Модел. и анализ информ. систем. Т. **18**, № **2** (2011) 18–38

УДК 519.681.2

Об одном представлении функции в модели императивной программы, заданной сетями Петри

Тарасов Г.В., Харитонов Д.И., Голенков Е.А.[1]

Институт автоматики и процессов управления ДВО РАН

e-mail: george@dvo.ru

получена 12 мая 2010

Ключевые слова: модель программы, модель потока управления, теория сетей Петри, объект сети Петри

Рассматривается подход к построению в терминах сетей Петри модели функции как элемента модели программы. Данный подход позволяет представить модель потока управления всей программы в целом как композицию составных моделей потоков управления каждой функции. На основе понятия объекта сети Петри описываются конструкции вызова функции и тела функции, а также описывается операция направленной композиции, отражающая передачу управления из вызывающей функции в вызываемую.

1. Введение

Теория сетей Петри широко применяется для построения моделей программ. Классический подход построения таких моделей заключается в описании потока управления последовательностью срабатывания переходов [1, 2]. Каждому переходу ставят в соответствие выполнение некоторого набора действий программы, а месту — состояние вычислительного процесса, которое он принимает после выполнения одного действия, но до наступления другого действия. Метки в моделях носят характер маркера выполнения вычислительного процесса. Их расположение по местам модели обусловливает выполнение тех или иных действий, поэтому места часто рассматривают как пред- и пост-условия выполнения действий. Наличие условности срабатывания переходов позволяет естественным образом представить поток управления основных алгоритмических конструкций императивных программ: условие, цикл и выбор. Заметим, что модели программ, построенные классическим способом, обладают минимальными выразительными возможностями и не покрывают все множество технологий, предлагаемых современными языками и системами программирования. Например, для объектно-ориентированного языка программирования C++

[1]Работа выполнена в рамках программ Президиума РАН №1 и №2.

основными конструкциями формирования структуры и логики программы являются функция, метод, объект и класс. Естественно, что все это должно находить отражение в соответствующих моделях.

В данной работе рассматривается подход к построению модели функции и вызова функции в императивных программах на языке C с использованием теории композициональных сетей Петри. Предполагается, что исходная программа состоит из множества модулей, каждый из которых состоит из набора функций. В этом множестве выделяется главная функция (main), с которой начинается выполнение программы. Произвольная функция каждого модуля может вызвать любую другую функцию, кроме функции main[2]. Основу модели рассматриваемой программы составляет понятие объекта сети Петри (или СП-объекта), введенное в [3]. Каждая функция, каждый модуль и вся программа в целом представляют собой отдельные СП-объекты. Вызов одного СП-объекта из другого описывает процесс передачи потока управления и моделируется операцией направленной композиции двух СП-объектов. Структура потока управления каждого СП-объекта строится исходя из принципов традиционного подхода и предложений, сформулированных в [4].

Далее в работе дается краткая базовая теория исчисления СП-объектов. После этого описываются принципы моделирования элементов программы: функции и вызова функции. Рассматривается технология построения модели модуля и полной модели программы. В заключении приводится пример параллельной программы и ее модели в терминах СП-объектов.

2. Объект сети Петри и операции над ним

Пусть $A = \{a_1, a_2, ..., a_k\}$ есть некоторое множество. Мультимножеством на множестве A называется функция $\mu : A \to 0, 1, 2, \ldots$, которая ставит в соответствие каждому элементу множества A некоторое неотрицательное целое число. Мультимножества удобно записывать как формальную сумму $n_1a_1+n_2a_2+...+n_ka_k$ или Σn_ia_i, где $n_i = \mu(a_i)$ есть число вхождений $a_i \in A$ в мультимножество. Как правило, при записи суммы её элементы с $n_i = 0$ опускаются. Объединение и пересечение двух мультимножеств $\mu_1 = n_1a_1+n_2a_2+...+n_ka_k$ и $\mu_2 = m_1a_1+m_2a_2+...+m_ka_k$ на множестве A определяется соответственно как $\mu_1+\mu_2 = (n_1+m_1)a_1+(n_2+m_2)a_2+...+(n_k+m_k)a_k$ и $\mu_1 - \mu_2 = (n_1 - m_1)a_1 + (n_2 - m_2)a_2 + ... + (n_k - m_k)a_k$, где последняя операция производится только тогда, когда $n_i > m_i$ для всех $1 \le i \le k$. $\mu_1 \le \mu_2$, если $n_i \le m_i$ для каждого $1 \le i \le k$, и $\mu_1 < \mu_2$, если $\mu_1 \le \mu_2$ и $\mu_1 \ne \mu_2$. Если $n_i = 0$ для всех i, тогда такое мультимножество будет обозначаться как $\mathbf{0}$. Также будем обозначать, что $a \in \mu$, если $\exists n > 0 : (a, n) \in \mu$. Множество всех конечных мультимножеств на множестве A будет обозначаться как $\mathcal{M}(A)$.

Определение 1. *Сеть Петри есть набор* $\Sigma = \langle S, T, {}^\bullet(), ()^\bullet \rangle$, *где*

1. S — *конечное множество мест;*

[2]Вызов функции main из самой себя (или из функций, вызванных из main) не противоречит правилам языка C, однако такие вызовы порождают явную или скрытую рекурсию в потоке управления программы. Непосредственно вопросы рекурсии в этой работе не затрагиваются, хотя предложенный подход позволяет промоделировать этот тип вызова.

2. T — *конечное множество переходов такое, что* $S \cap T = \emptyset$;

3. ${}^\bullet() : T \to \mathcal{M}(S)$ — *входная функция инцидентности;*

4. $()^\bullet : T \to \mathcal{M}(S)$ — *выходная функция инцидентности.*

Мультимножества ${}^\bullet t$ и $t^\bullet$ называются входными и выходными мультимножествами перехода $t \in T$ соответственно.

Далее будет использоваться привычная графическая нотация сетей Петри в виде двудольного ориентированного графа, где места изображаются кружками, а переходы - прямоугольниками. Места и переходы соединяются дугами, представляя входные и выходные функции инцидентности.

Определение 2. *Объект сети Петри (СП-объект или просто объект, для краткости) есть набор* $E = \langle \Sigma, \Gamma, M_0 \rangle$, *где*

1. $\Sigma = \langle S, T, {}^\bullet(), ()^\bullet \rangle$ — *сеть Петри, называемая структурой объекта;*

2. $\Gamma = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ — *множество точек доступа (ТД), каждая из которых имеет вид* $\alpha_i = \langle \Delta_i, \sigma_i \rangle$, *где*

   - α_i — *имя точки доступа* α_i,
   - Δ_i — *некоторый алфавит и*
   - $\sigma_i : T \to \mathcal{M}(\Delta_i)$ — *пометочная функция переходов;*

3. $M_0 \in \mathcal{M}(S)$ — *начальная маркировка.*

Менее формально СП-объект есть сеть Петри, снабженная множеством именованных пометок. Далее в статье СП-объекты обозначаются заглавными буквами E, F, G, а их точки доступа — в квадратных скобках после имени объекта $E[\alpha_1]$, и для СП-объектов используется следующая графическая нотация. СП-объект отображается прямоугольником, а его структура при необходимости отображается внутри прямоугольника. Маркировка изображается фишками, помещенными внутрь мест. Точки доступа объектов отображаются небольшими квадратами на границе прямоугольника. Имя точки доступа ставится рядом с квадратом (при необходимости рядом указывается алфавит точки доступа). Пометка перехода отображается внутри или рядом с ним и представляет собой запись через двоеточие имени точки доступа и слова из ее алфавита. Если из контекста понятно, к какой точке доступа относится пометка, то ее название в пометке перехода не указывается.

Примеры объектов показаны на рисунках 1(а, в). Для ссылки на соответствующие компоненты точки доступа будут использоваться следующие обозначения: $\Delta(\alpha_i) = \Delta_i$ и $\sigma(\alpha_i) = \sigma_i$. Пометка перехода t в точке доступа α будет обозначаться $\alpha(t) \equiv \sigma_\alpha(t)$. Пометочная функция σ расширяется на мультимножество переходов $\sigma : \mathcal{M}(T) \to \mathcal{M}(\Delta)$ следующим образом: если $\Theta = \Sigma n_i t_i \in \mathcal{M}(T)$, то $\sigma(\Theta) = \Sigma n_i \sigma(t_i)$.

Замечание 3. *Об единственности пометки перехода в структуре СП-объекта.*

Далее в статье будут рассматриваться только такие объекты, в структуре которых каждый переход может быть помечен только одной пометочной функцией:

$$\forall t \in T \ : \ (\ \exists! \alpha \in \Gamma \ : \ \alpha(t) \in \mathcal{M}(\Delta(\alpha)) \ ) \ \vee \ \alpha(t) = \boldsymbol{0}.$$

Определим ряд операций над объектами, которые будут использоваться в дальнейшем при построении модели функции.

Определение 4. *Формальное объединение объектов.*

Пусть даны два объекта $E_1 = \langle \Sigma_1, \Gamma_1, M_{01} \rangle$ *и* $E_2 = \langle \Sigma_2, \Gamma_2, M_{02} \rangle$, *где* $\Sigma_1 = \langle S_1, T_1, {}^\bullet()_1, ()^\bullet_1 \rangle$ *и* $\Sigma_2 = \langle S_2, T_2, {}^\bullet()_2, ()^\bullet_2 \rangle$. *Формальным объединением объектов* E_1 *и* E_2 *будем называть объект* $E = E_1 \oplus E_2 = \langle \Sigma, \Gamma, M_0 \rangle, \Sigma = \langle S, T, {}^\bullet(), ()^\bullet \rangle$ *такой, что*

$$S = S_1 \cup S_2, \qquad T = T_1 \cup T_2, \qquad M_0 = M_{01} + M_{02}$$

$${}^\bullet() = {}^\bullet()_1 \cup {}^\bullet()_2, \qquad ()^\bullet = ()^\bullet_1 \cup ()^\bullet_2, \qquad \Gamma = \Gamma_1 \cup \Gamma_2.$$

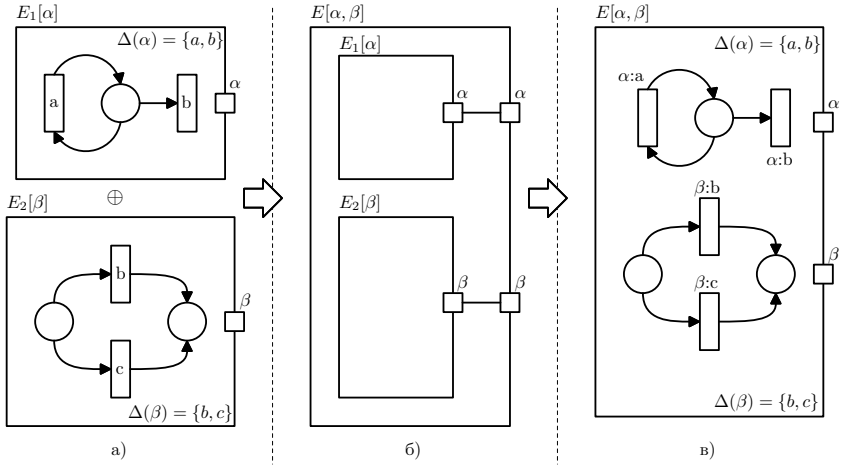

Рис. 1. Формальное объединение объектов: (а) исходные объекты, (б) композиционное представление, (в) результат объединения

Для графического отображения формального объединения объектов внутрь результирующего СП-объекта помещаются исходные СП-объекты. Все точки доступа внутренних объектов дублируются на границе внешнего объекта и соединяются соответствующими линиями. На рисунке 1 показан пример операции формального объединения двух объектов E_1 и E_2.

Определение 5. *Обьединение точек доступа.*

Пусть дан СП-обьект $E_1 = \langle \Sigma_1, \Gamma_1, M_{01} \rangle$, *у которого определено две точки доступа* $\alpha = \langle \Delta_\alpha, \sigma_\alpha \rangle \in \Gamma_1$ *и* $\beta = \langle \Delta_\beta, \sigma_\beta \rangle \in \Gamma_1$. *Тогда при помощи операции обьединения точек доступа* α *и* β *обьекта* E_1 *будет строиться новый СП-обьект* $E = (E_1)_{\gamma=\alpha+\beta}, E = \langle \Sigma, \Gamma, M_0 \rangle$ *такой, что* $\Sigma = \Sigma_1,\ \Gamma = \Gamma_1 \backslash \{\alpha, \beta\} \cup \{\gamma\},\ M_0 = M_{01}$, *где* $\gamma = \langle \Delta_\gamma, \sigma_\gamma \rangle$ *и*

$$\Delta(\gamma) = \Delta(\alpha) \cup \Delta(\beta), \qquad \sigma_\gamma(t) = \sigma_\alpha(t) \cup \sigma_\beta(t).$$

Операция объединения ТД вместо двух точек доступа формирует новую точку доступа, объединяющую их алфавит и функцию пометки. Для графического

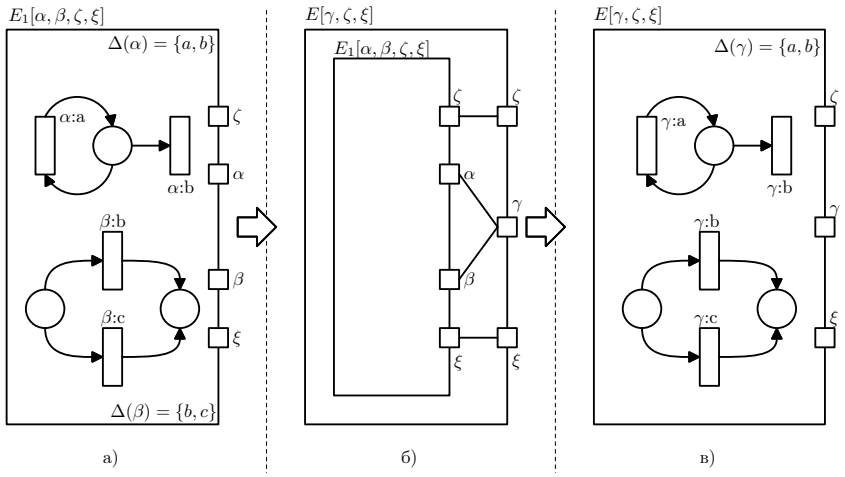

Рис. 2. Объединение точек доступа объекта: (а) исходный объект, (б) композиционное представление объединения ТД α и β, (в) результат объединения

отображения операции объединения точек доступа внутрь результирующего объекта помещается исходный объект с указанием всех его точек доступа. На границу результирующего объекта помещаются все точки доступа, не участвующие в объединении, и новая точка доступа. Исходные и результирующие точки доступа связываются соответствующими линиями. На рисунке 2 показан пример операции объединения точек доступа α и β у объекта E_1.

Определение 6. *Ограничение по точке доступа.*

Пусть дан СП-объект $E_1 = \langle \Sigma_1, \Gamma_1, M_{01} \rangle$, *где* $\Sigma_1 = \langle S_1, T_1, {}^\bullet()_1, ()_1^\bullet \rangle$ *и* $\alpha = \langle \Delta_\alpha, \sigma_\alpha \rangle \in \Gamma_1$. *Ограничением объекта* E_1 *по точке доступа* α *будем называть новый объект* $E = \langle \Sigma, \Gamma, M_0 \rangle = \partial_\alpha(E_1)$, *где* $\Sigma = \langle S, T, {}^\bullet(), ()^\bullet \rangle$ *такой, что*

$$S = S_1,\ T = T_1 \backslash \{t \in T \mid \sigma_\alpha(t) > \boldsymbol{0}\},\ {}^\bullet(t) = {}^\bullet(t)_1, (t)^\bullet = (t)_1^\bullet, t \in T,$$

$$M_0 = M_{01},\ \Gamma = \Gamma_1 \backslash \{\alpha\}.$$

Ограничение объекта по точке доступа α удаляет каждый переход, помеченный именем из $\Delta(\alpha)$, вместе с прилегающими дугами. На рисунке 3 показан пример операции ограничения исходного объекта $E_1[\alpha, \beta]$ по точке доступа α. Для отображения операции ограничения по точке доступа внутрь результирующего СП-объекта помещается исходный СП-объект с указанием всех его точек доступа. Точка доступа, по которой происходит ограничение, отображается закрашенным квадратом, остальные точки доступа выносятся на границу внешнего объекта.

Определение 7. *Простая композиция.*

Пусть задан СП-объект $E_1 = \langle \Sigma_1, \Gamma_1, M_{01} \rangle$ *и его точки доступа* $\alpha, \beta \in \Gamma_1$, *где* $\alpha = \langle \Delta_\alpha, \sigma_\alpha \rangle, \beta = \langle \Delta_\beta, \sigma_\beta \rangle$ *и* $\Sigma_1 = \langle S_1, T_1, {}^\bullet()_1, ()_1^\bullet \rangle$. *Операция простой композиции объекта* E_1 *по точкам доступа* α *и* β *формирует новый объект* $E = \langle \Sigma, \Gamma, M_0 \rangle$, *где* $\Gamma = \Gamma_1, M_0 = M_{01}$ *и* $\Sigma = \langle S, T, {}^\bullet(), ()^\bullet \rangle$:

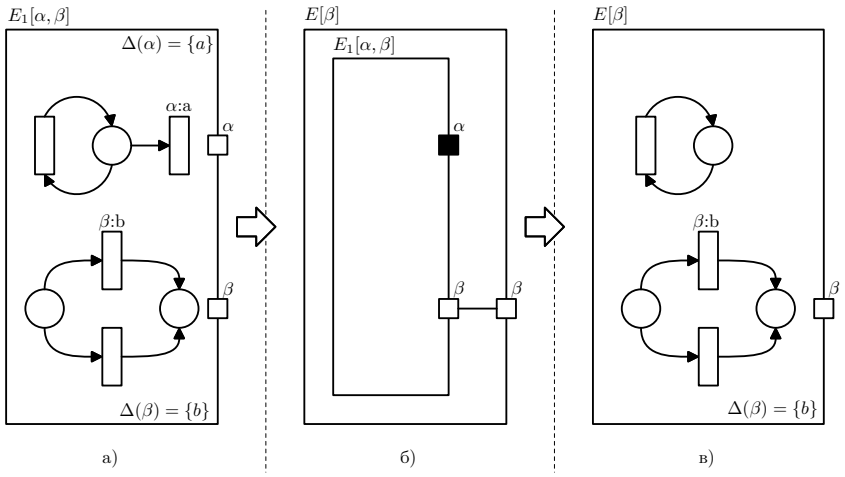

Рис. 3. Ограничение по точке доступа объекта: (а) исходный объект, (б) композициональное представление ограничения по ТД α, (в) результат операции.

1. $S = S_1$,
2. $T = T_1 \cup T_{syn}$, *где* $T_{syn} = \{\mu_1 + \mu_2 \mid \mu_1, \mu_2 \in \mathcal{M}(T_1), \sigma_\alpha(\mu_1) = \sigma_\beta(\mu_2) > \boldsymbol{0}$, *сумма* $\mu_1+\mu_2$ *минимальна, т.е. не существует суммы* $\mu_1'+\mu_2'$ *такой, что* $\mu_1'+\mu_2' < \mu_1+\mu_2$ *и* $\sigma_\alpha(\mu_1') = \sigma_\beta(\mu_2')\}$,
3. ${}^\bullet() = {}^\bullet()_1 \cup \{(\mu_1+\mu_2, {}^\bullet(\mu_1)_1 + {}^\bullet(\mu_2)_1) \mid \mu_1+\mu_2 \in T, \mu_1, \mu_2 \in \mathcal{M}(T_1)\}$,
4. $()^\bullet = ()_1^\bullet \cup \{(\mu_1+\mu_2, (\mu_1)_1^\bullet + (\mu_2)_1^\bullet) \mid \mu_1+\mu_2 \in T, \mu_1, \mu_2 \in \mathcal{M}(T_1)\}$,
5. $\forall t \in T_{syn}\ \forall \xi \in \Gamma\ :\ \xi(t) = \boldsymbol{0}$.

Операция простой композиции для одного (унарная форма) и двух объектов (бинарная форма) обозначается соответственно

$$E = [E_1]_\beta^\alpha, \qquad E = E_1\ {}_\alpha[\ ]_\beta\ E_2 \equiv [E_1 \oplus E_2]_\beta^\alpha.$$

Операция простой композиции в объект E_1 добавляет множество новых переходов синхронизации T_{syn}. Новые переходы задаются мультимножеством символов $\mu_1+\mu_2$, где $\mu_1, \mu_2 \in \mathcal{M}(T)$ и не имеют никакой пометки. Входные и выходные мультимножества данных переходов вычисляются соответственно: ${}^\bullet(\mu_1+\mu_2) = {}^\bullet(\mu_1) + {}^\bullet(\mu_1), (\mu_1+\mu_2)^\bullet = (\mu_1)^\bullet + (\mu_2)^\bullet$.

Определение 8. *Направленная композиция.*

Пусть задан объект $E_1 = \langle \Sigma_1, \Gamma_1, M_{01} \rangle$ *и его точки доступа* $\alpha, \beta \in \Gamma_1$. *Тогда операция направленной композиции (унарная форма) объекта* E_1 *по отношению к* α *и* β *строит новый объект* E *такой, что*

$$E = {}_\alpha\underrightarrow{E_1}_\beta \equiv \partial_\alpha([E_1]_\beta^\alpha).$$

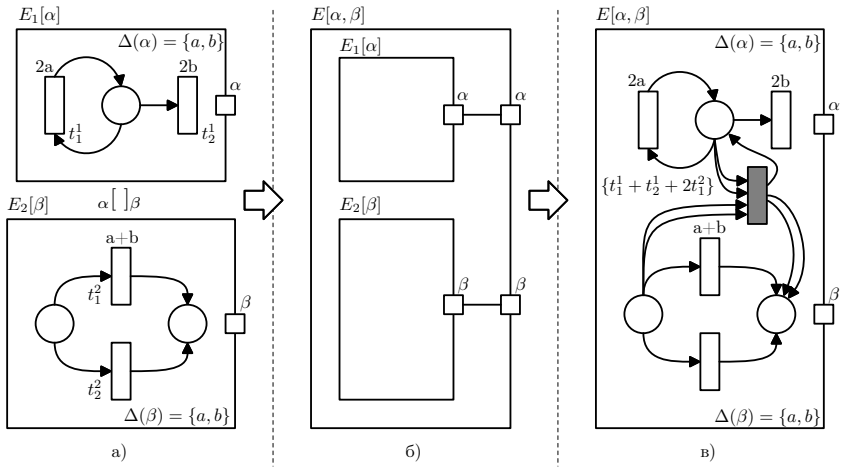

Рис. 4. Пример простой композиции объектов: (а) исходные объекты, (б) композициональное представление, (в) результат простой композиции.

Для двух объектов $E_1 = \langle \Sigma_1, \Gamma_1 \rangle$ *и* $E_2 = \langle \Sigma_2, \Gamma_2 \rangle$ *с точками доступа* $\alpha \in \Gamma_1$ *и* $\beta \in \Gamma_2$ *соответственно операция направленной композиции (бинарная форма) обозначается*

$$E = E_1 \xrightarrow[\beta]{\alpha} E_2 \equiv \partial_\alpha(E_1\ {}_\alpha[\ ]_\beta\ E_2).$$

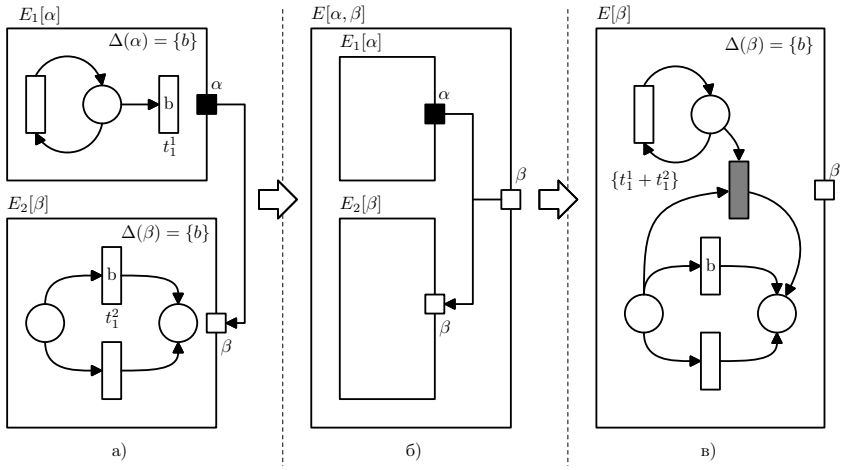

Рис. 5. Пример бинарной операции направленной композиции объектов: (а) исходные объекты, (б) композициональное представление, (в) результат направленной композиции

Из определения видно, что унарная направленная композиция выполняет две операции над объектом: простую композицию по ТД α и β и ограничение по ТД α. Бинарная направленная композиция выражается через унарную, предварительно строится формальное объединение двух исходных объектов.

Для графического отображения операции направленной композиции внутрь результирующего СП-объекта помещаются исходные СП-объекты с указанием тех точек доступа, относительно которых происходит выполнение операции. Направленность операции отображается стрелочкой, а точка доступа, по которой выполняется ограничение, отображается сплошным цветом. Стрелка обозначает направление синхронизации: события в объекте E_1 могут произойти, только если в объекте E_2 есть эквивалентные события. На границе результирующего СП-объекта дублируется точка доступа, в которую входит стрелка операции. Сама точка доступа результирующего объекта соединяется дополнительной линией со стрелкой операции, отображая таким образом источник своего появления в новом объекте. На рисунке 5 показан пример операции направленной композиции двух СП-объектов E_1 и E_2.

Сформулируем некоторые свойства, которыми обладают операции над СП-объектами. Подчеркнем, что выполнение данных свойств справедливо в контексте замечания 3. Для случаев произвольной пометки переходов данные свойства могут не выполняться.

Утверждение 9. *Некоторые свойства операций над СП-объектами.*

1. *Операция формального объединения объектов является коммутативной и ассоциативной:*

$$E_1 \oplus E_2 = E_2 \oplus E_1, \qquad E_1 \oplus (E_2 \oplus E_3) = (E_1 \oplus E_2) \oplus E_3.$$

2. *Операция объединения точек доступа объекта является коммутативной и ассоциативной:*

$$(E)_{\xi=\alpha+\beta} = (E)_{\xi=\beta+\alpha}, \qquad ((E)_{\xi=\alpha+\beta})_{\zeta=\xi+\gamma} = ((E)_{\xi=\beta+\gamma})_{\zeta=\xi+\alpha}.$$

   Доказательства *следуют из коммутативности и ассоциативности операций объединения множеств и мультимножеств и определений 4 и 5.*

3. *Ассоциативность операции ограничения по точке доступа:*

$$\partial_\alpha(\partial_\beta(E)) = \partial_\beta(\partial_\alpha(E)).$$

   Доказательство *следует из определения 6. Поскольку подмножества помеченных переходов при ограничении по α и β не пересекаются, то в обоих случаях получается одинаковое множество T, и, значит, равенство справедливо.*

4. *Для двух объектов $E_1 = \langle \Sigma_1, \Gamma_1, M_{01} \rangle$ и $E_2 = \langle \Sigma_2, \Gamma_2, M_{02} \rangle$ и их точек доступа $\zeta \in \Gamma_1$ и $\alpha, \xi \in \Gamma_2$ справедливо равенство:*

$$E_1 \ {}_\zeta[\ ]_\xi \ \partial_\alpha(E_2) = \partial_\alpha(E_1 \ {}_\zeta[\ ]_\xi \ E_2).$$

   Доказательство *следует из определений 4, 6.*

5. Ассоциативность направленной композиции объектов:

$$E_1 \underset{\gamma}{\overset{\alpha}{\rightarrow}} (E_2 \underset{\gamma}{\overset{\beta}{\rightarrow}} E_3) = E_2 \underset{\gamma}{\overset{\beta}{\rightarrow}} (E_1 \underset{\gamma}{\overset{\alpha}{\rightarrow}} E_3).$$

Доказательство *следует из определений 4 и 6:*

$$E_1 \underset{\gamma}{\overset{\alpha}{\rightarrow}} (E_2 \underset{\gamma}{\overset{\beta}{\rightarrow}} E_3) = \partial_\alpha(E_1\ {}_\alpha[\,]_\gamma\ \partial_\beta(E_2\ {}_\beta[\,]_\gamma\ E_3)) = \partial_\alpha(\partial_\beta(E_1\ {}_\alpha[\,]_\gamma\ (E_2\ {}_\beta[\,]_\gamma\ E_3))) =$$

$$\partial_\beta(\partial_\alpha(E_2\ {}_\beta[\,]_\gamma\ (E_1\ {}_\alpha[\,]_\gamma\ E_3))) = \partial_\beta(E_2\ {}_\beta[\,]_\gamma\ \partial_\alpha(E_1\ {}_\alpha[\,]_\gamma\ E_3)) = E_2 \underset{\gamma}{\overset{\beta}{\rightarrow}} (E_1 \underset{\gamma}{\overset{\alpha}{\rightarrow}} E_3).$$

Выполнение свойств ассоциативности для операций объединения точек доступа и направленной композиции позволяет не учитывать порядок применения этих операций на множестве исходных объектов. В результате для обозначения операции направленной композиции над объектом E_1, у которого множество точек доступа состоит из $\Pi = \{\pi_1, \ldots, \pi_n\}$ и точки доступа α, будем использовать запись вида

$$E = \underbrace{{}_{\pi_n}(\ldots{}_{\pi_1}(E_1)_{\alpha\ldots})_\alpha}_{\longrightarrow} = \underbrace{{}_{\Pi}(E_1)_\alpha}_{\longrightarrow}.$$

3. Модель функции и вызова функции в императивной программе

Используем расмотренный выше математический аппарат для построения модели функции применительно к языкам императивного программирования. В таких языках понятие “функции” часто ассоциируется с традиционным в структурном программировании понятием “подпрограмма”. То есть некоторая последовательность действий основной программы, выделенная для выполнения повторяющихся вычислений. В современных языках программирования терминология, связанная с понятием “функция”, стала достаточно размытой. В разных языках программирования применяют различные синонимы для выражения одной и той же сущности: функция, процедура, метод, подпрограмма и т.п. Несмотря на обилие понятий, а также сопутствующие синтаксические и семантические различия, суть выполнения функции (как подпрограммы) остается одинаковой и состоит в последовательном выполнении следующего набора действий:

1. передача управления в функцию осуществляется с помощью специальной команды “вызов функции”, при этом выполнение команд, следующих за вызовом, временно приостанавливается;

2. команды, составляющие “тело функции”, выполняются до тех пор, пока не встретится специальная команда “возврат из функции”;

3. по завершении выполнения функции, управление возобновляется со следующей за вызовом команды.

Дадим формальные определения моделей основных понятий: вызов функции и функция.

Определение 10. *Модель вызова функции f.*

Пусть задан некоторый СП-объект $E = \langle \Sigma, \Gamma, M_0 \rangle, \Sigma = \langle S, T, {}^\bullet(), ()^\bullet \rangle$ и алфавит $\Delta_f = \{begin_f, end_f\}$. В объекте E определена точка доступа $f \in \Gamma$ такая, что $f = \langle \Delta_f, \sigma_f \rangle$.

Тогда тройку элементов $\langle t_1, s, t_2 \rangle$ в структуре объекта E, состоящую из двух переходов и места $t_1, t_2 \in T$ и $s \in S$, таких, что

1. $t_1 \neq t_2$,
2. $(t_1)^\bullet = s = {}^\bullet(t_2)$, $(s)^\bullet = t_1$, ${}^\bullet(s) = t_2$,

будем называть моделью вызова функции f в контексте точки доступа f, если

$$\sigma_f(t_1) = begin_f, \sigma_f(t_2) = end_f.$$

Менее формально вызовом функции будем называть тройку помеченных и связанных определенным образом элементов сети, где

- первый переход моделирует передачу управления в функцию (помечается $begin_f$), такой переход будем называть *переход вызова*;
- место моделирует состояние ожидания завершения выполнения функции;
- второй переход моделирует возврат управления из функции (помечается end_f), такой переход будем называть *переход возврата*.

Данное определение естественным образом расширяется для описания вызовов одной и той же функции несколько раз и нескольких разных функций в рамках одного СП-объекта. В первом случае в структуре объекта E каждому вызову будет соответствовать другая тройка одинаково помеченных элементов. Во втором случае вызов каждой уникальной функции с именем f_i описывается отдельным алфавитом Δ_i и отдельной точкой доступа $f_i \in \Gamma$. Возможен и общий случай, где в объекте E одной точкой доступа описываются все вызовы всех функций. При этом доопределяются: алфавит Δ_f — соответствующими парами слов для вызова каждой уникальной функции с именем f_i, пометочная функция σ_f — соответствующими отображениями переходов на алфавит пометки. Более наглядно представленные комбинации показаны на рисунке 6.

Определим модель функции, полагая, что внутренняя структура потока управления функции уже описана некоторой сетью Петри.

Определение 11. *Модель функции f.*

Пусть задан СП-объект $E_f = \langle \Sigma_f, \Gamma_f, M_{0f} \rangle$, где $\Sigma_f = \langle S_f, T_f, {}^\bullet()_f, ()_f^\bullet \rangle$ — структура объекта, описывающая поток управления тела функции, и в Γ определено два непересекающихся подмножества точек доступа $(IN_f \cup OUT_f) \subset \Gamma_f, IN_f \cap OUT_f = \mathbf{0}$. $IN_f = \{in_f\}$ — подмножество входных точек доступа, состоящее из одной

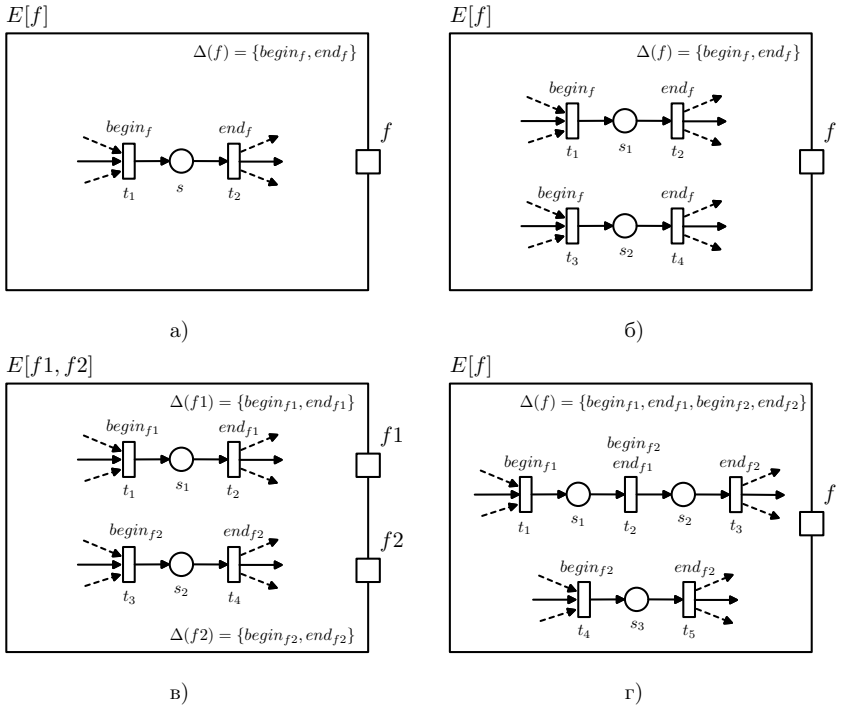

Рис. 6. Примеры описания вызовов функции: (а) случай единственной функции, (б) случай нескольких вызовов одной и той же функции, (в) случай вызова нескольких разных функций, (г) общий случай

точки доступа $in_f = \langle \Delta_f, \sigma_f \rangle, \Delta_f = \{begin_f, end_f\}$. $OUT_f = \{out_1, \ldots, out_n\}$ — *подмножество выходных точек доступа, удовлетворяющих определению 10.*

Если для структуры объекта Σ_f *и точек доступа* $in_f, out_1, \ldots, out_n$ *справедливы следующие утверждения:*

1. $\exists! t_b \in T \ : \ {}^\bullet(t_b) = \boldsymbol{0} \wedge in_f(t_b) = begin_f;$
2. $\forall t \in T_e \ : \ t \neq t_b \wedge (t)^\bullet = \boldsymbol{0} \wedge in_f(t) = end_f$, *где* $T_e \subset T$ *и* $T_e \neq \emptyset;$
3. $\forall out_i \in OUT_f \ : \ out_i(t_k) = begin_i \Longrightarrow {}^\bullet({}^\bullet(t_k)) = t_l \Longrightarrow out_i(t_l) = end_i;$

тогда такой СП-объект E_f *будем называть моделью функции* f.

Утверждения 1–3 в определении 11 обозначают:

1. существует всего один переход t_b, у которого отсутствуют входные дуги, такой переход будем называть *входным переходом*;
2. в структуре объекта существует непустое подмножество переходов T_e, у которых отсутствуют выходные дуги, такие переходы будем называть *выходными переходами*;
3. все выходные точки доступа описывают конструкции вызова других функций из f.

Будем использовать дополнительные обозначения: $In(E_f) = IN_f$ — функция In возвращает множество входных точек доступа СП-объекта, $Out(E_f) = OUT_f$ — функция Out возвращает соответственно множество выходных точек доступа СП-объекта.

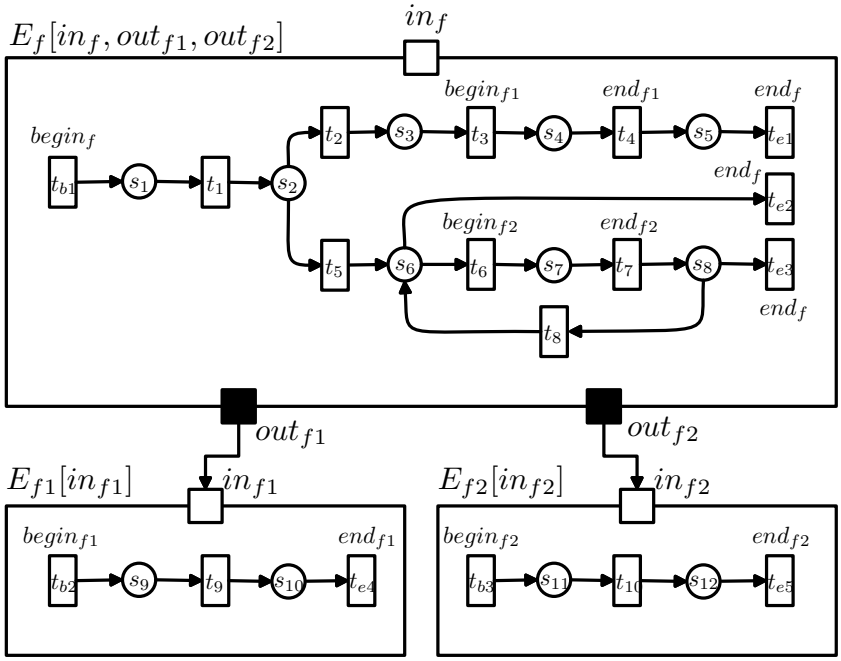

Рис. 7. Примеры моделей функций и вызовов функций

Примеры моделей функций и вызовов функций показаны на рисунке 7. Здесь объектами E_f, E_{f1} и E_{f2} представлены модели трех функций f, $f1$ и $f2$ соответственно. Каждый объект имеет по одной входной точке доступа in_i, через которую управление может быть передано в тело функции, моделируемой объектом. Объект E_f имеет две выходные точки доступа, каждая из которых описывает вызов соответствующей функции $f1$ и $f2$. Связь вызывающей и вызываемой функции определяется операцией направленной композиции по соответствующим точкам доступа. Процесс передачи управления словесно можно описать следующим образом. Предположим, что объект E_f находится в состоянии $M = \{0, 0, 1, 0, 0, 0, 0, 0\}$ (фишка в месте s_3). Тогда в объекте E_f может сработать переход t_3, помеченный $begin_{f1}$. В соответствии с правилами выполнения операции направленной композиции объекты E_f и E_{f1} объединяются, и из переходов t_3 (E_f) и t_{b2} (E_{f1}) формируется новый переход, который убирает фишку из s_3 и помещает по одной фишке в места s_4 (E_f) и s_9 (E_{f1}). Фишка в месте s_4 обозначает ожидание возврата управления из функции E_{f1} (*фишка ожидания*), а фишка в месте s_9 инициирует выполнение тела функции. Фишка ожидания уничтожится, когда сработает переход возврата t_4, который аналогичным образом связан с выходным переходом t_{e4} в объекте E_{f1}.

От описания моделей функции и ее вызова перейдем к моделированию программ. Будем использовать понятие *программный модуль* или просто *модуль* для обозначения некоторого набора функций, описанных в программе. Обобщим определение

модели функции до модуля.

Определение 12. *Модуль*[3].

Модуль есть СП-объект $E = \langle \Sigma, \Gamma, M_0 \rangle$, *для которого определены функции* In *и* Out. *Без ограничения общности считается, что*

$$In(E) \cup Out(E) = \Gamma,\ |In(E)| = 1,\ |Out(E)| \geq 0.$$

Минимальным модулем называется модуль, который состоит только из одной функции.

Менее формально модуль есть объект, который всегда имеет одну *входную точку доступа*, через которую могут быть вызваны компоненты этого модуля, и может иметь ноль и более *выходных точек доступа*, через которые он связан с другими модулями. Модуль, множество выходных точек доступа которого пусто, будем называть *полным модулем*.

Введем операцию композиции модулей как способа конструирования сложных модулей из набора простых. Обозначим для некоторого модуля F подмножество $SELF_F \subset \Gamma_F$ такое, что это множество состоит из всех выходных точек доступа этого модуля, алфавиты которых входят в алфавит входных точек доступа этого модуля:

$$SELF_F = \{out_i \mid out_i \in Out(F)\ \wedge\ \Delta(out_i) \subseteq \Delta(In(F))\}.$$

Точки доступа, принадлежащие множеству $SELF$, будем называть *внутренними точками доступа*.

Определение 13. *Операция композиции модулей.*

Пусть E_1 *и* E_2 *— модули. Тогда операция композиции двух модулей* E_1 *и* E_2 *строит новый модуль* $E = \langle \Sigma, \Gamma, M_0 \rangle$ *такой, что:*

$$E = E_1 \uplus E_2 = {}_{SELF}\underrightarrow{((E_1 \oplus E_2)_{in=in_1+in_2})}_{in},$$

где

- $In(E) = \{in\}$ *— входная точка доступа нового модуля,*
- $Out(E) = (Out(E_1) \cup Out(E_2)) \backslash SELF$ *— множество выходных точек доступа нового модуля,*
- in_1 *и* in_2 *— множества входных точек доступа СП-объектов* E_1 *и* E_2 *соответственно.*

Рассмотрим пример объединения двух модулей (рис. 8) M_1 и M_2, которые имеют по одной входной точке доступа in_1 и in_2 и подмножества выходных точек доступа $\{func, exp, sum\}$ и $\{sum, log\}$ соответственно. Предполагается, что через точку доступа $func$ из первого модуля вызывается некоторая функция во втором модуле. На

[3]Термин модуль хорошо согласуется с такими структурными понятиями языков программирования, как библиотека и функция, с понятиями класс и метод в ООП.

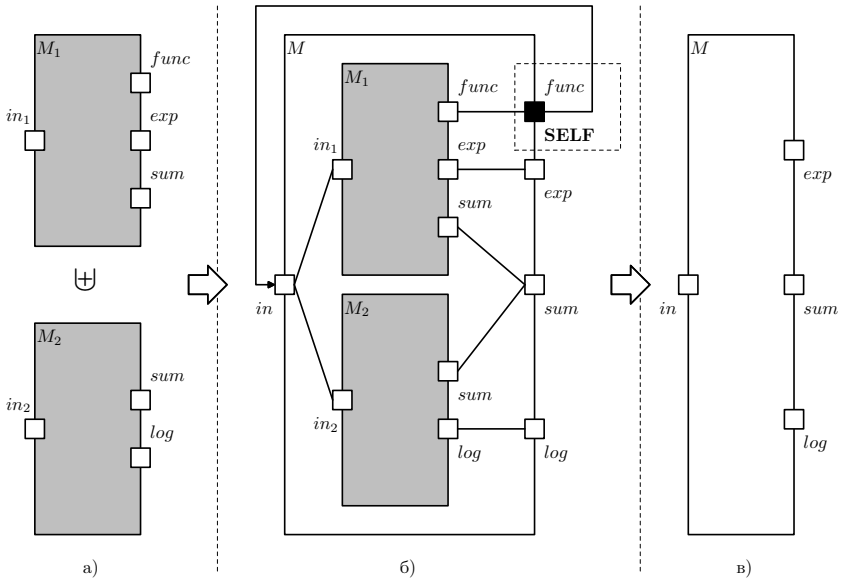

Рис. 8. Пример объединения модулей

рисунке 8(а) показаны исходные модули и их точки доступа. На рисунке 8(б) показана операция композиции исходных модулей, а именно, показаны все преобразования, производимые над исходными объектами: объекты формально объединяются в один объект M и выполняется объединение точек доступа in_1 и in_2, выделяется подмножество внутренних точек доступа $SELF$, состоящее из $\{func\}$, выполняется направленная композиция по точкам доступа из множеств $SELF = \{func\}$ и $In = \{in\}$. В результате (рис. 8(в)) получен новый модуль M, который включает все точки доступа исходных модулей за исключением $func$.

Таким образом, применяя операцию композиции модулей к некоторому набору функций программы, можно получить представление этого набора в виде одного СП-объекта, который содержит одну входную точку доступа для вызова любой функции из набора и некоторое множество выходных точек доступа для вызова внешних функций. Наличие или отсутствие выходных точек доступа у полученного по результатам композиции модуля зависит от степени вложенности вызовов функций, которые необходимо учитывать в конечной модели. Поясним на примере рисунка 8, на котором показано, что исходные модули и результат их композиции используют вызовы некоторых математических функций (sum, exp и log). Здесь может быть два варианта. Первый вариант состоит в том, что функции sum, exp и log считаются такими же функциями, как и все остальные, и необходимо включить их модели в общую модель. Второй вариант состоит в том, что можно рассматривать эти функции как элементарные, примитивные действия с точки зрения модели потока управления всей программы в целом. В таком случае выполнение этих функций можно моделировать одним переходом без использования соответствующих троек элементов, их пометок и точек доступа, и тогда результатом композиции является полный модуль, содержащий только входную ТД. Уровень вложенности зависит от целей моделирования и определяется человеком.

Рассмотрим конечную модель программы, выполнение которой в императивных языках начинается с некоторой *стартовой функции* (точки входа в программу). Прежде чем управление перейдет к этой функции, отрабатывает специальный блок исполняемого файла — *загрузчик*, который компилятор автоматически добавляет в образ исполняемого файла. Роль загрузчика состоит в инициализации системного окружения программы и последующего вызова стартовой функции. В данной статье мы не будем глубоко рассматривать внутреннюю структуру загрузчика, ограничимся лишь последним действием.

Определение 14. *Модель загрузчика.*

Пусть дан СП-объект $E = \langle \Sigma, \Gamma, M_0 \rangle$, *где* $\Sigma = \langle S, T, {}^\bullet(), ()^\bullet \rangle$ *и* $\Gamma = \{out\}$. $out = \langle \delta, \sigma \rangle$ *и* $\delta = \{begin, end\}$. *Будем называть* E *моделью загрузчика, если:*

- $\exists! s_b, s_e \in S \ : \ s_b \neq s_e, {}^\bullet s_b = \boldsymbol{0}, s_e^\bullet = \boldsymbol{0}$,
- $\forall s \in S \ : \ M_0(s) = \begin{cases} 1, & \text{если } s = s_b, \\ 0, & \text{в противном случае,} \end{cases}$
- $\exists! \langle t_1, s, t_2 \rangle, t_1, t_2 \in T, s \in S \ : \ s \neq s_b \neq s_e \wedge (t_1)^\bullet = s = {}^\bullet t_2$ *и* $out(t_1) = begin$ *и* $out(t_2) = end$.

Менее формально загрузчик можно определить как некоторый СП-объект с выделенными начальным s_b и конечным s_e местами, моделирующими соответственно начало и конец выполнения программы, с единственной конструкцией вызова стартовой функции через точку доступа *out* и с начальной маркировкой такой, что фишка находится в начальном месте s_b.

Определение 15. *Модель императивной программы.*

Пусть даны загрузчик E_1 *и полный модуль* E_2 *с точками доступа out и in соответственно такие, что*

- $Out(E_1) = \{out\}, \ In(E_2) = \{in\}, \ out = \langle \Delta_{out}, \ \sigma_{out} \rangle, \ in = \langle \Delta_{in}, \sigma_{in} \rangle$,
- $\Delta_{in} = \Delta_{out} = \{begin, end\}$.

Тогда объект $E = \partial_{in}(E_1 \overset{out}{\underset{in}{\to}} E_2)$ *будем называть моделью императивной программы.*

Пример модели императивной программы в терминах СП-объектов представлен на рисунке 9(в). Операция направленной композиции объектов объединяет эти объекты по точкам доступа. Операция ограничения удаляет оставшуюся точку доступа *in* из модели модуля за ее ненадобностью. В результате получаем объект без точек доступа, структура которого моделирует поток управления программы.

4. Пример модели императивной программы на языке программирования C

Используем рассмотренный выше математический аппарат для построения модели программы, написанной на некотором императивном языке программирования. В

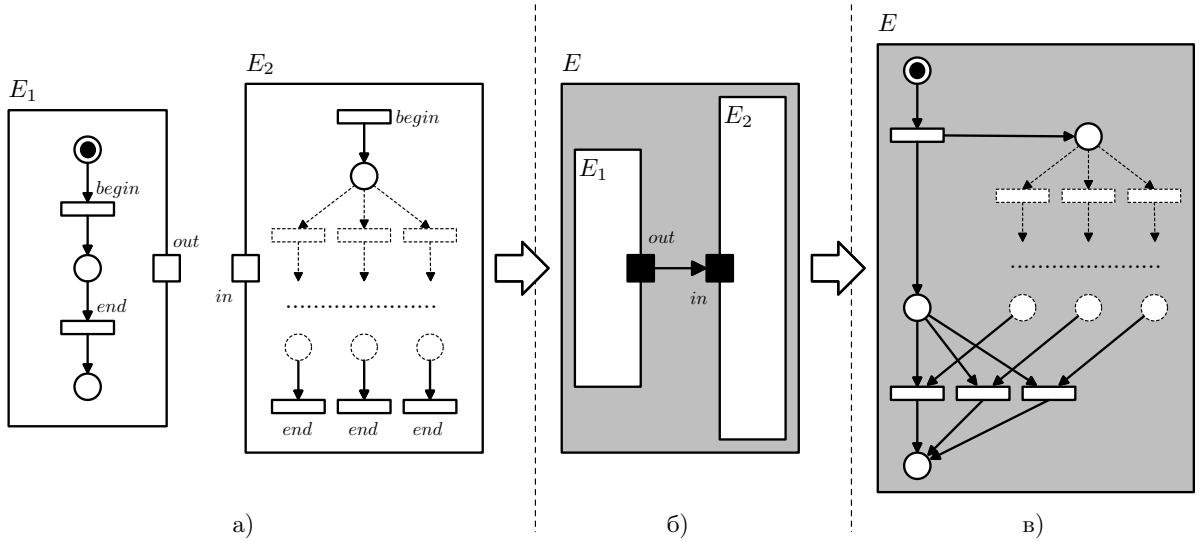

Рис. 9. Пример композиция загрузчика и полного модуля в модель программы.

качестве примера рассмотрим язык C как один из самых распространенных языков программирования общего назначения, предназначенный для решения широкого круга задач — от выполнения простой обработки данных до создания операционных систем. Кратко его можно характеризовать как императивный, структурный язык программирования. Отличительной особенностью языка C является наличие единственной обязательной функции `main`, которая является точкой входа в программу, и с которой начинается выполнение инструкций программы. Помимо функции `main`, программа на C может содержать произвольное[4] количество других функций, имена которых взаимно различны. Вложенность описания функций не допускается, т.е. все функции являются равнозначными и к ним можно обратиться из любого места программы.

Кратко опишем круг понятий, связанных с функцией в языке C. Описание каждой функции состоит из заголовка функции и ее тела. Заголовок функции описывает такие составляющие элементы функции, как имя функции, тип возвращаемого значения и список входных (и выходных) параметров. Описание тела функции следует сразу за описанием заголовка функции. Тело функции представляет собой блок утверждений, содержащий основные алгоритмические конструкции языка: выражение, условие, цикл, выбор, оператор вызова функции, оператор возврата из функции и другие конструкции. Выполнение утверждений осуществляется последовательно до тех пор, пока не будет выполнено последнее утверждение, или не встретится оператор возврата из функции. Передача управления в тело функции осуществляется с помощью конструкции “вызов функции”, которая состоит в выполнении набора действий на уровне системы команд процессора.

На рисунке 10 показана модель потока управления программы разделения множеств. Исходный текст программы представлен в приложении. Модель состоит из

[4]В рамках допустимого аппаратным обеспечением и операционной системой.

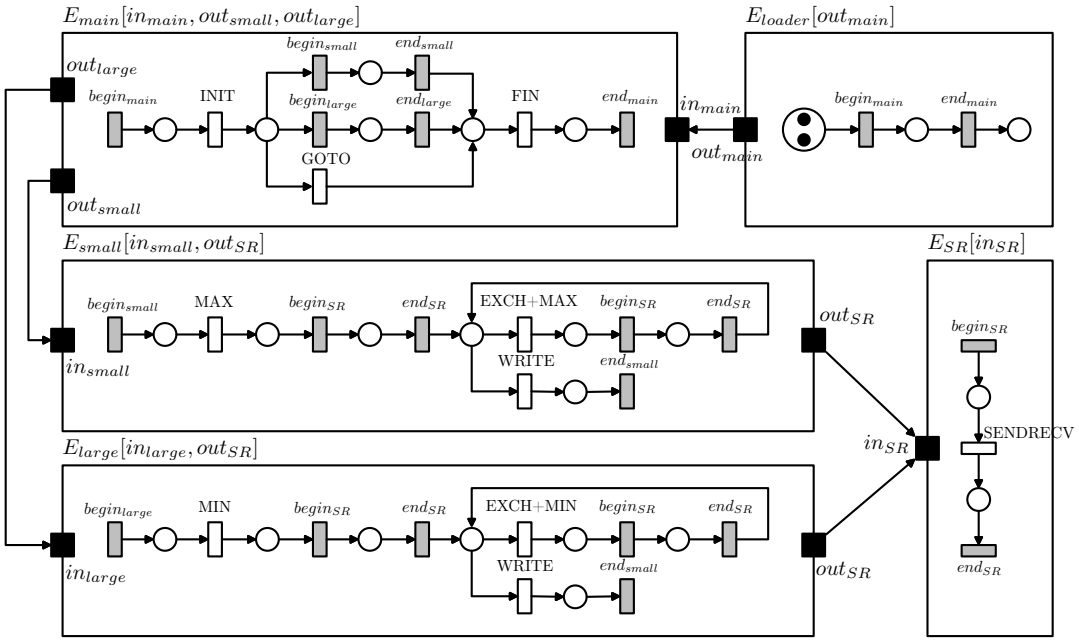

Рис. 10. Модель программы разделения множеств в сетях Петри

пяти СП-объектов. Объект E_{loader} — загрузчик программы. Начальное состояние объекта обозначено двумя фишками. Каждая фишка ассоциируется с одним процессом, который по смыслу программы либо собирает большие элементы двух множеств, либо меньшие элементы. Загрузчик выполняет всего одно действие — вызывает функцию `main` через переходы, помеченные $begin_{main}$ и end_{main} соответственно. Объекты E_{main} моделируют общие действия каждого процесса. Переход с пометкой $INIT$ соответствует строкам 38–40 программы. После инициализации выполняется конструкция выбора, которая соответствует строкам 41–42 программы. Данная конструкция моделирует все возможные варианты дальнейшего исполнения процессов, а именно либо передачу управления объекту E_{small} (вызов функции `Small`), либо передачу управления объекту E_{large} (вызов функции `Large`), либо выполнение фиктивного действия с пометкой $GOTO$, которое возможно, если ни одно из условий не будет выполнено. С точки зрения использованной нотации простых сетей Петри любая из фишек может возбудить любой из переходов, помеченных $begin_{small}$, $begin_{large}$ или $GOTO$. Однако же, положим, что этот выбор может быть каким-либо образом детерминирован и одна фишка будет возбуждать переход $begin_{small}$, а другая фишка — переход $begin_{large}$. Соответственно конструкции вызова функций внутри выбора передадут управление в объекты E_{small} и E_{large}. Данные объекты имеют схожие по структуре потоки управления и выполняют однотипные наборы действий, отличающиеся пометкой. Строки программы 17–26 соответствуют структуре обекта E_{small}, строки 27–36 — структуре объекта E_{large}. Оба объекта имеют несколько конструкций вызова общей функции взаимодействия `SendRecv`, поток управления которой моделируется объектом E_{SR}. Правила выполнения перехода $SENDRECV$ в данном примере не регламентируются. Предполагается, что обе фишки (оба процесса)

возбуждают его в соответствии с общими правилами срабатывания переходов. Синхронизацию процессов можно описать с использованием конструкции вызов функции и включением в модель еще одного СП-объекта, который будет моделировать структуру выполнения функции `MPI_SendRecv`.

5. Заключение

В данной работе предложен способ построения модели потока управления императивной программы в виде композиции моделей потоков управления составляющих ее функций. На основе понятия СП-объекта описаны конструкции вызов функции и тело функции, определена формальная операция направленной композиции объектов, позволяющая из композиционального представления получить развернутое конечное представление потока управления в виде одной сети Петри.

Композициональное представление потока управления в виде моделей отдельных функций обладает рядом существенных преимуществ по отношению к конечному представлению в виде одной модели. Во-первых, частично упрощается проблема взрыва числа состояний [7], а именно многократно уменьшается количество состояний в модели за счет того, что отсутствуют дублирующие состояния одной и той же функции, вызываемой из разных мест программы. При этом стек вызванных функций в данный момент времени отражают дополнительные места, участвующие в описании вызова функции. Во-вторых, упрощается процедура автоматической генерации модели по исходному коду программы, каждой функции в программе соответствует отдельный СП-объект, связанный с другими объектами набором формальных описаний. И, в-третьих, модульность на уровне модели предоставляет дополнительную гибкость на стадии анализа модели. На входе соответствующих алгоритмов анализа может быть задано не только полное композициональное или полное развернутое представление модели, но и задана выборочная комбинация некоторого подмножества функций, реализующих некоторую логическую часть или компоненту всей программы в целом.

Полученные результаты позволяют надеяться на дальнейшее успешное развитие представленного подхода. В частности, в работе не затронуты вопросы описания рекурсивных функций, хотя при беглом рассмотрении соответствующий аппарат СП-объектов и операций над ними вполне пригодны для композиционального представления рекурсии в императивных программах. Не рассмотрено описание виртуальных функций в объектно-ориентированных языках программирования, связанных с типизацией объектов во время исполнения. Также требуют отдельного исследования вопросы описания данных и способов передачи параметров в функцию и обратно.

6. Приложение

Исходный текст программы разделения множеств на языке C, рассмотренной в примере.

```
1   #include <mpi.h>
2   #include <stdio.h>
3   #include <stdlib.h>
4   #include "set.h"
5
6   int rank;
7   Set SET;
8   #define SMALL_PROC 0
9   #define LARGE_PROC 1
10
11  void SendRecv(int dst, int in, int& out) {
12      MPI_Status status;
13      MPI_Sendrecv(&in, 1, MPI_INT, dst, 100,
14                   &out, 1, MPI_INT, dst, 100,
15                   MPI_COMM_WORLD, &status);
16  }
17  void Small() {
18      int mx, x;
19      mx = SET.max();
20      SendRecv(LARGE_PROC, mx, x);
21      while (mx > x) {
22          SET - mx; SET + x; mx = SET.max();
23          SendRecv(LARGE_PROC, mx, x);
24      }
25      SET.Write("example.small.txt");
26  }
27  void Large() {
28      int mn, y;
29      mn = SET.min();
30      SendRecv(SMALL_PROC, mn, y);
31      while (mn < y) {
32          SET - mn; SET + y; mn = SET.min();
33          SendRecv(SMALL_PROC, mn, y);
34      }
35      SET.Write("example.large.txt");
36  }
37  int main(int argc, char* argv[]) {
38      MPI_Init(&argc, &argv);
39      MPI_Comm_rank(MPI_COMM_WORLD, &rank);
40      SET.Init(argc, argv);
41      if (rank == SMALL_PROC) Small();
42      else if (rank == LARGE_PROC) Large();
43      MPI_Finalize();
44      return 0;
45  }
```

Список литературы

1. Котов В.Е. Сети Петри. М.: Наука. Главная редакция физико-математической литературы, 1984. 160 с.

2. Питерсон Дж.Л. Теория сетей Петри и моделирование систем: Пер. с англ. М.: Мир, 1984. 264 с., ил.

3. Анисимов Н.А., Голенков Е.А., Харитонов Д.И. Композициональный подход к разработке параллельных и распределенных систем на основе сетей Петри // Программирование. 2001. №6. С. 30–43.

4. Голенков Е.А., Соколов А.С. Метод автоматического построения модели параллельной программы в терминах сетей Петри // Вычислительные методы и программирование. 2005. Т. 6, №2. С. 77–82.

5. Jensen K. Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1, Basic Concepts. Monographs in Theoretical Computer Science, Springer-Verlag, 2nd corrected printing, 1997.

6. Ломазова И.А. Вложенные сети Петри: моделирование и анализ распределенных систем с объектной структурой. М.: Научный мир, 2004. 208 с.

7. Кларк Э.М., Грамберг О., Пелед Д. Верификация моделей программ: Model Checking. Пер. с англ. / Под ред. Р. Смелянского. М.: МЦНМО, 2002. 416 с.: ил.

On a Function Representation in an Imperative Program Model Specified by Petri Nets

Tarasov G.V., Kharitonov D.I., Golenkov E.A.

Keywords: program model, control flow model, Petri net object

In the article an approach to constructing in terms of Petri nets a function model as a program model unit is considered. This approach makes it possible to present a program control flow model in a whole as a composition of constituent control flow models of each function. In the article constructions of function call and function body are introduced on the base of Petri net object, and then an operation of directed composition of Petri net object is defined that represents a control flow transfer between caller and called functions.

Сведения об авторах:

Тарасов Георгий Витальевич,

Институт автоматики и процессов управления ДВО РАН, научный сотрудник.

Научные интересы: параллельное программирование, архитектура распределенных систем, операционные системы, GRID-технологии.

Харитонов Дмитрий Иванович,

Институт автоматики и процессов управления ДВО РАН, научный сотрудник.

Научные интересы: параллельное программирование, распределенные системы, информационные системы, локальные и глобальные вычислительные сети.

Голенков Евгений Александрович,

Институт автоматики и процессов управления ДВО РАН,
старший научный сотрудник;
Дальневосточный федеральный университет,
доцент кафедры программного обеспечения ЭВМ.

Научные интересы лежат в области компьютерных сетей, параллельных вычислений, распределенных систем.