Дипломная работа

Максим Борисяк

8 июня 2013 г.

1 Введение

Введение.

1.1 Мотивация

Мотивация

1.2 Базовые понятия и определения *** Need review ***

В данном разделе будут введены базовые понятия, используемые в данной работе, в том числе, понятия атомарного блока, составного блока, потока данных, автомата Мили соответствующего блоку и так далее.

Основной единицей потока данных является атомарный блок.

Определение 1. ***Шаблоном атомарного блока*** *s (сокращенное от англ. scheme — схема) назовем кортеж имеющий два не пересекающихся набора портов: входной и выходной. $s = (FA, I, O)$, где:*

- $I = \{b_i^I | i = \overline{1, N_I}, N_I \in \mathbb{N}\}$ *— конечное множество уникальных входных портов;*
- $O = \{b_i^O | i = \overline{1, N_O}, N_O \in \mathbb{N}\}$ *— конечное множество уникальных выходных портов;*
- *FA — конечный автомат Мили, смысл и структура которого будет обсуждаться в дальнейшем.*

Сразу заметим, что не любая тройка (FA, I, O) является шаблоном атомарного блока, так как FA должен соответствовать множествам I и O.

Стоит отметить, что природа множеств O и I совершенно не важна, например, они могут быть просто множеством чисел: $I, O = \{i \in \overline{1, \ldots, N_{I,O}}\}$, на практике часто используют строки для именования портов, так как порты могут существенно различаться по смыслу, а их именование помогает избежать путаницы. Последний подход используется в рисунках.

Определение 2. ***Атомарным блоком*** *$b = (id, s, state)$, где:*

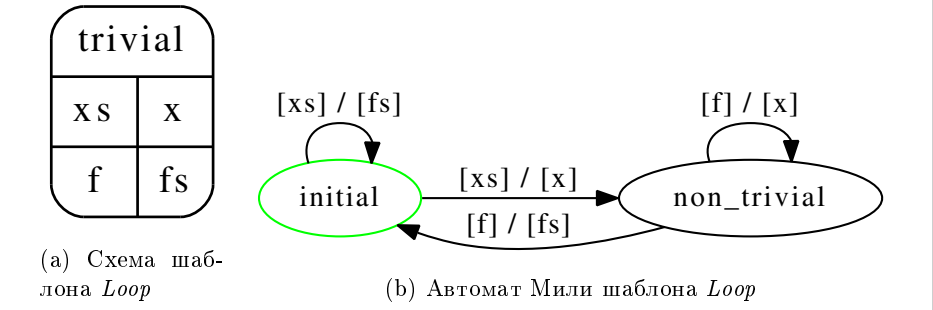
- *id — уникальный для всех блоков идентификатор;*
- *s — шаблон атомарного блока;*

- *state* $\in S$, *где* S *множество состояний* FA_s — *текущее состояние атомарного блока.*

Идентификатор *id* введен из-за следующих соображений: в потоке данных некие блоки могут описываться одним и тем же шаблоном, находиться в одном и том же состоянии, но все равно будут являться различными блоками, в частности, могут перейти в различные состояния в следующие моменты времени либо могут иметь в потоке данных разный смысл и, соответственно, иметь различные соединения с другими блоками.

В реальных системах для построения и запуска потока данных шаблон атомарного блока представляется в виде подпрограммы или модуля (функции, класса некого объектно-ориентированного языка программирования, динамической библиотеки), который используя данные некого набора своих входных портов, вычисляет и записывает данные в свои выходные порты. В таком случае, атомарный блок представляется в виде набора внутренних переменных, либо (что тоже самое) в виде экземпляра класса объектно-ориентированного языка программирования соответствующего шаблону. В такой интерпретации некую вычислимую функцию $f(x_1, x_2, \ldots, x_n)$ можно представить в виде шаблона блока b_f с входными портами $x_1, x_2, \ldots, x_n$ и одним выходным портом F, который при получении данных со всех входных портов выписывает значения функции $f(x_1, x_2, \ldots, x_n)$ в порт F.

На рисунке 1 схематично изображен шаблон блока *Loop*, для которого $inputs(Loop) = \{xs, f\}$, $outputs(Loop) = \{fs, x\}$.



(a) Схема шаблона *Loop*

(b) Автомат Мили шаблона *Loop*

Рис. 1: Схематичное изображение шаблона блока *Loop* (a) и соответствующего ему конечного автомата Мили (b).

Основной структурой для построения схемы потока данных является составной блок.

Определение 3. ***Шаблоном составного блока*** *будем называть кортеж* $c = (B, E, I, O)$, *где:*

- B *- конечное множество блоков (сразу заметим, что блоком может являться и составной блок, определенный ниже).*
- E *- множество ребер вида* $e_b = (b_1, b_1^O, b_2, b_2^I)$, $b_1, b_2 \in B$, $b_1^O \in outputs(b_1), b_2^I \in inputs(b)$, *либо вида* $e_I = (c^I, b, b^I)$, $e_O = (b, b^O, c^O)$, *где* $b \in B$, $b^I \in inputs(b)$, $b^O \in outputs(b)$, $c^O \in O$, $c^I \in I$. *Множества ребер* e_b *будем обозначать как* E^B, e_I *и* e_O *как* E^I *и* E^O *соответственно.*

- *I, O - наборы входных и выходных портов (по аналогии с атомарным блоком).*

В дальнейшем в выражениях вида I_c или O_c нижний индекс будет указывать на схему блока, которому принажлежат эти множества.

Для анализа внутренней структуры удобно рассматривать иное представление шаблона составного блока c, добавляя фиктивные блоки STOCK, $I_{\text{STOCK}} = O_{\text{STOCK}} = O_c$, и SOURCE, $I_{\text{SOURCE}} = O_{\text{SOURCE}} = I_c$:

$$\hat{E}^I = \{(\text{SOURCE}, \text{SOURCE}^O, b, b^I) | (\text{SOURCE}^O, b, b^I) \in E^I\}$$

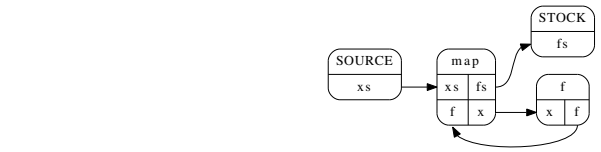
$$\hat{E}^O = \{(b, b^O, \text{STOCK}, \text{STOCK}^I) | (b, b^O, \text{STOCK}) \in E^O\}$$

$$\hat{c} = (B \cup \{\text{STOCK}, \text{SOURCE}\}, E^B \cup \hat{E}^I \cup \hat{E}^O)$$

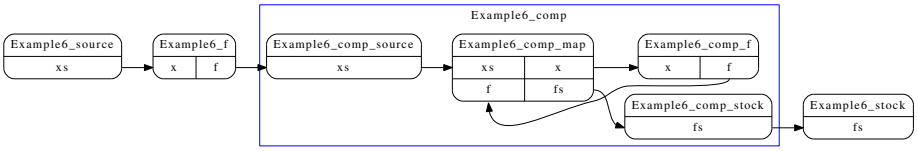
В этом случае шаблон составного блока $\hat{c}$ описывается графом с ребрами вида (u, u^O, v, v^I), а входными и выходными портами всего шаблона считаются входные и выходные порты блоков SOURCE и STOCK. На рисунке 2 изображен примеры шаблонов составных блоков во втрой интерпретации.

Определение 4. ***Составным блоком*** *будем называть $b = (id, c = (B, E, I, O), state)$, где*

- *id — уникальный идентификатор;*
- *c — шаблон составного блока;*
- *$state \in \Sigma \times 2^E = \prod S(B) \times 2^E \times 2^B$ - текущие состояние составного блока, включающее состояние каждого блока (подпространство $\Sigma = \prod S(B)$), активную волну ($\omega \in \Omega = 2^E$) и множество активных блоков ($\vartheta \in \Theta = 2^B$), смысл которых будет пояснен позже.*



(a) Шаблон составного блока



(b) Шаблон супер-составного блока. Внутри рамки показана структура шаблона внутреннего блока.

Рис. 2: Примеры графов составного и супер-составного блока.

Заметим, что составной блок (в исходной интерпретации) внешне не отличим от атомарного блока. Поэтому под понятием блока мы будем подразумевать либо атомарный блок, либо составной блок, различая их только

в том случае, когда речь заходит о их внутреннем строении. Составной блок может включать в себя другие составные блоки. На рисунке 2b показан пример составного блока, включающего другой составной блок, иными словами супер-составного блока. На практике свойство вложенности реализует принцип модульности программ — модулем в нашем случае будет шаблон составного блока. Бессмысленно рассматривать случаи, в которых структура составного блока бесконечна в глубину, например, в случае когда шаблон составного блок содержит содержит блок того же шаблона. Поэтому в дальнейшем, мы будем рассматривать только составные блоки конечной глубины.

Заметим, что составным блоком можно описать любую программу, имея в распоряжении необходимые примитивы — шаблоны атомарных блоков, реализующие Тьюринг-полную систему операций. Доказательства этого факта выходит за задачи данной работы и может быть найдено, например, в работе **[???]**. Из-за достаточной выразительности составного блока, схему потока данных мы определим просто как схему некого составного блока.

Теперь неформально поясним схему работы составного блока и смысл его структуры. В общепринятых определениях потока данных ребра графа шаблона составного блока обозначают зависимость по данным, то есть ребро $e = (u, u^O, v, v^I)$ представляет собой *FIFO*-канал соединения между портами блоков и можно словесно интерпретировать следующим образом: блок v в качестве данных из входного порта v^I должен использовать данные выходного порта u^O блока u при их наличии. Блок u после завершения своей работы может "испустить данные по порту u^O которые мгновенно "переносятся"на все ребра из u^O блока u, если эти ребра свободны. Когда блоку v требуются данные из порта v^I, данные с ребра e мгновенно "переносятся"в соответствующий порт блока. В этом и есть смысле подпространства $\Omega = 2^E$ в определении состояния составного блока — *активной волной* ω мы назовем множество ребер, которые содержат на данный момент данные.

Однако такая схема работы составного блока порождает множество неопределенностей в работе некоторых составных блоков, например, неопределенность выбора ребра для "поглощения"данных из определенного порта (состояние гонки). Неопределенности подобного рода мы будем рассматривать как ошибки, обнаружению которых посвящена часть данной работы. Такого рода неодназначности, особенно в реальных системах, то есть в условиях неопределенного времени вычислений блоков и их параллельного выполнения, может привести к совершенно иному ходу потока данных, нежели было задумано автором потока данных. Поэтому наличие подобного рода неопределенностей (состояний гонки) говорит скорее о неправильном построении схемы потока данных, чем о некой задумке автора потока данных. Более подробно и строго об неоднозначностях в поведении потока данных будет идти речь ниже. Стоит заметить, что подобного рода неоднозначности возникают и в процессе определения сетей процессов Кана, которые решаются наложением условий на правила поглощения и испускания, а также на класс функции процесса. Мы же пойдем другим путем, накладывая ограничения лишь на граф шаблона потока данных, оставляя подпрограммы и правила поглощения каждого шаблона блока без ограничений. В нашем случае правила поглощения состоят в том, что формально блок может поглотить любые доступные данные из любых портов по любым ребрам, но будем расценивать любые возможные неоднозначности как ошибку построения

потока данных.

Выше были рассмотрены определения и термины, которые с той или иной точностью присутствуют практически во всех моделях потока данных. В данной работе мы абстрагируемся от данных, а значит нас интересуют только возможные зависимости наборов входных портов от набора выходных портов, иными словами только качественное поведение блоков. Оказывается, конечный автомат Мили хорошо подходит для описания качественного поведения блоков. Заметим, что в данной работе используется не стандартное определение автомата Мили, а его недетерминированные аналог. Дадим теперь строгое определение конечного автомата Мили.

Определение 5. ***Конечный автомат Мили*** *— кортеж* $FA = (S, s_0, \Sigma, \Lambda, E)$*, где:*

- S *— конечное множество состояний;*
- $s_0 \in S$ *— начальное состояние;*
- Σ *— входной алфавит;*
- Λ *– выходной алфавит;*
- $E \subseteq S \times \Sigma \times \Lambda \times S$ *— отношение возможных переходов.*

Для удобства будем рассматривать функцию конечного автомата Мили $FA = (S, \cdot, \Sigma, \Lambda, E)$: $FA(p, \sigma) = \{(\lambda, q, \sigma') | (p, \sigma', \lambda, q) \in E, \sigma' \subseteq \sigma\}$. Стоит обратить внимание на нестандартную форму данной функции. Далее символами в алфавитах Σ и Λ будут множества, поэтому функция по текущему состоянию и входному множеству возвращает множество троек вида: выходной символ-множество, новое состояние и поглощенное множество.

Конечный автомат Мили выполняет преобразование последовательности входных символов из алфавита Σ в последовательность символов в алфавите Λ. Вернемся теперь к определению шаблона атомарного блока. В данной работе мы абстрагируемся от конкретного алгоритма преобразования входных данных в выходные и преобразования внутреннего состояния блока. Вместо этого мы будем описывать лишь качественное поведение алгоритма автоматом Мили FA, в котором входной алфавит $\Sigma = 2^I \setminus \varnothing$ (запрещается запуск по пустому набору входных портов), а выходной алфавит $\Lambda = 2^O$, иными словами ребро $e = (u, is, os, v)$ автомата FA описывает возможный переход из состояния u в состояние v, при считывании данных из портов is и выписывании данных в порты os (именно поэтому функция FA имеет странный на первый взгляд вид). На практике подпрограмма реализующая логику атомарного блока может иметь значительное количество внутренних состояний, однако, для наших целей важны лишь группы таких внутренних состояний, которые описывают возможное поведение на определенном наборе данных из входных портов. Более того практически во всех системах построения и запуска потока данных подпрограмма описывающая блок имеет детерминированное поведение. Неизбежный недетерминизм в автомате Мили, описывающий блок, возникает из-за исключения из рассмотрения самих данных, так как при разных значениях на одном и том же наборе портов из одного состояния, блок может перейти в различные конечные состояние, даже выписав при этом данные в один и тот же набор портов.

На рисунке 1b изображен автомат Мили для шаблона атомарного блока *Loop*, описывающего цикл. Поясним эту схему. Изначально блок с шаблоном *Loop* находиться в состоянии *initial*. Далее он может считать из порта *xs* начальные параметры цикла (например, некую коллекцию данных). Если условие выхода из цикла изначально выполнено (например, пустая коллекция), то блок переходит в прежнее состояние выписав в порт *fs*. Иначе, блок отправляет данные по порту x переходя в состояние *non_trivial*. В состоянии *non_trivial* блок имеет не тривиальное внутреннее состояние, например, остаток коллекции. В этот состоянии блок ожидает поступление преобразованных некой внешней функцией данных на порт *f*. И в этом случае существует два варианта перехода, в зависимости от выполнения условия выхода из цикла: $(non\_trivial, \{f\}, \{x\}, non\_trivial)$ и $(non\_trivial, \{f\}, \{fs\}, trivial)$ для случаев продолжения цикла и выхода из него соответственно.

Конечный автомат Мили может быть также определен для составного блока. Единственное отличие от автомата Мили для атомарного блока состоит в том, что зная все автоматы внутренних блоков, можно вычислить автомат составного блока как будет показано ниже.

1.2.1 Алгоритм работы потока данных ***Need review***

Теперь опишем алгоритм вычисления потока данных, представленного в виде составного блока c со схемой $C = (\hat{B}_c = B_c \cup \{\text{SOURCE}, \text{STOCK}\}, \hat{E}_c = E_c \cup E_c^I \cup E_c^O)$ в представлении графа. Мы будем рассматривать вычисление потока данных в предположении, что время работы блока может быть абсолютно любым (но конечным) при любых условиях, что полностью соответствует реальным потокам данных. В таком случае нас прежде всего будет интересовать всевозможные поведения потока данных с точки зрения относительных расположений времен запуска и завершения любого блока. Введем модельное время $t \in \mathbb{Z}_+$. Каждое множество, связанное с состоянием системы в момент времени t, снабдим верхним индексом, например, $\omega^t \in \Omega_c$.

Определение 6. *Определим некоторые вспомогательные функции.*

$$\begin{aligned} O_v(\omega) &= \{v^O | (v, v^O, \cdot, \cdot) \in \omega\}, \\ I_v(\omega) &= \{v^I | (\cdot, \cdot, v, v^I) \in \omega\}, \\ \Delta_{O,v} &= \{(v, v^O, \cdot, \cdot) \in E_c | v^O \in O\}, \\ \Delta_{I,v}(\omega) &= \{\tilde{\omega}_{I,v} \subseteq \omega\},\ \forall \tilde{\omega}_I \forall v^I \in I \exists! e \in \omega_I : e = (\cdot, \cdot, v, v^I) \end{aligned}$$

Последняя функция возвращает множество вариантов поглощения по набору входных портов I блока v.

Определение 7. *Фронтом* $\Psi_c(\omega, s)$ *активной волны* $\omega = \{(u, u^O, v, v^I)\} \in \Omega_c$ *в состоянии* $(s = (s_{u_1}, s_{u_2}, \ldots, s_{u_m}), \omega, \cdot)$ *назовем подмножество блоков* $\Psi_c(\omega, s) \subseteq B_c$, *такое что:*

$$\Psi_c(\omega, s) = \{v \in B | FA_v(s_v, I_v(\omega)) \neq \varnothing\}$$

Иными словами, фронт активной волны в заданном состоянии — множество блоков, которые могут быть запущенны на следующем шаге.

Определение 8. *Семейство* $\pi_v^t \in E_v,\ FA_v = (\cdot, \cdot, E_v, \cdot)$ *показывает текущий переход, который совершает блок* v *в момент времени* t.

Заметим, что значение π_v^t имеет смысл только для работающих на шаге t блоков.

Определение 9. *Условием излучения назовем* $\zeta(\omega^t, \omega^O) \Leftrightarrow (\omega^O \cap \omega^t = \varnothing)$. *Также определим семейства* ζ_v^t*:*

$$\zeta^t = \{v \in B_c | zeta(\omega^t, \omega_v^O)\}$$

$$\omega_v^O = \{e \in E_c | e = (v, v^O, \cdot, \cdot), v^O \in O_v^t\},\ (\cdot, O_v^t, \cdot, \cdot) \in \pi_v^t$$

Условие излучение звучит довольно просто: блок может излучить, если все ребра соединенные с выходным набором портов свободны.

Пусть $\phi^t \subseteq \vartheta^t$ — множество блоков, которые закончат работу после шага t. Для любого времени это множество может быть произвольным подмножеством ϑ^t. Однако для любого семейства этих множеств должно выполняться условие конечной работы блока: $\forall t \forall b \in \theta^t \exists \tau > t : b \in \phi^\tau$.

Так же для удобства введем обозначение для любого множества A_c блоков из B_c — $A_c(v) \Leftrightarrow v \in A_c, v \in B_c$.

Теперь выпишем алгоритм преобразования потока данных в виде условий на переходы каждого блока из одного множества в другое. В каждый момент времени блок v может находиться в одном из трех состояний (множеств): свободен ($v \in \xi^t$), работает($v \in \theta^t$), заблокирован($v \in \chi^t$). Время работы блоков задается $\phi^t \subseteq \vartheta^t$ — множеством блоков, которые закончат работу после шага t. Для любого времени это множество может быть произвольным подмножеством ϑ^t. Однако для любого семейства этих множеств должно выполняться условие конечной работы блока: $\forall t \forall b \in \theta^t \exists \tau > t : b \in \phi^\tau$. Эти состояние взяты из реальных систем запуска потоков данных и служат для интуитивного понимания модели. В дальнейшем, однако, мы избавимся от этих состояний. Также введем семейства — переход автомата Мили блока, который совершает блок в текущий момент.

$$\xi^t(v) \wedge \neg\Psi_c(\omega^t, s^t)(v) \quad \Rightarrow \quad \begin{cases} \xi^{t+1}(v), \\ s_v^{t+1} = s_v^t, \\ \omega_{I,v}^t = \omega_{O,v}^t = \varnothing \end{cases} \tag{1}$$

$$\xi^t(v) \wedge \Psi_c(\omega^t, s^t)(v) \quad \Rightarrow \quad \begin{cases} \theta^{t+1}(v), \\ (\cdot, \cdot, I_v^t) \in FA_v(s_v^t, I_v(\omega^t)), \\ \pi_v^{t+1} \in \{(s_v^t, I_v^t, \cdot, s_v^{t+1}) \in E_v\}, \\ \omega_{I,v}^t \in \Delta_{I_v^t, v}(\omega^t), \\ \omega_{O,v}^t = \varnothing \end{cases} \tag{2}$$

$$\theta^t(v) \wedge \neg\phi^t(v) \quad \Rightarrow \quad \begin{cases} \theta^{t+1}(v), \\ s_v^{t+1} = s_v^t, \\ \pi_v^{t+1} = \pi_v^t, \\ \omega_{I,v}^t = \omega_{O,v}^t = \varnothing \end{cases} \tag{3}$$

$$\theta^t(v) \wedge \phi^t(v) \wedge \zeta^t(v) \quad \Rightarrow \quad \begin{cases} \xi^{t+1}(v), \\ (s_v^t, \cdot, O_v^t, s_v^{t+1}) = \pi_v^t, \\ \omega_{O,v}^t = \Delta_{O_v^t, v}, \\ \omega_{I,v}^t = \varnothing \end{cases} \tag{4}$$

$$\theta^t(v) \wedge \phi^t(v) \wedge \neg\zeta^t(v) \quad \Rightarrow \quad \begin{cases} \chi^{t+1}(v), \\ \pi_v^{t+1} = \pi_v^t, \\ \omega_{O,v}^t = \omega_{I,v}^t = \varnothing \end{cases} \tag{5}$$

$$\chi^t(v) \wedge \zeta^t(v) \quad \Rightarrow \quad \begin{cases} \xi^{t+1}(v), \\ (s_v^t, \cdot, O_v^t, s_v^{t+1}) = \pi_v^t, \\ \omega_{O,v}^t = \Delta_{O_v^t, v}, \\ \omega_{I,v}^t = \varnothing \end{cases} \tag{6}$$

$$\chi^t(v) \wedge \neg\zeta^t(v) \quad \Rightarrow \quad \begin{cases} \chi^{t+1}(v), \\ s_v^{t+1} = s_v^t, \\ \pi_v^{t+1} = \pi_v^t, \\ \omega_{O,v}^t = \omega_{I,v}^t = \varnothing \end{cases} \tag{7}$$

$$\omega^{t+1} = \omega^t \setminus (\cup_{v \in B_c} \omega_{I,v}^t) \cup (\cup_{v \in B_c} \omega_{O,v}^t), \tag{8}$$

$$s^{t+1} = (s_{v_1}^t, \ldots, s_{v_m}^t) \tag{9}$$

Начальные условия:

$$\omega^0 = \varnothing, \tag{10}$$

$$s_v^0 = s_{0,v}, \forall v \in B_c, \tag{11}$$

$$\theta^0 = \{\text{SOURCE}\}, \tag{12}$$

$$\xi^0(v) = \{v \in B_c | v \neq \text{SOURCE}\} \tag{13}$$

Работу каждого блока можно схематично изобразить в виде автомата, см. рисунок 3.

Поясним эти правила. Каждое правило (1) — (7) описывает условие перехода в конечном автомате на рисунке 3.

Правила (1) и (2) говорят о поведении свободного блока — если блок может запуститься из его текущего состояния s_v^t (то есть хватает данных на портах для какого-либо перехода), то он поглощает данные $\omega_{I,v}^t$ и начинает некий переход π_v^{t+1} из текущего состояния, иначе остается свободным дальше.

Тройка правил (3), (4) и (5) описывают поведение работающего блока v. Если множество ϕ^t не содержит рассматриваемый блок, он продолжает

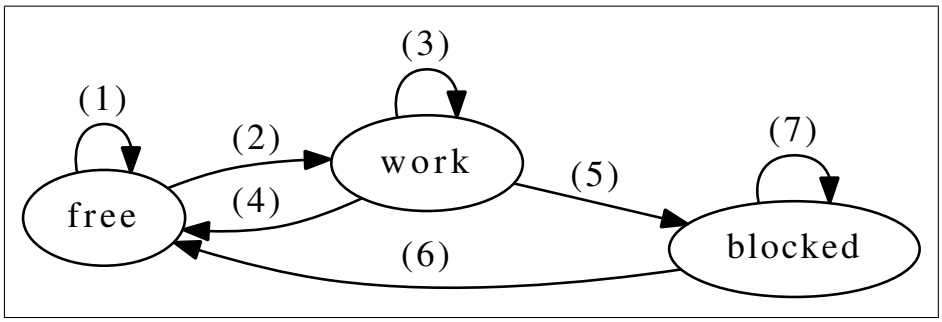


Рис. 3: Схема работы блока

работу. Иначе, возможны два варианта: либо блок может завершиться и испустить данные $\zeta^t(v)$ (то есть ребра $\Delta_{O_v^t,v}$, связанные с испускаемыми портами O_v^t, свободны), то блок благополучно завершает работу $\xi^{t+1}(v)$ и испускает данные ($\omega_{O,v}^t$), либо блок переходит в состояние блокировки до момента, когда требуемые ребра освободятся, которое описывается правилами (6) и (7).

Заметим, что мы специально различаем состояния блокировки и работы, хотя можно было бы считать, что заблокированный блок просто работает дополнительное время. Состояние блокировки отличается тем, что блок в этом состоянии не требует вычислительных ресурсов, что будет важно в дальнейшем.

Заметим, что в правилах поведения потока данных сразу видны все неопределенности, о которых говорилось ранее — перед запуском блока, определяется переход, который будет совершать блок, вместе с набором поглощаемых портов и ребра с которых эти данные будут поглащены. Подробнее мы будем рассматривать эти неопределенности далее.

2 Анализ модели

2.1 Построение конечного автомата Мили по программе блока ***Need review***

Введенная выше модель оперирует с упрощенными описаниями алгоритмов блоков — каждый алгоритм заменяется на описывающий его качественное поведение автомат Мили. Каждое состояние конечного автомата соответствует группам внутренних состояний алгоритма блока. Разбиение на группы можно производить с большой свободой. Сразу же заметим, для дальнейшего анализа будет полезно выделить особое состояние конечного автомата Мили для блока — начальное состояние (далее будет обозначаться словом INITIAL), соответствующее начальным значениям внутренних переменных. Это состояние обладает значительной особенностью, а именно, блок находящийся в нем, фактически не занимает ресурсов памяти вычислительного узла, поэтому может быть просто перенесен на другой вычислительный узел без дополнительных затрат на копирование внутреннего состояния с одного вычислительного узла на другой. Процедуры перемещения состояния алгоритма блока на другой вычислительный узел играют

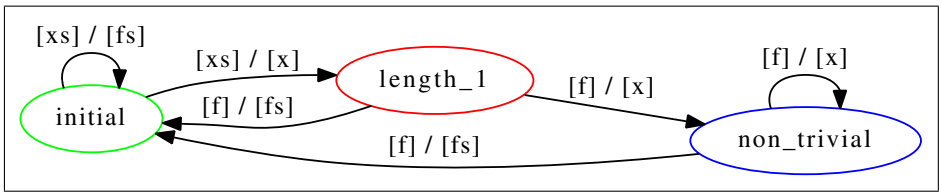


Рис. 4: Избыточный конечный автомат Мили для шаблона блока *For-Loop*

важную роль в управлении запуском потока данных в распределенной среде (более подробно запуск потока данных в распределенной среде будет рассмотрен ниже), так как ожидающий данных блок будет заблокирован (или по, крайней мере, может быть дополнительно ограничен в вычислительных ресурсах), если на вычислительном узле на момент поступления данных первому блоку выполняется алгоритм другого блока, но может быть инициализирован на свободном (если таковые имеются) вычислительном узле, если блок находится в начальном состоянии.

Построение автомата Мили для алгоритма блока — неодназначная задача, требующая понимания качественного поведения алгоритма, а значит участия человека. В данной работе подробно не рассматриваются алгоритмы (которые, по мнению автора, будут бесполезны на практике в силу огромных вычислительных затрат) и методики построения атоматов Мили для алгоритмов блоков. Ограничемся лишь рассмотрение показательного примера с шаблоном блока *For-Loop*. В зависимости от реализации, блок *For-Loop* принимает на вход данные о количестве необходимых операций, для опреденности будем рассматривать шаблон блока, реализующий аналог функции *map* популярной в функциональных языках программирования, которая поэлементно применяет переданную ей в качестве аргумента функцию к списку значений и выдает список результатов. Типичный пример составного блока с блоком шаблона $For - Loop$ показан на рисунке 2a (сам блок шаблона *For-Loop* именуется *map*), а автомат Мили шаблона блока *For-Loop* показан на рисунке 1b. Сам блок шаблона *For-Loop* требует внешнего блока-функции, который подключен к портам x и f. Между вызовами (то есть испусканием очередного элемента списка) блока-функции, блок шаблона *For-Loop* имеет внутреннее состояние, в котором хранит, как минимум, не обработанную часть списка и частичный результат преобразования. Поэтому в конечном автомате Мили шаблона блока *For-Loop* имеются два состояния: *initial* — начальное состояние ожидания данных, *non_trivial* — состояние, в котором результирующий список находится в стадии формирования.

Фактически, для любого алгоритма можно построить автомат Мили с не более, чем двумя состояниями: начальным (или тривиальным) и не тривиальным (с отличным от начального внутренним состоянием) состояниями, однако, в некоторых случаях автомат Мили будет вносить дополнительную неопределенность в поведение блока, не связанную с неопределенностью входных данных при анализе. Возможен и обратный случай — в предыдущем примере с шаблоном блока *For-Loop* можно ввести отдельное состояние, отвечающее за списки длины 1. В данном случае автомат Мили будет выглядеть, как показано на рисунке 4, дополнительное состояние именуется *length_1*. Это дополнительное состояние, конечно, несет новую информа-

цию о текущем состоянии блока, но она абсолютно бесполезна для анализа схемы потока данных.

2.2 Эквивалентность траекторий запуска потока данных ***Need review***

Рассмотрим подробнее правила работы потока данных. В правилах помимо неопределенностей в выборе вариантов поведения, присутствует множество ϕ^t, задаваемое извне. Фактически это множество определяет время работы каждого запущенного блока и введено для возможности описания любой возможной конфигурации времен работы алгоритмов блоков. Однако при анализе потока данных эти времена неизвестны и здесь не будут рассматриваться их оценки. Для анализа большее значения играет не сами времена работы блоков, а зависимости типа "запуск блока A был вызван завершением работы блоков $B_1, B_2, \ldots, B_l$". В дальнейшем, для выяснения всевозможных поведений потока данных в терминах этих зависимостей, мы будем эмулировать запуск потока данных по правилам (1)-(9), в который, в частности, положим времена работы всех блоков равными единице. Модельное время в данном случае отражает лишь один из возможных случаев запуска потока данных и служит только для удобства понимания. Вначале определим понятие траектории запуска потока данных и введем на множестве траекторий запуска отношение эквивалентности.

Определение 10. *Траекторию запуска потока данных* $\Lambda = \{\Lambda^t\}_{t=0}^{\infty}$ *мы определим как семейство всех множеств упомянутых в правилах работы потока данных* (1)-(9)*, то есть* $\Lambda^t = (\xi^t, \theta^t, \chi^t, \ldots)$.

Введем также определения конечной и успешной траекторий и времени работы потока данных на этой траектории.

Определение 11. *Траектория запуска потока данных* $\Lambda = \{\Lambda^t\}_{t=0}^{\infty}$ *называется конечной, если* $\exists T_{end} : \forall t \geq T_{end} : \theta^t = \varnothing$. *Минимальное* T_{end} *называется временем работы потока данных.*

Определение 12. *Траектория запуска потока данных* $\Lambda = \{\Lambda^t\}_{t=0}^{\infty}$ *называется успешной, если* $\exists T_{end} : \forall t \geq T_{end} : \xi^t = B_c \wedge \omega^t = \varnothing$.

Понятно, что минимальные T_{end} из определений **??** и **??** совпадают, так как при отсутствии работающих блоков фронт активной волны ω^t не может меняться (в силу правила работы потока данных (8)), а значит правило (6) не может быть выполнено для заблокированных блоков, так как оно полностью определяется фронтом активной волны. Отсюда же следует, что успешная траектория всегда конечна. В дальнейшем мы будем предполагать, что все траектории рассматриваемого потока данных являются конечными.

Множества π_v^t, $\omega_{I,v}^t$ и $\omega_{O,v}^t$ полностью описывают отношение эквивалентности между траекториями.

Определение 13. *Для каждой конечной траектории запуска множества* π_v^t, I_v^t *порождают граф причинности* $G = (S, E)$. *Для каждого блока пронумеруем все времена его запуска* t_v^n. *Тогда* G *можно определить как:*

- $S = \{(v, n, \pi_v^{t_v^n})\}$,

- *из* $s_1 = (v, \cdot, \pi_1)$ *в* $s_2 = (v, \cdot, \pi_2)$ *есть ребро тогда и только тогда, когда часть поглощенной* $\omega_{I,v}^t$ *в результате перехода* π_2 *была испушена в результате перехода* π_1*; при этом ребро дополнительно помечается множеством соответствующих входных портов блока* v.

Отображение $G_\Lambda = G(\Lambda)$ индуцирует классы эквивалентности траекторий запуска потока данных, иными словами $\Lambda_1 \sim \Lambda_2 \Leftrightarrow G(\Lambda_1) = G(\Lambda_2)$. Отметим, что данное отношение эквивалентности имеет смысл только для успешных траекторий, так как в нем не учитываются остаточные активные волны и заблокированные блоки. Наличие неуспешных траекторий потока данных мы будем рассматривать как ошибку проектирования схемы потока данных. Отметим, что это правило полностью соответствует практике.

2.3 Неопределенности в запуске потока данных ***Need review***

Как было упомянуто выше, в анализе схемы потока данных неизбежно возникают два рода неопределенностей: первый вызван отсутствием знания входных данных при анализе схемы, неопределенности второго рода связанны с самой схемой потока данных, которая может допускать такие неоднозначности как, например, несколько вариантов поглощения данных блоком. Неоднозначности первого рода неизбежны, и поэтому следует обрабатывать все варианты порожденные ими. Все эти неоднозначности второго рода возникают как следствие состояний гонки (race condition), которых традиционно стараются избегать, так как результат выполнения всего потока данных становиться зависим от времен работы блоков, которые, в свою очередь, могут зависеть от данных и параметров вычислительных узлов, что говорит о неправильном построении схемы потока данных.

Сформулируем теперь критерии неопределенностей первого и второго рода.

Любые неопределенности связаны с правилом запуска потока данных (2) — правило запуска блока, что видно из правил работы потока данных, так как, только это правило содержит включения, вместо строгих равенств.

Определение 14. *Будем говорить, что состояние* s *автомата Мили* $FA_v = (S_v, \cdot, \cdot, \cdot, E_v)$ *блока* v *допускает неопределенность первого рода, тогда и только тогда, когда:* $\exists I_v : |\{e \in E_v | e = (s, I_v, \cdot, \cdot)\}| > 1$*, или иными словами, существует набор входных портов* I_v*, такой что, из состояния* s *возможны несколько переходов по входным портам* I_v.

Например, в автомате Мили шаблона блока *For-Loop* (рисунок 1b) каждое состояние допускает неопределенность первого рода. Причины возникновения этих неопределенностей были рассметренны выше.

Определение 15. *Блок* $v \in B_c$ *в состоянии* λ *траектории запуска потока данных на шаге* t *допускает неопределенность второго рода, если:* $|\omega_{I,v}^t| > 1$*, где* $\omega_{I,v}^t$ *определена в правиле* (2) *и содержиться в кортеже* λ.

Определение 16. ***Корректным потоком данных*** *будем называть поток данных, любая возможная траектория которого не допускает неопределенностей второго рода.*

Рассмотрим все возможные траектории запуска потока данных — они образуют префиксное дерево, так как стартуют из одного и того же состояния потока данных. Заметим, что неопределенности первого рода соответствуют ветвлениям этого дерева — все ветви разбиваются на непересекающиеся группы, где каждая группа отвечает варианту перехода блоков, находящихся в состоянии, допускающем неопределенность первого рода. Аналогично с неопределенностями второго рода.

Теорема 1. *Все ветвления префиксного дерева траекторий запуска потока данных вызваны неопределенностями первого и/или второго рода. Если* Σ *— множество префиксов всех траекторий, тогда:*

$$\forall \Lambda \in \Sigma : Z_\Sigma(\Lambda) = {Z'}_\Sigma(\Lambda)$$

$$\begin{aligned} Z_\Sigma(\Lambda) &= |\{\lambda | \Lambda \circ \lambda \in \Sigma\}| \\ Z'_\Sigma(\Lambda) &= \prod_{v \in B_c} \sum_{(\cdot,\cdot,I) \in FA_v(s_v^t, I_v(\omega^t))} N_{\pi,v}(I) N_{\omega,v}(I, \omega^t), \end{aligned}$$

где $|\Lambda| = t$, $N_{\pi,v}(I) = |\{e \in E_v | e = (s, I, \cdot, \cdot)\}|$ *— количество вариантов перехода при поглощении данных из портов* I_v, $N_{\omega,v}(I, \omega^t) = |\Delta_{I,v}(\omega^t)|$ *- количество вариантов поглощения данных по портам* I_v.

Доказательство. Доказательство напрямую следует из того факта, что число состояний λ, в которые можно перейти пройдя траекторию Λ, определяется правилом (2), так как остальные правила запуска потока данных однозначны. Для блока v количество вариантов переходов на шаге t:

$$N_v^t = \sum_{I \in \Xi} \sum_{\pi_I \in \Pi_I} |\Delta_{I,v}(\omega^t)|, \tag{14}$$

где $\Xi = \{I | (\cdot, \cdot, I) \in FA_v(s_v^t, I_v(\omega^t))\}$, $\Pi_I = \{(s_v^t, I, \cdot, \cdot) \in E_v\}$. Учитывая независимость блоков, а так же независимость поглощения данных по портам и переходов в автомате Мили для любого блока, получаем требуемую формулу.

Рассмотрим смысл формулы (**??**). Для каждого блока производиться суммирование по вариантам входных портов для поглощения данных, что вместо с формулой $|\omega_{I,v}^t| = \sum_{I \in \Xi} |\Delta_{I,v}(\omega^t)|$ означает, что данная часть формулы описывает неопределенность второго рода. Суммирование по π_I учитывает неопределенности первого рода. □

2.4 Проверка схемы потока данных на корректность

Далее мы будем рассматривать только схемы потоков данных, все траектории которого успешны.

Ясно, что обработка всех возможных траекторий запуска схемы потока данных алгоритмически невозможна, так как количество всех траекторий

бесконечно. Однако, как уже упоминалось, нам важны лишь классы эквивалентности этих траекторий. В случае с корректными потоками данных нахождение представителей классов эквивалентности возможно при помощи относительно простого алгоритма.

Определение 17. *Корректным семейством траекторий, назовем поддерево префиксного дерева всех траекторий запуска потока данных, ветвления которого суть неопределенности первого рода.*

Теорема 2. *Каждый класс эквивалентности траекторий запуска потока данных содержит корректное семейство траекторий, в котором все времена работы блоков равны* 1*, тогда и только тогда, когда поток данных является корректным.*

Доказательство. □

2.5 Нахождение классов эквивалентности траекторий запуска потока данных

Теорема **??** лежит в основе алгоритма нахождения представителя для каждого класса эквивалентности траекторий и одновременной проверки на корректность потока данных. Заметим, что и в случае некорректного потока данных, можно модифицировать алгоритм, который также вычисляет представителя каждого класса.

»»»> bddb0126f0783b117e40929dd3c5eb8316049c99