Глава 7

Оптимизация программ

Реализация ГП с помощью конструкции `switch` при всех достоинствах, перечисленных выше, обычно не минимальна по памяти и не обладает максимальным быстродействием.

Если построенная программа удовлетворяет ограничениям по указанным показателям, то проблем не возникает. В противном же случае, прежде чем отказаться от программирования с использованием указанной конструкции, должны быть выполнены оптимизирующие преобразования.

Если не рассматривать глобальные преобразования программы, например замену одного ГП взаимосвязанной совокупностью ГП, то могут быть предложены следующие три класса преобразований:

— раздельная оптимизация булевых формул в каждом операторе `if` (например, применение операций | и & вместо || и &&);

— совместная оптимизация булевых формул, входящих в различные операторы `if`, с последующей передачей вычисленных значений этим операторам;

— использование приоритетов и умолчаний.

Первый и второй классы преобразований подробно рассмотрены в Приложениях 1 и 2. Поэтому изложим только преобразования третьего класса.

Первоначально преобразования этого класса будем изучать на примере одной вершины. В общем случае существуют три варианта ее исходного задания:

— противоречива и не полна;

— не противоречива, но не полна;

— не противоречива и полна.

При этом необходимо отметить, что, несмотря на то что в настоящей работе предлагается в качестве спецификации применять только полные и непротиворечивые ГП, на практике, например, ввиду их громоздкости могут использоваться и другие разновидности графов, например ГП с указанием приоритетов на дугах, исходящих из рассматриваемой вершины. При этом ортогональность и полнота обеспечиваются на программном уровне.

Рассмотрим первый вариант исходного задания на примере фрагмента ГП (рис. 7.1). Для устранения противоречивости в нем выберем приоритеты (например, предположим, что при $x_1 = x_2 = 1$ переход должен

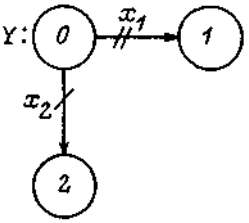

Рис. 7.1

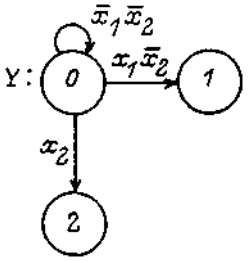

Рис. 7.2

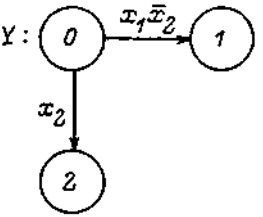

Рис. 7.3

осуществляться по x_2), а для устранения неполноты будем считать, что все неопределенные входные наборы сохраняют нулевое состояние автомата.

Размещая проверки значений переменных с большим приоритетом раньше проверок остальных переменных, построим фрагмент программы для нулевой вершины, все операторы if в которой имеют одинаковую структуру:

```
case 0: if(x2)     {Y=2; break;}
        if(x1)     {Y=1; break;}
        break;
```

В этом случае сохранение состояния автомата осуществляется последним оператором `break`.

Если нарушить однотипность операторов if, то программа может быть упрощена:

```
case 0: if(x2)   {Y=2; break;}
        if(x1)    Y=1;
        break;
```

Первый оператор `break` может быть исключен, если использовать конструкцию `else`:

```
case 0: if(x2)     Y=2;
        else
        if(x1)     Y=1;
        break;
```

Эта конструкция в свою очередь может быть исключена, если выбрать порядок записи операторов if по приоритетам, противоположный принятому ранее — чем выше приоритет, тем ниже размещается оператор:

```
case 0: if(x1)    Y=1;
        if(x2)    Y=2;
        break;
```

В силу того что спецификация в форме графа переходов может быть противоречива и не полна, а в правильно построенной программе по этим вопросам уже приняты решения, то можно утверждать, что каждая из четырех программ, рассмотренных выше, реализует для нулевой вершины полный и непротиворечивый фрагмент ГП (рис. 7.2).

Переходя ко второму варианту исходного задания вершин ГП, рассмотрим рис. 7.3. Взаимная ортогональность булевых формул в операторах if делает возможным произвольный порядок их расположения в конструкции case. Для обеспечения лучшей читаемости фрагмента программы расположим эти операторы в соответствии с возрастанием номеров следующих состояний:

```
case 0: if(x1&x̄2) {Y=1; break;}
        if(x2)    {Y=2; break;}
        break;
```

Эта программа может быть упрощена:

```
case 0: if(x1&x̄2) {Y=1; break;}
        if(x2)     Y=2;
        break;
```

Осуществим дальнейшее ее упрощение:

```
case 0: if(x1&x̄2) Y=1;
        if(x2)    Y=2;
        break;
```

Эти программы, так же как и предыдущие, реализуют фрагмент ГП (рис. 7.2).

Пусть исходная спецификация полна и непротиворечива (третий вариант) (рис. 7.4). Построим фрагмент программы для нулевой вершины графа переходов без учета формирования в ней значений выходных переменных, обладающий наименьшим разнообразием строк из всех рассмотренных:

```
case 0: if(x̄1&x̄2) {Y=0; break;}
        if(x1&x̄2) {Y=1; break;}
        if(x2)    {Y=2; break;}
```

Эта программа может быть записана за счет применения умолчаний и приоритетов в одной из следующих четырех форм:

```
case 0: if(x1&x̄2) {Y=1; break;}
        if(x2)    {Y=2; break;}
        break;

case 0: if(x1&x̄2) {Y=1; break;}
        if(x2)     Y=2;
        break;

case 0: if(x1&x̄2) Y=1;
        if(x2)    Y=2;
        break;

case 0: if(x1)    Y=1;
        if(x2)    Y=2;
        break;
```

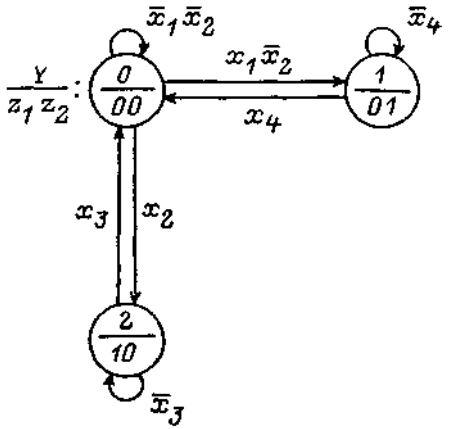

Рис. 7.4

Последний оператор b r e a k в этих программах располагается отдельной строкой с целью мнемонического отображения одной из исходящих из рассматриваемой вершины дуг — петли.

Рассматривая не одну вершину, а граф переходов (рис. 7.4) в целом, можно утверждать, что при использовании конструкции s w i t c h и обобщенной программной модели автомата Мура первого рода (разд. 5.1.2) он может быть реализован целым спектром программ, от наиболее сложной, но полностью изоморфной ГП:

```
        switch (Y) {
case 0: z1=0; z2=0;
          if(x̄1&x̄2)    {Y=0; break;}
          if(x1&x̄2)    {Y=1; break;}
          if(x2)       {Y=2; break;}
case 1: z1=0; z2=1;
          if(x4)       {Y=0; break;}
          if(x̄4)       {Y=1; break;}
case 2: z1=1; z2=0;
          if(x3)       {Y=0; break;}
          if(x̄3)       {Y=2; break;}
                    }
```

до наиболее простой, в которой изоморфизм уменьшен:

```
        switch (Y) {
case 0:     z1=0;     z2=0;
          if(x1) Y=1;
          if(x2) Y=2;
          break;
case 1: /*  z1=0; */ z2=1;
          if(x4) Y=0;
          break;
case 2:     z1=1; /* z2=0; */
          if(x3) Y=0;
          break;
                    }.
```

В приведенных программах новые значения выходных переменных формируются через цикл после получения нового состояния. В тех случаях, когда это недопустимо, например из-за большого времени цикла, необходимо перейти от обобщенной программной модели автомата Мура первого рода к программной модели автомата Мура второго рода (разд. 5.1.2):

```
        switch (Y) {
case 0: if(x1)    Y=1;
        if(x2)    Y=2;
        break;
case 1: if(x4)    Y=0;
        break;
case 2: if(x3)    Y=0;
        break;
                  }
        switch (Y)  {
case 0:    z1=0;    z2=0;     break;
case 1: /* z1=0; */ z2=1;     break;
case 2:    z1=1; /* z2=0; */  break;
                  }
```

или к обобщенной программной модели автомата Мура второго рода:

```
        switch (Y) {
case 0: if(x1)  {Y=1; /* z1=0; */ z2=1;}
        if(x2)  {Y=2;    z1=1; /* z2=0; */}
        break;
case 1: if(x4)  {Y=0; /* z1=0; */ z2=0;}
        break;
case 2: if(x3)  {Y=0;    z1=0; /* z2=0; */}
        break;
                  }.
```

Последняя модель, несмотря на меньшую сложность, менее естественна для описания автоматов Мура, так как в i-й вершине записывают значения выходов не в этой вершине, как это имеет место в предшествующей модели, а в следующей. Это приводит в общем случае (разд. 5.1.2) к необходимости повторения строк значений выходов для состояний, в которые имеются переходы из нескольких других состояний, в то время как для автоматов этого класса значения выходов в каждом состоянии желательно указывать однократно. Повторение указанных строк нарушает изоморфизм текста программы и графа переходов, по которому она строится.

Для упрощения чтения приведенных программ сохраняемые значения выходных переменных записаны как комментарии непосредственно на месте соответствующей переменной. Это позволяет уменьшить объем программного кода при сохранении понятности программы. Такой подход к исключению умолчаний может быть назван «умолчания без умолчаний».

При этом отметим, что указанные комментарии не могут быть записаны в случае, если отсутствующая переменная принимает различные значения в зависимости от предыстории.

Изложенный подход к исключению умолчаний концептуально совпадает с общей тенденцией повышения надежности при написании программ с помощью языков высокого уровня за счет уменьшения числа умолчаний, что характерно, например, для такого языка, как «Алгол-68».

Однако исключение флагов и умолчаний значений выходных переменных при построении управляющих программ даже в рамках языка «Алгол-68» в отличие от подхода, предлагаемого в настоящей работе, не оговаривается.

Разработанный подход позволяет на программном уровне сохранить все достоинства ГП, изложенные в разд. 2.3.3, и в том числе самые главные — простоту и корректность внесения изменений, а также хорошую «читаемость».

Применение конструкции `switch` на программном уровне позволяет объединить достоинства таких алгоритмических моделей, как ГСА и ГП, ввиду того что сама эта конструкция является некоторой разновидностью ГСА, переходы в которой осуществляются по номерам состояний изоморфного ГП.

Кроме умолчаний значений выходных переменных существует еще одна специфическая разновидность умолчаний, используемых при применении ФЭЗ (разд. 5.2.3), запуск которых осуществляется, например, с помощью обращения к процедуре **`_time(i, j)`**, а сброс — с помощью процедуры **`reset_time(i)`**, где i — номер ФЭЗ, а j — длительность задержки в секундах. В графе переходов запуску функционального элемента задержки соответствует единица на определенной позиции строки, в которой указываются значения выходных переменных, а сбросу соответствует ноль. При этом значения, относящиеся к ФЭЗ, обычно размещаются после значений выходных переменных.

Эти обозначения были использованы в ГП (рис. 5.40) для примера *5.14,* который может быть реализован следующей программой:

```
        switch (Y) {
case 0: z1=0; z2=0; z3=0; reset_time(1);
        if(x1&x̄4)              Y=1;
        if(x4)                 Y=2;
        break;
case 1: z1=0; z2=1; z3=0;       _time(1,3);
        if(x4&t[1])            Y=2;
        if(x̄4&t[1])            Y=4;
        break;
case 2: z1=0; z2=0; z3=0; reset_time(1);
        if(x2)                 Y=3;
        break;
case 3: z1=1; z2=0; z3=0;       _time(1,3);
        if(x3&t[1])            Y=0;
        if(x̄3&t[1])            Y=4;
        break;
case 4: z1=0; z2=0; z3=1; reset_time(1);
        if(x5)                 Y=0;
        break;
              }.
```

Однако из рассмотрения ГП (рис. 5.40) следует, что в четвертой вершине нет необходимости сбрасывать ФЭЗ, так как он и так будет сброшен в нулевой вершине. Поэтому в четвертой вершине можно не обращаться к ФЭЗ, сохраняя накопленное значение времени, что обозначим символом звездочка (*) в четвертой позиции строки значений выходных и временных переменных в этой вершине. При этом в последней программе в операторе `case 4` обращение к процедуре `reset_time (1)` также может быть исключено.

Рассматриваемый ФЭЗ является троичным по входу, и поэтому если в некоторой вершине нет обращения к этому элементу, то на соответствующей позиции строки значений выходных переменных должна быть указана звездочка. При этом невозможны как дальнейшее «накопление времени», так и сброс этого элемента.

Если в примере *5.14* предположить, что контроль открытия и закрытия осуществляется с помощью различных ФЭЗ, то строки значений выходных переменных в вершинах ГП (рис. 5.40) увеличиваются до пяти позиций и приобретают вид: 00000; 0101*; 000**; 100*1; 001**.

В большинстве случаев, и в частности в рассмотренных примерах, при желании символы «звездочка» могут быть заменены нулями. Это делает все выходные переменные автомата двоичными.

При этом для последнего примера справедливы записи: 00000; 01010; 00000; 10001; 00100. Устранение звездочек выполнять обычно нецелесообразно, так как при этом, как показано выше, текст программы усложняется.

Замена звездочек нулями недопустима в тех случаях, когда информация о срабатывании ФЭЗ используется не сразу после наступления этого события, например, как это имеет место на рис. 14.21.

Естественно, что звездочки для ФЭЗ могут применяться в ГП одновременно с прочерками для выходных переменных, значения которых умалчиваются.