Chapter 5

Linear Dependence Problem

5.1 Introduction

In Chapter 4, we introduced the basic dependence concepts in the context of a perfect nest \( L \) of \( m \) loops. In this chapter, we consider the mathematical problem of actually computing the dependence of a statement on a statement, caused by a pair of variables. To derive the complete dependence information for \( L \), one needs to find the dependence caused by each pair of variables in the program.

We deal with variables that are elements of arrays; scalars can be easily included as a degenerate special case. The first part of the problem is to decide if the two sets of memory locations represented by instances of two variables in \( L \) are disjoint. Since these sets would be necessarily disjoint if the variables were elements of two distinct arrays, we assume from the beginning that the variables being compared are elements of the same array. In real programs, an array subscript is usually a simple linear (affine) function of one or more index variables. We restrict ourselves to the case where each subscript of each variable is a linear function of the index variables of the loop nest. Comparison of two such variables that are elements of the same array leads to a system of linear diophantine equations. We also assume that the limits of a loop in \( L \) are linear functions
of the index variables of loops that contain it. Those limits then will lead to a system of linear inequalities. In fact, more complex limits could be allowed as long as the range of each index variable is defined by a set of linear inequalities (e.g., the limits in the first transformed program in Example 2.1).

Let $S$ and $T$ denote two (not necessarily distinct) statements in the body of $L$. Let $u$ denote a variable of $S$ and $v$ a variable of $T$, and assume that they are elements of the same array. For ease of writing, explicit reference to these variables will often be suppressed. Thus, for example, we will say ‘$T$ does not depend on $S’$ to mean that ‘the variable $u$ of $S$ and the variable $v$ of $T$ do not cause a dependence of $T$ on $S.’$ The type (flow, anti-, output, input) of the dependence is irrelevant in this discussion. It is determined by whether $u$ is an output or input variable of $S$, and whether $v$ is an output or input variable of $T$. In a typical example, we will take $u$ to be the output variable of $S$ and $v$ to be an input variable of $T$. Then, they may cause one or both of the following two dependences: a flow dependence of $T$ on $S$, an anti-dependence of $S$ on $T$. It is sometimes convenient to say that the variables $u, v$ cause a dependence between $S$ and $T$ (or, there is a dependence between $S$ and $T$), if they cause a dependence of $T$ on $S$, or of $S$ on $T$, or both.

Recall from Section 4.3 that the variables $u$ and $v$ cause a dependence of $T$ on $S$, if there are index values $i$ and $j$ of $L$ such that

1. The instance of $u$ for the index value $i$ and the instance of $v$ for the index value $j$ represent the same memory location (denote it by $M$);

2. The instance $S(i)$ is executed before the instance $T(j)$, so that $M$ is referenced by $S(i)$ first;

3. During the execution of $L$, the location $M$ is not written in the time period from the end of execution of $S(i)$ to the beginning of execution of $T(j)$.

From now on, we will not use the third condition. By not using condition 3, we may sometimes label as a dependence what is only