用计算函数模型增强数据流分析

（ 摘 要 ）

精确的数据流分析，需要充分利用条件分支语句提供的逻辑语义。但是由于条件分支语句的逻辑语义可以相当复杂，传统的数据流分析方法为了分析过程的简洁高效，只好对其予以忽略。近年来，随着科学计算程序的日益复杂化和并行化编译器应用范围的日益广泛化，如何处理条件分支语句逻辑语义的问题，也愈发显得难以回避了。采用逻辑推理系统进行相应的处理是一个比较自然的设想。然而完备的逻辑推理系统集成到实用化的编译系统里，编译系统会变得庞大、低效和难以实现，因此在目前的技术状况下还是令人怀疑的。

为了在编译过程中简洁有效地处理条件分支语句，本文提出了对应于程序段的计算函数模型。在这个模型框架下表示条件分支语句的逻辑语义，并利用文中提出的不确定性消解方法，可以把通常需要逻辑推理来处理的数据流分析问题，转化为空间区域之间覆盖关系的判定问题。而这个问题在并行化编译的理论和实践中已有比较成熟的解决办法。相对于在数组区域覆盖运算系统上再集成逻辑推理运算系统的方案，计算函数模型显得是一个更加自然简洁的策略。 理论上它自然地拓广了数组引用区域的概念，使其同时携带了数据引用区域和引用发生的条件区域。而实现上，只需对现有的区域覆盖系统予以相应的扩充即可。对于处理常见的简单情形，有望成为一种简洁有效的数据流分析策略。

用计算函数模型进行精确的数据流分析时，条件谓词之间的逻辑关系被转化为文中定义的Ω，Φ区域之间的覆盖关系。本文详细地讨论在各种常见的程序构造下如何表示、计算和传递Ω区域，Φ区域；并在数据流分析过程中，如何利用Ω区域、Φ区域之间的覆盖关系消除条件分支语句带来的不确定性，以得到更精确的数据流信息。

Enhancing Dataflow Analysis With Computation Function Model

（ Abstract ）

A precise dataflow analysis should effectively exploit the semantic information presented by conditional branch statements. However, because of the potentially enormous complexity inherent in this problem, most traditional systems have to ignore the semantic information of logical conditions, thus avoiding the problem rather than solving it. Recently, as scientific computing programs are going to be more and more complex and the application of parallelizing compilers are going to be spread wider and wider, it turns out to be an avoidable problem on how to effectively deal with the logical semantics of conditional branch statements. Unfortunately, although seemingly quite natural at a first glimpse, it is a difficult, if not an unfeasible, solution to integrate a complete logical reasoning system into a production parallelizing compiler as that will make the compiler system too encumbered, inefficient and difficult to be implemented with current technologies.

In order to solve this problem efficiently and effectively, we introduce here the Computation Function Model corresponding to a program section under analysis. Representing logical semantics within the framework of this model and applying those ideas presented in this paper, we successfully convert the logical reasoning problems into problems on how to decide the coverage relationships between Ω or Φ Regions, which are substantially more feasible problems to be solved by contemporary parallelizing compilers. Compared with two calculus systems to be implemented in a single compiler system, the region calculus system and logical reasoning system inherent in the

reasoning scheme, computation function model seems more simple and efficient. Theoretically it extends the concept of array region so that it brings both data reference region and the logic conditional region that ensures the data reference. Meanwhile, practically it just needs a simple extension to the current region calculus system. Therefore, dataflow analysis within our computation function model can hopefully offer an effective solution for those commonly occurring simple cases that are more critical than those complex but rare ones for practical parallelizing compilation.

Within the framework of Computation Function Model, the logical relationships between branch conditions are represented as the coverage relationship of regions. We not only discuss issues on how to represent, compute and propagate the Ω and Φ Region of conditional reference, but also present methods on how to resolve the nondeterminism caused by conditional branch statements in order to get more precise dataflow information.

目录

1. 背景介绍

1.1 引入计算函数模型的实际背景

数据流分析〖16〗（**Dataflow Analysis**）和相关性测试〖13〗（**Dependence Test**）是并行化编译器分析程序潜在并行性的主要方法。实践证明，在并行化科学计算程序方面，它们已经取得了令人鼓舞的成果〖1〗。而依然存在的诸多问题之一是，现有的程序分析方法还不善于利用条件分支语句的逻辑语义，因而难以充分开发程序的潜在并行性。

并行化编译器主要开发大计算量程序的粗粒度并行性。需要并行化的代码段是程序中计算量大，执行时间占总执行时间相当比例的关键代码段。理论推测和实际统计一致表明：应用程序中广泛出现的条件分支语句，在所难免地会影响对于程序关键代码段的并行性分析。如果不能有效地利用条件分支语句提供的语义信息，那么对程序关键代码段的并行性分析就会流于保守，错失并行计算带来的巨大收益。

数据流分析和相关性测试在程序分析中应用的基本原则是:在控制流简单数据流复杂的情况下使用相关性测试的方法，典型的场合是 DO 循环中数组引用之间相互关系的分析；而在控制流复杂的情况下通常使用数据流分析的方法。相关性测试在其可以应用的范围里是精确的，但其处理的范围比较窄；数据流分析的方法处理范围广，但不够精确，而且处理循环的能力比较差。

传统的数据流分析〖2〗〖3〗侧重分析过程的简捷，对条件分支语句的逻辑语义基本上予以忽略。文献〖4〗等在分析了程序自动并行化方面依然存在的主要问题后认为，充分利用条件分支语句的逻辑语义势在必行。并在〖5〗中用带条件谓词的符号表示法处理该类问题，其实质是 Michael J. Wolfe 提出的 $\phi-function$ 表示法的扩充；此外，还有〖6〗中提出的 **Guarded Array Region** 等。它们都是试图在条件谓词上实施逻辑推理来导出精确的数据流信息。但遗憾的是：完备的自动化逻辑推理系统，实现代价太高，集成到实用化的并行化编译器里并不现实。本文提出的与程序段相对应的计算函数模型〖8〗〖9〗，试图为此找到一种

简单有效的解决方案。

本质上，逻辑推理和计算函数模型是同一个问题的求解在不同数学模型下的表示与运算。而计算函数模型相对于逻辑推理的好处在于：概念上的简洁统一，该模型拓广了数据流分析中的数组区域及其覆盖的概念以统一处理条件的或非条件的定义-引用数据流关系，问题的表述和求解简洁统一； 宜于实现在现有系统中，概念上的简洁统一使得实现是在既有系统上的拓广扩充，无需在数据流分析的框架下再引入逻辑推理系统。

1.2 区域覆盖技术在数据流分析中的应用背景

区域覆盖的概念源自数组数据流分析。并行化编译器的编译对象主要是高性能计算领域中的科学计算程序。该类程序普遍的特点之一是：数据结构简单，数组和标量构成了数据集的主体。因此数组数据流分析是并行化编译器的关键技术之一。数组区域[2][3]（**Array Region**）的表示、运算及其相互之间的覆盖关系是数组数据流分析中的主要问题。

文献〖7〗中提出的相关-覆盖方法就是通过把数组数据流分析中的区域覆盖技术和相关性测试技术有机地结合起来，用以解决数组私有化的判定问题。实验测试表明：相关-覆盖判定法是目前实效最好的数组私有化方法之一[1][7]。但是该方法目前还不能处理条件分支语句产生的流不确定性。鉴于数组私有化是众多并行化变换中最有效的方法之一[4][7][15]，有必要扩充相关-覆盖方法，使之能够处理条件分支语句产生的流不确定性。

在计算函数模型下，条件读写引用的语义可以表示为我们即将在后文中定义的$\Omega_p \xrightarrow{x} Set_p(x)$或$\Phi$区域。他们在形式上和概念上都是对数组区域定义的一个拓广。利用Ω区域，Φ区域的覆盖关系，可以在既有数组数据流分析的框架下处理条件分支语句产生的流不确定性，增强传统的数组数据流分析。此外，Ω区域的覆盖技术还可以应用在符号分析等问题上。

1.3 有关的符号约定:

条件分支语句形式多样，如 **C/C++**，**Java**，**FORTRAN**，**Pascal** 等程序设计语言中出现的各种 **IF** 语句、**Switch** 语句、**case** 语句等，其中以 **IF** 语句出现的尤为广泛，为了叙述的简洁，本文约定：提到的 **IF** 语句和条件分支语句含义等价；另外约定下列标记符号和相关的算符：

符号	名称	解释
P	程序段	一段被编译的程序（in Fortran，C etc.）
r	w	对变量(包括数组，下同)的读/写引用
x，X	引用，引用集	对变量的引用(x 可以取值为 r, r_i, w, w_i)
R_P	W_P	程序段 P 中对被考察变量所有读/写引用
NaW_P^r	无反相关写集	程序段 P 中关于 r 的无反相关写集*
Set_P (x)	引用元素集	x 在 P 中引用的所有变量,数组元素的集合（x 不在 P 中时为空集） $Set_P(X)=\bigcup_{x\in X} Set_P(x)$

*这里定义：对于程序段 P 中的读引用 r，称 P 中写引用 w 是关于 r 的无反相关写，如果 r 与 w 在 P 中无反相关。P 中所有关于 r 的无反相关写构成 P 中关于 r 的无反相关写集 NaW_P^r。

2.计算函数模型的基本概念

2.1 程序段的计算函数模型

在可计算理论里,任何程序都可以视为一个从输入数据空间到输出数据空间的函数映射。设 **m, n** 分别为输入，输出空间的维数， 那么程序对应于映射：$\psi_P^{(m)}: Z^m \to Z^n$。同理，在数据流分析过程中，对于当前被分析的程序段 P，也可以采用这种函数映射式的程序模型。

定义 2.1：程序段 P 的定义变量集 $In(P)=\{x_1,x_2,\cdots,x_m\}$表示程序段 P 动态执行时，P 中任何读引用读取之前没有被 P 定义的所有变量的集合；**程序段 P 的定义域：** $Dom(P)=D_{x_1}\times D_{x_2}\times\cdots\cdots\times D_{x_m}$。

定义 2.2：程序段 P 的映象变量集 $Out(P)=\{y_1,y_2,\cdots,y_n\}$表示程序段 P 执行时被 P 所定义的，并且被程序其他部分读引用的变量集合；**程序段 P 的值域：** $Range(P)=D_{y_1}\times D_{y_2}\times\cdots\cdots\times D_{y_m}$。

这里 D_{x_i}是x_i的定义域， D_{y_i}是y_i的定义域，依据变量的不同类型可以有不同的取值范围。

定义 2.3：程序段 P 的计算函数模型定义为从 P 的定义域$Dom(P)$到 P 值域$Range(P)$的向量值函数映射： $f_P: Dom(P)\to Range(P)$。

上述定义中把数组，数组的子区域(Subarray)视为一组分立的变量，带下标的数组元素引用视为一个普通变量。 有些程序段有对其输入变量集进行约束的条件谓词$\varphi(x_1,x_2,\cdots,x_m)$，此时程序段 P 的定义域应该是：

$Dom(P)=\{(x_1,x_2,\cdots,x_m)\mid\varphi(x_1,x_2,\cdots,x_m)=True\}$。

为了在上述计算函数程序模型的基础上进行推理,我们还假设下面的程序执行方式：

确定性执行原理：对于程序段 P 定义域中的任何确定的一点（对应着程序段

P 的一次实例执行）：$(x_1,x_2,\cdots,x_m)\in Dom(P)$，程序段 P 的执行是完全确定的，这种确定性包括：

- P 的 $In(P)=\{x_1,x_2,\cdots,x_m\}$ 和 $Out(P)=\{y_1,y_2,\cdots,y_n\}$ 确定；
- P 计算结束后 $Out(P)=\{y_1,y_2,\cdots,y_n\}$ 中的值也是确定的。即 P 是确定性的程序，f_P 是严格数学意义下的函数映射；
- 条件分支语句的转向确定。

确定性执行原理对于我们后面的推理有着基本的重要性。

2.2 不确定性概念的引入

定义 2.4：如果 $\exists A\subset In(P), A\neq\varnothing$，当 A 中的变量取不同的值时，$In(P)$ 或 $Out(P)$ 也取值为不同的变量集，则称程序段 P 是*数据流不确定的*，而 A 中的变量是造成这种不确定性的变量，在编译器中需要进行符号分析，我们称之为*关键值变量或符号变量*。

我们称程序段 P 的数据流不确定现象为流不确定性，流不确定性主要由条件分支语句，指针别名等产生；也会由于某些关键值的不确定，通过数据引用，DO 语句的迭代范围等导致。其直观意义是，程序段 P 对于某些变量是否进行了读写由于是在一定逻辑条件下进行的，或读写的范围由于某些变量值的不明确而变的不能确定。

当数据流确定时，如果对于编译器来说，变量定义的具体数值不明确，我们称之为*值不确定性*。值不确定性可以由表达式，条件分支语句，程序变量的输入值等导致。其直观含义是：程序段 P 的读写引用区域是确定的，但程序段 P 中具体数值计算的映射关系不明确。

确定性执行原理下，程序数据流信息的不确定性源自程序输入数据的不确定（可以是 $Dom(P)$ 中的任何一点），或计算结构的复杂度超出编译器的分析能力。

任何程序 P 都可以视为是某个可计算函数 f 的实现，编译器的任务是产生一个同 f 等价的映射 $f_{optimized}$：它和 f 具有在 $Dom(f)$ 内完全相同的值对应关系，只

是 $f_{optimized}$ 必需具有更加适合于底层计算模型的计算结构以获取高性能。编译器不是映射 f 的具体实施，不关心具体的数值计算过程，所以值不确定性对编译器无足轻重；但流不确定性意味着变量值在程序中的传递关系不明确(计算函数的复合关系不明确)，是编译器对于程序计算结构的识别和理解不明确。因此流不确定性对于编译器是关键的。编译器在对程序的一个局部进行并行性的分析与变换时，需要识别，消解编译时刻可以确定的流不确定性以保持在本质的计算约束下并行化，优化程序。否则，就只能在不确定性的可能范围里做出保守的假设。

2.3 条件谓词在计算函数模型下的语义

在计算函数模型下，程序段 P 中任何条件分支语句的条件谓词，在经过一定的变量替换后都对应着 $Dom(P)$ 中一个特定子区域。（在无任何条件约束的情况下为 $Dom(P)$）

设 $Predicate(t_1,t_2,...,t_k)$ 是条件分支语句中的逻辑条件，$t_1,t_2,...,t_k$ 是条件谓词中直接出现的谓词变量，那么 $Predicate(t_1,t_2,...,t_k)=True$ 确定了 $D_T=D_{t_1}\times D_{t_2}\times\cdots\cdots\times D_{t_k}$ 空间中的一个区域 Ω_T，由程序中从 P 的定义变量到参数变量 $t_1,t_2,...,t_k$ 的一系列赋值语句，表达式等确定的映射关系：

$$\varphi:(x_1,x_2,...,x_m)\to(t_1,t_2,...,t_k)\qquad \begin{aligned} t_1&=t_1(x_1,x_2,...,x_m)\\ t_2&=t_2(x_1,x_2,...,x_m)\\ &\cdots\cdots\\ t_k&=t_k(x_1,x_2,...,x_m)\end{aligned},$$

可知，D_T 是 $Dom(P)$ 的一个导出参数空间。$\Omega_P=\varphi^{-1}(\Omega_T)$ 表示在 P 的定义域中，是在 P 的定义域内，该条件分支执行的程序状态空间区域。而 Ω_T 表示在参数空间 D_T 中。这里的 $\Omega_P=\varphi^{-1}(\Omega_T)$ 表示：

$$\Omega_P=\{(x_1,x_2,...,x_m)\,|\,(x_1,x_2,...,x_m)\in Dom(P);\varphi(x_1,x_2,...,x_m)=(t_1,t_2,...,t_k)\in\Omega_T\}$$

事实上，谓词 $p(x_1,x_2,\cdots,x_n)$ 是程序状态空间中特定空间区域或点集 Ω_p 的特征函数：

$$p(x_1,x_2,\cdots,x_n)\Leftrightarrow(x_1,x_2,\cdots,x_n)\in\Omega_p,\ \forall(x_1,x_2,...,x_n)\in Dom(P)$$

定义 2.5： 以程序段 P 中的条件谓词 $Predicate(t_1,t_2,...,t_k)$ 作为特征函数，在 P 的定义域 $Dom(P)=D_{x_1}\times D_{x_2}\times\cdots\cdots\times D_{x_m}$ 中确定的空间区域 Ω_P，以及所有表示在 $Dom(P)$ 的导出空间 $D_T=D_{t_1}\times D_{t_2}\times\cdots\cdots\times D_{t_k}$ 中的区域 Ω_T 称为该谓词的 **Omega 区域，记为** $\Omega_P^{Predicate}$，在不导致岐意时，简化为 $\Omega_{Predicate}$。

数据集空间 *DataSpace*，定义为程序中由数据变量的存储单元组成的存储空间。标量 S 的 *DataSpace* 是标量 S 所代表的一个存储单元空间；数组 A 的 *DataSpace* 是数组 A 所代表的 $dim(A)$ 维存储空间，其上下界由数组说明提供；程序段 P 的 *DataSpace* 由程序段 P 的所有变量，包括标量和数组的全体，组成的存储空间。下面的讨论中主要涉及数组的 *DataSpace*。

为了导出写集覆盖读的充要条件，我们需要考虑 Omega 区域和数组引用区域的笛卡积。它代表程序状态空间和程序数据集空间的笛卡积：$Dom(P)\times DataSpace$ 中的一个特定区域：$\Omega_x\times Set_P(x)$。其直观意义是：程序段 P 在 满足 Ω_x 的程序状态下引用数据区域 $Set_P(x)$。

定义 2.6 程序段 P 中引用 x 的**状态-数据引用区域**，简称 **Φ 区域**，定义为：$\Phi_P^x=\Omega_P^x\times Set_P(x)$。$\Omega_P^x$ 表示在程序段 P 范围内约束引用 x 的 Omega 区域，在不产生岐意时简记为 Ω_x。Φ_P^x 也记为 $\Phi_P(x)$。定义：$\Phi_P(X)=\bigcup_{x\in X}\Phi_P(x)$。

引理 2.1 设程序段 P 中的逻辑谓词 $p(x_1,x_2,\cdots,x_n)$ 确定的 Omega 区域为 Ω_p，则：

1. $\neg p(x_1,x_2,\cdots,x_n)\Leftrightarrow(x_1,x_2,\cdots,x_n)\in\overline{\Omega}_p=Dom(P)-\Omega_p$
2. $p_1(x_1,x_2,\cdots,x_n)\wedge p_2(x_1,x_2,\cdots,x_n)\Leftrightarrow(x_1,x_2,\cdots,x_n)\in\Omega_{p_1}\cap\Omega_{p_2}$
3. $p_1(x_1,x_2,\cdots,x_n)\vee p_2(x_1,x_2,\cdots,x_n)\Leftrightarrow(x_1,x_2,\cdots,x_n)\in\Omega_{p_1}\cup\Omega_{p_2}$
4. $p_2(x_1,x_2,\cdots,x_n)\rightarrow p_1(x_1,x_2,\cdots,x_n)\Leftrightarrow\Omega_{p_1}\supseteq\Omega_{p_2}$
5. $p_1(x_1,x_2,\cdots,x_n)\leftrightarrow p_2(x_1,x_2,\cdots,x_n)\Leftrightarrow\Omega_{p_1}=\Omega_{p_2}$
6. $p(x_1,x_2,\cdots,x_n)\rightarrow\bigvee_{i=1}^{n}p_i(x_1,x_2,\cdots,x_n)\Leftrightarrow\bigcup_{i=1}^{n}\Omega_{p_i}\supseteq\Omega_p$

证明是平凡的，略。

引理 2.2 设 P 是关于程序变量的逻辑谓词集，Ω_P 是以 P 中谓词为特征函数生成

的Ω区域的集合，则点集运算系统$\left[\Omega_{\mathrm{P}};\bar{},\cap,\cup\right]$同态于逻辑运算系统$\left[\mathrm{P};\neg,\wedge,\vee\right]$。

由引理 2.1，易知Ω_{P}的生成映射$\varphi:\mathrm{P}\to\Omega_{\mathrm{P}}$是到上的同态映射，证明略。

引理 2.1，2.2 表明：条件分支语句中，逻辑谓词对应的 Omega 区域，其相互间的覆盖关系对应着逻辑谓词之间的逻辑关系。因此计算函数模型的引入，可以把逻辑推理的问题变换为空间区域之间覆盖关系的判定问题。

逻辑谓词及其对应的 Omega 区域在表示形式上是一致的：都是关于程序变量的等式或不等式组等。区别在于作用于其上的运算系统不同，区域覆盖的方法更加容易和现有技术融和，更加宜于实现。判定 Omega 区域之间的覆盖关系时，没有必要把变量全部替换为 P 的定义变量，而只需把两个 Omega 区域化到同一个参数变量空间即可。文献〖5〗中提出了一个逐步反向替代的变量替换方法。对实用程序，特别是科学计算类的程序，的分析统计表明，多数条件分支语句的逻辑条件谓词确定了其参数空间里的线性凸区域，或线性凸区域中按照一定规律排布的整数点集。而针对这种区域、点集的集合运算处理方法（如线性不等式组，等）在并行化编译器的实践中已经比较成熟。

在 FORTRAN 程序及 C 语言程序中，这里的程序段 P 可以是程序中各种实体：整个程序，某个子程序，某块代码，DO 的循环体，甚至是一个语句或表达式。这种统一的看法有助于推出一个统一的处理方法。它既可以具有局部有效性，又可以进行全局的通盘筹划处理。

2.4 计算函数模型下条件分支语句的语义

条件分支语句的一般形式：

$$
\begin{array}{lll}
\textit{if} & & \textit{if} \\
[\,] \quad P_1 \to B_1; & & [\,] \quad \Omega_{P_1} \to B_1; \\
[\,] \quad P_2 \to B_2; & & [\,] \quad \Omega_{P_2} \to B_2; \\
\cdots \quad \cdots & \textit{or} & \cdots \quad \cdots \\
[\,] \quad P_n \to B_n; & & [\,] \quad \Omega_{P_n} \to B_n; \\
\textit{fi} & & \textit{fi}
\end{array}
$$

由于B_1，B_2，…，B_n执行的不确定性导致了程序对数据读写的不确定性，

这种不确定性由各个可执行语句体 B_i 的前置谓词 P_i 决定。 P_i 的引用区域是完全确定的，不会导致不确定性。

条件分支语句对变量定义的语义作用:

设编译器当前分析的程序段 P，条件分支语句对其中变量定义产生的语义作用可以是：

- **值不确定性定义(写)** 参见图 1,2 中的示例，作为一个整体，程序段的所有读写区域是确定的，但变量定义的数值不确定。 此时 IF 语句对于变量定义的语义作用是计算表达式的一部分，主要意图是在不同条件下赋予变量不同的数值。如果条件分支语句的此类语义作用不影响程序的计算结构(比如其产生的变量值出现在其他 IF 语句的条件表达式，DO 语句的上下界表达式，或数组下标表达式中)时，可以把它做为一种表达式来处理。 而当条件分支语句下的定义结果影响程序的计算结构时，则该值不确定性定义是**符号变量定义**，需要进行符号分析。我们采用下面的标记表示其语义：$\Omega_p \xrightarrow{w} ExpVal_i$，其中 Ω_p 是条件谓词 p 的 Omega 区域，w 表示是写/定义，ExpVal 表示相应的计算表达式。
- **流不确定性定义(条件写)** 参见图 3 的示例。IF 语句决定的是变量，特别是数组区域(Array Region)在一定条件下的写区域。逻辑条件成立的不确定导致数据定义，引用关系的不确定，其中的逻辑语义暗示了程序计算结构的约束关系。相应的语义可以标记为：$\Omega_p \xrightarrow{w} Set_p(x)$，

```
Smax = A(1)
DO   I = 2, N
     IF (Smax < A(I))   THEN
          Smax = A(I)
     ENDIF
ENDDO
```

$$Smax = \max_{i=1}^{n}(A(i))$$

图 1

```
DO 103   K=1,7,1
     QIN(K)=QN1(I,K)
     IF (QN1(I,K).LT..0) THEN
          QIN(K)=.0
     ENDIF
ENDDO
QIN(K) = max( QN1(K) , 0.0 )
```

图 2

条件分支语句产生的流不确定性示例:

```
[1]     DO   K = 2, 5, 1
[2]         IF (RS(4+K) .LE. CUT2) THEN
[3]             RL(4+K) = SQRT(RS(4+K))
[4]         ENDIF
[5]     ENDDO
[6]     IF (KC .EQ. 0) THEN
[7]         DO K = 11, 14, 1
[8]             FF((-5)+K) = FF((-5)+K)+ AB2 * EXP(-B2 * RL(K - 5 )) / RL((-5)+K)
[9]         ENDDO
[10] ENDIF
```

整个程序段是否定义，引用了 RL（6 ： 9）是不确定的。

图 3.

流不确定性一定由值不确定性引发,并且值不确定性定义和流不确定性定义是相对于当前程序段 P 的，对于程序段 P 是流不确定性的定义在一个更大的程序范围里可能会成为值不确定性定义，反之亦然。区分条件分支语句的上述两种语义是出于数据流分析过程的需要。并且区别这两种条件分支语句对于变量定义的语义作用不能简单地通过对条件分支语句的词法分析和语法分析得到，而是需要一定范围里的数据流分析以确定其计算语义。

条件分支语句对变量读引用的语义作用:

条件分支语句对于变量读引用的语义作用是:该读引用是在一定逻辑条件下进行的，这里称之为**条件引用(读)**。相应逻辑条件确定的 Omega 区域内是该读引用发生的程序状态集。并且也是我们用来化解定义-引用之间数据流不确定性的关键条件。条件读的语义在计算函数模型下表示为:

$$\Omega_p \xrightarrow{r} SymbolicVariable \quad ; \quad \Omega_p \xrightarrow{r} Set_p(x)$$

条件分支语句的处理:

1. 对于由条件分支语句导致的单纯的值不确定性定义予以忽略。由于科学计算程序中。相当部分的条件分支语句是值不确定性定义，这种忽略可以大大减轻编译器的计算量。 而对于条件分支语句下的符号变量定义，设法推测其值的情况。此时值的信息会导出流的信息。本文第四部分介绍如何利用计算函数模型辅助符号分析，处理其中的逻辑条件。
2. 流不确定性定义的处理。利用条件读中的逻辑条件进行定义-引用数据流分析，消解编译时刻可以明确的数据流的不确定性。下面证明的覆盖定理将利用前面介绍的 Omega 区域之间的覆盖关系来解决这个问题。需要指出的是，许多场合下的数据流不确定性是不能在静态得以明确的,它们本质上是动态确定的数据流关系。

3. Ω区域，Φ区域的表示与计算

3.0 引言

在最一般的意义下，条件谓词的Ω区域在$Dom(P)$空间中可能是非常复杂的空间区域，因而是难以计算的；判定其相互之间的覆盖关系在一般意义下是 NP 问题。令人却步的复杂度源于程序逻辑语义本身的潜在复杂性。

值得庆幸的是，编译器的分析优化并不需要处理所有的可能情况，而只需处理好实际情况中常见，影响程序优化、并行化变换的情形。对 **SPEC95fp**，**PERFECT** 等测试程序包的分析表明：条件谓词的 Omega 区域在其直接出现的变量集$(t_1,t_2,...,t_k)$上通常可以表示为线性凸区域。把$\Omega_T(t_1,t_2,...,t_k)$视为 Omega 区域的参数形式。当需要判定覆盖关系的 Omega 区域处在不同的参数空间时，进行参数的反向替代以使他们处于同一参数空间并进行覆盖关系的比较，如果参数替换过程产生过于复杂的表达式或过于复杂的 Omega 区域，使得判定 Omega 区域之间的覆盖关系变的过于复杂，可以视为静态不可判定的问题。对于程序段 P，参数的替代最终只能进行到所有的参数全部替换为$In(P)$中的变量，所以这个替换过程可行，并且一定结束。

并行化编译器主要分析结构化的程序段，它们一般由表达式，赋值语句，条件分支语句，各种循环语句以及顺序结构等构成。并行化变换主要优化确定次数的循环，例如 **FORTRAN** 中的 **DO** 循环，因为它们是最具有并行性的程序构造。**DOWHILE** 和 **REPEAT-UNTIL** 类型的循环本质上是递归计算，难以有效地并行计算，因此这里不再考虑。

3.1 条件谓词的Ω区域表示

条件分支语句的逻辑谓词主要有这样几种常见的基本形式：数值关系的比较，如：$=,\leq,\geq,\neq,<,>$；整型变量的 MOD 运算；绝对值运算；以及在此基础上的逻辑运算。由引理 1.1，逻辑谓词可以直接表示为 Omega 区域的特征函数表达

式：

1. 数值关系的比较，整型变量的 MOD 运算直接化为空间区域的约束方程、不等式。例如：

IF (X > 0) 化为 $\Omega = \{Dom(P) | X > 0\}$

IF (X <=4) 化为 $\Omega = \{Dom(P) | X \le 4\}$

IF (MOD(X , 2) = 0) 化为 $\Omega = \{Dom(P) | mod(X,2) = 0\}$

2. 绝对值运算化为线性方程组。例如：

IF (|X| < 1.0) 化为 $\Omega = \{Dom(P) | X < 1.0; X > -1.0\}$。

3. 逻辑运算化为区域之间的交并补运算：

IF (Cond1 .and. Cond2) 化为 $\Omega = \Omega_1 \cap \Omega_2$

IF (Cond1 .OR. Cond2) 化为 $\Omega = \Omega_1 \cup \Omega_2$

IF (.NOT. Cond1) 化为 $\Omega = \neg(\Omega_1)$

例如：

IF (.NOT. (X > 4)) 化为： $\Omega = \neg\Omega_1 = \{Dom(P) | X \le 4\}$

IF ((X > 2) .AND. (Y > 1)) 化为：$\Omega = \Omega_1 \cap \Omega_2 = \{Dom(P) | X > 2; Y > 1\}$

为了描述上的简洁性，在以下的讨论中，我们将把 $\{Dom(P) | p(x_1, x_2, \cdots, x_n)\}$ 简记为 $\{p(x_1, x_2, \cdots, x_n)\}$，而区域表示中出现的‘；’表示与关系。

3.2 嵌套在条件分支语句下的情形

设嵌套在条件分支语句 *if* 内部的 Omega 区域为 Ω_i，则在 *if* 之外其相应的 Omega 区域 Ω_o 是 Ω_i 与其所在 *if* 分支之条件谓词 Omega 区域的交。

由 Φ 区域的定义：$\Phi_P^x = \Omega_{px} \times Set_P(x)$，易知，Φ 区域的计算只是相应 Ω 区域的计算。

例 1：

```
[1]       IF ( X > 4 ) THEN
[2]           IF ( Y > 0 ) THEN
[3]               DO I = 1 , 10
[4]  w                A ( I ) = …
[5]               ENDDO
[6]           ENDIF
[7]       ENDIF
```

程序段[3]到[5]中，w 的数组引用区域是 A（1:10）；程序段[2]到[6]中，[2]之条件谓词的 Omega 区域是$\Omega_2 = \{Y > 0\}$。数组定义点 w 在程序段[2]到[6]范围中的Φ区域：

$$\Phi_{2-6}(w) = \Omega_2 \times Set_{2-6}(w) = \{Y > 0\} \times \{A(\phi) | \phi = 1,\cdots,10\}$$

程序段[1]到[7]下，嵌套的 Omega 区域是$\Omega = \Omega_1 \cap \Omega_2 = \{X > 4; Y > 0\}$，数组定义点 w 的Φ区域：

$$\Phi_{1-7}(w) = \Omega \times Set_{1-7}(w) = \{X > 4; Y > 0\} \times \{A(\varphi) | \phi = 1,\cdots,10\}$$

3.3 嵌套在 DO 语句中的情形

嵌套在 DO 语句中的 IF 语句，其计算语义可以非常复杂。这里只需考虑对数据流分析有利的几种常见情形。其他复杂情形可以用 2.4 中提出的忽略值不确定性 IF 语句的方法排除在分析范围之外。

嵌套在 DO 语句中的逻辑条件可以含有标量或数组变量。

如果嵌套在 DO 语句中的 IF 语句，其条件中仅含标量，而且是非递归标量，则该 IF 语句为循环不变条件分支语句，可以提到 DO 循环之外，用 3.2 所述方式处理。

如果含递归标量（Induction Variable），那么该逻辑条件是对循环的迭代空间进行约束的逻辑谓词。设条件引用 x 嵌套在程序段 P 的 n 重 DO 循环中，相

应的循环变量为$i_1,i_2,\cdots i_n$， $t_1,t_2,\cdots t_k$是直接出现在谓词$P(t_1,t_2,...,t_k)$中的变量，该谓词确定的$D_T = D_{t_1} \times D_{t_2} \times \cdots\cdots \times D_{t_k}$中的区域$\Omega_T$，可以表为$i_1,i_2,\cdots i_n$和$In(P)$上的方程和不等式组。当需要考虑 X 在 n 重循环之外的数据流关系时，需要把 Omega 区域投射到循环外程序段 P 的定义变量集$In(P)$上。投射过程把循环变量$i_1,i_2,\cdots i_n$从 Omega 区域表达式中消去，事实上消去的过程和数组引用区域向 DO 循环外扩的过程对应。循环变量的语义作用从逻辑条件约束转变为数组区域的约束。

例 2：

```
1.  do   LL = 1,31
2.       N = LL - 2
3.       do   I = 1, 31
4.            N = N + 1
5.            if   ( N >= 31) then
6. w               Y(I, LL) = Y (I, LL) + S(I, LL) + TEM (I, LL)
7.            endif
8.       enddo
9.  enddo
```

在循环体中(语句 4-7 中)，w 的区域为：

$$\begin{aligned}\Phi_{4-7}^{w} &= \Omega_{4-7}^{w} \times Set_{4-7}(x) = \{N \geq 31\} \times \{Y(\phi_1,\phi_2) | \phi_1 = LL; \phi_2 = I\} \\ &= \{1 \leq LL, I \leq 31; LL + I - 2 \geq 31\} \times \{Y(\phi_1,\phi_2) | \phi_1 = LL; \phi_2 = I\} \\ &= \{Y(\phi_1,\phi_2) | \phi_1 = LL; \phi_2 = I; 1 \leq LL, I \leq 31; LL + I - 2 \geq 31\}\end{aligned}$$

出循环后，w 在整个程序段 P 中的区域为：

$$\begin{aligned}\Phi_{P}^{w} &= \Omega_{P}^{w} \times Set_P(x) \\ &= Dom(P) \times \{Y(\phi_1,\phi_2) | \phi_1 + \phi_2 - 2 \geq 31; 1 \leq \phi_1,\phi_2 \leq 31\} \\ &= \{Y(\phi_1,\phi_2) | 1 \leq \phi_1,\phi_2 \leq 31; \phi_1 + \phi_2 - 2 \geq 31\}\end{aligned}$$

语句 1-9 中的条件区域是 P 的定义域： $\Omega_w = Dom(P)$，

引用区域：

$$\begin{bmatrix} o & o & . & . & . & o & o & o \\ o & . & & & . & o & o & x \\ . & . & & & & o & x & x \\ . & . & & & o & x & x & x \\ . & . & & o & x & . & . & . \\ o & o & o & x & . & . & . & . \\ o & o & x & x & . & . & . & x \\ o & x & x & x & . & . & x & x \end{bmatrix}_{31\times31}$$

如果嵌套在 DO 循环中的 IF 语句，其逻辑条件含有数组变量并且数组引用的下标表达式中含有递归标量（否则视为标量处理），此时的逻辑条件是关于该数组特定区域的向量式的逻辑条件，需要对逻辑条件进行向量化扩展。理论上，向量化的逻辑条件可以蕴含着相当丰富和复杂的语义，但在并行化常见的科学计算类程序时，特别需要处理好下列两种常见的模式：

1. 向量式逻辑条件的条件归约。此时数组的下标表达式一般是严格单调的，设循环的迭代的空间大小为 n，则根据各个被判定数组元素相对该逻辑条件是否成立可以有 2^n种情形，但在编译器里通常只需考虑全部成立或全部不成立的情形。条件全部成立时，归约变量进行一般的归约运算，相应的Ω区域为各个分量满足逻辑条件时的联立；条件全部不成立时，归约变量不变。相应的Ω区域为各个分量不满足逻辑条件时的联立。 参见例 3。
2. 向量式逻辑条件对应下的向量引用。此时逻辑条件和向量引用中的数组下标表达式一般都是严格单调的，根据逻辑条件中数组元素的取值决定是否对相应的引用数组元素进行读写引用。这里也只需考虑向量式逻辑条件全部成立或全部不成立的情形，并且相应的Ω区域计算方法同上，而被引用的数组区域同一般的数组区域计算方法相同。参见例 4。

例 3：

```
[1]     KC = 0
[2]     DO K = 1, 9, 1
[3]         IF (RS(K) .GT. CUT2) THEN
[4]                 KC = KC + 1
[5]         ENDIF
[6]     ENDDO
```

[3]中的条件是 RS（1:9）的向量化逻辑条件。KC 的定义在[1]到[6]的范围是值不确定性定义。 如果 KC 是符号变量，则需要对其进行符号分析。KC 对向

量条件：*RS(K) . GT. CUT2，K=1, …, 9* 进行了条件归约。因此 KC 的取值范围：$0 \le KC \le 9$。

KC=0 对应于条件[3]关于 *K=1, …, 9* $0 \le K \le 9$ 全部不成立的 $Dom(P)$ 中的一个区域：$\Omega = \bigcap_{K=1}^{9} \{RS(K) \le CUT2\}$；

KC=1 表示有且只有一个 K 使得 $RS(K) > CUT2$ 成立的 $Dom(P)$ 中的一组区域：

$$\Omega = \bigcup_{i=1}^{9} \Omega_i \quad : \quad \Omega_i = \left(\bigcap_{K \in [1..9]-i} \{RS(K) \le CUT2\} \right) \cap (RS(i) > CUT2),$$

以此类推。

KC=9 表示 $\forall K, 1 \le K \le 9$ 条件[3]都成立的区域：$\Omega = \bigcap_{K=1}^{9} \{RS(K) > CUT2\}$

这种形式的向量化条件关系在科学计算程序中比较常见。

例 4：

```
[1]     DO   K = 2, 5, 1
[2]          IF   (RS(4+K) .LE. CUT2)   THEN
[3]               RL(4+K) = SQRT(RS(4+K))
[4]          ENDIF
[5]     ENDDO
```

[1]行到[5]行中的 IF 语句导致了程序段 P 中的一个条件写，RL(6:9)的定义和 RS(6:9)满足(RS(4+K) .LE. CUT2)的情况是一致的。当条件全部成立时：$\Omega_w = \bigcap_{K=6}^{9} \{RS(K) \le CUT2\}$，数组区域 RL(6:9)也全部被定义，条件全部不成立时，数组区域 RL(6:9)也全部没有被定义。

需要指出的是：上述两例中，由于Ω区域的表达式中出现了数组元素，并且是对一定范围中的每个元素都成立，例如：$\Omega_w = \bigcap_{K=6}^{9} \{RS(K) \le CUT2\}$ 等，因此有必要对现有的区域运算系统在表示和运算方法上进行相应的扩充。

3.4 过程间的传递

当被分析的程序段 P 含有过程调用时，需要在调用过程和被调过程之间传递数据流信息。条件谓词的 Omega 区域主要是关于标量的约束关系，找出虚参和实参之间的对应关系并进行标量的替换即可。数组引用区域的跨过程传递复杂的多，此时虚参数组与实参数组可以有不同的形状定义，实参数组也无需是从一个数组的首元素开始，因此这里要解决数组区域的 RESHAPE 问题[3]。

3.5 区域运算及区域覆盖关系的判定

区域之间的基本运算：交，并，减。

n 维空间中任意两个区域之间的集合运算和覆盖关系判定是很困难的。但在实际应用程序中，逻辑条件的 Omega 区域经常是线性凸区域，甚至是各维之间相互独立的正则区域。

变换到同一变量集之后，设两个 n 维空间区域：Ω_1，Ω_2 分别由如下不等式组表示：

$$\Omega_1: \quad \sum_{j=1}^{n} a_{ij} \cdot x_j \le c_i \quad i = 1,2,\cdots,m \qquad (1)$$

$$\Omega_2: \quad \sum_{j=1}^{n} b_{ij} \cdot x_j \le c_i^{'} \quad i = 1,2,\cdots,k \qquad (2)$$

Ω_1，Ω_2 中的等式 $\sum_{j=1}^{n} a_{ij} \cdot x_j = c_i$ 可以通过代入消元法消除，如果等式中的变量集与不等式组的变量集相互独立，则分解为两个子问题。只含等式的子问题用求解线性方程组的方法处理，否则用下面的方法处理。

交运算是两个区域的交集，其结果是合并两个不等式组；并运算是区域的并集，可能是非凸线性空间区域。非凸线性空间区域可以用线性凸区域的列表进行表示；减运算也会导致非凸线性空间区域。

判 定 Ω_1 覆盖 Ω_2 等价于判定区域 Ω_2 处于平面组：

$$\sum_{j=1}^{n} a_{ij} \cdot x_j = c_i \quad i = 1,2,\cdots,m \qquad (3)$$

每个平面的内侧。判定的方法是：

如果方程（3）在约束：$\sum_{j=1}^{n} b_{ij} \cdot x_j < c_i^{'} \quad i = 1,2,\cdots,k$ 下无解，则区域Ω_2处于（3）的同一侧；如果方程（3）上任意一点到Ω_2内任意一点的向量与（3）的内向法向量的内积大于零，则Ω_2处于平面（3）的内侧。

实际不等式组中同时含有≤,<时，需要处理边界条件。正则区域的判定要简单的多，只需对各维作实轴上的区域覆盖判定。实际问题中经常遇到正则区域的覆盖关系判定。

文献〖10〗中的关于 Diophantine 方程，不等式组的 Omega 测试法也可以用来测试整系数区域的相交和覆盖关系，特别是线性凸区域内整数点集之间的相交和覆盖关系判定。

覆盖关系$\Phi_P^r \subseteq \bigcup_{w \in W} \Phi_P^w$的判定要复杂许多，因为$\bigcup_{w \in W^*} \Phi_P(w)$可以是非凸线性空间区域。判定时需要从列表中逐个取出元素去减去Φ_P^r区域列表中的元素，Φ_P^r变成空区域时，表明可以覆盖，否则认为不可覆盖。

Ω区域和Φ区域的各维可以是不同的数据类型，因此可以有不同类型的定义域，比如整型变量和实型变量。但是由于在一般的逻辑条件中不会有不同数据类型变量之间的耦和，因此在相应的Ω区域和Φ区域中，不同类型变量的各维一定是相互独立的，其运算和覆盖关系的判定可以分别进行，不会导致不同数据类型之间耦和的复杂性。

当需要判定覆盖关系的两个空间区域非常复杂时，问题的复杂性源自逻辑谓词之间逻辑关系的复杂性。

4. 用计算函数模型增强数据流分析

4.1 数组数据流分析中的区域覆盖技术

在数据流分析中确定读引用的数据源以得出程序段的确切数据流信息是许多并行化、优化变换的前提。但是由于条件分支语句限定下的条件定义没有 **kill** 掉其他定义点的写，甚至是该定义点以前实例的定义，由此导致了读引用对于同一存储变量的读取可能读到程序段 P 当前实例之外的定义点，产生流不确定性。忽略这种不确定性意味着无条件地接受其他数据源的可能，得到的分析结果趋于保守。

在计算函数模型的框架下可以把问题表述为：

对于一个读引用 r：$\Omega_{pr} \xrightarrow{r} Set_p(r)$（表示在计算函数模型下，下同），以及可能为 r 提供数据源的定义点集 $W = \{w: \Omega_{pw} \xrightarrow{w} Set_p(w)\}$，如何确定 W 和 r 之间的数据流关系。

下面的讨论针对特定的数组 A，标量的情形可以简单的推出；程序段 P 可以是任何复杂嵌套情形下的结构化程序段。

定义 4.1 **写覆盖读**：对于程序段 P 中一对读写 r，w（或写集 W）。如果在 P 的任何执行实例中：写的执行在读 r 之前执行并且 r 读到的数据集都被写 w（或写集 W）的定义数据集所包含，换句话说：写 w(或写集 W)完全可以满足读 r 的数据引用需求。则我们称写 w 覆盖了读 r（或写集 W 覆盖了读 r），记为：$w \xrightarrow{f} r$。或 $W \xrightarrow{f} r$。

定理 4.1 如果程序段 P 中一对读写 r，w：$w \in NaW_P^r$，满足下列条件：$\Omega_r \subseteq \Omega_w$ *and* $Set_P(r) \subseteq Set_P(w)$，那么写 w 覆盖了读 r，$w \xrightarrow{f} r$。

证明： 由于$\Omega_r \subseteq \Omega_w$，程序段 P 执行读 r 时，写 w 也一定被执行到，由$Set_P(r) \subseteq Set_P(w)$可知，w 和 r 之间一定存在相关性，可能是流相关，也可能是反相关，但由题设：$w \in NaW_P^r$，w 和 r 之间存在的一定是流相关性。即写完全有能力向读 r 提供数据，因此：$w \xrightarrow{f} r$。

证毕。

如果$w \xrightarrow{f} r$，并且 w 输出相关于任何同 r 有流相关的写，w 是 r 的确切数据源。

定理 4.2（覆盖定理） 对于程序段 P 中的读 r，和写集 W：$W \subseteq NaW_P^r$，$W \xrightarrow{f} r$的充要条件是：读的状态-数据引用区域被写集的状态-数据引用区域之并所覆盖：$\Phi_P^r \subseteq \bigcup_{w \in W} \Phi_P^w$，即：$(\Omega_r \times Set_P(r)) \subseteq \bigcup_{w \in W} (\Omega_w \times Set_P(w))$

证明： **充分性**：$\Phi_P^r \subseteq \bigcup_{w \in W} \Phi_P^w$ 推出$W \xrightarrow{f} r$。

程序段 P 任何一次执行到读 r，对应的程序状态为：$s \in \Omega_r$

其对存储区域任何一点$X \in Set_P(r)$的引用必定满足下列条件：

$\exists w \in W$，使得$s \in \Omega_w$且$X \in Set_P(w)$。（否则与题设：$\Phi_P^r \subseteq \bigcup_{w \in W} \Phi_P^w$ 矛盾。）

由于$X \in Set_P(r) \wedge X \in Set_P(w)$，w 和 r 之间一定存在相关性，可能是流相关，也可能是反相关，但由题设：$w \in W \subseteq NaW_P^r$，w 和 r 之间存在的一定是流相关。 由写覆盖读的定义，充分性得证。

必要性：$W \xrightarrow{f} r$ 推出$\Phi_P^r \subseteq \bigcup_{w \in W} \Phi_P^w$。

反证法：设$(s, X) \in \Phi_P^r$但$(s, X) \notin \bigcup_{w \in W} \Phi_P^w$，则由覆盖的定义，当程序段 P 以 s 进入且引用点 X 时，W 无法覆盖 r，与题设$W \xrightarrow{f} r$不符。可见必要性成立。

证毕。

推论 1 如果程序段 P 中一对读写 r，w：$w \in NaW_P^r$，且满足下列条件：$(\Omega_r \times Set_P(r)) \subseteq (\Omega_w \times Set_P(w))$， 那么写 w 覆盖了读 r，$w \xrightarrow{f} r$。

推论 2 设 r 为程序段 P 中的读，如果 $\Phi_P^r \subseteq \bigcup_{w \in NaW_P^r} \Phi_P^w$，那么 r 为 P 中的非暴露读。

推论 3 对于程序段 P 中的读集 R，和写集 W：$W \xrightarrow{f} R$ 的条件是：

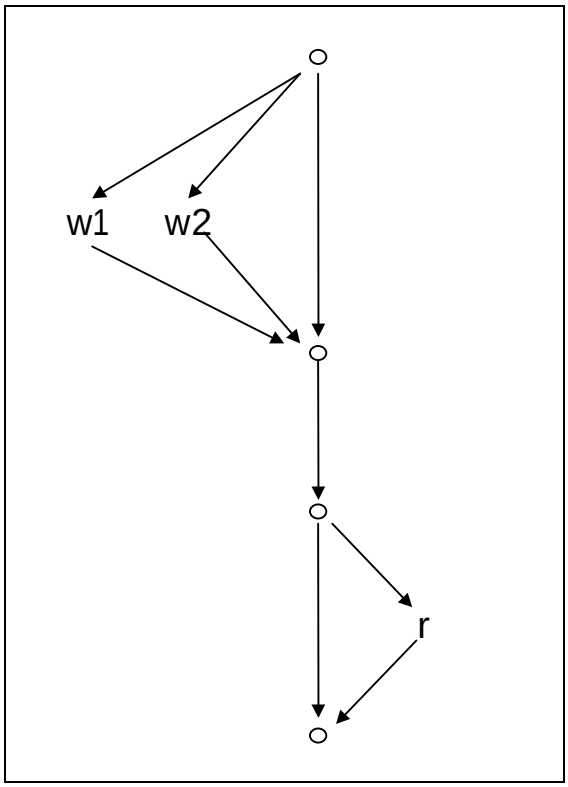
$\forall r \in R, \exists W^* \subseteq W \wedge W^* \subseteq NaW_P^r \wedge W^* \xrightarrow{f} r$，其中 $W^* \xrightarrow{f} r$ 的充要条件由覆盖定理确定：$\Phi_P^r \subseteq \bigcup_{w \in W^*} \Phi_P^w$

推论 1 是定理 4.1 的另外一种表示形式;推论 2 应用在数组私有化的判定过程中；推论 3 是覆盖定理的一种广义形式。

由覆盖定理可知，在数组数据流分析中可以利用 Ω 区域，Φ 区域的覆盖关系消解由条件分支语句导致的数据流不确定性：

流不确定性消解原理 对于程序段 P 中特定的读 r：$\Omega_{pr} \xrightarrow{r} Set_p(r)$，以及程序段 P 中所有可能为 r 提供数据源的写集 NaW_P^r，判定 NaW_P^r 的一个特定的子集 $W = \{w | \Omega_{pw} \xrightarrow{w} Set_p(w)\}$ 能否满足读引用 r 的数据引用需求：$W^* \xrightarrow{f} r$，可以通过判定下列条件来确定： $\Phi_P^r \subseteq \Phi_P(W)$。判定单独一个写能否覆盖读时，只需判定： $\Omega_r \subseteq \Omega_w$ *and* $Set_P(r) \subseteq Set_P(w)$。

流不确定性消解的非形式化示意:



我们用左边的程序流图来简单地示意流不确定性消解的大致思想。

图中的读 r 所读的存储范围在传统的数据流分析中由于无法为 w1，w2 所定义的存储范围所覆盖（这里假设 w1，w2 写的范围大于 r 的读范围）从而使得 r 的读范围成为 UE 集的一部分。然而事实上，可能 r 的读一定会在 w1，w2 处被定义从而使得 r 的读范围不是 UE 的一部分。我们通过导致 w1，w2 与 r 的逻辑关系的分析可以确定这一点。例如下面的程序:

```
w:      IF (C > 0)  X(1:10) =...
r:      IF (C > 2)  ....  = X(2:6)
```

X（2: 6）是非暴露的读，但传统的数据流分析得不出这样的精确结果，它

们会认为 r 的定值可能由 w 提供，也可能由其他地方的定义提供，从而是一个流不确定性的读写数据流关系。如果我们利用逻辑关系 （C > 2）蕴含 （C > 0），那么可知 X（2: 6）是非暴露的。我们介绍的区域覆盖方法事实上利用了（C > 0）区域覆盖（C > 2） 来消解这种流不确定性，得出 r 的读范围一定是 w 提供的定义。而在上面的程序流程图的示例中我们可以通过区域覆盖的判定方法确定 r 的读范围是否由我们关心的 w1 或 w2 提供，或由 w1 和 w2 联合提供。

拓广的区域覆盖技术可以增强数组私有化的判定。

如果把数组引用区域从单纯的数组区域推广到Φ区域，那么文献〖7〗中的相关-覆盖方法就可以推广到能够处理条件分支语句产生的，静态可消解的流不确定性。下面约定：BL 表示循环的次数，L_i 表示第 i 次循环迭代，L_v 表示任意的 L_i，$2 \le i \le BL$。并且约定读写都是针对当前被考察数组的。由于拓广是简单自然的，证明从略。

文献〖7〗中证明：程序段 P 的暴露集 $UE_P \subseteq \bigcup_{r \in R_P} (Set_P(r) - Set_P(NaW_P^r))$。经过推广到Φ区域后，$UE_P$ 不但蕴含了 P 中暴露的数据集，而且还携带了相应的逻辑条件：

定理 4.3 对于程序段 P，$UE_P \subseteq \bigcup_{r \in R_P} \left(\Phi_P(r) - \Phi_P\left(NaW_P^r\right)\right)$

定义 4.2 如果 $\forall 1 \le i \le BL-1; \Phi_{L_i}(W) \subseteq \Phi_{L_{i+1}}(W)$，那么我们称数组在循环 L 中是**写自覆盖的**。

拓广后的数组私有化判定准则相应地变为：

基本判定准则：循环 L 中，如果 $\forall k, 2 \le k \le BL: UE_{L_k} \cap \bigcup_{i=1}^{k-1} \Phi_{L_i}\left(W_{L_i}\right) = \varnothing$，那么该数组是可私有化的数组。

判定准则 1 循环 L 中，$r \in R_{Lv}$，如果 $\Phi_{Lv}(r) \subseteq \Phi_{Lv}\left(NaW_{Lv}^r\right)$，那么称 r 不妨碍私有化。如果任意 $r \in R_{Lv}$ 均不妨碍私有化，数组在循环 L 中是可私有化的。

判定准则 2 循环 L 中，$r \in R_{Lv}$，如果数组在 L 中写自覆盖：且 $\Phi_{Lv}\left(W_{Lv}\right) \subseteq \Phi_{Lv}\left(NaW_{Lv}^r\right)$，那么 r 不妨碍私有化。如果任意 $r \in R_{Lv}$ 均不妨碍私有化，数组在循环 L 中是可私有化的。

4.2 符号分析中的区域覆盖技术

符号分析对于精确的数据流分析和相关性测试十分重要。符号变量一般是标量，确定这些变量的数值可以使编译器分析出更加精确的数据流信息。

在计算函数模型下，条件分支语句定义的符号变量，其符号值可以表示为：

$$Symbol=\begin{cases}\Omega_{p1}\xrightarrow{w}ExpVal_1\\ \cdots\cdots\\ \Omega_{pn}\xrightarrow{w}ExpVal_n\\ \Omega_{other}\xrightarrow{w}ExpVal_o\end{cases}$$

由确定性执行原理可知，Ω_{p1}，…，Ω_{pn}，Ω_{other} 是 $Dom(P)$ 的一个划分。我们称这种表示法为区域函数法。

相应要解决的问题是：对于一个关键值 Symbol 的读引用 R：

$\Omega_{pr}\xrightarrow{r}Symbol$，　　以及 Symbol 可能的取值：

$Symbol=\left\{\Omega_{p1}\xrightarrow{w}ExpVal_1 \quad \cdots \quad \Omega_{pn}\xrightarrow{w}ExpVal_n \quad \Omega_{other}\xrightarrow{w}ExpVal_{other}\right\}$，

如何确定 R 实际读取的 Symbol 变量之值。

事实上，化解值不确定性和消解流不确定性原理上是一致的，可以得出下面的结论：

定理 4.4　程序段 P 中，对符号变量 Symbol 的读引用 r：$\Omega_{pr}\xrightarrow{r}Symbol$，r 能够读到 $ExpVal_i$ 的充要条件是：$\Omega_{pr}\cap\Omega_{pi}\neq\varnothing$；特别地，如果 $\Omega_{pr}\subseteq\Omega_{pi}$，则 r 读到的一定是 $ExpVal_i$；如果 $\Omega_{pr}\subseteq\bigcup_{i\in D}\Omega_{pi}$，则 R 读到的一定是 $\{ExpVal_i|i\in D\}$ 中的某个数值。

证明类似于流确定性的消解，略。

我们用 $Symbol(\Omega_{pr})$ 表示在状态条件 Ω_{pr} 下读符号变量 Symbol 所读取的变

量值表达式。

文献〖5〗中推广的 $\phi-function$ 表示法是区域函数表示法在 n = 1 时的情形。因此对文献〖5〗中表示法的运算可以推广到这种表示法上，例如求符号变量的极值：

$$max(Symbol) \le max(ExpVal_1, \cdots, ExpVal_n, ExpVal_o)$$

$$min(Symbol) \ge min(ExpVal_1, \cdots, ExpVal_n, ExpVal_o)$$

定义两个符号表达式是相容的，如果经过变量的反向替换后，一个表达式的非常数项是另一个表达式非常数项的子集。比较两个相容符号变量的分配律：

$Sym1 > Sym2 \Leftrightarrow \prod_{i=1}^{other}\left(Sym1\left(\Omega_i^{(2)}\right) > ExpVal_i^{(2)}\right)$，$\Omega_i^{(2)}$ 和 $ExpVal_i^{(2)}$ 表示第二个符号变量 Sym2 中第 i 个 $\Omega_{pi} \xrightarrow{w} ExpVal_i$ 相应的 Omega 区域和表达式。

当区域之间的覆盖关系过于复杂时，用近似的判定规则：

$$\left(min(Sym1) > max(Sym2)\right) \Rightarrow (Sym1 > Sym2)$$

区域函数表示法还可以方便地表示 **Index Array**。

例 4：

```
[1]     DO   J = 1，  JMAX
[2]          JPLUS（J）= J +  ！
[3]     ENDDO
[4]     JPLUS（JMAX）= 1
```

可以表为：$A(K) = \left\{1 \le K \le JMAX - 1 \xrightarrow{w} K + 1 \quad ; \quad K = JMAX \xrightarrow{w} 1\right\}$

4.3 区域覆盖技术的应用范例

本例出自 Perfect Benchmarks 的 MDG。子程序 INTERF 中的循环 DO 1000 占 MDG 总运行时间的 90%以上，由于传统的数据流方法无法确定数组 RL 是否可以私有化，使得该循环没有被并行执行。本文对该循环有保持问题本质的简化。

在这个例子里，我们希望确定数组 RL 是否对于外层循环是可私有化。因此我们考虑的当前程序段 P 是：Line 3 to Line 22。

由 3.3 节中的介绍可知：第[3]行到第[9]行定义的标量 KC 是程序状态空间 $Dom(P)$ 的区域索引值（**Region Index**）。由于 KC 的值出现在[10]和[16]行里，KC 是典型的 IF 语句下的符号变量定义。需要推测其值的情况，参见 3.3 的例 2。

```
[1]     DO   I = 1, NMOL - 1, 1
[2]         DO   J = I+1, NMOL, 1
[3]             KC = 0
[4]             DO K = 1, 9, 1
[5]                 RS(K) = XL(K)*XL(K)+YL(K)*YL(K)+ZL(K)*ZL(K)
[6]                 IF (RS(K) .GT. CUT2) THEN
[7]                     KC = KC + 1
[8]                 ENDIF
[9]             ENDDO
[10]            IF (KC .NE. 9) THEN
[11]                DO   K = 2, 5, 1
[12]                    IF (RS(4+K) .LE. CUT2) THEN
[13]                        RL(4+K) = SQRT(RS(4+K))
[14]                    ENDIF
[15]                ENDDO
[16]                IF (KC .EQ. 0) THEN
[17]                    DO K = 11, 14, 1
[18]                        FTEMP = AB2 * EXP(-B2 * RL(K - 5 )) / RL((-5)+K)
[19]                        FF((-5)+K) = FF((-5)+K)+FTEMP
[20]                    ENDDO
[21]                ENDIF
[22]            ENDIF
[23]        ENDDO
[24]    ENDDO
```

[11]行到[15]行中的 IF 语句产生了程序段 P 中的一个条件写：**RL(4+K)** = *SQRT(RS(4+K))*。RL(6:9)的定义和 RS(6:9)满足(RS(4+K) .LE. CUT2)的情况是一致的。当条件全部成立时：

$\Omega_w = \bigcap_{K=6}^{9}\{RS(K) \le CUT2\}$，数组区域 RL(6:9)也全部被定义，条件全部不成立时，数组区域 RL(6:9)也全部没有被定义，其他情况类似。

[16]行到[21]行中的 IF 语句产生了程序段 P 中的条件读引用区域：RL(6:9)。该条件读的条件谓词：KC = 0，可知 RL(6:9)的读 Omega 区域是 KC = 0 所索引的区域：$\Omega_r = \bigcap_{K=1}^{9}\{RS(K) \le CUT2\}$。该条件读的数组引用区域:

$Set_P(r) = RL(6{:}9)$被[11]到[15]的条件写的全写区域$Set_P(w) = RL(6{:}9)$所覆盖：$Set_P(r) = Set_P(w)$，且$\Omega_r \subset \Omega_w$：

$$\left(\Omega_r = \bigcap_{K=1}^{9}\{RS(K) \le CUT2\}\right) \subset \left(\Omega_w = \bigcap_{K=6}^{9}\{RS(K) \le CUT2\}\right)$$

由前面提出的流不确定性消解原理可知，此处条件读的确切数据源是[11]行到[15]行的条件写，因此不会产生跨循环的流相关，数组 RL 可以私有化。

在 **SGI Challenge 4L**（4 处理器）**SMP** 计算机上的实测表明：本例的分析变换使得 **MDG** 的加速比从 1.0 提高到了 3.7。

5. 相关工作比较

由于逻辑条件在程序计算语义上的复杂性，传统数据流分析方法一般都是采取忽略其逻辑语义的简化处理方法来近似地分析数据流关系[15]。既有的并行化编译系统，例如 *StanfordUniversity* 研究开发的 **SUIF**[11]，**复旦大学**并行处理研究所研究开发的 **AFT**[1] 在进行数据流分析时一般忽略条件分支语句的逻辑条件。在传统的科学计算程序中，程序结构的静态特性相当显著，控制流比较简单。影响程序性能的关键代码段中，只有为数不多的条件分支语句，而且语义一般是值不确定性，上述的忽略逻辑语义式的近似方法对数据流分析的精确度影响不大。但是随着科学计算程序的日益复杂化，以及并行化编译器潜在应用范围的日益扩大，人们开始注意到如何充分、有效地利用条件分支语句提供的程序逻辑语义。

Parafrase-2 中的 **Join-function** 以及 **Michael J. Wolfe** 提出的 ϕ – *function*，事实上并没有利用条件分支语句的逻辑条件。文献〖4〗〖5〗中对 ϕ – *function* 进行了扩充，使之能够携带逻辑条件，并在 *UIUC* 开发的 **Polaris**[12] 并行化编译器上应用于符号分析等问题。文献〖6〗中提出的 **GAR：Guarded Array Region**，是对数组区域在概念上进行扩充的一个尝试。但是在 **GAR** 中，作为 **Guard** 的逻辑谓词，其运算采用逻辑推理系统；而数组区域表示为数组数据空间中的凸区域，运算是集合运算系统。两个不同的运算系统运用在同一个对象上，使得 **GAR** 在理论和实现上都相当复杂。此外，完备的逻辑推理系统集成到实用的编译器中，在目前还是令人怀疑的。

本文在计算函数模型的框架下，把逻辑条件的语义运算转化为空间区域的集合运算，其形式与数组区域的运算是一致的。因此Φ区域比 **GAR** 是一个更加自然的对数组区域的概念拓广。Φ区域的运算无需再实现一个新的运算系统，而只需在数组区域运算系统的基础上予以增强和扩充即可。通过本文的示例可知，Φ区域覆盖法简洁有效。此外，把符号变量的值表示为区域函数的形式，并利用区域覆盖的关系进行符号分析，其分析能力不亚于〖5〗中的方法。

6. 结论与展望

任何试图利用条件分支语句的逻辑语义来得到精确数据流信息的方法，本质上都是在不同的数学模型下，对逻辑条件进行表示和运算。 在计算函数模型的框架下，条件分支语句的语义可以自然简洁地表示。其形式：$\Omega_p \xrightarrow{x} Set_p(x)$或$\Phi$区域，是数组引用区域在概念上的一个拓广，它同时包含数组引用的数据引用区域和引用的逻辑条件信息；而条件分支语句语义作用下的相应的数据流信息和符号变量的取值信息也可以通过读写引用的Ω区域或Φ区域的区域运算和覆盖关系的判定得以明确。这里的表示和计算方式与目前数组数据流分析采用的比较成熟的区域覆盖技术基本一致，因此实现起来只需对既有系统进行扩充，而无需引入一套推理系统。运用本文提出的不确定性消解原理，通过手工形式化变换 **PERFECT Club Benchmarks，SPEC95fp** 等测试程序包里的实用测试程序表明：这里提出的方法简洁有效，可以在相对较小的代价下获得更加精确的数据流信息。

随着计算机科学和信息技术的飞速发展，计算与网络通信在人们日常的工作生活中，在人类长远的生存发展中日益扮演起愈发重要的角色。科学计算已经成为继传统的理论推导和科学试验之后的第三种科研手段；商务运作方式的自动化、信息化；工业制造、规划、生产的自动化；**Internet** 上客户机与服务器模式的兴起…… 所有这些都在不断向现有的计算模式提出新的需求和挑战。 追求高性能高可靠性的并行与分布式计算（Parallel & Distributed Computing）、编程一次跑遍天下的 **Java** 移动式计算（Mobile Computing）、由整个 **Internet** 上所有计算资源、信息资源和服务资源充作强大后盾的网络计算机…… 新概念新思路在不断地推陈出新。信息革命的浪潮在席卷全球的同时，也向弄潮儿们发出了连珠炮般艰深复杂的质疑。 用并行计算机解决高性能科学计算和其他重要的计算任务，既是其中非常重要、持久的战略性需求和挑战之一。 如何为现有的并行计算机产生简洁高效的并行计算软件，并由此引导下一代新型并行计算机的设计与实现，已经成为计算机科学迫切需要解决的具有战略意义的关键问题。

专业程序员的并行程序设计（Parallel Programming）和程序的自动并行化编译（Automatic Parallelizing Compilation）是产生并行程序的两条主要途径。并行程序设计建立在并行程序设计语言或并行编程环境的基础上，主要依赖程序员的人脑智力。对于一个对目标机器相当熟悉的程序员来说，这种方式可以充分发挥人类智能当中的许多潜在优势，比如：机智的推理和敏锐的直觉等。然而这种方式也往往需要程序员作出大量人类并不擅长的重复、机械的运算任务。产生的程序可能会充分发挥机器的潜在性能，也可能低效、甚至不正确。程序的自动并行化编译可以把许多机械重复的计算，运作的完美无缺。但同时也并不令人见怪的是，这种方式无法利用许多人类智慧的许多知识背景，缺乏逻辑推理等基本技能。尽管两种并行程序的开发方式仍然在各自的领域里不断前进着。而一种折衷的方式已然在悄然实施，这就是上述两种方式的有机结合。

在人工智能尚未取得真正令人振奋的成果之前，要把一个计算量巨大的计算任务完美高效地映射到现有并行计算机的计算模式上去，使之完全契合物理实现的运作方式以换取高性能和高可靠性，最有希望的方式可能就是人机协作、共同开发的模式了。但设计一个怎样的接口（**Interface**）才能使得人尽其长，机尽其用。这似乎仍然是一个由于种种技术问题的存在，而依然没有被斟酌清楚的问题。其涉及范围之广，可以从理论上关于计算模型的认识直至物理器件实现技术的发展趋势；从人脑的思维方式到科学的人机交互模式，直至未来的软件设计与构造技术；从静态可以确定的类属计算结构信息直至动态运算的实例事件流程和统计分布信息；从程序设计方法学，程序正确性验证直至并行程序动态运行的监控和调试……这是一个复杂的、具有重大现实意义的系统化课题。即充满了艰巨的挑战，也同时蕴含着重大突破的机遇。

参考文献

1 Zhu Chuan-Qi, Zang Binyu, Chen Tong. ***An Automatic Parallelizer***. Journal of Software, 1996,7(3):180-186. 朱传琪，臧斌宇，陈彤。程序自动并行化系统 AFT。软件学报，1996,7(3):180-186.

2 Mary W Hall, B.R.Murphy, S.P.Amarasinghe, S.W.Liao, M.S.Lam ***Interprocedural Analysis for Parallelization***. In: C-H. Huang et al eds. Proceedings of the Eighth International Workshop on Languages & Compilers for Parallel Computing. Columbus, Ohio: Springer, 1995. pp 5.1-5.14

3 Beatrice Creusillet, Francois Irigoin: ***Interprocedural Array region Analysis*** In: C-H. Huang et al eds. Proceedings of the Eighth International Workshop on Languages and Compilers for Parallel Computing. Columbus, Ohio: Springer, 1995 pp 4.1-4.15

4 Peng Tu . David Padua : ***Automatic Array Privatization*** In: Proceedings of Languages and Compilers for Parallel computing , Pages 500-521, August 1993.

5 Peng Tu. David Padua : ***Gated SSA-Based Demand-Driven Symbolic Analysis for Parallelizing Compilers.*** In: Michael Wolfe, Denis Nicole et al eds. Conference Proceedings of the 1995 International Conference on Supercomputing, Bacelona, Spain: ACM press. July 1995 pp 414-423.

6 Trung Nguyen, Junjie Gu, Zhiyuan Li : ***An Interprocedural Parallelizing Compiler and Its Support for Memory Hierarchy Research***. In: C-H. Huang et al Eds. Proceedings of the Eighth International Workshop on Languages and Compilers for Parallel Computing. Columbus, Ohio: Springer, 1995 pp 7.1 - 7.15

7 Chen,Tong., Zang,Binyu., Zhu,Chuan-Qi., "***A new method for array privatization*** ", In Proceedings of High Performance Computing and communication'94, pp 43-50, Aug. 1994

8 Hu Shiliang, Zang Binyu, Zhu Chuan-Qi. ***Enhancing Dataflow Analysis with Computation Function Model***. Journal of Software. Adopted, Serial number: 3071 胡世亮，臧斌宇，朱传琪. 用计算函数模型增强数据流分析. 软件学报，已录用，编号：3071.

9 Hu Shiliang, Zang Binyu, Ling Bing, Zhu Chuan-Qi. ***The Region Coverage Method in Dataflow Analysis***. Journal of Software. Adopted, Serial number: 3070 胡世亮，臧斌宇，凌冰，朱传琪. 数据流分析中的区域覆盖技术. 软件 学报，已录用，编号：3070.

10 William Pugh, ***A Practical Algorithm for Exact Dependence Analysis***, Communication of the ACM Aug.1992/Vol.35, No.8 pp 102-114

11 R.Wilson, R.French, C.Wilson, S.Amarasinghe, J.Anderson, S.Tjian, S.-W.Liao, C.-W Tseng, M.Hall, M.Lam, and J.Hennessy, "***SUIF: An Infrastructure for Research on Parallelizing and Optimizing Compilers***", ACM SIGPLAN Notices, Vol.29, No.12, December 1994. Pp31-37.

12 B. Blume, R. Eigenmann, K. Faigin, J. Grout, J. Hoeflinger, D. Padua, P. Petersen, B. Pottenger, L. Raughwerger, P. Tu, S. Weatherford, "***Polaris: The Next Generation in Parallelizing Compilers***", In: Proceedings of the 7th International Workshop on Languages & Compilers for Parallel Computing, Ithaca, New York: Springer-verlag, 1994, 141-154.

13 Utpal Banerjee: ***Dependence Analysis for Supercomputing***. Kluwer Academic Publishers, Norwell, Mass.,1988

14 Utpal Banerjee: ***Unimodular Transformations of Double Loops*.**

15 Zhiyuan Li . ***Array Privatization for parallel execution of loops***. In Proc. of ICS'92 pages 313-322, 1992.

16 Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman: ***Compilers Principles, Techniques, and Tools*** Addison-Wesley Publishing Caompany ISBN 0-201-10088-6

17 Michael J. Wolfe. ***High Performance Compilers for Parallel Computing***. 1996 Addson-Wesly Publishing Company, Inc.

18 K. D. Cooper, Marry. W. Hall, and Ken Kennedy. ***A Methodology for Procedure Cloning*** Comput. Lang. Vol, 19, No. 2, pp 105-117, 1993

19 Ken Kennedy, Kathryn S. McKinley, ***Optimizing for Parallelism and Data Locality***, Proceedings of the 1992 International Conference on Supercomputing, pp.323-334, July 1992.

20 Saman Amarasinghe, Jennifer M. Anderson, Christopher S. .Wilson, Shih-Wei Liao, Brian R. Murphy, Robert S. French, Monica S. Lam and Marry W. Hall, ***Multiprocessors from a software perspective***, IEEE Micro 1996, 52-61.

致 谢

感谢导师朱传琪教授三年来的言传身教和谆谆教诲。老师在学术上造诣高深，在学风上严谨求实。多年来坚定执着地带领一批青年学子在当今并行处理的国际最前沿开拓探索，那不倦不懈的求索，教育感召了一届又一届的莘莘学子。

感谢臧斌宇副教授及其他几位老师的指点和帮助。臧老师的热诚、勤恳、博学为复旦大学并行处理研究所带来了勃勃生机。这股朝气也曾感染激励了我在这里的学习和工作。 感谢张瑜，凌冰等各位师兄弟姐妹，感谢他们的热诚帮助和珍贵友谊。

感谢含辛茹苦的母亲，以及所有那些曾经帮助过我的人，没有他们的恩泽，我走不到今天，想不到明天；感谢从小领受的教育，纯朴上进而又片面； 感谢自小长大的特殊环境，锻造磨炼出了一种偏执的激情。 谢远方那片深邃湛蓝的天空，它不慎赋予了我生命，使我有幸游历如此美好的人世； 谢脚下这块蛮荒无际的大地，它必将我渺小的身躯连同涌动的思潮吞噬、埋葬。