Министерство образования и науки
Российской Федерации
Федеральное агентство по образованию
Ярославский государственный университет
им. П. Г. Демидова
Кафедра теоретической информатики

В. А. Башкин

Модели потоков работ

Методические указания

*Рекомендовано Научно-методическим советом
университета для студентов, обучающихся
по специальности
Прикладная математика и информатика*

Ярославль 2009

УДК 004.436.4; 004.75
ББК 3973.2–018.1я73
Б 33

Рекомендовано
Редакционно-издательским советом университета
в качестве учебного издания. План 2009 года

Б 33 **Башкин, В. А.** Модели потоков работ: метод. указания / В. А. Башкин; Яросл. гос. ун-т. им. П. Г. Демидова — Ярославль: ЯрГУ, 2009. — 43 с.

В методических указаниях излагаются принципы моделирования и анализа потоков работ (workflow) при помощи сетей Петри. Рассматриваются основные элементы и свойства моделей распределенных систем и процессов.

Предназначены для магистрантов факультета информатики и вычислительной техники ЯрГУ, обучающихся по специальности 010500.68 Прикладная математика и информатика (маг.), очной формы обучения (дисциплина «Модели распределенных систем», блок СД).

Библиогр.: 18 назв.

УДК 004.436.4; 004.75
ББК 3973.2–018.1я73

Оглавление

Предисловие

Потоки работ (workflow)

Существует много различных видов работ, например, выпечка хлеба, производство автомобилей, разработка архитектурных проектов или опрос людей для сбора статистических данных. Во всех этих примерах мы можем выделить какой-нибудь один осязаемый “предмет”, который производится или изменяется: хлеб, автомобиль, дом, статистический отчет. Такой “предмет” мы будем называть экземпляром работы или просто экземпляром. В качестве синонимов могут использоваться такие термины, как работа, продукт или услуга. Экземпляр не обязан быть конкретным объектом; он может быть и более абстрактным, как, например, судебный процесс или страховое требование. Строительный проект или сборка одного автомобиля на заводе также являются примерами экземпляра работы.

Работа с экземпляром дискретна по своей природе. Каждый экземпляр работы имеет начало и конец и может быть отделен от любого другого. Каждый экземпляр вызывает выполнение процесса. Процесс состоит из нескольких задач, которые должны быть выполнены, и набора условий, которые определяют порядок выполнения задач. Процесс можно также называть процедурой. Задача — это логическая единица работы, выполняемая как единое целое одним ресурсом. Ресурс — это общее название для человека, машины, а также группы людей или машин, которые выполняют специфические задачи. Ресурс не обязательно выполняет свою задачу независимо, но он всегда ответственен за нее.

В качестве примера процесса рассмотрим работу со страховыми требованиями некоторой вымышленной страховой компании (пример взят из книги [1]). От клиента компании поступила заявка (страховое требование) на выплату возмещения по факту возникновения какого-то страхового случая (пожар, ДТП, наводнение, ...). Страховая компания должна рассмотреть эту заявку

и как-то на неё отреагировать (выплатить возмещение или отказать). В этой работе мы можем выделить следующие задачи:

1) *регистрация* получения требования;

2) *определение типа* требования (например, пожар, путешествие, транспортная, профессиональная страховка);

3) проверка *полиса* клиента для подтверждения того, что он соответствует типу требования;

4) проверка *взносов* клиента для подтверждения того, что платежей достаточно;

5) *отказ*, если задачи 3 или 4 дают отрицательный результат;

6) подготовка *письма об отказе*;

7) оценка *выплачиваемой суммы*, исходя из деталей требования;

8) назначение *оценщика* для выяснения обстоятельств ущерба и определения его объема;

9) рассмотрение *экстренных мер* для ограничения дальнейшего ущерба или уменьшения потерь;

10) обеспечение *экстренных мер*, если их необходимость определена задачей 9;

11) определение или пересмотр *выплачиваемой суммы* и информирование об этом клиента;

12) регистрация *реакции* клиента: согласие или возражение;

13) анализ *возражения* и решение о пересмотре выплат (задача 11) или о рассмотрении дела в суде (задача 14);

14) рассмотрение дела в суде;

15) *выплата* требования;

16) *закрытие* требования (отправка дела в архив).

В этом примере выделены шестнадцать задач, которые не обязательно должны выполняться в указанном порядке. Две или несколько задач, которые должны быть выполнены строго в указанном порядке, называются *последовательностью*. В некоторых случаях отдельные задачи могут вообще не выполняться. Примером такого рода задачи является назначение эксперта. Если требование не вызывает сомнений и запрашиваемая сумма ниже определенного значения, то эксперта привлекать не нужно. Другие примеры необязательных задач: принятие экстренных мер, анализ возражений или рассмотрение дела в суде. В некоторых случаях необходимо выполнить только одну из двух или большего числа задач. Это называется *выбором*. Возможны также ситуации, когда некоторые задачи могут выполняться *параллельно*. В нашем примере это проверка полиса и проверка взносов. При этом обе указанные задачи должны быть обязательно закончены до начала выполнения задачи "отказ". Это называется *синхронизацией*. Наш пример процесса содержит также *итерацию* (или повторение) — анализ возражений и пересмотр выплачиваемой суммы могут выполняться повторно. Теоретически такое повторение может продолжаться до бесконечности.

На рисунке 1 последовательность выполнения задач представлена в виде диаграммы процесса: стрелка от задачи A к задаче B означает, что A должна выполняться раньше, чем B. Легко заметить, что диаграмма содержит больше информации, чем список задач. Например, на ней показано, что требование может быть закрыто только при условии, что все необходимые экстренные меры уже были приняты.

На диаграмме каждая отдельная задача изображается прямоугольником. Прямоугольник с чертой сверху — первая задача

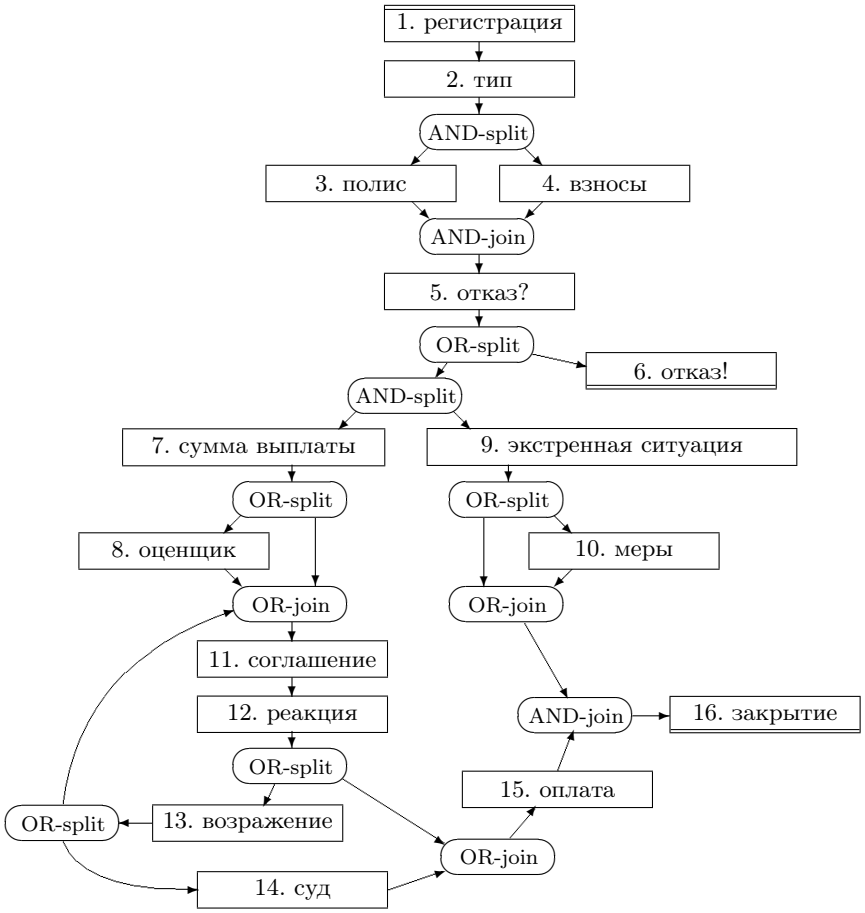


Рис. 1: Процесс обработки страхового требования

процесса (*регистрация*), прямоугольники с чертой снизу — заключительные задачи (*отказ!* и *закрытие*).

Овалы обозначают способы ветвления и соединения отдельных нитей процесса. Ветвления вида "AND-split" соответствуют разбиению процесса на параллельные подпроцессы, каждый из которых должен быть выполнен до конца. Последующая синхронизация обозначается слиянием "AND-join". Это слияние произойдет только после завершения всех предшествующих подпроцессов. Таким образом, задача 5 начнёт выполняться только после завершения и задачи 3, и задачи 4.

Ветвления вида "OR-split"[1] соответствуют выбору ровно одного из нескольких возможных путей развития процесса. Следовательно, после задачи 5 может выполниться или задача 6, или параллельный вызов задач 7 и 9. Дополняющей структурой к ветвлению "OR-split" является слияние "OR-join", функции которого очевидны из названия.

Подводя итог, можно выделить четыре основные конструкции в структуре процессов: последовательность, выбор, параллелизм и итерацию. Все они часто встречаются на практике, и, в принципе, все процессы могут моделироваться с помощью этих четырех конструкций.

Корректность схем потоков работ

Задача верификации схем потоков работ появилась сравнительно недавно, с возникновением информационных систем, управляющих процессами. Такие системы называются Системами Управления Потоками Работ (СУПР, по аналогии с СУБД). В них в качестве программы используется написанная на каком-то графическом языке схема процесса (наподобие той, которая изображена на рисунке 1), в соответствии с которой ядро СУПР

[1] Фактически термин "OR-split" означает здесь "XOR-split", то есть не "ИЛИ", а "исключающее ИЛИ". Однако мы и в дальнейшем будем называть его "OR-split", следуя стандартной терминологии.

(workflow engine) автоматически маршрутизирует отдельные задачи конкретных экземпляров процесса. В частности, запускает и останавливает механизмы и приложения, пересылает документы исполнителям, отслеживает тайм-ауты и т. д.

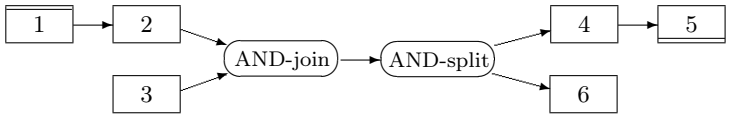
Очевидно, что необходимым условием успешности работы такой системы является корректность заложенной в неё схемы процесса (например, отсутствие тупиков). На рисунке 2 показаны некоторые распространенные ошибки, которые могут возникать при определении процесса:

1. *Задачи без условий на входе и/или выходе.* Если для задачи не заданы входные условия, то не понятно, когда ее можно будет выполнить. Если же не определены условия на выходе, то это значит, что задача не вносит вклад в успешное завершение работы с экземпляром, и, следовательно, может быть опущена. В ситуации A имеется одна задача (задача 3) без входного условия и одна задача (задача 6) без условия на выходе.

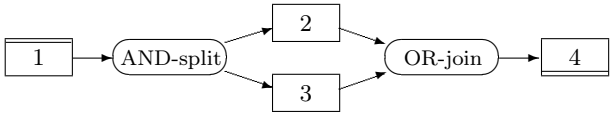
2. *Заблокированные задачи*: задачи, которые никогда не могут быть выполнены. Очевидно, что процесс, содержащий заблокированные ("мертвые") задачи, является нежелательным. В ситуациях B и C задача 4 никогда не может быть выполнена.

3. *Тупик* (deadlock): "зависание" экземпляра до того, как было достигнуто условие завершения процесса. Если в ситуации E на первом шаге выбор будет сделан в пользу задачи 2, то экземпляр процесса завершится успешно (в итоге выполнится задача 7). Если же вместо задачи 2 будет выполнена задача 3, то процесс попадет в тупик: для выполнения И-слияния не будет хватать результата задачи 5, и задача 7 никогда не сможет быть запущена.

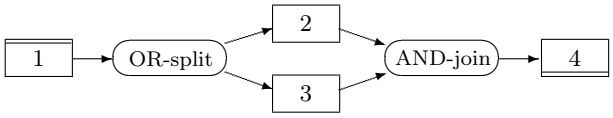
4. *Активный тупик* (livelock): попадание экземпляра в бесконечный цикл. В ситуации D каждый экземпляр будет оста-



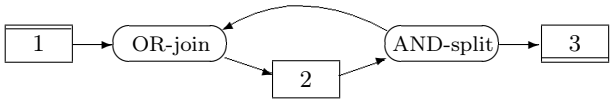
A) Задачи без входа и без выхода



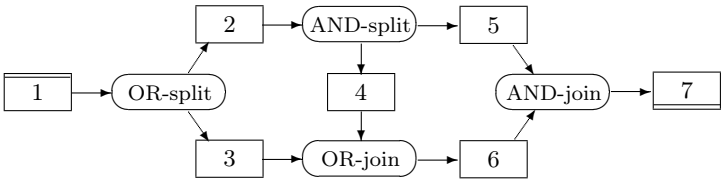
B) Распараллеливание без синхронизации



C) Синхронизация без распараллеливания



D) Бесконечный цикл



E) Смешивание выбора и распараллеливания

Рис. 2: Примеры некорректных ситуаций

ваться до бесконечности в цикле, состоящем из выполнения задачи 2. Другими словами, в этой ситуации имеется итеративный маршрут без возможности выхода из цикла.

5. *После достижения условия завершения продолжается выполнение некоторых задач.* Хорошее определение процесса имеет явно обозначенное начало и конец. После того, как условие завершения достигнуто, ни одна задача больше не должна выполняться. В ситуации C задача 2 может сработать после достижения условия завершения (срабатывания задачи 3). Такая ситуация, конечно, нежелательна.

Приведенные примеры показывают, что, даже не зная ничего о реальном содержании процесса, можно по структуре определения процесса выявлять некоторые типичные ошибки. Эти ошибки связаны с маршрутизацией экземпляров. Для того чтобы можно было обнаруживать такие ошибки с помощью компьютерных программ, нужно точно определить понятие корректности.

В данных методических указаниях рассматривается формальный подход к проверке корректности потоков работ, основанный на хорошо зарекомендовавшем себя формализме для моделирования и анализа процессов — сетях Петри. Использование формального подхода дает ряд существенных преимуществ. Во-первых, оно требует точных определений и, в отличие от многих неформальных схем, исключает двусмысленности, неопределенности и противоречия. Во-вторых, формализм может использоваться для обоснования процессов, что позволяет, например, создавать запас образцов корректных и эффективных процессов. Все это напрямую связано с тем обстоятельством, что формальное описание, как правило, дает возможность применять аналитические методы как для анализа исполнения, так и для верификации логических свойств процессов. Как будет показано далее, с помощью аналитических методов удается проверить, действительно ли данный

экземпляр процесса успешно завершается по истечении некоторого периода времени.

Сети Петри были введены в 1962 году Карлом Адамом Петри [14] как инструмент для моделирования и анализа процессов. Этот формализм позволяет представлять потоки работ в ясной и доступной форме. Несмотря на то что сети Петри являются графическим средством, они имеют строгое математическое обоснование. В отличие от многих других графических представлений, сети Петри полностью формализованы. Благодаря существованию такого базиса мы можем формулировать строгие математические утверждения о свойствах моделируемых процессов. Имеются также всевозможные методы и инструменты, которые могут использоваться для анализа заданной сети Петри.

За прошедшие годы модель, предложенная Карлом Адамом Петри, расширялась в различных направлениях. Это позволяет моделировать сложные процессы доступным образом.

1. Обыкновенные сети Петри

1.1. Мультимножества

Понятие мультимножества является естественным обобщением понятия множества. В мультимножестве один и тот же объект может находиться в нескольких экземплярах.

Пусть X — непустое множество.

Определение 1.1. *Мультимножеством* M над множеством X называется функция $M : X \to Nat$.

Мощность мультимножества $|M| = \sum_{x \in X} M(x)$.

Числа $\{M(x) \mid x \in X\}$ называются коэффициентами мультимножества, коэффициент $M(x)$ определяет число экземпляров элемента x в M. Если $\forall x \in X \ M(x) \leq 1$, то M является обычным множеством.

Мультимножество M *конечно*, если конечно множество

$$\{x \in X \mid M(x) > 0\}.$$

Множество всех конечных мультимножеств над данным множеством X обозначается как $\mathcal{M}(X)$.

Операции и отношения теории множеств естественно расширяются на конечные мультимножества.

Определение 1.2. Пусть $M_1, M_2, M_3 \in \mathcal{M}(X)$. Полагаем:

- $M_1 = M_2 \ \Leftrightarrow \ \forall x \in X \ M_1(x) = M_2(x)$ — отношение равенства;
- $M_1 \subseteq M_2 \ \Leftrightarrow \ \forall x \in X \ M_1(x) \leq M_2(x)$ — отношение включения;
- $M_1 \subset M_2 \ \Leftrightarrow \ M_1 \subseteq M_2 \ \wedge \ \exists x \in X \ M_1(x) \leq M_2(x)$ — отношение строгого включения;

- $M_1 = M_2 + M_3 \Leftrightarrow \forall x \in X\ M_1(x) = M_2(x) + M_3(x)$ — операция сложения двух мультимножеств;
- $M_1 = M_2 \cap M_3 \Leftrightarrow \forall x \in X\ M_1(x) = min(M_2(x), M_3(x))$ — операция пересечения двух мультимножеств;
- $M_1 = M_2 - M_3 \Leftrightarrow \forall x \in X\ M_1(x) = M_2(x) \ominus M_3(x)$ — разность двух мультимножеств (где $\ominus$ — вычитание до нуля);
- $M_1 = kM_2,\ k \in Nat \Leftrightarrow \forall x \in X\ M_1(x) = kM_2(x)$ — операция умножения мультимножества на скаляр.

Пример 1.1. Рассмотрим M_1 и M_2 — мультимножества над множеством $X = \{a, b, c\}$, такие, что $M_1 = \{a, a, b\}$, $M_2 = \{b, c\}$.

Выполняется:

$M_1(a) = 2,\ M_1(b) = 1,\ M_1(c) = 0;$

$M_1 \not\subset M_2,\ M_1 \not\supset M_2;$

$M_1 + M_2 = \{a, a, b, b, c\};$

$M_1 \cap M_2 = \{b\};$

$M_1 - M_2 = \{a, a\};$

$2M_1 = \{a, a, a, a, b, b\}.$

Одним из способов записи мультимножеств являются векторы над *Nat*. При этом различным координатам вектора сопоставляются различные элементы X.

Пример 1.2. Мультимножества M_1 и M_2 из примера 1.1 могут быть записаны как $M_1 = (2, 1, 0)$ и $M_2 = (0, 1, 1)$.

1.2. Сети Петри

Определение 1.3. *Обыкновенной сетью Петри* (ordinary Petri Net) называется набор $N = (P, T, F)$, где
P — конечное множество позиций;
T — конечное множество переходов, $P \cap T = \emptyset$;
$F : (P \times T) \cup (T \times P) \to Nat$ — функция инцидентности.

Графически сеть Петри изображается как двудольный ориентированный граф. Вершины-позиции изображаются кружками и характеризуют локальные состояния сети, вершины-переходы изображаются прямоугольниками и соответствуют действиям моделируемой системы. Дуги в графе (стрелки) соответствуют элементам F. В сетях Петри допустимы кратные дуги, которые обозначаются соответствующим количеством стрелок.

Определение 1.4. Пусть $N = (P, T, F)$ — обыкновенная сеть Петри. *Разметкой* (состоянием) сети N называется функция вида $M : P \to Nat$, сопоставляющая каждой позиции сети некоторое натуральное число или ноль.

Разметка может рассматриваться как мультимножество над множеством позиций сети, то есть $M \in \mathcal{M}(P)$.

Графически разметка изображается при помощи маркеров (называемых “фишками”) — черных точек внутри позиций. При разметке M в каждую позицию p помещается ровно $M(p)$ фишек. Если не хватает места на рисунке, то вместо точек в позиции рисуется число $M(p)$.

Определение 1.5. *Маркированной (размеченной) сетью Петри* называется пара (N, M_0) — сеть Петри N вместе с некоторой выделенной разметкой M_0, называемой начальной разметкой.

Определение 1.6. Пусть $N = (P, T, F)$ — сеть Петри.

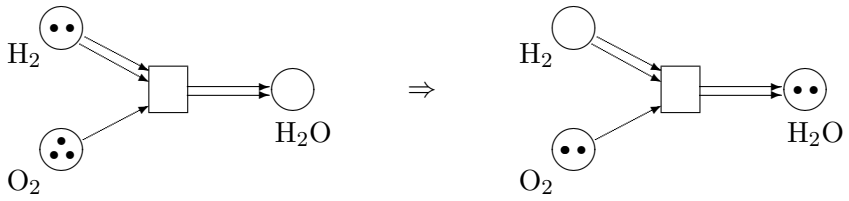


Рис. 1.1: Сеть Петри, моделирующая химическую реакцию

- Для перехода $t \in T$ через ${}^\bullet t$ и $t^\bullet$ обозначим мультимножества его входных и выходных позиций, такие, что

$$\forall p \in P \quad {}^\bullet t(p) =_{def} F(p,t),\ t^\bullet(p) =_{def} F(t,p).$$

- Переход $t \in T$ *готов к срабатыванию* при разметке M, если ${}^\bullet t \subseteq M$ (все входные позиции содержат достаточное количество фишек).

- Готовый к срабатыванию переход t может *сработать*, порождая новую разметку $M' =_{def} M - {}^\bullet t + t^\bullet$ (используется обозначение $M \xrightarrow{t} M'$).

Фишки, находящиеся в той или иной позиции, моделируют наличие в системе того или иного ресурса, используемого или порождаемого при срабатывании переходов. Например, в сети на рисунке 1.1 фишки изображают молекулы водорода, кислорода и воды до и после химической реакции синтеза воды. Сама реакция моделируется переходом.

Определение 1.7. Пусть (N, M_0) — маркированная сеть Петри. Разметка M сети N называется *достижимой*, если существует последовательность переходов $\sigma \in T^*$, переводящая сеть из начального состояния M_0 в состояние M:

$$M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \ldots \xrightarrow{t_k} M,\ t_1.t_2.\ldots.t_k = \sigma,$$

что обозначается как $M_0 \xrightarrow{\sigma} M$ или просто как $M_0 \to M$.

Множество всех достижимых разметок (состояний) маркированной сети обозначается как $R(N, M_0)$.

На рисунках 1.2–1.4 приведены примеры того, как при помощи обыкновенных сетей Петри можно моделировать некоторые элементы реальных систем.

На рисунке 1.2 изображен буфер ограниченной емкости. Независимо от поведения сети количество фишек в позиции *буфер* не превысит двух. При этом количество фишек в служебной позиции *свободно* показывает, сколько еще "места" осталось в буфере. Предложенная структура сети универсальна — мы можем изменять моделируемую емкость буфера, просто увеличивая количество фишек в начальной разметке позиции *свободно*. В начальном состоянии буфер пуст, на входе имеются три фишки "данных".

Сеть 1.3 моделирует систему разделения доступа для двух различных процессов к общей ячейке памяти. Чтобы исключить одновременный доступ процессов к памяти (чтение и запись в данном случае не различаются), использовано классическое семафорное решение. Имеется один общий ключ (моделируется фишкой в позиции *ключ*), который дает право на доступ. Пока работающий процесс не вернул его на место, ожидающий процесс не может перейти в рабочее состояние. В начальном состоянии оба процесса находятся в стадии ожидания доступа.

На рисунке 1.4 показан элемент памяти FIFO (очередь), состоящий из двух ячеек (позиции 1 и 2). Позиция 1 моделирует первую ячейку FIFO (куда поступают данные), позиция 2 — последнюю (откуда они забираются). В начальном состоянии системы обе ячейки свободны, на входе имеются три фишки "данных".

Сети Петри достаточно выразительны для того, чтобы моделировать все виды маршрутизации процесса (рисунок 1.5).

Следующая теорема ("свойство монотонности сетей Петри") отражает важнейшую конструктивную особенность сетей Петри

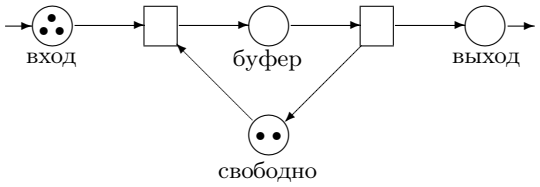


Рис. 1.2: Буфер объема 2

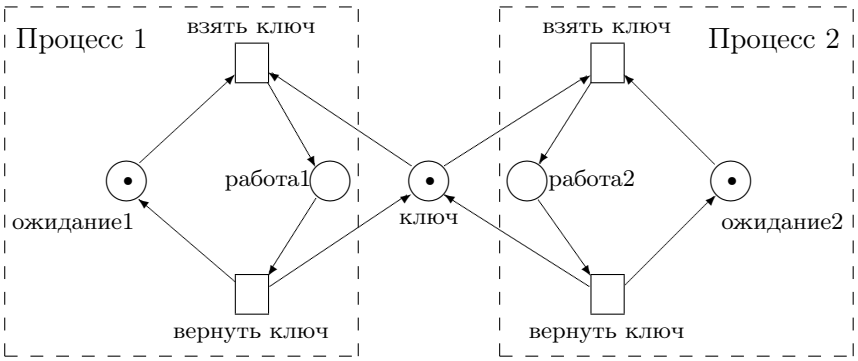


Рис. 1.3: Семафор (разделенный доступ к памяти)

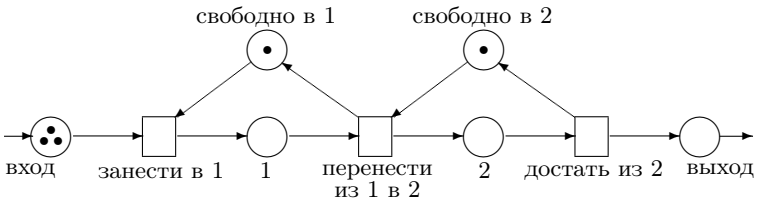


Рис. 1.4: Очередь (FIFO) из двух ячеек

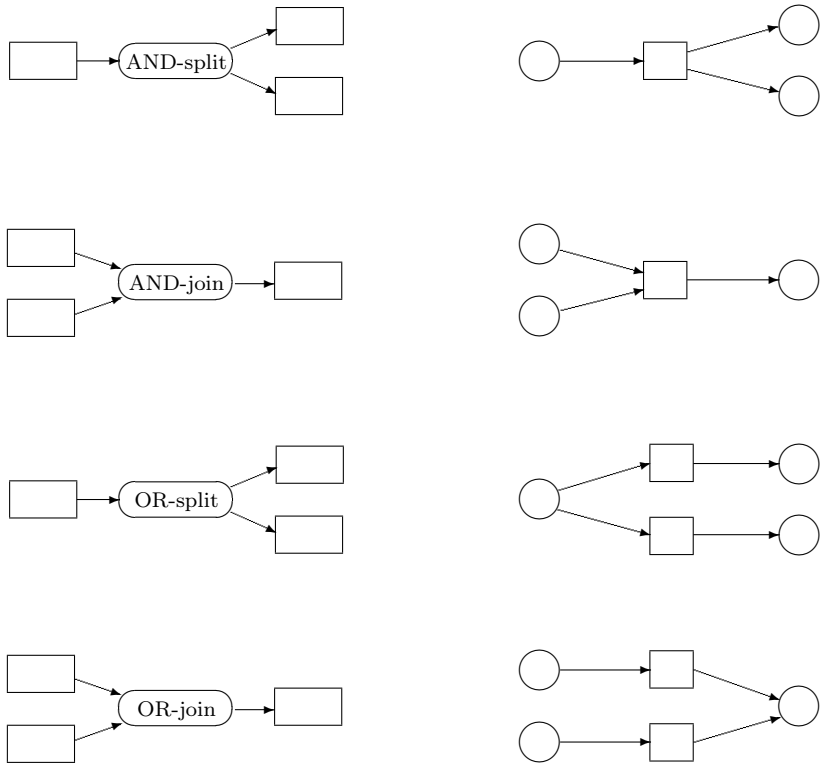


Рис. 1.5: Моделирование разветвлений и слияний сетями Петри

— способность сохранять старое поведение при простом увеличении начальной разметки.

Теорема 1.1. *Пусть $N = (P, T, F)$ — обыкновенная сеть Петри, $M, K \in \mathcal{M}(P)$ — некоторые разметки, $\sigma \in T^*$ — последовательность переходов. Тогда*

1. $M \xrightarrow{\sigma} M' \quad \Rightarrow \quad (M + K) \xrightarrow{\sigma} (M' + K)$;
2. $M' \in R(N, M) \quad \Rightarrow \quad (M' + K) \in R(N, M + K)$.

Доказательство теоремы можно найти, например, в [4], § 1.2.

Свойство монотонности позволяет структурировать множество достижимых состояний при помощи естественного частичного порядка на множестве разметок сети — отношения вложенности мультимножеств. Подобная хорошая структурированность пространства состояний (которой нет, например, в машинах Тьюринга) приводит к разрешимости ряда ключевых алгоритмических свойств. В частности, для сетей Петри построены алгоритмы проверки свойств останова и достижимости.

Определение 1.8. Сеть Петри $N = (P, T, F)$ называется *сетью со свободным выбором*, если для любых двух переходов t_1 и t_2 из того, что ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \varnothing$ следует, что ${}^\bullet t_1 = {}^\bullet t_2$.

Сети Петри со свободным выбором проще обыкновенных сетей. Они обладают рядом интересных свойств. В частности, многие их свойства можно проверять при помощи полиномиальных алгоритмов (а не экспоненциальных, как в случае обыкновенных сетей). Определить то, что сеть Петри являются сетью со свободным выбором, можно непосредственно по её определению, последовательно перебрав множество пар переходов и проверив для каждой пары их предусловия.

Термин “свободный выбор” в данном случае обладает следующей интерпретацией: если два перехода в сети зависят от какого-то одного ресурса, то они равноправны в его использовании и не

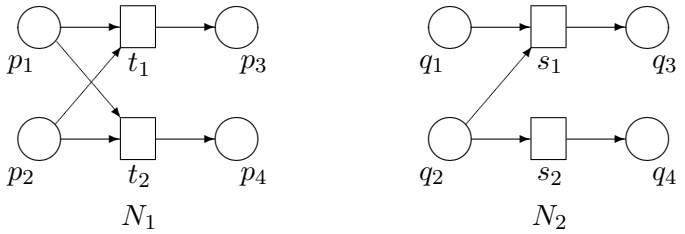


Рис. 1.6: Сети Петри со свободным выбором (N_1) и без свободного выбора (N_2)

зависят от каких-то других ресурсов. То есть сам это ресурс может быть "уверен" в том, что если в данный конкретный момент времени его может забрать первый переход (t_1 активен), то и второй может сделать то же самое (t_2 тоже активен). Выбор между переходами абсолютно свободный.

В сети N_1 на рисунке 1.6 переходы t_1 и t_2 конфликтуют из-за фишек в позициях p_1 и p_2. Но при этом они равноправны, то есть не зависят от каких-то дополнительных (различных) ресурсов. В сети N_2 переходы s_1 и s_2 также конфликтуют из-за одной фишки в позиции q_2. Но здесь нет свободы выбора, так как переход s_2 находится в "привилегированном" положении, поскольку не зависит больше ни от каких фишек. Переход s_2 зависит ещё и от разметки позиции q_1.

1.3. Методы анализа

Рассмотрим некоторые свойства систем и процессов, которые могут быть формализованы при помощи сетей Петри.

Пусть (N, M_0) — маркированная сеть Петри.

Определение 1.9. Позиция $p \in P$ называется *ограниченной*, если $\exists n \in Nat$, такое, что $\forall M \in R(N, M_0)$ выполняется $M(p) \leq n$.

Маркированная сеть называется *ограниченной*, если все её позиции ограничены.

Другими словами, количество фишек в ограниченной позиции ограничено. Очевидно, что множество состояний ограниченной сети конечно, то есть ограниченная сеть эквивалентна конечному автомату. Например, представленная на рисунке 1.3 модель семафора является ограниченной сетью, то есть мы можем проверить её свойства простым перебором достижимых состояний.

Определение 1.10. Позиция $p \in P$ называется *безопасной*, если $\forall M \in R(N, M_0)$ выполняется $M(p) \leq 1$. Маркированная сеть называется *безопасной*, если все её позиции безопасны.

Понятие безопасности является сужением понятия ограниченности. Безопасные позиции можно рассматривать как логические переменные, принимающие одно из двух возможных значений.

Определение 1.11. Маркированная сеть называется *консервативной*, если $\exists n \in Nat$, такое, что $\forall M \in R(N, M_0)$ выполняется $|M| = n$.

Другими словами, в консервативной сети общее количество фишек (во всех позициях) постоянно и не изменяется в ходе её функционирования. Очевидно, что любая консервативная сеть является ограниченной, то есть консервативность — это сужение понятия ограниченности сети.

Определение 1.12. Переход $t \in T$ называется *живым*, если $\forall M \in R(N, M_0)\ \exists M' \in R(N, M) \ : \ {}^\bullet t \subseteq M'$. Маркированная сеть называется *живой*, если все её переходы живы.

Другими словами, живой переход всегда остаётся потенциально активным (то есть активным если не в данном состоянии, то в каком-то из достижимых от данного). В живой сети невозможны состояния, при которых какие-нибудь переходы становятся избыточными. Все так или иначе всегда смогут сработать.

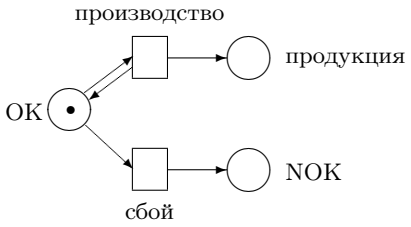


Рис. 1.7: Модель ненадёжной производящей системы

Пример 1.3. Система на рисунке 1.3 ограничена, безопасна, не консервативна и жива.

Определение 1.13. Переход $t \in T$ называется *мертвым*, если $\forall M \in R(N, M_0)\ {}^\bullet t \not\subseteq M$.

Другими словами, мертвый переход не активен при всех достижимых разметках. Обратите внимание, что из не-мертвости перехода не следует его живость. Переход не может быть одновременно и живым и мертвым, однако существуют переходы, которые не являются ни мертвыми, ни живыми.

Пример 1.4. Система на рисунке 1.7 не ограничена и не жива. Неограниченность, и, следовательно, небезопасность и неконсервативность объясняются неограниченностью позиции *продукция* (так как переход *производство* может сработать любое количество раз). В то же время сеть не жива, так как в любой момент может один раз сработать переход *сбой*, после чего в системе не останется ни одного активного перехода.

Данная сеть моделирует простейшую систему, в которой сбой непоправим и приводит к прекращению работы. Легко видеть, что небольшие модификации сети позволяют получить модели для более сложных случаев, например, для систем с восстановлением после сбоев, систем с резервным производством, систем с журналированием сбоев и т. д.

Определение 1.14. Сеть Петри N называется *хорошо сформированной*, если существует состояние M такое, что сеть (N, M) — живая и ограниченная.

Пусть (N, M_0) — размеченная сеть Петри. Рассмотрев срабатывания всех переходов сети, активных при разметке M_0, мы можем построить конечное множество разметок, достижимых от M_0 за один шаг. В свою очередь, от каждой из новых разметок мы можем построить все достижимые за один шаг разметки, и так далее. Результатом этого процесса будет представление множества достижимых разметок в виде дерева. Узлы дерева соответствуют разметкам, а дуги — срабатываниям переходов, преобразующим одну разметку в другую.

Получаемое дерево в общем случае будет бесконечным, ведь сеть может содержать циклы и/или быть неограниченной. Чтобы ограничить бесконечный рост дерева, в обозначения разметок позиций добавляют специальный символ ω, соответствующий "неограниченному" (неисчерпаемому, бесконечному) числу фишек. При этом стандартные операции и отношения переносятся на расширенное множество натуральных чисел $Nat \cup \{\omega\}$ следующим образом: для любого натурального n полагаем

$$\omega > n,\ \omega + n = \omega,\ \omega - n = \omega \ \text{ и } \ \omega \geq \omega.$$

Пример 1.5.
$(2, \omega, 1, \omega) > (2, \omega, 1, 1000)$;
$(3, 5, \omega) + (2, 1, 0) = (5, 6, \omega)$;
$(10, \omega) - (2, 1000) = (8, \omega)$.

Алгоритм 1.1. *(полное покрывающее дерево сети Петри)*

1. *Выберем в качестве корня начальную разметку M_0, пометим этот узел признаком "новый".*

2. *Пока существуют "новые" узлы, выполним следующее:*

2.1. Выберем один из таких узлов — разметку M.

2.2. Если M совпадает с какой-то другой разметкой, находящейся на пути от корня к M, то сменим признак узла M на "старый" и вернемся на шаг 2.

2.3. Если при разметке M ни один из переходов не активен, то сменим признак узла M на "тупик" и вернемся на шаг 2.

2.4. Для каждого перехода t, активного при разметке M, выполним следующие шаги:

2.4.1. Вычислим разметку M', получающуюся из M при срабатывании t (то есть $M' = M - {}^\bullet t + t^\bullet$).

2.4.2. Просмотрим весь путь в дереве от корня до M. Если на этом пути встретилась разметка M'', такая что $M'' \subseteq M'$ (то есть M' покрывает M''), то для каждой позиции p, такой что $M''(p) < M'(p)$, заменим в разметке M' координату $M'(p)$ на символ ω. То есть все позиции со строго увеличившимся количеством фишек обозначим как "неисчерпаемые".

2.4.3. Добавим в дерево узел M' с признаком "новый" и дугу от M к M' с именем t.

2.5. Сменим признак узла M на "старый".

Доказательство сходимости этого алгоритма можно найти, например, в [4], § 2.2.

На рисунке 1.8 приведен пример построения полного покрывающего дерева сети Петри. Здесь обычная рамка узла означает признак "старый", двойная — признак "тупик".

Построение полного покрывающего дерева позволяет выяснить ряд свойств сети Петри:

- Позиция p *ограничена* тогда и только тогда, когда ни в одном из узлов дерева она не помечена символом ω.

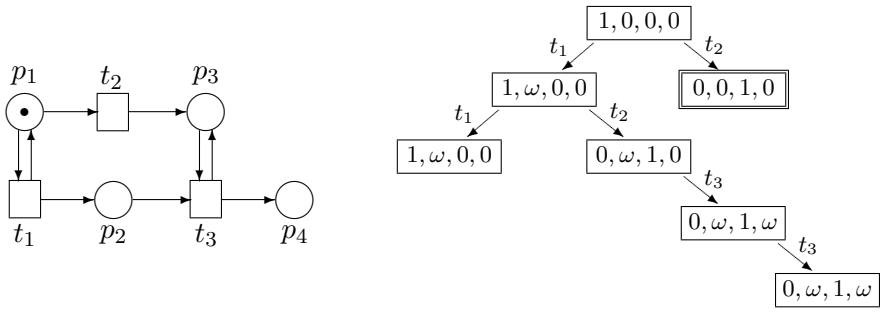


Рис. 1.8: Построение полного покрывающего дерева

- Позиция p *безопасна* тогда и только тогда, когда во всех узлах дерева она помечена нулем или единицей.
- Сеть *консервативна* тогда и только тогда, когда общее количество фишек в любом узле дерева совпадает с общим количеством фишек в корне.
- Переход t *мёртв* тогда и только тогда, когда им не помечена ни одна из дуг в дереве.
- Если разметка M достижима от начальной, то в дереве найдётся узел M', такой что $M \subseteq M'$ (при этом, возможно, часть компонент в M' будет обозначена ω).

Пример 1.6. В сети, изображенной на рисунке 1.8, позиции p_1 и p_3 ограничены и безопасны. Сеть не консервативна, ни один из переходов не является ни живым, ни мёртвым. Примеры покрываемых разметок: $(1, 50, 0, 0), (0, 100, 1, 5000)$.

Обратите внимание, что *достижимость* конкретной разметки в сети Петри нельзя определить через полное покрывающее дерево. Решается только более слабое свойство *покрываемости*, то есть достижимости какой-то разметки, покрывающей данную

(при этом алгоритм не дает метода точного нахождения наименьшей из покрывающих разметок).

Можно показать, что проблема живости перехода сводится к проблеме достижимости разметки ([4], §2.4). Проблема достижимости — одна из ключевых в теории сетей Петри. В настоящее время её разрешимость доказана [13], однако алгоритм проверки весьма сложен и поэтому здесь не приводится. Заметим, что для упрощённых сетей Петри (например, для сетей с не более чем двумя неограниченными позициями или для сетей со свободным выбором) разработаны достаточно простые и эффективные алгоритмы решения проблем достижимости и живости.

Отдельно остановимся на случае ограниченных сетей Петри. Для таких систем проблема достижимости разметки и проблема живости перехода могут быть легко разрешены при помощи полного покрывающего дерева:

- В ограниченной сети Петри разметка M *достижима* от начальной тогда и только тогда, когда в дереве найдётся узел, помеченный M.
- В ограниченной сети Петри переход t *жив* тогда и только тогда, когда выполняются следующие два свойства:
  1. В дереве нет узлов (листьев) с признаком “тупик”.
  2. Если соединить дугами все различные узлы дерева с одинаковыми разметками, то в полученном графе в каждом цикле найдётся дуга с пометкой t.

Итак, в случае ограниченных сетей полное покрывающее дерево превращается в конечный граф достижимости, по которому можно установить практически любые свойства системы. Очевидно, что этот граф можно рассматривать как диаграмму переходов некоторого недетерминированного конечного автомата. Отсюда легко можно сделать вывод о том, что ограниченные сети Петри эквивалентны конечным автоматам.

2. Сети потоков работ

2.1. Определение и свойства

Далее мы приводим определение специального подкласса сетей Петри, который используется при моделировании потоков работ (workflow) — это так называемые WF-сети [1].

Определение 2.1. Пусть $N = (P, T, F)$ — обыкновенная сеть Петри. Сеть N называется *WF-сетью* (сетью потока работ), если

1) в множестве P имеются две специальные позиции i и o, такие, что ${}^\bullet i = o^\bullet = \emptyset$;

2) любой элемент множества $P \cup T$ лежит на пути из i в o.

Позиция i называется *начальной*, а позиция o — *финальной* позицией сети N. Альтернативные термины — позиция-источник и позиция-сток. Начальная разметка сети потока работ состоит из одной фишки в позиции i (и обозначается как i).

Пример 2.1. На рисунке 2.1 изображена сеть Петри, моделирующая описанный в предисловии процесс обработки страховых требований. По сути дела, модель получена добавлением в схему процесса позиций (промежуточных, а также одной начальной и одной финальной) и нескольких служебных переходов (организующих распараллеливание и передачи управления).

2.2. Бездефектность (soundness)

Правильное завершение процесса, моделируемого сетью, гарантируется выполнением следующего свойства:

Определение 2.2. WF-сеть $N = (P, T, F)$ называется *бездефектной* (sound), если

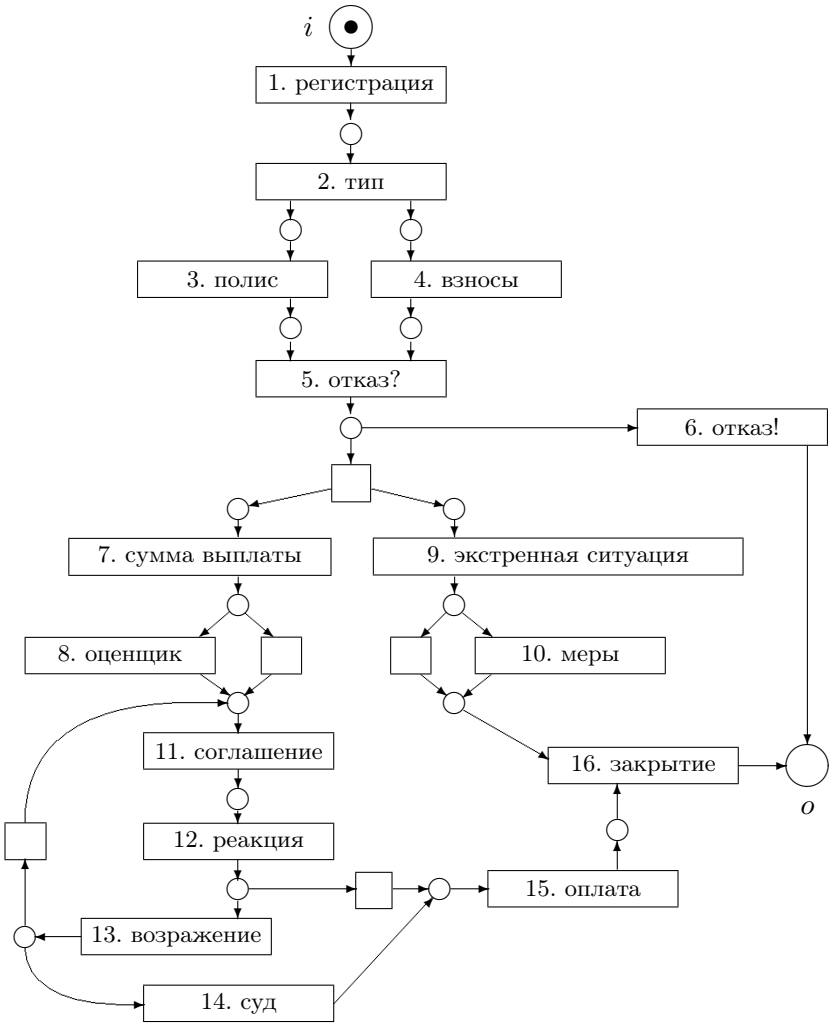


Рис. 2.1: Формальная модель потока работ процесса обработки страхового требования

1. Для любого состояния M, достижимого из состояния i, существует последовательность срабатываний, переводящая состояние M в состояние o:

$$\forall M \in R(N,i) \quad o \in R(N,M).$$

2. Состояние o является единственным состоянием, которое достижимо из состояния i и содержит хотя бы одну фишку в позиции o:

$$o + M' \in R(N,i) \;\Rightarrow\; M' = \emptyset.$$

3. В сети (N,i) нет мертвых переходов:

$$\forall t \in T \; \exists M \in R(N,i) : \; {}^\bullet t \subseteq M.$$

Другими словами, из любого достижимого состояния бездефектной сети достижимо финальное состояние, при этом в финальном состоянии не может остаться никаких "лишних" фишек. Первые два условия называют также требованием правильной завершаемости процесса [2].

Третье условие означает, что в бездефектной сети нет лишних переходов. Каждый переход теоретически может сработать (при благоприятном для него ходе процесса).

Легко видеть, что сеть на рисунке 2.1 бездефектна. То есть любой экземпляр процесса, управляемый в соответствии с этой схемой, будет вести себя корректно (не попадет в тупик или бесконечный цикл и в итоге правильно завершится).

Рассмотрим пример сети с ошибками:

Пример 2.2. Сеть на рисунке 2.2 моделирует некий процесс обработки заявок, который разделен на два параллельных подпроцесса. Известно, что каждый из подпроцессов может зависнуть, поэтому система отслеживает наступление тайм-аутов. В случае тайма-аута (хотя бы одного из двух) должна запуститься задача *NOK*, отменяющая заявку.

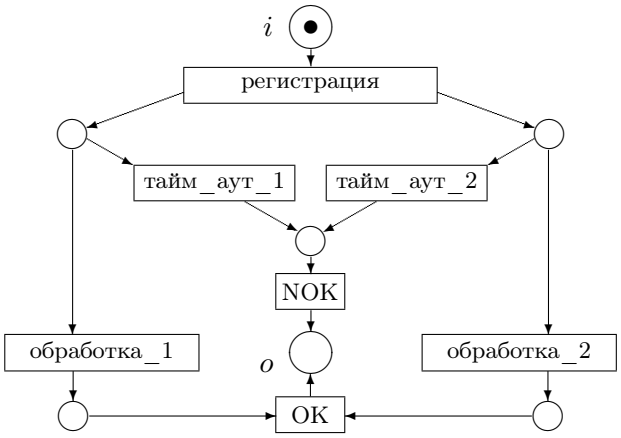


Рис. 2.2: Некорректный процесс

Некорректность в данном случае проявляется в том, что при наступлении одного из тайм-аутов (например, *тайм_ аут_ 1*) процесс может перейти в заключительное состояние (срабатывание NOK и появление фишки в позиции o) до того, как все прочие задания станут неактивными (в частности, активным останется задание *обработка_ 2*). Таким образом, у экземпляра процесса отсутствует четкий признак завершения.

Исправить ситуацию можно разными способами, например, так, как показано на рисунке 2.3. Здесь изменено условие срабатывания перехода NOK — он теперь может сработать только после тайм-аутов в обоих подпроцессах. Также добавлены четыре новых заключительных перехода ($N1, N1', N2$ и $N2'$), соответствующие всем остальным возможным случаям неуспешного завершения экземпляра процесса.

Пусть необходимо определить, является ли данная WF-сеть $N = (P, T, F)$ бездефектной. В [11] было показано, что бездефект-

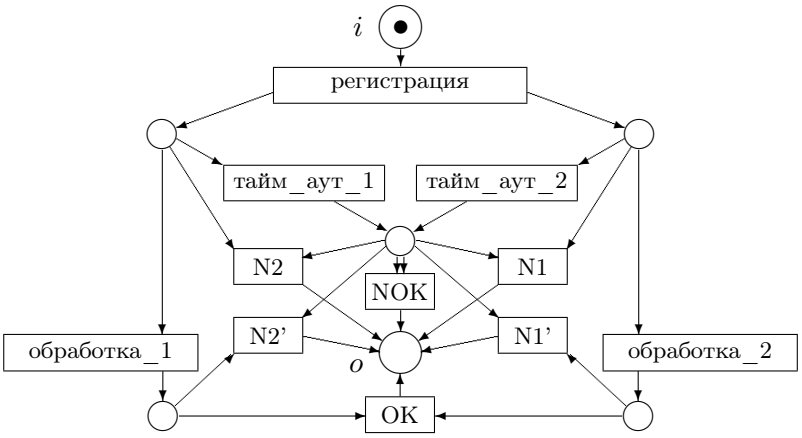


Рис. 2.3: Исправленный процесс

ность соответствует живости и ограниченности. Чтобы связать бездефектность с живостью и ограниченностью, определим расширенную сеть $\underline{N} = (\underline{P}, \underline{T}, \underline{F})$. Сеть Петри $\underline{N}$ получается добавлением нового перехода t^*, соединяющего позиции o и i. Расширенная сеть Петри $\underline{N} = (\underline{P}, \underline{T}, \underline{F})$ определяется следующим образом:

$$\underline{P} =_{\text{def}} P, \quad \underline{T} =_{\text{def}} T \cup \{t^*\}, \quad \underline{F} =_{\text{def}} F \cup \{(o, t^*), (t^*, i)\}.$$

Далее мы будем называть такую расширенную сеть *замыканием* сети N. Пример замыкания приведен на рисунке 2.4.

Справедлива следующая теорема.

Теорема 2.1. *WF-сеть N является бездефектной в том и только том случае, когда сеть $(\underline{N}, i)$ — живая и ограниченная.*

Доказательство см. в [11].

Из этой теоремы следует, что для проверки бездефектности сетей потоков работ можно использовать стандартные методы анализа сетей Петри, рассмотренные в предыдущей главе. Строится

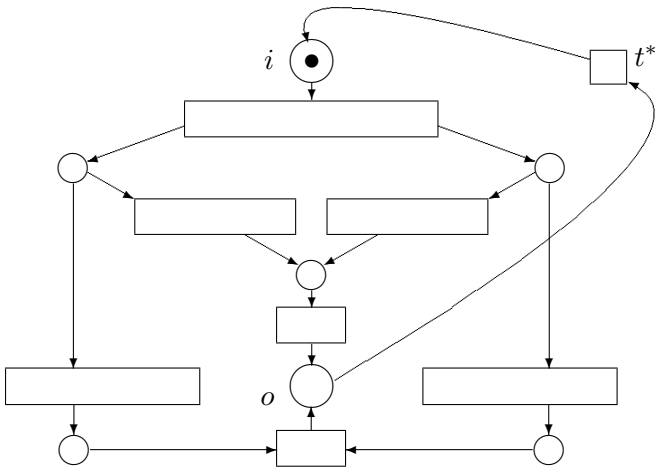


Рис. 2.4: Замыкание сети

полное покрывающее дерево сети $(\underline{N}, i)$, затем по нему определяется ограниченность сети, и, в случае ограниченности, проверяется живость всех переходов.

Пример 2.3. Если построить полное покрывающее дерево для сети на рисунке 2.4, то окажется, что все позиции этой сети не ограничены. Таким образом, сеть на рисунке 2.2 действительно не является бездефектной.

В свою очередь, полное покрывающее дерево для исправленной сети (рисунок 2.3) не содержит ω, то есть соответствующий процесс корректен.

2.3. Структурированные сети

Теорема 2.1 позволяет проверить бездефектность уже построенной схемы потока работ. Однако в ряде случаев мы можем ещё

на этапе построения схемы попытаться избежать некоторых ошибок. Например, не добавлять в сеть переходы без входных или без выходных позиций. Кроме того, естественным способом поддержания "относительной правильности" модели является требование того, чтобы при каждом добавлении в сеть И-разветвления разработчик сразу же добавлял бы соответствующее И-слияние (аналогично для ИЛИ-разветвления и ИЛИ-слияния).

Рассмотрим подклассы WF-сетей, которые благодаря самой своей структуре позволяют избежать многих некорректностей в схеме процесса. Все теоремы приводятся без доказательств, доказательства см. в [1].

WF-сети со свободным выбором

Большинство существующих в настоящее время систем управления потоками работ (СУПР) абстрагируются от состояний между задачами, то есть состояния в них не представлены явно (в виде позиций). В этих системах для описания процедур потоков работ используются строительные блоки И-разветвление, И-слияние, ИЛИ-разветвление и ИЛИ-слияние. И-разветвление и И-слияние используются для параллельной маршрутизации. ИЛИ-разветвление и ИЛИ-слияние — для условной маршрутизации. Поскольку эти системы абстрагируются от состояний, выбор в них всегда делается внутри блока ИЛИ-разветвление. Если же мы моделируем ИЛИ-разветвление с помощью сетей Петри, то конструкции ИЛИ-разветвления соответствует несколько переходов с общим для них множеством входных позиций. Это означает, что процедуры потоков работ в этих СУПР моделируются сетями Петри со свободным выбором.

Действительно, легко заметить, что описание процесса, составленное только из конструкций И-разветвление, И-слияние, ИЛИ-разветвление и ИЛИ-слияние, удовлетворяет свойству свободного выбора. А именно, если два перехода t_1 и t_2 имеют общую входную позицию ($^\bullet t_1 \cap {}^\bullet t_2 \neq \varnothing$), то они являются частью некоторого

ИЛИ-разветвления, то есть “свободного выбора” между несколькими альтернативами. Поэтому множества входных позиций переходов t_1 и t_2 должны удовлетворять условию $({}^\bullet t_1 = {}^\bullet t_2)$.

Сеть, изображенная на рисунке 2.1, является WF-сетью со свободным выбором, а WF-сеть на рисунке 2.3 — нет; например, переходы N2 и N2’ имеют общую входную позицию, но множества их входных позиций не совпадают.

Сети Петри со свободным выбором широко изучались, поскольку они представляются хорошим компромиссом между выразительностью и возможностями анализа. Для этого класса сетей Петри имеются сильные теоретические результаты и эффективные методы анализа. В частности, доказано, что проблема проверки, является ли данная сеть со свободным выбором живой и ограниченной, может быть решена за полиномиальное время. Следовательно, верна следующая теорема:

Теорема 2.2. *Следующая проблема разрешима за полиномиальное время: Определить, является ли данная WF-сеть со свободным выбором бездефектной.*

Теорема 2.2 говорит о том, что для сетей со свободным выбором имеются эффективные алгоритмы проверки бездефектности. Более того, всякая бездефектная WF-сеть со свободным выбором безопасна (при начальном состоянии с единственной фишкой в позиции i):

Теорема 2.3. *Бездефектная WF-сеть со свободным выбором является безопасной.*

Для WF-сетей безопасность — желательное свойство, так как не имеет смысла иметь несколько фишек в позиции, представляющей условие. Условие может быть либо истинным (1 фишка), либо ложным (нет фишек).

Хотя большинство СУПР допускает только потоки работ со свободным выбором, условие свободного выбора для WF-сетей не

исчерпывает структурную характеризацию "хороших" потоков работ. С одной стороны, бывают WF-сети без свободного выбора, которые представляют разумные потоки работ (см. рисунок 2.3). С другой стороны, некоторые бездефектные WF-сети со свободным выбором не имеют смысла. Тем не менее свойство свободы выбора является весьма желательным. Если поток работ можно моделировать WF-сетью со свободным выбором, то так и нужно делать. Спецификация потока работ, основанная на WF-сети со свободным выбором, может исполняться большинством систем потоков работ. К тому же WF-сеть со свободным выбором является более ясной и понятной, для нее имеются эффективные методы анализа. Нарушающие условие свободного выбора конструкции являются потенциальным источником аномального поведения, которое трудно обнаружить.

Хорошо структурированные WF-сети

Другой способ получить структурную характеризацию "хороших" потоков работ состоит в соблюдении баланса между И/ИЛИ-разветвлениями и И/ИЛИ-слияниями. Понятно, что два параллельных потока, инициированных И-разветвлением, не должны соединяться с помощью ИЛИ-слияния. Аналогично два альтернативных потока, порожденных ИЛИ-разветвлением, не должны синхронизироваться И-слиянием. Как показано на рисунке 2.5, И-разветвление должно дополняться И-слиянием, а ИЛИ-разветвление — соответственно ИЛИ-слиянием.

Определение 2.3. Сеть Петри N называется *хорошо организованной*, если для любой пары вершин x, y, из которых одна является позицией, а другая — переходом, и для любых двух простых путей[1] C_1 и C_2, ведущих из вершины x в вершину y,

$$\alpha(C_1) \cap \alpha(C_2) = \{x, y\} \quad \Rightarrow \quad C_1 = C_2,$$

где $\alpha(C)$ — множество вершин на пути C.

[1]Простой путь — путь без самопересечений.

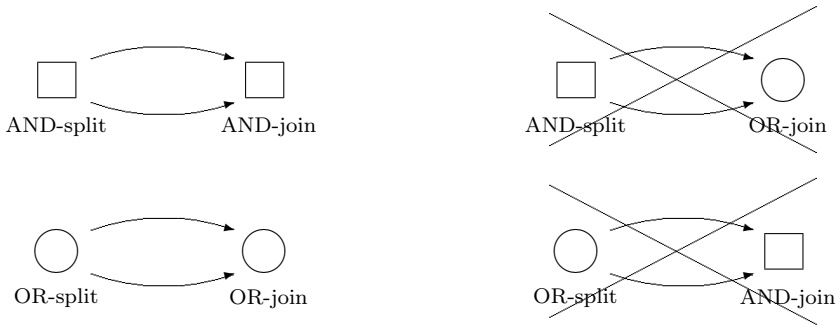


Рис. 2.5: Корректные и некорректные способы организации разветвлений и слияний в схемах потоков работ

Свойство хорошей организованности разрешимо за полиномиальное время путем применения модифицированного алгоритма нахождения максимального потока в сети. Оказывается, для хорошо организованных сетей свойство сильной связности означает хорошую сформированность (см. определения в ч. 1):

Теорема 2.4. *Сильно связная и хорошо организованная сеть Петри является хорошо сформированной.*

Будем называть WF-сеть хорошо структурированной, если ее замыкание является хорошо организованной сетью.

Определение 2.4. WF-сеть N называется *хорошо структурированной*, если $\underline{N}$ — хорошо организованная сеть.

Хорошо структурированные WF-сети обладают рядом ценных свойств. Для них бездефектность может быть проверена за полиномиальное время, а бездефектная и хорошо структурированная WF-сеть является безопасной:

Теорема 2.5. *Следующая проблема разрешима за полиномиальное время: для заданной хорошо структурированной WF-сети проверить, является ли она бездефектной.*

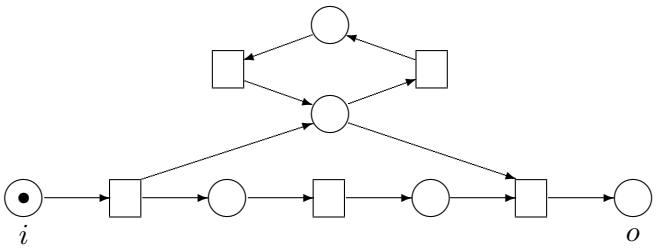


Рис. 2.6: Хорошо структурированная WF-сеть

Теорема 2.6. *Бездефектная хорошо структурированная WF-сеть является безопасной.*

Хорошо структурированные WF-сети и WF-сети со свободным выбором имеют похожие свойства. И в том и в другом случае бездефектность можно проверить очень эффективно, а из бездефектности следует безопасность. Несмотря на эту похожесть, существуют бездефектные хорошо структурированные WF-сети, которые не являются сетями со свободным выбором (рисунок 2.6), и существуют бездефектные WF-сети со свободным выбором, которые не являются хорошо структурированными. На самом деле, можно построить бездефектную WF-сеть, которая не является ни сетью со свободным выбором, ни хорошо структурированной.

Пример 2.4. На рисунке 2.7 изображена сеть Петри, моделирующая ещё один вид процесса обработки страховых требований (слегка отличающийся от рассмотренного ранее).

Прежде всего требование регистрируется (задача *регистрация*), затем параллельно выполняются две задачи: клиенту посылается анкета (задача *послать_анкету*) и производится оценка требования (задача *оценка*). Если анкета возвращается в течение двух недель, то выполняется задача *обработка_анкеты*. Если в течение двух недель анкета не возвращена, то результат анкетирования игнорируется (задача *тайм-аут*). На основании резуль-

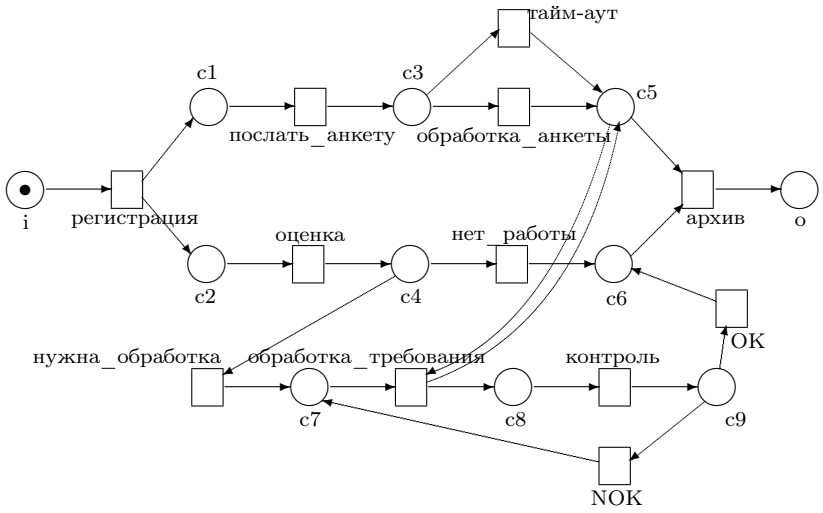


Рис. 2.7: Бездефектная WF-сеть, которая не является ни сетью со свободным выбором, ни хорошо структурированной сетью

тата оценки требование либо рассматривается, либо нет. Реальная работа с требованием (задача *обработка_ требования*) откладывается до того момента, когда будет выполнено условие c5, т. е. либо анкета обработана, либо истекло время ее ожидания. Обработка требования контролируется с помощью задачи *контроль*. В конце концов, выполняется задача *архив*.

Поскольку задача *обработка_ требования* проверяет наличие фишки в позиции c5, данная сеть не является ни хорошо структурированной, ни сетью со свободным выбором.

Краткие выводы

Мы рассмотрели две структурные характеристики сетей Петри (свободный выбор и хорошая структурированность). Обе эти характеристики очень полезны для анализа определений систем потоков работ. Для WF-сети, которая удовлетворяет хотя бы одной из них, имеются высокоэффективные алгоритмы анализа. Однако, как было показано, существуют сети потоков работ, которые не являются ни сетями со свободным выбором, ни хорошо структурированными сетями. Рассмотрим в качестве примера сеть на рисунке 2.7. Хотя данная сеть представляет собой разумно построенную систему потоков работ, большинство современных СУПР не поддерживают такие “продвинутые” средства маршрутизации, как используемая задачей *обработка_ требования* проверка фишки в позиции c5. Поэтому в том случае, когда WF-сеть не является хорошо структурированной сетью или сетью со свободным выбором, имеет смысл локализовать причину нарушения этих свойств и проверить, действительно ли было необходимо использование настолько сложных конструкций. Если же использование конструкций, нарушающих свойства хорошей структурированности и свободы выбора, было и на самом деле необходимо, следует проверить их особенно внимательно как потенциальный источник ошибок. Тем самым можно также улучшить читабельность определения потоков работ и удобство их сопровождения.

Литература

[1] Аалст, ван дер, В. Управление потоками работ: модели, методы и системы / В. ван дер Аалст, К. ван Хей. — М.: Научный мир, 2007.

[2] Ачасова, С. М. Корректность параллельных вычислительных процессов / С. М. Ачасова, О. Л. Бандман. — Новосибирск: Наука, 1990.

[3] Калянов, Г. Н. Моделирование, анализ, реорганизация и автоматизация бизнес-процессов / Г. Н. Калянов. — М.: Финансы и статистика, 2006.

[4] Котов, В. Е. Сети Петри / В. Е. Котов. — М.: Наука, 1984.

[5] Ломазова, И. А. Сети Петри и анализ поведенческих свойств распределенных систем / И. А. Ломазова. — Ярославль: ЯрГУ, 2002.

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

[7] О'Лири, Д. ERP системы. Современное планирование и управление ресурсами предприятия. Выбор, внедрение, эксплуатация. / Д. О'Лири. — М.: ООО "Вершина", 2004.

[8] Питерсон, Дж. Сети Петри и моделирование систем / Дж. Питерсон. — М.: Мир, 1984.

[9] Фаулер, М. Архитектура корпоративных программных приложений / М. Фаулер. — М.: Вильямс, 2008.

[10] Чаадаев, В. К. Бизнес-процессы в компаниях связи / В. К. Чаадаев. — М.: Эко-Трендз, 2004.

[11] Aalst, van der, W. M. P. Verification of Workflow Nets / W. M. P. van der Aalst. — In P. Azéma and G. Balbo, editors, Application and Theory of Petri Nets 1997. Lecture Notes in Computer Science, vol. 1248. Berlin, Springer–Verlag, 1997. — p. 407–426.

[12] Aalst, van der, W. M. P. The Application of Petri Nets to Workflow Management / W. M. P. van der Aalst. — The Journal of Circuits, Systems and Computers, 8(1), 1998. — p. 21–66.

[13] Mayr, E. W. An algorithm for the general Petri net reachability problem / E. W. Mayr. — SIAM J. Comput., 13(3), 1984. — p. 441–460.

[14] Petri, C. A. Kommunikation mit Automaten / C. A. Petri. — PhD theses. Bonn: Institute für Instrumentelle Mathematik, 1962.

[15] Reisig, W. Petri net models of distributed algorithms / W. Reisig. — Lecture Notes in Computer Science, vol. 1000, 1995.

[16] Классификация BPM-систем / `http://bpms.ru`

[17] Моделирование и анализ потоков работ сетями Петри / `http://is.tm.tue.nl/staff/wvdaalst/publications`

[18] Business Process and Workflow Management Coalition / `http://www.wfmc.org`

Учебное издание

Башкин Владимир Анатольевич

Модели потоков работ

Методические указания

Редактор, корректор И. В. Бунакова
Компьютерный набор, вёрстка В. А. Башкина

Подписано в печать 08.07.09. Формат 60×84/16.
Бумага офсетная. Гарнитура “Times New Roman”.
Усл. печ. л. 2,5. Уч.-изд. л. 1,9.
Тираж 100 экз. Заказ

Оригинал-макет подготовлен в редакционно-издательском отделе ЯрГУ

Отпечатано на ризографе.

Ярославский государственный университет им. П. Г. Демидова.

150000, Ярославль, ул. Советская, 14.