**Варианты решения Задачи обработки данных с использованием асинхронных автоматных схем**

Варианты построения схем обработки данных непрерывного действия рассмотрим на примере задачи преобразования последовательности вещественных чисел *a*0, *a*1,… в последовательность *e*(*a*0), *e*(*a*1),… , где закон преобразования *e* задан условным выражением

*E*1(*a*), при *a* < 0,

*e*(*a*) = (1)

*E*2(*a*), при *a* ≥ 0.

Здесь *E*1 и *E*2 – произвольно заданные функции, использующие *a* в качестве аргумента. Результаты обработки данных на выходе должны следовать строго в той последовательности, в какой они поступают на вход ААС.

На рисунке 1 изображен исходный вариант структуры схемы, реализующей преобразование (1).

*y*1

*y*2

*sr*

*b*1

*b*2

*x*1

*y*1

*rc*

*b*3

*b*4

*x*2

*x*1

*x*1

*y*1

Рисунок 1 – Исходный вариант структуры схемы

Неформально процесс обработки данных, реализуемый схемой, может быть представлен следующим образом. Компонент *p*1 получает элементы последовательности от источника *sr* и помещает их во внутренние буферы схемы *b*1 и *b*2, сортируя в зависимости от знака. Компоненты *p*2 и *p*3 вычисляют, соответственно, значения функций *E*1 и *E*2, которые затем объединяются компонентом *p*4 в единую последовательность, передаваемую в приемник *rc*.

Для произвольных знакопеременных последовательностей порядок следования результатов на выходе будет зависеть от распределения отрицательных и положительных чисел во входной последовательности, а также от времени вычисления значений функций *E*1 и *E*2.

Корректное решение задачи может быть получено различными способами. В основе большинства решений лежит идея **нумерации элементов входной последовательности** и выдачи элементов-результатов на выход схемы в порядке присвоенных им номеров. Номер элемента входной последовательности *i* генерируется компонентом *p*1 и передается компонентам *p*2и *p*3 вместе со значением элемента; компонент *p*4 обеспечивает требуемый порядок выдачи результатов либо **путем непосредственного сравнения номеров *i* двух данных** (решение 1)**,** либо с помощью счетчика *j* (решение 2). Предлагаемые решения могут отличаться способами хранения элементов компонентом *p*4. Отложенные элементы-результаты с номерами *i*,превышающими номер *j* текущего выдаваемого элемента, могут храниться во внешнем петлевом буфере компонента *p*4 или в **его локальных переменных**. В качестве первого решения рассмотрен вариант ААС, в которой порядок выдачи результатов определяется путем непосредственного сравнения номеров *i* двух данных на входах компонента *p*4.

**1. Решение задачи при помощи схемы, локально упорядочивающей элементы на выходе (сортировка методом слияния)**

В данном варианте решения задачи для временного хранения элементов выходной последовательности, которые пока не могут быть выданы в приемник *rc*, используются локальные переменные компонента *p*4. Компонент *p*4 в этом случае будет сортировать нумерованные блоки, поступающие от компонентов *p*2 и *p*3 через буферы *b*3 и *b*4, осуществляя слияние двух упорядоченных потоков данных в один поток результатов, выдаваемый в приемник *rc*. Необходимо отметить, что при реализации сортировки методом слияния отпадает необходимость формировании номера *j* выдаваемого результата при помощи соответствующего счетчика. При выдаче результата в *rc,* сравнивается номер очередного элемента из буферов *b*3 или *b*4 и номер сохраненного в локальной переменной *а* отложенного элемента выходной последовательности. В приемник *rc* выдается результат с меньшим номером.

Структура схемы, при таком подходе к решению задачи, не будет отличаться от структуры исходной схемы *seqConversion*0, изображенной на рисунке 1. Ниже представлена текстовая спецификация этой схемы. Компоненты *p*1, ..., *p*4 имеют уникальные шаблоны *sigma*1,.., *sigma*4.

Диаграмма переходов компонента *p*4 для рассматриваемого случая показана на рисунке 2.

11

***s***1

***s***2

*f*11

1\*

*f*21

\*1

*f*01

***s***0

Рисунок 2 – Диаграмма переходов **компонента *p*4** для случая  
сортировки входных последовательностей методом слияния

Текстовая спецификация схемы имеет вид:

**scheme** seqConversion1; {схема seqConversion3}

**type**

dataBlock:**record** d:real;i:integer;**end**; {тип dataBlock}

**templet** sigma1(**output** y1,y2:dataBlock); {**шаблон sigma1**}

**source**

sr:real; {источник данных sr}

**var**

a:real; {элемент последовательности}

i:integer=0; {номер элемента}

**begin**

**state** s0: {**состояние s0**}

read(sr,a); {читать sr в a}

**if** a<0 **then** {если a<0,то}

**begin**

use(y1); {использовать y1}

y1.d:=a; {значение элемента a в y1.d}

y1.i:=i; {номер элемента в y1.i}

**end**

**else** {иначе}

**begin**

use(y2); {использовать y2}

y2.d:=a; {значение элемента a в y2.d}

y2.i:=i; {номер элемента в y1.i}

**end**;

i:=i+1; {увеличить тек.номер элемента}

next(s0); {выдать y1 или y2,перейти в s0}

**end**; {конец шаблона sigma1}

**templet** sigma2(**input** x1:dataBlock **output** y1:dataBlock); {**шаблон sigma2**}

**function** E1(x:real):real; {описание функции E1}

. . .

**begin**

**state** s0: {**состояние s0**}

**when** nem(x1)**using** x1 **do** {используя x1}

use(y1); {использовать y1}

y1.d:=E1(x1.d); {значение E1(a) в y1.d}

y1.i:=x1.i; {номер элемента в y1.i}

next(s0); {выдать y1,перейти в s0}

**end**; {конец шаблона sigma2}

**templet** sigma3(**input** x1:dataBlock **output** y1:dataBlock); {**шаблон sigma3**}

**function**E2(x:real):real; {описание функции E2}

. . .

**begin**

**state** s0: {**состояние s0**}

**when** nem(x1)**using** x1 **do** {используя x1}

use(y1); {использовать y1}

y1.d:=E2(x1.d); {значение E2(a) в y1.d}

y1.i:=x1.i; {номер элемента в y1.i}

next(s0); {выдать y1,перейти в s0}

**end**; {конец шаблона sigma3}

**templet** sigma41(**input** x1,x2:dataBlock); {**шаблон sigma42**}

**receiver**

rc:real; {приемник данных rc}

**var**

a:dataBlock; {локальная переменная для}

**begin** {хранения промежут. результатов}

**state** s0: {**состояние s0** }

**when** nem(x1,x2)**using** x1,x2 **do** {**ветвь f01,** используя x1,x2}

**if** x1.i<x2.i **then** {если номер x1.i< x2.i, то}

**begin**

write(rc,x1.d); {записать x1.d в rc}

a:=x2; {сохранить x2 в переменной a}

**next** s1;{перейти в s1}

**end**

**else** {иначе}

**begin**

write(rc,x2.d); {записать x2.d в rc}

a:=x1; {сохранить x1 в переменной a}

**next** s2; {перейти в s2}

**end**;

**state** s1: {**состояние s1, приоритет у x1**}

**when** nem(x1)**using** x1 **do** {**ветвь f11**, используя x1}

**if** x1.i<a.i **then** {если номер элемента на вх. x1}

**begin** {меньше номера элемента a.i,то}

write(rc,x1.d); {записать x1.d в rc}

**next** s1; {перейти в s1}

**end**

**else** {иначе}

**begin**

write(rc,a.d); {записать в rc элемент a.d}

a:=x1; {сохранить x1 в переменной a}

**next** s2; {перейти в s2}

**end**;

**state** s2: {**состояние s2, приоритет у x2**}

**when** nem(x2)**using** x2 **do** {**ветвь f21**, используя x2}

**if** x2.i<a.i **then** {если номер элемента на вх. x2}

**begin** {меньше номера элемента a.i,то}

write(rc,x2.d); {записать x2.d в rc}

**next** s2 {перейти в s2}

**end**

**else** {иначе}

**begin**

write(rc,a.d); {записать в rc элемент a.d}

a:=x2; {сохранить x2 в переменной a}

**next** s1; {перейти в s1}

**end**

**end**; {конец шаблона sigma42}

**buf** {описания буферов b1-b4}

b1,b2,b3,b4:dataBlock[2];

**comp** {описания компонентов p1-p4}

p1:sigma1(**out** b1,b2);

p2:sigma2(**in** b1 **out** b3);

p3:sigma3(**in** b2 **out** b4);

p4:sigma4(**in** b3,b4);

**end** {конец схемы seqConversion1}

Рассмотренный вариант сортировки методом слияния двух упорядоченных последовательностей имеет два недостатка.

Во-первых, для выдачи первого результата необходимо **наличие данных в обоих входных буферах компонента *p*4.** Только в этом случае элемент-результат с меньшим номером выдается в приемник *rc*, с большим номером − сохраняется в локальной переменной *a* компонента *p*4. Поэтому, когда первые *n* элементов входной последовательности, имеющих один знак и преобразованных быстрее первого элемента с другим знаком, могли бы быть выданы из входного буфера в приемник *rc*, компонент *p*4 не делает этого, ожидая единственной рабочей ситуации 11 на своих входах.

Во-вторых, при очень длинных входных последовательностях может возникнуть переполнение разрядной сетки при инкременте номера *i* в компоненте *p*1. В этом случае в компоненте *p*4 сравнение *x*1.*i* < *a*.*i* в состоянии ***s***1 или *x*2.*i* < *a*.*i* в состоянии ***s***2 даст неверный результат, и схема выйдет из строя. Следовательно, корректная работа схемы определяется длиной последовательности входных элементов, номера которых не должны превосходить 2*m*-1, где *m* – разрядность ячейки памяти, отведенной для хранения значений переменной *i*.

**2. Решение задачи при помощи схемы, реализующей метод счетчикового упорядочения последовательностей**

Для устранения указанных недостатков предлагается использовать метод, который можно назвать методом счетчикового упорядочения последовательностей. Этот метод предполагает, что в начальном состоянии ***s***0 рабочей ситуацией является любая из трех ситуаций на входах компонента – 01, 10, 11 – или, другими словами, для продолжения функционирования компоненту достаточно, чтобы данные присутствовали хотя бы на одном из его входов. Если элемент данных есть только на одном из входов, он считывается из соответствующего входного буфера, и номер *i* элемента сравнивается с содержимым счетчика *j* на равенство. При несовпадении значений *i* и *j* входной элемент запоминается в локальной переменной компонента *p*4. В ходе дальнейшего анализа в каждом состоянии компонента *p*4 номер элемента последовательности *i* также сравнивается на равенство с содержимым счетчика *j.* Поскольку значения *i* и *j* изменяются синхронно, переполнение разрядной сетки при инкременте как переменной *i* так и переменной *j* не приведет к отказу работы схемы.

Диаграмма переходов компонента *p*4 для метода счетчикового упорядочения последовательностей элементов, поступающих на входы *x*1, *x*2 компонента *p*4, показана на рисунке 3.

S0

*f*02

*f*01

10

11

*f*03

01

***s***1

***s***2

*f*11

1\*

*f*21

1\*

S0

***s***0

Рисунок 3 – Диаграмма переходов компонента *p*4, реализующая  
счетчиковое упорядочение последовательностей

Как и в предыдущем варианте, изменения не касаются структуры всей схемы. Ниже приведено текстовое описание ААС, реализующей метод счетчикового упорядочения последовательностей.

**scheme** seqConversion4; {схема seqConversion4}

. . .

**templet** sigma42 (**input** x1,x2:dataBlock); {**шаблон sigma4**}

**receiver**

rc:real; {приемник данных rc}

**var**

a:dataBlock; {локальная переменная}

j:integer=0; {номер элемента результата}

**begin**

**state** s0: {**состояние s0**}

**when** nem(x1)**and** emp(x2)**using** x1 **do** {**ветвь f01,** используя x1}

**if** x1.i=j **then** {если номер x1.i равен j,то}

**begin**

write(rc,x1.d); {записать x1.d в rc}

j:=j+1; {увеличить номер результата}

**next** s0; {перейти в s0}

**end**

**else** {иначе}

**begin**

a:=x1; {сохранить x1 в переменной a}

**next** s2; {перейти в s2}

**end**;

**when** nem(x1,x2)**using** x1,x2 **do** {**ветвь f02,** используя x1,x2}

**if** x1.i=j **then** {если номер x1.i равен j, то}

**begin**

write(rc,x1.d); {записать x1.d в rc}

j:=j+1; {увеличить номер результата}

a:=x2; {сохранить x2 в переменной a}

**next** s1;{перейти в s1}

**end**

**else** {иначе}

**begin**

write(rc,x2.d); {записать x2.d в rc}

j:=j+1; {увеличить номер результата}

a:=x1; {сохранить x1 в переменной a}

**next** s2; {перейти в s2}

**end**;

**when** nem(x2)**and** emp(x1)**using** x2 **do** {**ветвь f03,** используя x2}

**if** x2.i=j **then** {если номер x2.i равен j,то}

**begin**

write(rc,x2.d); {записать x2.d в rc}

j:=j+1; {увеличить номер результата}

**next** s0; {перейти в s0}

**end**

**else** {иначе}

**begin**

a:=x2; {сохранить x2 в переменной a}

**next** s1; {перейти в s1}

**end**;

**state** s1: {**состояние s1**}

**when** nem(x1)**using** x1 **do** {**ветвь f11,** используя x1}

**if** x1.i=j **then** {если номер x1.i равен j,то}

**begin**

write(rc,x1.d); {записать x1.d в rc}

j:=j+1; {увеличить номер результата}

**next** s1; {перейти в s1}

**end**

**else** {иначе}

**begin**

write(rc,a.d); {записать a.d в rc}

j:=j+1; {увеличить номер результата}

a:=x1; {сохранить x1 в переменной a}

**next** s2; {перейти в s2}

**end**;

**state** s2: {**состояние s2**}

**when** nem(x2)**using** x2 **do** {**ветвь f21,** используя x2}

**if** x2.i=j **then** {если номер x2.i равен j,то}

**begin**

write(rc,x2.d); {записать x2.d в rc}

j:=j+1; {увеличить номер результата}

**next** s2; {перейти в s2}

**end**

**else** {иначе}

**begin**

write(rc,a.d); {записать a.d в rc}

j:=j+1; {увеличить номер результата}

a:=x2; {сохранить x2 в переменной a}

**next** s1; {перейти в s1}

**end;**

**end**; {конец шаблона sigma43}

. . .

**end** {конец схемы seqConversion4}

**3. Решение задачи при помощи схемы с управляющим буфером**

Поставленная задача имеет решение, не требующее нумерации блоков. В основе этого решения лежит идея управления работой компонента *p*4 компонентом *p*1 через связывающий их буфер *b*5. Структура соответствующей схемы показана на рисунке 4.

*y*1

*y*2

*sr*

*b*1

*b*2

*x*1

*y*1

*rc*

*b*3

*b*4

*x*2

*x*1

*x*1

*y*1

*x*3

*y*3

*b*5

Рисунок 4 – Структура схемы с управляющим буфером

В процессе функционирования компонент *p*1, непосредственно перед посылкой в буферы *b*1, *b*2 элементов входной последовательности, помещает в буфер *b*5 значение *true*, если элемент входной последовательности *a* ≥ 0, и значение *false*, если *a* < 0. Компонент *p*4 упорядочивает выходные данные, выбирая элементы *E*1(*a*), *E*2(*a*) из буферов *b*3, *b*4 в зависимости от логических значений, получаемых им из буфера *b*5.

Схема с управляющим буфером имеет следующую спецификацию:

**scheme** seqConversion3; {схема seqConversion5}

**templet** sigma11(**input** y1,y2:real **output** y3:boolean); {**шаблон sigma11**}

**source**

sr:real; {источник данных sr}

**var**

a:real; {элемент последовательности}

**begin**

**state** s0: {**состояние s0**}

read(sr,a); {читать sr в a}

**if** a<0 **then** {если a<0,то}

**begin**

use(y1,y3); {использовать y1,y3}

y1:=a; {y1 присвоить a}

y3:= **false**; {y3 присвоить false}

**end**

**else** {иначе}

**begin**

use(y2,y3); {использовать y2,y3}

y2:=a; {y2 присвоить a}

y3:= **true**; {y3 присвоить true}

**end**;

**next**(s0); {выдать y1 или y2,y3,пер. в s0}

**end**; {конец шаблона sigma1}

**templet** sigma2(**input** x1:real **output** y1:real); {**шаблон sigma2**}

**function** E1(par:real):real; {описание функции E1}

. . .

**begin**

**state** s0: {**состояние s0**}

**when** nem(x1)**using** x1 **do**  {используя x1}

use(y1); {использовать y1}

y1:=E1(x1); {y1 присвоить E1(x1)}

next(s0); {выдать y1,перейти в s0}

**end**; {конец шаблона sigma2}

**templet** sigma3(**input** x1:real **output** y1:real); {**шаблон sigma3**}

**function** E2(par:real):real; ; {описание функции E2}

. . .

**begin**

**state** s0: {**состояние s0**}

**when** nem(x1)**using** x1 **do** {используя x1}

use(y1); {использовать y1}

y1:=E2(x1); {y1 присвоить E2(x1)}

next(s0); {выдать y1,перейти в s0}

**end**; {конец шаблона sigma3}

**templet** sigma44(**input** x1,x2:real;x3:boolean); {**шаблон sigma44**}

**receiver**

rc:real; {приемник данных rc}

**begin**

**state** s0: {**состояние s0**}

**when** nem(x3)**using** x3 **do** {используя x3}

**if** **not** x3 **then** {если false,то}

next(s1); {перейти в s1}

**else** {иначе}

**next**(s2); {перейти в s2}

**state** s1: {**состояние s1**}

**when** nem(x1)**using** x1 **do** {используя x1}

write(rc,x1); {записать x1 в rc}

**next**(s0); {перейти в s0}

**state** s2: {**состояние s2**}

**when** nem(x2)**using** x2 **do** {используя x2}

write(rc,x2); {записать x2 в rc}

**next**(s0); {перейти в s0}

**end**; {конец шаблона sigma44}

**buf** {описания буферов b1-b5}

b1,b2,b3,b4:real[1]; {размер буферов равен 1}

b5:boolean[4]; {размер буферов равен 4}

**comp** {описания компонентов p1-p4}

p1:sigma1(**out** b1,b2,b5);

p2:sigma2(**in** b1 **out** b3);

p3:sigma3(**in** b2 **out** b4);

p4:sigma4(**in** b3,b4,b5);

**end** {конец схемы seqConversion3}

В заключение следует отметить, что рассмотренные примеры решения казалось бы несложной задачи обработки данных, в полной мере демонстрируют возможность применения асинхронных автоматных схем для решения задач рассматриваемого класса. При этом важно подчеркнуть, что предлагаемый подход к представлению асинхронных автоматных схем дает также возможность создания коллекций (библиотек) шаблонов компонентов, наиболее часто используемых при построении реальных схем.