman, R.: On the Theory of Dynamic Programming. In: Proceedings of the National Academy of Sciences (1952)
3. Braun, C., Chatzikokolakis, K., Palamidessi, C.: Quantitative notions of leakage for one-try attacks. In: Proceedings of MFPS 2009 (2009)
4. Chatzikokolakis, K., Palamidessi, C., Panangaden, P.: Anonymity Protocols as Noisy Channels. Postproceedings of the Symp. on Trustworthy Global Computing. LNCS. Springer, Heidelberg (2006)
5. Chatzikokolakis, K., Palamidessi, C., Panangaden, P.: On the bayes risk in information-hiding protocols. Journal of Computer Security 16(5), 531–571 (2008)
6. Michael, R., Clarkson, A.C.: Myers, and Fred B. Schneider: Belief in information flow. In: Proceedings of 18th IEEE Computer Security Foundations Workshop, pp. 31–45. Aix-en-Provence, France (2005)
7. Cover, T., Thomas, J.: Elements of Information Theory. John Wiley&Sons, Inc., Hoboken (2006)
8. Denning, D.E.: Cyptography and Data Security. Addison-Wesley, Reading (1982)
9. Denning, D.E.: A lattice model of secure information flow. Communications of the ACM 19(5) (May 1976)

10. Di Pierro, A., Hankin, C., Wiklicky, H.: Measuring the confinement of probabilistic systems. Theoretical Computer Science 340(1), 3–56 (2005)
11. Gray III, J.W.: Toward a methematical foundataion for information flow security. In: Proceedings of the 1991 IEEE Symposium on Security and Privacy, Oakland, California (May 1991)
12. Chen, H., Malacaria, P.: Quantifying Maximal Loss of Anonymity in Protocols. In: Proceedings of ASIACCS 2009, Sydney, NSW, Australia, March 10-12 (2009)
13. Chen, H., Malacaria, P.: Quantitative Analysis of Leakage for Multi-threaded Programs. In: Proceedings of ACM 2007 workshop on Programming languages and analysis for security (2007)
14. Chen, H., Malacaria, P.: Studying Maximum Information Leakage Using Karush–Kuhn–Tucker Conditions. In: Proceedings of the 7th International Workshop on Security Issues in Concurrency
15. Clark, D., Hunt, S., Malacaria, P.: David Clark, Sebastian Hunt, Pasquale Malacaria: A static analysis for quantifying information flow in a simple imperative language. Journal of Computer Security 15 (2007)
16. Clark, D., Hunt, S., Malacaria, P.: Quantitative Analysis of the leakage of confidential data. Electronic Notes in Theoretical Computer Science 59 (2002)
17. Clark, D., Hunt, S., Malacaria, P.: Quantified interference for a while language. Electronic Notes in Theoretical Computer Science 112, 149–166 (2005)
18. Backes, M., Kopf, B., Rybalchenko, A.: Automatic Discovery and Quantification of Information Leaks. In: Proceedings of the 30th IEEE Symposium on Security and Privacy, S&P 2009 (2009)
19. Lowe, G.: Quantifying information flow. In: Proceedings of the Workshop on Automated Verification of Critical Systems (2001)
20. Mclean, J.: Security models and information flow. In: Proceedings of the 1990 IEEE Symposium on Security and Privacy. Oakland, California (May 1990)
21. Millen, J.: Covert channel capacity. In: Proceedings of the 1987 IEEE Symposium on Research in Security and Privacy (1987)
22. Malacaria, P., Chen, H.: Lagrange Multipliers and Maximum Information Leakage in Different Observational Models. In: Proceedings of ACM SIGPLAN Third Workshop on Programming Languages and Analysis for Security (June 2008)
23. Malacaria, P.: Assessing security threats of looping constructs. In: Proceedings of ACM Symposium on Principles of Programming Language (2007)
24. Malacaria, P.: Risk Assessment of Security Threats for Looping Constructs. Journal of Computer Security (2009)
25. Puterman, M.L.: Markov decision processes: discrete stochastic dynamic programming., 2nd edn., illustrated. Wiley-Interscience, Hoboken (2005)
26. Sabelfeld, A., Sands, D.: Probabilistic noninterference for multi-threaded programs. In: Proceedings of IEEE Computer Security Foundations Workshop, July 2000, pp. 200–214 (2000)
27. Shannon, C.E., Weaver, W.: A Mathematical Theory of Communication. Univ. of Illinois Press, Urbana (1963)
28. Smith, G.: On the Foundation of Quantitative Information Flow. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 288–302. Springer, Heidelberg (2009)
29. Chatzikokolakis, K., Chothia, T., Guha, A.: Calculating Probabilistic Anonymity from Sampled Data (manuscript) (2009), http://www.cs.bham.ac.uk/~tpc/Papers/CalcProbAnon.pdf