1. **数据结构描述**
   1. **有限状态机**
      1. **有限状态机的规约**

Telnet协议规定了客户如何把字符传给远程服务，以及客户如何显示该远程服务所返回的数据。在连接上所进行的大多数通信都是由各个数据字符所组成的。在客户端，这些数据字符来自用户从键盘键入的数据；在服务器端，它们来自远程会话所产生的输出。除了数据字符以外，Telnet还允许客户和服务器交换控制信息。

大多数Telnet的实现都用有限状态机(Finite state machine，FSM)来说明命令序列的确切语法以及对命令序列的解释。作为规约工具，有限状态机提供了对协议的精确描述。它确切地说明了发送方如何在数据流中嵌入命令序列，同时，它也确切说明了接收方如何解释这种序列。更为重要的是，有限状态机可以直接转换成遵从该协议的程序。因此，可以验证最终的程序是否符合协议规约。

* + 1. **有限状态机的实现**

由于构造有限状态机的有效实现是可能的，也由于这种机器能够很容易地描述面向字符的协议，所以我们的示例代码用了三个独立的有限状态机：一个状态机控制客户如何对来自键盘的字符进行响应，另一个控制客户如何处理在TCP连接上到达的来自服务器的字符，第三个状态机处理选项子协商的细节。所有这三个有限状态机都使用同样的数据结构，这使操作这些数据结构的某些过程可以被共享。

有限状态机的状态转移图使用了传统的表示方法。从一个状态转换到另一状态都有—个形如α/β的标志．其中α表示引起该转移的特定输入字符，β表示发生该转移所完成的动作。从状态X转移到状态Y上如果有标志α/β，表示如果在状态X时字符α到达，执行动作β，然后转换到状态Y。

为了能高效地处理，我们的实现把有限状态机的转移编码成一个如图4-1所示的转移矩阵。

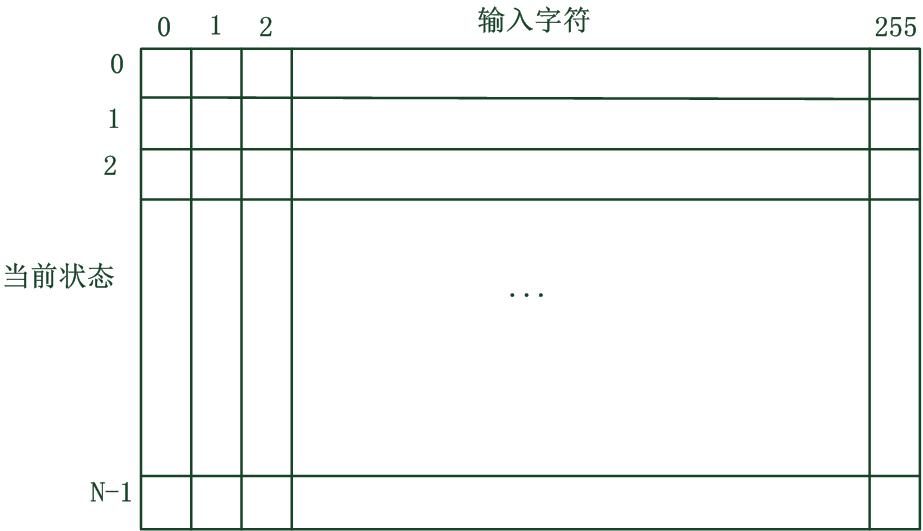


图4-1 实现有限状态机的转移矩阵

说明：

每行对应一个状态而每列对应一个可能的输入字符，在运行时，客户端维护记录当前状态的变量。当有一个字符到达时，客户端用当前状态变量及该字符的数字值在转移矩阵中进行索引。当取得一个索引值，就可以执行该索引值指定的操作并转移到下一个状态。

* + 1. **压缩的有限状态机表示**

用c代码来初始化一个大矩阵将是非常繁琐的。而且，如果转移矩阵的每个元素都要包含关于要采取的动作以及下一状态的全部信息，该矩阵将消耗很大的内存空间。为了使转移矩阵更小也为了使初始化的工作更为容易，我们的代码使用了有限状态机的压缩表示。

就本质来讲，所选的数据结构允许程序员构造一种压缩的数据结构，用它来表示有限状态机，并且让程序在运行时构造相关的转移矩阵。文件fsm\_struct.h中包含了在压缩表示中所用的结构fsm\_trans的声明：

/\* 有限状态机转移矩阵复用结构 \*/

struct fsm\_trans {

unsigned char ft\_state; /\* 当前状态 \*/

short ft\_char; /\* 输入字符 \*/

unsigned char ft\_next; /\* 下一个状态 \*/

int (\*ft\_action)(FILE \*sfp, FILE \*tfp, int c); /\* 对应的操作 \*/

};

压缩的有限状态机表示由一维的fsm\_trans结构数组组成。每个元索指定了一种转移。ft\_state字段指明转移开始时的有限状态机状态，ft\_next 字段指明了下一个状态，ft\_char字段指定了引起转移的输入字符(或者用TC\_ANY来表示没有明确声明的其他所有字符)，ft\_action字段给出了要调用的过程的地址，用来完成与状态转移相关的动作。

* + 1. **在运行时压缩的有限状态机的表示**

本客户端软件并没有把压缩表示中的所有信息都复制到转移矩阵中，而是采用了这样的方法，它不改变压缩表示的内容并用它来维护转移信息。为此，软件在转移矩阵的每个元素中存放一个整数。该整数是压缩表示数组中某一项的索引，而该项与所要进行的转移相对应。图4-2展示了该数据结构：

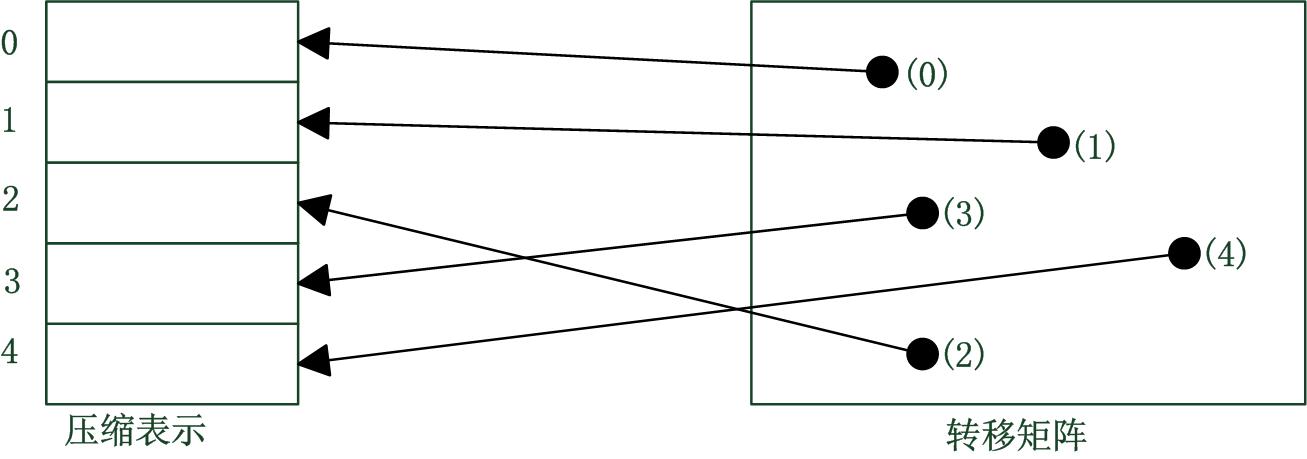


图4-2 运行时有限状态机数据结构

说明：

转移矩阵中的每一项都包含一个指向压缩表示中某元素的索引。运行时，根据有限状态机的当前状态和输入字符查找转移矩阵，得到一个整数索引值，该索引值就是压缩的转移矩阵的数组下标，然后根据该下标进行压缩的转移矩阵中指定的操作，并转移到下一个状态。

* 1. **针对来自服务器数据的主有限状态机**

图4-3展示了一个说明Telnet协议的主有限状态机，包括与上面所述的选项协商相对应的状态。我们可以把该状态机看作是指明了客户如何处理从服务器收到的字符序列。

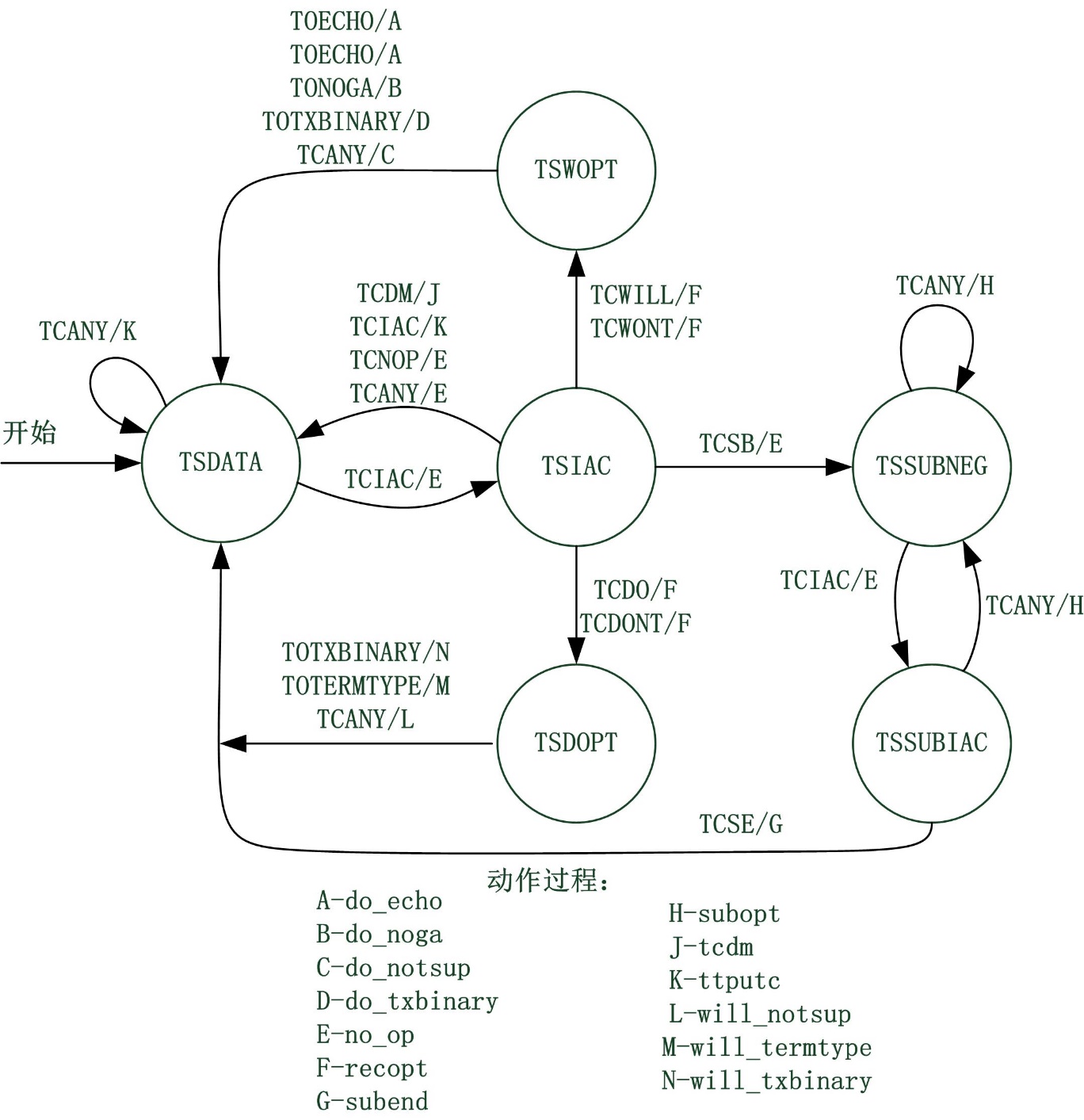


图4-3主有限状态机

说明：

它描述Telnet如何在数据中编码命令序列。

当客户端程序开始执行的时候，它从标有TSDATA的状态开始。状态TSDATA对应着这种情况：客户希望接收正常的字符并把它们发送到用户的显示器上(即客户还没有开始读命令序列)。例如，如果字符TCANY到达，则客户保持在状态TSDATA，并执行动作K(即客户调用过程ttputc，把字符显示在用户的终端屏幕上，然后沿着回路返回到同一状态。

如果在有限状态机处于状态TSDATA时字符TCIAC到达．则客户转移到状态TSIAC，并执行图中所标的动作E。该说明指出动作E对应着“无操作”。一旦客户转到状态TSIAC，它便开始解释命令序列了；如果跟随TCIAC之后的字符是一个动词(如TCDO)，客户将转移到某个处理选项状态。Telnet的有限状态机只需要六个状态，因为协议的解释只依赖于字符到达的那段简短时间。

例如，在TCIAC字符之后，服务器可能发送某个选项请求或响应：TCDO、TCDONT、TCWILL、TCWONT，或者可能发送选项子协商请求。选项子协商允许发送方在选项中包含可变长度的字符串(例如，客户用于向服务器传送终端类型的选项可用于协商。这样，它可以发送含有该终端名的编码的字符串)。尽管子协商允许可变长度的命令序列，但是有限状态机仅需要2个状态来处理它，因为一个2字符的序列就终止子协商了。当客户先遇到子协商请求时，它便进入状态TSSUBNEG；当接收到字符TCIAC时，它便进人状态TSSUBIAC如果字符TCSE紧跟其后，它便退出子协商。如果有任何其他的2字符序列出现，有限状态机仍停留在状态TSSUBNEG。

* 1. **套接字输出有限状态机**

图11显示了套接字输出有限状态机。该有限状态机仅有三种状态。多数操作是在单个转移上处理，该转移将客户从表示本地处理的状态带到表示(普通)远程系统交互的状态。

套接字输出有限状态机从状态KSREMOTE开始，它使客户把从键盘来的每个字符都发送到远程服务器上去。当用户键人键盘退出(Keyboard escape)键时，客户进人状态KSLOCAL．并等待一个击键动作。紧跟Escape的大多数击键都没有意义，但也有几个会引起客户动作，接着便返回到状态KSREMOTE。

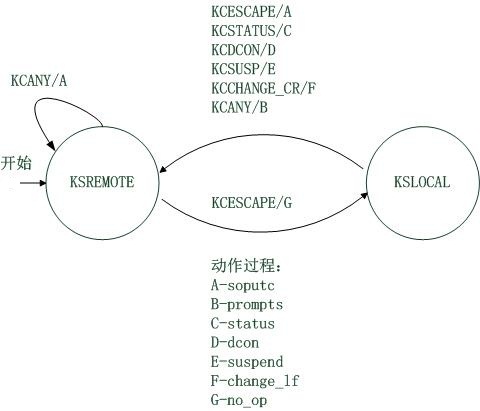


图4-4 套接字输出有限状态机

说明：

用于定义对每个用户输入字符所做的动作。大多数字符被客户发送到远程服务器上去。但是，该设计允许用户从数据连接中退出，然后与本地客户程序交互。

* 1. **选项子协商有限状态机**

图4-5显示了客户使用的第三种有限状态机。它处理在选项子协商过程中到达的字符序列。因为它只识别一种可能的选项子协商(终端类型)，所以该有限状态机只需要三个状态。

理解子协商的最简单的方法就是想像它描述了理论有限状态机中状态TSSUBNEG的内部结构。当主有限状态机运行在状态TSSUBNEG时，它调用过程subopt处理每个传入字符。subopt运行子协商有限状态机。子协商有限状态机依赖于选项立即做出决定。如果它发现终端类型的子协商，则机器移到状态SS\_TERMTYPE。否则，它直接变到状态SS\_END，并忽略剩下的子协商字符串。

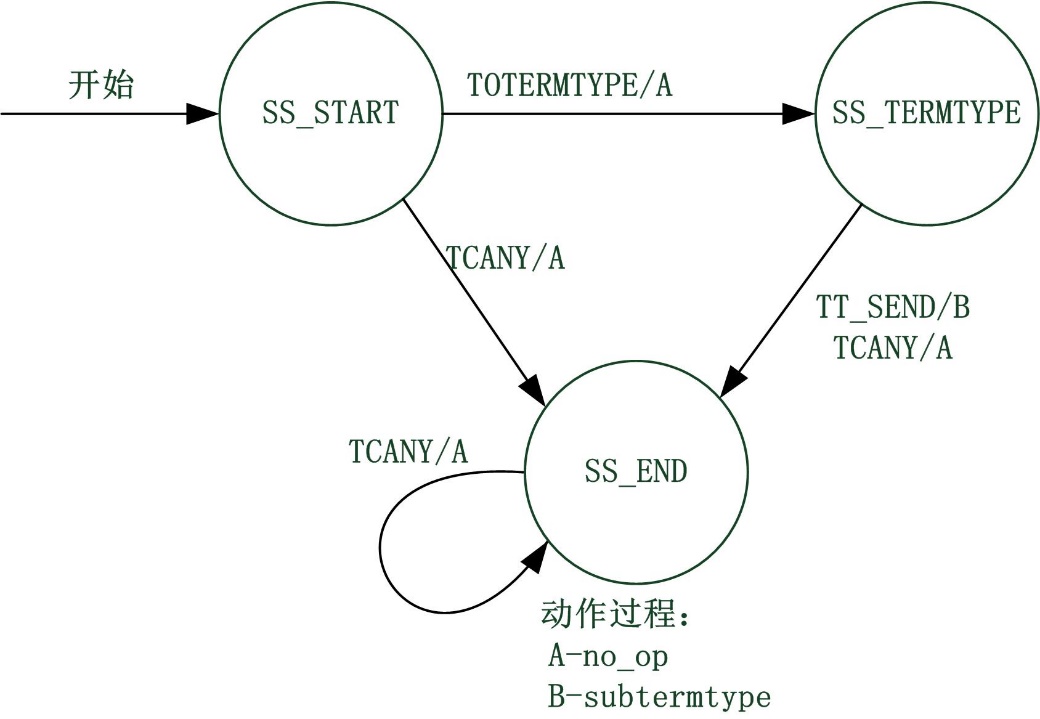


图4-5 选项子协商所用的简单有限状态机

说明：

每当客户完成一次选项子协商后，都要把该状态机重新初始化。

一旦有限状态机到达状态SS\_TERMTYPE，它就检测子协商动词。如果动词是TT\_SEND，它便调用subtermtype并忽略其他子协商。

次选项子协商后，都要把该状态机重新初始化。

一旦有限状态机到达状态SS\_TERMTYPE，它就检测子协商动词。如果动词是TT\_SEND，它便调用subtermtype并忽略其他子协商。