Теория процессов

А.М.Миронов

Содержание

1 Введение **6**

1.1 Предмет теории процессов 6
1.2 Верификация процессов 8
1.3 Спецификация процессов 9

2 Понятие процесса **11**

2.1 Представление поведения систем процессами 11
2.2 Неформальное понятие процесса и примеры процессов 12
2.2.1 Неформальное понятие процесса 12
2.2.2 Пример процесса 12
2.2.3 Другой пример процесса 14
2.3 Действия 15
2.4 Определение понятия процесса 17
2.5 Понятие трассы 19
2.6 Достижимые и недостижимые состояния 19
2.7 Замена состояний 20

3 Операции на процессах **21**

3.1 Префиксное действие 21
3.2 Пустой процесс 22
3.3 Альтернативная композиция 23
3.4 Параллельная композиция 28
3.5 Ограничение 41
3.6 Переименование 43
3.7 Свойства операций на процессах 44

4 Эквивалентность процессов **50**

4.1 Понятие эквивалентности процессов и связанные с ним задачи 50
4.2 Трассовая эквивалентность процессов 51
4.3 Сильная эквивалентность 53
4.4 Критерии сильной эквивалентности 56
4.4.1 Логический критерий сильной эквивалентности 56

4.4.2 Критерий сильной эквивалентности, основанный на понятии бимоделирования . . . 59
4.5 Алгебраические свойства сильной эквивалентности . . . 59
4.6 Распознавание сильной эквивалентности . . . 67
4.6.1 Отношение $\mu(P_1, P_2)$ . . . 67
4.6.2 Полиномиальный алгоритм распознавания сильной эквивалентности . . . 68
4.7 Минимизация процессов . . . 72
4.7.1 Свойства отношений вида $\mu(P, P)$ . . . 72
4.7.2 Минимальные процессы относительно $\sim$ . . . 74
4.7.3 Алгоритм минимизации процессов . . . 76
4.8 Наблюдаемая эквивалентность . . . 78
4.8.1 Определение наблюдаемой эквивалентности . . . 78
4.8.2 Логический критерий наблюдаемой эквивалентности . . 81
4.8.3 Критерий наблюдаемой эквивалентности, основанный на понятии наблюдаемого БМ . . . 83
4.8.4 Алгебраические свойства наблюдаемой эквивалентности 85
4.8.5 Распознавание наблюдаемой эквивалентности и минимизация процессов относительно $\approx$ . . . 86
4.8.6 Другие критерии эквивалентности процессов . . . 87
4.9 Наблюдаемая конгруэнция . . . 88
4.9.1 Мотивировка понятия наблюдаемой конгруэнции . . . 88
4.9.2 Определение понятия наблюдаемой конгруэнции . . . 90
4.9.3 Логический критерий наблюдаемой конгруэнтности . . 91
4.9.4 Критерий наблюдаемой конгруэнтности, основанный на понятии НБМ . . . 92
4.9.5 Алгебраические свойства наблюдаемой конгруэнции . . 93
4.9.6 Распознавание наблюдаемой конгруэнтности . . . 102
4.9.7 Минимизация процессов относительно наблюдаемой конгруэнции . . . 103

5 Рекурсивные определения процессов 104
5.1 Процессные выражения . . . 104
5.2 Понятие рекурсивного определения процессов . . . 105
5.3 Вложение процессов . . . 106
5.4 Предел последовательности вложенных процессов . . . 107
5.5 Процессы, определяемые процессными выражениями . . . 110
5.6 Эквивалентность РО . . . 111
5.7 Переходы на $PExpr$ . . . 112
5.8 Доказательство эквивалентности процессов при помощи РО . . 114
5.9 Проблемы, связанные с понятием РО . . . 115

6 Примеры доказательства свойств процессов 116
6.1 Потоковые графы . . . 116
6.2 Мастерская . . . 117
6.3 Неконфликтное использование ресурса . . . 121
6.4 Планировщик . . . 124
6.5 Семафор . . . 133

7 Процессы с передачей сообщений 136
7.1 Действия с передачей сообщений . . . 136
7.2 Вспомогательные понятия . . . 137
7.2.1 Типы, переменные, значения и константы . . . 137
7.2.2 Функциональные символы . . . 138
7.2.3 Выражения . . . 139
7.3 Понятие процесса с передачей сообщений . . . 141
7.3.1 Множество переменных процесса . . . 141
7.3.2 Начальное условие . . . 142
7.3.3 Операторы . . . 142
7.3.4 Определение процесса . . . 144
7.3.5 Функционирование процесса . . . 144
7.4 Представление процессов в виде блок-схем . . . 146
7.4.1 Понятие блок-схемы . . . 146
7.4.2 Функционирование блок-схемы . . . 150
7.4.3 Построение процесса, определяемого блок-схемой . . . 151
7.5 Пример процесса с передачей сообщений . . . 153
7.5.1 Понятие буфера . . . 154
7.5.2 Представление поведения буфера в виде блок-схемы . . . 155
7.5.3 Представление поведения буфера в виде процесса . . . 156
7.6 Операции на процессах с передачей сообщений . . . 158
7.6.1 Префиксное действие . . . 158
7.6.2 Альтернативная композиция . . . 158
7.6.3 Параллельная композиция . . . 158
7.6.4 Ограничение . . . 159
7.6.5 Переименование . . . 160
7.7 Эквивалентность процессов . . . 160
7.7.1 Понятие конкретизации процесса . . . 160
7.7.2 Понятие эквивалентности процессов . . . 161
7.8 Процессы с составными операторами . . . 162
7.8.1 Мотивировка понятия процесса с составными операторами . . . 162
7.8.2 Понятие составного оператора . . . 163
7.8.3 Понятие процесса с СО . . . 163
7.8.4 Функционирование процесса с СО . . . 164
7.8.5 Операции на процессах с СО . . . 165

7.8.6 Преобразование процессов с передачей сообщений в процессы с СО . . . 166
7.8.7 Конкатенация СО . . . 166
7.8.8 Редукция процессов с СО . . . 168
7.8.9 Пример редукции . . . 170
7.8.10 Понятие конкретизации процесса с СО . . . 174
7.8.11 Отношения эквивалентности на процессах с СО . . . 175
7.8.12 Метод доказательства наблюдаемой эквивалентности процессов с СО . . . 176
7.8.13 Пример доказательства наблюдаемой эквивалентности процессов с СО . . . 180
7.8.14 Дополнительные замечания . . . 182
7.8.15 Другой пример доказательства наблюдаемой эквивалентности процессов с СО . . . 186
7.9 Рекурсивные определения процессов . . . 191

8 Примеры процессов с передачей сообщений 194
8.1 Разделение множеств . . . 194
8.1.1 Задача разделения множеств . . . 194
8.1.2 Распределённый алгоритм решения задачи разделения множеств . . . 194
8.1.3 Процессы *Small* и *Large* . . . 196
8.1.4 Анализ алгоритма разделения множеств . . . 197
8.2 Вычисление квадрата . . . 201
8.3 Сети Петри . . . 205
8.4 Протоколы передачи данных в компьютерных сетях . . . 206
8.4.1 Понятие протокола . . . 206
8.4.2 Методы исправления искажений в кадрах . . . 207
8.4.3 Методы обнаружения искажений в кадрах . . . 210
8.4.4 Пример протокола . . . 214
8.4.5 Протокол с чередующимися битами . . . 221
8.4.6 Двунаправленная передача . . . 225
8.4.7 Дуплексный протокол с чередующимися битами . . . 226
8.4.8 Двунаправленная конвейерная передача . . . 228
8.4.9 Протокол скользящего окна с возвратом . . . 229
8.4.10 Протокол скользящего окна с выборочным повтором . . . 233
8.5 Криптографические протоколы . . . 238
8.5.1 Понятие криптографического протокола . . . 238
8.5.2 Шифрование сообщений . . . 241
8.5.3 Формальное описание КП . . . 242
8.5.4 Примеры КП . . . 243

9 Представление структур данных в виде процессов 249
9.1 Понятие структуры данных 249
9.2 СД "память с 2^k ячейками" 250
9.3 СД "стек" 253
9.4 СД "очередь" 254

10 Семантика языка параллельного программирования 255
10.1 Описание языка параллельного программирования 255
10.1.1 Конструкции языка $\mathcal{L}$ 255
10.1.2 Программы на языке $\mathcal{L}$ 258
10.2 Семантика языка $\mathcal{L}$ 258
10.2.1 Семантика выражений 259
10.2.2 Семантика деклараций 260
10.2.3 Семантика операторов 261

11 Исторический обзор и современное состояние дел 263
11.1 Робин Милнер 263
11.2 Исчисление взаимодействующих систем (CCS) 264
11.3 Теория взаимодействующих последовательных процессов (CSP) 265
11.4 Алгебра взаимодействующих процессов (ACP) 265
11.5 Процессные алгебры 266
11.6 Мобильные процессы 268
11.7 Гибридные системы 269
11.8 Другие математические теории и программные средства, связанные с моделированием процессов 269
11.9 Бизнес-процессы 270

Глава 1

Введение

1.1 Предмет теории процессов

Теория процессов является одним из разделов математической теории программирования, который изучает математические модели поведения динамических систем, называемые **процессами**.

Говоря неформально, **процесс** представляет собой модель такого поведения, которое заключается в исполнении **действий**. Такими действиями могут быть, например,

- приём или передача каких-либо объектов, или
- преобразование этих объектов.

Основные достоинства теории процессов как математического аппарата, предназначенного для моделирования и анализа динамических систем, заключаются в следующем.

1. Аппарат теории процессов хорошо подходит для формального описания и анализа поведения **распределённых динамических систем**, т.е. таких систем, которые состоят из нескольких взаимодействующих компонентов, причем
   - все эти компоненты работают параллельно, и
   - взаимодействие компонентов происходит путём пересылки сигналов или сообщений от одних компонентов другим компонентам.

   Важнейшим примером распределенных систем являются программно-аппаратные вычислительные комплексы, в которых
   - один класс компонентов определяется совокупностью компьютерных программ, которые функционируют в этом комплексе

- другой класс компонентов связан с аппаратной платформой, на базе которой функционирует этот комплекс
- третий класс компонентов представляет собой совокупность информационных ресурсов (базы данных, базы знаний, электронные библиотеки, и т.п.), которые используются для обеспечения функционирования этого комплекса
- также может приниматься во внимание класс компонентов, связанных с человеческим фактором.

2. Методы теории процессов позволяют анализировать с приемлемой сложностью модели с очень большим и даже бесконечным множеством состояний. Это возможно благодаря разработанной в теории процессов технике символьных преобразований алгебраических выражений, описывающих процессы.

   Важнейшим примером моделей с бесконечным множеством состояний являются модели компьютерных программ с переменными, множества значений которых имеют очень большой размер. В таких моделях программ в целях удобства проведения рассуждений большие множества значений некоторых переменных заменяются на соответствующие бесконечные множества. Например, множество значений переменных типа `double`, представляющее собой конечную (но очень большую) совокупность действительных чисел, может быть заменено на бесконечное множество всех действительных чисел.

   Во многих случаях представление анализируемой программы в виде модели с бесконечным множеством состояний существенно упрощает проведение рассуждений об этой программе. Анализ модели этой программы с конечным, но очень большим множеством состояний классическими методами теории автоматов, которые основаны на

   - явном представлении множества состояний этой модели, и
   - анализе этой модели путём явного оперирования с её состояниями

   может иметь очень высокую вычислительную сложность, и в некоторых случаях замена

   - задачи анализа исходной конечной модели
   - на задачу анализа соответствующей бесконечной модели такими методами, которые основаны на алгебраических преобразованиях выражений, описывающих эту модель,

   может дать существенный выигрыш с точки зрения вычислительной сложности.

3. Методы теории процессов хорошо подходят для изучения **иерархических** систем, т.е. таких систем, которые имеют многоуровневую структуру.

   Каждая компонента таких систем рассматривается как подсистема, которая, в свою очередь, может состоять из нескольких подкомпонентов. Каждая из этих подкомпонентов может взаимодействовать

   - с другими подкомпонентами, и
   - с системами более высокого уровня.

Главными источниками задач и объектами применения результатов теории процессов являются распределенные компьютерные системы.

Вместе с тем, теория процессов может использоваться также для моделирования и анализа поведения систем самой различной природы, важнейшими примерами которых являются **организационные системы**. К их числу относятся

- системы управления деятельностью предприятий,
- государственные структуры,
- системы организации коммерческих процессов (например, системы организации торговых сделок, аукционов, и т.п.)

Процессы, относящиеся к функционированию таких систем, принято называть **бизнес-процессами**.

1.2 Верификация процессов

Наиболее важный класс задач, для решения которых предназначена теория процессов, связан с проблемой верификации процессов.

Проблема **верификации** процесса заключается в построении формального доказательства того, что анализируемый процесс обладает заданными свойствами.

Для многих процессов данная задача представляет исключительную актуальность. Например, безопасная эксплуатация таких систем, как

- системы управления атомными электростанциями,
- медицинские устройства с компьютерным управлением,
- бортовые системы управления самолетов и космических аппаратов,
- системы управления секретными базами данных,

- системы электронной коммерции

невозможна без удовлетворительного решения задачи верификации свойств корректности и безопасности процессов, функционирующих в таких системах, так как нарушения данных свойств в системах подобного типа могут привести к существенному ущербу для экономики и самой жизни людей.

Точная постановка задачи верификации состоит из следующих частей.

1. Построение процесса, представляющего собой математическую модель поведения анализируемой системы.

2. Представление проверяемого свойства в виде математического объекта (называемого **спецификацией**).

3. Построение **математического доказательства** утверждения о том, что построенный процесс удовлетворяет спецификации.

1.3 Спецификация процессов

Спецификация процесса представляет собой описание свойств этого процесса в виде некоторого математического объекта.

Примером спецификации является требование надежности передачи данных через ненадежную среду. При этом не указывается, как именно должна обеспечиваться эта надежность.

В качестве спецификации могут выступать, например, следующие объекты.

1. Логическая формула, выражающая некоторое требование к процессу.

   Например, таким требованием может быть условие того, что если процесс получил некоторый запрос, то через некоторое заданное время процесс выдаст ответ на этот запрос.

2. Представление анализируемого процесса на более высоком уровне абстракции.

   Данный вид спецификаций используется при многоуровневом проектировании процессов: реализацию процесса на каждом уровне проектирования можно рассматривать как спецификацию для реализации этого процесса на следующем уровне проектирования.

3. Некоторый эталонный процесс, относительно которого предполагается, что он обладает заданным свойством.

   В этом случае задача верификации заключается в построении доказательства эквивалентности эталонного и анализируемого процессов.

При построении спецификаций следует руководствоваться следующими принципами.

1. Одно и то же свойство процесса может быть выражено на разных языках спецификаций (ЯС), и

   - на одном ЯС оно может иметь простую спецификацию,
   - а на другом – сложную.

   Например, спецификация, описывающая связь между входными и выходными значениями для программы, вычисляющей разложение целого числа на простые множители, имеет

   - сложный вид на языке логики предикатов, но
   - простой вид, если выразить эту спецификацию в виде некоторой эталонной программы.

   Поэтому для представления свойства процесса в виде спецификации важно выбрать такой ЯС, на котором спецификация этого свойства имела бы наиболее ясный и простой вид.

2. Если свойство процесса изначально было выражено на естественном языке, то при переводе его в спецификацию важно обеспечить соответствие между

   - естественно-языковым описанием этого свойства, и
   - его спецификацией,

   т.к. в случае несоблюдения этого условия результаты верификации не будут иметь смысла.

Глава 2

Понятие процесса

2.1 Представление поведения систем процессами

Один из возможных методов математического моделирования поведения динамических систем заключается в представлении поведения этих систем в виде **процессов**.

Процесс, как правило, не учитывает всех деталей поведения анализируемой системы. Одно и то же поведение может быть представлено различными процессами, отражающими

- разную степень абстракции при построении модели этого поведения, и
- разные уровни детализации действий, исполняемых системой.

Если целью построения процесса для представления поведения некоторой системы является проверка свойств этого поведения, то выбор уровня детализации действий системы должен производиться с учётом тех свойств, которые необходимо проанализировать. Построение процесса, представляющего поведение анализируемой системы, должно производиться с учётом следующих принципов.

1. Описание процесса не должно быть чрезмерно детальным, т.к. излишняя сложность этого описания может вызвать существенные вычислительные проблемы при формальном анализе этого процесса.
2. Описание процесса не должно быть чрезмерно упрощённым, оно должно
   - отражать те аспекты поведения моделируемой системы, которые имеют отношение к проверяемым свойствам, и
   - сохранять все свойства поведения этой системы, которые представляют интерес для анализа

т.к. в случае несоблюдения этого условия результаты анализа такого процесса не будут иметь смысла.

Прежде чем сформулировать точное определение процесса, мы приведём неформальное понятие процесса, и рассмотрим простейшие примеры процессов.

2.2 Неформальное понятие процесса и примеры процессов

2.2.1 Неформальное понятие процесса

Как было сказано выше, мы понимаем под процессом модель поведения динамической системы на некотором уровне абстракции.

Процесс можно представлять себе как граф P, компоненты которого имеют следующий смысл.

- Вершины графа P называются **состояниями**, и изображают ситуации (или классы ситуаций), в которых может находиться моделируемая система в различные моменты своего функционирования.

  Одно из состояний является выделенным, оно называется **начальным состоянием** процесса P.

- Рёбра графа P имеют метки, изображающие **действия**, которые может исполнять моделируемая система.

- Функционирование процесса P описывается переходами по рёбрам графа P от одного состояния к другому. Функционирование начинается из начального состояния.

  Метка каждого ребра изображает действие процесса, исполняемое при переходе от состояния в начале ребра к состоянию в его конце.

2.2.2 Пример процесса

В качестве первого примера рассмотрим процесс, представляющий собой простейшую модель поведения некоторого торгового автомата.

Мы будем представлять себе этот автомат как машину, которая имеет

- монетоприемник,
- кнопку, и
- лоток для выдачи товара.

Когда покупатель хочет приобрести товар, он

- опускает монету в монетоприемник,
- нажимает на кнопку

и после этого в лотке появляется товар.

Предположим, что наш автомат торгует шоколадками по цене 1 монета за штуку.

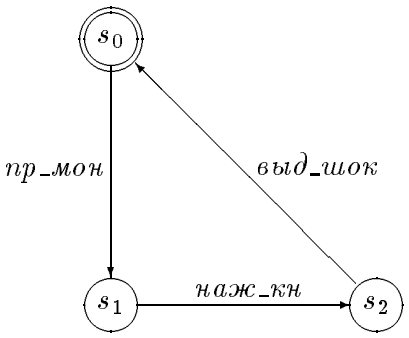
Опишем действия такого автомата.

- По инициативе покупателя, в автомате могут происходить следующие действия:
  - попадание в щель монеты, и
  - нажатие кнопки.
- В ответ автомат может осуществлять реакцию: выдавать в лоток шоколадку.

Обозначим действия короткими именами:

- прием монеты мы обозначим символом *пр_мон*,
- нажатие кнопки - символом *наж_кн*, и
- выдачу шоколадки - символом *выд_шок*.

Процесс нашего торгового автомата выглядит следующим образом:



Данная диаграмма объясняет, как именно функционирует торговый автомат:

- вначале автомат находится в состоянии s_0, в этом состоянии он ожидает появления в приемнике монеты
  (то, что состояние s_0 является начальным, изображается на диаграмме двойным кружочком вокруг идентификатора этого состояния)

- когда монета появляется, автомат переходит в состояние s_1 и ждет нажатия на кнопку,
- после нажатия кнопки автомат
  - переходит в состояние s_2,
  - выдает шоколадку, и
  - возвращается в состояние s_0.

2.2.3 Другой пример процесса

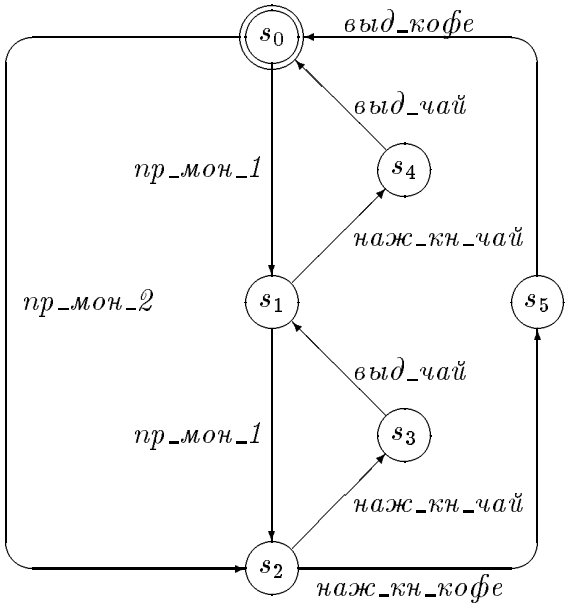
Рассмотрим более сложный пример торгового автомата, который отличается от предыдущего тем, что продаёт два вида товаров: чай и кофе, причём стоимость чая - 1 рубль, а стоимость кофе - 2 рубля.

Автомат имеет две кнопки: одну - для чая, другую - для кофе.

Покупатель может платить монетами достоинством в 1 рубль и 2 рубля. Данные монеты будут обозначаться знакосочетаниями *мон_1* и *мон_2* соответственно.

Если покупатель опустил в монетоприемник монету *мон_1*, он может купить только чай. Если же он опустил монету *мон_2*, он может купить кофе или два чая. Кофе можно купить также, опустив в монетоприёмник две монеты *мон_1*.

Процесс такого торгового автомата выглядит следующим образом:



Для формального определения понятия процесса мы должны уточнить понятие действия. Это уточнение излагается в параграфе 2.3.

2.3 Действия

Для задания процесса P, представляющего собой модель поведения некоторой динамической системы, должно быть указано некоторое множество $Act(P)$ **действий**, которые может выполнять процесс P.

Мы будем предполагать, что действия всех процессов являются элементами некоторого универсального множества Act всех возможных действий, которые может выполнить какой-либо процесс, т.е. для любого процесса P

$$Act(P) \subseteq Act$$

Выбор множества $Act(P)$ действий процесса P зависит от целей моделирования. В разных ситуациях для представления модели анализируемой системы в виде некоторого процесса могут выбираться разные множества действий.

Мы будем предполагать, что множество действий Act делится на 3 следующих класса.

1. **Входные действия**, которые изображаются знакосочетаниями вида

   $$\alpha?$$

   Действие вида $\alpha?$ интерпретируется как ввод в процесс некоторого объекта с именем α.

2. **Выходные действия**, которые изображаются знакосочетаниями вида

   $$\alpha!$$

   Действие вида $\alpha!$ интерпретируется как вывод из процесса некоторого объекта с именем α.

3. **Внутреннее** (или **невидимое**) действие, которое обозначается символом τ.

   Внутренним мы называем такое действие процесса P, которое не связано с его взаимодействием с **окружающей средой** (т.е. с процессами, являющимися внешними по отношению к процессу P, и с которыми он может взаимодействовать).

   Например, внутреннее действие может быть связано с взаимодействием компонентов процесса P.

   В действительности, внутренние действия могут быть самыми разнообразными, но для обозначения всех внутренних действий мы будем использовать один и тот же символ τ. Это отражает наше желание не различать все внутренние действия, т.к. они не являются "наблюдаемыми" извне процесса P.

Обозначим знакосочетанием $Names$ совокупность имён объектов, которые могут вводиться в какой-либо процесс нли выводиться из него.

Мы не накладываем никаких ограничений на виды объектов, с которыми могут оперировать процессы, поэтому множество $Names$ предполагается бесконечным.

Множество Act по определению представляет собой дизъюнкное объединение

$$\begin{array}{rcl} Act & = & \{\alpha? \mid \alpha \in Names\} \quad \cup \\ & \cup & \{\alpha! \mid \alpha \in Names\} \quad \cup \\ & \cup & \{\tau\} \end{array} \tag{2.1}$$

Отметим, что объекты, которые вводятся в процесс и выводятся из него, могут иметь самую различную природу (как материальную, так и не материальную). Например, ими могут быть

- материальные ресурсы,
- люди,
- деньги,
- информация,
- энергия,
- и т.д.

Кроме того, сами понятия ввода и вывода могут иметь виртуальный характер, т.е. слова "ввод" и "вывод" могут использоваться лишь как метафоры, а в действительности никакого ввода или вывода какого-либо реального объекта может и не происходить. Говоря неформально, мы будем рассматривать действие процесса P как

- **входное**, если его инициатором является процесс, внешний по отношению к P, и
- **выходное**, если оно не является внутренним, и его инициатором является сам процесс P.

Для каждого имени $\alpha \in Names$ действия $\alpha?$ и $\alpha!$ называются **комплементарными**.

Мы будем использовать следующие обозначения.

1. Для каждого действия $a \in Act \setminus \{\tau\}$ знакосочетание $\bar{a}$ обозначает действие, комплементарное к a, т.е.

$$\overline{\alpha?} \stackrel{\text{def}}{=} \alpha!, \quad \overline{\alpha!} \stackrel{\text{def}}{=} \alpha?$$

2. Для каждого действия $a \in Act \setminus \{\tau\}$ знакосочетание $name(a)$ обозначает имя, указанное в действии a, т.е.

$$name(\alpha?) \stackrel{\text{def}}{=} name(\alpha!) \stackrel{\text{def}}{=} \alpha$$

3. Для каждого подмножества $L \subseteq Act \setminus \{\tau\}$

   - $\overline{L} \stackrel{\text{def}}{=} \{\overline{a} \mid a \in L\}$
   - $names(L) \stackrel{\text{def}}{=} \{name(a) \mid a \in L\}$

2.4 Определение понятия процесса

Процессом называется тройка P вида

$$P = (S, s^0, R) \tag{2.2}$$

компоненты которой имеют следующий смысл.

- S - множество, элементы которого называются **состояниями** процесса P.
- $s^0 \in S$ – некоторое выделенное состояние, называемое **начальным состоянием** процесса P.
- R – подмножество вида

  $$R \subseteq S \times Act \times S$$

  Элементы множества R называются **переходами**.

  Если переход из R имеет вид (s_1, a, s_2), то

  – мы будем говорить, что этот переход является переходом из состояния s_1 в состояние s_2 с выполнением действия a,

  – состояния s_1 и s_2 называются **началом** и **концом** этого перехода соответственно, а действие a называется **меткой** этого перехода, и

  – иногда, в целях повышения наглядности, мы будем обозначать данный переход знакосочетанием

  $$s_1 \xrightarrow{\ a\ } s_2 \tag{2.3}$$

Функционирование процесса $P = (S, s^0, R)$ заключается в порождении последовательности переходов вида

$$s^0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \dots$$

и выполнении действий $a_0, a_1, a_2 \dots$, соответствующих этим переходам.

Более подробно: на каждом шаге функционирования $i \geq 0$

- процесс находится в некотором состоянии s_i $(s_0 = s^0)$,
- если есть хотя бы один переход из R с началом в s_i, то процесс
  - недетерминированно выбирает переход с началом в s_i, помеченный таким действием a_i, которое можно выполнить в текущий момент времени,
    (если таких переходов нет, то процесс временно приостанавливает свою работу до того момента, когда появится хотя бы один такой переход)
  - выполняет действие a_i, и после этого
  - переходит в состоние s_{i+1}, которое является концом выбранного перехода
- если в R нет переходов с началом в s_i, то процесс заканчивает свою работу.

Знакосочетание $Act(P)$ обозначает множество всех действий из $Act \setminus \{\tau\}$, которые могут быть выполнены процессом P, т.е.

$$Act(P) \stackrel{\text{def}}{=} \{a \in Act \setminus \{\tau\} \mid \exists\, (\, s_1 \xrightarrow{a} s_2\,) \in R\}$$

Процесс (2.2) называется **конечным**, если его компоненты S и R являются конечными множествами.

Конечный процесс можно изображать геометрически, в виде диаграммы на плоскости, в которой

- каждому состоянию соответствует некоторый кружочек на плоскости, в котором может быть написан идентификатор, представляющий собой имя этого состояния,
- каждому переходу соответствует стрелка, соединяющая начало этого перехода и его конец, причём на стрелке написана метка этого перехода,
- начальное состояние выделяется некоторым образом (например, вместо обычного кружочка рисуется двойной кружочек).

Примеры таких диаграмм содержатся в параграфах 2.2.2 и 2.2.3.

2.5 Понятие трассы

Пусть $P = (S, s^0, R)$ – некоторый процесс.

Трассой процесса P называется конечная или бесконечная последовательность

$$a_1, a_2, \ldots$$

элементов множества Act, такая что существует последовательность состояний процесса P

$$s_0, s_1, s_2, \ldots$$

обладающая следующими свойствами:

- s_0 совпадает с начальным состоянием s^0 процесса P
- для каждого $i \geq 1$ множество R содержит переход

$$s_i \xrightarrow{a_i} s_{i+1}$$

Множество всех трасс процесса P мы будем обозначать через $Tr(P)$.

2.6 Достижимые и недостижимые состояния

Пусть P – процесс вида (2.2).

Состояние s процесса P называется **достижимым**, если $s = s^0$ или существует последовательность переходов в P, имеющая вид

$$s_0 \xrightarrow{a_1} s_1\,, \quad s_1 \xrightarrow{a_2} s_2\,, \quad \ldots \quad s_{n-1} \xrightarrow{a_n} s_n$$

в которой $n \geq 1$, $s_0 = s^0$ и $s_n = s$.

Состояние называется **недостижимым**, если оно не является достижимым.

Нетрудно видеть, что после после того, как

- из S будут удалены недостижимые состояния, и
- из R будут удалены переходы, в которых присутствуют недостижимые состояния,

получившийся процесс P' (который иногда называют **достижимой частью** процесса P) будет представлять точно такое же поведение, которое представлял исходный процесс. По этой причине мы будем рассматривать такие процессы P и P' как одинаковые.

2.7 Замена состояний

Пусть

- P – процесс вида (2.2),
- s – некоторое состояние из S
- s' – произвольный элемент, не принадлежащий множеству S.

Обозначим символом P' процесс, который получается из P заменой s на s' в множествах S и R, т.е., в частности, каждый переход в P вида

$$s \xrightarrow{a} s_1 \quad \text{или} \quad s_1 \xrightarrow{a} s$$

заменяется на переход

$$s' \xrightarrow{a} s_1 \quad \text{или} \quad s_1 \xrightarrow{a} s'$$

соответственно.

Как и в предыдущем параграфе, нетрудно видеть, что P' будет представлять точно такое же поведение, которое представлял P, и по этой причине мы можем рассматривать такие процессы P и P' как одинаковые.

Также отметим, что заменять можно не одно состояние, а произвольное подмножество состояний процесса P. Такую замену можно представить как задание взаимно однозначного отображения

$$f : S \to S' \tag{2.4}$$

и результатом такой замены по определению является процесс P' вида

$$P' = (S', (s')^0, R') \tag{2.5}$$

где

- $(s')^0 \stackrel{\text{def}}{=} f(s^0)$, и
- для каждой пары $s_1, s_2 \in S$ и каждого $a \in Act$

$$(\, s_1 \xrightarrow{a} s_2 \,) \in R \quad \Leftrightarrow \quad (\, f(s_1) \xrightarrow{a} f(s_2) \,) \in R'.$$

Поскольку такие процессы P и P' представляют одинаковое поведение, мы можем рассматривать их как одинаковые.

Отметим, что в литературе по теории процессов такие процессы P и P' иногда называют не одинаковыми, а **изоморфными**. Отображение (2.4) с указанными выше свойствами называют **изоморфизмом** между P и P'. Процесс P' называют **изоморфной копией** процесса P.

Глава 3

Операции на процессах

В этой главе мы определим некоторые операции на процессах, при помощи которых из одних процессов мы сможем строить другие, более сложные процессы.

3.1 Префиксное действие

Первая такая операция - префиксное действие.

Пусть заданы

- процесс $P = (S, s^0, R)$, и
- действие $a \in Act$.

Действие операции **префиксного действия** $a.$ на процесс P заключается в том, что

- к множеству состояний P добавляется новое состояние s, которое будет начальным состоянием нового процесса, и
- к множеству переходов добавляется переход

$$s \xrightarrow{a} s^0$$

Получившийся процесс обозначается знакосочетанием

$$a.P$$

Проиллюстрируем действие данной операции на примере торгового автомата из параграфа 2.2.2. Обозначим процесс, представляющий поведение этого автомата, символом $P_{\text{та}}$.

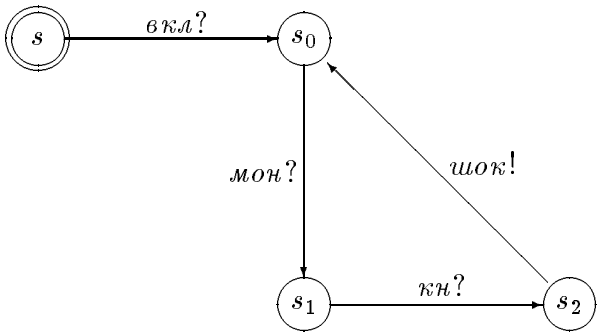
Расширим множество действий данного автомата новым входным действием *вкл*?, которое будет означать включение этого автомата в сеть.

Процесс *вкл*?. $P_{\text{та}}$ представляет поведение нового торгового автомата, который в начальном состоянии не может

- ни принимать монет,
- ни воспринимать нажатия на кнопку,
- ни выдавать шоколадок.

Единственное, что он может - это стать включенным. После этого его поведение ничем не будет отличаться от поведения исходного автомата.

Графовое представление процесса $вкл?.P_{\text{та}}$ выглядит следующим образом:



3.2 Пустой процесс

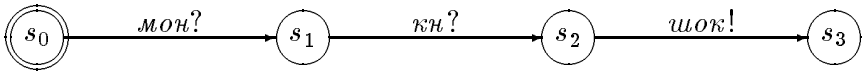
Среди всех процессов существует один наиболее простой. Этот процесс имеет всего одно состояние, и не имеет переходов. Для обозначения такого процесса мы будем использовать константу (т.е. нульарную операцию) $\mathbf{0}$.

Возвращаясь к примерам с торговыми автоматами, можно сказать, что процесс $\mathbf{0}$ представляет поведение сломанного автомата, то есть такого автомата, который вообще не может ничего делать.

Путем применения операций префиксного действия к процессу $\mathbf{0}$ можно определять поведение более сложных автоматов. Рассмотрим, например, такой процесс:

$$P = мон?.кн?.шок!.\mathbf{0}$$

Графовое представление этого процесса выглядит следующим образом:



Этот процесс задает поведение автомата, который обслуживает ровно одного покупателя, и после этого ломается.

3.3 Альтернативная композиция

Следующая операция на процессах - это бинарная операция альтернативной композиции.

Данная операция используется в том случае, когда по паре процессов P_1 и P_2, надо построить процесс P, который будет функционировать

- либо как процесс P_1,
- либо как процесс P_2,

причём выбор процесса, в соответствии с которым P будет функционировать, может определяться

- как самим P,
- так и окружающей средой, в которой функционирует P.

Например, если P_1 и P_2 имеют вид

$$\begin{array}{l} P_1 = \alpha? \,.\, P_1' \\ P_2 = \beta? \,.\, P_2' \end{array} \tag{3.1}$$

и в начальный момент времени окружающая среда

- может ввести в P объект α, но
- не может ввести в P объект β

то P должен выбрать то поведение, которое является единственно возможным в данной ситуации, т.е. работать так же, как процесс P_1.

Отметим, что в данном случае выбирается такой процесс, первое действие в котором может быть выполнено в текущий момент времени. Выбрав P_1, и выполнив действие $\alpha?$, процесс P обязан продолжать работу в соответствии со своим выбором, т.е. в соответствии с процессом P_1'. Не исключено, что после выполнения действия $\alpha?$

- будет невозможно выполнить ни одного действия, работая в соответствии с процессом P_1'
- хотя в этот момент появится возможность выполнить первое действие в P_2.

Но в этот момент процесс P уже не может изменить свой выбор (т.е. выбрать P_2 вместо P_1'). Процесс P может лишь находиться в состоянии ожидания того, когда появится возможность работать в соответствии с процессом P_1'.

Если же в начальный момент времени окружающая среда может ввести в P как α, так и β, то P выбирает процесс, в соответствии с которым он будет работать,

- недетерминированно (т.е. произвольно), или
- с учётом некоторых дополнительных факторов.

Точное определение операции альтернативной композиции выглядит следующим образом.

Пусть процессы P_1 и P_2 имеют вид

$$P_i = (S_i, s_i^0, R_i) \quad (i = 1, 2)$$

причём множества состояний S_1 и S_2 не имеют общих элементов.

Альтернативной композицией процессов P_1 и P_2 называется процесс

$$P_1 + P_2 = (S, s^0, R)$$

компоненты которого определяются следующим образом.

- S получается добавлением к $S_1 \cup S_2$ нового состояния s^0, которое будет начальным состоянием процесса $P_1 + P_2$
- R содержит все переходы из R_1 и R_2.
- для каждого перехода из R_i $(i = 1, 2)$ вида

  $$s_i^0 \xrightarrow{a} s$$

  R содержит переход

  $$s^0 \xrightarrow{a} s$$

Если же множества S_1 и S_2 имеют общие элементы, то для определения процесса $P_1 + P_2$ сначала надо заменить в S_2 те состояния, которые входят также и в S_1, на новые элементы, а также модифицировать соответствующим образом R_2 и s_2^0.

Рассмотрим в качестве примера торговый автомат, который продаёт газированную воду, причём

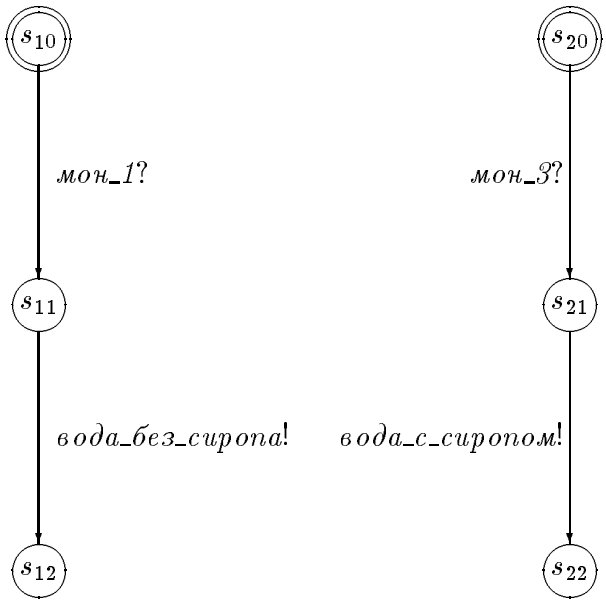
- если покупатель опускает в него монету *мон_1* достоинством в 1 копейку, то автомат выдаёт стакан воды без сиропа,
- а если покупатель опускает в него монету *мон_3* достоинством в 3 копейки, то автомат выдаёт стакан воды с сиропом

причём сразу после продажи одного стакана воды автомат ломается.

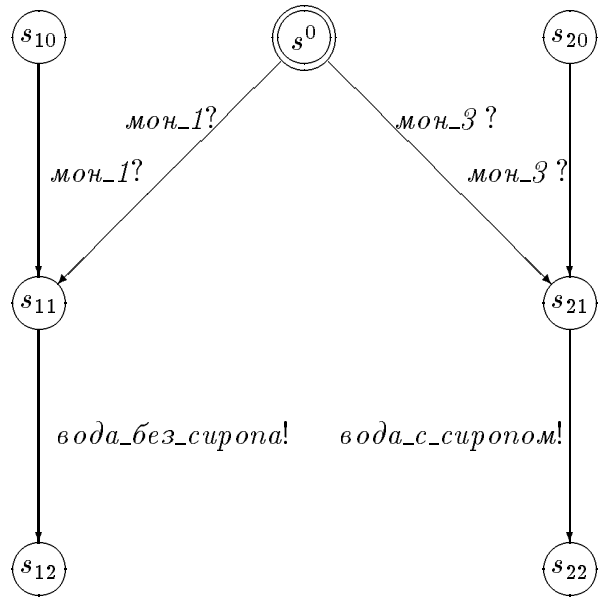
Поведение данного автомата описывается следующим процессом:

$$\begin{aligned} P_{\text{газ\_вода}} &= \textit{мон\_1}\ ?\ .\ \textit{вода\_без\_сиропа}\ !\ .\ \mathbf{0} \ + \\ &+ \textit{мон\_3}\ ?\ .\ \textit{вода\_с\_сиропом}\ !\ .\ \mathbf{0} \end{aligned} \tag{3.2}$$

Рассмотрим графовое представление процесса (3.2).
Графовые представления слагаемых в сумме (3.2) имеют вид

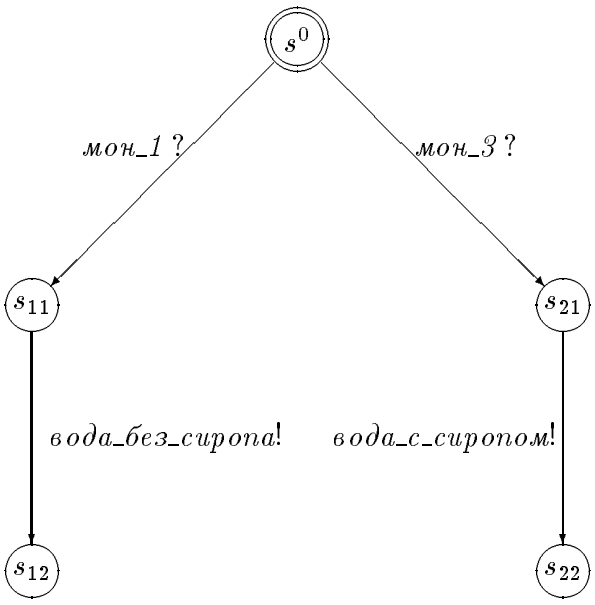


Согласно определению альтернативной композиции, графовое представление процесса (3.2) получается добавлением к предыдущей диаграмме нового состояния и соответствующих переходов, в результате чего получается следующая диаграмма:



Поскольку состояния s_{10} и s_{20} недостижимы, то, следовательно, их (а также связанные с ними переходы) можно удалить, в результате чего получится

диаграмма



которая и является искомым графовым представлением процесса (3.2).

Рассмотрим другой пример. Опишем разменный автомат, в который можно вводить купюры достоинством в 1000 рублей. Автомат должен выдать

- либо 2 купюры по 500 рублей,
- либо 10 купюр по 100 рублей

причем выбор способа размена осуществляется независимо от желания клиента. Сразу после одного сеанса размена автомат ломается.

$$P_{\text{размен}} = \\ = \mathit{1\_no\_1000}\,?\,.(\mathit{2\_no\_500}\,!\,.\mathbf{0} \;+\; \mathit{10\_no\_100}\,!\,.\mathbf{0})$$

На этих двух примерах видно, что альтернативная композиция может использоваться для описания как минимум двух принципиально различных ситуаций.

1. Во-первых, она может выражать зависимость поведения системы от поведения её окружения.

   Например, в случае автомата $P_{\text{газ\_вода}}$, реакция автомата определяется действием покупателя, а именно, достоинством монеты, которую он ввел в автомат.

   В данном случае процесс, представляющий поведение моделируемого торгового автомата, является **детерминированным**, то есть его фукнционирование однозначно определяется входными действиями.

2. Во-вторых, на примере автомата $P_{\text{размен}}$ мы видим, что при одних и тех же входных действиях возможна различная реакция автомата.

   Это - пример **недетерминизма**, то есть неопределенности поведения системы.

   Неопределённость в поведении систем может происходить по крайней мере по двум причинам.

   (a) Во-первых, поведение систем может зависеть от **случайных факторов**.

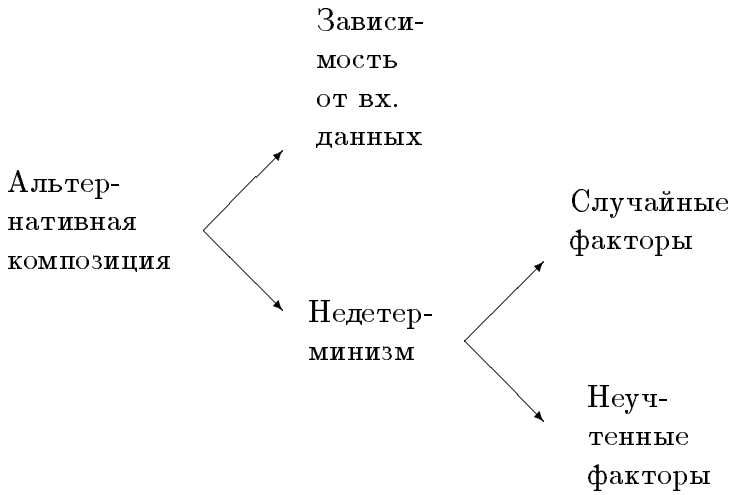
   Такими факторами могут быть, например,

   - сбои в аппаратуре,
   - коллизии в компьютерной сети
   - отсутствие купюр необходимого достоинства в банкомате
   - или что-либо еще

   (b) Во-вторых, модель всегда есть некоторая абстракция или упрощение реальной системы. А при этом некоторые факторы, влияющие на поведение этой системы, могут быть просто исключены из рассмотрения.

   В частности, на примере процесса $P_{\text{размен}}$ мы видим, что реальная причина выбора варианта поведения автомата может не учитываться в процессе, представляющем собой модель поведения этого автомата.

Можно схематически изобразить вышеперечисленные варианты использования альтернативной композиции следующим образом:



3.4 Параллельная композиция

Операция параллельной композиции используется для построения моделей поведения динамических систем, состоящих из нескольких взаимодействующих компонентов.

Прежде чем дать формальное определение этой операции, мы обсудим понятие параллельного функционирования двух систем Sys_1 и Sys_2, которые мы рассматриваем как компоненты одной системы Sys, т.е.

$$Sys \stackrel{\text{def}}{=} \{Sys_1, Sys_2\} \tag{3.3}$$

Пусть поведение систем Sys_1 и Sys_2 представлено процессами P_1 и P_2 соответственно. Поведение системы Sys_i $(i = 1, 2)$ в составе системы Sys описывается тем же процессом P_i, которым описывается поведение этой системы, рассматриваемой индивидуально.

Обозначим символом $\{P_1, P_2\}$ процесс, описывающий поведение системы (3.3). Целью этого параграфа является явное определение процесса $\{P_1, P_2\}$ (т.е. построение множеств его состояний и переходов) по информации о процессах P_1 и P_2.

Ниже для упрощения изложения мы будем отождествлять понятия

"процесс P", и

"система, поведение которой описывается процессом P"

Как было отмечено выше, функционирование произвольного процесса P может быть интерпретировано как обход графа, соответствующего этому процессу, с выполнением действий, которые являются метками проходимых рёбер.

Мы будем предполагать, что при прохождении каждого ребра $s \xrightarrow{a} s'$

- переход от s к s' происходит мгновенно, и
- факт выполнения действия a имеет место именно в момент этого перехода.

В действительности, выполнение каждого из действий происходит в течение некоторого промежутка времени, но мы будем считать, что для каждого проходимого ребра $s \xrightarrow{a} s'$

- до завершения выполнения действия a процесс P находится в состоянии s, и
- после завершения выполнения действия a процесс P мгновенно переходит в состояние s'.

Поскольку выполнение разных действий имеет разную продолжительность, то мы будем считать, что во время своего функционирования процесс P находится каждом состоянии, в которое он попадает, неопределённый промежуток времени.

Таким образом, работа процесса P предсталяет собой чередование следующих двух видов деятельности:

- ожидание в течение неопределённого промежутка времени в одном из состояний, и
- мгновенный переход из одного состояния в другое.

Ожидание в одном из состояний может происходить

- не только по причине того, что в этот момент происходит выполнение некоторого действия,
- но также и по причине того, что процесс P в текущий момент просто не может выполнить какое-либо действие.

Например, если

- $P = \alpha?.\, P'$, и
- в начальный момент никто не предлагает процессу P объект α

то P будет ждать, когда какой-либо процесс предложит ему объект α.

Как мы знаем, для каждого процесса

- его действия являются либо входными, либо выходными, либо внутренними, и
- каждое входное и выходное действие является результатом взаимодействия этого процесса с другим процессом.

Каждое входное или выходное действие процесса P_i $(i = 1, 2)$ представляет собой

- либо результат взаимодействия P_i с процессом, не входящим в совокупность $\{P_1, P_2\}$,
- либо результат взаимодействия P_i с процессом P_j, где $j \in \{1, 2\} \setminus \{i\}$.

С точки зрения процесса $\{P_1, P_2\}$, действия второго типа являются внутренними действиями этого процесса, так как они

- не представляют собой результат взаимодействия процесса $\{P_1, P_2\}$ с окружающей средой, а

- являются результатом взаимодействия компонентов этого процесса.

Таким образом, каждое действие процесса $\{P_1, P_2\}$ представляет собой

(a) либо результат взаимодействия одного из процессов P_i с процессом, не входящим в $\{P_1, P_2\}$,

(b) либо внутреннее действие одного из процессов, входящих в $\{P_1, P_2\}$ (т.е. внутреннее действие P_1 или P_2),

(c) либо внутреннее действие, которое является результатом взаимодействия процессов P_1 и P_2, и заключается в том, что

- один из этих процессов $P_i \quad (i = 1, 2)$ передаёт другому процессу $P_j \quad (j \in \{1, 2\} \setminus \{i\})$ некоторый объект, и
- процесс P_j в тот же самый момент времени принимает от процесса P_i этот объект

(такой вид взаимодействия называют **синхронным** взаимодействием, или **рукопожатием**).

Каждому возможному варианту поведения процесса $P_i \quad (i = 1, 2)$ можно сопоставить **нить**, обозначаемую символом σ_i, и представляющую собой вертикальную линию, на которой нарисованы точки с метками, где

- метки точек обозначают действия, исполняемые процессом P_i, и
- помеченные точки расположены в хронологическом порядке, т.е.
  - сначала идёт точка, помеченная первым действием процесса P_i,
  - под ней - точка, помеченная вторым действием процесса P_i,
  - и т.д.

Для каждой помеченной точки p на нити мы будем обозначать знакосочетанием $act(p)$ метку этой точки.

Представим себе, что на плоскости параллельно нарисованы нити

$$\sigma_1 \quad \sigma_2 \tag{3.4}$$

где $\sigma_i \quad (i = 1, 2)$ представляет возможный вариант поведения процесса P_i в составе процесса $\{P_1, P_2\}$.

Рассмотрим те помеченные точки на нитях из (3.4), которые соответствуют действиям типа (c), т.е взаимодействиям процессов P_1 и P_2. Пусть p - одна из таких точек, и пусть она находится, например, на нити σ_1.

Согласно определению понятия взаимодействия, в тот же самый момент времени, в который выполняется действие $act(p)$, процесс P_2 выполняет комплементарное действие, т.е. на нити σ_2 есть точка p', такая, что

- $act(p') = \overline{act(p)}$, и
- действия $act(p)$ и $act(p')$ исполняются в один и тот же момент времени.

Отметим, что

- на нити σ_2 может быть несколько точек с меткой $\overline{act(p)}$, но ровно одна из этих точек соответствует тому действию, которое исполняется совместно с действием, соответствующим точке p, и
- на нити σ_1 может быть несколько точек с меткой $act(p)$, но ровно одна из этих точек соответствует тому действию, которое исполняется совместно с действием, соответствующим точке p'.

Преобразуем нашу диаграмму нитей (3.4) следующим образом: для каждой пары точек p, p' с вышеуказанными свойствами

- соединим точки p и p' стрелкой, начало которой - та из этих точек, которая имеет метку вида $\alpha!$, а конец - точка с меткой $\alpha?$,
- нарисуем на этой стрелке метку α, и
- заменим метки точек p и p' на τ.

Стрелка, соединяющая точки p и p', называется **синхронизационной стрелкой**. Такие стрелки обычно рисуют горизонтально, для чего точки на нитях располагают так, чтобы соединяемые стрелками точки располагались на одинаковой высоте.

После того, как мы сделаем такие преобразования для всех пар точек, в которых происходят действия вида (c), у нас получится диаграмма, которую в литературе по параллельным вычислениям принято называть **Message Sequence Chart (MSC)**. Эта диаграмма представляет собой один из вариантов функционирования процесса $\{P_1, P_2\}$.

Мы будем обозначать знакосочетанием

$$Beh\{P_1, P_2\}$$

совокупность всех MSC, каждая из которых соответствует некоторому варианту функционирования процесса $\{P_1, P_2\}$.

Рассмотрим следующий пример: $\{P_1, P_2\}$ состоит из двух процессов P_1 и P_2, где

- P_1 - торговый автомат, поведенние которого задается выражением

$$P_1 = мон?.\ шок!.\mathbf{0} \tag{3.5}$$

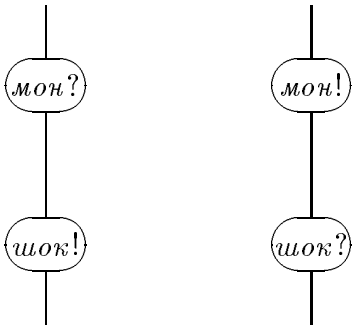
(т.е. он получает монету, выдаёт шоколадку, и после этого ломается)

- P_2 - покупатель, поведенние которого задается выражением

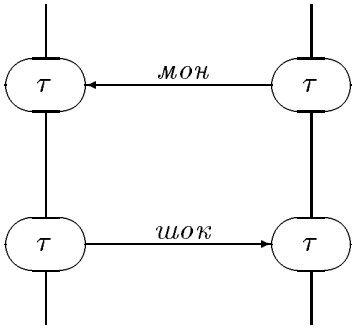
$$P_2 = мон!.\ шок?.\mathbf{0} \tag{3.6}$$

(т.е. он опускает монету, получает шоколадку, и после этого перестаёт функционировать в роли покупателя)

Нити этих процессов имеют вид



Если все действия на этих нитях являются действиями типа (c), то данная диаграмма преобразуется в следующую MSC:



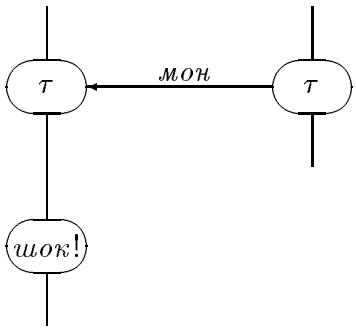
Однако возможен и следующий вариант функционирования процесса $\{P_1, P_2\}$:

- первое действие P_1 и P_2 имеет тип (c), т.е. покупатель опускает в автомат монету, а автомат её принимает
- второе действие автомата P_1 является взаимодействием с процессом, который является внешним по отношению к $\{P_1, P_2\}$, т.е., например, к автомату подошёл вор, и взял шоколадку, прежде чем это смог сделать покупатель P_2.

В этой ситуации покупатель не может выполнить второе действие как внутреннее действие процесса $\{P_1, P_2\}$. Согласно описанию процесса, в данном случае возможны два варианта поведения.

1. P_2 будет находиться в состоянии бесконечного ожидания.

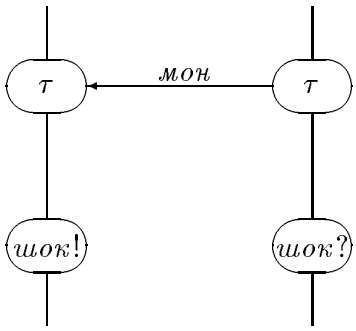
   Соответствующая этому варианту MSC имеет вид



2. P_2 сможет успешно завершить работу.

   Это произойдёт в том случае, если некоторый процесс, внешний по отношению к $\{P_1, P_2\}$, передаст шоколадку процессу P_2.

   Соответствующая этому варианту MSC имеет вид



Теперь рассмотрим вопрос о том, каким образом процесс $\{P_1, P_2\}$ может быть определён явно, т.е. в терминах состояний и переходов с метками из множества Act.

На первый взгляд, данный вопрос является некорректным, так как $\{P_1, P_2\}$ представляет собой модель **параллельного** функционирования процессов P_1 и P_2, при котором

- в один и тот же момент времени могут быть исполнены действия обоими процессами, входящими в $\{P_1, P_2\}$, и,
- следовательно, процесс $\{P_1, P_2\}$ может совершать такие действия, которые представляют собой пары действий из множества Act, т.е. они не принадлежат множеству Act.

Заметим на это, что абсолютная одновременность имеет место лишь для тех пар действий, которые порождают одно внутреннее действие процесса $\{P_1, P_2\}$ типа (c). Для всех остальных пар действий процессов P_1 и P_2, даже если они произошли с точки зрения внешнего наблюдателя одновременно, мы можем предполагать без ограничения общности, что одно из них произошло немного раньше или немного позже другого.

Таким образом, мы можем считать, что процесс $\{P_1, P_2\}$ функционирует последовательно, т.е. что при любом варианте функционирования процесса $\{P_1, P_2\}$ выполняемые им действия образуют некоторую линейно упорядоченную последовательность

$$tr = (act_1, act_2, \ldots) \tag{3.7}$$

в которой действия упорядочены по времени их выполнения: сначала произошло act_1, затем - act_2, и т.д.

Поскольку каждый возможный вариант функционирования процесса $\{P_1, P_2\}$ представляется некоторой MSC, то можно считать, что последовательность (3.7) получается некоторой *линеаризацией* этой MSC (т.е. "вытягиванием" её в цепочку).

Для определения понятия линеаризации MSC мы введём несколько вспомогательных понятий и обозначений. Пусть C – некоторая MCS. Тогда

- знакосочетание $Points(C)$ обозначает множество всех точек, входящих в MSC C,
- для каждой точки $p \in Points(C)$ знакосочетание $act(p)$ обозначает действие, приписанное точке p
- для каждой пары точек $p, p' \in Points(C)$ знакосочетание

  $$p \to p'$$

  означает, что имеет место одно из следующих условий:

  - p и p' находятся на одной и той же нити, и p' расположена ниже, чем p, или
  - существует синхронизационная стрелка с началом в p и концом в p'
- для каждой пары точек $p, p' \in Points(C)$ знакосочетание

  $$p \leq p'$$

  означает, что либо $p = p'$, либо существует последовательность точек $p_1, \ldots, p_k$, такая, что

  - $p = p_1, \;\; p' = p_k$

– для каждого $i = 1, \ldots, k-1 \quad p_i \to p_{i+1}$

Отношение $\leq$ на точках MSC можно рассматривать как отношение хронологической упорядоченности, т.е. знакосочетание $p \leq p'$ можно интерпретировать как утверждение о том, что

- точки p и p' совпадают или соединены синхронизационной стрелкой (т.е. действия в p и в p' совпадают),
- или действие в p' произошло позже, чем произошло действие в p.

Точное определение понятия линеаризации MSC имеет следующий вид. Пусть заданы MSC C и последовательность действий tr вида (3.7). Обозначим множество индексов элементов последовательности tr знакосочетанием $Ind(tr)$, т.е.

$$Ind(tr) = \{1, 2, \ldots\}$$

(данное множество может быть как конечным, так и бесконечным).

Последовательность tr называется **линеаризацией** MSC C, если существует сюръективное отображение

$$lin : Points(C) \to Ind(tr)$$

удовлетворяющее следующим условиям.

1. для каждой пары $p, p' \in Points(C)$

$$p \leq p' \quad \Rightarrow \quad lin(p) \leq lin(p')$$

2. для каждой пары $p, p' \in Points(C)$ соотношение

$$lin(p) = lin(p')$$

имеет место тогда и только тогда, когда

- $p = p'$, или
- существует синхронизационная стрелка с началом в p и концом в p'

3. для каждой точки $p \in Points(C)$

$$act(p) = act_{lin(p)}$$

т.е. отображение lin

- сохраняет хронологический порядок,

- отождествляет те точки MSC C, которые соответствуют одному действию процесса $\{P_1, P_2\}$, и
- не отождествляет никакие другие точки.

Обозначим символом $Lin(C)$ совокупность всех линеаризаций MSC C.

Теперь задачу явного описания процесса $\{P_1, P_2\}$ можно сформулировать следующим образом: построить процесс P, удовлетворяющий условию

$$Tr(P) = \bigcup_{C \in Beh\{P_1,P_2\}} Lin(C) \tag{3.8}$$

т.е. в процессе P должны быть представлены все линеаризации любого возможного совместного поведения процессов P_1 и P_2.

Условие (3.8) обосновывается следующим соображением: поскольку мы не знаем,

- как согласованы между собой часы в процессах P_1 и P_2, и
- какова продолжительность пребывания в каждом из состояний, в которые эти процессы попадают

то мы должны считать возможным любой порядок исполнения действий, который не противоречит отношению хронологической упорядоченности.

Приступим к построению процесса P, удовлетворяющему условию (3.8). Пусть процессы P_1 и P_2 имеют вид

$$P_i = (S_i, s_i^0, R_i) \quad (i = 1, 2)$$

Рассмотрим произвольную линеаризацию tr произвольной MSC из $Beh\{P_1, P_2\}$

$$tr \;=\; (\; a_1,\; a_2,\; \ldots\;)$$

Нарисуем линию, которую мы будем интерпретировать как временну́ю шкалу. Выделим на ней точки p_1, p_2, ..., помеченные действиями $a_1, a_2, \ldots$, причём помеченные точки расположены в том порядке, в котором соответствующие действия перечислены в tr, т.е. сначала идёт точка p_1 с меткой a_1, за ней - точка p_2 с меткой a_2, и т.д.

Обозначим символами $I_0, I_1, I_2, \ldots$ следующие участки этой линии:

- I_0 представляет собой совокупность всех точек линии перед точкой p_1, т.е.
$$I_0 \stackrel{\text{def}}{=} \;]-\infty, p_1[$$
- для каждого $i \geq 1$ участок I_i состоит из точек между p_i и p_{i+1}, т.е.
$$I_i \stackrel{\text{def}}{=} \;]p_i, p_{i+1}[$$

Каждый из этих участков I_i можно интерпретировать как промежуток времени, в течение которого процесс P не выполняет никаких действий, т.е. в моменты времени между p_i и p_{i+1} процессы P_1 и P_2 находятся в фиксированных состояниях $(s_1)_i$ и $(s_2)_i$ соответственно.

Обозначим символом s_i пару $((s_1)_i, (s_2)_i)$. Данную пару можно интерпретировать как состояние всего процесса P, в котором он находится в каждый момент времени из промежутка I_i.

По определению последовательности tr, имеет место одна из двух ситуаций.

1. Действие a_i имеет тип (a) или (b), т.е. оно было выполнено одним из процессов, входящих в P.

   Возможны два случая.

   (a) Это действие было выполнено процессом P_1.

   В этом случае мы имеем следующую связь между состояниями s_i и s_{i+1}:

   - $(s_1)_i \xrightarrow{a_i} (s_1)_{i+1} \quad \in R_1$
   - $(s_2)_{i+1} = (s_2)_i$

   (b) Это действие было выполнено процессом P_2.

   В этом случае мы имеем следующую связь между состояниями s_i и s_{i+1}:

   - $(s_2)_i \xrightarrow{a_i} (s_2)_{i+1} \quad \in R_2$
   - $(s_1)_{i+1} = (s_1)_i$

2. Действие a_i имеет тип (c).

   В этом случае мы имеем следующую связь между состояниями s_i и s_{i+1}:

   - $(s_1)_i \xrightarrow{a} (s_1)_{i+1} \quad \in R_1$
   - $(s_2)_i \xrightarrow{\overline{a}} (s_2)_{i+1} \quad \in R_2$

   для некоторого $a \in Act \setminus \{\tau\}$.

Сформулированные выше свойства последовательности tr можно переформулировать следующим образом: tr является трассой процесса

$$(S, s^0, R) \tag{3.9}$$

компоненты которого определяются следующим образом:

- $S \stackrel{\text{def}}{=} S_1 \times S_2 \stackrel{\text{def}}{=} \{(s_1, s_2) \mid s_1 \in S_1, s_2 \in S_2\}$
- $s^0 \stackrel{\text{def}}{=} (s_1^0, s_2^0)$

- для
  - каждого перехода $s_1 \xrightarrow{a} s_1'$ из R_1, и
  - каждого состояния $s \in S_2$

  R содержит переход
  $$(s_1, s) \xrightarrow{a} (s_1', s)$$
- для
  - каждого перехода $s_2 \xrightarrow{a} s_2'$ из R_2, и
  - каждого состояния $s \in S_1$

  R содержит переход
  $$(s, s_2) \xrightarrow{a} (s, s_2')$$
- для каждой пары переходов с комплементарными метками
  $$\begin{array}{lll} s_1 \xrightarrow{a} s_1' & \in R_1 \\ s_2 \xrightarrow{\overline{a}} s_2' & \in R_2 \end{array}$$

  R содержит переход
  $$(s_1, s_2) \xrightarrow{\tau} (s_1', s_2')$$

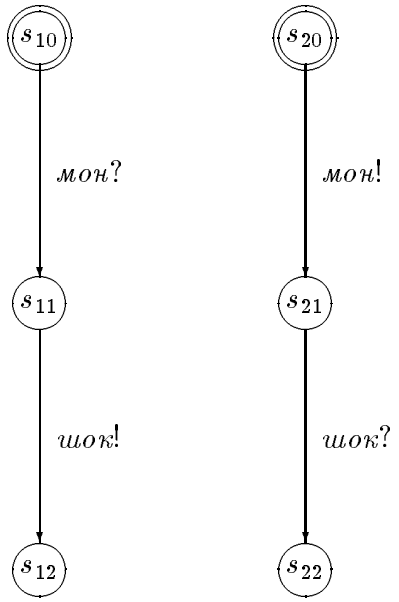
Нетрудно показать и обратное: каждая трасса в вышеопределённом процессе (3.9) является линеаризацией некоторой MSC C из множества $Beh\{P_1, P_2\}$.

Таким образом, в качестве искомого процесса P можно взять процесс (3.9). Данный процесс называется **параллельной композицией** процессов P_1 и P_2, и обозначается знакосочетанием

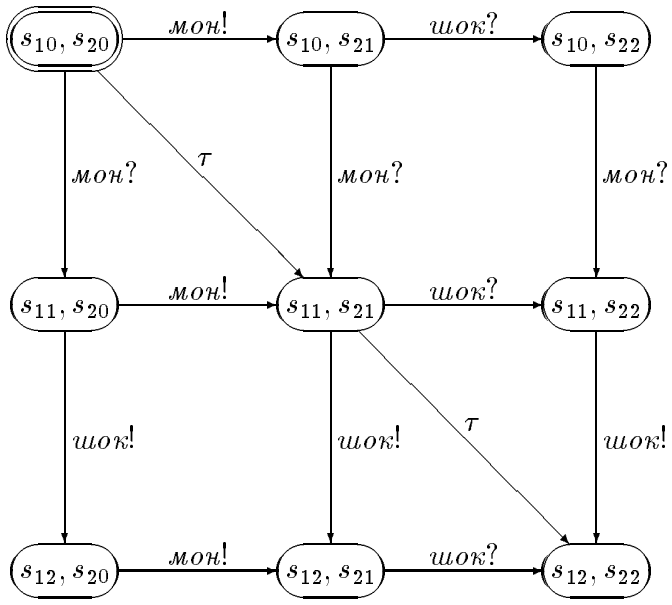
$$P_1 \,|\, P_2$$

Приведём в качестве примера процесс $P_1 \,|\, P_2$, где процессы P_1 и P_2 представляют поведение торгового автомата и покупателя (см. (3.5) и (3.6)).

Графовые представления этих процессов имеют вид



Графовое представление процесса $P_1|P_2$ имеет вид



Заметим, что размер множества состояний процесса $P_1|P_2$ равен произведению размеров множеств состояний процессов P_1 и P_2, т.е. размер описания процесса $P_1 \,|\, P_2$ может существенно превышать суммарную сложность размеров описаний его компонентов P_1 и P_2. Это может сделать невозможным анализ такого процесса по причине его высокой сложности.

Поэтому в практических задачах при анализе процессов вида $P_1 \mid P_2$, вместо явного построения процесса $P_1 \mid P_2$ строится процесс, в котором каждая MSC из $Beh\{P_1, P_2\}$ представлена не всеми возможными линеаризациями, а хотя бы одной линеаризацией. Сложность такого процесса может быть существенно меньше по сравнению со сложностью процесса $P_1|P_2$.

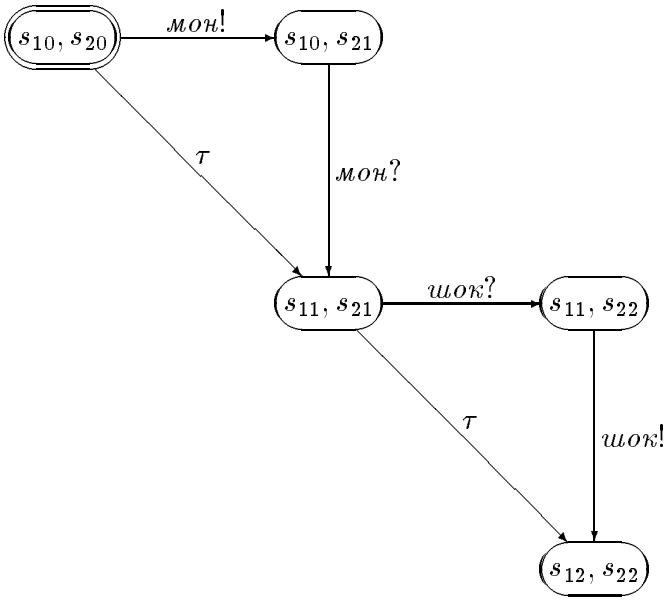
Построение процесса такого вида имеет смысл, например, в том случае, когда анализируемое свойство φ процесса $P_1 \mid P_2$ обладает следующим качеством:

- если φ истинно *для одной из линеаризаций* произвольной MSC $C \in Beh\{P_1, P_2\}$,
- то φ истинно *для всех линеаризаций* этой MSC.

Как правило, процесс, в котором каждая MSC из $Beh\{P_1, P_2\}$ представлена не всеми возможными линеаризациями, а хотя бы одной линеаризацией, строится как некоторый **подпроцесс** процесса $P_1|P_2$, т.е. получается из $P_1|P_2$ удалением некоторых состояний и связанных с ними переходов. Поэтому такие процессы называют **редуцированными**.

Проблема построения редуцированных процессов называется проблемой **редукции частичных порядков (partial order reduction)**. Эта проблема интенсивно исследуется многими ведущими специалистами в области верификации.

Приведём в качестве примера редуцированный процесс для рассмотренного выше процесса $P_1 \mid P_2$, состоящего из торгового автомата и покупателя.



В заключение отметим, что задача анализа процессов, состоящих из нескольких взаимодействующих компонентов, наиболее часто возникает в си-

туации, когда такими компонентами являются компьютерные программы и аппаратные устройства, функционирующие совместно в рамках единого программно-аппаратного комплекса.

Взаимодействие между такими программами осуществляется при помощи процессов-посредников, т.е. программы взаимодействуют друг с другом через посредство некоторых процессов, которые синхронно взаимодействуют с каждой из программ.

Взаимодействие между параллельно работающими программами обычно реализуется одним из следующих двух способов.

1. **Взаимодействие через общую память.**

   В данном случае такими посредниками являются ячейки памяти, к которым имеют доступ обе программы.

   Взаимодействие может осуществляться, например, так: одна программа пишет информацию в эти ячейки, а другая читает содержимое ячеек.

2. **Взаимодействие путем посылки сообщений.**

   В данном случае посредником является канал, с которым программы могут осуществлять следующие операции:

   - посылка сообщения в канал передающей программой, и
   - приём сообщения принимающей программой.

   Канал может представлять собой буфер, хранящий несколько сообщений. Сообщения в канале могут быть организованы по принципу очереди (т.е. сообщения покидают канал в том же порядке, в котором они в него поступают).

3.5 Ограничение

Пусть $P = (S, s^0, R)$ некоторый процесс, и L – произвольное подмножество множества $Names$.

Ограничением P по L называется процесс

$$P \setminus L = (S, s^0, R')$$

который получается из P удалением тех переходов, которые имеют метки с именами из L, т.е.

$$R' \stackrel{\text{def}}{=} \left\{ \left(s \xrightarrow{a} s' \right) \in R \;\middle|\; \begin{array}{l} a = \tau, \text{ или} \\ name(a) \notin L \end{array} \right\}$$

Операция ограничения используется, как правило, совместно с операцией параллельной композиции для представления таких процессов, которые

- состоят из нескольких компонентов, и
- взаимодействие между этими компонентами должно удовлетворять некоторым ограничениям.

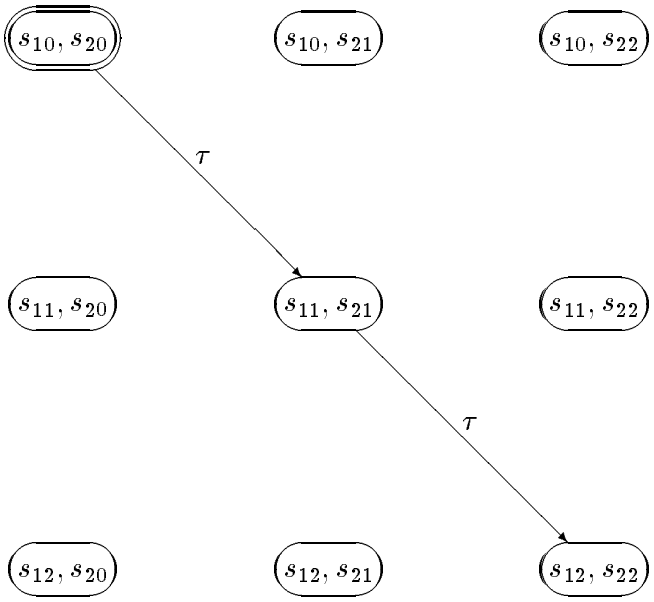
Например, пусть P_1 и P_2 – торговый автомат и покупатель, которые рассматривались в предыдущем параграфе.

Мы хотели бы описать процесс, являющийся моделью такого параллельного функционирования процессов P_1 и P_2, при котором эти процессы могут совершать действия, связанные с покупкой-продажей шоколадки, только совместно.

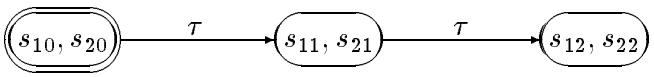
Искомый процесс может быть получен из процесса $P_1|P_2$ применением операции ограничения по множеству имён всех действий, которые связаны с покупкой-продажей шоколадки, т.е. данный процесс описывается выражением

$$P \stackrel{\text{def}}{=} (P_1|P_2) \setminus \{мон, шок\} \tag{3.10}$$

Графовое представление процесса (3.10) имеет вид



После удаления недостижимых состояний получится процесс, имеющий следующее графовое представление:

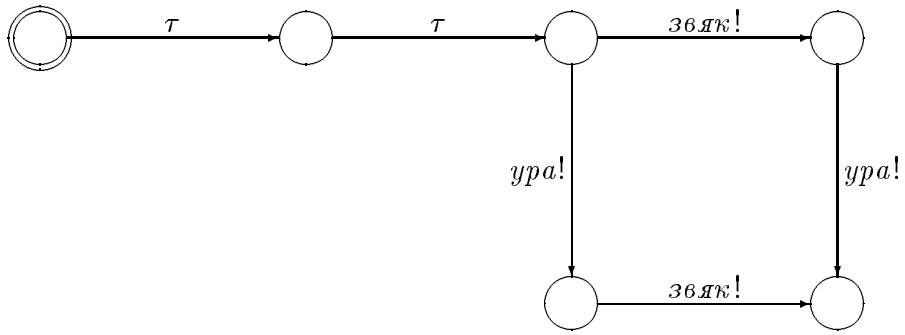


Рассмотрим другой пример. Немного изменим определения торгового автомата и покупателя: пусть они еще и сигнализируют об успешном выпол-

нении своей работы, т.е. их процессы имеют, например, следующий вид:

$$P_1 \stackrel{\text{def}}{=} \textit{мон}?.\textit{шок}!.\textit{звяк}!.\mathbf{0}$$
$$P_2 \stackrel{\text{def}}{=} \textit{мон}!.\textit{шок}?.\textit{ура}!.\mathbf{0}$$

В этом случае графовое представление процесса (3.10) после удаления недостижимых состояний имеет вид



Данный процесс допускает исполнение тех невнутренних действий, которые не связаны с покупкой и продажей шоколадки.

Отметим, что в данном случае в процессе (3.10) присутствует недетерминизм, хотя в компонентах P_1 и P_2 его нет. Причиной возникновения этого недетерминизма является наше неполное знание о моделируемой системе: поскольку мы не имеем точных знаний о длительности действий *звяк*! и *ура*!, то модель системы должна допускать любой порядок их выполнения.

3.6 Переименование

Следующая операция, которую мы рассмотрим - это унарная операция **переименования**.

Для задания этой операции необходимо определить функцию

$$f : Names \to Names$$

называемую **переименованием**.

Действие данной операции на процесс P заключается в изменении меток переходов:

- метки вида $\alpha?$ заменяются на $f(\alpha)?$, и
- метки вида $\alpha!$ заменяются на $f(\alpha)!$

Получившийся процесс обозначается знакосочетанием $P[f]$.

Если переименование f действует нетождественно лишь на имена из списка

$$\alpha_1, \ldots, \alpha_n$$

и отображает их в имена

$$\beta_1, \ldots, \beta_n$$

соответственно, то для $P[f]$ мы будем иногда использовать эквивалентное обозначение

$$P[\beta_1/\alpha_1, \ldots, \beta_n/\alpha_n]$$

Операция переименования позволяет многократно использовать один и тот же процесс P в качестве компоненты при построении более сложного процесса P'. Эта операция используется для предотвращения "конфликтов" между именами действий, используемых в различных вхождениях P в P'.

3.7 Свойства операций на процессах

В этом параграфе мы приводим некоторые простейшие свойства определённых выше операций на процессах. Все эти свойства имеют вид равенств. Для первых двух свойств мы даём их обоснование, остальные свойства приводятся без комментариев ввиду их очевидности.

Напомним (см. параграф 2.7), что мы считаем два процесса равными, если

- они изоморфны, или
- один из этих процессов можно получить из другого путём удаления некоторых недостижимых состояний и связанных с ними переходов.

1. Операция + ассоциативна, т.е. для любых процессов P_1, P_2 и P_3 верно равенство

$$(P_1 + P_2) + P_3 = P_1 + (P_2 + P_3)$$

Действительно, пусть процессы P_i $(i = 1, 2, 3)$ имеют вид

$$P_i = (S_i, s_i^0, R_i) \quad (i = 1, 2, 3) \tag{3.11}$$

причём множества состояний S_1, S_2 и S_3 попарно не пересекаются. Тогда обе части данного равенства равны процессу $P = (S, s^0, R)$, компоненты которого определяются следующим образом:

- $S \stackrel{\text{def}}{=} S_1 \cup S_2 \cup S_3 \cup \{s^0\}$, где s^0 – новое состояние, не входящее в S_1, S_2 и S_3
- R содержит все переходы из R_1, R_2 и R_3

- для каждого перехода из R_i $(i = 1, 2, 3)$ вида

$$s_i^0 \xrightarrow{a} s$$

R содержит переход $s^0 \xrightarrow{a} s$

Из свойства ассоциативности операции + следует, что допустимы выражения вида

$$P_1 + \ldots + P_n \tag{3.12}$$

так как при любой расстановке скобок в выражении (3.12) получится один и тот же процесс.

Процесс, являющийся значением выражения (3.12) можно описать явно следущим образом.

Пусть процессы P_i $(i = 1, \ldots, n)$ имеют вид

$$P_i = (S_i, s_i^0, R_i) \quad (i = 1, \ldots, n) \tag{3.13}$$

причём множества состояний $S_1, \ldots, S_n$ попарно не пересекаются. Тогда процесс, являющийся значением выражения (3.12), имеет вид

$$P = (S, s^0, R)$$

где компоненты S, s^0, R определяются следующим образом:

- $S \stackrel{\text{def}}{=} S_1 \cup \ldots \cup S_n \cup \{s^0\}$, где s^0 – новое состояние, не входящее в $S_1, \ldots, S_n$
- R содержит все переходы из $R_1, \ldots, R_n$
- для каждого перехода из R_i $(i = 1, \ldots, n)$ вида

$$s_i^0 \xrightarrow{a} s$$

R содержит переход $s^0 \xrightarrow{a} s$

2. Операция $\mid$ ассоциативна, т.е. для любых процессов P_1, P_2 и P_3 верно равенство

$$(P_1 \mid P_2) \mid P_3 = P_1 \mid (P_2 \mid P_3)$$

Действительно, пусть процессы P_i $(i = 1, 2, 3)$ имеют вид (3.11). Тогда обе части данного равенства равны процессу $P = (S, s^0, R)$ компоненты которой определяются следующим образом:

- $S \stackrel{\text{def}}{=} S_1 \times S_2 \times S_3 \stackrel{\text{def}}{=}$
$\stackrel{\text{def}}{=} \{(s_1, s_2, s_3) \mid s_1 \in S_1, s_2 \in S_2, s_3 \in S_3\}$

- $s^0 \stackrel{\text{def}}{=} (s_1^0, s_2^0, s_3^0)$
- для
  - каждого перехода $s_1 \xrightarrow{a} s_1'$ из R_1, и
  - каждой пары состояний $s_2 \in S_2, s_3 \in S_3$

  R содержит переход

$$(s_1, s_2, s_3) \xrightarrow{a} (s_1', s_2, s_3)$$

- для
  - каждого перехода $s_2 \xrightarrow{a} s_2'$ из R_2, и
  - каждой пары состояний $s_1 \in S_1, s_3 \in S_3$

  R содержит переход

$$(s_1, s_2, s_3) \xrightarrow{a} (s_1, s_2', s_3)$$

- для
  - каждого перехода $s_3 \xrightarrow{a} s_3'$ из R_3, и
  - каждой пары состояний $s_1 \in S_1, s_2 \in S_2$

  R содержит переход

$$(s_1, s_2, s_3) \xrightarrow{a} (s_1, s_2, s_3')$$

- для
  - каждой пары переходов с комплементарными метками

$$\begin{array}{lll} s_1 \xrightarrow{a} s_1' & \in R_1 \\ s_2 \xrightarrow{\overline{a}} s_2' & \in R_2 \end{array}$$

    и
  - каждого состояния $s_3 \in S_3$

  R содержит переход

$$(s_1, s_2, s_3) \xrightarrow{\tau} (s_1', s_2', s_3)$$

- для
  - каждой пары переходов с комплементарными метками

$$\begin{array}{lll} s_1 \xrightarrow{a} s_1' & \in R_1 \\ s_3 \xrightarrow{\overline{a}} s_3' & \in R_3 \end{array}$$

    и
  - каждого состояния $s_2 \in S_2$

R содержит переход

$$(s_1, s_2, s_3) \xrightarrow{\tau} (s_1', s_2, s_3')$$

- для
  - каждой пары переходов с комплементарными метками

$$\begin{array}{lll} s_2 & \xrightarrow{a} & s_2' \quad \in R_2 \\ s_3 & \xrightarrow{\overline{a}} & s_3' \quad \in R_3 \end{array}$$

    и
  - каждого состояния $s_1 \in S_1$

  R содержит переход

$$(s_1, s_2, s_3) \xrightarrow{\tau} (s_1, s_2', s_3')$$

Из свойства ассоциативности операции $\mid$ следует, что допустимы выражения вида

$$P_1 \mid \ldots \mid P_n \tag{3.14}$$

так как при любой расстановке скобок в выражении (3.14) получится один и тот же процесс.

Процесс, являющийся значением выражения (3.14) можно описать явно следущим образом.

Пусть процессы P_i $(i = 1, \ldots, n)$ имеют вид (3.13). Тогда процесс, являющийся значением выражения (3.14), имеет вид

$$P = (S, s^0, R)$$

где компоненты S, s^0, R определяются следующим образом:

- $S \stackrel{\text{def}}{=} S_1 \times \ldots \times S_n \stackrel{\text{def}}{=}$
  $\stackrel{\text{def}}{=} \{(s_1, \ldots, s_n) \mid s_1 \in S_1, \ldots, s_n \in S_n\}$
- $s^0 \stackrel{\text{def}}{=} (s_1^0, \ldots, s_n^0)$
- для
  - каждого $i \in \{1, \ldots, n\}$
  - каждого перехода $s_i \xrightarrow{a} s_i'$ из R_i, и
  - каждого списка состояний

$$s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n$$

    где $\forall\, j \in \{1, \ldots, n\} \quad s_j \in S_j$

R содержит переход

$$(s_1, \ldots, s_n) \xrightarrow{a} (s_1, \ldots, s_{i-1}, s'_i, s_{i+1}, \ldots, s_n)$$

- для
  - каждой пары индексов $i, j \in \{1, \ldots, n\}$, где $i < j$
  - каждой пары переходов с комплементарными метками

$$\begin{array}{ll} s_i \xrightarrow{a} s'_i & \in R_i \\ s_j \xrightarrow{\overline{a}} s'_j & \in R_j \end{array}$$

  и
  - каждого списка состояний

$$s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_{j-1}, s_{j+1}, \ldots, s_n$$

  где $\forall\ k \in \{1, \ldots, n\} \quad s_k \in S_k$

  R содержит переход

$$(s_1, \ldots, s_n) \xrightarrow{\tau} (s_1, \ldots, s_{i-1}, s'_i, s_{i+1}, \ldots, s_{j-1}, s'_j, s_{j+1}, \ldots, s_n)$$

3. Операция $+$ коммутативна, т.е. для любых процессов P_1 и P_2 верно равенство
$$P_1 + P_2 = P_2 + P_1$$

4. Операция $\mid$ коммутативна, т.е. для любых процессов P_1 и P_2 верно равенство
$$P_1 \mid P_2 = P_2 \mid P_1$$

5. $\mathbf{0}$ является нейтральным элементом относительно операции $\mid$:
$$P \mid \mathbf{0} = P$$

   Для операции $+$ аналогичное свойство тоже имеет место, если вместо равенства процессов будет использовано понятие сильной эквивалентности процессов (которое определяется ниже). Данное свойство, а также свойство идемпотентности операции $+$ доказываются в параграфе 4.5 (теорема 4).

6. $\mathbf{0} \setminus L = \mathbf{0}$

7. $\mathbf{0}[f] = \mathbf{0}$

8. $P \setminus L = P$, если $L \cap names(Act(P)) = \emptyset$.

   (напомним, что $Act(P)$ обозначает множество действий $a \in Act \setminus \{\tau\}$, таких, что P содержит переход с меткой a)

9. $(a.P) \setminus L = \begin{cases} \mathbf{0}, & \text{если } a \neq \tau \text{ и } name(a) \in L \\ a.(P \setminus L), & \text{иначе} \end{cases}$

10. $(P_1 + P_2) \setminus L = (P_1 \setminus L) + (P_2 \setminus L)$

11. $(P_1 \mid P_2) \setminus L = (P_1 \setminus L) \mid (P_2 \setminus L)$, если

$$L \cap names(Act(P_1) \cap \overline{Act(P_2)}) = \emptyset$$

12. $(P \setminus L_1) \setminus L_2 = P \setminus (L_1 \cup L_2)$

13. $P[f] \setminus L = (P \setminus f^{-1}(L))[f]$

14. $P[id] = P$, где id – тождественная функция

15. $P[f] = P[g]$, если сужения функций f и g на множество $names(Act(P))$ совпадают.

16. $(a.P)[f] = f(a).(P[f])$

17. $(P_1 + P_2)[f] = P_1[f] + P_2[f]$

18. $(P_1 \mid P_2)[f] = P_1[f] \mid P_2[f]$, если сужение функции f на множество

$$names(Act(P_1) \cup Act(P_2))$$

является инъективной функцией.

19. $(P \setminus L)[f] = P[f] \setminus f(L)$, если f иньективна.

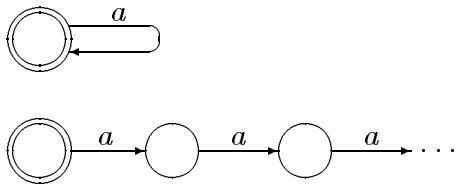
20. $P[f][g] = P[g \circ f]$

Глава 4

Эквивалентность процессов

4.1 Понятие эквивалентности процессов и связанные с ним задачи

Одно и то же поведение может быть представлено различными процессами. Например, рассмотрим два процесса:



Хотя у первого процесса всего одно состояние, а у второго множество состояний бесконечно, но эти процессы представляют одно и то же поведение, которое заключается в постоянном выполнении одного и того же действия a.

Представляет интерес поиск подходящего определения эквивалентности процессов, согласно которому процессы эквивалентны тогда и только тогда, когда они имеют одно и то же поведение.

В этой главе мы излагаем несколько определений понятия эквивалентности процессов. Выбор того или иного варианта данного понятия в конкретной ситуации должен определяться тем, как именно в данной ситуации понимается одинаковость поведения процессов.

В параграфах 4.2 и 4.3 вводятся понятия трассовой эквивалентности и сильной эквивалентности процессов. Данные понятия используются в той ситуации, когда все действия, выполняющиеся в процессах, имеют одинаковый статус.

В параграфах 4.8 и 4.9 предлагаются другие варианты понятия эквивалентности процессов: наблюдаемая эквивалентность и наблюдаемая конгруэнция. Данные понятия используются в тех ситуациях, когда мы рассматри-

ваем невидимое действие τ как несущественное, и считаем две трассы одинаковыми, если одна может быть получена из другой путём вставок и/или удалений невидимых действий τ.

С каждым из возможных вариантов понятия эквивалентности процессов связаны две естественные задачи.

1. Распознавание для двух заданных процессов, являются ли они эквивалентными.

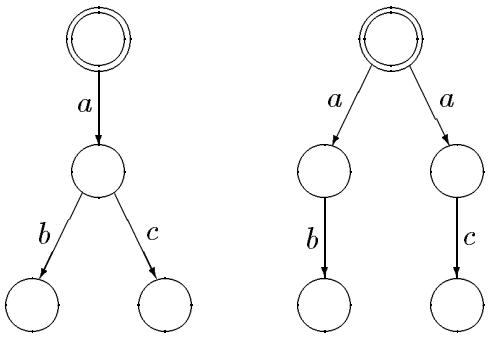
2. Построение по заданному процессу P такого процесса P', который является наименее сложным (например, имеет минимальное число состояний) среди всех процессов, которые эквивалентны P.

4.2 Трассовая эквивалентность процессов

Как уже было сказано выше, мы хотели бы рассматривать два процесса как эквивалентные, если они описывают одно и то же поведение. Поэтому, если мы рассматриваем поведение процесса как порождение некоторой трассы, то необходимым условием эквивалентности процессов P_1 и P_2 является совпадение множеств их трасс:

$$Tr(P_1) = Tr(P_2) \tag{4.1}$$

В некоторых ситуациях условие (4.1) можно использовать в качестве определения эквивалентности между P_1 и P_2. Однако нижеследующий пример показывает, что это условие не отражает один важный аспект поведения процессов.



(4.2)

Множества трасс этих процессов совпадают:

$$Tr(P_1) = Tr(P_2) = \{\varepsilon, a, ab, ac\}$$

(где ε – пустая последовательность).

Однако эти процессы отличаются тем, что

- в левом процессе после выполнения первого действия (a) сохраняется возможность выбора следующего действия (b или c), в то время как
- в правом процессе после выполнения первого действия такого выбора нет:
  - если первый переход был совершён по левому ребру, то вторым действием может быть только действие b,
  - а если по правому - то только действие c,

  т.е. второе действие было предопределено ещё до выполнения первого действия.

Если мы не хотели бы рассматривать эти процессы как эквивалентные, то условие (4.1) надо некоторым образом усилить. Один из вариантов такого усиления излагается ниже. Для того, чтобы его сформулировать, определим сначала понятие трассы из некоторого состояния процесса.

Каждый вариант поведения процесса $P = (S, s^0, R)$ мы интерпретируем как порождение некоторой последовательности переходов

$$s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} \ldots \tag{4.3}$$

начинающейся с начального состояния, т.е. $s_0 = s^0$.

Мы можем рассматривать порождение последовательности (4.3) не только с начального, а с произвольного состояния $s \in S$, т.е. рассматривать последовательность вида (4.3), в которой $s_0 = s$. Последовательность $(a_1, a_2, \ldots)$ меток этих переходов мы будем называть **трассой с началом в** s. Множество всех таких трасс мы будем обозначать знакосочетанием $Tr_s(P)$.

Пусть P_1 и P_2 – процессы вида

$$P_i = (S_i, s_i^0, R_i) \qquad (i = 1, 2)$$

Рассмотрим произвольную конечную последовательность переходов в P_1 вида

$$s_1^0 = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} s_n \quad (n \geq 0) \tag{4.4}$$

(случай $n = 0$ соответствует пустой последовательности переходов (4.4), к которой $s_n = s_1^0$)

Последовательность (4.4) можно рассматривать как начальный этап функционирования процесса P_1, а каждую трассу из $Tr_{s_n}(P_1)$ – как некоторое продолжение этого этапа.

Процессы P_1 и P_2 называются **трассово эквивалентными**, если

- для каждого начального этапа (4.4) функционирования процесса P_1 существует начальный этап функционирования процесса P_2

$$s_2^0 = s_0' \xrightarrow{a_1} s_1' \xrightarrow{a_2} \ldots \xrightarrow{a_n} s_n' \tag{4.5}$$

который имеет точно такую же трассу $a_1 \dots a_n$, что и (4.4), и в конце которого имеется точно такой же выбор дальнейшего продолжения, что и в конце этапа (4.4), т.е.

$$Tr_{s_n}(P_1) = Tr_{s'_n}(P_2) \tag{4.6}$$

- и, кроме того, имеет место симметричное условие: для каждой последовательности переходов в P_2 вида (4.5) должна существовать последовательность переходов в P_1 вида (4.4), такая, что имеет место равенство (4.6).

Данные условия имеют недостаток: в их формулировке участвуют потенциально неограниченные множества последовательностей переходов вида (4.4) и (4.5), а также потенциально неограниченные множества трасс из (4.6). Поэтому проверка данных условий представляется затруднительной даже в том случае, когда процессы P_1 и P_2 конечны.

Представляет интерес задача нахождения условий, равносильных условиям трассовой эквивалентности, которые можно было бы алгоритмически проверять для заданных процессов P_1 и P_2 в том случае, когда эти процессы конечны.

Иногда рассматривают такую эквивалентность между процессами, которая отличается от трассовой эквивалентности заменой условия (4.6) на более слабое условие:

$$Act(s_n) = Act(s'_n)$$

где для каждого состояния s знакосочетание $Act(s)$ обозначает множество всех действий $a \in Act$, таких, что существует переход с началом в s и помеченный действием a.

4.3 Сильная эквивалентность

Ещё одним вариантом понятия эквивалентности процессов является **сильная эквивалентность**. Для определения этого понятия мы введём вспомогательные обозначения.

После того, как процесс

$$P = (S, s^0, R) \tag{4.7}$$

выполнит первое действие и перейдёт в новое состояние s^1, его поведение будет неотличимо от поведения процесса

$$P' \stackrel{\text{def}}{=} (S, s^1, R) \tag{4.8}$$

имеющего те же компоненты, что и P, за исключением начального состояния.

Мы будем использовать обозначение

$$P \xrightarrow{a} P' \tag{4.9}$$

как сокращённую запись утверждения о том, что

- P и P' – процессы вида (4.7) и (4.8) соответственно, и
- R содержит переход $s^0 \xrightarrow{a} s^1$.

(4.9) можно интерпретировать как утверждение о том, что процесс P, может

- выполнить действие a, и после этого
- вести себя как процесс P'.

Понятие сильной эквивалентности основано на следующем понимании эквивалентности процессов: если мы рассматриваем процессы P_1 и P_2 как эквивалентные, то должно быть выполнено следующее условие:

- если один из этих процессов P_i может
  - выполнить некоторое действие $a \in Act$,
  - и после этого вести себя как некоторый процесс P_i'
- то и другой процесс P_j $(j \in \{1,2\} \setminus \{i\})$ тоже должен обладать способностью
  - выполнить то же самое действие a,
  - после чего вести себя как некоторый процесс P_j', который эквивалентен P_i'.

Таким образом, искомая эквивалентность должна представлять собой некоторое бинарное отношение μ на множестве всех процессов, обладающее следующими свойствами.

(1) Если $(P_1, P_2) \in \mu$, и для некоторого процесса P_1' верно утверждение

$$P_1 \xrightarrow{a} P_1' \tag{4.10}$$

то должен существовать процесс P_2', такой, что выполнены условия

$$P_2 \xrightarrow{a} P_2' \tag{4.11}$$

и

$$(P_1', P_2') \in \mu \tag{4.12}$$

(2) Симметричное свойство: если $(P_1, P_2) \in \mu$, и для некоторого процесса P_2' верно (4.11), то должен существовать процесс P_1', такой, что выполнены условия (4.10) и (4.12).

Заметим, что мы не требуем, чтобы μ было отношением эквивалентности.

Обозначим символом $\mathcal{M}$ совокупность всех бинарных отношений, которые обладают вышеприведёнными свойствами.

Множество $\mathcal{M}$ непусто: оно содержит, например, диагональное отношение, которое состоит из всех пар вида (P, P), где P – произвольный процесс.

Встаёт естественный вопрос о том, какое же из отношений, входящих в $\mathcal{M}$, можно использовать для определения понятия сильной эквивалентности.

Мы предлагаем наиболее простой ответ на этот вопрос: мы будем считать P_1 и P_2 сильно эквивалентными в том и только в том случае, когда существует хотя бы одно отношение $\mu \in \mathcal{M}$, которое содержит пару (P_1, P_2).

Таким образом, искомое отношение сильной эквивалентности на множестве всех процессов мы определяем как объединение всех отношений из $\mathcal{M}$. Данное отношение обозначается символом $\sim$.

Нетрудно доказать, что

- $\sim \in \mathcal{M}$, и
- $\sim$ является отношением эквивалентности, т.к.
  - рефлексивность $\sim$ следует из того, что диагональное отношение принадлежит $\mathcal{M}$,
  - симметричность $\sim$ следует из того, что если $\mu \in \mathcal{M}$, то $\mu^{-1} \in \mathcal{M}$
  - транзитивность $\sim$ следует из того, что если $\mu_1 \in \mathcal{M}$ и $\mu_2 \in \mathcal{M}$, то $\mu_1 \circ \mu_2 \in \mathcal{M}$.

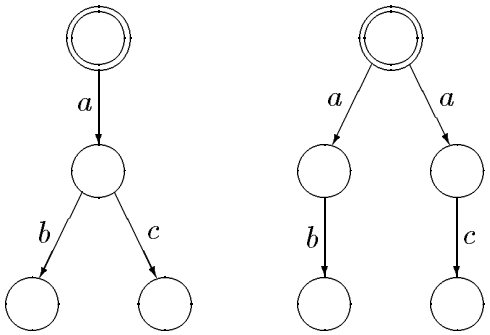
Если процессы P_1 и P_2 сильно эквивалентны, то этот факт обозначается знакосочетанием

$$P_1 \sim P_2$$

Нетрудно доказать, что если процессы P_1 и P_2 сильно эквивалентны, то они трассово эквивалентны.

Для иллюстрации понятия сильной эквивалентности рассмотрим пару примеров.

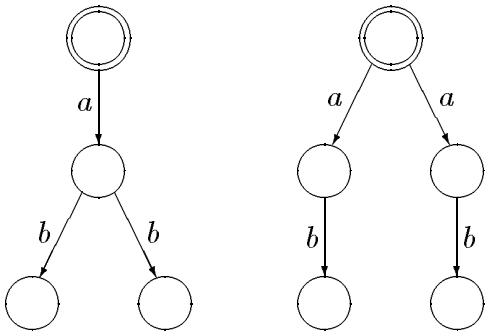
1. Процессы



(4.13)

не являются сильно эквивалентными, так как они не являются трассово эквивалентными

2. Процессы



являются сильно эквивалентными.

4.4 Критерии сильной эквивалентности

4.4.1 Логический критерий сильной эквивалентности

Обозначим символом Fm множество **формул**, определяемое следующим образом.

- Символы $\top$ и $\bot$ являются формулами.
- Если φ – формула, то $\neg\varphi$ тоже формула.
- Если φ и ψ – формулы, то $\varphi \wedge \psi$ тоже формула.
- Если φ – формула и $a \in Act$, то $\langle a \rangle \varphi$ тоже формула.

Пусть заданы процесс P и формула $\varphi \in Fm$. **Значение** формулы φ на процессе P представляет собой элемент $P(\varphi)$ множества $\{0, 1\}$, определяемый следующим образом.

- $P(\top) = 1, \quad P(\bot) = 0$
- $P(\neg\varphi) = 1 - P(\varphi)$
- $P(\varphi \wedge \psi) = P(\varphi) \cdot P(\psi)$
- $P(\langle a \rangle \varphi) = \begin{cases} 1, & \text{если существует процесс } P' : \\ & P \xrightarrow{a} P',\ P'(\varphi) = 1 \\ 0, & \text{в противном случае} \end{cases}$

Теория процесса P – это совокупность $Th(P)$ формул, определяемая следующим образом:

$$Th(P) = \{\varphi \in Fm \mid P(\varphi) = 1\}$$

Теорема 1.
Пусть процессы P_1 и P_2 конечны. Тогда

$$P_1 \sim P_2 \quad \Leftrightarrow \quad Th(P_1) = Th(P_2)$$

Доказательство.
Импликация "$\Rightarrow$" доказывается индукцией по структуре формулы φ.
Докажем импликацию "$\Leftarrow$". Пусть имеет место равенство

$$Th(P_1) = Th(P_2) \tag{4.14}$$

Обозначим символом μ бинарное отношение на множестве всех процессов, которое состоит из всех пар процессов с одинаковыми теориями, т.е.

$$\mu \stackrel{\text{def}}{=} \{(P_1, P_2) \mid Th(P_1) = Th(P_2)\}$$

Докажем, что μ удовлетворяет определению сильной эквивалентности. Пусть это не так, т.е., например, для некоторого $a \in Act$

(a) существует процесс P_1', такой, что

$$P_1 \xrightarrow{a} P_1'$$

(b) но не существует процесса P_2', такого, что

$$P_2 \xrightarrow{a} P_2' \tag{4.15}$$

и $Th(P_1') = Th(P_2')$.

Условие (b) может иметь место в двух ситуациях:

1. не существует процесса P_2', для которого верно (4.15)

2. существует процесс P_2', для которого верно (4.15), но для каждого такого процесса

$$Th(P_1') \neq Th(P_2')$$

Докажем, что в обоих ситуациях существует формула φ, удовлетворяющая условию

$$P_1(\varphi) = 1, \quad P_2(\varphi) = 0$$

что будет противоречить предположению (4.14).

1. Если имеет место первая ситуация, то в качестве φ можно взять формулу $\langle a \rangle \top$.

2. Если имеет место вторая ситуация, то пусть список всех таких процессов P_2', для которых верно (4.15), имеет вид

$$P'_{2,1}, \ldots, P'_{2,n}$$

По предположению, для каждого $i = 1, \ldots, n$ имеет место неравенство

$$Th(P_1') \neq Th(P'_{2,i})$$

т.е. для каждого $i = 1, \ldots, n$ существует формула φ_i, такая, что

$$P_1'(\varphi_i) = 1, \qquad P'_{2,i}(\varphi_i) = 0$$

В этой ситуации в качестве искомой формулы φ можно взять формулу $\langle a \rangle(\varphi_1 \wedge \ldots \wedge \varphi_n)$. ■

Например, пусть P_1 и P_2 – процессы, изображённые на рисунке (4.13). Как было сказано выше, эти процессы не являются сильно эквивалентными. В качестве обоснования утверждения $P_1 \not\sim P_2$ можно, например, предъявить формулу

$$\varphi \stackrel{\text{def}}{=} \langle a \rangle(\langle b \rangle \top \wedge \langle c \rangle \top)$$

Нетрудно доказать, что $P_1(\varphi) = 1$ и $P_2(\varphi) = 0$.

Представляет интерес задача нахождения по двум заданным процессам P_1 и P_2 списка формул

$$\varphi_1, \ldots, \varphi_n$$

как можно меньшего размера, таких, что $P_1 \sim P_2$ тогда и только тогда, когда

$$\forall\, i = 1, \ldots, n \qquad P_1(\varphi_i) = P_2(\varphi_i)$$

4.4.2 Критерий сильной эквивалентности, основанный на понятии бимоделирования

Теорема 2.

Пусть заданы два процесса

$$P_i = (S_i, s_i^0, R_i) \qquad (i = 1, 2)$$

$P_1 \sim P_2$ тогда и только тогда, когда существует отношение $\mu \subseteq S_1 \times S_2$ удовлетворяющее следующим условиям.

0. $(s_1^0, s_2^0) \in \mu$.

1. Для каждой пары $(s_1, s_2) \in \mu$ и каждого перехода из R_1 вида

   $$s_1 \xrightarrow{a} s_1'$$

   существует переход из R_2 вида

   $$s_2 \xrightarrow{a} s_2'$$

   такой, что $(s_1', s_2') \in \mu$.

2. Для каждой пары $(s_1, s_2) \in \mu$ и каждого перехода из R_2 вида

   $$s_2 \xrightarrow{a} s_2'$$

   существует переход из R_1 вида

   $$s_1 \xrightarrow{a} s_1'$$

   такой, что $(s_1', s_2') \in \mu$.

Отношение μ, удовлетворяющее данным условиям, называется **бимоделированием (БМ)** между P_1 и P_2.

4.5 Алгебраические свойства сильной эквивалентности

Теорема 3.

Сильная эквивалентность является конгруэнцией, т.е. если $P_1 \sim P_2$, то

- для каждого $a \in Act \quad a.P_1 \sim a.P_2$
- для каждого процесса $P \quad P_1 + P \sim P_2 + P$
- для каждого процесса $P \quad P_1|P \sim P_2|P$

- для каждого $L \subseteq Names \quad P_1 \setminus L \sim P_2 \setminus L$
- для каждого переименования $f \quad P_1[f] \sim P_2[f]$

Доказательство.
Как было установлено в параграфе 4.4.2, соотношение $P_1 \sim P_2$ эквивалентно тому, что существует БМ μ между P_1 и P_2. Используя это μ, мы построим БМ для обоснования каждого из вышеприведённых соотношений.

- Пусть символы $s^0_{(1)}$ и $s^0_{(2)}$ обозначают начальные состояния $a.P_1$ и $a.P_2$ соответственно.

  Тогда отношение
  $$\{(s^0_{(1)}, s^0_{(2)})\} \cup \mu$$
  является БМ между $a.P_1$ и $a.P_2$.

- Пусть
  - символы $s^0_{(1)}$ и $s^0_{(2)}$ обозначают начальные состояния P_1+P и P_2+P соответственно, и
  - символ S обозначает множество состояний процесса P.

  Тогда
  - отношение
    $$\{(s^0_{(1)}, s^0_{(2)})\} \cup \mu \cup Id_S$$
    является БМ между $P_1 + P$ и $P_2 + P$, и
  - отношение
    $$\{((s_1, s), (s_2, s)) \mid (s_1, s_2) \in \mu,\ q \in S\}$$
    является БМ между $P_1|P$ и $P_2|P$.

- Отношение μ является БМ
  - между $P_1 \setminus L$ и $P_2 \setminus L$, и
  - между $P_1[f]$ и $(P_2[f]$. ■

Теорема 4.
Для каждого процесса $P = (S, s^0, R)$ имеют место следующие свойства.

1. $P + \mathbf{0} \sim P$
2. $P + P \sim P$

Доказательство.

1. Обозначим символом символом s_0^0 начальное состояние процесса $P + \mathbf{0}$.

   БМ между $P + \mathbf{0}$ и P имеет вид

$$\{(s_0^0, s^0)\} \cup Id_S$$

2. Согласно определению операции $+$, процессы в левой части доказываемого соотношения следует рассматривать как две дизъюнктные изоморфные копии процесса P вида

$$P_{(i)} = (S_{(i)}, s^0_{(i)}, R_{(i)}) \quad (i = 1, 2)$$

   где $S_{(i)} = \{s_{(i)} \mid s \in S\}$.

   Обозначим символом символом s_0^0 начальное состояние процесса $P + P$.

   БМ между $P + P$ и P имеет вид

$$\{(s_0^0, s^0)\} \cup \{(s_{(i)}, s) \mid s \in S,\ i = 1, 2\} \quad \blacksquare$$

Ниже для

- каждого процесса $P = (S, s^0, R)$, и
- каждого состояния $s \in S$

мы будем использовать знакосочетание $P(s)$ для обозначения процесса

$$(S, s, R)$$

который отличается от P только начальным состоянием.

Теорема 5.

Пусть $P = (S, s^0, R)$ – некоторый процесс, и совокупность всех выходящих из s^0 переходов имеет вид

$$\{\, s^0 \xrightarrow{a_i} s^i \mid i = 1, \ldots, n\}$$

Тогда

$$P \sim a_1.P_1 + \ldots + a_n.P_n \tag{4.16}$$

где для каждого $i = 1, \ldots, n$

$$P_i \stackrel{\text{def}}{=} P(s^i) \stackrel{\text{def}}{=} (S, s^i, R)$$

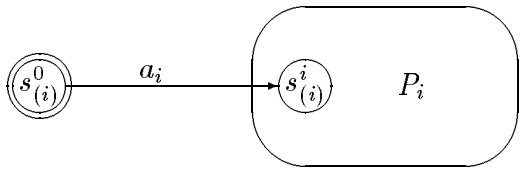
Доказательство.

Для построения БМ, доказывающего соотношение (4.16), мы заменим все процессы P_i в его правой части на их дизъюнктные копии, т.е. будем считать, что для каждого $i = 1, \ldots, n$ процесс P_i имеет вид

$$P_i = (S_{(i)}, s^i_{(i)}, R_{(i)})$$

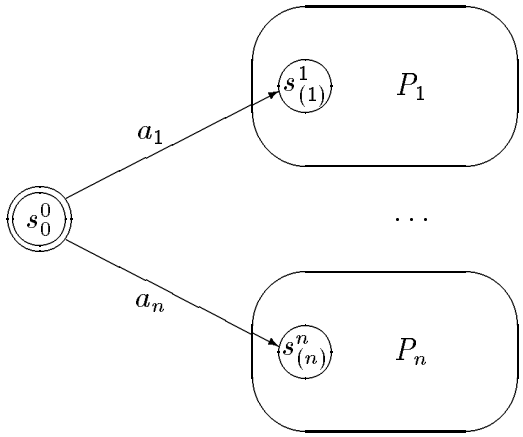
и для каждого $i = 1, \ldots, n$ соответствующая биекция между S и $S_{(i)}$ сопоставляет каждому состоянию $s \in S$ состояние, обозначаемое символом $s_{(i)}$.

Таким образом, можно считать, что каждое из слагаемых $a_i.P_i$ в правой части (4.16) имеет вид



и множества состояний этих слагаемых попарно не пересекаются.

Согласно определению операции $+$, правая часть (4.16) имеет вид



БМ между левой и правой частями (4.16) имеет вид

$$\{(s^0, s^0_0)\} \ \cup \ \{(s, s_{(i)}) \mid s \in S, \ i = 1, \ldots, n\} \quad \blacksquare$$

Теорема 6 (теорема о разложении).
Пусть P – процесс вида

$$P \ = \ P_1 \mid \ldots \mid P_n \tag{4.17}$$

где для каждого $i \in \{1, \ldots, n\}$ процесс P_i имеет вид

$$P_i = \sum_{j=1}^{n_i} a_{ij}.\, P_{ij} \tag{4.18}$$

Тогда P сильно эквивалентен сумме

1. всех процессов вида

$$a_{ij}.\left(P_1 \mid \ldots \mid P_{i-1} \mid P_{ij} \mid P_{i+1} \mid \ldots \mid P_n\right) \tag{4.19}$$

2. и всех процессов вида

$$\tau.\left(\begin{array}{l} P_1 \mid \ldots \mid P_{i-1} \mid P_{ik} \mid P_{i+1} \mid \ldots \\ \ldots \mid P_{j-1} \mid P_{jl} \mid P_{j+1} \mid \ldots \mid P_n \end{array}\right) \tag{4.20}$$

где $1 \le i < j \le n, \quad a_{ik}, a_{jl} \ne \tau$, и $a_{ik} = \overline{a_{jl}}$.

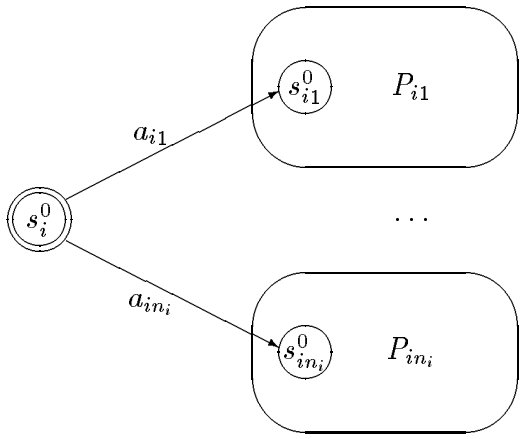
Доказательство.

По теореме 5, P сильно эквивалентен сумме, каждое слагаемое в которой соответствует некоторому переходу, выходящему из начального состояния s^0 процесса P: для каждого перехода в P вида

$$s^0 \xrightarrow{\ a\ } s$$

эта сумма содержит слагаемое $a.P(s)$.

Согласно (4.18), для каждого $i = 1, \ldots, n$ процесс P_i имеет вид



где $s^0_i, s^0_{i1}, \ldots, s^0_{in_i}$ – начальные состояния процессов

$$P_i, \quad P_{i1}, \quad \ldots, \quad P_{in_i}$$

соответственно.

Обозначим

- символом S_i множество состояний процесса P_i, и
- символом S_{ij} (где $j = 1, \ldots, n_i$) – множество состояний процесса P_{ij}.

Мы можем считать, что S_i является дизъюнктным объединением

$$S_i = \{s_i^0\} \cup S_{i1} \cup \ldots \cup S_{in_i} \tag{4.21}$$

Согласно описанию процесса вида (4.17), которое приведено в пункте 2 параграфа 3.7, можно считать, что компоненты процесса P имеют следующий вид.

- Множество состояний процесса P имеет вид

$$S_1 \times \ldots \times S_n \tag{4.22}$$

- Начальным состоянием s^0 процесса P является список

$$(s_1^0, \ldots, s_n^0)$$

- Переходы процесса P, выходящие из его начального состояния, имеют следующий вид.
  - Переходы вида

$$s^0 \xrightarrow{a_{ij}} (s_1^0, \ldots, s_{i-1}^0, s_{ij}^0, s_{i+1}^0, \ldots, s_n^0) \tag{4.23}$$

  - Переходы вида

$$s^0 \xrightarrow{\tau} \left(\begin{array}{c} s_1^0, \ldots, s_{i-1}^0, s_{ik}^0, s_{i+1}^0, \ldots \\ \ldots s_{j-1}^0, s_{jl}^0, s_{j+1}^0, \ldots, s_n^0 \end{array} \right) \tag{4.24}$$

    где $1 \leq i < j \leq n, \quad a_{ik}, a_{jl} \neq \tau$, и $a_{ik} = \overline{a_{jl}}$.

Таким образом, множество переходов процесса P, выходящих из s^0, находится во взаимно однозначном соответствии с множеством слагаемых вида (4.19) и (4.20).

Для доказательства теоремы 6 достаточно доказать, что

- Для каждого $i = 1, \ldots, n$, и каждого $j = 1, \ldots, n_i$ имеет место эквивалентность

$$\begin{array}{l} P(s_1^0, \ldots, s_{i-1}^0, s_{ij}^0, s_{i+1}^0, \ldots, s_n^0) \sim \\ \sim \left(P_1 \mid \ldots \mid P_{i-1} \mid P_{ij} \mid P_{i+1} \mid \ldots \mid P_n\right) \end{array} \tag{4.25}$$

- Для
  - любых i, j, таких, что $1 \leq j < j \leq n$, и
  - любых $k = 1, \ldots, n_i, \quad l = 1, \ldots, n_j$

имеет место эквивалентность

$$\begin{array}{l} P\left(\begin{array}{c} s_1^0, \ldots, s_{i-1}^0, s_{ik}^0, s_{i+1}^0, \ldots \\ \ldots s_{j-1}^0, s_{jl}^0, s_{j+1}^0, \ldots, s_n^0 \end{array}\right) \sim \\ \sim \left(\begin{array}{c} P_1 \mid \ldots \mid P_{i-1} \mid P_{ik} \mid P_{i+1} \mid \ldots \\ \ldots \mid P_{j-1} \mid P_{jl} \mid P_{j+1} \mid \ldots \mid P_n \end{array}\right) \end{array} \tag{4.26}$$

Мы докажем лишь эквивалентность (4.25) (эквивалентность (4.26) доказывается аналогично).

Множество состояний процесса

$$\Big(P_1 \mid \ldots \mid P_{i-1} \mid P_{ij} \mid P_{i+1} \mid \ldots \mid P_n\Big) \tag{4.27}$$

имеет вид

$$S_1 \times \ldots \times S_{i-1} \times S_{ij} \times S_{i+1} \times \ldots \times S_n \tag{4.28}$$

Из (4.21) следует, что $S_{ij} \subseteq S_i$, т.е. множество (4.28) является подмножеством множества (4.22) состояний процесса

$$P(s_1^0, \ldots, s_{i-1}^0, s_{ij}^0, s_{i+1}^0, \ldots, s_n^0) \tag{4.29}$$

Определим искомое БМ μ между процессами (4.27) и (4.29) как диагональное отношение

$$\mu \stackrel{\text{def}}{=} \{(s, s) \mid s \in (4.28)\}$$

Очевидно, что

- пара начальных состояний процессов (4.27) и (4.29) принадлежит μ,
- каждый переход процесса (4.27) является также переходом процесса (4.29), и
- если начало некоторого перехода процесса (4.29) принадлежит подмножеству (4.28), то конец этого перехода тоже принадлежит подмножеству (4.28) (для обоснования этого утверждения отметим, что все переходы в P_i с началом в S_{ij} имеют конец тоже в S_{ij}).

Таким образом, μ является БМ, и это доказывает эквивалентность (4.25). ■

Следующая теорема является усилением теоремы 6. Для её формулировки мы будем использовать следующее обозначение. Если f – произвольное переименование, то тот же символ f обозначает функцию вида

$$f : Act \to Act$$

определяемую следующим образом.

- $\forall\, \alpha \in Names \quad f(\alpha!) \stackrel{\text{def}}{=} f(\alpha)!$, и $f(\alpha?) \stackrel{\text{def}}{=} f(\alpha)?$
- $f(\tau) \stackrel{\text{def}}{=} \tau$

Теорема 7.
Пусть P – процессы вида

$$P \;=\; \Big(P_1[f_1] \mid \ldots \mid P_n[f_n]\Big) \setminus L$$

где для каждого $i \in \{1, \ldots, n\}$

$$P_i \sim \sum_{j=1}^{n_i} a_{ij}.\; P_{ij}$$

Тогда P сильно эквивалентен сумме

1. всех процессов вида

$$f_i(a_{ij}).\; \left(\left(\begin{array}{l} P_1[f_1] \mid \ldots \\ \ldots \mid P_{i-1}[f_{i-1}] \mid P_{ij}[f_i] \mid P_{i+1}[f_{i+1}] \mid \ldots \\ \ldots \mid P_n[f_n] \end{array} \right) \setminus L \right)$$

   где $a_{ij} = \tau$ или $name(f_i(a_{ij})) \notin L$, и

2. всех процессов вида

$$\tau.\; \left(\left(\begin{array}{l} P_1[f_1] \mid \ldots \\ \ldots \mid P_{i-1}[f_{i-1}] \mid P_{ik}[f_i] \mid P_{i+1}[f_{i+1}] \mid \ldots \\ \ldots \mid P_{j-1}[f_{j-1}] \mid P_{jl}[f_j] \mid P_{j+1}[f_{j+1}] \mid \ldots \\ \ldots \mid P_n[f_n] \end{array} \right) \setminus L \right)$$

   где $1 \le i < j \le n, \quad a_{ik}, a_{jl} \neq \tau$, и $f_i(a_{ik}) = \overline{f_j(a_{jl})}$.

Доказательство.
Данная теорема непосредственно следует из

- предыдущей теоремы,
- теоремы 3,
- свойств 6, 9, 10, 16 и 17 из параграфа 3.7, и
- первого утверждения из теоремы 4.

4.6 Распознавание сильной эквивалентности

4.6.1 Отношение $\mu(P_1, P_2)$

Пусть заданы два процесса

$$P_i = (S_i, s_i^0, R_i) \qquad (i = 1, 2)$$

Определим функцию $'$ на множестве всех отношений из S_1 в S_2, которая сопоставляет каждому отношению $\mu \subseteq S_1 \times S_2$ отношение $\mu' \subseteq S_1 \times S_2$, определяемое следующим образом:

$$\mu' \stackrel{\text{def}}{=} \left\{ (s_1, s_2) \in S_1 \times S_2 \;\middle|\; \begin{array}{l} \forall a \in Act \\ \forall s_1' \in S_1 : (s_1 \stackrel{a}{\to} s_1') \in R_1 \\ \quad \exists s_2' \in S_2 : \left\{ \begin{array}{l} (s_2 \stackrel{a}{\to} s_2') \in R_2 \\ (s_1', s_2') \in \mu \end{array} \right. \\ \forall s_2' \in S_2 : (s_2 \stackrel{a}{\to} s_2') \in R_2 \\ \quad \exists s_1' \in S_1 : \left\{ \begin{array}{l} (s_1 \stackrel{a}{\to} s_1') \in R_1 \\ (s_1', s_2') \in \mu \end{array} \right. \end{array} \right\}$$

Нетрудно доказать, что для каждого $\mu \subseteq S_1 \times S_2$

$$\begin{array}{l} \mu \text{ удовлетворяет условиям 1 и 2} \\ \text{из определения БМ} \end{array} \quad \Leftrightarrow \quad \mu \subseteq \mu'$$

В частности,

$$\mu \text{ – БМ между } P_1 \text{ и } P_2 \quad \Leftrightarrow \quad \left\{ \begin{array}{l} (s_1^0, s_2^0) \in \mu \\ \mu \subseteq \mu' \end{array} \right.$$

Нетрудно доказать, что функция $'$ монотонна, т.е. если $\mu_1 \subseteq \mu_2$, то $\mu_1' \subseteq \mu_2'$.

Обозначим символом μ_{max} объединение всех отношений из совокупности

$$\{\mu \subseteq S_1 \times S_2 \mid \mu \subseteq \mu'\} \tag{4.30}$$

Заметим, что отношение μ_{max} принадлежит совокупности (4.30), так как для каждого $\mu \in (4.30)$ из

- включения $\mu \subseteq (\bigcup\limits_{\mu \in (4.30)} \mu) = \mu_{max}$, и
- монотонности функции $'$

следует, что для каждого $\mu \in (4.30)$

$$\mu \subseteq \mu' \subseteq \mu_{max}'$$

Поэтому $\mu_{max} = \bigcup\limits_{\mu \in (4.30)} \mu \subseteq \mu'_{max}$, т.е. $\mu_{max} \in (4.30)$.

Заметим, что имеет место равенство

$$\mu_{max} = \mu'_{max}$$

так как из включения $\mu_{max} \subseteq \mu'_{max}$ и из монотонности функции $'$ следует включение

$$\mu'_{max} \subseteq \mu''_{max}$$

т.е. $\mu'_{max} \in (4.30)$, откуда, в силу максимальности μ_{max}, следует включение

$$\mu'_{max} \subseteq \mu_{max}$$

Таким образом, отношение μ_{max} является

- наибольшим элементом совокупности (4.30), и
- наибольшей неподвижной точкой функции $'$.

Мы будем обозначать это отношение знакосочетанием

$$\mu(P_1, P_2) \tag{4.31}$$

Из теоремы 2 следует, что

$$P_1 \sim P_2 \quad \Leftrightarrow \quad (s_1^0, s_2^0) \in \mu(P_1, P_2)$$

Из определения отношения $\mu(P_1, P_2)$ вытекает, что данное отношение состоит из всех пар $(s_1, s_2) \in S_1 \times S_2$, таких, что

$$P_1(s_1) \sim P_2(s_2)$$

Отношение $\mu(P_1, P_2)$ можно рассматривать как **меру близости** между P_1 и P_2.

4.6.2 Полиномиальный алгоритм распознавания сильной эквивалентности

Пусть P_1 и P_2 – процессы вида

$$P_i = (S_i, s_i^0, R_i) \qquad (i = 1, 2)$$

Если множества S_1 и S_2 конечны, то задача проверки истинности соотношения

$$P_1 \sim P_2 \tag{4.32}$$

очевидно является алгоритмически разрешимой: например, можно перебрать все отношения $\mu \subseteq S_1 \times S_2$ и для каждого из них проверить условия 0, 1 и 2 из определения БМ. Алгоритм заканчивает свою работу, когда

- нашлось хотя бы одно отношение $\mu \subseteq S_1 \times S_2$ которое удовлетворяет условиям 0, 1 и 2 из определения БМ, в этом случае он выдаёт ответ

$$P_1 \sim P_2$$

или

- все отношения $\mu \subseteq S_1 \times S_2$ перебраны, и ни одно из них не удовлетворяет условиям 0, 1 и 2 из определения БМ, в этом случае он выдаёт ответ

$$P_1 \not\sim P_2$$

Если $P_1 \not\sim P_2$, то вышеприведённый алгоритм выдаст ответ после перебора всех отношений между S_1 и S_2, число которых – $2^{|S_1|\cdot|S_2|}$, т.е. данный алгоритм имеет экспоненциальную сложность.

Данную задачу можно решить гораздо более эффективным алгоритмом, который имеет полиномиальную сложность. Для построения такого алгоритма мы рассмотрим следующую последовательность отношений между S_1 и S_2

$$\{\mu_i \mid i \geq 1\} \tag{4.33}$$

где $\mu_1 \stackrel{\text{def}}{=} S_1 \times S_2$, и $\forall\, i \geq 1 \quad \mu_{i+1} \stackrel{\text{def}}{=} \mu_i'$.

Из соотношения $\mu_1 \supseteq \mu_2$ и монотонности функции $'$ следует, что

$$\begin{array}{l} \mu_2 = \mu_1' \supseteq \mu_2' = \mu_3 \\ \mu_3 = \mu_2' \supseteq \mu_3' = \mu_4 \\ \text{и т.д.} \end{array}$$

Таким образом, последовательность (4.33) монотонна:

$$\mu_1 \supseteq \mu_2 \supseteq \ldots$$

Поскольку все члены последовательности (4.33) являются подмножествами конечного множества $S_1 \times S_2$, то данная последовательность не может бесконечно убывать, она стабилизируется на некотором члене, т.е. для некоторого $i \geq 1$ имеет место соотношение

$$\mu_i = \mu_{i+1} = \mu_{i+2} = \ldots$$

Докажем, что член μ_i, на котором наступает стабилизация, совпадает с отношением $\mu(P_1, P_2)$.

- Т.к. $\mu_i = \mu_{i+1} = \mu_i'$, т.е. μ_i – неподвижная точка функции $'$, то

$$\mu_i \subseteq \mu(P_1, P_2) \tag{4.34}$$

поскольку $\mu(P_1, P_2)$ – наибольшая неподвижная точка функции $'$.

- Т.к. для каждого $j \geq 1$ имеет место включение

$$\mu(P_1, P_2) \subseteq \mu_j \tag{4.35}$$

поскольку

  - включение (4.35) верно для $j = 1$, и
  - если включение (4.35) верно для некоторого j, то, в силу монотонности функции $'$, имеем соотношения

$$\mu(P_1, P_2) = \mu(P_1, P_2)' \subseteq \mu_j' = \mu_{j+1}$$

    т.е. включение (4.35) будет верно для $j + 1$

то, в частности, (4.35) верно для $j = i$.

Из (4.34) и (4.35) для $j = i$ следует равенство

$$\mu_i = \mu(P_1, P_2) \tag{4.36}$$

Таким образом, задача проверки истинности соотношения $P_1 \sim P_2$ может быть решена путём

- нахождения первого члена μ_i последовательности (4.33), который удовлетворяет условию $\mu_i = \mu_{i+1}$, и
- проверки для этого μ_i соотношения

$$(s_1^0, s_2^0) \in \mu_i \tag{4.37}$$

Алгоритм выдаёт ответ

$$P_1 \sim P_2$$

тогда и только тогда, когда выполняется (4.37).

Для вычисления членов последовательности (4.33) можно использовать нижеследующий алгоритм, который по отношению $\mu \subseteq S_1 \times S_2$ вычисляет

отношение μ':

$\mu' := \emptyset$
цикл для каждого $(s_1, s_2) \in \mu$
| включить $:= \top$
| **цикл для каждого** $s_1', a:\ s_1 \xrightarrow{a} s_1'$
| | найдено $:= \bot$
| | **цикл для каждого** $s_2':\ s_2 \xrightarrow{a} s_2'$
| | | найдено $:=$ найдено $\vee (s_1', s_2') \in \mu$
| | **конец цикла**
| | включить $:=$ включить $\wedge$ найдено
| **конец цикла**
| **цикл для каждого** $s_2', a:\ s_2 \xrightarrow{a} s_2'$
| | найдено $:= \bot$
| | **цикл для каждого** $s_1':\ s_1 \xrightarrow{a} s_1'$
| | | найдено $:=$ найдено $\vee (s_1', s_2') \in \mu$
| | **конец цикла**
| | включить $:=$ включить $\wedge$ найдено
| **конец цикла**
| **если** включить **то** $\mu' := \mu' \cup \{(s_1, s_2)\}$
конец цикла

Заметим, что данный алгоритм корректен только в том случае, когда $\mu' \subseteq \mu$ (что имеет место в том случае, когда этот алгоритм используется для вычисления членов последовательности (4.33)). В общей ситуации внешний цикл должен иметь вид

цикл для каждого $(s_1, s_2) \in S_1 \times S_2$

Оценим сложность данного алгоритма. Обозначим символом A число

$$A \stackrel{\text{def}}{=} \max(|Act(P_1)|, |Act(P_2)|) + 1$$

- Внешний цикл делает не более $|S_1| \cdot |S_2|$ итераций.
- Оба цикла, содержащиеся в во внешнем цикле, делают не более $|S_1| \cdot |S_2| \cdot A$ итераций.

Поэтому сложность этого алгоритма можно оценить функцией

$$O(|S_1|^2 \cdot |S_2|^2 \cdot A)$$

Поскольку для вычисления того члена последовательности (4.33), на котором наступает её стабилизация, нужно вычислить не больше чем $|S_1| \cdot |S_2|$ членов этой последовательности, то, следовательно, искомое отношение $\mu_i = \mu(P_1, P_2)$ может быть вычислено за время

$$O(|S_1|^3 \cdot |S_2|^3 \cdot A)$$

4.7 Минимизация процессов

4.7.1 Свойства отношений вида $\mu(P, P)$

Теорема 8.

Для каждого процесса $P \stackrel{\text{def}}{=} (S, s^0, R)$ отношение $\mu(P, P)$ является эквивалентностью.

Доказательство.

1. **Рефлексивность** отношения $\mu(P, P)$ следует из того, что диагональное отношение
$$Id_S = \{(s, s) \mid s \in S\}$$
удовлетворяет условиям 1 и 2 из определения БМ, т.е. $Id_S \in (4.30)$.

2. **Симметричность** отношения $\mu(P, P)$ следует из того, что если отношение μ удовлетворяет условиям 1 и 2 из определения БМ, то обратное отношение μ^{-1} тоже удовлетворяет этим условиям, т.е. если $\mu \in (4.30)$, то $\mu^{-1} \in (4.30)$.

3. **Транзитивность** отношения $\mu(P, P)$ следует из того, что произведение
$$\mu(P, P) \circ \mu(P, P)$$
удовлетворяет условиям 1 и 2 из определения БМ, т.е.
$$\mu(P, P) \circ \mu(P, P) \subseteq \mu(P, P) \qquad \blacksquare$$

Обозначим символом $P_\sim$ процесс, компоненты которого имеют следующий вид.

- Множество состояний процесса $P_\sim$ представляет собой совокупность классов эквивалентности, на которые разбивается множество S по отношению $\mu(P, P)$.
- Начальным состоянием является класс $[s^0]$, который содержит начальное состояние s^0 процесса P.
- Множество переходов процесса $P_\sim$ состоит из всех переходов вида
$$[s_1] \xrightarrow{a} [s_2]$$
где $s_1 \xrightarrow{a} s_2$ – произвольный переход из R.

Процесс $P_\sim$ называется **фактор-процессом** процесса P по эквивалентности $\mu(P,P)$.

Теорема 9.
Для каждого процесса P отношение

$$\mu \stackrel{\text{def}}{=} \{\ (s,[s])\ \mid s \in S\}$$

является БМ между P и $P_\sim$.

Доказательство.
Проверим для μ свойства из определения БМ.
Свойство 0 верно по определению начального состояния процесса $P_\sim$. Свойство 1 верно по определению множества переходов процесса $P_\sim$.
Докажем свойство 2. Пусть $P_\sim$ содержит переход

$$[s] \xrightarrow{a} [s']$$

Докажем, что существует переход в R вида

$$s \xrightarrow{a} s''$$

такой, что $(s'',[s']) \in \mu$, т.е. $[s''] = [s']$, т.е.

$$(s'',s') \in \mu(P,P)$$

Из определения множества переходов процесса $P_\sim$ следует, что R содержит переход вида

$$s_1 \xrightarrow{a} s_1' \tag{4.38}$$

где $[s_1] = [s]$ и $[s_1'] = [s']$, т.е.

$$\begin{array}{l}(s_1,s) \in \mu(P,P) \quad \text{и} \\ (s_1',s') \in \mu(P,P)\end{array}$$

Так как $\mu(P,P)$ – БМ, то из

- $(4.38) \in R$, и
- $(s_1,s) \in \mu(P,P)$

следует, что R содержит переход вида

$$s \xrightarrow{a} s_1'' \tag{4.39}$$

где $(s_1'',s_1') \in \mu(P,P)$.

Так как $\mu(P,P)$ транзитивно, то из

$$\begin{array}{l}(s_1'', s_1') \in \mu(P,P) \quad \text{и} \\ (s_1', s') \in \mu(P,P)\end{array}$$

следует

$$(s_1'', s') \in \mu(P,P)$$

Таким образом, в качестве искомого s'' можно взять s_1''. ■

Из теоремы 9 следует, что для каждого процесса P

$$P \sim P_{\sim}$$

4.7.2 Минимальные процессы относительно $\sim$

Процесс P называется **минимальным относительно** $\sim$, если

- каждое его состояние достижимо, и
- $\mu(P,P) = Id_S$
  (где S – множество состояний процесса P).

Ниже минимальные процессы относительно $\sim$ называются просто **минимальными** процессами.

Теорема 10.
Пусть процессы P_1 и P_2 минимальны, и $P_1 \sim P_2$.
Тогда P_1 и P_2 изоморфны.

Доказательство.
Пусть P_i $(i = 1,2)$ имеет вид (S_i, s_i^0, R_i), и пусть $\mu \subseteq S_1 \times S_2$ – БМ между P_1 и P_2.
Поскольку μ^{-1} тоже является БМ, и композиция двух БМ – тоже БМ, то

- $\mu \circ \mu^{-1}$ – БМ между P_1 и P_1
- $\mu^{-1} \circ \mu$ – БМ между P_2 и P_2

откуда, используя определение отношений $\mu(P_i, P_i)$, и определение минимальности процесса, получаем включения

$$\begin{array}{l}\mu \circ \mu^{-1} \subseteq \mu(P_1,P_1) = Id_{S_1} \\ \mu^{-1} \circ \mu \subseteq \mu(P_2,P_2) = Id_{S_2}\end{array} \tag{4.40}$$

Докажем, что отношение μ является функциональным, т.е. для каждого $s \in S_1$ существует единственный элемент $s' \in S_2$, такой, что $(s,s') \in \mu$.

- Если $s = s_1^0$, то полагаем $s' \stackrel{\text{def}}{=} s_2^0$.
- Если $s \neq s_1^0$, то, поскольку каждое состояние в P_1 достижимо, то в P_1 существует путь

$$s_1^0 \xrightarrow{a_1} \ldots \xrightarrow{a_n} s$$

Так как μ - БМ, то в P_2 существует путь

$$s_2^0 \xrightarrow{a_1} \ldots \xrightarrow{a_n} s'$$

причём $(s, s') \in \mu$.

Таким образом, в обоих случаях существует элемент $s' \in S_2$, такой, что $(s, s') \in \mu$.

Докажем единственность элемента s' со свойством $(s, s') \in \mu$. Если для некоторого элемента $s'' \in S_2$ имеет место соотношение $(s, s'') \in \mu$, то $(s'', s) \in \mu^{-1}$, откуда следует

$$(s'', s') \in \mu^{-1} \circ \mu = Id_{S_2}$$

поэтому $s'' = s'$.

По аналогичным соображениям, отношение μ^{-1} тоже является функциональным.

Из условий (4.40) нетрудно вывести биективность функции, которая соответствует отношению μ. По определению БМ, отсюда вытекает изоморфность P_1 и P_2. ■

Теорема 11.
Пусть

- процесс P_2 получается из процесса P_1 удалением недостижимых состояний, и
- $P_3 \stackrel{\text{def}}{=} (P_2)_\sim$.

Тогда процесс P_3 минимален, и

$$P_1 \sim P_2 \sim P_3$$

Доказательство.
Так как каждое состояние в P_2 достижимо, то из определения множества переходов процесса вида $P_\sim$ следует, что каждое состояние P_3 тоже достижимо.

Теперь докажем, что

$$\mu(P_3, P_3) = Id_{S_3} \tag{4.41}$$

т.е. предположим, что $(s', s'') \in \mu(P_3, P_3)$, и докажем, что $s' = s''$.

Из определения процесса вида $P_\sim$ следует, что существуют состояния $s_1, s_2 \in S_2$, такие, что

$$\begin{aligned} s' &= [s_1] \\ s'' &= [s_2] \end{aligned}$$

где $[\cdot]$ обозначает класс эквивалентности по отношению $\mu(P_2, P_2)$.

Из теоремы 9 следует, что

$$\begin{aligned} (s_1, s') &\in \mu(P_2, P_3) \\ (s'', s_2) &\in \mu(P_3, P_2) \end{aligned}$$

Поскольку композиция любых БМ тоже является БМ, то композиция

$$\mu(P_2, P_3) \circ \mu(P_3, P_3) \circ \mu(P_3, P_2) \tag{4.42}$$

является БМ между P_2 и P_2, поэтому

$$(4.42) \subseteq \mu(P_2, P_2) \tag{4.43}$$

Поскольку $(s_1, s_2) \in (4.42)$, то, ввиду (4.43), получаем:

$$s' = [s_1] = [s_2] = s''$$

В заключение отметим, что

- соотношение $P_1 \sim P_2$ тривиально, и
- соотношение $P_2 \sim P_3$ следует из теоремы 9. ■

4.7.3 Алгоритм минимизации процессов

Описанный в параграфе 4.6.2 алгоритм можно использовать также для решения задачи **минимизации конечных процессов**, которая заключается в том, чтобы по заданному конечному процессу P построить процесс с наименьшим числом состояний, который сильно эквивалентен P.

Для построения такого процесса сначала строится процесс P', получаемый из P удалением недостижимых состояний. Искомый процесс имеет вид $P'_\sim$.

Множество состояний процесса P' может быть построено, например, следующим образом. Пусть P имеет вид

$$P = (S, s^0, R)$$

Рассмотрим последовательность подмножеств множества S

$$S_0 \subseteq S_1 \subseteq S_2 \subseteq \ldots \tag{4.44}$$

определяемую следующим образом.

- $S_0 \stackrel{\text{def}}{=} \{s^0\}$
- для каждого $i \geq 0$ множество S_{i+1} получается добавлением к S_i всех состояний $s' \in S$, таких, что

$$\exists s \in S,\ \exists a \in Act :\ (s \xrightarrow{a} s') \in R$$

Поскольку множество S по предположению конечно, то последовательность (4.44) не может неограниченно возрастать. Пусть S_i – тот член последовательности (4.44), на котором эта последовательность стабилизируется. Очевидно, что

- все состояния из S_i достижимы, и
- все состояния из $S \setminus S_i$ недостижимы.

Поэтому множеством состояний процесса P' является множество S_i.

Пусть S' - множество состояний процесса P'. Заметим, что при вычислении отношения $\mu(P', P')$ требуется вычислить не более чем $|S'|$ членов последовательности (4.33). Это верно потому, что в данном случае каждое из отношений в последовательности (4.33) является эквивалентностью (так как если бинарное отношение μ на множестве состояний произвольного процесса является эквивалентностью, то отношение μ' тоже будет эквивалентностью). Поэтому каждый из членов последовательности (4.33) определяет некоторое разбиение множества S', и для каждого $i \geq 1$, если $\mu_{i+1} \neq \mu_i$, то разбиение, соответствующее отношению μ_{i+1}, является измельчением разбиения, соответствующего отношению μ_i. Нетрудно показать, что таких измельчений может быть не больше, чем количество элементов в множестве S'.

Теорема 12.
Процесс $P'_\sim$ имеет наименьшее число состояний среди всех конечных процессов, которые сильно эквивалентны P.

Доказательство.
Пусть P_1 – некоторый конечный процесс, такой, что $P_1 \sim P$. Выделим из P_1 достижимую часть P'_1 и построим процесс $(P'_1)_\sim$. Как было установлено выше,

$$P_1 \sim P'_1 \sim (P'_1)_\sim$$

Кроме того, поскольку $P \sim P' \sim P'_\sim$ и $P \sim P_1$, то, следовательно,

$$P'_\sim \sim (P'_1)_\sim \tag{4.45}$$

Как было доказано в теореме 11, процессы $P'_\sim$ и $(P'_1)_\sim$ минимальны. Отсюда и из (4.45) по теореме 10 следует, что процессы $P'_\sim$ и $(P'_1)_\sim$ изоморфны. В частности, они имеют одинаковое число состояний. Поскольку

- число состояний процесса $(P_1')_\sim$ не превосходит числа состояний процесса P_1', так как состояния процесса $(P_1')_\sim$ являются классами разбиения множества состояний процесса P_1', и
- число состояний процесса P_1' не превосходит числа состояний процесса P_1, так как множество состояний процесса P_1' является подмножеством множества состояний процесса P_1,

то, следовательно, число состояний процесса $P_\sim'$ не превосходит числа состояний процесса P_1. ■

4.8 Наблюдаемая эквивалентность

4.8.1 Определение наблюдаемой эквивалентности

Ещё одним вариантом понятия эквивалентности процессов является **наблюдаемая эквивалентность**. Данное понятие используется в тех ситуациях, когда мы рассматриваем невидимое действие τ как несущественное, и считаем две трассы одинаковыми, если одна может быть получена из другой путём вставок и/или удалений невидимых действий τ.

Для определения понятия наблюдаемой эквивалентности мы введём вспомогательные обозначения.

Пусть P и P' – некоторые процессы.

1. Знакосочетание
$$P \xrightarrow{\tau^*} P' \tag{4.46}$$
означает, что
   - или $P = P'$
   - или существует последовательность процессов
$$P_1, \ldots, P_n \quad (n \geq 2)$$
таких, что
     - $P_1 = P, \quad P_n = P'$
     - для каждого $i = 1, \ldots, n-1$
$$P_i \xrightarrow{\tau} P_{i+1}$$
(4.46) можно интерпретировать как утверждение о том, что процесс P, может незаметным для наблюдателя образом превратиться в процесс P'.

2. Для каждого действия $a \in Act \setminus \{\tau\}$ знакосочетание

$$P \xrightarrow{a_\tau} P' \tag{4.47}$$

означает, что существуют процессы P_1 и P_2 со следующими свойствами:

$$P \xrightarrow{\tau^*} P_1\,, \quad P_1 \xrightarrow{a} P_2\,, \quad P_2 \xrightarrow{\tau^*} P'$$

(4.47) можно интерпретировать как утверждение о том, что процесс P, может

- некоторым образом эволюционировать, так, что внешним проявлением этой эволюции будет лишь исполнение действия a,
- после чего вести себя как процесс P'.

Если имеет место (4.47), то мы будем говорить, что процесс P может **наблюдаемо выполнить** a, и после этого вести себя как P'.

Понятие наблюдаемой эквивалентности основано на следующем понимании эквивалентности процессов: если мы рассматриваем процессы P_1 и P_2 как эквивалентные, то должны быть выполнены следующие условия.

1.
   - Если один из этих процессов P_i может незаметно превратиться в некоторый процесс P_i',
   - то и другой процесс $P_j \quad (j \in \{1,2\} \setminus \{i\})$ тоже должен обладать способностью незаметно превратиться в некоторый процесс P_j', который эквивалентен P_i'.

2.
   - Если один из этих процессов P_i может
     - наблюдаемо выполнить некоторое действие $a \in Act \setminus \{\tau\}$,
     - и после этого вести себя как некоторый процесс P_i'
   - то и другой процесс $P_j \quad (j \in \{1,2\} \setminus \{i\})$ должен обладать способностью
     - наблюдаемо выполнить то же действие a,
     - после чего вести себя как некоторый процесс P_j', который эквивалентен P_i'.

Используя обозначения (4.46) и (4.47), вышеприведённое неформально описанное понятие наблюдаемой эквивалентности можно выразить равносильным образом как некоторое бинарное отношение μ на множестве всех процессов, обладающее следующими свойствами.

(1) Если $(P_1, P_2) \in \mu$, и для некоторого процесса P_1' верно утверждение

$$P_1 \xrightarrow{\tau} P_1' \tag{4.48}$$

то должен существовать процесс P_2', такой, что выполнены условия

$$P_2 \xrightarrow{\tau^*} P_2' \tag{4.49}$$

и

$$(P_1', P_2') \in \mu \tag{4.50}$$

(2) Симметричное свойство: если $(P_1, P_2) \in \mu$, и для некоторого процесса P_2' верно утверждение

$$P_2 \xrightarrow{\tau} P_2' \tag{4.51}$$

то должен существовать процесс P_1', такой, что выполнены условия

$$P_1 \xrightarrow{\tau^*} P_1' \tag{4.52}$$

и (4.50).

(3) Если $(P_1, P_2) \in \mu$, и для некоторого процесса P_1' верно утверждение

$$P_1 \xrightarrow{a} P_1' \tag{4.53}$$

то должен существовать процесс P_2', такой, что выполнены условия

$$P_2 \xrightarrow{a_\tau} P_2' \tag{4.54}$$

и (4.50).

(4) Симметричное свойство: если $(P_1, P_2) \in \mu$, и для некоторого процесса P_2' верно утверждение

$$P_2 \xrightarrow{a} P_2' \tag{4.55}$$

то должен существовать процесс P_1', такой, что выполнены условия

$$P_1 \xrightarrow{a_\tau} P_1' \tag{4.56}$$

и (4.50).

Заметим, что мы не требуем, чтобы μ было отношением эквивалентности.

Обозначим символом $\mathcal{M}_\tau$ совокупность всех бинарных отношений, которые обладают вышеприведёнными свойствами.

Множество $\mathcal{M}_\tau$ непусто: оно содержит, например, диагональное отношение, которое состоит из всех пар вида (P, P), где P – произвольный процесс.

Как и в случае сильной эквивалентности, встаёт естественный вопрос о том, какое же из отношений, входящих в $\mathcal{M}_\tau$, можно использовать для определения понятия наблюдаемой эквивалентности.

Так же, как и в случае сильной эквивалентности, мы предлагаем следующий ответ на этот вопрос: мы будем считать P_1 и P_2 наблюдаемо эквивалентными в том и только в том случае, когда существует хотя бы одно отношение $\mu \in \mathcal{M}_\tau$, которое содержит пару (P_1, P_2), т.е. искомое отношение наблюдаемой эквивалентности на множестве всех процессов мы определяем как объединение всех отношений из $\mathcal{M}_\tau$. Данное отношение обозначается символом $\approx$.

Нетрудно доказать, что

- $\approx \in \mathcal{M}_\tau$,
- $\approx$ является отношением эквивалентности, т.к.
  - рефлексивность $\approx$ следует из того, что диагональное отношение принадлежит $\mathcal{M}_\tau$,
  - симметричность $\approx$ следует из того, что если $\mu \in \mathcal{M}_\tau$, то $\mu^{-1} \in \mathcal{M}_\tau$
  - транзитивность $\approx$ следует из того, что если $\mu_1 \in \mathcal{M}_\tau$ и $\mu_2 \in \mathcal{M}_\tau$, то $\mu_1 \circ \mu_2 \in \mathcal{M}_\tau$.

Если процессы P_1 и P_2 наблюдаемо эквивалентны, то этот факт обозначается знакосочетанием

$$P_1 \approx P_2$$

Нетрудно доказать, что если процессы P_1 и P_2 сильно эквивалентны, то они наблюдаемо эквивалентны.

4.8.2 Логический критерий наблюдаемой эквивалентности

Логический критерий наблюдаемой эквивалентности аналогичен критерию из параграфа 4.4.1. В данном критерии используется то же самое множество формул. Понятие значения формулы на процессе отличается от аналогичного понятия в параграфе 4.4.1 лишь для формул вида $\langle a \rangle \varphi$:

- значение формулы $\langle \tau \rangle \varphi$ на процессе P равно

$$\begin{cases} 1, & \text{если существует процесс } P' : \\ & \quad P \xrightarrow{\tau^*} P',\ P'(\varphi) = 1 \\ 0, & \text{в противном случае} \end{cases}$$

- значение формулы $\langle a \rangle \varphi$ (где $a \neq \tau$) в P равно

$$\begin{cases} 1, & \text{если существует процесс } P' : \\ & \quad P \xrightarrow{a_\tau} P',\ P'(\varphi) = 1 \\ 0, & \text{в противном случае} \end{cases}$$

Для каждого процесса P мы будем обозначать знакосочетанием $Th_\tau(P)$ совокупность всех формул, которые имеют на этом процессе значение 1 (относительно модифицированного определения понятия значения формулы на процессе).

Теорема 13.
Пусть процессы P_1 и P_2 конечны. Тогда

$$P_1 \approx P_2 \quad \Leftrightarrow \quad Th_\tau(P_1) = Th_\tau(P_2) \quad \blacksquare$$

Как и в случае $\sim$, представляет интерес задача нахождения по двум заданным процессам P_1 и P_2 списка формул

$$\varphi_1, \ldots, \varphi_n$$

как можно меньшего размера, таких, что $P_1 \approx P_2$ тогда и только тогда, когда

$$\forall\, i = 1, \ldots, n \qquad P_1(\varphi_i) = P_2(\varphi_i)$$

Используя теорему 13, можно легко доказать, что

$$\text{для каждого процесса } P \qquad P \;\approx\; \tau.P \tag{4.57}$$

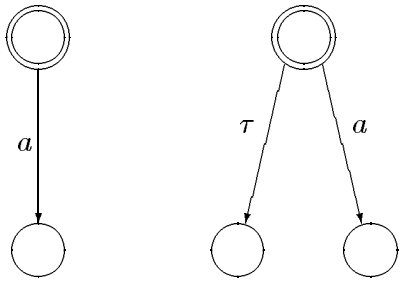
Заметим, что, согласно (4.57), имеет место соотношение

$$\mathbf{0} \;\approx\; \tau.\,\mathbf{0}$$

однако соотношение

$$\mathbf{0} + a.\mathbf{0} \;\approx\; \tau.\,\mathbf{0} + a.\mathbf{0} \quad (\text{где } a \neq \tau) \tag{4.58}$$

неверно, в чём нетрудно убедиться при рассмотрении графового представления левой и правой частей в (4.58):



Формула, которая принимает разные значения на этих процессах, может иметь, например, такой вид:

$$\neg\langle\tau\rangle\neg\langle a\rangle\top$$

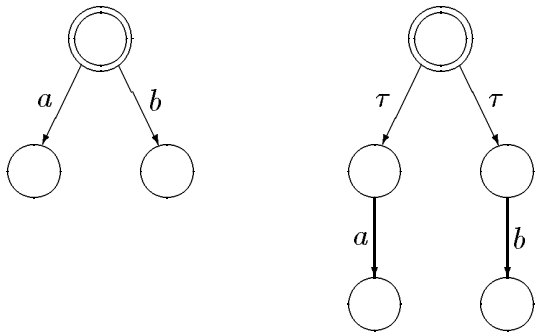
Таким образом, отношение $\approx$ не является конгруэнцией, т.к. оно не сохраняет операцию $+$.

Другой пример: если $a, b \in Act \setminus \{\tau\}$ и $a \neq b$, то

$$a.\mathbf{0} + b.\mathbf{0} \quad \not\approx \quad \tau.a.\mathbf{0} + \tau.b.\mathbf{0}$$

хотя $a.\mathbf{0} \approx \tau.a.\mathbf{0}$ и $b.\mathbf{0} \approx \tau.b.\mathbf{0}$.

Графовое представление этих процессов имеет вид



Отсутствие наблюдаемой эквивалентности между этими процессами обосновывается формулой

$$\langle\tau\rangle\neg\langle a\rangle\top$$

4.8.3 Критерий наблюдаемой эквивалентности, основанный на понятии наблюдаемого БМ

Для отношения $\approx$ также имеет место аналог критерия, основанного на понятии БМ (теорема 2 из параграфа 4.4.2). Для его формулировки мы введём вспомогательные обозначения.

Пусть $P = (S, s^0, R)$ – некоторый процесс, и s_1, s_2 – пара его состояний. Тогда

- знакосочетание

  $$s \xrightarrow{\tau^*} s'$$

  означает, что

  - или $s = s'$,
  - или существует последовательность состояний

    $$s_1, \ldots, s_n \quad (n \geq 2)$$

    такая, что $s_1 = s$, $s_n = s'$, и для каждого $i = 1, \ldots, n-1$

    $$(\, s_i \xrightarrow{\tau} s_{i+1}\,) \in R$$

- знакосочетание

$$s \xrightarrow{a_\tau} s' \quad (\text{где } a \neq \tau)$$

означает, что существуют состояния s_1 и s_2, такие, что

$$s \xrightarrow{\tau^*} s_1\,, \quad s_1 \xrightarrow{a} s_2\,, \quad s_2 \xrightarrow{\tau^*} s'\,.$$

Теорема 14.
Пусть заданы два процесса

$$P_i = (S_i, s_i^0, R_i) \qquad (i = 1, 2)$$

$P_1 \approx P_2$ тогда и только тогда, когда существует отношение $\mu \subseteq S_1 \times S_2$ удовлетворяющее следующим условиям.

0. $(s_1^0, s_2^0) \in \mu$.

1. Для каждой пары $(s_1, s_2) \in \mu$ и каждого перехода из R_1 вида

$$s_1 \xrightarrow{\tau} s_1'$$

существует состояние $s_2' \in S_2$, такое, что

$$s_2 \xrightarrow{\tau^*} s_2'$$

и

$$(s_1', s_2') \in \mu \tag{4.59}$$

2. Для каждой пары $(s_1, s_2) \in \mu$ и каждого перехода из R_2 вида

$$s_2 \xrightarrow{\tau} s_2'$$

существует состояние $s_1' \in S_1$, такое, что

$$s_1 \xrightarrow{\tau^*} s_1'$$

и (4.59).

3. Для каждой пары $(s_1, s_2) \in \mu$ и каждого перехода из R_1 вида

$$s_1 \xrightarrow{a} s_1' \quad (a \neq \tau)$$

существует состояние $s_2' \in S_2$, такое, что

$$s_2 \xrightarrow{a_\tau} s_2'$$

и (4.59).

4. Для каждой пары $(s_1, s_2) \in \mu$ и каждого перехода из R_2 вида

$$s_2 \xrightarrow{a} s_2' \quad (a \neq \tau)$$

существует состояние $s_1' \in S_1$, такое, что

$$s_1 \xrightarrow{a_\tau} s_1'$$

и (4.59).

Отношение μ, удовлетворяющее данным условиям, называется **наблюдаемым БМ (НБМ)** между P_1 и P_2.

4.8.4 Алгебраические свойства наблюдаемой эквивалентности

Теорема 15.

Отношение наблюдаемой эквивалентности сохраняет все операции на процессах, за исключением операции $+$, т.е. если $P_1 \approx P_2$, то

- для каждого $a \in Act \quad a.P_1 \approx a.P_2$
- для каждого процесса $P \quad P_1|P \approx P_2|P$
- для каждого $L \subseteq Names \quad P_1 \setminus L \approx P_2 \setminus L$
- для каждого переименования $f \quad P_1[f] \approx P_2[f]$

Доказательство.

Как было установлено в параграфе 4.8.3, соотношение $P_1 \approx P_2$ эквивалентно тому, что существует НБМ μ между P_1 и P_2. Используя это μ, мы построим НБМ для обоснования каждого из вышеприведённых соотношений.

- Пусть символы $s^0_{(1)}$ и $s^0_{(2)}$ обозначают начальные состояния $a.P_1$ и $a.P_2$ соответственно.

  Тогда отношение

  $$\mu \cup \{(s^0_{(1)}, s^0_{(2)})\}$$

  является НБМ между $a.P_1$ и $a.P_2$.

- Пусть символ S обозначает множество состояний процесса P. Тогда отношение

  $$\{((s_1, s), (s_2, s)) \mid (s_1, s_2) \in \mu,\ q \in S\}$$

  является НБМ между $P_1|P$ и $P_2|P$.

- Отношение μ является НБМ
  - между $P_1 \setminus L$ и $P_2 \setminus L$, и
  - между $P_1[f]$ и $P_2[f]$. ■

4.8.5 Распознавание наблюдаемой эквивалентности и минимизация процессов относительно $\approx$

Для решения задач

1. распознавания для двух заданных конечных процессов, являются ли они наблюдаемо эквивалентными, и

2. построения по заданному конечному процессу P такого процесса P', который имеет наименьшее число состояний среди всех процессов, наблюдаемо эквивалентных P

могут быть построены теория и основанные на ней алгоритмы, которые аналогичны теории и алгоритмам, изложенным в параграфах 4.6 и 4.7. Мы не будем детально излагать эту теорию, т.к. она почти дословно повторяет соответствующую теорию для случая $\sim$. В этой теории для произвольной пары процессов

$$P_i = (S_i, s_i^0, R_i) \qquad (i = 1, 2)$$

тоже определяется функция на отношениях из $S_1 \times S_2$, которая сопоставляет каждому отношению μ некоторое отношение μ'_τ, такое, что

$$\begin{array}{l}\mu \text{ удовлетворяет условиям 1, 2, 3, 4} \\ \text{из определения НБМ}\end{array} \quad \Leftrightarrow \quad \mu \subseteq \mu'_\tau$$

В частности,

$$\mu \text{ – НБМ между } P_1 \text{ и } P_2 \quad \Leftrightarrow \quad \left\{ \begin{array}{l} (s_1^0, s_2^0) \in \mu \\ \mu \subseteq \mu'_\tau \end{array} \right.$$

Обозначим символом $\mu_\tau(P_1, P_2)$ объединение всех отношений из совокупности

$$\{\mu \subseteq S_1 \times S_2 \mid \mu \subseteq \mu'_\tau\} \tag{4.60}$$

Данное отношение является наибольшим элементом совокупности (4.60), и обладает свойством

$$P_1 \approx P_2 \quad \Leftrightarrow \quad (s_1^0, s_2^0) \in \mu_\tau(P_1, P_2)$$

Из определения отношения $\mu_\tau(P_1, P_2)$ вытекает, что оно состоит из всех пар $(s_1, s_2) \in S_1 \times S_2$, таких, что

$$P_1(s_1) \approx P_2(s_2)$$

Отношение $\mu_\tau(P_1, P_2)$ можно рассматривать как ещё одну меру близости между P_1 и P_2.

При построении полиномиального алгоритма вычисления $\mu_\tau(P_1, P_2)$, аналогичного алгоритму из параграфа 4.6.2, следует учитывать следующее соображение. Всякий раз, когда для заданной пары s, s' состояний некоторого процесса P требуется проверить условие

$$s \xrightarrow{\tau^*} s'$$

достаточно анализировать последовательности переходов вида

$$s \xrightarrow{\tau} s_1 \xrightarrow{\tau} s_2 \xrightarrow{\tau} \ldots$$

длина которых не превосходит числа состояний процесса P.

4.8.6 Другие критерии эквивалентности процессов

Доказать сильную или наблюдаемую эквивалентность процессов P_1 и P_2 можно также с помощью излагаемых ниже критериев. В некоторых случаях использование этих критериев гораздо проще всех других способов доказательства соответствующей эквивалентности P_1 и P_2.

Бинарное отношение μ на множестве процессов называется

- БМ (mod $\sim$), если $\mu \subseteq (\sim \mu \sim)'$
- НБМ (mod $\sim$), если $\mu \subseteq (\sim \mu \sim)'_\tau$
- НБМ (mod $\approx$), если $\mu \subseteq (\approx \mu \approx)'_\tau$

Нетрудно доказать, что

- если μ – БМ (mod $\sim$), то $\mu \subseteq \sim$, и
- если μ – НБМ (mod $\sim$ или mod $\approx$), то $\mu \subseteq \approx$.

Таким образом, для доказательства $P_1 \sim P_2$ или $P_1 \approx P_2$ достаточно найти подходящее

- БМ (mod $\sim$), или
- НБМ (mod $\sim$ или mod $\approx$)

соответственно, такое, что

$$(P_1, P_2) \in \mu$$

4.9 Наблюдаемая конгруэнция

4.9.1 Мотивировка понятия наблюдаемой конгруэнции

Понятие эквивалентности процессов может быть определено неоднозначно. В предыдущих параграфах уже были рассмотрены различные виды эквивалентности процессов, каждый из которых отражал определённую точку зрения на то, какие виды поведения следует считать одинаковыми. В добавление к этим понятиям эквивалентности процессов, можно ещё определить, например, такие эквивалентности, которые

- учитывают длительность исполнения действий, т.е., в частности, одно из условий эквивалентности процессов P_1 и P_2 может иметь следующий вид:
  - если один из этих процессов P_i может в течение некоторого промежутка времени незаметно превратиться в процесс P_i',
  - то и другой процесс $P_j \quad (j \in \{1,2\} \setminus \{i\})$ тоже должен обладать способностью в течение примерно такого же промежутка времени незаметно превратиться в процесс P_j', который эквивалентен P_i' (понятие "примерно такого же промежутка времени" может уточняться различным образом)
- или учитывают свойство **справедливости (fairness)** в поведении процессов, т.е. не позволяют рассматривать как эквивалентные такие два процесса,
  - один из которых обладает свойством справедливости,
  - а другой - не обладает

  где одно из определений свойства справедливости имеет следующий вид: процесс называется **справедливым**, если не существует бесконечной последовательности переходов этого процесса вида

$$s_0 \xrightarrow{\tau} s_1 \xrightarrow{\tau} s_2 \xrightarrow{\tau} \dots$$

  такой, что состояние s_0 достижимо, и для каждого $i \geq 0$

$$Act(s_i) \setminus \{\tau\} \neq \emptyset$$

  отметим, что отношение наблюдаемой эквивалентности не учитывает свойство справедливости: существуют два процесса P_1 и P_2, такие, что $P_1 \approx P_2$, но P_1 обладает свойством справедливости, а P_2 не обладает этим свойством, например

– в качестве P_1 можно взять процесс $a.\mathbf{0}$, где $a \neq \tau$,

– а в качестве P_2 – процесс $a.\mathbf{0} \mid \tau^*$, где процесс τ^* имеет одно состояние и один переход с меткой τ

- и т.д.

Решение о том, какое именно из понятий эквивалентности между процессами следует выбрать в конкретной ситуации, существенно зависит от целей, для достижения которых предназначено данное понятие.

В настоящем параграфе мы определяем ещё один вид эквивалентности процессов, называемый **наблюдаемой конгруэнцией**, Данная эквивалентность обозначается символом $\overset{+}{\approx}$. Мы определяем эту эквивалентность, исходя из следующих условий, которым она должна удовлетворять.

1. Процессы, находящиеся в отношении $\overset{+}{\approx}$, должны быть наблюдаемо эквивалентными.

2. Пусть

   - процесс P построен в виде композиции процессов

     $$P_1, \ldots, P_n$$

     в которой используются операции

     $$a., \quad +, \quad |, \quad \backslash L, \quad [f] \tag{4.61}$$

   - и мы решили заменить одну из частей этой композиции (например, P_i), на другой процесс P_i', который мы считаем

     – эквивалентным компоненте P_i, но

     – более предпочтительным для нас по некоторым причинам, чем P_i (например, P_i' имеет меньшую сложность, чем P_i).

     Мы хотели бы, чтобы процесс, получаемый из P в результате такой замены, был бы эквивалентен исходному процессу P.

Нетрудно доказать, что эквивалентность μ на множестве процессов удовлетворяет сформулированным выше условиям тогда и только тогда, когда

$$\begin{cases} \mu \subseteq \approx \\ \mu \text{ является конгруэнцией} \\ \qquad \text{относительно операций (4.61)} \end{cases} \tag{4.62}$$

Условия (4.62) определяют искомую эквивалентность неоднозначно. Например, данным условиям удовлетворяют

- тождественное отношение (состоящее из пар вида (P, P)), и
- сильная эквивалентность $(\sim)$.

Ниже мы докажем, что среди всех эквивалентностей, удовлетворяющих условиям (4.62), существует наибольшая (относительно включения). Вполне естественно считать именно эту эквивалентность искомой эквивалентностью.

4.9.2 Определение понятия наблюдаемой конгруэнции

Для определения понятия наблюдаемой конгруэнции мы введём вспомогательное обозначение.

Пусть P и P' – некоторые процессы. Знакосочетание

$$P \xrightarrow{\tau^+} P'$$

означает, что существует последовательность процессов

$$P_1, \ldots, P_n \quad (n \geq 2)$$

таких, что

- $P_1 = P, \quad P_n = P'$
- для каждого $i = 1, \ldots, n-1$

$$P_i \xrightarrow{\tau} P_{i+1}$$

Мы будем говорить, что процессы P_1 и P_2 находятся в отношении **наблюдаемой конгруэнции**, и обозначать этот факт знакосочетанием

$$P_1 \overset{+}{\approx} P_2$$

если выполнены следующие условия.

(0) $P_1 \approx P_2$.

(1) Если для некоторого процесса P_1' верно утверждение

$$P_1 \xrightarrow{\tau} P_1' \tag{4.63}$$

то существует процесс P_2', такой, что

$$P_2 \xrightarrow{\tau^+} P_2' \tag{4.64}$$

и

$$P_1' \approx P_2' \tag{4.65}$$

(2) Симметричное условие: если для некоторого процесса P_2' верно утверждение

$$P_2 \xrightarrow{\tau} P_2' \tag{4.66}$$

то существует процесс P_1', такой, что

$$P_1 \xrightarrow{\tau^+} P_1' \tag{4.67}$$

и (4.65).

Нетрудно доказать, что наблюдаемая конгруэнция является отношением эквивалентности.

4.9.3 Логический критерий наблюдаемой конгруэнтности

Логический критерий наблюдаемой конгруэнтности двух процессов получается небольшой модификацией логического критерия наблюдаемой эквивалентности, из параграфа 4.8.2.

Множество формул Fm^+, используемых в данном критерии, является расширением множества формул Fm из параграфа 4.4.2 путём использования дополнительной модальной связки $\langle\tau^+\rangle$:

- каждая формула из Fm принадлежит Fm^+, и
- для каждой формулы $\varphi \in Fm$ знакосочетание

  $$\langle\tau^+\rangle\varphi$$

  является формулой из множества Fm^+.

Для каждой формулы $\varphi \in Fm^+$ её значение на процессе P определяется следующим образом.

- Если $\varphi \in Fm$, то её значение в P определяется так же, как в параграфе 4.8.2.
- Если $\varphi = \langle\tau^+\rangle\psi$, где $\psi \in Fm$, то

$$P(\varphi) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{если существует процесс } P': \\ & P \xrightarrow{\tau^+} P',\ P'(\psi) = 1 \\ 0, & \text{в противном случае} \end{cases}$$

Для каждого процесса P мы будем обозначать знакосочетанием $Th_\tau^+(P)$ совокупность всех формул из Fm^+, которые имеют на этом процессе значение 1.

Теорема 16.
Пусть процессы P_1 и P_2 конечны. Тогда

$$P_1 \overset{+}{\approx} P_2 \quad \Leftrightarrow \quad Th^+_\tau(P_1) = Th^+_\tau(P_2) \quad \blacksquare$$

Как и в случаях $\sim$ и $\approx$, представляет интерес задача нахождения по двум заданным процессам P_1 и P_2 списка формул

$$\varphi_1, \ldots, \varphi_n \ \in Fm^+$$

как можно меньшего размера, таких, что $P_1 \overset{+}{\approx} P_2$ тогда и только тогда, когда

$$\forall\, i = 1, \ldots, n \qquad P_1(\varphi_i) = P_2(\varphi_i)$$

4.9.4 Критерий наблюдаемой конгруэнтности, основанный на понятии НБМ

Введём вспомогательное обозначение. Пусть

- P – процесс вида (S, s^0, R), и
- s_1, s_2 – пара состояний из S.

Тогда знакосочетание

$$s \xrightarrow{\ \tau^+\ } s'$$

означает, что существует последовательность состояний

$$s_1, \ldots, s_n \quad (n \geq 2)$$

такая, что $s_1 = s, \ \ s_n = s'$, и для каждого $i = 1, \ldots, n-1$

$$(\, s_i \xrightarrow{\ \tau\ } s_{i+1}\,) \ \in R$$

Теорема 17.
Пусть заданы два процесса

$$P_i = (S_i, s_i^0, R_i) \qquad (i = 1, 2)$$

$P_1 \overset{+}{\approx} P_2$ тогда и только тогда, когда существует отношение $\mu \subseteq S_1 \times S_2$ удовлетворяющее следующим условиям.

0. μ – НБМ между P_1 и P_2
   (понятие НБМ изложено в параграфе 4.8.3).

1. Для каждого перехода из R_1 вида

$$s_1^0 \xrightarrow{\tau} s_1'$$

существует состояние $s_2' \in S_2$, такое, что

$$s_2^0 \xrightarrow{\tau^+} s_2'$$

и

$$(s_1', s_2') \in \mu \tag{4.68}$$

2. Для каждого перехода из R_2 вида

$$s_2^0 \xrightarrow{\tau} s_2'$$

существует состояние $s_1' \in S_1$, такое, что

$$s_1^0 \xrightarrow{\tau^+} s_1'$$

и (4.68). ■

Ниже знакосочетание НБМ$^+$ является сокращённой записью фразы

"НБМ, удовлетворяющее условиям 1 и 2 теоремы 17".

4.9.5 Алгебраические свойства наблюдаемой конгруэнции

Теорема 18.

Наблюдаемая конгруэнция действительно является конгруэнцией, т.е. если $P_1 \overset{+}{\approx} P_2$, то

- для каждого $a \in Act \quad a.P_1 \overset{+}{\approx} a.P_2$
- для каждого процесса $P \quad P_1 + P \overset{+}{\approx} P_2 + P$
- для каждого процесса $P \quad P_1|P \overset{+}{\approx} P_2|P$
- для каждого $L \subseteq Names \quad P_1 \setminus L \overset{+}{\approx} P_2 \setminus L$
- для каждого переименования $f \quad P_1[f] \overset{+}{\approx} P_2[f]$

Доказательство.

Как было установлено в параграфе 4.9.4, соотношение $P_1 \overset{+}{\approx} P_2$ эквивалентно тому, что существует НБМ$^+$ μ между P_1 и P_2. Используя это μ, для обоснования каждого из вышеприведённых соотношений мы построим соответствующее НБМ$^+$.

- Пусть символы $s^0_{(1)}$ и $s^0_{(2)}$ обозначают начальные состояния $a.P_1$ и $a.P_2$ соответственно.

  Тогда отношение
  $$\{(s^0_{(1)}, s^0_{(2)})\} \cup \mu$$
  является НБМ$^+$ между $a.P_1$ и $a.P_2$

- Пусть
  - символы $s^0_{(1)}$ и $s^0_{(2)}$ обозначают начальные состояния P_1+P и P_2+P соответственно, и
  - символ S обозначает множество состояний процесса P.

  Тогда отношение
  $$\{(s^0_{(1)}, s^0_{(2)})\} \cup \mu \cup Id_S$$
  является НБМ$^+$ между $P_1 + P$ и $P_2 + P$

- Пусть символ S обозначает множество состояний процесса P. Тогда отношение
  $$\{((s_1, s), (s_2, s)) \mid (s_1, s_2) \in \mu,\ q \in S\}$$
  является НБМ$^+$ между $P_1|P$ и $P_2|P$

- Отношение μ является НБМ$^+$
  - между $P_1 \setminus L$ и $P_2 \setminus L$, и
  - между $P_1[f]$ и $P_2[f]$. ■

Теорема 19.
Для любых процессов P_1 и P_2
$$P_1 \approx P_2 \quad \Leftrightarrow \quad \begin{cases} P_1 \overset{+}{\approx} P_2 & \text{или} \\ P_1 \overset{+}{\approx} \tau.P_2 & \text{или} \\ \tau.P_1 \overset{+}{\approx} P_2 \end{cases}$$

Доказательство.
Импликация "$\Leftarrow$" следует из включения $\overset{+}{\approx} \subseteq \approx$, и того, что
$$\text{для любого процесса } P \quad P \approx \tau.P \tag{4.69}$$

Докажем импликацию "$\Rightarrow$". Предположим, что
$$P_1 \approx P_2 \tag{4.70}$$
и
$$\text{неверно, что } P_1 \overset{+}{\approx} P_2 \tag{4.71}$$

(4.71) может иметь место, например, в следующем случае:

$$\begin{array}{l}\text{существует процесс } P_1', \text{ такой, что} \\ P_1 \xrightarrow{\tau} P_1'\end{array} \tag{4.72}$$

и

$$\begin{array}{l}\text{не существует процесса } P_2' \approx P_1', \\ \text{такого, что } P_2 \xrightarrow{\tau^+} P_2'\end{array} \tag{4.73}$$

Докажем, что в этом случае имеет место соотношение

$$P_1 \overset{+}{\approx} \tau.P_2$$

Согласно определению наблюдаемой конгруэнции, надо доказать, что выполнены следующие условия.

(0) $P_1 \approx \tau.P_2$.

Это условие следует из (4.70) и (4.69).

(1) Докажем, что если для некоторого процесса P_1' верно утверждение

$$P_1 \xrightarrow{\tau} P_1' \tag{4.74}$$

то для некоторого процесса $P_2' \approx P_1'$ верно утверждение

$$\tau.P_2 \xrightarrow{\tau^+} P_2' \tag{4.75}$$

Из (4.70), (4.74), и из определения наблюдаемой эквивалентности следует, что для некоторого процесса $P_2' \approx P_1'$ верно утверждение

$$P_2 \xrightarrow{\tau^*} P_2' \tag{4.76}$$

(4.75) следует из $\tau.P_2 \xrightarrow{\tau} P_2$ и (4.76).

(2) Докажем, что если для некоторого процесса P_2' верно утверждение

$$\tau.P_2 \xrightarrow{\tau} P_2' \tag{4.77}$$

то для некоторого процесса $P_1' \approx P_2'$ верно утверждение

$$P_1 \xrightarrow{\tau^+} P_1'$$

Из определения операции префиксного действия и из (4.77) следует, что

$$P_2' = P_2$$

Таким образом, надо доказать, что

$$\begin{array}{l}\text{для некоторого процесса } P_1' \approx P_2 \\ \text{верно утверждение } P_1 \xrightarrow{\tau^+} P_1'\end{array} \tag{4.78}$$

Пусть P_1' есть в точности тот процесс, который упоминается в предположении (4.72). Из предположения (4.70) следует, что

$$\begin{array}{l}\text{существует процесс } P_2' \approx P_1', \\ \text{такой, что } P_2 \xrightarrow{\tau^*} P_2'\end{array} \tag{4.79}$$

Сопоставляя (4.79) и (4.73), получаем, $P_2' = P_2$, т.е. мы доказали (4.78).

Другая причина, по которой может иметь место (4.71) заключается в том, что

- существует процесс P_2', такой, что $P_2 \xrightarrow{\tau} P_2'$, и
- не существует процесса $P_1' \approx P_2'$, такого, что

$$P_1 \xrightarrow{\tau^+} P_1'$$

В этом случае аналогичными рассуждениями можно доказать, что имеет место соотношение

$$\tau.P_1 \overset{+}{\approx} P_2 \quad \blacksquare$$

Теорема 20.
Отношение $\overset{+}{\approx}$ совпадает с отношением

$$\{(P_1, P_2) \mid \forall\, P \quad P_1 + P \approx P_2 + P\} \tag{4.80}$$

Доказательство.
Включение $\overset{+}{\approx} \subseteq$ (4.80) следует из того, что

- $\overset{+}{\approx}$ – конгруэнция (т.е., в частности, $\overset{+}{\approx}$ сохраняет операцию $+$), и
- $\overset{+}{\approx} \subseteq \approx$.

Докажем включение (4.80) $\subseteq \overset{+}{\approx}$.
Пусть $(P_1, P_2) \in$ (4.80).
Поскольку для каждого процесса P имеет место соотношение

$$P_1 + P \approx P_2 + P \tag{4.81}$$

то, полагая в (4.81) $P \overset{\text{def}}{=} \mathbf{0}$, имеем:

$$P_1 + \mathbf{0} \approx P_2 + \mathbf{0} \tag{4.82}$$

Поскольку

- для любого процесса P имеет место свойство

$$P + \mathbf{0} \sim P$$

- и, кроме того, $\sim \subseteq \approx$

то из (4.82) следует, что

$$P_1 \approx P_2 \tag{4.83}$$

Если неверно, что $P_1 \overset{+}{\approx} P_2$, то из (4.83) по теореме 19 следует, что

- либо $P_1 \overset{+}{\approx} \tau.P_2$
- либо $\tau.P_1 \overset{+}{\approx} P_2$

Рассмотрим, например, случай

$$P_1 \overset{+}{\approx} \tau.P_2 \tag{4.84}$$

(другой случай разбирается аналогично).

Так как $\overset{+}{\approx}$ – конгруэнция, то из (4.84) следует, что для любого процесса P

$$P_1 + P \overset{+}{\approx} \tau.P_2 + P \tag{4.85}$$

Из (4.81), (4.85) и включения $\overset{+}{\approx} \subseteq \approx$ следует, что для любого процесса P

$$P_2 + P \approx \tau.P_2 + P \tag{4.86}$$

Докажем, что имеет место соотношение

$$P_2 \overset{+}{\approx} \tau.P_2 \tag{4.87}$$

(4.87) равносильно существованию процесса $P_2' \approx P_2$, такой, что

$$P_2 \xrightarrow{\tau^+} P_2' \tag{4.88}$$

Выберем произвольное действие $b \in Act \setminus \{\tau\}$, которое не входит в P_2 (здесь мы используем предположение из параграфа 2.3 о том, что множество $Names$, а значит, и множество Act, является бесконечным).

Соотношение (4.86) должно быть верно в случае, когда P имеет вид $b.\mathbf{0}$, т.е. должно быть верно соотношение

$$P_2 + b.\mathbf{0} \approx \tau.P_2 + b.\mathbf{0} \tag{4.89}$$

Так как имеет место соотношение

$$\tau.P_2 + b.\mathbf{0} \xrightarrow{\tau} P_2$$

то из (4.89) по определению отношения $\approx$ следует, что для некоторого процесса $P_2' \approx P_2$ имеет место соотношение

$$P_2 + b.\mathbf{0} \xrightarrow{\tau^*} P_2' \tag{4.90}$$

Случай $P_2 + b.\mathbf{0} = P_2'$ невозможен, так как процесс в левой части этого равенства содержит действие, которое не содержит процесс в правой части этого равенства. Согласно (4.90), отсюда следует соотношение

$$P_2 + b.\mathbf{0} \xrightarrow{\tau^+} P_2' \tag{4.91}$$

Из определения операции + следует, что (4.91) возможно в том и только в том случае, когда имеет место (4.88).

Таким образом, мы доказали, что для некоторого процесса $P_2' \approx P_2$ имеет место (4.88), т.е. мы доказали (4.87).

Из (4.84) и (4.87) следует, что $P_1 \overset{+}{\approx} P_2$. ■

Теорема 21.

$\overset{+}{\approx}$ является наибольшей конгруэнцией, содержащейся в $\approx$, т.е. для каждой конгруэнции ν на множестве всех процессов имеет место импликация:

$$\nu \subseteq \approx \quad \Rightarrow \quad \nu \subseteq \overset{+}{\approx}$$

Доказательство.

Докажем, что если $(P_1, P_2) \in \nu$, то $P_1 \overset{+}{\approx} P_2$.

Пусть $(P_1, P_2) \in \nu$. Так как ν – конгруэнция, то

$$\text{для каждого процесса } P \quad (P_1 + P, P_2 + P) \in \nu \tag{4.92}$$

Если $\nu \subseteq \approx$, то из (4.92) следует, что

$$\text{для каждого процесса } P \quad P_1 + P \approx P_2 + P \tag{4.93}$$

Согласно теореме 20, из (4.93) следует, что $P_1 \overset{+}{\approx} P_2$. ■

Теорема 22.

Для отношений $\sim$, $\approx$ и $\overset{+}{\approx}$ верны включения

$$\sim \subseteq \overset{+}{\approx} \subseteq \approx \tag{4.94}$$

Доказательство.

Включение $\overset{+}{\approx} \subseteq \approx$ верно по определению $\overset{+}{\approx}$.

Включение $\sim \subseteq \overset{+}{\approx}$ следует

- из включения $\sim \subseteq \approx$, и
- из того, что если $P_1 \sim P_2$, то данная пара удовлетворяет условиям, изложенным в определении отношения $\overset{+}{\approx}$. ■

Отметим, что оба включения в (4.94) – собственные:

- $a.\tau.\mathbf{0} \not\sim a.\mathbf{0}$, но $a.\tau.\mathbf{0} \stackrel{+}{\approx} a.\mathbf{0}$
- $\tau.\mathbf{0} \not\stackrel{+}{\approx} \mathbf{0}$, но $\tau.\mathbf{0} \approx \mathbf{0}$

Теорема 23.

1. Если $P_1 \approx P_2$, то для каждого $a \in Act$

$$a.P_1 \stackrel{+}{\approx} a.P_2$$

   В частности, для каждого процесса P

$$a.\tau.P \stackrel{+}{\approx} a.P \tag{4.95}$$

2. Для любого процесса P

$$P + \tau.P \stackrel{+}{\approx} \tau.P \tag{4.96}$$

3. Для любых процессов P_1 и P_2 и любого $a \in Act$

$$a.(P_1 + \tau.P_2) + a.P_2 \stackrel{+}{\approx} a.(P_1 + \tau.P_2) \tag{4.97}$$

4. Для любых процессов P_1 и P_2

$$P_1 + \tau.(P_1 + P_2) \stackrel{+}{\approx} \tau.(P_1 + P_2) \tag{4.98}$$

Доказательство.
Для каждого из доказываемых соотношений мы построим НМБ$^+$ между его левой и правой частями.

1. Как было установлено в теореме 14 из параграфа 4.8.3, соотношение $P_1 \approx P_2$ эквивалентно тому, что существует НБМ μ между P_1 и P_2.

   Пусть символы $s^0_{(1)}$ и $s^0_{(2)}$ обозначают начальные состояния $a.P_1$ и $a.P_2$ соответственно.

   Тогда отношение

$$\{(s^0_{(1)}, s^0_{(2)})\} \cup \mu$$

   является НБМ$^+$ между $a.P_1$ и $a.P_2$.

   (4.95) следует

   - из вышедоказанного утверждения, и
   - из соотношения $\tau.P \approx P$, которое верно согласно (4.57).

2. Пусть P имеет вид

$$P = (S, s^0, R)$$

Обозначим символами $S_{(1)}$ и $S_{(2)}$ дубликаты множества S в процессах P и $\tau.P$ соответственно, входящих в левую часть соотношения (4.96). Элементы этих дубликатов мы будем обозначать символами $s_{(1)}$ и $s_{(2)}$ соответственно, где s – произвольный элемент множества S.

Пусть символы s_l^0 и s_r^0 обозначают начальные состояния процессов в левой и правой частях соотношения (4.96) соответственно. Тогда отношение

$$\{(s_l^0, s_r^0)\} \cup \{(s_{(i)}, s) \mid s \in S,\ i = 1, 2\}$$

является НБМ$^+$ между левой и правой частями соотношения (4.96).

3. Пусть $P_i = (S_i, s_i^0, R_i) \quad (i = 1, 2)$. Мы можем считать, что $S_1 \cap S_2 = \emptyset$. Обозначим

   - символом s_τ^0 начальное состояние процесса

$$P_1 + \tau.P_2 \tag{4.99}$$

   - символом s^0 – начальное состояние процесса

$$a.(P_1 + \tau.P_2) \tag{4.100}$$

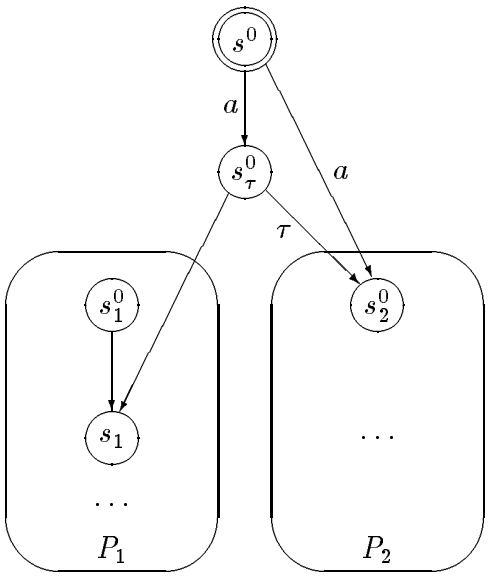
Заметим, что (4.100) совпадает с правой частью (4.97).

Левая часть (4.97) сильно эквивалентна процессу P', который получается из (4.100) добавлением перехода

$$s^0 \xrightarrow{a} s_2^0$$

это легко увидеть при рассмотрении графового представления процесса

P', которое имеет вид



Нетрудно доказать, что процесс P' наблюдаемо конгруэнтен процессу (4.100). Множества состояний этих процессов можно рассматривать как дубликаты $S_{(1)}$ и $S_{(2)}$ одного и того же множества S, и НБМ$^+$ между P' и (4.100) имеет вид

$$\{(s_{(1)}, s_{(2)}) \mid s \in S\} \tag{4.101}$$

Поскольку

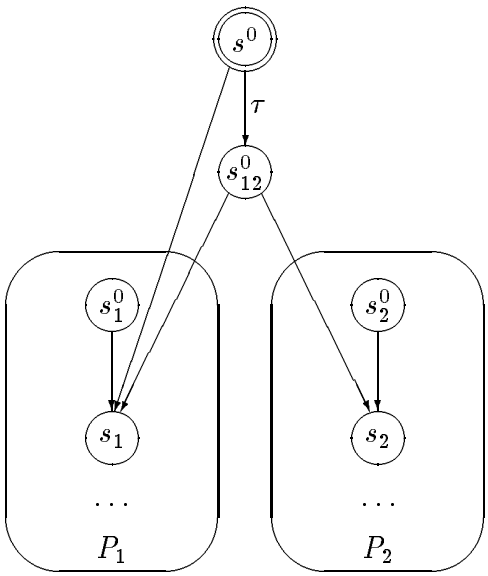
- по теореме 22, имеет место включение $\sim \subseteq \overset{+}{\approx}$, и
- (4.100) совпадает с правой частью (4.97),

то мы доказали наблюдаемую конгруэнтность левой и правой частей соотношения (4.97). ■

4. Рассуждения в данном случае аналогичны рассуждениям в предыдущем случае. Мы не будем излагать их детально, отметим лишь, что

   - левая часть соотношения (4.98) находится в отношении сильной эквивалентности с процессом P', графовое представление которой

имеет вид



где

- s^0_1 и s^0_2 – начальные состояния процессов P_1 и P_2,
- s^0_{12} – начальное состояние процесса $P_1 + P_2$

- правая часть соотношения (4.98), которую мы обозначим символом P'', получается из P' удалением переходов вида

$$s^0 \longrightarrow s_1$$

Нетрудно доказать, что $P' \overset{+}{\approx} P''$. Множества состояний этих процессов можно рассматривать как дубликаты $S_{(1)}$ и $S_{(2)}$ одного и того же множества S, и НБМ$^+$ между P' и P'' имеет вид (4.101). ■

4.9.6 Распознавание наблюдаемой конгруэнтности

Для решения задачи распознавания для двух заданных конечных процессов, являются ли они наблюдаемо конгруэнтными, можно использовать следующую теорему.

Теорема 24.
Пусть P_1 и P_2 - конечные процессы. Соотношение

$$P_1 \overset{+}{\approx} P_2$$

имеет место тогда и только тогда, когда

$$\begin{cases} (s^0_1, s^0_2) \in \mu_\tau(P_1, P_2) \\ \mu_\tau(P_1, P_2) \ - \text{ НБМ}^+ \end{cases} \quad \blacksquare$$

4.9.7 Минимизация процессов относительно наблюдаемой конгруэнции

Для решения задачи минимизации конечных процессов относительно наблюдаемой конгруэнции можно использовать следующие теоремы.

Теорема 25.

Пусть $P = (S, s^0, R)$ - произвольный процесс. Обозначим символом $P_{\approx}$ **фактор-процесс** процесса P по эквивалентности $\mu_\tau(P, P)$, т.е. процесс, компоненты которого имеют следующий вид.

- Множество состояний процесса $P_{\approx}$ представляет собой совокупность классов эквивалентности множества S по отношению $\mu_\tau(P, P)$.
- Начальным состоянием является класс $[s^0]$.
- Переходы процесса $P_{\approx}$ имеют вид

$$[s_1] \xrightarrow{a} [s_2]$$

где $s_1 \xrightarrow{a} s_2$ – произвольный переход из R.

Тогда $P \overset{+}{\approx} (P_{\approx})$. ■

Теорема 26.

Пусть процесс P' получается из процесса P путём удаления недостижимых состояний. Тогда $P'_{\approx}$ имеет наименьшее число состояний среди всех процессов, которые наблюдаемо конгруэнтны P. ■

Глава 5

Рекурсивные определения процессов

В некоторых случаях процесс удобнее задавать не явным описанием множеств его состояний и переходов, а при помощи рекурсивного определения.

5.1 Процессные выражения

Для того, чтобы сформулировать понятие рекурсивного определения процессов, мы введём понятие **процессного выражения**.

Множество $PExpr$ **процессных выражений (ПВ)** определяется индуктивно, т.е.

- указываются элементарные ПВ, и
- описываются правила построения новых ПВ из уже имеющихся.

Каждое из правил построения ПВ имеет своё название, которое указывается жирным шрифтом перед описанием этого правила.

процессные константы:

Мы будем предполагать, что задано счётное множество **процессных констант**, причём каждой процессной константе сопоставлен некоторый процесс, называемый **значением** этой константы.

Существует процессная константа, значением которой является пустой процесс **0**, эта константа обозначается тем же символом **0**.

Каждая процессная константа является ПВ.

процессные имена:

Мы будем предполагать, что задано счётное множество **процессных имён**.

Каждое процессное имя является ПВ.

префиксное действие:
Для каждого $a \in Act$ и каждого ПВ P знакосочетание $a.P$ является ПВ.

выбор:
Для любых ПВ P_1, P_2 знакосочетание $P_1 + P_2$ является ПВ.

параллельная композиция:
Для любых ПВ P_1, P_2 знакосочетание $P_1 \mid P_2$ является ПВ.

ограничение:
Для каждого подмножества $L \subseteq Names$ и каждого ПВ P знакосочетание $P \setminus L$ является ПВ.

переименование:
Для каждого переименования f и каждого ПВ P знакосочетание $P[f]$ является ПВ.

5.2 Понятие рекурсивного определения процессов

Рекурсивным определением (РО) процессов называется список формальных равенств вида

$$\begin{cases} A_1 = P_1 \\ \dots \\ A_n = P_n \end{cases} \tag{5.1}$$

где

- $A_1, \dots, A_n$ – различные процессные имена, и
- $P_1, \dots, P_n$ – ПВ, удовлетворяющие следующему условию: для каждого $i = 1, \dots, n$ каждое процессное имя, входящее в P_i, совпадает с одним из имён $A_1, \dots, A_n$.

Мы будем предполагать, что каждому процессному имени соответствует единственное РО, в котором это имя является левой частью одного из равенств.

В параграфе 5.5 мы определим соответствие, которое сопоставляет каждому ПВ P некоторый процесс $[\![P]\!]$. Для определения этого соответствия мы сначала изложим

- понятие **вложения процессов**, и
- понятие **предела** последовательности вложенных процессов.

а также утверждения, связанные с этими понятиями.

5.3 Вложение процессов

Пусть заданы два процесса

$$P_i = (S_i, s_i^0, R_i) \quad (i = 1, 2)$$

и f – инъективное отображение из S_1 в S_2.

Мы будем говорить, что f является **вложением** P_1 в P_2, если

- $f(s_1^0) = s_2^0$, и
- для любых $s', s'' \in S_1$ и любого $a \in Act$

$$(s' \xrightarrow{a} s'') \in R_1 \quad \Leftrightarrow \quad (f(s') \xrightarrow{a} f(s'')) \in R_2$$

Для каждой пары процессов P_1, P_2 знакосочетание

$$P_1 \hookrightarrow P_2$$

является сокращённой записью утверждения о том, что существует вложение P_1 в P_2.

Теорема 27. Пусть $P_1 \hookrightarrow P_2$. Тогда

- $a.P_1 \hookrightarrow a.P_2$
- $P_1 + P \hookrightarrow P_2 + P$
- $P_1 \,|\, P \hookrightarrow P_2 \,|\, P$
- $P_1 \setminus L \hookrightarrow P_2 \setminus L$, и
- $P_1[f] \hookrightarrow P_2[f]$. ■

Ниже мы рассматриваем выражения, построенные из процессов, и символов операций над процессами ($a.$, $+$, $|$, $\setminus L$, $[f]$). Понятие такого выражения отличается от понятия ПВ, и мы называем такие выражения **выражениями над процессами**. Для каждого выражения над процессами определён процесс, являющийся значением этого выражения. В нижеследующих рассуждениях мы будем обозначать выражение над процессами и его значение одним и тем же символом.

Теорема 28.
Пусть

- P – выражение над процессами, в которое входят процессы

$$P_1, \ldots, P_n$$

- для каждого $i = 1, \ldots, n \quad P_i \hookrightarrow P'_i$, и
- P' – выражение, получаемое из P заменой для каждого $i = 1, \ldots, n$ каждого вхождения процесса P_i на соответствующий процесс P'_i.

Тогда $P \hookrightarrow P'$.

Доказательство.

Данная теорема доказывается индукцией по структуре выражения P: мы докажем, что для каждого подвыражения Q выражения P верно утверждение

$$Q \hookrightarrow Q' \tag{5.2}$$

где Q' – подвыражение выражения P', которое соответствует подвыражению Q .

базис индукции:

Если $Q = P_i$, то $Q' = P'_i$, и (5.2) верно по предположению.

индуктивный переход:

Из теоремы 27 следует, что для каждого подвыражения Q выражения P имеет место импликация: если для каждого собственного подвыражения Q_1 выражения Q (т.е. $Q_1 \neq Q$)

$$Q_1 \hookrightarrow Q'_1$$

то верно (5.2).

Таким образом, (5.2) верно для каждого подвыражения Q выражения P. В частности, (5.2) верно для P. ■

5.4 Предел последовательности вложенных процессов

Пусть задана последовательность процессов

$$\{P_k \mid k \geq 0\} \tag{5.3}$$

такая, что

$$\forall k \geq 0 \quad P_k \hookrightarrow P_{k+1} \tag{5.4}$$

Последовательность процессов (5.3), удовлетвряющая условию (5.4), называется **последовательностью вложенных процессов**.

Определим процесс P, называемый **пределом** последовательности (5.3).

Пусть процессы P_k $(k \geq 0)$ имеют вид

$$P_k = (S_k, s_k^0, R_k)$$

Мы можем считать, что все множества S_k $(k \geq 0)$ попарно не пересекаются.

Из (5.4) следует, что для каждого $k \geq 0$ существует инъективное отображение

$$f_k : S_k \to S_{k+1}$$

такое, что

- $f_k(s_k^0) = s_{k+1}^0$, и
- для любых $s', s'' \in S_k$ и любого $a \in Act$

$$(s' \stackrel{a}{\to} s'') \in R_k \quad \Leftrightarrow \quad (f_k(s') \stackrel{a}{\to} f_k(s'')) \in R_{k+1}$$

Обозначим

- символом S объединение $\bigcup\limits_{k \geq 0} S_k$, и
- символом ρ – минимальную эквивалентность на S, обладающую следующим свойством:

$$\forall\, k \geq 0, \quad \forall s \in S_k \quad (s, f_k(s)) \in \rho$$

Компоненты искомого процесса P имеют следующий вид.

1. Состояниями процесса P являются классы разбиения множества S по эквивалентности ρ.
2. Начальным состоянием процесса P является класс, содержащий s_0^0.
3. Переходы процесса P имеют вид

$$[s'] \xrightarrow{\ a\ } [s'']$$

где $(\, s' \xrightarrow{\ a\ } s'') \in R_k$ для некоторого $k \geq 0$.

Определённый выше предел последовательности (5.3) мы будем обозначать знакосочетанием

$$\lim_{k \to \infty} P_k$$

Из определения предела последовательности (5.3) непосредственно следует, что для каждого $k \geq 0$

$$P_k \hookrightarrow \lim_{k \to \infty} P_k$$

Теорема 29.

Пусть заданы последовательности вложенных процессов

$$\{P_k \mid k \geq 0\} \quad \text{и} \quad \{Q_k \mid k \geq 0\}$$

Тогда

- $\lim\limits_{k\to\infty}(a.P_k) = a.(\lim\limits_{k\to\infty} P_k)$
- $\lim\limits_{k\to\infty}(P_k + Q_k) = (\lim\limits_{k\to\infty} P_k) + (\lim\limits_{k\to\infty} Q_k)$
- $\lim\limits_{k\to\infty}(P_k \mid Q_k) = (\lim\limits_{k\to\infty} P_k) \mid (\lim\limits_{k\to\infty} Q_k)$
- $\lim\limits_{k\to\infty}(P_k \setminus L) = (\lim\limits_{k\to\infty} P_k) \setminus L$
- $\lim\limits_{k\to\infty}(P_k[f]) = (\lim\limits_{k\to\infty} P_k)[f]$ ■

Ниже мы будем использовать следующее обозначение: если

- P – ПВ, в которое входят процессные имена $A_1, \ldots, A_n$, и
- $P_1, \ldots, P_n$ – некоторые процессы

то знакосочетание

$$P(P_1/A_1, \ldots, P_n/A_n)$$

обозначает выражение над процессами (а также его значение), получаемое из P заменой для каждого $i = 1, \ldots, n$ каждого вхождения процессного имени A_i на соответствующий процесс P_i.

Теорема 30.
Пусть заданы

- ПВ P, в которое входят процессные имена $A_1, \ldots, A_n$, и
- последовательности вложенных процессов

$$\{P_i^{(k)} \mid k \geq 0\} \quad (i = 1, \ldots, n)$$

Тогда

$$P((\lim_{k\to\infty} P_1^{(k)})/A_1, \ldots, (\lim_{k\to\infty} P_n^{(k)})/A_n) =$$
$$= \lim_{k\to\infty} P(P_1^{(k)}/A_1, \ldots, P_n^{(k)}/A_n)$$

Доказательство.
Данная теорема доказывается индукцией по структуре ПВ P, с использованием теоремы 29. ■

5.5 Процессы, определяемые процессными выражениями

В этом параграфе мы излагаем правило, которое сопоставляет каждому ПВ P процесс $[\![P]\!]$, определяемый этим ПВ.

Процессы, определяемые процессными константами, являются значениями этих констант.

Процессы, определяемые ПВ вида

$$a.P, \quad P_1 + P_2, \quad P_1 \mid P_2, \quad P \setminus L, \quad P[f]$$

являются результатами применения соответствующих операций к процессам определяемым ПВ P, P_1 и P_2, т.е.

$$\begin{array}{l} [\![a.P]\!] \stackrel{\text{def}}{=} a.[\![P]\!] \\ [\![P_1 + P_2]\!] \stackrel{\text{def}}{=} [\![P_1]\!] + [\![P_2]\!] \\ [\![P_1 \mid P_2]\!] \stackrel{\text{def}}{=} [\![P_1]\!] \mid [\![P_2]\!] \\ [\![P \setminus L]\!] \stackrel{\text{def}}{=} [\![P]\!] \setminus L \\ [\![P\ [f]\ ]\!] \stackrel{\text{def}}{=} [\![P]\!]\ [f] \end{array}$$

Опишем теперь правило, сопоставляющее процессы процессным именам.

Пусть задано РО вида (5.1). Определим последовательность списков процессов

$$\{(P_1^{(k)}, \ldots, P_n^{(k)}) \mid k \geq 0\} \tag{5.5}$$

следующим образом:

- $P_1^{(0)} \stackrel{\text{def}}{=} \mathbf{0}, \quad \ldots, \quad P_n^{(0)} \stackrel{\text{def}}{=} \mathbf{0}$
- если процессы $P_1^{(k)}, \ldots, P_n^{(k)}$ уже определены, то для каждого $i = 1, \ldots, n$

$$P_i^{(k+1)} \stackrel{\text{def}}{=} P_i(P_1^{(k)}/A_1, \ldots, P_n^{(k)}/A_n)$$

Докажем, что для каждого $k \geq 0$ и для каждого $i = 1, \ldots, n$

$$P_i^{(k)} \hookrightarrow P_i^{(k+1)} \tag{5.6}$$

Доказательство будем вести индукцией по k.

базис индукции:

Если $k = 0$, то $P_i^{(0)}$ по определению совпадает с процессом $\mathbf{0}$, который можно вложить в любой процесс.

индуктивный переход:

Пусть для каждого $i = 1, \ldots, n \quad P_i^{(k-1)} \hookrightarrow P_i^{(k)}$.

По определению процессов из совокупности (5.5), имеют место соотношения

$$P_i^{(k)} = P_i(P_1^{(k-1)}/A_1, \ldots, P_n^{(k-1)}/A_n)$$
$$P_i^{(k+1)} = P_i(P_1^{(k)}/A_1, \ldots, P_n^{(k)}/A_n)$$

Соотношение $P_i^{(k)} \hookrightarrow P_i^{(k+1)}$ следует из теоремы 28. ■

Определим для каждого $i = 1, \ldots, n$ процесс $[\![A_i]\!]$ как предел

$$[\![A_i]\!] \stackrel{\text{def}}{=} \lim_{k\to\infty} P_i^{(k)}$$

Из теоремы 30 следует, что для каждого $i = 1, \ldots, n$ верна цепочка равенств

$$\begin{aligned} & P_i([\![A_1]\!]/A_1, \ldots, [\![A_n]\!]/A_n) = \\ & = P_i((\lim_{k\to\infty} P_1^{(k)})/A_1, \ldots, (\lim_{k\to\infty} P_n^{(k)})/A_n) = \\ & = \lim_{k\to\infty} P_i(P_1^{(k)}/A_1, \ldots, P_n^{(k)}/A_n) = \\ & = \lim_{k\to\infty} (P_i^{(k+1)}) = [\![A_i]\!] \end{aligned}$$

т.е. список процессов

$$[\![A_1]\!], \ldots, [\![A_n]\!]$$

является решением системы уравнений, соответствующей РО (5.1) (переменными в этой системе уравнений являются процессные имена).

5.6 Эквивалентность РО

Пусть заданы два РО вида

$$\left\{\begin{array}{l} A_1^{(1)} = P_1^{(1)} \\ \ldots \\ A_n^{(1)} = P_n^{(1)} \end{array}\right. \quad \text{и} \quad \left\{\begin{array}{l} A_1^{(2)} = P_1^{(2)} \\ \ldots \\ A_n^{(2)} = P_n^{(2)} \end{array}\right. \tag{5.7}$$

Для каждого списка процессов $Q_1, \ldots, Q_n$ мы будем обозначать выражение над процессами (и его значение)

$$P_i^{(j)}(Q_1/A_1^{(j)}, \ldots, Q_n/A_n^{(j)}) \qquad (i = 1, \ldots, n;\ j = 1, 2)$$

сокращённо в виде знакосочетания

$$P_i^{(j)}(Q_1, \ldots, Q_n)$$

Пусть задана некоторая эквивалентность μ на множестве всех процессов.

Мы будем говорить, что РО (5.7) являются **эквивалентными** относительно μ, если для

- каждого списка процессов $Q_1, \ldots, Q_n$, и
- каждого $i = 1, \ldots, n$

имеет место соотношение

$$\left(P_i^{(1)}(Q_1, \ldots, Q_n) \, , \; P_i^{(2)}(Q_1, \ldots, Q_n) \right) \in \mu$$

Теорема 31.
Пусть заданы

- два РО вида (5.7), и
- конгруэнция μ на множестве процессов.

Если РО (5.7) эквивалентны относительно μ, то процессы, определяемые этими РО, т.е.

$$\{[\![A_i^{(1)}]\!] \mid i = 1, \ldots, n\} \quad \text{и} \quad \{[\![A_i^{(2)}]\!] \mid i = 1, \ldots, n\}$$

тоже эквивалентны относительно μ, т.е. для каждого $i = 1, \ldots, n$ имеет место соотношение

$$\left([\![A_i^{(1)}]\!] \, , \; [\![A_i^{(2)}]\!] \right) \in \mu \quad \blacksquare$$

5.7 Переходы на $PExpr$

Существует другой способ определения соответствия между ПВ и процессами. Данный способ связан с определением **множества переходов** $\mathcal{R}$ на совокупности $PExpr$ всех ПВ. Каждый переход из $\mathcal{R}$ представляет собой тройку

$$(P, a, P') \tag{5.8}$$

где $P, P' \in PExpr$, и $a \in Act$.

Если (5.8) $\in \mathcal{R}$, то мы сокращённо обозначаем этот факт в виде знакосочетания

$$P \xrightarrow{\;a\;} P' \tag{5.9}$$

Понятие перехода определяется индуктивно, т.е.

- указываются тройки вида (5.8), которые являются переходами по определению, и
- описываются правила построения новых переходов из уже имеющихся.

В этом параграфе мы предполагаем, что

- значением каждой процессной константы является конечный процесс, и

- каждый конечный процесс является значением некоторой процессной константы.

В нижеследующих правилах, определяющих множество переходов $\mathcal{R}$, символы P, P' обозначают произвольные ПВ, и символ a обозначает произвольное действие из Act.

1. если P – процессная константа, то

$$P \xrightarrow{a} P'$$

где P' – процессная константа, такая, что

- значения P и P' имеют вид

$$(S, s^0, R) \quad \text{и} \quad (S, s^1, R)$$

соответственно, и

- R содержит переход $s^0 \xrightarrow{a} s^1$

2. $a.P \xrightarrow{a} P$

3. если $P \xrightarrow{a} P'$, то

- $P + Q \xrightarrow{a} P'$, и
- $Q + P \xrightarrow{a} P'$
- $P \,|\, Q \xrightarrow{a} P' \,|\, Q$, и
- $Q \,|\, P \xrightarrow{a} Q \,|\, P'$
- если $L \subseteq Names$, $a \neq \tau$, и $name(a) \notin L$, то

$$P \setminus L \xrightarrow{a} P' \setminus L$$

- для каждого переименования f

$$P[f] \xrightarrow{f(a)} P'[f]$$

4. если $a \neq \tau$, то из

$$P_1 \xrightarrow{a} P_1' \quad \text{и} \quad P_2 \xrightarrow{\bar{a}} P_2'$$

следует, что

$$P_1 \,|\, P_2 \xrightarrow{\tau} P_1' \,|\, P_2'$$

5. для каждого РО (5.1) и каждого $i \in \{1, \ldots, n\}$

$$\begin{array}{ll} \text{если} & P_i \xrightarrow{a} P' \\ \text{то} & A_i \xrightarrow{a} P' \end{array} \tag{5.10}$$

Можно доказать, что для каждого ПВ P существует лишь конечное множество переходов с началом P, т.е. имеющих вид

$$P \xrightarrow{a} P'$$

Для каждого ПВ $P \in PExpr$ процесс $[\![P]\!]$, соответствующий этому ПВ, имеет вид

$$(PExpr, P, \mathcal{R})$$

При данном определении соответствия между ПВ и процессами имеет место следующая теорема.

Теорема 32.
Для каждого РО (5.1) и каждого $i = 1, \ldots, n$

$$[\![A_i]\!] \sim P_i([\![A_1]\!]/A_1, \ldots, [\![A_n]\!]/A_n)$$

(т.е. список процессов $[\![A_1]\!], \ldots, [\![A_n]\!]$ является решением системы уравнений, соответствующей РО (5.1) **с точностью до** $\sim$). ■

5.8 Доказательство эквивалентности процессов при помощи РО

Можно доказывать эквивалентность ($\sim$ или $\overset{+}{\approx}$) двух процессов путём предъявления РО, такого, что оба этих процесса являются компонентами с одинаковыми номерами некоторых решений системы уравнений, соответствующей этому РО.

Соответствующие эквивалентности обосновываются теоремой 33.

Для формулировки этой теоремы мы введём следующее вспомогательное понятие.

Пусть заданы

- бинарное отношение μ на множестве всех процессов, и
- РО вида (5.1).

Мы будем говорить, что список процессов, определяемый РО (5.1), единствен с точностью до μ, если для каждой пары списков процессов

$$(Q_1^{(1)}, \ldots, Q_n^{(1)}) \quad \text{и} \quad (Q_1^{(2)}, \ldots, Q_n^{(2)})$$

удовлетворяющей следующему условию: для каждого $i = 1, \ldots, n$

$$\begin{array}{l} (\ [\![Q_i^{(1)}]\!]\ ,\ P_i(Q_1^{(1)}/A_1, \ldots, Q_n^{(1)}/A_n)\ ) \ \in \mu \\ (\ [\![Q_i^{(2)}]\!]\ ,\ P_i(Q_1^{(2)}/A_1, \ldots, Q_n^{(2)}/A_n)\ ) \ \in \mu \end{array}$$

имеет место соотношение

$$\forall i = 1, \ldots, n \quad \left([\![Q_i^{(1)}]\!] \, , \, [\![Q_i^{(2)}]\!] \right) \in \mu$$

Теорема 33.
Пусть задано РО вида (5.1).

1. Если каждое вхождение каждого процессного имени A_i в каждое ПВ P_j содержится в подвыражении вида $a.Q$, то список процессов, определяемый РО (5.1), единствен с точностью до $\sim$.

2. Если

   - каждое вхождение каждого A_i в каждое P_j содержится в подвыражении вида $a.Q$, где $a \neq \tau$, и
   - каждое вхождение каждого A_i в каждое P_j содержится только в подвыражениях вида $a.Q$ и $Q_1 + Q_2$

   то список процессов, определяемый РО (5.1), единствен с точностью до $\overset{+}{\approx}$. ■

5.9 Проблемы, связанные с понятием РО

1. Распознавание существования конечных процессов, эквивалентных (относительно $\sim$, $\approx$, $\overset{+}{\approx}$) процессам вида $[\![A]\!]$.

2. Построение алгоритмов нахождения минимальных процессов, эквивалентных процессам вида $[\![A]\!]$ в том случае, когда эти процессы конечны.

3. Распознавание эквивалентности процессов вида $[\![A]\!]$
   (эти процессы могут быть бесконечными, и методы из главы 4 для них не подходят).

4. Распознавание эквивалентности РО.

5. Нахождение необходимых и достаточных условий единственности списка процессов, определяемого РО (с точностью до $\sim$, $\overset{+}{\approx}$).

Глава 6

Примеры доказательства свойств процессов

6.1 Потоковые графы

Если процесс P можно представить в виде алгебраического выражения

$$P(P_1, \ldots, P_n) \tag{6.1}$$

в которое входят процессы $P_1, \ldots, P_n$, соединённые символами операций

- параллельной композиции,
- ограничения, и
- переименования

то P называется **структурной композицией** процессов $P_1, \ldots, P_n$.

Если процесс P является структурной композицией, то ему можно сопоставить некоторый графический объект

$$G(P)$$

называемый **потоковым графом (ПГ)** процесса P.

Представление структурной композиции в виде ПГ повышает наглядность и облегчает понимание взаимосвязи между её компонентами.

Для построения ПГ $G(P)$ для процесса P вида (6.1) строятся ПГ, соответствующие всем подвыражениям выражения (6.1).

Элементарные ПГ:

Для каждого $i = 1, \ldots, n$, и каждого вхождения P_i в выражение (6.1), ПГ $G(P_i)$, соответствующий этому вхождению, имеет вид овала, внутри которого написано знакосочетание P_i.

На периметре овала рисуется несколько кружочков, называемых **портами**.

Каждый порт соответствует некоторому действию из множества $Act(P_i)$, причём

- если это действие имеет вид $\alpha!$, то соответствующий этому действию порт рисуется чёрным цветом, и
- если это действие имеет вид $\alpha?$, то соответствующий этому действию порт рисуется белым цветом.

Около каждого порта написана его метка, равная тому действию из $Act(P_i)$, которому соответствует этот порт.

Отметим, что если P_i имеет несколько вхождений в выражение (6.1), то для каждого такого вхождения рисуется отдельный элементарный ПГ $G(P_i)$.

Параллельная композиция:
Если (6.1) содержит подвыражение вида $P' \mid P''$, то $G(P' \mid P'')$ получается

- дизъюнктным объединением $G(P')$ и $G(P'')$, и
- соединением стрелочками в этом дизъюнктном объединении портов ПГ $G(P')$ и $G(P'')$ с комплементарными метками: если
  - один из этих ПГ содержит порт с меткой $\alpha!$, и
  - другой из этих ПГ содержит порт с меткой $\alpha?$,

  то рисуется стрелочка с меткой α от первого порта к второму.

Ограничение:
$G(P' \setminus L)$ получается из $G(P')$ удалением меток портов, имена которых принадлежат L.

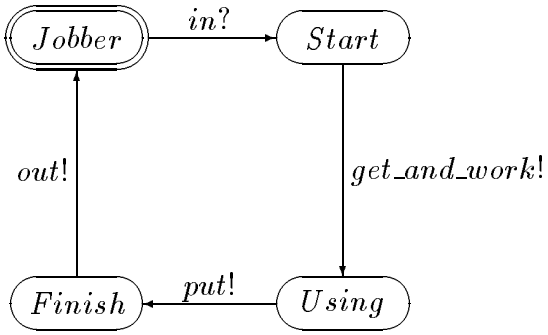
Переименование:
$G(P'[f])$ получается из $G(P')$ соответствующим переименованием меток портов.

В излагаемых ниже примерах процессов приводятся ПГ, соответствующие этим процессам.

6.2 Мастерская

Рассмотрим модель мастерской, в которой работают двое рабочих, пользующиеся для работы одним молотком.

Поведение каждого рабочего в мастерской описывается процессом $Jobber$



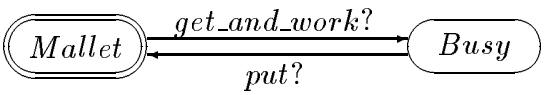
- Действия $in?$ и $out!$ используются для взаимодействия рабочего с заказчиком, и обозначают соответственно
  - прием материала, и
  - выдачу готового изделия.
- Действия $get\_and\_work!$ и $put!$ используются для взаимодействия рабочего с молотком, и обозначают соответственно
  - взятие молотка и выполнение с его помощью некоторых действий, и
  - возвращение молотка на место.

Обратим внимание, что действие $get\_and\_work!$ состоит из нескольких действий, которые мы не детализируем и агрегируем все их в одно действие.

Согласно графовому представлению процесса $Jobber$, рабочий

- сначала принимает материал,
- затем берет молоток и работает,
- после чего кладет молоток,
- выдает готовое изделие,
- и все повторяется сначала.

Поведение молотка мы представляем при помощи следующего процесса $Mallet$:



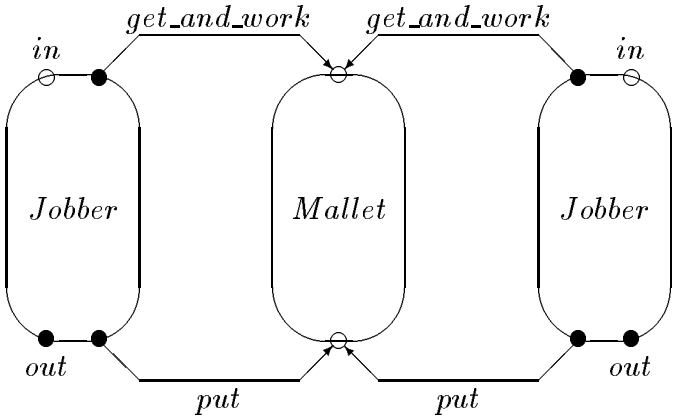
Отметим, что объект “молоток” и процесс “молоток” - это разные понятия.

Функционирование мастерской определяется при помощи следующего процесса Job_Shop:

$$Job\_Shop = (Jobber \mid Jobber \mid Mallet) \setminus L$$

где $L = \{get\_and\_work,\ put\}$.

Потоковый граф процесса Job_Shop имеет следующий вид.

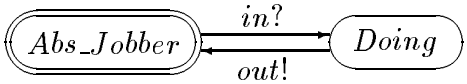


Введем теперь понятие "абстрактного рабочего", про которого известно, что он циклически

- принимает материал, и
- выдает готовые изделия

но ничего неизвестно о подробностях процесса его работы.

Поведение "абстрактного рабочего" мы зададим при помощи следующего процесса Abs_Jobber:



Поведение "абстрактной мастерской" мы зададим при помощи следующего процесса Abs_Job_Shop:

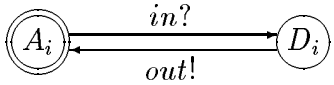
$$Abs\_Job\_Shop = Abs\_Jobber \mid Abs\_Jobber$$

"Абстрактную мастерскую" мы будем использовать как **спецификацию** мастерской. Процесс "абстрактная мастерская" представляет поведение мастерской без учета деталей ее реализации.

Докажем соответствие мастерской ее спецификации, то есть наличие наблюдаемой конгруэнции

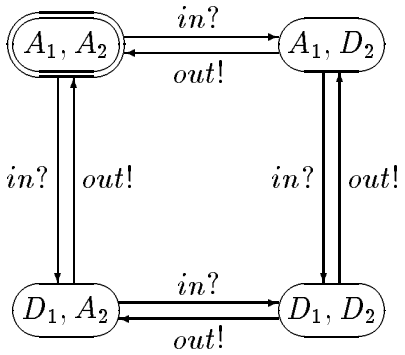
$$Job\_Shop \overset{+}{\approx} Abs\_Job\_Shop \tag{6.2}$$

Процесс Abs_Job_Shop является параллельной композицией двух процессов Abs_Jobber. В целях предотвращения коллизии в обозначениях, мы выберем различные идентификаторы для обозначения состояний этих процессов. Пусть, например, эти процессы имеют вид

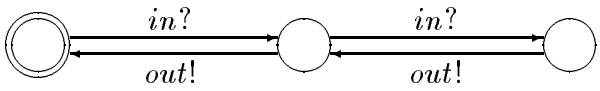


где $i = 1, 2$.

Параллельная композиция этих процессов имеет вид



Применяя к данному процессу процедуру минимизации относительно наблюдаемой эквивалентности, мы получим процесс



(6.3)

Процесс Job_Shop имеет $4 \cdot 4 \cdot 2 = 32$ состояния, и мы не приводим его здесь ввиду его большой громоздкости. Если минимизировать этот процесс относительно наблюдаемой эквивалентности, то получится процесс, изоморфный процессу (6.3). Это означает, что имеет место соотношение

$$Job\_Shop \approx Abs\_Job\_Shop \tag{6.4}$$

Поскольку из начальных состояний процессов Job_Shop и Abs_Job_Shop не выходит рёбер с меткой τ, то отсюда и из (6.4) следует искомое соотношение (6.2).

6.3 Неконфликтное использование ресурса

Предположим, что имеется некоторая фирма, сотрудники которой объединены в несколько групп.

В здании, где работает фирма, выделена одна комната, которую каждая из групп может использовать для проведения своих рабочих совещаний.

Предположим, что необходимо обеспечить неконфликтное использование этой комнаты группами. Это означает, что когда одна из групп проводит в комнате совещание, другой группе должно быть запрещено проводить своё совещание в этой комнате.

Для решения этой задачи создаётся специальный процесс – **диспетчер**.

Если какая-либо из групп хочет провести совещание в этой комнате, она должна послать диспетчеру заявку на предоставление ей права пользования комнатой для проведения этого совещания.

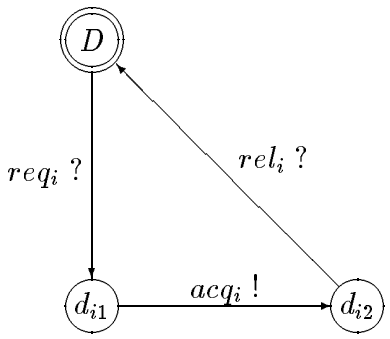
Когда диспетчер разрешает какой-либо группе использовать комнату, он посылает ей уведомление об этом.

Окончив совещание, группа должна сообщить об этом диспетчеру, чтобы диспетчер знал, что комната стала свободной и доступна для других групп.

Рассмотрим описание работы указанной системы при помощи теории процессов.

Пусть число групп равно n $(n \geq 2)$.

Диспетчер мы опишем как процесс с именем D, графовое представление которого содержит для каждого $i = 1, \ldots, n$ подграф



т.е.

$$D \sim \sum_{i=1}^{n} req_i?.\ acq_i!.\ rel_i?.\ D$$

Действия, входящие в $Act(D)$, имеют следующий смысл:

- $req_i?$ – получение заявки от i-й группы
- $acq_i!$ – уведомление i-й группы о том, что она имеет право пользоваться комнатой

- $rel_i?$ – получение сообщения от i-й группы об освобождении ею комнаты.

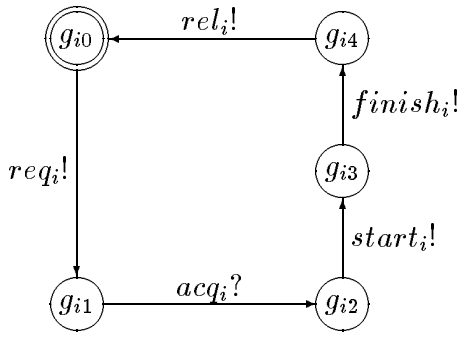
Опишем теперь поведение каждой группы.

(Мы будем описывать только взаимодействие групп с диспетчером и с комнатой, и не будем касаться прочих их функций).

Мы будем представлять

- начало проведения совещания в комнате действием $start!$, и
- окончание совещания - действием $finish!$.

Поведение i-й группы мы опишем в виде процесса G_i, который имеет следующее графовое представление:



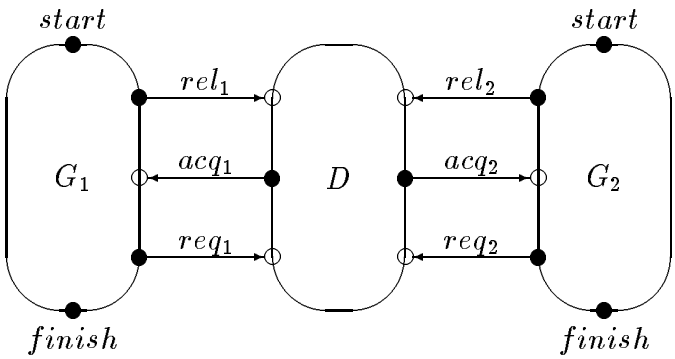
т.е. $G_i \sim req_i!.\ acq_i?.\ start!.\ finish!.\ rel_i!.\ G_i$.

Совместное поведение диспетчера и групп можно описать следующим процессом:

$$Sys = (D \mid G_1 \mid \ldots \mid G_n) \setminus L$$

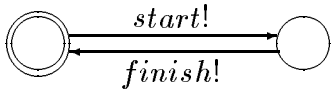
где $L = \{req_i, acq_i, rel_i \mid i = 1, \ldots, n\}$.

Потоковый граф системы, состоящей из диспетчера и групп для $n = 2$ показан на рисунке



Покажем теперь, что алгоритмы работы диспетчера и групп действительно обеспечивают неконфликтный режим использования комнаты, который заключается в том, что после начала проведения совещания в комнате какой-либо группой (то есть после выполнения этой группой действия $start!$) никакая другая группа не может начать проводить в этой комнате своё совещание (т.е. тоже выполнить действие $start!$), до тех пор первая группа не закончит своё совещание (т.е. пока не будет выполнено действие $finish!$).

Определим процесс $Spec$ следующим образом:



т.е. $Spec \sim start!.\ finish!.\ Spec$.

Наличие неконфликтного режима использования комнаты эквивалентно истинности соотношения

$$Sys \approx Spec \tag{6.5}$$

Это соотношение можно рассматривать как требование к системе, состоящей из диспетчера и групп.

Докажем соотношение (6.5).

Преобразуем процесс Sys, применив несколько раз теорему о разложении:

$$\begin{aligned}
&Sys \sim \\
&\sim \sum_{i=1}^{n} \tau.\left(\begin{array}{l} acq_i!.\ rel_i?.\ D \mid G_1 \mid \ldots \\ \ldots \mid acq_i?.\ start!.\ finish!.\ rel_i!.\ G_i \mid \ldots \\ \ldots \mid G_n \end{array}\right) \setminus L \sim \\
&\sim \sum_{i=1}^{n} \tau.\tau.\left(\begin{array}{l} rel_i?.\ D \mid G_1 \mid \ldots \\ \ldots \mid start!.\ finish!.\ rel_i!.\ G_i \mid \ldots \\ \ldots \mid G_n \end{array}\right) \setminus L \sim \\
&\sim \sum_{i=1}^{n} \tau.\tau.start!.\left(\begin{array}{l} rel_i?.\ D \mid G_1 \mid \ldots \\ \ldots \mid finish!.\ rel_i!.\ G_i \mid \ldots \\ \ldots \mid G_n \end{array}\right) \setminus L \sim \\
&\sim \sum_{i=1}^{n} \tau.\tau.start!.\ finish!.\left(\begin{array}{l} rel_i?.\ D \mid G_1 \mid \ldots \\ \ldots \mid rel_i!.\ G_i \mid \ldots \\ \ldots \mid G_n \end{array}\right) \setminus L \sim \\
&\sim \sum_{i=1}^{n} \tau.\tau.start!.\ finish!.\ \tau.\underbrace{\left(\begin{array}{l} D \mid G_1 \mid \ldots \\ \ldots \mid G_i \mid \ldots \\ \ldots \mid G_n \end{array}\right) \setminus L}_{Sys} = \\
&= \sum_{i=1}^{n} \tau.\tau.start!.\ finish!.\ \tau.Sys
\end{aligned}$$

Воспользовавшись тождествами

$$P + P \sim P \quad \text{и} \quad \alpha.\tau.P \overset{+}{\approx} \alpha.P$$

получаем отсюда, что

$$Sys \overset{+}{\approx} \tau.start!.\ finish!.\ Sys$$

Рассмотрим теперь уравнение

$$X = \tau.start!.\ finish!.\ X \tag{6.6}$$

Согласно теореме 33 из параграфа 5.8, решение этого уравнения с точностью до $\overset{+}{\approx}$ единственно.

Как было показано выше, процесс Sys является решением уравнения (6.6) с точностью до $\overset{+}{\approx}$.

Процесс $\tau.Spec$ тоже является решением уравнения (6.6) с точностью до $\overset{+}{\approx}$, так как

$$\tau.Spec \sim \tau.start!.\ finish!.\ Spec \overset{+}{\approx}$$
$$\overset{+}{\approx} \tau.start!.\ finish!.\ (\tau.Spec)$$

Следовательно, имеет место соотношение

$$Sys \overset{+}{\approx} \tau.Spec$$

из которого следует (6.5).

6.4 Планировщик

Предположим, что имеется n процессов

$$P_1, \ldots, P_n \tag{6.7}$$

и для каждого $i = 1, \ldots, n$ среди действий, которые может выполнить P_i, есть два служебных действия:

- действие $\alpha_i?$, которое представляет собой сигнал

$$P_i \textit{ начинает работу} \tag{6.8}$$

- действие $\beta_i?$, которое представляет собой сигнал

$$P_i \textit{ заканчивает работу} \tag{6.9}$$

Мы предполагаем, что все имена

$$\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_n \tag{6.10}$$

различны, и для каждого $i = 1, \ldots, n$ имена из

$$names(Act(P_i)) \setminus \{\alpha_i, \beta_i\}$$

не совпадают ни с одним именем из (6.10). Обозначим множество имён из (6.10) символом L.

Для каждого $i = 1, \ldots, n$ действия из множества

$$Act(P_i) \setminus \{\alpha_i?, \beta_i?\}$$

называются **собственными действиями** процесса P_i.

Произвольная трасса каждого процесса P_i может содержать действия $\alpha_i?$ и $\beta_i?$ в любом количестве и в любом порядке.

Мы хотели бы создать новый процесс P, в котором все процессы $P_1, \ldots, P_n$ работали бы совместно, причём эта совместная работа должна подчиняться определённому режиму. Процесс P должен иметь вид

$$P = (P_1 \mid \ldots \mid P_n \mid Sch) \setminus L$$

где процесс Sch называется **планировщиком** и предназначен для задания требуемого режима работы процессов $P_1, \ldots, P_n$. Процесс Sch должен выполнять только действия из множества

$$\{\alpha_1!, \ldots, \alpha_n!, \beta_1!, \ldots, \beta_n!\} \tag{6.11}$$

В силу определения процесса P, для каждого $i = 1, \ldots, n$

- каждое исполнение процессом $P_i \in (6.7)$ действия $\alpha_i?$ или $\beta_i?$ в составе процесса P может произойти только одновременно с исполнением комплементарного действия процессом Sch, и
- исполнение этих действий будет невидимо за пределами процесса P.

Говоря неформально, каждый процесс P_i, работающий в составе процесса P, может начать или закончить какой-либо сеанс своей работы тогда и только тогда, когда планировщик Sch разрешит ему это сделать.

Режим, которому должна подчиняться работа процессов $P_1, \ldots, P_n$, заключается в следующих двух условиях.

1. Для каждого $i = 1, \ldots, n$ произвольная трасса процесса P_i, работающего в составе процесса P, должна иметь вид

   $$\alpha_i? \ \ldots \ \beta_i? \ \ldots \alpha_i? \ \ldots \ \beta_i? \ \ldots$$

   где точки изображают собственные действия процесса P_i, т.е. функционирование процесса P_i должно представлять собой последовательность сеансов вида

   $$\alpha_i? \ \ldots \ \beta_i? \ \ldots$$

   где каждый сеанс

- начинается с сигнала $\alpha_i?$ о начале сеанса
- далее выполняются некоторые собственные действия процесса P_i
- затем производится сигнал $\beta_i?$ о завершении сеанса,
- после чего процесс P_i может производить некоторые собственные действия (например, связанные с подготовкой к следующему сеансу).

2. Процессы P_i должны начинать новые сеансы только по очереди в циклическом порядке, т.е.
   - сначала может начать свой первый сеанс только процесс P_1
   - потом может начать свой первый сеанс процесс P_2
   - ...
   - затем может начать свой первый сеанс процесс P_n
   - после этого может начать свой второй сеанс процесс P_1
   - затем может начать свой второй сеанс процесс P_2
   - и т.д.

Отметим, что мы не требуем, чтобы каждый процесс P_i получал разрешение начать свой k-й сеанс только после того, как предыдущий процесс P_{i-1} завершит свой k-й сеанс. Однако мы требуем, чтобы каждый процесс P_i получал разрешение начать новый сеанс, только если он выполнил действие $\beta_i?$, сигнализирующее о завершении своего предыдущего сеанса.

Исполнение собственных действий процессами P_i может производиться в произвольном порядке, в том числе допускается и взаимодействие между этими процессами во время их работы в составе процесса P.

Описанный режим можно равносильным образом выразить в виде следующих двух условий на произвольную трассу

$$tr \in Tr(Sch)$$

В формулировке этих условий мы будем использовать следующее обозначение: если

$$tr \in Tr(Sch) \quad \text{и} \quad M \subseteq Act$$

то знакосочетание $tr\,|_M$ обозначает последовательность действий, получаемую из tr удалением всех действий, не принадлежащих M.

Условия, выражающие описанный выше режим, имеют следующий вид:

$$\begin{array}{l} \forall\, tr \in Tr(Sch),\ \forall\, i = 1, \ldots, n \\ tr\,|_{\{\alpha_i, \beta_i\}} \ = \ (\alpha_i!\ \beta_i!\ \alpha_i!\ \beta_i!\ \alpha_i!\ \beta_i!\ \ldots) \end{array} \tag{6.12}$$

и

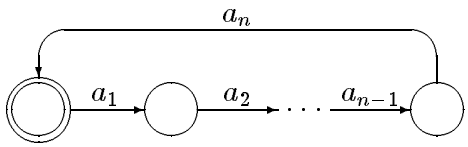
$$\begin{array}{l} \forall\, tr \in Tr(Sch) \\ tr \mid_{\{\alpha_1,\ldots,\alpha_n\}} \;=\; (\alpha_1! \;\ldots\; \alpha_n!\, \alpha_1! \;\ldots\; \alpha_n! \;\ldots) \end{array} \tag{6.13}$$

Данные условия можно выразить равносильным образом в виде условия наличия наблюдаемой эквивалентности между некоторыми процессами. Чтобы определить эти процессы, мы введём вспомогательные обозначения.

1. Пусть $a_1 \ldots a_n$ – некоторая последовательность действий из Act. Тогда знакосочетание

   $$(a_1 \ldots a_n)^*$$

   обозначает процесс, графовое представление которого имеет следующий вид:

2. Пусть заданы

   - некоторый процесс P, и
   - некоторое множество действий

   $$\{a_1, \ldots, a_k\} \subseteq Act \setminus \{\tau\} \tag{6.14}$$

   Знакосочетание

   $$hide\ (P, a_1, \ldots, a_k) \tag{6.15}$$

   обозначает процесс

   $$(\ P \mid (\overline{a_1})^* \mid \ldots \mid (\overline{a_k})^*\ ) \setminus names(\{a_1, \ldots, a_k\})$$

Процесс (6.15) можно рассматривать как процесс, получаемый из P заменой меток переходов: все метки переходов в P, принадлежащие множеству (6.14), заменяются на τ.

Используя введённые обозначения, условие (6.12) можно выразить следующим образом: для каждого $i = 1, \ldots n$

$$\begin{array}{l} hide \left(\begin{array}{ll} Sch, & \alpha_1!, \ldots, \alpha_{i-1}!, \alpha_{i+1}!, \ldots, \alpha_n! \\ & \beta_1!, \ldots, \beta_{i-1}!, \beta_{i+1}!, \ldots, \beta_n! \end{array} \right) \approx \\ \approx (\alpha_i!.\ \beta_i!)^* \end{array} \tag{6.16}$$

Условие (6.13) можно выразить следующим образом:

$$hide\ (Sch, \beta_1!, \ldots, \beta_n!) \approx (\alpha_1!.\ \ldots \alpha_n!)^* \tag{6.17}$$

Нетрудно заметить, что существует несколько планировщиков, удовлетворяющих данным условиям. Например, данным условиям удовлетворяют следующие планировщики:

- $Sch = (\alpha_1! \, \beta_1! \, \ldots \, \alpha_n! \, \beta_n!)^*$
- $Sch = (\alpha_1! \, \ldots \, \alpha_n! \, \beta_1! \, \ldots \, \beta_n!)^*$

Однако такие планировщики налагают слишком жёсткие ограничения на работу процессов $P_1, \ldots, P_n$.

Нам хотелось бы, чтобы планировщик допускал максимальную свободу в поведении процессов $P_1, \ldots, P_n$ в составе процесса P. Это одначает, что если в какой-либо момент времени

- какой-либо процесс P_i имеет намерение выполнить действие $a \in \{\alpha_i?, \beta_i?\}$, и
- это намерение процесса P_i не противоречит описанному выше режиму

то планировщик не должен отказывать процессу P_i в возможности выполнить это действие в текущий момент времени, т.е. среди действий планировщика, которые он может выполнить в текущий момент времени, должно присутствовать действие $\overline{a}$ (не факт, что именно это действие будет выполнено в текущий момент времени, но среди возможных действий оно должно присутствовать).

Сформулированное выше неформальное описание максимальной свободы в поведении планировщика можно формально уточнить следующим образом:

- каждому состоянию s планировщика можно сопоставить метку $label(s)$, имеющую вид пары (i, X) где
  - $i \in \{1, \ldots, n\}$, i = номер того процесса, который имеет право начать очередной сеанс в текущий момент времени
  - $X \subseteq \{1, \ldots, n\} \setminus \{i\}$, X = список активных процессов на текущий момент времени (т.е. таких процессов, которые начали очередной сеанс, но пока его не закончили)
- начальное состояние планировщика имеет метку $(1, \emptyset)$
- множество переходов состоит из
  - переходов вида

    $$s \xrightarrow{\alpha_i!} s'$$

    где
    - $label(s) = (i, X)$

  * $label(s') = (next(i), X \cup \{i\})$, где
    $next(i) \stackrel{\text{def}}{=} \begin{cases} i+1, & \text{если } i < n, \text{ и} \\ 1, & \text{если } i = n \end{cases}$

– и переходов вида

$$s \xrightarrow{\beta_j!} s'$$

где

  * $label(s) = (i, X)$,
  * $label(s') = (i, X \setminus \{j\})$, причём $j \in X$

Сформулированное описание свойств требуемого планировщика можно рассматривать как его определение:

- в качестве множества состояний планировщика мы можем взять просто множество пар вида

$$\{(i, X) \in \{1, \ldots, n\} \times \mathcal{P}(\{1, \ldots, n\}) \mid i \notin X\}$$

  (т.е. каждое состояние совпадает со своей меткой)

- и определить множество переходов так, как описано выше.

Обозначим такой планировщик знакосочетанием Sch_0.

Описание планировщика Sch_0 содержит существенный недостаток: размер такого описания экспоненциально зависит от числа процессов (6.7), которыми нужно управлять (число состояний Sch_0 равно $n \cdot 2^{n-1}$), что не позволяет быстро модифицировать такой планировщик в том случае, когда множество процессов (6.7) изменяется (например, к нему добавляются новые процессы, или удаляются старые).

Мы можем использовать Sch_0 только как эталон, с которым мы каким-либо способом будем сравнивать другие планировщики.

Для решения поставленной задачи мы определим другой планировщик Sch. Мы будем определять его

- не путём явного описания состояний и переходов,

- а путём задания некоторого выражения, которое определяет его в терминах композиции процессов достаточно простого вида.

Данное описание будет лишено сформулированного выше недостатка.

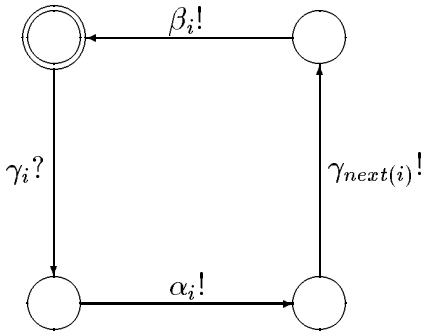
В описании планировщика Sch мы будем использовать новые вспомогательные имена $\gamma_1, \ldots, \gamma_n$. Обозначим множество этих имён символом Γ.

Процесс Sch определяется следующим образом:

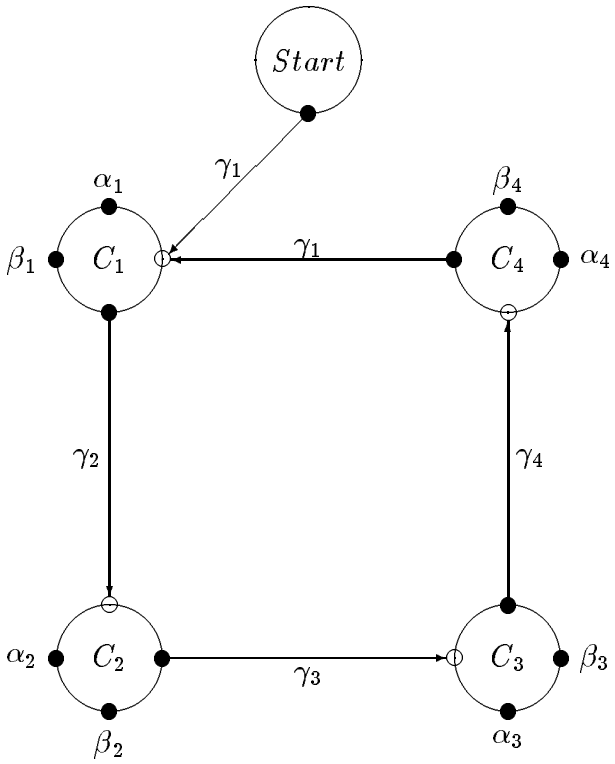
$$Sch \stackrel{\text{def}}{=} (Start \mid C_1 \mid \ldots \mid C_n) \setminus \Gamma \tag{6.18}$$

где

- $Start \stackrel{\text{def}}{=} \gamma_1!.\,\mathbf{0}$
- для каждого $i = 1, \ldots, n$ процесс C_i имеет вид



Потоковый граф процесса Sch в случае $n = 4$ имеет следующий вид:



Приведём неформальное пояснение функционирования процесса Sch.

Назовём процессы $C_1, \ldots, C_n$, участвующие в определении планировщика Sch, **циклерами**. Циклер C_i называется

- **выключенным**, если он находится в своём начальном состоянии, и

- **включённым**, если он находится не в начальном состоянии.

Процесс $Start$ включает первый циклер C_1 и после этого "умирает". Каждый циклер C_i отвечает за работу процесса P_i. Циклер C_i

- включает следующий циклер $C_{next(i)}$ после того, как он дал разрешение процессу P_i начать очередной сеанс работы, и
- выключается после того, как он дал разрешение процессу P_i закончить очередной сеанс работы.

Докажем, что процесс (6.18) удовлетворяет условию (6.17) (проверку условия (6.16) мы опустим).

Согласно определению процесса (6.15), условие (6.17) имеет вид

$$(Sch \,|\, (\beta_1?)^* \,|\, \ldots \,|\, (\beta_n?)^*) \setminus B \approx (\alpha_1!.\ \ldots \alpha_n!)^* \tag{6.19}$$

где $B = \{\beta_1, \ldots, \beta_n\}$.

Обозначим $Sch' \stackrel{\text{def}}{=}$ левая часть (6.19).

Докажем, что

$$Sch' \stackrel{+}{\approx} \tau.\alpha_1!.\ \ldots \alpha_n!.\ Sch' \tag{6.20}$$

Отсюда, по свойству единственности (с точностью до $\stackrel{+}{\approx}$) решения уравнения

$$X \ = \ \tau.\alpha_1!.\ \ldots \alpha_n!.\ X$$

будет следовать соотношение

$$Sch' \ \stackrel{+}{\approx} (\tau\ \alpha_1!\ \ldots \alpha_n!\ )^*$$

из которого, в свою очередь, следует (6.19).

Мы будем преобразовывать левую часть соотношения (6.20) так, чтобы получилась правая часть этого соотношения. Для этого мы будем использовать свойства 8, 11 и 12 операций на процессах, сформулированные в параграфе 3.7. Напомним эти свойства:

- $P \setminus L = P$, если $L \cap names(Act(P)) = \emptyset$
- $(P_1 \,|\, P_2) \setminus L = (P_1 \setminus L) \,|\, (P_2 \setminus L)$, если

$$L \cap names(Act(P_1) \cap \overline{Act(P_2)}) = \emptyset$$

- $(P \setminus L_1) \setminus L_2 = P \setminus (L_1 \cup L_2) = (P \setminus L_2) \setminus L_1$

Используя данные свойства, можно преобразовать левую часть соотношения (6.20) следующим образом.

$$\begin{array}{l} Sch' = \\ = (Sch \,|\, (\beta_1?)^* \,|\, \ldots \,|\, (\beta_n?)^*) \setminus B = \\ = \left(\begin{array}{l} ((Start \,|\, C_1 \,|\, \ldots \,|\, C_n) \setminus \Gamma) \,| \\ \quad |\, (\beta_1?)^* \,|\, \ldots \,|\, (\beta_n?)^* \end{array} \right) \setminus B = \\ = (Start \,|\, C'_1 \,|\, \ldots \,|\, C'_n) \setminus \Gamma \end{array} \tag{6.21}$$

где

$$C'_i = (C_i \,|\, (\beta_i?)^*) \setminus \{\beta_i\}$$

Заметим, что для каждого $i = 1, \ldots, n$ имеет место соотношение

$$C'_i \overset{+}{\approx} \gamma_i?.\ \alpha_i!.\ \gamma_{next(i)}!.\ C'_i \tag{6.22}$$

Действительно, по теореме о разложении

$$\begin{array}{l} C'_i = ((\gamma_i?.\ \alpha_i!.\ \gamma_{next(i)}!.\ \beta_i!.\ C_i) \,|\, (\beta_i?)^*) \setminus \{\beta_i\} \sim \\ \sim \gamma_i?.\ \alpha_i!.\ \gamma_{next(i)}!.\ \tau.C'_i \overset{+}{\approx} \text{правая часть (6.22)} \end{array}$$

Используя данное замечание и теорему о разложении, цепочку равенств (6.21) можно продолжить следующим образом:

$$\begin{array}{l} (Start \,|\, C'_1 \,|\, C'_2 \,|\, \ldots \,|\, C'_n) \setminus \Gamma \overset{+}{\approx} \\ \overset{+}{\approx} (\underbrace{\gamma_1!.\ \mathbf{0}}_{=Start} \,|\, \underbrace{\gamma_1?.\ \alpha_1!.\ \gamma_2!.\ C'_1}_{\overset{+}{\approx} C'_1} \,|\, C'_2 \,|\, \ldots \,|\, C'_n) \setminus \Gamma \sim \\ \sim \tau.\ (\mathbf{0} \,|\, \alpha_1!.\ \gamma_2!.\ C'_1 \,|\, C'_2 \,|\, \ldots \,|\, C'_n) \setminus \Gamma = \\ = \tau.\ (\alpha_1!.\ \gamma_2!.\ C'_1 \,|\, C'_2 \,|\, \ldots \,|\, C'_n) \setminus \Gamma \sim \\ \sim \tau.\ \alpha_1!.\ (\gamma_2!.\ C'_1 \,|\, C'_2 \,|\, \ldots \,|\, C'_n) \setminus \Gamma \overset{+}{\approx} \\ \overset{+}{\approx} \tau.\ \alpha_1!.\ (\gamma_2!.\ C'_1 \,|\, \underbrace{\gamma_2?.\ \alpha_2!.\ \gamma_3!.\ C'_2}_{\overset{+}{\approx} C'_2} \,|\, \ldots \,|\, C'_n) \setminus \Gamma \sim \\ \sim \tau.\ \alpha_1!.\ \tau.\ (C'_1 \,|\, \alpha_2!.\ \gamma_3!.\ C'_2 \,|\, \ldots \,|\, C'_n) \setminus \Gamma \sim \ldots \sim \\ \sim \tau.\ \alpha_1!.\ \tau.\ \alpha_2!.\ \ldots \tau.\ \alpha_n!.\ (C'_1 \,|\, \ldots \,|\, \gamma_1!.\ C'_n) \setminus \Gamma \overset{+}{\approx} \\ \overset{+}{\approx} \tau.\ \alpha_1!.\ \ldots \alpha_n!.\ (C'_1 \,|\, \ldots \,|\, \gamma_1!.\ C'_n) \setminus \Gamma \overset{+}{\approx} \\ \overset{+}{\approx} \tau.\ \alpha_1!.\ \ldots \alpha_n!.\ (\underbrace{\gamma_1?.\ \alpha_1!.\ \gamma_2!.\ C'_1}_{\overset{+}{\approx} C'_1} \,|\, \ldots \,|\, \gamma_1!.\ C'_n\ ) \setminus \Gamma \sim \\ \sim \tau.\ \alpha_1!.\ \ldots \alpha_n!.\ \underbrace{\tau.\ (\alpha_1!.\ \gamma_2!.\ C'_1 \,|\, \ldots \,|\, C'_n) \setminus \Gamma} \end{array} \tag{6.23}$$

Выражение, подчёркнутое фигурной скобкой в последней строке данной цепочки, совпадает с выражением в четвёртой строке этой цепочки, которое, в свою очередь, находится в отношении $\overset{+}{\approx}$ с Sch'.

Мы получили, что последнее выражение в цепочке (6.23)

- находится в отношении $\overset{+}{\approx}$ с правой частью соотношения (6.20), и
- с другой стороны, данное выражение находится в отношении $\overset{+}{\approx}$ с левой частью соотношения (6.20)

Таким образом, соотношение (6.20) доказано. ■

Читателю предоставляется в качестве упражнения доказать

- истинность условия (6.16), и
- соотношение $Sch \approx Sch_0$.

Читателю предлагается самостоятельно определить и доказать корректность планировщика, управляющего совокупностью P_1, ..., P_n **процессов с приоритетами**, в которой каждому процессу P_i сопоставлен некоторый **приоритет**, представляющий собой число $p_i \in [0, 1]$, причём $\sum_{i=1}^{n} p_i = 1$.

Планировщик должен реализовать такой режим работы процессов P_1, ..., P_n, при котором

- для каждого $i = 1, \ldots, n$ доля количества сеансов, выполненных процессом P_i, относительно общего количества сеансов, выполненных всеми процессами $P_1, \ldots, P_n$, асимптотически стремилась бы к p_i, при неограниченном увеличении времени работы процессов P_1, ..., P_n, причём
- данный планировщик должен обеспечивать максимальную свободу функционирования процессов $P_1, \ldots, P_n$.

6.5 Семафор

В этом примере, как и в примере из предыдущего параграфа, предполагается, что заданы n процессов

$$P_1, \ldots, P_n \tag{6.24}$$

причём для каждого $i = 1, \ldots, n$ процесс P_i имеет вид

$$P_i \;=\; (\alpha_i?\; a_{i1}\; \ldots\; a_{ik_i}\; \beta_i?)^*$$

где

- $\alpha_i?$ и $\beta_i?$ – служебные действия, представляющие собой сигналы о начале и о конце очередного сеанса работы процесса P_i, и
- $a_{i1}, \ldots, a_{ik_i}$ – собственные действия процесса P_i.

Мы хотели бы создать такой процесс P, в котором все процессы $P_1, \ldots, P_n$ работали бы совместно, причём эта совместная работа должна подчиняться следующему режиму:

- если в некоторый момент времени какой-либо из процессов P_i начал очередной сеанс своей работы исполнением действия $\alpha_i?$,
- то этот сеанс должен быть **непрерываемым**, т.е. все последующие действия процесса P должны быть действиями процесса P_i, до тех пор, пока P_i не завершит этот сеанс.

Данное требование можно выразить в терминах трасс: каждая трасса процесса P должна иметь вид

$$\alpha_i?\ a_{i1}\ \ldots\ a_{ik_i}\ \beta_i?\ \alpha_j?\ a_{j1}\ \ldots\ a_{jk_j}\ \beta_i?\ \ldots$$

т.е. каждая трасса tr процесса P должна представлять собой конкатенацию трасс

$$tr_1\ \cdot\ tr_2\ \cdot\ tr_3\ \ldots$$

каждая из которых представляет собой сеанс работы какого-либо процесса из (6.24).

Искомый процесс P мы определим следующим образом:

$$P :=\ (\ P_1[f_1]\ \mid \ldots \mid\ P_n[f_n]\ \mid\ Sem\ )\setminus\{\pi,\varphi\}$$

где

- Sem – процесс, предназначенный для задания требуемого режима работы процессов $P_1, \ldots, P_n$, данный процесс называется **семафором** и имеет вид
$$Sem\ =\ (\ \pi!\ \varphi!\ )^*$$
- $f_i : \alpha_i \mapsto \pi,\ \beta_i \mapsto \varphi$

Спецификация процесса P представляется следующим соотношением:

$$\begin{array}{rcl} P & \overset{+}{\approx} & \tau.a_{11}.\ldots.a_{1k_1}.\,P\ +\ldots+ \\ & + & \tau.a_{n1}.\ldots.a_{nk_n}.\,P \end{array} \qquad (6.25)$$

Доказательство того, что процесс P удовлетворяет этой спецификации, производится при помощи теоремы о разложении:

$$
\begin{aligned}
P \;=\; & (\; P_1[f_1] \mid \ldots \mid P_n[f_n] \mid Sem \;) \setminus \{\pi, \varphi\} \sim \\
\sim\; & \left(\begin{array}{l} \pi?.a_{11}.\ldots.a_{1k_1}.\varphi?.P_1[f_1] \mid \ldots \mid \\ \mid \pi?.a_{n1}.\ldots.a_{nk_n}\varphi?.P_n[f_n] \mid \\ \mid \pi!.\,\varphi!.\,Sem \end{array} \right) \setminus \{\pi, \varphi\} \sim \\
\sim\; & \tau.\left(\begin{array}{l} a_{11}.\ldots.a_{1k_1}.\varphi?.P_1[f_1] \mid \ldots \mid \\ \mid \pi?.a_{n1}.\ldots.a_{nk_n}\varphi?.P_n[f_n] \mid \\ \mid \varphi!.\,Sem \end{array} \right) \setminus \{\pi, \varphi\} + \\
& + \ldots + \\
& + \tau.\left(\begin{array}{l} \pi?.a_{11}.\ldots.a_{1k_1}.\varphi?.P_1[f_1] \mid \ldots \mid \\ \mid a_{n1}.\ldots.a_{nk_n}\varphi?.P_n[f_n] \mid \\ \mid \varphi!.\,Sem \end{array} \right) \setminus \{\pi, \varphi\} \sim \\
\sim\; & \ldots \sim \\
\sim\; & \tau.a_{11}.\ldots.a_{1k_1}.\tau.\; P \;+\ldots+\; \tau.a_{n1}.\ldots.a_{nk_n}.\tau.\; P \overset{+}{\approx} \\
\overset{+}{\approx}\; & \tau.a_{11}.\ldots.a_{1k_1}.\; P \;+\ldots+\; \tau.a_{n1}.\ldots.a_{nk_n}.\; P \;\blacksquare
\end{aligned}
$$

В заключение обратим внимание на следующий момент. Наличие префикса "τ." в каждом слагаемом в правой части соотношения (6.25) означает, что выбор альтернативы в начальный момент функционирования процесса P определяется

- не в окружающей среде процесса P,
- а внутри процесса P.

Если бы этого префикса не было, то это означало бы, что выбор альтернативы в начальный момент функционирования процесса P определяется окружающей средой процесса P.

Глава 7

Процессы с передачей сообщений

7.1 Действия с передачей сообщений

Введённое и изученное в предыдущих главах понятие процесса допускает различные обобщения.

Одно из таких обобщений заключается в добавлении к действиям из Act некоторых **параметров**, т.е. рассматриваются такие процессы, у которых выполняемые действия имеют вид

$$(a, p)$$

где $a \in Act$, и p – параметр, который может иметь, например следующий смысл:

- сложность (или стоимость) выполнения действия a
- приоритет действия a по отношению к другим действиям
- момент времени, в который произошло действие a
- длительность действия a
- вероятность совершения действия a
- или что-либо другое.

В настоящей главе мы рассматриваем один из вариантов такого обобщения, который связан с добавлением к действиям из Act параметров, представляющих собой **сообщения**, передаваемые при выполнении этих действий.

Напомним, что, согласно нашей неформальной интерпретации понятия действия,

- выполнение процессом действия вида $\alpha!$ заключается в передаче другому процессу объекта с именем α, и
- выполнение процессом действия вида $\alpha?$ заключается в получении от другого процесса объекта с именем α.

Обобщим данную интерпретацию следующим образом. Будем считать, что к объектам, которыми обмениваются процессы, можно добавлять **сообщения**, т.е. действия процессов могут иметь вид

$$\alpha!v \quad \text{и} \quad \alpha?v \tag{7.1}$$

где $\alpha \in Names$, и v – **сообщение**, которое может представлять собой

- число,
- символьную строку,
- банкноту,
- материальный ресурс,
- и т.п.

Выполняя действие вида $\alpha!v$ или $\alpha?v$, процесс передаёт или получает вместе с объектом α сообщение v.

Напомним, что понятие передаваемого объекта, как и понятия приёма и передачи, могут иметь виртуальный характер (более подробно см. параграф 2.3).

Для формального описания процессов, которые могут выполнять действия вида (7.1), мы обобщим понятие процесса.

7.2 Вспомогательные понятия

7.2.1 Типы, переменные, значения и константы

Мы будем предполагать, что задано множество $Types$ **типов**, причём каждому типу $t \in Types$ сопоставлено множество D_t **значений** типа t.

Типы могут обозначаться идентификаторами. Для наиболее часто используемых типов существуют общепринятые идентификаторы, например,

- тип целых чисел обозначается `int`
- тип булевых значений (0 и 1) обозначается `bool`
- тип "символы" обозначается `char`

- тип "символьные строки" обозначается `string`

Также мы будем предполагать, что заданы следующие множества.

- Множество Var, элементы которого называются **переменными**, причём каждой переменной $x \in Var$ сопоставлен тип $t(x) \in Types$.

  Каждая переменная $x \in Var$ может принимать **значения** в множестве $D_{t(x)}$, т.е. в различные моменты времени переменная x может быть связана с различными элементами множества $D_{t(x)}$.

- Множество Con, элементы которого называются **константами**. Каждой константе $c \in Con$ сопоставлены

  - тип $t(c) \in Types$, и
  - значение $[\![c]\!] \in D_{t(c)}$, называемое **интерпретацией** константы c.

7.2.2 Функциональные символы

Мы будем предполагать, что задано множество **функциональных символов (ФС)**, причём каждому ФС f сопоставлены

- **функциональный тип** $t(f)$ (называемый ниже просто **типом**), представляющий собой знакосочетание

$$(t_1, \ldots, t_n) \to t \tag{7.2}$$

  где $t_1, \ldots, t_n, t \in Types$, и

- частичная функция

$$[\![f]\!] : D_{t_1} \times \ldots \times D_{t_n} \to D_t$$

  называемая **интерпретацией** ФС f.

Например, ниже мы будем рассматривать следующие ФС:

$$+, \quad -, \quad \cdot, \quad \textbf{head}, \quad \textbf{tail}, \quad [\,]$$

где

- ФС $+$ и $-$ имеют тип

$$(\texttt{int}, \texttt{int}) \to \texttt{int}$$

  функции $[\![+]\!]$ и $[\![-]\!]$ представляют собой соответствующие арифметические операции

- ФС $\cdot$ имеет тип

$$(\texttt{string}, \texttt{string}) \to \texttt{string}$$

функция $[\![\cdot]\!]$ сопоставляет каждой паре строк (u, v) строку, получаемую приписыванием v к u справа (т.е. конкатенацию u и v).

- ФС **head** имеет тип

$$\texttt{string} \to \texttt{char}$$

функция $[\![\mathbf{head}]\!]$ сопоставляет каждой непустой строке её первый символ
(значение функции $[\![\mathbf{head}]\!]$ на пустой строке не определено)

- ФС **tail** имеет тип

$$\texttt{string} \to \texttt{string}$$

функция $[\![\mathbf{tail}]\!]$ сопоставляет каждой непустой строке u строку, получаемую из u удалением её первого символа
(значение функции $[\![\mathbf{tail}]\!]$ на пустой строке не определено)

- ФС [ ] имеет тип

$$\texttt{char} \to \texttt{string}$$

функция $[\![\ [\ ]\ ]\!]$ сопоставляет каждому символу строку, состоящую из одного этого символа

- ФС **length** имеет тип

$$\texttt{string} \to \texttt{int}$$

функция $[\![\mathbf{length}]\!]$ сопоставляет каждой строке её длину (т.е. количество символов в этой строке).

7.2.3 Выражения

Выражения строятся стандартным образом из переменных, констант и ФС. Каждое выражение e имеет тип $t(e) \in Types$, определяемый структурой этого выражения.

Правила построения выражений имеют следующий вид.

- Каждая переменная или константа является выражением того типа, который сопоставлен этой переменной или константе.

- Если
  - f – ФС, имеющий тип (7.2), и
  - $e_1, \ldots, e_n$ – выражения типов $t_1, \ldots, t_n$ соответственно

  то знакосочетание $f(e_1, \ldots, e_n)$ является выражением типа t.

Если каждой переменной x, входящей в выражение e, сопоставлено значение $\xi(x)$, то выражению e можно сопоставить значение, обозначаемое знакосочетанием $\xi(e)$, и определяемое стандартным образом:

- если $e = x$ (переменная), то $\xi(e) \stackrel{\text{def}}{=} \xi(x)$
  (значение $\xi(x)$ предполагается заданным)
- если $e = c$ (константа), то $\xi(e) \stackrel{\text{def}}{=} [\![c]\!]$
- если $e = f(e_1, \ldots, e_n)$, то значение $\xi(e)$ выражения e определено, если
  - все значения $\xi(e_1), \ldots, \xi(e_n)$ определены, и
  - значение функции $[\![f]\!]$ определено на списке

    $$(\xi(e_1), \ldots, \xi(e_n))$$

  в этом случае

  $$\xi(e) \stackrel{\text{def}}{=} [\![f]\!](\xi(e_1), \ldots, \xi(e_n))$$

Ниже мы будем использовать следующие обозначения.

- Знакосочетание $Expr$ обозначает совокупность всех выражений.
- Знакосочетание Fm обозначает совокупность тех выражений, которые имеют тип `bool`.

  Выражения из Fm называются **формулами**.

  При построении формул могут использоваться обычные булевские связки ($\wedge, \vee, \rightarrow$ и т.д.), интерпретируемые стандартным образом.

  Символ $\top$ обозначает тождественно истинную формулу, а символ $\bot$ – тождественно ложную формулу.

  Формулы вида $\wedge(b_1, b_2)$, $\vee(b_1, b_2)$, и т.п. мы будем записывать в более привычном виде $b_1 \wedge b_2$, $b_1 \vee b_2$, и т.д.

  В некоторых случаях формулы вида

  $$b_1 \wedge \ldots \wedge b_n \quad \text{и} \quad b_1 \vee \ldots \vee b_n$$

  будут записываться в виде

  $$\left\{ \begin{array}{c} b_1 \\ \ldots \\ b_n \end{array} \right\} \quad \text{и} \quad \left[\begin{array}{c} b_1 \\ \ldots \\ b_n \end{array} \right]$$

  соответственно.

- Выражения вида $+(e_1, e_2)$, $-(e_1, e_2)$ и $\cdot(e_1, e_2)$ будут записываться в более привычном виде $e_1 + e_2$, $e_1 - e_2$ и $e_1 \cdot e_2$.
- Выражения вида $\mathbf{head}(e)$, $\mathbf{tail}(e)$, $[\,](e)$, и $\mathbf{length}(e)$ будут записываться в виде $\hat{e}$, e', $[e]$ и $|e|$ соответственно.
- Костанта, интерпретацией которой является пустая строка, будет обозначаться символом ε.

7.3 Понятие процесса с передачей сообщений

В этом параграфе мы излагаем понятие процесса с передачей сообщений. Данное понятие получается из исходного понятия процесса, изложенного в параграфе 2.4, следующей модификацией.

- К числу компонентов процесса добавляются
  - компонента X_P, называемая **множеством переменных** этого процесса, и
  - компонента I_P, называемая **начальным условием**.
- Метки переходов представляют собой не действия, а **операторы**.

Прежде чем излагать формальное определение понятия процесса с передачей сообщений, мы объясним смысл вышеперечисленных понятий.

Для краткости, мы будем в этой главе называть процессы с передачей сообщений просто **процессами**.

7.3.1 Множество переменных процесса

Мы будем предполагать, что с каждым процессом P связано множество переменных

$$X_P \subseteq Var$$

В каждый момент времени i работы процесса P ($i = 0, 1, 2, \ldots$) каждой переменной $x \in X_P$ сопоставлено **значение** $\xi_i(x) \in D_{t(x)}$. Значения переменных могут изменяться во время работы процесса.

Означиванием переменных из X_P называется произвольный набор ξ значений, сопоставленных этим переменным, т.е. каждое означивание ξ имеет вид

$$\xi = \{\xi(x) \in D_{t(x)} \mid x \in X_P\}$$

Таким образом, в каждый момент времени i работы процесса P определено некоторое означивание ξ_i переменных из X_P.

Для каждого процесса P знакосочетание $Eval(X_P)$ обозначает совокупность всевозможных означиваний переменных из X_P.

Ниже мы будем предполагать, что для каждого процесса P все выражения, относящиеся к процессу P, содержат переменные только из множества X_P.

7.3.2 Начальное условие

Другой новой компонентой процесса P является формула $I_P \in Fm$, называемая **начальным условием**. Данная формула выражает условие на означивание ξ_0 переменных процесса P в начальный момент его работы: ξ_0 должно удовлетворять условию

$$\xi_0(I_P) = 1$$

7.3.3 Операторы

Главное отличие нового определения понятия процесса от старого заключается в том, что

- в старом определении метка каждого перехода является **действием**, которое совершает процесс при выполнении этого перехода, а
- в новом определении метка каждого перехода является **оператором**, который представляет собой **схему действия**, приобретающую вид конкретного действия лишь при конкретном выполнении этого оператора.

В определении понятия оператора мы будем использовать то же самое множество $Names$, которое было введено в параграфе 2.3.

Обозначим символом $\mathcal{O}$ множество, элементы которого называются **операторами**, и подразделяются на следующие четыре класса.

1. **Операторы ввода**, которые представляют собой знакосочетания вида

$$\alpha\,?\,x \tag{7.3}$$

где $\alpha \in Names$ и $x \in Var$.

Действие, соответствующее оператору (7.3), выполняется путём ввода в процесс объекта, который имеет

- имя α, и
- дополнительный параметр, представляющий собой сообщение.

Введённое сообщение v записывается в переменную x, т.е. после исполнения данного действия значение переменной x становится равным v.

2. **Операторы вывода**, которые представляют собой знакосочетания вида

$$\alpha\,!\,e \tag{7.4}$$

где $\alpha \in Names$ и $e \in Expr$.

Действие, соответствующее оператору (7.4), исполняется путём вывода из процесса объекта, который имеет

- имя α, и
- дополнительный параметр, представляющий собой сообщение вида v, которое равно значению выражения e на текущих значениях переменных процесса.

3. Операторы **присваивания** (первый вид внутренних операторов), которые представляют собой знакосочетания вида

$$x \;:=\; e \tag{7.5}$$

где

- $x \in Var$, и
- $e \in Expr$, причём $t(e) = t(x)$

Действие, соответствующее оператору (7.5), исполняется путём обновления значения переменной x: после исполнения этого оператора её значение становится равным значению выражения e на текущих значениях переменных процесса P

4. Операторы **проверки условия** (второй вид внутренних операторов), которые представляют собой знакосочетания вида

$$b\,?$$

где $b \in Fm$.

Действие, соответствующее такому оператору, исполняется путём вычисления значения формулы b на текущих значениях переменных процесса P, и

- если оно равно 0, то выполнение всего действия считается невозможным,
- иначе - выполнение действия считается завершённым.

7.3.4 Определение процесса

Процессом называется пятёрка P вида

$$P = (X_P, I_P, S_P, s_P^0, R_P) \tag{7.6}$$

компоненты которой имеют следующий смысл:

1. X_P – множество переменных процесса P
2. I_P – формула, называемая **начальным условием** процесса P
3. S_P – множество **состояний** процесса P
4. $s_P^0 \in S_P$ – **начальное состояние**
5. R_P – подмножество вида

$$R_P \subseteq S_P \times \mathcal{O} \times S_P$$

Элементы множества R_P называются **переходами**.

Если переход из R_P имеет вид (s_1, op, s_2), то мы будем обозначать его знакосочетанием

$$s_1 \xrightarrow{op} s_2$$

и говорить, что

- состояние s_1 является **началом** этого перехода,
- состояние s_2 – его **концом**,
- оператор op является **меткой** этого перехода.

Также мы будем предполагать, что для каждого процесса P множество его переменных X_P содержит специальную переменную at_P, множеством значений которой является множество S_P состояний процесса P.

7.3.5 Функционирование процесса

Функционирование процесса P заключается в обходе множества его состояний (начиная с начального состояния s_P^0), с выполнением операторов, являющихся метками проходимых переходов.

Более подробно: на каждом шаге функционирования $i \geq 0$

- процесс находится в некотором состоянии s_i
  $(s_0 = s_P^0)$

- определено некоторое означивание ξ_i переменных процесса P ($\xi_0(I_P)$ должно быть равно 1)
- если есть хотя бы один переход из R_P с началом в s_i, то процесс
  - недетерминированно выбирает переход с началом в s_i, помеченный таким оператором op_i, который можно выполнить в текущий момент времени,
    (если таких переходов нет, то процесс временно приостанавливает свою работу до того момента, когда появится хотя бы один такой переход)
  - выполняет оператор op_i, и после этого
  - переходит в состояние s_{i+1}, которое является концом выбранного перехода
- если в R_P нет переходов с началом в s_i, то процесс заканчивает свою работу.

Для каждого $i \geq 0$ означивание ξ_{i+1} определяется

- по означиванию ξ_i, и
- по оператору op_i, который исполняется на i–м шаге функционирования процесса P.

Связь между означиваниями ξ_i, ξ_{i+1} и оператором op_i имеет следующий вид:

1. если $op_i = \alpha\,?\,x$, и при выполнении этого оператора в процесс было введено сообщение v, то

$$\begin{array}{l} \xi_{i+1}(x) = v \\ \forall y \in X_P \setminus \{x, at_P\} \qquad \xi_{i+1}(y) = \xi_i(y) \end{array}$$

2. если $op_i = \alpha\,!\,e$, то при выполнении этого оператора процесс выводит сообщение

$$\xi_i(e)$$

   а значения переменных из $X_P \setminus \{at_P\}$ не изменяются:

$$\forall x \in X_P \setminus \{at_P\} \qquad \xi_{i+1}(x) = \xi_i(x)$$

3. если $op_i = (x := e)$, то

$$\begin{array}{l} \xi_{i+1}(x) = \xi_i(e) \\ \forall x \in X_P \setminus \{x, at_P\} \qquad \xi_{i+1}(x) = \xi_i(x) \end{array}$$

4. если $op_i = b?$ и $\xi_i(b) = 1$, то

$$\forall x \in X_P \setminus \{at_P\} \qquad \xi_{i+1}(x) = \xi_i(x)$$

Мы будем предполагать, что для каждого $i \geq 0$ значение переменной at_P при означивании ξ_i равно тому состоянию $s \in S_P$, в котором процесс P находится в момент времени i, т.е.

- $\xi_0(at_P) = s_P^0$
- $\xi_1(at_P) = s_1$, где s_1 – конец первого перехода
- $\xi_2(at_P) = s_2$, где s_2 – конец второго перехода
- и т.д.

7.4 Представление процессов в виде блок-схем

В целях повышения наглядности, процессы иногда изображают в виде **блок-схем**.

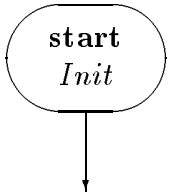
Отметим, что язык блок-схем возник в программировании, где использование этого языка позволяет существенно облегчить описание и понимание функционирования алгоритмов и программ.

7.4.1 Понятие блок-схемы

Блок-схема представляет собой ориентированный граф, каждому узлу n которого сопоставлен **оператор** $op(n)$ одного из перечисляемых ниже видов.

Каждый узел n блок-схемы изображается в виде геометрической фигуры (прямоугольника, овала или кружочка). Если $op(n)$ не является оператором выбора или оператором соединения, то он изображается внутри этой фигуры.

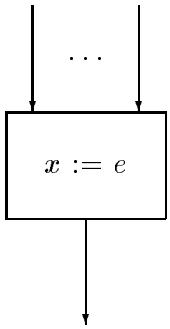
оператор начала:



(7.7)

где $Init$ – формула, называемая **начальным условием**.

оператор присваивания:

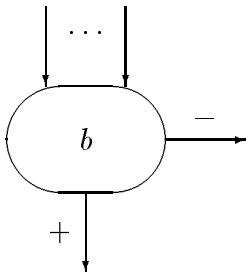


(7.8)

где

- $x \in Var$,
- $e \in Expr$, причём $t(e) = t(x)$

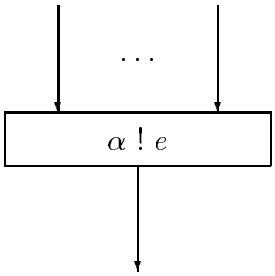
оператор условного перехода:



(7.9)

где $b \in Fm$.

оператор посылки сообщения:

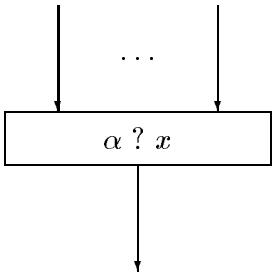


(7.10)

где

- α – имя (например, имя процесса, которому посылается сообщение), и
- e – выражение, значенем которого является посылаемое сообщение.

оператор получения сообщения:

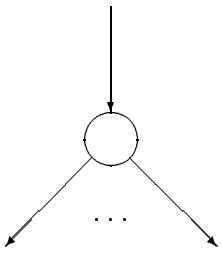


(7.11)

где

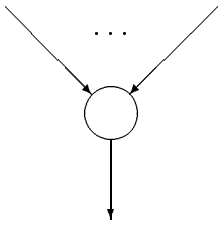
- α – имя (например, имя процесса, от которого приходит сообщение), и
- x – переменная, в которую следует записать получаемое сообщение.

оператор выбора:



(7.12)

оператор соединения:

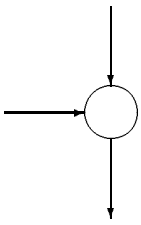


(7.13)

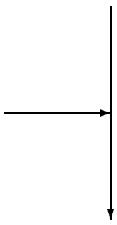
Иногда

- кружочек, изображающий оператор соединения, не рисуют,
- и также не рисуют концы некоторых стрелок, ведущих в этот оператор

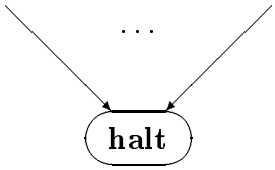
т.е., например, фрагмент блок-схемы вида



может быть изображён следующим образом:



оператор остановки:



(7.14)

Блок-схемы должны удовлетворять следующим условиям:

- узел вида (7.7) может быть только один
- из узлов вида (7.7), (7.8), (7.10), (7.11), (7.13) выходит только одно ребро
- из узлов вида (7.9) выходят одно или два ребра, причём
  - если из узла вида (7.9) выходит одно ребро, оно имеет метку “+”, и
  - если из узла вида (7.9) выходят два ребра, то одно из них имеет метку “+”, а другое – метку “−”
- в узлы вида (7.12) входит только одно ребро
- из узлов вида (7.14) не выходит ни одного ребра

7.4.2 Функционирование блок-схемы

Функционирование блок-схемы представляет собой последовательность переходов от одного узла к другому по рёбрам, начиная с узла n_0 вида (7.7) (называемого **начальным узлом**), с выполнением операторов, сопоставленных проходимым узлам.

Более подробно: каждый шаг функционирования $i \geq 0$ связан с некоторым узлом n_i, который называется **текущим узлом**, и

- если n_i не имеет вид (7.14), то после выполнения оператора, соответствующего узлу n_i, происходит переход по ребру, выходящему из n_i, к следующему узлу n_{i+1}, который будет текущим узлом на следующем шаге функционирования
- если же n_i имеет вид (7.14), то функционирование блок-схемы завершается.

Обозначим символом X совокупность всех переменных, входящих в блок-схему. На каждом шаге i функционирования ($i = 0, 1, \ldots$) каждой переменной $x \in X$ сопоставлено значение $\xi_i(x)$. Совокупность значений переменных

$$\{\xi_i(x) \mid x \in X\}$$

обозначается символом ξ_i и называется **означиванием** переменных из X на i–м шаге функционирования блок-схемы. Значения переменных в начальный момент времени должны удовлетворять начальному условию, т.е. должно быть верно соотношение

$$\xi_0(Init) = 1$$

Операторы выполняются следующим образом.

- Оператор (7.8)
  - вычисляет значение выражения e на текущем означивании ξ_i переменных из X, и
  - заносит это значение в переменную x

  т.е.

$$\xi_{i+1}(x) \stackrel{\text{def}}{=} \xi_i(e)$$
$$\forall\, y \in X \setminus \{x\} \quad \xi_{i+1}(y) \stackrel{\text{def}}{=} \xi_i(y)$$

- Оператор (7.9) вычисляет значение формулы b на текущем означивании ξ_i переменных из X, и
  - если $\xi_i(b) = 1$, то происходит переход к следующему узлу по ребру с меткой “+”,

– иначе -

* если из текущего узла выходят два ребра, то происходит переход к следующему узлу по ребру с меткой “–”, и
* если из текущего узла выходит одно ребро, то в данный момент времени выполнение оператора, соответствующего текущему узлу, считается невозможным.

- Оператор (7.10) может выполниться в текущий момент времени только в том случае, когда в этот момент времени процесс может послать объект с именем α.

  Если это возможно, то выполняется посылка

  $$\alpha\,!\,\xi_i(e)$$

- Оператор (7.11) может выполниться в текущий момент времени только в том случае, когда в этот момент времени процесс может принять объект с именем α.

  Если это возможно, то

  – этот объект принимается,

  – сообщение v, содержащееся в этом объекте, записывается в переменную x, т.е.

  $$\begin{array}{l}\xi_{i+1}(x) \stackrel{\text{def}}{=} v \\ \forall y \in X \setminus \{y\} \quad \xi_{i+1}(y) \stackrel{\text{def}}{=} \xi_i(y)\end{array}$$

- Если текущий узел помечен оператором (7.12), то

  – из рёбер, которые из него выходят, выбирается ребро, ведущее в узел, помеченный таким оператором, который возможно выполнить в текущий момент времени, и

  – происходит переход в этот узел.

  (если таких операторов, которые возможно выполнить в текущий момент времени, несколько, то выбор производится недетерминированно)

- Оператор (7.14) завершает функционирование блок-схемы.

7.4.3 Построение процесса, определяемого блок-схемой

Алгоритм построения процесса по блок-схеме имеет следующий вид.

1. На каждом ребре блок-схемы рисуется точка.

2. Для

   - каждого узла n блок-схемы, который не имеет вида (7.12) или (7.13), и
   - каждой пары F_1, F_2 рёбер блок-схемы, таких что F_1 входит в n, а F_2 - выходит из n

   выполняются следующие действия:

   (a) рисуется стрелка f, соединяющая точку на F_1 с точкой на F_2,

   (b) на этой стрелке f рисуется метка $\langle f \rangle$, определяемая следующим образом:

      i. если $op(n)$ имеет вид (7.8), то

      $$\langle f \rangle \stackrel{\text{def}}{=} (x := e)$$

      ii. если $op(n)$ имеет вид (7.9), и ребро, выходящее из n, помечено символом "+", то

      $$\langle f \rangle \stackrel{\text{def}}{=} b\,?$$

      iii. если $op(n)$ имеет вид (7.9), и ребро, выходящее из n, помечено символом "−", то

      $$\langle f \rangle \stackrel{\text{def}}{=} \neg b\,?$$

      iv. если $op(n)$ имеет вид (7.10) или (7.11), то $\langle f \rangle$ совпадает с $op(n)$.

3. Для каждого узла n вида (7.12) и каждого ребра F, выходящего из n, выполняются следующие действия:

   - стрелка, соответствующая тому узлу, в который входит F, заменяется на стрелку
     - с тем же концом и с той же меткой,
     - но с началом – на ребре, входящем в n
   - точка на ребре F удаляется.

4. Для каждого узла n вида (7.13) и каждого ребра F, входящего в n, выполняются следующие действия:

   - стрелка, соответствующая тому узлу, из которого выходит F, заменяется на стрелку
     - с тем же началом и с той же меткой,
     - но с концом – на ребре, выходящем из n

- точка на ребре F удаляется.

5. Состояниями искомого процесса являются оставшиеся нарисованные точки.

6. Начальное состояние s_P^0 определяется следующим образом.

   - Если точка, нарисованная на том ребре блок-схемы, которое выходит из её начального узла, не была удалена, то эта точка является начальным состоянием s_P^0.
   - Если же эта точка была удалена, то нетрудно заметить, что это могло произойти только в том случае, когда концом ребра, выходящего из начального узла блок-схемы, является узел n вида (7.13). В этом случае начальным состоянием s_P^0 является точка, нарисованная на ребре, выходящем из n.

7. Переходы процесса соответствуют нарисованным стрелкам: для каждой такой стрелки f процесс содержит переход

$$s_1 \xrightarrow{\langle f \rangle} s_2$$

   где s_1 и s_2 – начало и конец стрелки f соответственно.

8. Множество переменных процесса содержит

   - все переменные, входящие в какой-либо из операторов блок-схемы,
   - а также переменную at_P.

9. Начальное условие процесса совпадает с начальным условием $Init$ блок-схемы.

7.5 Пример процесса с передачей сообщений

В этом параграфе мы рассмотрим в качестве примера процесс “буфер”:

- сначала мы определим этот процесс в виде блок-схемы, и
- затем мы построим по этой блок-схеме представление процесса “буфер” в стандартной форме.

7.5.1 Понятие буфера

Пусть n – некоторое положительное целое число.

Под **буфером размера** n мы в этом параграфе будем понимать систему (называемую ниже просто **буфером**), которая обладает следующими свойствами.

- В буфер можно вводить символы. Символы, введённые в буфер, можно выводить из буфера.

  Если некоторый символ c введён в буфер, то мы будем говорить, что символ c **содержится** в буфере (до тех пор пока этот символ не будет выведен из буфера).

  В буфере может одновременно содержаться не более n символов.

- В каждый момент времени совокупность символов, содержащихся в буфере, представляет собой упорядоченную последовательность

$$c_1, \ldots, c_k \qquad (0 \leq k \leq n) \tag{7.15}$$

  которая называется **содержимым буфера**. Число k называется **размером** содержимого буфера.

  Случай $k = 0$ соответствует ситуации, когда в буфере не содержится ни одного символа, в этом случае мы будем говорить, что **буфер пуст**.

- Если в текущий момент времени содержимое буфера имеет вид (7.15), и $k < n$, то в буфер можно ввести произвольный символ c, и после выполнения этой операции содержимое буфера примет вид

$$c_1, \ldots, c_k, c$$

- Если в текущий момент времени содержимое буфера имеет вид (7.15), и $k > 0$, то из буфера можно вывести символ, расположенный в начале его содержимого (т.е. c_1), после выполнения этой операции содержимое буфера примет вид

$$c_2, \ldots, c_k$$

Таким образом, в каждый момент времени содержимое буфера представляет собой очередь символов, причём

- каждая операция ввода символа в буфер добавляет вводимый символ в конец этой очереди, и

- каждая операция вывода символа из буфера

  – выводит из буфера первый элемент этой очереди, и

– удаляет этот элемент из очереди

Очереди, операции с которыми выглядят описанным выше образом, называются **очередями типа FIFO** (First Input - First Output).

7.5.2 Представление поведения буфера в виде блок-схемы

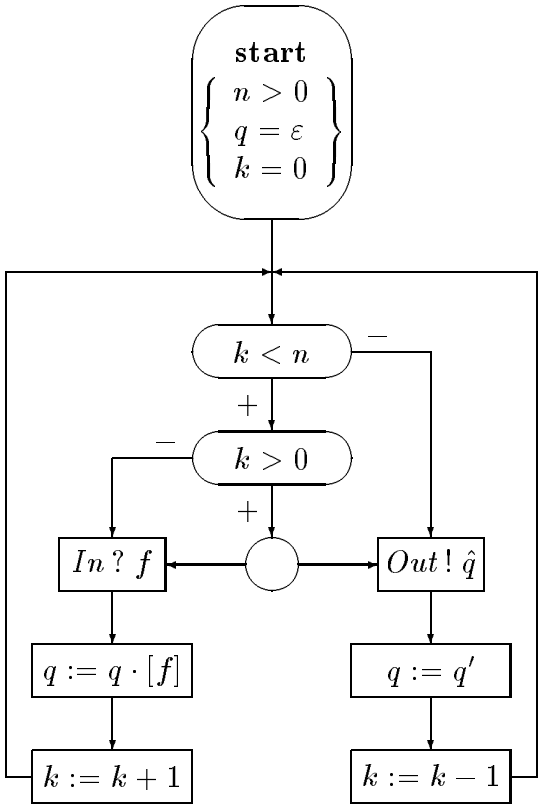
Одним из возможных формальных уточнений понятия буфера является процесс $Buffer_n$, описание которого приводится в этом параграфе в виде блок-схемы. В этом процессе

- операция ввода символа в буфер представляется действием с именем In, и
- операция вывода символа из буфера представляется действием с именем Out.

Процесс $Buffer_n$ имеет следующие переменные:

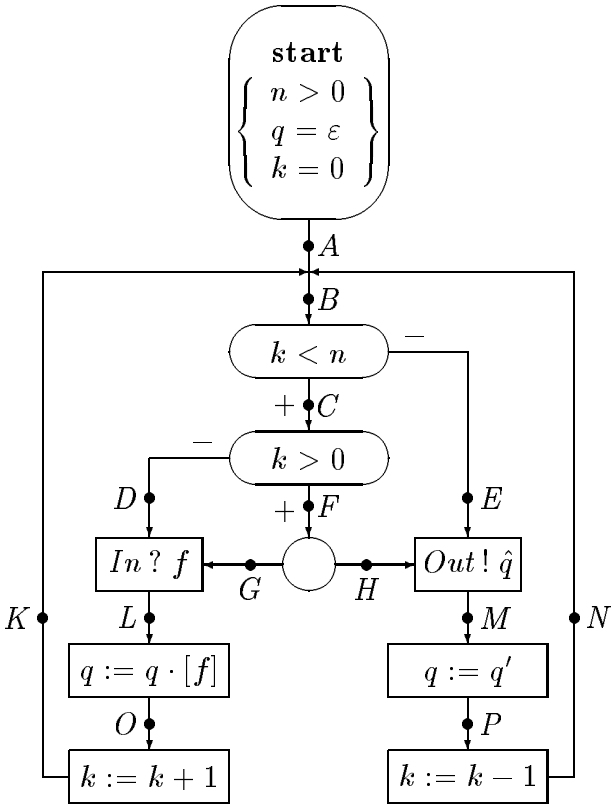
- переменная n типа `int`, её значение не изменяется, оно равно максимальному размеру содержимого буфера
- переменная k типа `int`, её значение равно размеру содержимого буфера в текущий момент времени
- переменная q типа `string`, её значение равно содержимому буфера в текущий момент времени
- переменная f типа `char`, в эту переменную будут записываться вводимые символы при выполнении операции ввода.

Блок-схема, представляющая процесс $Buffer_n$, имеет следующий вид (используемые в этой блок-схеме обозначения были определены в параграфе 7.2.3):



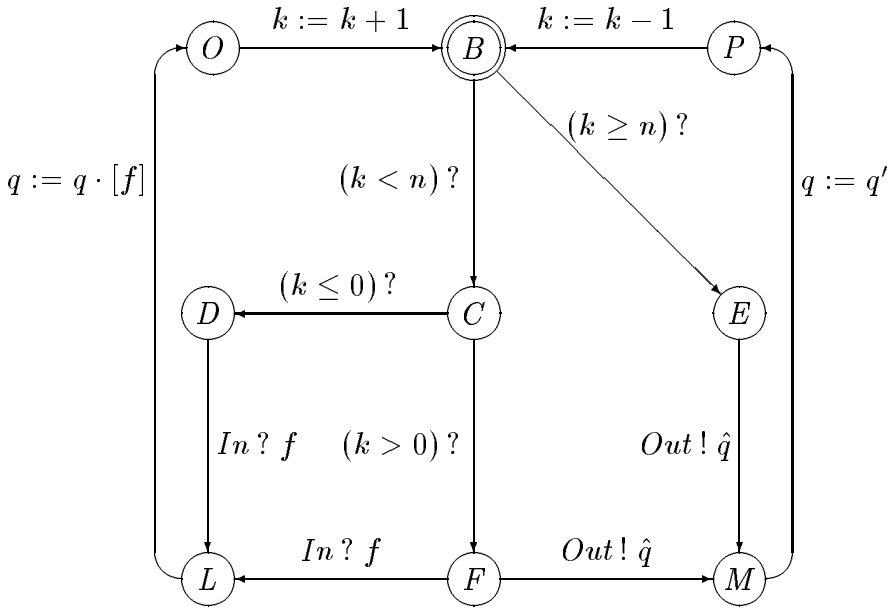
7.5.3 Представление поведения буфера в виде процесса

Для построения процесса $Buffer_n$, который соответствует определённой выше блок-схеме, мы нарисуем точки на её рёбрах:



При построении процесса по этой блок-схеме будут удалены точки A, G, H, K и N.

Искомый процесс $Buffer_n$ имеет следующий вид:



7.6 Операции на процессах с передачей сообщений

Операции на процессах с передачей сообщений аналогичны операциям на процессах в исходном смысле данного понятия (см. главу 3).

7.6.1 Префиксное действие

Пусть заданы процесс P и оператор op.

Процесс $op.P$ получается из процесса P добавлением

- к множеству его состояний – нового состояния s, которое является начальным в $op.P$,
- к множеству переходов – нового перехода с меткой op из s в s_P^0
- к множеству переменных – всех переменных, входящих в op.

7.6.2 Альтернативная композиция

Пусть процессы P_1 и P_2 таковы, что $S_{P_1} \cap S_{P_2} = \emptyset$.

Тогда можно определить процесс $P_1 + P_2$, называемый **альтернативной композицией** процессов P_1 и P_2:

- множества его состояний, переходов, и начальное состояние определяются так же, как определяются соответствующие компоненты процесса $P_1 + P_2$ в главе 3
- $X_{P_1+P_2} \stackrel{\text{def}}{=} X_{P_1} \cup X_{P_2}$
- $I_{P_1+P_2} \stackrel{\text{def}}{=} I_{P_1} \wedge I_{P_2}$

Если же $S_{P_1} \cap S_{P_2} \neq \emptyset$, то для определения процесса $P_1 + P_2$ сначала надо заменить в P_2 те состояния, которые входят также и в P_1, на новые элементы, и модифицировать соответствующим образом другие компоненты P_2.

7.6.3 Параллельная композиция

Пусть процессы P_1 и P_2 таковы, что $X_{P_1} \cap X_{P_2} = \emptyset$.

Тогда можно определить процесс $P_1 \mid P_2$, называемый **параллельной композицией** процессов P_1 и P_2:

- множество его состояний и начальное состояние определяются так же, как определяются соответствующие компоненты процесса $P_1 \mid P_2$ в главе 3

- $X_{P_1+P_2} \stackrel{\text{def}}{=} X_{P_1} \cup X_{P_2}$
- $I_{P_1+P_2} \stackrel{\text{def}}{=} I_{P_1} \wedge I_{P_2}$
- множество переходов процесса $P_1 \mid P_2$ определяется следующим образом:
  - для
    - каждого перехода $s_1 \xrightarrow{op} s_1'$ процесса P_1, и
    - каждого состояния s процесса P_2

    процесс $P_1 \mid P_2$ содержит переход
    $$(s_1, s) \xrightarrow{op} (s_1', s)$$
  - для
    - каждого перехода $s_2 \xrightarrow{op} s_2'$ процесса P_2, и
    - каждого состояния s процесса P_1

    процесс $P_1 \mid P_2$ содержит переход
    $$(s, s_2) \xrightarrow{op} (s, s_2')$$
  - для каждой пары переходов вида
    $$\begin{array}{ll} s_1 \xrightarrow{op_1} s_1' & \in R_{P_1} \\ s_2 \xrightarrow{op_2} s_2' & \in R_{P_2} \end{array}$$

    где
    - один из операторов op_1, op_2 имеет вид $\alpha\,?\,x$,
    - а другой – $\alpha\,!\,e$, причём $t(x) = t(e)$
      (имя в обоих операторах – одно и то же)

    процесс $P_1 \mid P_2$ содержит переход
    $$(s_1, s_2) \xrightarrow{x := e} (s_1', s_2')$$

Если же $X_{P_1} \cap X_{P_2} \neq \emptyset$, то для определения процесса $P_1 \mid P_2$ сначала надо заменить в одном из процессов те переменные, которые входят также и в другой процесс, на новые переменные.

7.6.4 Ограничение

Пусть заданы процесс P и подмножество $L \subseteq Names$.

Ограничением P по L называется процесс

$$P \setminus L$$

который получается из P удалением тех переходов, метки которых содержат имена из L.

7.6.5 Переименование

Пусть заданы процесс P и функция $f : Names \to Names$.

Действие операции переименования $[f]$ на процесс P заключается в изменении имён в метках переходов: каждое имя α в какой-либо из этих меток заменяется на $f(\alpha)$.

Получившийся процесс обозначается знакосочетанием $P[f]$.

Если f действует нетождественно лишь на имена из списка

$$\alpha_1, \ldots, \alpha_n$$

и отображает их в имена

$$\beta_1, \ldots, \beta_n$$

соответственно, то для $P[f]$ мы будем иногда использовать эквивалентное обозначение

$$P[\beta_1/\alpha_1, \ldots, \beta_n/\alpha_n]$$

7.7 Эквивалентность процессов

7.7.1 Понятие конкретизации процесса

Пусть P – произвольный процесс. Обозначим знакосочетанием $Conc(P)$ процесс в исходном смысле данного понятия (см. параграф 2.4), который называется **конкретизацией** процесса P, и имеет следующие компоненты.

1. Состояниями $Conc(P)$ являются

   - всевозможные означивания из $Eval(X_P)$,
   - а также дополнительное состояние s^0, которое является начальным в $Conc(P)$

2. Для

   - каждого перехода $s_1 \xrightarrow{op} s_2$ процесса P, и
   - каждого означивания $\xi \in Eval(X_P)$, такого, что $\xi(at_P) = s_1$

   $Conc(P)$ содержит переход

   $$\xi \xrightarrow{a} \xi'$$

   если $\xi'(at_P) = s_2$, и имеет место один из следующих случаев:

   - $op = \alpha\,?\,x, \quad a = \alpha\,?\,v$, где v – произвольное значение из $D_{t(x)}$
     $\xi'(x) = v, \quad \forall y \in X_P \setminus \{x, at_P\} \qquad \xi'(y) = \xi(y)$

- – $op = \alpha\,!\,e, \quad a = \alpha\,!\,\xi(e)$
  – $\forall x \in X_P \setminus \{at_P\} \qquad \xi'(x) = \xi(x)$
- – $op = \ (x := e), \quad a = \tau$
  – $\xi'(x) = \xi(e), \quad \forall y \in X_P \setminus \{x, at_P\} \qquad \xi'(y) = \xi(y)$
- – $op = \ b\,?, \quad \xi(b) = 1, \quad a = \tau$
  – $\forall x \in X_P \setminus \{at_P\} \qquad \xi'(x) = \xi(x)$

3. Для

   - каждого означивания $\xi \in Eval(X_P)$, такого, что $\xi(I_P) = 1$, и
   - каждого перехода в $Conc(P)$ вида $\xi \xrightarrow{a} \xi'$

   $Conc(P)$ содержит переход $s^0 \xrightarrow{a} \xi'$.

Из определений

- понятия функционирования процесса с передачей сообщений (см. пункт 7.3.5), и
- понятия функционирования процесса в исходном смысле (см. пункт 2.4)

следует, что имеется взаимно однозначное соответствие между

- множеством всех вариантов функционирования процесса P, и
- множеством всех вариантов функционирования его конкретизации $Conc(P)$.

Читателю предлагается самостоятельно исследовать перестановочность функции $Conc$ с операциями на процессах, т.е. установить истинность или ложность соотношений вида

$$Conc(P_1 \,|\, P_2) = Conc(P_1) \,|\, Conc(P_2)$$

и т.п.

7.7.2 Понятие эквивалентности процессов

На множестве процессов с передачей сообщений можно определить те же отношения эквивалентности, которые можно определить для процессов в исходном смысле данного понятия (см. главу 4).

Мы будем считать, что любые два процесса с передачей сообщений P_1, P_2 по определению находятся в том же самом отношении эквивалентности ($\sim$, $\approx$, $\overset{+}{\approx}$, ...), в котором находятся их конкретизации, т.е.

$$P_1 \sim P_2 \quad \Leftrightarrow \quad Conc(P_1) \sim Conc(P_2), \quad \text{и т.д.}$$

Читателю предлагается самостоятельно

- исследовать связь операций на процессах с различными отношениями эквивалентности, т.е. установить свойства, аналогичные свойствам, изложенным в параграфах 3.7, 4.5, 4.8.4, 4.9.5

- сформулировать и обосновать необходимые и достаточные условия эквивалентности ($\approx, \overset{+}{\approx}, \ldots$) процессов, не использующие понятие конкретизации процесса.

7.8 Процессы с составными операторами

7.8.1 Мотивировка понятия процесса с составными операторами

Сложность задачи анализа процесса существенно зависит от размера его описания (в частности, от количества состояний). Поэтому для построения эффективных алгоритмов анализа процессов необходим поиск методов понижения сложности описания анализируемых процессов. В этом параграфе мы рассматриваем один из таких методов.

Мы обобщаем понятие процесса до понятия процесса с составными операторами. Составной оператор является комбинацией нескольких обычных операторов. За счёт того, что мы объединяем последовательность обычных операторов в один составной, у нас появляется возможность исключить из описания процесса те состояния, в которых он находится на промежуточных шагах выполнения этой последовательности операторов.

Мы определяем понятие редукции процессов с составными операторами, с таким расчётом, чтобы при выполнении редукции получался процесс,

- имеющий менее сложное описание, и

- эквивалентный (в некотором смысле) исходному процессу.

С использованием описанных выше понятий задача анализа процесса может решаться следующим образом.

1. Сначала мы сопоставляем исходному процессу P процесс P' с составными операторами, в некотором смысле совпадающий с P (говоря неформально, мы просто рассматриваем каждый оператор, входящий в P, как составной оператор).

2. Затем мы редуцируем P', получая процесс P'', сложность которого может быть существенно меньше сложности исходного процесса P.

3. После этого мы выполняем анализ процесса P'', и по результатам этого анализа мы составляем заключение о свойствах исходного процесса P.

7.8.2 Понятие составного оператора

Составным оператором (СО) называется конечная последовательность Op операторов

$$Op = (op_1, \ldots, op_n) \qquad (n \geq 1) \tag{7.16}$$

обладающая следующими свойствами:

1. op_1 является оператором проверки условия,
   формулу, входящую в этот оператор, мы будем обозначать знакосочетанием
   $$cond\,(Op)$$
2. последовательность $(op_2, \ldots, op_n)$
   - не содержит операторов проверки условия, и
   - содержит не более одного оператора ввода или вывода.

Пусть Op – некоторый СО.

- Op называется **СО ввода** (или **вывода**), если среди операторов, входящих в Op, есть оператор ввода (или вывода).
- Op называется **внутренним**, если все операторы, входящие в Op – внутренние.
- Если Op является СО ввода или вывода, то знакосочетание
  $$name\,(Op)$$
  обозначает имя, входящее в Op.
- Если ξ – некоторое означивание переменных, входящих в $cond\,(Op)$, то мы будем говорить, что Op **открыт на** ξ, если
  $$\xi(cond\,(Op)) = 1$$

7.8.3 Понятие процесса с СО

Понятие **процесса с СО** отличается от понятия процесса из параграфа 7.3.4 только тем, что метки переходов у процесса с СО представляют собой СО.

7.8.4 Функционирование процесса с СО

Функционирование процесса с СО

- определяется почти так же, как определяется функционирование процесса в параграфе 7.3.5, и
- тоже представляет собой обход множества его состояний (начиная с начального состояния), с выполнением СО, являющихся метками проходимых переходов.

Пусть $P = (X_P, I_P, S_P, s_P^0, R_P)$ - процесс с СО.
На каждом шаге функционирования $i \geq 0$

- процесс P находится в некотором состоянии s_i $(s_0 = s_P^0)$
- определено некоторое означивание ξ_i переменных из X_P
  $(\xi_0(I_P) = 1, \quad \xi_i(at_P) = s_i)$
- если есть хотя бы один переход из R_P с началом в s_i, то процесс
  - недетерминированно выбирает переход с началом в s_i, помеченный таким СО Op_i, который обладает следующими свойствами:
    * Op_i открыт на ξ_i
    * если среди операторов, входящих в Op_i, есть оператор вида

      $$\alpha\,?\,x \quad \text{или} \quad \alpha\,!\,e$$

      то процесс P может в текущий момент времени выполнить действие вида

      $$\alpha\,?\,v \quad \text{или} \quad \alpha\,!\,v$$

      соответственно

    (если таких переходов нет, то процесс временно приостанавливает свою работу до того момента, когда появится хотя бы один такой переход)
  - выполняет последовательно все операторы, входящие в Op_i, изменяя соответствующим образом текущее означивание после выполнения каждого оператора, входящего в Op_i, и после этого
  - переходит в состояние s_{i+1}, которое является концом выбранного перехода
- если в R_P нет переходов с началом в s_i, то процесс заканчивает свою работу.

7.8.5 Операции на процессах с СО

Определения операций на процессах с СО почти совпадают с соответствующими определениями из параграфа 7.6, поэтому мы лишь укажем отличия в этих определениях.

- В определениях всех операций на процессах с СО вместо операторов участвуют СО.
- Определение операции $|$ отличается только в том пункте, в котором определяются "диагональные" переходы. Для процессов с СО данный пункт выглядит следующим образом:

  для каждой пары переходов вида

  $$\begin{array}{lll} s_1 & \xrightarrow{Op_1} & s_1' \quad \in R_{P_1} \\ s_2 & \xrightarrow{Op_2} & s_2' \quad \in R_{P_2} \end{array}$$

  где один из СО Op_1, Op_2 имеет вид

  $$(op_1,\ \ldots,\ op_i,\ \alpha\,?\,x,\ op_{i+1},\ \ldots,\ op_n)$$

  а другой –

  $$(op_1',\ \ldots,\ op_j',\ \alpha\,!\,e,\ op_{j+1}',\ \ldots,\ op_m')$$

  где

  – $t(x) = t(e)$,
  – подпоследовательности $(op_{i+1},\ \ldots,\ op_n)$ и $(op_{j+1}',\ \ldots,\ op_m')$ могут быть пустыми

  процесс $P_1 \,|\, P_2$ содержит переход

  $$(s_1, s_2) \xrightarrow{Op} (s_1', s_2')$$

  где Op имеет вид

  $$\left(\begin{array}{l} (cond(Op_1) \wedge cond(Op_2))\,?, \\ op_2,\ \ldots,\ op_i, \\ op_2',\ \ldots,\ op_j', \\ (x := e), \\ op_{i+1},\ \ldots,\ op_n, \\ op_{j+1}',\ \ldots,\ op_m' \end{array}\right)$$

7.8.6 Преобразование процессов с передачей сообщений в процессы с СО

Каждый процесс с передачей сообщений можно преобразовать в процесс с СО путём замены меток его переходов: для каждого перехода

$$s_1 \xrightarrow{op} s_2$$

его метка op заменяется на СО Op, определяемый следующим образом.

- Если op – оператор проверки условия, то

$$Op \stackrel{\text{def}}{=} (op)$$

- Если op – оператор присваивания, ввода или вывода, то

$$Op \stackrel{\text{def}}{=} (\top ?\, , op)$$

  где $\top$ – тождественно истинная формула.

Для каждого процесса с передачей сообщений P мы будем обозначать соответствующий ему процесс с СО тем же символом P.

7.8.7 Конкатенация СО

В этом параграфе мы вводим понятие **конкатенации** СО: для некоторых пар СО (Op_1, Op_2) мы определяем СО, обозначаемый знакосочетанием

$$Op_1 \cdot Op_2 \tag{7.17}$$

и называемый **конкатенацией** Op_1 и Op_2.

СО (7.17) отражает идею последовательного выполнения операторов из Op_1 и Op_2: в (7.17)

- сначала выполняются операторы, входящие в Op_1,
- а затем - операторы, входящие в Op_2.

Необходимым условием для того, чтобы можно было определить конкатенацию (7.17), является условие того, чтобы хотя бы один из СО Op_1, Op_2 был внутренним.

Ниже мы будем использовать следующие обозначения.

1. Для каждого
   - СО $Op = (op_1, \ldots, op_n)$, и
   - оператора присваивания op

знакосочетание $Op \cdot op$ обозначает СО

$$(op_1, \ldots, op_n, op) \tag{7.18}$$

2. Для каждого

   - внутреннего СО $Op = (op_1, \ldots, op_n)$, и
   - оператора ввода или вывода op

   знакосочетание $Op \cdot op$ обозначает СО (7.18)

3. Для каждого

   - СО $Op = (op_1, \ldots, op_n)$, и
   - оператора проверки условия $op = b?$

   знакосочетание $Op \cdot op$ обозначает объект, который

   - либо является СО,
   - либо не определён.

   Данный объект определяется рекурсивно следующим образом.

   Если $n = 1$, то

   $$Op \cdot op \stackrel{\text{def}}{=} ((cond\,(Op) \wedge b)\,?)$$

   иначе –

   - если op_n – оператор присваивания, и имеет вид $(x := e)$, то

     $$Op \cdot op \stackrel{\text{def}}{=} \underbrace{((op_1, \ldots, op_{n-1}) \cdot op_n(op))}_{(*)} \cdot op_n$$

     где

     – $op_n(op)$ – оператор проверки условия, получаемый из op заменой всех вхождений в него переменной x на выражение e

     – если объект $(*)$ не определён, то $Op \cdot op$ тоже не определён

   - если op_n – оператор вывода, то $Op \cdot op$ есть СО

     $$((op_1, \ldots, op_{n-1}) \cdot op) \cdot op_n \tag{7.19}$$

   - если op_n – оператор ввода, и имеет вид $\alpha\,?\,x$, то $Op \cdot op$

     – не определён, если op зависит от x

     – равен СО (7.19), в противном случае.

Теперь можно сформулировать определение конкатенации СО.

Пусть заданы два СО Op_1, Op_2, причём Op_2 имеет вид

$$Op_2 = (op_1, \ldots, op_n)$$

Мы будем говорить, что **определена конкатенация** СО Op_1 и Op_2, если выполнены следующие условия:

- хотя бы один из СО Op_1, Op_2 является внутренним
- определены все объекты в скобках в выражении

$$(\ldots((Op_1 \cdot op_1) \cdot op_2) \cdot \ldots) \cdot op_n \tag{7.20}$$

Если эти условия выполнены, то конкатенацией Op_1 и Op_2 называется СО

$$Op_1 \cdot Op_2$$

который равен значению выражения (7.20).

7.8.8 Редукция процессов с СО

Пусть P - процесс с СО.

Редукция процесса P представляет собой последовательность

$$P = P_0 \longrightarrow P_1 \longrightarrow \ldots \longrightarrow P_n \tag{7.21}$$

преобразований этого процесса, каждое из которых производится согласно какому-либо из излагаемых ниже правил. Каждое из этих преобразований (кроме первого) производится над результатом предыдущего преобразования.

Результатом редукции (7.21) является результат последнего из преобразований (т.е. процесс P_n).

Правила редукции имеют следующий вид.

Правило 1 (конкатенация).

Пусть s – некоторое состояние процесса с СО, которое не является начальным, и

- совокупность всех переходов этого процесса с концом s имеет вид

$$s_1 \xrightarrow{Op_1} s, \quad \ldots, s_n \xrightarrow{Op_n} s$$

- совокупность всех переходов этого процесса с началом s имеет вид

$$s \xrightarrow{Op'_1} s'_1, \quad \ldots, s \xrightarrow{Op'_m} s'_m$$

- $s \notin \{s_1, \ldots, s_n, s'_1, \ldots, s'_m\}$
- для каждого $i = 1, \ldots, n$ и каждого $j = 1, \ldots, m$ определена конкатенация

$$Op_i \cdot Op_j$$

Тогда данный процесс можно преобразовать в процесс,

- состояниями которого являются состояния исходного процесса, за исключением s
- переходами которого являются
  - те переходы исходного процесса, началом или концом которых не является s,
  - а также переходы вида

$$s_i \xrightarrow{Op_i \cdot Op'_j} s'_j$$

  для каждого $i = 1, \ldots, n$ и каждого $j = 1, \ldots, m$
- – начальное состояние которого, а также
  – множество переменных, и
  – начальное условие

  совпадают с соответствующими компонентами исходного процесса.

Правило 2 (склейка).

Пусть P – процесс с СО, который содержит два перехода с общим началом и общим концом

$$s_1 \xrightarrow{Op} s_2, \qquad s_1 \xrightarrow{Op'} s_2 \tag{7.22}$$

причём метки этих переходов различаются только в первой компоненте, т.е. Op и Op' имеют вид

$$Op = (op_1, op_2, \ldots, op_n)$$
$$Op' = (op'_1, op_2, \ldots, op_n)$$

Правило 2 заключается в замене пары переходов (7.22) на один переход вида

$$s_1 \xrightarrow{Op} s_2$$

где $Op = ((cond(Op) \vee cond(Op'))\,?, op_2, \ldots, op_n)$

Правило 3 (удаление несущественных присваиваний).

Пусть P – некоторый процесс с СО.

Обозначим знакосочетанием $op(P)$ совокупность всех операторов, входящих в какой-либо из СО процесса P.

Будем называть переменную $x \in X_P$ **несущественной**, если

- x не входит ни в один из
  - операторов проверки условия, и
  - операторов вывода

  из $op(P)$
- если x входит в правую часть какого-либо оператора присваивания из $op(P)$ вида $(y := e)$, то переменная y – несущественная.

Правило 3 заключается в удалении из всех CO редуцируемого процесса операторов присваивания вида $(x := e)$, где переменная x – несущественная.

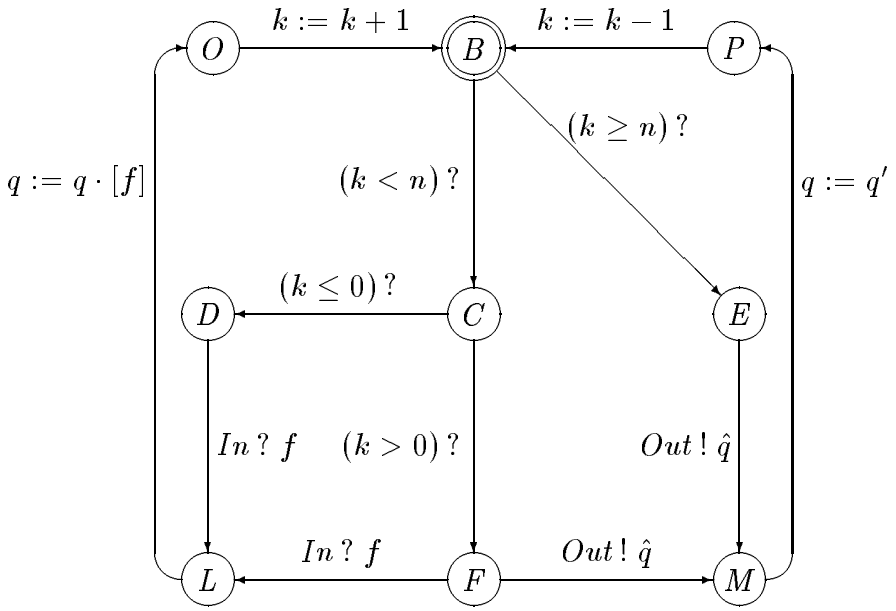
7.8.9 Пример редукции

Рассмотрим в качестве примера редукцию процесса $Buffer_n$ (графовое представление которого приведено в параграфе 7.5.3).

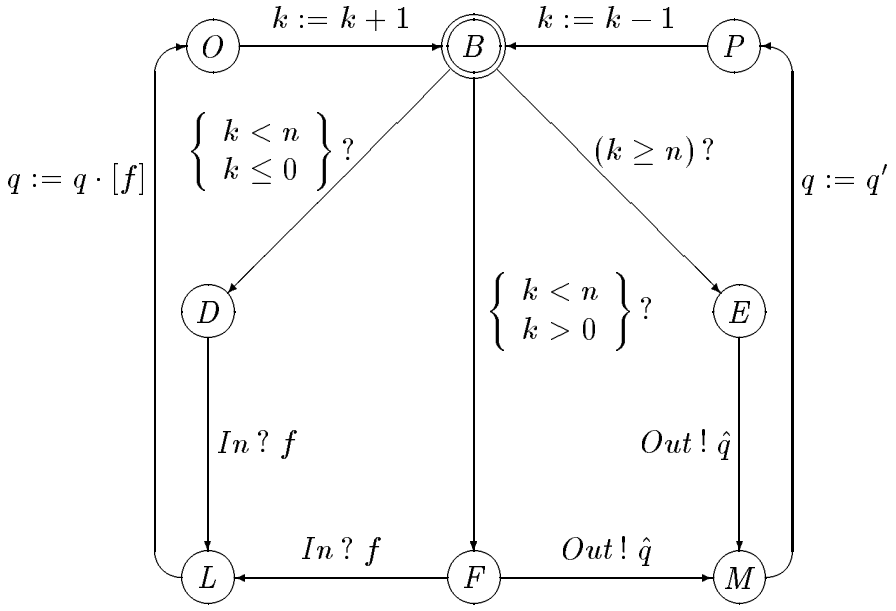
Ниже мы будем использовать следующее соглашение:

- если в CO Op
$$cond(Op) = \top$$
то первый оператор в таком CO мы писать не будем
- операторы, входящие в CO, можно располагать не только по горизонтали, но и по вертикали
- скобки, в которые заключена последовательность операторов, из которых состоит CO, можно опускать.

Исходный процесс $Buffer_n$ имеет следующий вид:

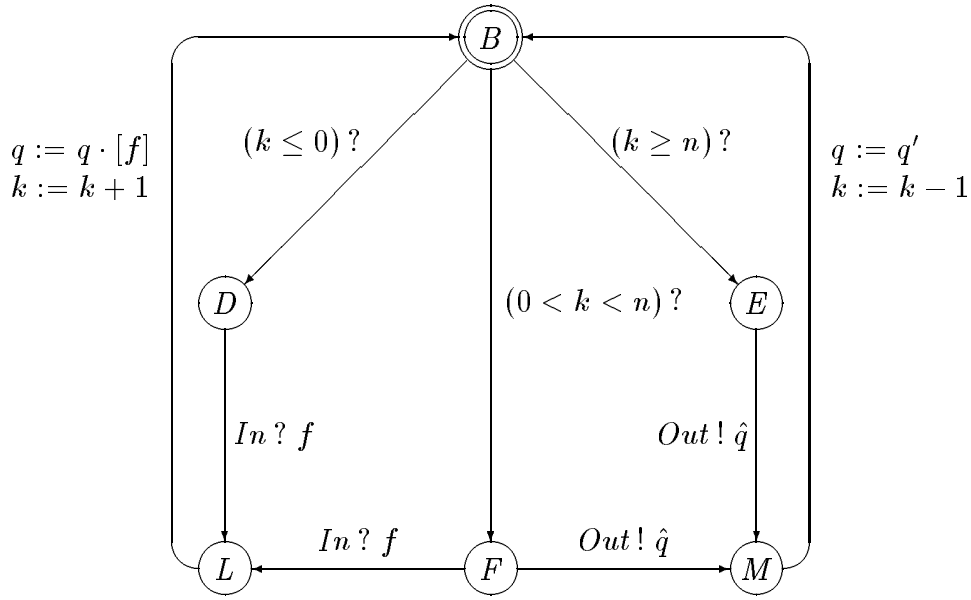


Первый шаг редукции заключается в удалении состояния C (применяется правило 1 для $s = C$):

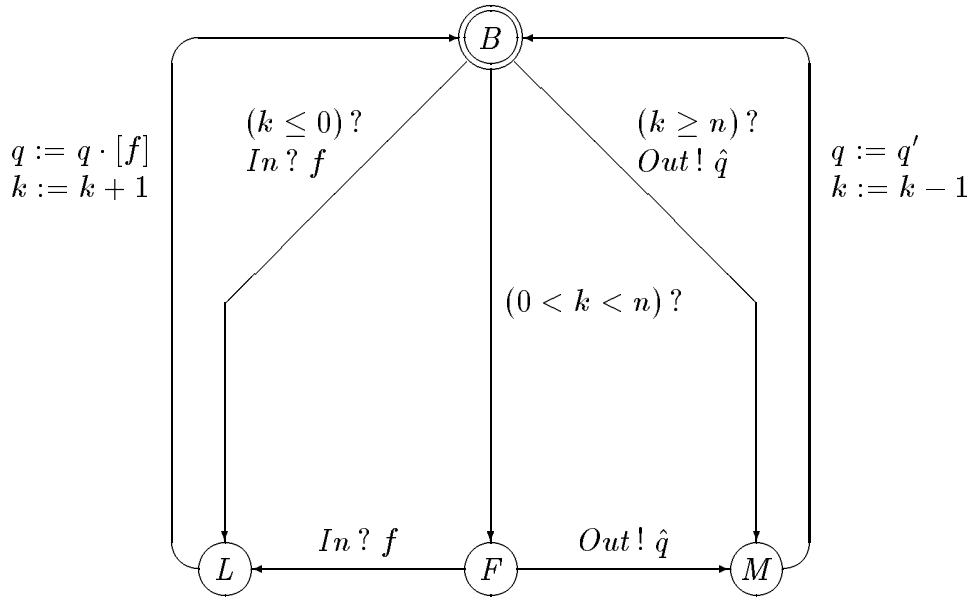


Поскольку $n > 0$, то формулу $(k < n) \wedge (k \leq 0)$ в метке перехода из B в D можно заменить на равносильную формулу $k \leq 0$.

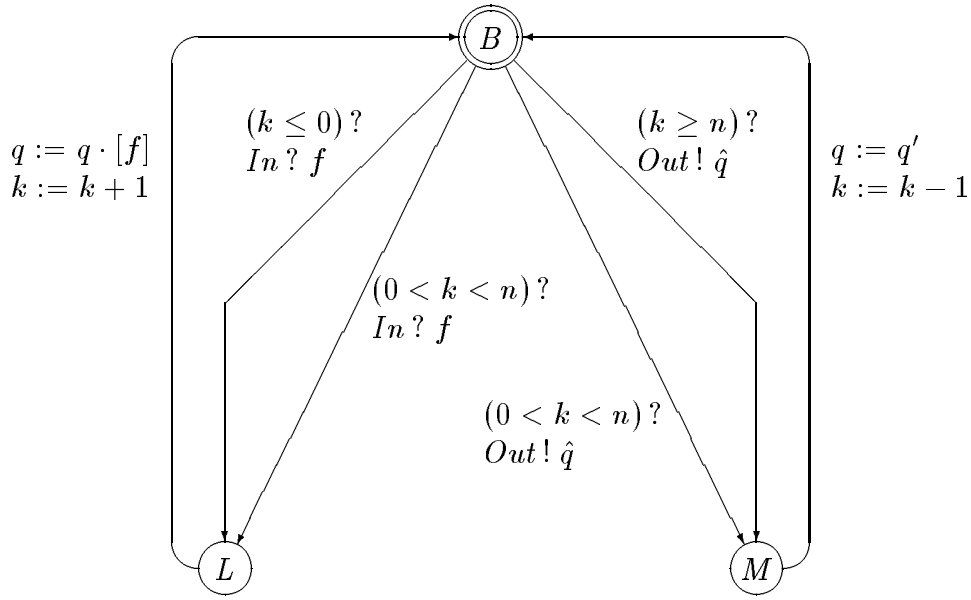
Второй и третий шаги редукции – удаление состояний O и P:



Четвёртый и пятый шаги редукции – удаление состояний D и E:

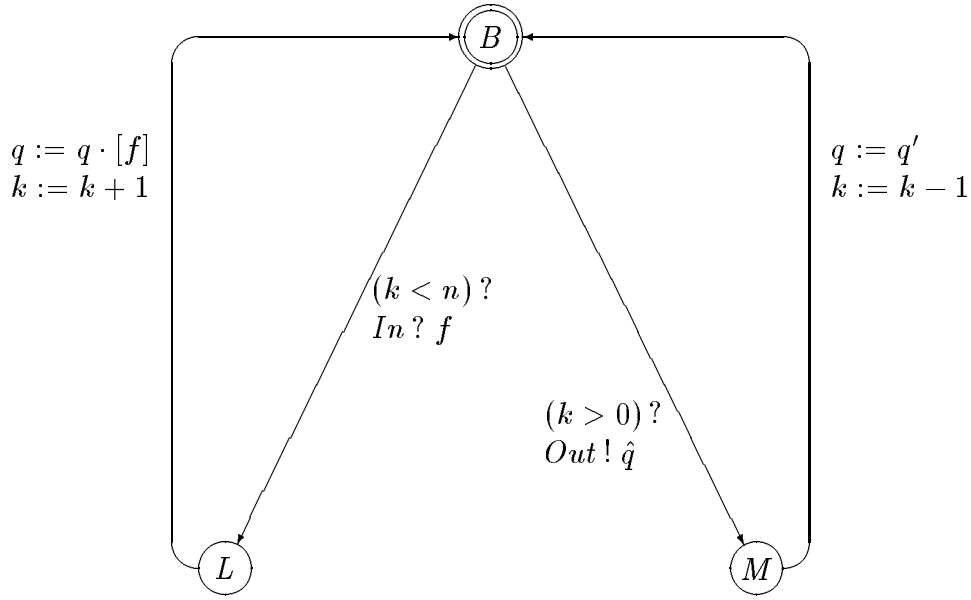


Шестой шаг редукции – удаление состояния F:

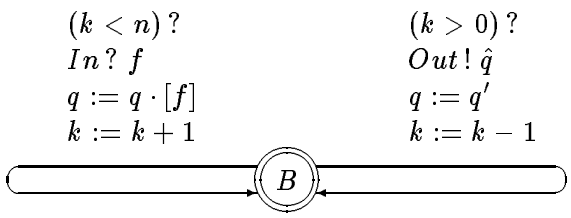


Седьмой и восьмой шаги редукции – применение правила 2 к переходам вида $B \to L$ и $B \to M$. В получившемся процессе мы заменяем

- формулу $(0 < k < n) \vee (k \le 0)$ – на равносильную ей формулу $(k < n)$
- формулу $(0 < k < n) \vee (k \ge n)$ – на равносильную ей формулу $(k > 0)$



Девятый и десятый шаги редукции – удаление состояний L и M. В результате мы получаем не редуцируемый далее процесс с СО с одним состоянием:



(7.23)

7.8.10 Понятие конкретизации процесса с СО

Для процессов с СО можно определить понятие конкретизации, аналогично тому, как это было сделано для обычных процессов с передачей сообщений в параграфе 7.7.1.

Для каждого процесса с СО P знакосочетание $Conc(P)$ обозначает процесс в исходном смысле данного понятия (см. параграф 2.4), который называется **конкретизацией** процесса P, и имеет следующие компоненты.

1. Состояниями $Conc(P)$ являются
   - всевозможные означивания из $Eval(X_P)$,
   - а также дополнительное состояние s^0, которое является начальным в $Conc(P)$
2. Для
   - каждого перехода $s_1 \xrightarrow{Op} s_2$ процесса P, и
   - каждого означивания $\xi \in Eval(X_P)$, такого, что
     - $\xi(at_P) = s_1$, и
     - Op открыт на ξ

   $Conc(P)$ содержит переход

   $$\xi \xrightarrow{a} \xi'$$

   если $\xi'(at_P) = s_2$, и имеет место один из следующих случаев:

   (a) Op – внутренний, $a = \tau$, и имеет место соотношение

   $$\xi \xrightarrow{Op} \xi'$$

   которое означает следующее: если Op имеет вид $(op_1, \ldots, op_n)$, то существует последовательность $\xi_1, \ldots, \xi_n$ означиваний из $Eval(X_P)$, такая, что

- $\forall\, x \in X_P \setminus \{at_P\} \quad \xi(x) = \xi_1(x), \quad \xi'(x) = \xi_n(x)$, и
- для каждого $i = 2, \ldots, n$, если op_i имеет вид $(x := e)$, то

$$\xi_i(x) = \xi_{i-1}(e), \quad \forall y \in X_P \setminus \{x, at_P\} \qquad \xi_i(y) = \xi_{i-1}(y)$$

(b)
- $Op = Op_1 \cdot (\alpha\,?\,x) \cdot Op_2$,
- $a = \alpha\,?\,v$, где v – произвольное значение из $D_{t(x)}$, и
- существуют означивания ξ_1 и ξ_2 из $Eval(X_P)$, такие, что

$$\xi \xrightarrow{Op_1} \xi_1\,, \qquad \xi_2 \xrightarrow{Op_2} \xi'$$
$$\xi_2(x) = v, \quad \forall y \in X_P \setminus \{x, at_P\} \qquad \xi_2(y) = \xi_1(y)$$

(c)
- $Op = Op_1 \cdot (\alpha\,!\,e) \cdot Op_2$,
- существует означивание ξ_1 из $Eval(X_P)$, такое, что

$$\xi \xrightarrow{Op_1} \xi_1\,, \qquad \xi_1 \xrightarrow{Op_2} \xi'\,, \qquad a = \alpha\,!\,\xi_1(e)$$

3. Для

- каждого означивания $\xi \in Eval(X_P)$, такого, что $\xi(I_P) = 1$, и
- каждого перехода в $Conc(P)$ вида $\xi \xrightarrow{a} \xi'$

$Conc(P)$ содержит переход $s^0 \xrightarrow{a} \xi'$.

Читателю предлагается самостоятельно исследовать взаимосвязь между

- конкретизацией произвольного процесса с передачей сообщений P, в том смысле, в каком это понятие было определено в параграфе 7.7.1, и
- конкретизацией процесса с СО, полученного в результате редукции процесса P.

7.8.11 Отношения эквивалентности на процессах с СО

Пусть P_1 и P_2 – процессы с СО.

Мы будем говорить, что P_1 и P_2 **наблюдаемо эквивалентны**, и обозначать этот факт знакосочетанием

$$P_1 \approx P_2$$

если их конкретизации наблюдаемо эквивалентны (в исходном смысле данного понятия, см. параграф 4.8).

Аналогичным образом определяется отношение $\overset{+}{\approx}$ наблюдаемой конгруэнции на процессах с СО.

Используя понятие редукции процессов с СО, можно определить ещё одно отношение эквивалентности на множестве процессов с СО. Данное отношение

- обозначается символом $\overset{r}{\approx}$, и
- представляет собой минимальную конгруэнцию на множестве процессов с СО, обладающую следующим свойством: если P' получается из P в результате применения какого-либо правила редукции, то $P \overset{r}{\approx} P'$

(т.е. $\overset{r}{\approx}$ является пересечением всех конгруэнций на множестве процессов с СО, обладающих вышеуказанным свойством).

Читателю предлагается самостоятельно

- исследовать связь операций на процессах с СО с отношениями $\approx$ и $\overset{+}{\approx}$, т.е. установить свойства, аналогичные свойствам, изложенным в параграфах 3.7, 4.5, 4.8.4, 4.9.5
- сформулировать и обосновать необходимые и достаточные условия наблюдаемой эквивалентности процессов с СО, не использующие понятие конкретизации
- исследовать взаимосвязь между отношениями $\approx$, $\overset{+}{\approx}$ и $\overset{r}{\approx}$
- найти такие правила редукции, чтобы было верно включение

$$\overset{r}{\approx} \subseteq \overset{+}{\approx}$$

7.8.12 Метод доказательства наблюдаемой эквивалентности процессов с СО

Один из возможных методов доказательства наблюдаемой эквивалентности процессов с СО основан на излагаемой ниже теореме 34. Для формулировки этой теоремы мы введём вспомогательные понятия и обозначения.

1. Пусть P – процесс с СО.

   Составной переход (СП) в P – это (возможно пустая) последовательность CT переходов процесса P вида

   $$CT = \quad s_0 \xrightarrow{Op_1} s_1 \xrightarrow{Op_2} \ldots \xrightarrow{Op_n} s_n \qquad (n \geq 0) \qquad (7.24)$$

   такая, что

   - среди $Op_1, \ldots, Op_n$ – не более одного СО ввода или вывода
   - определена конкатенация

     $$(\ldots(Op_1 \cdot Op_2) \cdot \ldots) \cdot Op_n$$

     которую мы будем обозначать тем же символом CT.

Если последовательность (7.24) пуста, то её конкатенацией CT по определению является СО ($\top$?).

Состояние s_0 называется **началом** СП (7.24), а состояние s_n – его **концом**.

Знакосочетание $s_0 \xrightarrow{CT} s_n$ является сокращённой записью утверждения о том, что CT – это

- СП с началом s_0 и концом s_n
- а также – СО, соответствующий этому СП.

2. Пусть φ и ψ – формулы.

   Знакосочетание $\varphi \leq \psi$ является сокращённой записью утверждения о том, что формула $\varphi \to \psi$ является тождественно истинной.

3. Пусть $Op = (op_1, \ldots, op_n)$ – внутренний СО, и φ – формула.

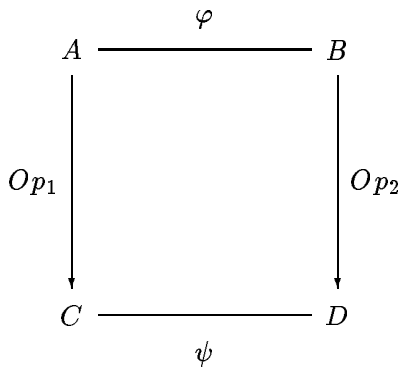
   Знакосочетание $Op(\varphi)$ обозначает формулу, определяемую рекурсивно:

$$Op(\varphi) \stackrel{\text{def}}{=} \begin{cases} cond\,(Op) \to \varphi, & \text{если } n = 1 \\ (op_1, \ldots, op_{n-1})\,(op_n(\varphi)), & \text{если } n > 1 \end{cases}$$

   где $op_n(\varphi)$ обозначает формулу, определяемую следующим образом: если op_n имеет вид $(x := e)$, то $op_n(\varphi)$ получается из φ заменой каждого вхождения в неё переменной x на выражение e.

4. Пусть φ, ψ – формулы, и Op_1, Op_2 – СО.

   Мы будем говорить, что верна диаграмма



(7.25)

   если выполнено одно из следующих условий.

   (a) Op_1 и Op_2 – внутренние СО, и верно неравенство

$$\varphi \leq (Op_1 \cdot Op_2)(\psi)$$

(b) Op_1 и Op_2 можно представить в виде конкатенаций

$$Op_1 = Op_3 \cdot (\alpha\,?\,x) \cdot Op_4$$
$$Op_2 = Op_5 \cdot (\alpha\,?\,y) \cdot Op_6$$

где Op_3, Op_4, Op_5, Op_6 – внутренние СО, и верно неравенство

$$\varphi \leq (Op_1' \cdot Op_2')(\psi)$$

где

- $Op_1' = Op_3 \cdot (x := z) \cdot Op_4$
- $Op_2' = Op_5 \cdot (y := z) \cdot Op_6$
- z – новая переменная (т.е. не входящая в φ, ψ, Op_1, Op_2)

(c) Op_1 и Op_2 можно представить в виде конкатенаций

$$Op_1 = Op_3 \cdot (\alpha\,!\,e_1) \cdot Op_4$$
$$Op_2 = Op_5 \cdot (\alpha\,!\,e_2) \cdot Op_6$$

где Op_3, Op_4, Op_5, Op_6 – внутренние СО, и верно неравенство

$$\varphi \leq \left\{ \begin{array}{l} (Op_3 \cdot Op_5)(e_1 = e_2) \\ (Op_3 \cdot Op_4 \cdot Op_5 \cdot Op_6)(\psi) \end{array} \right\}$$

Теорема 34.
Пусть P_1 и P_2 – два процесса с СО:

$$P_i = (X_{P_i}, I_{P_i}, S_{P_i}, s^0_{P_i}, R_{P_i}) \qquad (i = 1, 2)$$

не имеющие общих состояний и общих переменных.

P_1 и P_2 находятся в отношении $\approx$, если существует функция μ вида

$$\mu : S_{P_1} \times S_{P_2} \to Fm$$

обладающая следующими свойствами.

1. $I_{P_1} \wedge I_{P_2} \leq \mu(s^0_{P_1}, s^0_{P_2})$.
2. Для
   - каждой пары $(A_1, A_2) \in S_{P_1} \times S_{P_2}$, и
   - каждого перехода $A_1 \xrightarrow{Op} A_1'$ процесса P_1, такого, что

$$cond(Op) \ \wedge \ \mu(A_1, A_2) \neq \bot \qquad (7.26)$$

существует совокупность СП процесса P_2 с началом A_2

$$\{\, A_2 \xrightarrow{CT_i} A_2^i \mid i \in \Im\} \tag{7.27}$$

удовлетворяющая следующим условиям:

(a) имеет место неравенство

$$cond\,(Op) \;\wedge\; \mu(A_1, A_2) \;\leq\; \bigvee_{i \in \Im} cond\,(CT_i) \tag{7.28}$$

(b) для каждого $i \in \Im$ верна диаграмма

$$\begin{array}{ccc} & \mu(A_1, A_2) & \\ A_1 & \longrightarrow & A_2 \\ \Big\downarrow Op & & \Big\downarrow CT_i \\ A_1' & \longrightarrow & A_2^i \\ & \mu(A_1', A_2^i) & \end{array} \tag{7.29}$$

3. Свойство, симметричное предыдущему: для

   - каждой пары $(A_1, A_2) \in S_{P_1} \times S_{P_2}$, и
   - каждого перехода $A_2 \xrightarrow{Op} A_2'$ процесса P_2, такого, что верно (7.26)

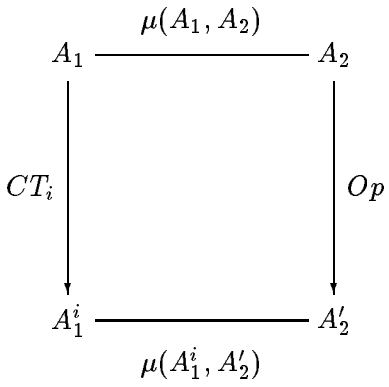
   существует совокупность СП процесса P_1 с началом A_1

$$\{\, A_1 \xrightarrow{CT_i} A_1^i \mid i \in \Im\} \tag{7.30}$$

   удовлетворяющая следующим условиям:

   (a) имеет место неравенство (7.28)

   (b) для каждого $i \in \Im$ верна диаграмма



(7.31)

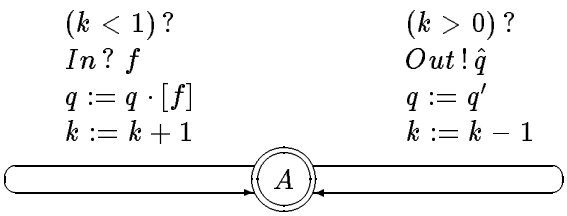
7.8.13 Пример доказательства наблюдаемой эквивалентности процессов с СО

В качестве примера использования теоремы 34 докажем наблюдаемую эквивалентность

$$Buffer_1 \approx Buf$$

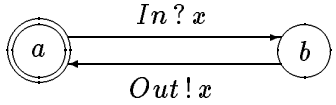
где

- $Buffer_1$ – это рассмотренный выше процесс $Buffer_n$ (см. (7.23)) при $n = 1$, т.е. процесс, имеющий вид



  его начальное условие – $(k = 0) \wedge (q = \varepsilon)$, и

- Buf – процесс, имеющий вид



  начальное условие этого процесса $= \top$.

Определим функцию $\mu : \{A\} \times \{a, b\} \to Fm$ следующим образом:

$$\mu(A, a) \stackrel{\text{def}}{=} (k = 0) \wedge (q = \varepsilon)$$
$$\mu(A, b) \stackrel{\text{def}}{=} (k = 1) \wedge (q = [x])$$

Проверим свойства 1, 2, и 3 для функции μ.

1. Свойство 1 в данном случае представляет собой неравенство

$$((k = 0) \wedge (q = \varepsilon)) \wedge \top \leq ((k = 0) \wedge (q = \varepsilon))$$

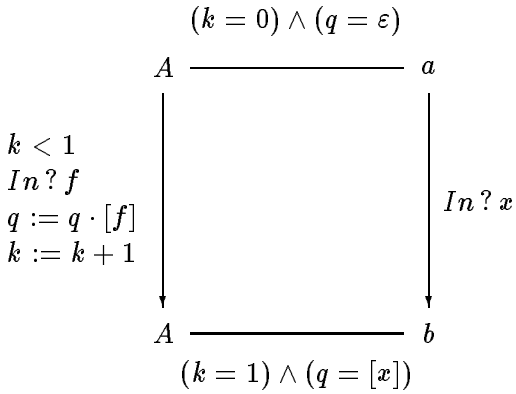
которое, очевидно, верно.

2. Проверим свойство 2.

   - Для пары (A, a) нам надо рассмотреть лишь левый переход в процессе $Buffer_1$ (так как для правого перехода не выполнено условие (7.26)).

     В качестве (7.27) мы возьмем совокупность, состоящую из единственного перехода из a в b.

     Диаграмма (7.29) в данном случае имеет вид



(7.32)

     Пользуясь тем, что

$$\forall\, \varphi, \psi, \theta \in Fm \qquad (\varphi \leq \psi \to \theta \quad \Leftrightarrow \quad \varphi \wedge \psi \leq \theta) \tag{7.33}$$

     запишем неравенство, соответствующее этой диаграмме, в виде

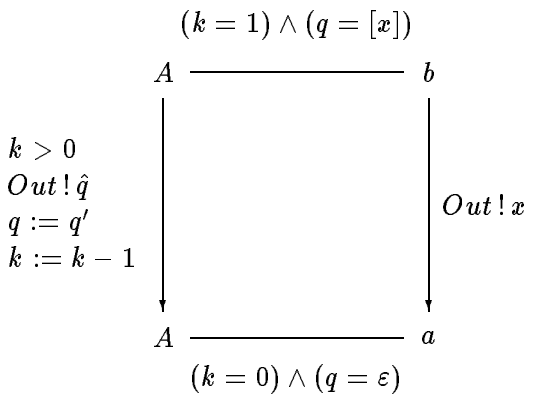
$$\left\{ \begin{array}{l} k = 0 \\ q = \varepsilon \\ k < 1 \end{array} \right\} \leq \left\{ \begin{array}{l} k + 1 = 1 \\ q \cdot [z] = [z] \end{array} \right\} \tag{7.34}$$

     Очевидно, что данное неравенство верно.

   - Для пары (A, b) нам надо рассмотреть лишь правый переход в процессе $Buffer_1$ (так как для левого перехода не выполнено условие (7.26)).

     В качестве (7.27) мы возьмем совокупность, состоящую из единственного перехода из b в a.

Диаграмма (7.29) в данном случае имеет вид



(7.35)

Пользуясь (7.33), запишем неравенство, соответствующее этой диаграмме, в виде

$$\left\{ \begin{array}{l} k = 1 \\ q = [x] \\ k > 0 \end{array} \right\} \leq \left\{ \begin{array}{l} \hat{q} = x \\ k - 1 = 0 \\ q' = \varepsilon \end{array} \right\} \tag{7.36}$$

Очевидно, что это неравенство верно.

3. Проверим свойство 3.

   - Для пары (A, a) и для единственного перехода из a в b в качестве (7.30) мы возьмем совокупность, состоящую из левого перехода из A в A.

     Диаграмма (7.31) в данном случае имеет вид (7.32). Как уже было установлено, данная диаграмма верна.

   - Для пары (A, b) и для единственного перехода из b в a в качестве (7.30) мы возьмем совокупность, состоящую из правого перехода из A в A.

     Диаграмма (7.31) в данном случае имеет вид (7.35). Как уже было установлено, данная диаграмма верна.

7.8.14 Дополнительные замечания

Для повышения удобства применения теоремы 34 можно использовать следующие понятия и утверждения.

Инварианты процессов

Пусть P – процесс с СО.

Формула Inv с переменными из X_P называется **инвариантом** процесса P, если она обладает следующими свойствами.

- $I_P \leq Inv$
- для каждого перехода $s \xrightarrow{Op} s'$ процесса P
  - если Op – внутренний, то $Inv \leq Op(Inv)$
  - если Op – СО ввода, и имеет вид $Op_1 \cdot (\alpha\,?\,x) \cdot Op_2$, то

    $$Inv \leq (Op_1 \cdot (x := z) \cdot Op_2)(Inv)$$

    где z – переменная, не входящая в X_P
  - если Op – СО вывода, и имеет вид $Op_1 \cdot (\alpha\,!\,e) \cdot Op_2$, то

    $$Inv \leq (Op_1 \cdot Op_2)(Inv)$$

С использованием понятия инварианта процесса теорему 34 можно модифицировать следующим образом.

Теорема 35.
Пусть

- P_1 и P_2 – два процесса с СО:

  $$P_i = (X_{P_i}, I_{P_i}, S_{P_i}, s^0_{P_i}, R_{P_i}) \qquad (i = 1, 2)$$

  не имеющие общих состояний и общих переменных, и
- формулы Inv_1 и Inv_2 являются инвариантами процессов P_1 и P_2 соответственно.

P_1 и P_2 находятся в отношении $\approx$, если существует функция μ вида

$$\mu : S_{P_1} \times S_{P_2} \to Fm$$

обладающая следующими свойствами.

1. $I_{P_1} \wedge I_{P_2} \leq \mu(s^0_{P_1}, s^0_{P_2})$.
2. Для
   - каждой пары $(A_1, A_2) \in S_{P_1} \times S_{P_2}$, и

- каждого перехода $A_1 \xrightarrow{Op} A_1'$ процесса P_1, такого, что

$$\left\{ \begin{array}{l} cond\,(Op) \\ \mu(A_1, A_2) \\ Inv_1 \\ Inv_2 \end{array} \right\} \neq \bot \tag{7.37}$$

существует совокупность СП процесса P_2 с началом A_2

$$\{\, A_2 \xrightarrow{CT_i} A_2^i \mid i \in \Im\} \tag{7.38}$$

удовлетворяющая следующим условиям:

(a) имеет место неравенство

$$\left\{ \begin{array}{l} cond\,(Op) \\ \mu(A_1, A_2) \\ Inv_1 \\ Inv_2 \end{array} \right\} \leq \bigvee_{i \in \Im} cond\,(CT_i) \tag{7.39}$$

(b) для каждого $i \in \Im$ верна диаграмма

$$\begin{array}{ccc} & \left\{ \begin{array}{l} \mu(A_1, A_2) \\ Inv_1 \\ Inv_2 \end{array} \right\} & \\ A_1 & \longrightarrow & A_2 \\ \Big\downarrow Op & & \Big\downarrow CT_i \\ A_1' & \longrightarrow & A_2^i \\ & \mu(A_1', A_2^i) & \end{array} \tag{7.40}$$

3. Свойство, симметричное предыдущему: для

- каждой пары $(A_1, A_2) \in S_{P_1} \times S_{P_2}$, и
- каждого перехода $A_2 \xrightarrow{Op} A_2'$ процесса P_2, такого, что верно (7.37)

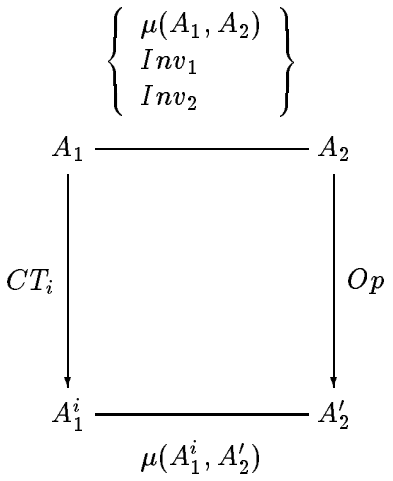
существует совокупность СП процесса P_1 с началом A_1

$$\{\, A_1 \xrightarrow{CT_i} A_1^i \mid i \in \Im\} \tag{7.41}$$

удовлетворяющая следующим условиям:

(a) имеет место неравенство (7.39)

(b) для каждого $i \in \Im$ верна диаграмма



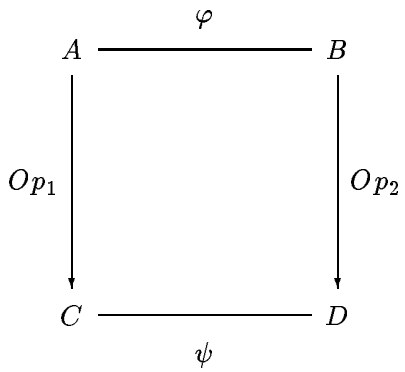
(7.42)

Композиция диаграмм

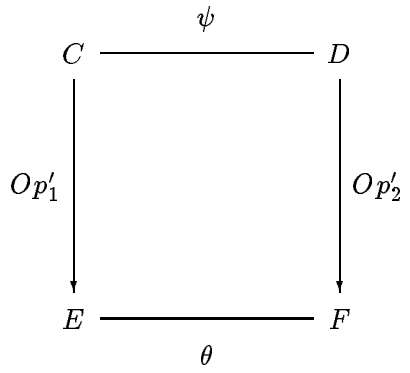
Теорема 36.

Пусть

- φ, ψ, θ – формулы
- Op_1, Op_2 – внутренние СО, такие, что верна диаграмма

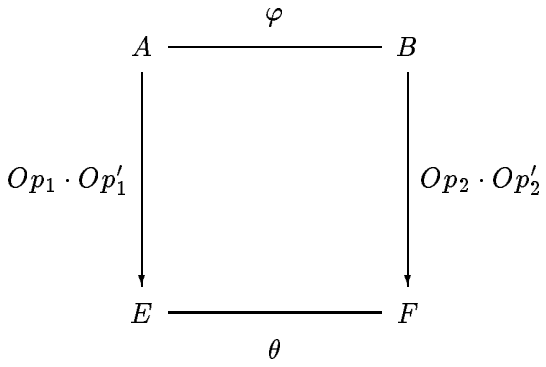


- Op_1', Op_2' – СО, такие, что верна диаграмма



- $\{Op_1, Op_1'\}$ и $\{Op_2, Op_2'\}$ не имеют общих переменных.

Тогда верна диаграмма



7.8.15 Другой пример доказательства наблюдаемой эквивалентности процессов с СО

В качестве примера использования теорем из параграфа 7.8.14 докажем наблюдаемую эквивалентность

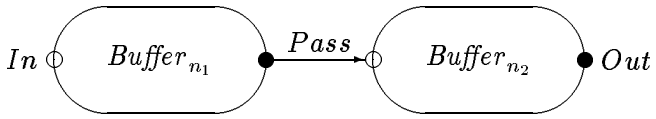
- процесса

$$(Buffer_{n_1}[Pass/Out] \mid Buffer_{n_2}[Pass/In]) \setminus \{Pass\} \qquad (7.43)$$

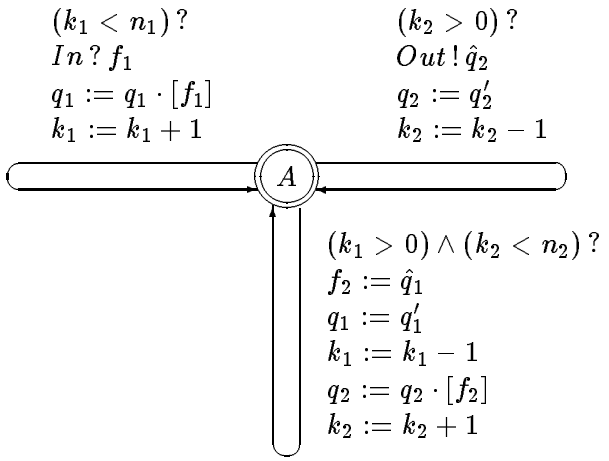
  где $Pass \notin \{In, Out\}$, и

- процесса $Buffer_{n_1+n_2}$.

Процесс (7.43) представляет собой последовательную композицию даух буферов размеров n_1 и n_2. Его потоковый граф имеет вид



Согласно определению операций на процессах с СО (см. параграф 7.8.5), графовое представление процесса (7.43) выглядит следующим образом:

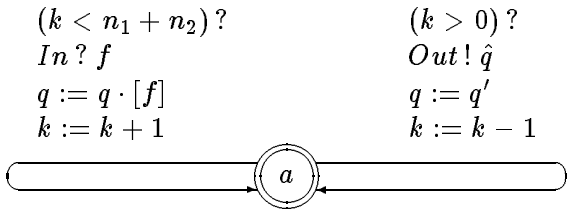


(7.44)

Начальным условием процесса (7.44) является формула

$$\left\{ \begin{array}{l} (n_1 > 0) \ \wedge \ (k_1 = 0) \ \wedge \ (q_1 = \varepsilon) \\ (n_2 > 0) \ \wedge \ (k_2 = 0) \ \wedge \ (q_2 = \varepsilon) \end{array} \right\}$$

Графовое представление процесса $Buffer_{n_1+n_2}$ имеет вид



Начальным условием процесса $Buffer_{n_1+n_2}$ является формула

$$(n_1 + n_2 > 0) \ \wedge \ (k = 0) \ \wedge \ (q = \varepsilon)$$

Нетрудно проверить, что формула

$$Inv \stackrel{\text{def}}{=} \left\{ \begin{array}{l} 0 \le k_1 \le n_1 \\ |q_1| = k_1 \\ 0 \le k_2 \le n_2 \\ |q_2| = k_2 \\ n_1 > 0 \\ n_2 > 0 \end{array} \right\}$$

является инвариантом процесса (7.44). Этот факт следует, в частности, из истинности для каждой строки u и каждого символа a соотношений

$$\left\{ \begin{array}{l} |u| > 0 \quad \Rightarrow \quad |u'| = |u| - 1 \\ |u \cdot [a]| = |[a] \cdot u| = |u| + 1 \end{array} \right.$$

В качестве инварианта второго процесса мы возьмём формулу $\top$.

Определим функцию $\mu : \{A\} \times \{a\} \to Fm$ следующим образом:

$$\mu(A, a) \stackrel{\text{def}}{=} \left\{ \begin{array}{l} q = q_2 \cdot q_1 \\ k = k_2 + k_1 \end{array} \right\}$$

Проверим свойства 1, 2, и 3 для функции μ.

1. Свойство 1 в данном случае представляет собой неравенство

$$\left\{ \begin{array}{l} (n_1 > 0) \ \wedge \ (k_1 = 0) \ \wedge \ (q_1 = \varepsilon) \\ (n_2 > 0) \ \wedge \ (k_2 = 0) \ \wedge \ (q_2 = \varepsilon) \\ (n_1 + n_2 > 0) \ \wedge \ (k = 0) \ \wedge \ (q = \varepsilon) \end{array} \right\} \le \left\{ \begin{array}{l} q = q_2 \cdot q_1 \\ k = k_2 + k_1 \end{array} \right\}$$

   которое, очевидно, верно.

2. Проверим свойство 2.

   - Для левого перехода процесса (7.44) неравенство (7.37) верно. В качестве (7.38) мы возьмем совокупность, единственным элементом которой является левый переход процесса $Buffer_{n_1+n_2}$.
     Неравенство (7.39) в данном случае имеет вид

$$\left\{ \begin{array}{l} k_1 < n_1 \\ q = q_2 \cdot q_1 \\ k = k_2 + k_1 \\ Inv \end{array} \right\} \le (k < n_1 + n_2)$$

     что, очевидно, верно.

Пользуясь (7.33), запишем неравенство, соответствующее диаграмме (7.40) для данного случая в виде

$$\left\{\begin{array}{l} q = q_2 \cdot q_1 \\ k = k_2 + k_1 \\ Inv \\ k_1 < n_1 \\ k < n_1 + n_2 \end{array}\right\} \leq \left\{\begin{array}{l} q \cdot [z] = q_2 \cdot q_1 \cdot [z] \\ k + 1 = k_2 + k_1 + 1 \end{array}\right\} \qquad (7.45)$$

Нетрудно установить, что последнее неравенство верно.

- Для среднего (внутреннего) перехода процесса (7.44) неравенство (7.37) верно. В качестве (7.38) мы возьмем совокупность, единственным элементом которой является пустой СП процесса $Buffer_{n_1+n_2}$.
  Неравенство (7.39) в данном случае верно по тривиальной причине: правая часть в нём имеет вид $\top$.
  Пользуясь соотношением (7.33), запишем неравенство, соответствующее диаграмме (7.40) для данного случая, в виде

$$\left\{\begin{array}{l} q = q_2 \cdot q_1 \\ k = k_2 + k_1 \\ Inv \\ k_1 > 0 \\ k_2 < n_2 \end{array}\right\} \leq \left\{\begin{array}{l} q = (q_2 \cdot [\hat{q}_1]) \cdot q_1' \\ k = k_2 + 1 + k_1 - 1 \end{array}\right\} \qquad (7.46)$$

  Данное неравенство следует из
  - ассоциативности операции конкатенации, и
  - истинности для каждой строки u соотношения

$$|u| > 0 \quad \Rightarrow \quad u = [\hat{u}] \cdot u'$$

- Для правого перехода процесса (7.44) неравенство (7.37) верно. В качестве (7.38) мы возьмем совокупность, единственным элементом которой является правый переход процесса $Buffer_{n_1+n_2}$.
  Неравенство (7.39) в данном случае имеет вид

$$\left\{\begin{array}{l} k_2 > 0 \\ q = q_2 \cdot q_1 \\ k = k_2 + k_1 \\ Inv \end{array}\right\} \leq (k > 0)$$

  что, очевидно, верно.

Пользуясь соотношением (7.33), запишем неравенство, соответствующее диаграмме (7.40) для данного случая, в виде

$$\left\{\begin{array}{l} q = q_2 \cdot q_1 \\ k = k_2 + k_1 \\ Inv \\ k_2 > 0 \\ k > 0 \end{array}\right\} \leq \left\{\begin{array}{l} \hat{q}_2 = \hat{q} \\ q' = q'_2 \cdot q_1 \\ k - 1 = k_2 - 1 + k_1 \end{array}\right\} \qquad (7.47)$$

Данное неравенство следует из истинности для каждой пары строк u, v соотношения

$$|u| > 0 \quad \Rightarrow \quad \left\{\begin{array}{l} (u \cdot v)\hat{} = \hat{u} \\ (u \cdot v)' = u' \cdot v \end{array}\right\}$$

3. Проверим свойство 3.

- Для левого перехода процесса $Buffer_{n_1+n_2}$ неравенство (7.37) верно. В качестве (7.41) мы возьмем совокупность, состоящую из двух СП:
  - левого перехода процесса (7.44), и
  - последовательности из пары переходов,
    - первым элементов которой является средний (внутренний) переход процесса (7.44),
    - а вторым – левый переход процесса (7.44)

  Неравенство (7.39) в данном случае имеет вид

$$\left\{\begin{array}{l} k < n_1 + n_2 \\ q = q_2 \cdot q_1 \\ k = k_2 + k_1 \\ Inv \end{array}\right\} \leq (k_1 < n_1) \ \vee \ \left\{\begin{array}{l} k_1 > 0 \\ k_2 < n_2 \\ k_1 - 1 < n_1 \end{array}\right\}$$

  Это неравенство верно (при его обосновании используется содержащийся в Inv конъюнктивный член $n_1 > 0$).

  Истинность неравенств, соответствующих диаграммам (7.42) для обоих элементов совокупности (7.41), следует из (7.45), (7.46) и теоремы 36.

- Для правого перехода процесса $Buffer_{n_1+n_2}$ неравенство (7.37) верно. В качестве (7.41) мы возьмем совокупность, состоящую из двух СП:
  - правого перехода процесса (7.44), и
  - последовательности из пары переходов,

* первым элементов которой является средний (внутренний) переход процесса (7.44),
* а вторым – правый переход процесса (7.44)

Неравенство (7.39) в данном случае имеет вид

$$\left\{ \begin{array}{l} k > 0 \\ q = q_2 \cdot q_1 \\ k = k_2 + k_1 \\ Inv \end{array} \right\} \leq (k_2 > 0) \ \vee \ \left\{ \begin{array}{l} k_1 > 0 \\ k_2 < n_2 \\ k_2 + 1 > 0 \end{array} \right\}$$

Это неравенство верно (при его обосновании используется содержащийся в Inv конъюнктивный член $n_2 > 0$).

Истинность неравенств, соответствующих диаграммам (7.42) для обоих элементов совокупности (7.41), следует из (7.46), (7.47) и теоремы 36.

7.9 Рекурсивные определения процессов

Для процессов с передачей сообщений можно ввести понятие **рекурсивного определения**, которое аналогично понятию РО, изложенному в главе 5.

РО для процессов с передачей сообщений очень похожи на функциональные программы.

Понятие РО определяется на основе понятия **процессного выражения (ПВ)**, которое почти совпадает с соответствующим понятием из параграфа 5.1, поэтому мы лишь укажем отличия в определениях этих понятий.

- Во всех ПВ вместо действий используются операторы.
- Каждое процессное имя A имеет **тип** $t(A)$, который представляет собой последовательность обычных типов из $Types$:

$$t(A) = (t_1, \ldots, t_n) \qquad (n \geq 0)$$

- Каждое процессное имя A входит в каждое ПВ только вместе со списком выражений соответствующих типов, т.е. каждое вхождение процессного имени A в произвольное ПВ содержится в подвыражении вида

$$A(e_1, \ldots, e_n), \qquad \text{где } (t(e_1), \ldots, t(e_n)) = t(A)$$

Каждому ПВ P соответствует совокупность $fv(P)$ **свободных переменных** этого ПВ, которая состоит из всех переменных, имеющих свободные вхождения в P.

Понятие свободного и связанного вхождения переменной в ПВ определяется почти так же, как определяется аналогичное понятие в логике предикатов, только в случае ПВ роль кванторов играют операторы ввода и присваивания: каждое свободное вхождение переменной x в ПВ P становится связанным в ПВ $(\alpha?x).P$ и в ПВ $(x := e).P$.

Рекурсивное определение (РО) процессов представляет собой список формальных равенств вида

$$\begin{cases} A_1(x_{11}, \ldots, x_{1k_1}) = P_1 \\ \ldots \\ A_n(x_{n1}, \ldots, x_{nk_n}) = P_n \end{cases} \tag{7.48}$$

где

- $A_1, \ldots, A_n$ – процессные имена,
- для каждого $i = 1, \ldots, n$ список $(x_{i1}, \ldots, x_{ik_i})$ в левой части i–го равенства представляет собой список различных переменных
- $P_1, \ldots, P_n$ – ПВ, которые удовлетворяют
  - условиям, изложенным в определении РО в параграфе 5.2,
  - а также следующему условию: для каждого $i = 1, \ldots, n$ совокупность $\{x_{i1}, \ldots, x_{ik_i}\}$ совпадает с совокупностью $fv(P_i)$ свободных переменных ПВ P_i.

РО (7.48) можно интерпретировать как функциональную программу, состоящую из рекурсивных определений функций

$$A_1(x_{11}, \ldots, x_{1k_1}), \ldots, A_n(x_{n1}, \ldots, x_{nk_n})$$

Для каждого $i = 1, \ldots, n$ переменные x_{i1}, ..., x_{ik_i} можно рассматривать как формальные параметры функции $A_i(x_{i1}, \ldots, x_{ik_i})$.

Читателю предлагается самостоятельно определить соответствие, которое сопоставляет каждому ПВ вида $A(x_1, \ldots, x_n)$, где

- A – процессное имя, и
- $x_1, \ldots, x_n$ – список различных переменных соответствующих типов

процесс $[\![A(x_1, \ldots, x_n)]\!]$.

Примеры процессов, задаваемых в виде РО, будут изложены в главе 9.

Проблемы, связанные с РО в случае процессов с передачей сообщений, имеют тот же вид, что и для обычных РО:

- распознавание существования конечных процессов, эквивалентных процессам вида $[\![A(x_1, \ldots, x_n)]\!]$

- построение алгоритмов нахождения минимальных процессов, эквивалентных процессам вида $[\![A(x_1, \ldots, x_n)]\!]$ в том случае, когда эти процессы конечны
- распознавание эквивалентности процессов вида $[\![A(x_1, \ldots, x_n)]\!]$
- распознавание эквивалентности РО
- нахождение необходимых и достаточных условий единственности списка процессов, определяемого РО.

Глава 8

Примеры процессов с передачей сообщений

8.1 Разделение множеств

8.1.1 Задача разделения множеств

Пусть задана пара U, V конечных непересекающихся множеств, причём каждому элементу $x \in U \cup V$ сопоставлено некоторое число $w(x)$, называемое **весом** этого элемента.

Требуется преобразовать эту пару в такую пару множеств U', V', чтобы

- $|U| = |U'|, \quad |V| = |V'|$
  (для каждого конечного множества M знакосочетание $|M|$ обозначает количество элементов в M)
- для каждого $u \in U'$ и каждого $v \in V'$ выполнялось неравенство

$$w(u) \leq w(v)$$

8.1.2 Распределённый алгоритм решения задачи разделения множеств

Задачу разделения множеств можно решить путём выполнения нескольких сеансов обмена элементами между этими множествами. Каждый сеанс состоит из следующих действий:

- нахождение элемента mx с максимальным весом в левом множестве
- нахождение элемента mn с минимальным весом в правом множестве
- перенесение

– mx из левого множества в правое, и

– mn из правого множества в левое.

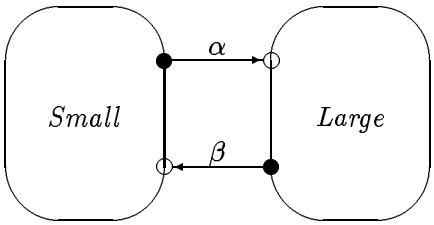
Для реализации этой идеи предлагается использовать распределённый алгоритм, определяемый как процесс вида

$$(Small \mid Large) \setminus \{\alpha, \beta\} \tag{8.1}$$

где

- процесс $Small$ выполняет операции, связанные с левым множеством, и
- процесс $Large$ выполняет операции, связанные с правым множеством.

Потоковый граф, соответствующий этому алгоритму, имеет вид



Ниже мы будем использовать следующие обозначения:

- для каждого подмножества $W \subseteq U \cup V$ знакосочетания $\max(W)$ и $\min(W)$ обозначают элемент множества W с максимальным и минимальным весом соответственно,
- для
  – любых подмножеств $W_1, W_2 \subseteq U \cup V$, и
  – любого $u \in U \cup V$

  выражения
  $$W_1 \leq u, \quad u \leq W_1, \quad W_1 \leq W_2$$
  являются сокращённой записью выражений
  $$\begin{array}{l} \forall x \in W_1 \quad w(x) \leq w(u) \\ \forall x \in W_1 \quad w(u) \leq w(x) \\ \forall x \in W_1,\ \forall y \in W_2 \quad w(x) \leq w(y) \end{array}$$
  соответственно

Аналогичный смысл имеют выражения

$$\max(W), \quad \min(W), \quad W \leq u, \quad u \leq W, \quad W_1 \leq W_2$$

в которых символы W, W_i и u обозначают переменные, значениями которых являются

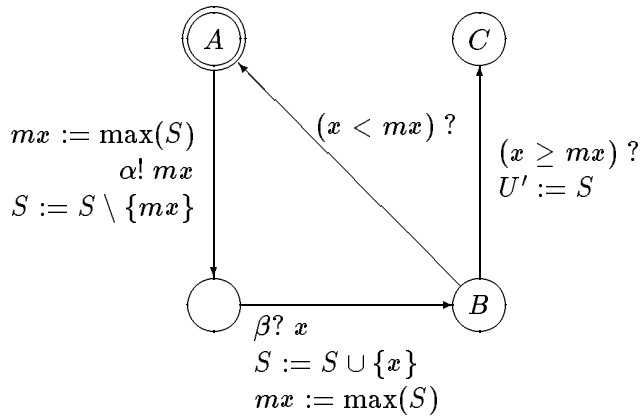
- помножества множества $U \cup V$, и
- элементы множества $U \cup V$

соответственно.

8.1.3 Процессы *Small* и *Large*

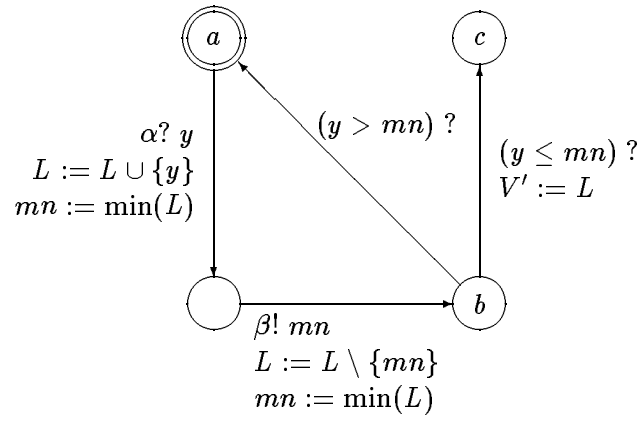
Процессы *Small* и *Large* могут быть определены в виде блок-схем, которые затем можно преобразовать в процессы с СО и редуцировать. Мы не будем давать описания этих блок-схем и излагать их преобразование в процессы с СО и этапы их редукции, приведём лишь соответствующие редуцированные процессы с СО.

Процесс *Small* имеет следующий вид:
$Init = (S = U)$



(8.2)

Процесс *Large* имеет следующий вид:
$Init = (L = V)$



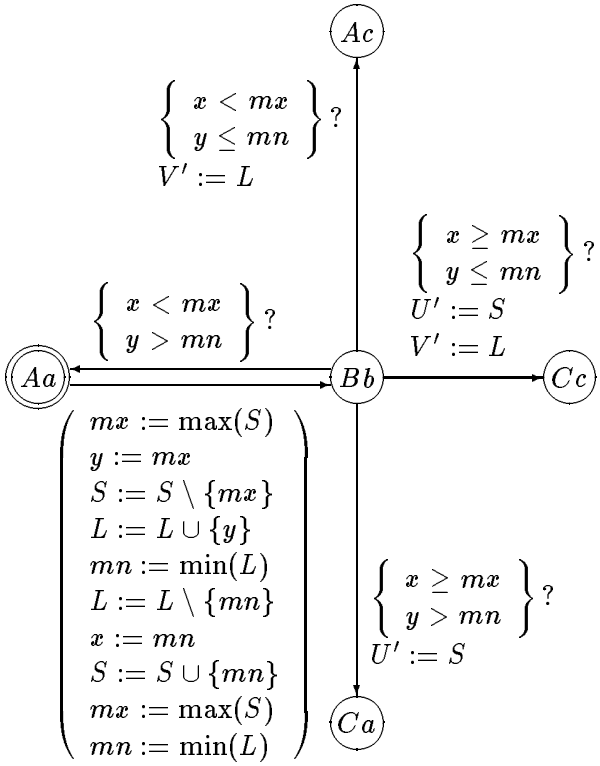
(8.3)

8.1.4 Анализ алгоритма разделения множеств

Процесс, описываемый выражением (8.1), получается путём

- выполнения над процессами (8.2) и (8.3) операций параллельной композиции и ограничения, в соответствии с определением (8.1), и
- выполнения редукции получившегося процесса.

Редуцированный процесс выглядит следующим образом:



(8.4)

На данной диаграмме видно, что у процесса (8.4) имеются такие состояния (Ac и Ca),

- из которых не выходит ни одного перехода (такие состояния называются **терминальными**), т.е., попав в которые, процесс не может продолжать свою работу,
- но попадание в эти состояния не является нормальным завершением работы процесса.

Попадание процесса в такие состояния называется **тупиком**, или **дедлоком**.

Процесс (8.1) действительно может попасть в одно из таких состояний, например, при

$$U = \{3\} \quad \text{и} \quad V = \{1, 2\}$$

(где вес каждого числа совпадает с его значением).

Тем не менее, процесс (8.1) обладает следующими свойствами:

- данный процесс всегда завершает свою работу (т.е. попадает в одно из терминальных состояний - Ac, Cc или Ca)
- после завершения работы процесса выполняются соотношения

$$\left.\begin{array}{l} S \cup L = U \cup V \\ |S| = |U|, \;\; |L| = |V| \\ S \leq L \end{array}\right\} \tag{8.5}$$

Для обоснования этих свойств мы будем использовать функцию

$$f(S, L) \stackrel{\text{def}}{=} | \{(s, l) \in S \times L \mid w(s) > w(l)\} |$$

Кроме того, при анализе последовательности операторов присваивания, выполняемых при переходе от состояния Aa к состоянию Bb, удобно представлять эту последовательность схематически как последовательность следующих действий:

1. $S \xrightarrow{y:=\max(S)} L$
   (перенесение элемента $y := \max(S)$ из S в L)
2. $L \xrightarrow{x:=\min(L)} S$
3. $mx := \max(S)$
4. $mn := \min(L)$

Нетрудно установить, что имеют место следующие утверждения.

1. Если в текущий момент времени i
   - процесс находится в состоянии Aa, и
   - значения S_i, L_i переменных S и L в этот момент удовлетворяют равенству
     $$f(S_i, L_i) = 0$$
     т.е. имеет место соотношение
     $$S_i \leq L_i$$

то $S_{i+1} = S_i$ и $L_{i+1} = L_i$. Кроме того, после выполнения перехода от состояния Aa к состоянию Bb значения переменных x, y, mx и mn будут удовлетворять соотношениям

$$y = x = mx \leq mn$$

и, таким образом, следующим переходом будет переход от состояния Bb к состоянию Cc, т.е. процесс нормально завершит свою работу.

При этом

- значения переменных U' и V' будут равны S_i и L_i соответственно,
- и, следовательно, значения переменных U' и V' будут удовлетворять требуемым соотношениям.

2. Если в текущий момент времени i

   - процесс находится в состоянии Aa, и
   - значения S_i, L_i переменных S и L в этот момент удовлетворяют неравенству
   $$f(S_i, L_i) > 0$$

   то после выполнения перехода от состояния Aa к состоянию Bb (т.е. в момент $i+1$) новые значения S_{i+1}, L_{i+1} переменных S и L будут удовлетворять неравенству

   $$f(S_{i+1}, L_{i+1}) < f(S_i, L_i) \tag{8.6}$$

   Кроме того, значения переменных x, y, mx, mn в момент $i+1$ будут удовлетворять соотношениям

   $$\begin{array}{ll} y = \max(S_i), & x = \min(L_i) \\ mx = \max(S_{i+1}), & mn = \min(L_{i+1}) \\ x < y, \quad x \leq mx, & mn \leq y \end{array}$$

   Отсюда следует, что если в момент $i+1$ процесс будет переходить из состояния Bb в одно из терминальных состояний (Ac, Cc или Ca), то это возможно

   (a) либо если $x = mx$

   (b) либо если $y = mn$

   В случае (a) имеют место соотношения

   $$S_{i+1} \leq mx = x \leq L_i$$

откуда, используя

$$x < y \quad \text{и} \quad L_{i+1} \subseteq L_i \cup \{y\}$$

получаем:

$$S_{i+1} \leq L_{i+1} \tag{8.7}$$

В случае (b) имеют место соотношения

$$S_i \leq y = mn \leq L_{i+1}$$

откуда, используя

$$x < y \quad \text{и} \quad S_{i+1} \subseteq S_i \cup \{x\}$$

получаем соотношение (8.7).

Таким образом, во всех возможных случаях попадания в какое-либо терминальное состояние имеет место соотношение

$$S \leq L$$

Истинность других соотношений, перечисленных в (8.5), усматривается непосредственно.

Из первого и второго утверждения следует, что данный процесс не может зациклиться, так как зацикливание возможно только в том случае, когда

- процесс бесконечно часто попадает в состояние Aa, и
- при каждом попадании в состояние Aa значение функции f на текущих значениях переменных S, T положительно.

Невозможность данной ситуации следует из

- неравенства (8.6), и
- свойства фундированности множества натуральных чисел (т.е. из того, что в данном множестве нет бесконечных убывающих цепей).

Читателю предлагается самостоятельно

- найти необходимые и достаточные условия, которым должны удовлетворять разделяемые множества U и V, чтобы приведённый выше алгоритм завершал свою работу с этими U и V нормально, и
- разработать такой алгоритм разделения множеств, который бы работал корректно на любых разделяемых множествах U и V.

8.2 Вычисление квадрата

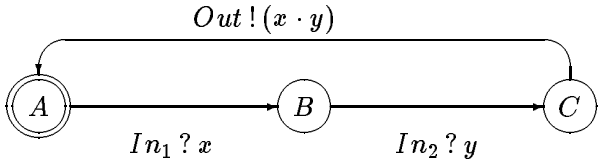
Предположим, что мы имеем систему "умножитель", у которой есть

- два входных порта с именами In_1 и In_2, и
- один выходной порт с именем Out.

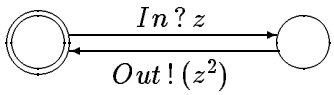
Работа умножителя заключается в том, что он периодически

- получает на свои входные порты два значения, и
- выдаёт на выходном порте их произведение.

Поведение умножителя описывается процессом Mul:



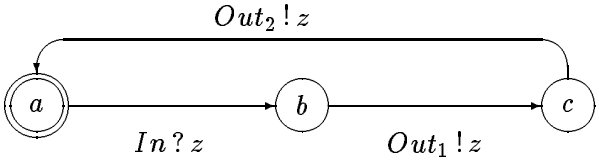
Используя этот умножитель, мы хотим построить систему "вычислитель квадрата", поведение которой описывается процессом $Square\_Spec$:



Искомую систему мы построим как композицию

- вспомогательной системы "дубликатор", имеющей
  - входной порт In, и
  - выходные порты Out_1 и Out_2

  поведение которой описывается процессом Dup:

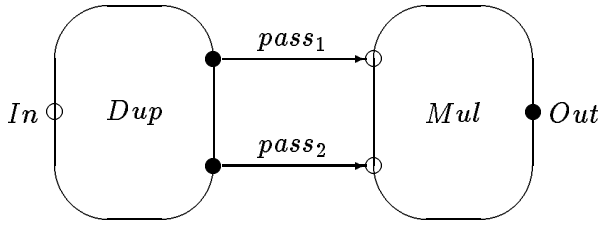


  т.е. дубликатор копирует свой вход на два выхода, и
- умножителя, который будет получать на свои входные порты те значения, которые будет выдавать дубликатор.

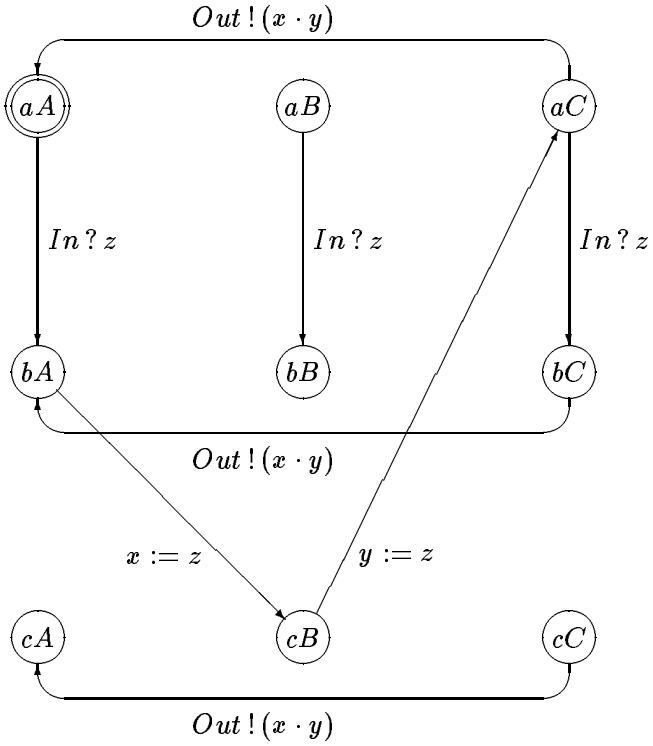
Процесс $Square$, соответствующий такой композиции, определяется следующим образом:

$$Square \stackrel{\text{def}}{=} \stackrel{\text{def}}{=} \left(\begin{array}{l} Dup[pass_1/Out_1, pass_2/Out_2] \\ \mid Mul[pass_1/In_1, pass_2/In_2] \end{array} \right) \setminus \{pass_1, pass_2\}$$

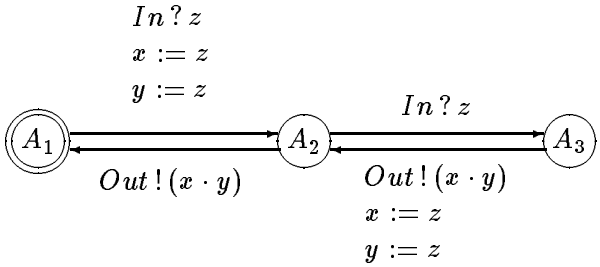
Потоковый граф процесса $Square$ имеет вид



Однако процесс $Square$ не соответствует спецификации $Square\_Spec$. Данный факт нетрудно установить при помощи построения графового представления процесса $Square$, который, согласно определению операций параллельной композиции, ограничения и переименования, имеет следующий вид:



После редукции данного процесса мы получим диаграмму



(8.8)

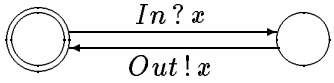
из которой видно, что

- процесс $Square$ может последовательно совершить два действия ввода, не выполняя между ними действия вывода,
- а процесс $Square\_Spec$ этого сделать не может.

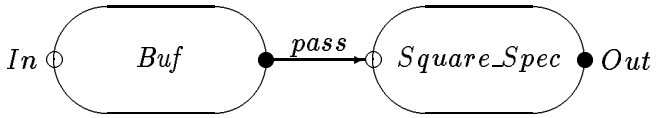
Процесс $Square$ соответствует другой спецификации:

$$Square\_Spec' \stackrel{\text{def}}{=} \left(\begin{array}{l} Buf[pass/Out] \ \mid \\ \mid Square\_Spec[pass/In] \end{array} \right) \setminus \{pass\}$$

где Buf – буфер длины 1, поведение которого изображается диаграммой



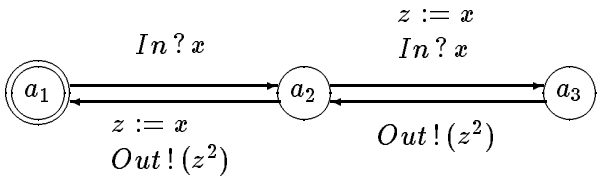
Потоковый граф процесса $Square\_Spec'$ имеет вид



Графовое представление процесса $Square\_Spec'$ получается путём

- применения операций параллельной композиции, ограничения и переименования к процессам Buf и $Square\_Spec$, и
- выполнения редукции получившегося процесса.

Редуцированный процесс $Square\_Spec'$ имеет вид

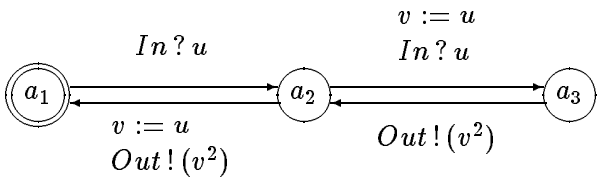


(8.9)

Соответствие процесса $Square$ спецификации $Square\_Spec'$ можно понимать как истинность соотношения

$$(8.8) \approx (8.9)$$

Доказать наблюдаемую эквивалентность процессов (8.8) и (8.9) можно, например, при помощи теоремы 34. Для того, чтобы её можно было применить, переименуем переменные в процессе (8.9) (чтобы множества переменных анализируемых процессов не пересекались). Мы получим процесс, эквивалентный процессу (8.9):



(8.10)

Для доказательства соотношения (8.8) $\approx$ (8.10) при помощи теоремы 34 мы определим функцию

$$\mu : \{A_1, A_2, A_3\} \times \{a_1, a_2, a_3\} \to Fm$$

следующим образом:

- $\mu(A_i, a_j) \stackrel{\text{def}}{=} \bot$, если $i \neq j$
- $\mu(A_1, a_1) \stackrel{\text{def}}{=} \top$
- $\mu(A_2, a_2) \stackrel{\text{def}}{=} (x = y = z = u)$
- $\mu(A_3, a_3) \stackrel{\text{def}}{=} \left\{ \begin{array}{l} x = y = v \\ z = u \end{array} \right\}$

Детальная проверка истинности соответствующих диаграмм остаётся читателю в качестве несложного упражнения.

8.3 Сети Петри

Одной из математических моделей, предназначенных для описания поведения распределённых систем, являются **сети Петри**.

Сеть Петри – это граф, множество вершин которого делится на два класса: позиции (V) и переходы (T). Каждое ребро соединяет позицию с переходом. Каждому переходу $t \in T$ соответствует два множества позиций:

- $in(t) \stackrel{\text{def}}{=} \{v \in V \mid \text{существует ребро из } v \text{ в } t\}$
- $out(t) \stackrel{\text{def}}{=} \{v \in V \mid \text{существует ребро из } t \text{ в } v\}$

Разметка сети Петри представляет собой отображение ξ вида

$$\xi : V \to \{0, 1, 2, \ldots\}$$

Функционирование сети Петри заключается в преобразовании её разметки, которое происходит в результате срабатывания переходов. Разметка ξ_0 в момент времени 0 предполагается заданной. Если в текущий момент времени $i \geq 0$ сеть имела разметку ξ_i, то сработать в этот момент может любой из переходов $t \in T$, который удовлетворяет условию

$$\forall\, v \in in(t) \quad \xi_i(v) > 0$$

Если в момент времени i сработал переход t, то разметка ξ_{i+1} в момент времени $i+1$ определяется следующим образом:

$$\begin{array}{l} \forall\, v \in in(t) \quad \xi_{i+1}(v) := \xi(v) - 1 \\ \forall\, v \in out(t) \quad \xi_{i+1}(v) := \xi(v) + 1 \\ \forall\, v \in V \setminus (in(t) \cup out(t)) \quad \xi_{i+1}(v) := \xi(v) \end{array}$$

Каждой сети Петри $\mathcal{N}$ можно сопоставить процесс $P_{\mathcal{N}}$, который моделирует работу этой сети. Компоненты процесса $P_{\mathcal{N}}$ имеют следующий вид.

- $X_{P_{\mathcal{N}}} \stackrel{\text{def}}{=} \{x_v \mid v \in V\} \cup \{at_{P_{\mathcal{N}}}\}, \quad I_{P_{\mathcal{N}}} \stackrel{\text{def}}{=} \bigwedge\limits_{v \in V} (x_v = \xi_0(v)), \quad S_{P_{\mathcal{N}}} \stackrel{\text{def}}{=} \{s^0\}$
- Пусть t – произвольный переход сети $\mathcal{N}$, и множества $in(t)$ и $out(t)$ имеют вид $\{u_1, \ldots, u_n\}$ и $\{v_1, \ldots, v_m\}$ соответственно.

  Тогда процесс $P_{\mathcal{N}}$ содержит переход из s^0 в s^0 с меткой

$$\left(\begin{array}{l} (x_{u_1} > 0) \wedge \ldots \wedge (x_{u_n} > 0)\ ? \\ x_{u_1} := x_{u_1} - 1, \ldots, x_{u_n} := x_{u_n} - 1 \\ x_{v_1} := x_{v_1} + 1, \ldots, x_{v_m} := x_{v_m} + 1 \end{array} \right)$$

8.4 Протоколы передачи данных в компьютерных сетях

8.4.1 Понятие протокола

Важным примером процессов с передачей сообщений являются **протоколы передачи данных в компьютерных сетях** (называемые ниже просто **протоколами**).

Каждый протокол можно рассматривать как совокупность нескольких взаимодействующих процессов, в которую входят

- процессы, выполняющие формирование, отправление, приём и обработку сообщений (такие процессы называются **агентами** протокола, а сообщения, пересылаемые от одного агента другому, называются **кадрами (frame)**), и
- процесс, являющийся моделью поведения **среды**, через которую передаются кадры (такую среду обычно называют **каналом связи**).

Проходя через среду, кадры могут искажаться или пропадать (например, в результате воздействия радиопомех). Поэтому каждый кадр должен содержать

- не только ту информацию, которую один агент желает передать другому, но и
- средства, позволяющие получателю этого кадра выяснить, был ли этот кадр искажён в процессе передачи.

Ниже мы рассматриваем некоторые методы распознавания искажений в кадрах. Эти методы делятся на два класса:

1. методы, позволяющие
   - не только установить факт искажения ,
   - но и определить искажённые биты кадра и исправить их

   (рассматриваются в параграфе 8.4.2), и
2. методы, позволяющие лишь установить факт искажения кадра (рассматриваются в параграфе 8.4.3).

8.4.2 Методы исправления искажений в кадрах

Методы распознавания искажений в кадрах, позволяющие

- не только установить факт искажения, но и
- определить номера искажённых битов и исправить их

используются в таких ситуациях, когда вероятность того, что каждый передаваемый кадр будет искажён при передаче, является высокой. Например, такая ситуация имеет место в беспроводной связи.

Каждый кадр представляет собой битовую строку. Искажение кадра заключается в инвертировании некоторых битов этого кадра.

Если известно максимальное количество битов кадра, которые могут быть инвертированы, то для распознавания инвертированных битов и их исправления могут быть использованы методы **кодирования с исправлением ошибок**, которые составляют одно из направлений **теории кодирования**.

В этом параграфе мы рассматриваем метод кодирования с исправлением ошибок в простейшем случае, когда в кадре может быть инвертировано не более одного бита. Данный метод называется **кодом Хэмминга** для исправления одной ошибки (существуют коды Хэмминга для исправления произвольного количества ошибок).

Идея данного метода заключается в том, что биты кадра делятся на два класса:

- информационные биты (содержащие ту информацию, которую отправитель хочет передать получателю), и
- контрольные биты (значения которых вычисляются по значениям информационных битов).

Пусть кадр f имеет вид $(b_1, \ldots, b_n)$, и

- количество информационных битов в нём равно k,
- а количество контрольных битов – r,

т.е. $n = k + r$.

Поскольку отправитель может размещать свою информацию в k информационных битах, то мы можем считать, что информация, которую отправитель передаёт получателю в кадре f, представляет собой битовую строку M размера k. Кадр, получаемый из строки M дополнением её контрольными битами, мы будем обозначать знакосочетанием $\varphi(M)$.

Для каждого кадра f обозначим знакосочетанием $\langle f \rangle$ совокупность всех кадров, получаемых из f инвертированием не более одного бита. Очевидно, что количество элементов в $\langle f \rangle$ равно $n + 1$.

Предположение о том, что в кадре $\varphi(M)$ при передаче может произойти искажение не более чем в одном бите, можно переформулировать следующим образом: получатель может получить вместо $\varphi(M)$ любой кадр из множества $\langle\varphi(M)\rangle$.

Нетрудно видеть, что для того, чтобы получатель для каждого $M \in \{0,1\}^k$ мог по произвольному кадру из $\langle\varphi(M)\rangle$ однозначно восстановить M, необходимо и достаточно выполнение условия

$$\forall\, M_1, M_2 \in \{0,1\}^k \quad M_1 \neq M_2 \;\Rightarrow\; \langle\varphi(M_1)\rangle \cap \langle\varphi(M_2)\rangle = \emptyset \tag{8.11}$$

т.е. совокупность подмножеств множества $\{0,1\}^n$ вида

$$\{\langle\varphi(M)\rangle \mid M \in \{0,1\}^k\}$$

состоит из непересекающихся подмножеств.

Поскольку

- количество таких подмножеств $= 2^k$, и
- каждое из этих подмножеств состоит из $n+1$ элемента

то для выполнения условия (8.11) должно быть верно неравенство

$$(n+1)\cdot 2^k \leq 2^n$$

которое можно переписать в виде

$$(k+r+1) \leq 2^r \tag{8.12}$$

Нетрудно доказать, что при каждом фиксированном $k > 0$ неравенство (8.12) (где r предполагается положительным) эквивалентно неравенству

$$r_0 \leq r$$

где r_0 зависит от k, и представляет собой нижнюю оценку количества контрольных битов.

Число r_0 нетрудно вычислить, когда k имеет вид

$$k = 2^m - m - 1, \qquad \text{где } m \geq 1 \tag{8.13}$$

в этом случае (8.12) можно переписать в виде

$$2^m - m \leq 2^r - r \tag{8.14}$$

что эквивалентно $m \leq r$ (т.к. функция $2^x - x$ монотонна при $x \geq 1$).

Таким образом, в данном случае нижняя оценка количества контрольных битов r_0 равна m.

Оказывается, что данная оценка достижима: существует метод кодирования с исправлением одной ошибки, при котором количество r контрольных битов совпадает с минимально возможным значением r_0. Ниже мы излагаем метод такого кодирования.

Если k имеет вид (8.13), и $r = r_0 = m$, то $n = 2^m - 1$, т.е. номера битов кадра $(b_1, \ldots, b_n)$ можно представлять кортежами из $\{0,1\}^m$: каждый номер $i \in \{1, \ldots, n\}$ представляется двоичной записью числа i, дополненной слева нулями до кортежа длины m.

Мы будем полагать, что номера контрольных битов имеют кортежную запись

$$(0 \ldots 1 \ldots 0)$$

(одна единица, и остальные нули).

Значения контрольных битов мы будем вычислять следующим образом: для каждого $j = 1, \ldots, m$ значение контрольного бита с кортежным номером $(0 \ldots 1 \ldots 0)$ (единица в j–й позиции) равно сумме по модулю 2 значений информационных битов, кортежная запись номеров которых содержит 1 в j–й позиции.

При получении кадра $(b_1, \ldots, b_n)$ мы проверяем m равенств

$$\sum_{i_j=1} b_{i_1 \ldots i_m} = 0 \qquad (j = 1, \ldots, m) \tag{8.15}$$

(где сумма рассматривается по модулю 2).

Возможны следующие случаи.

- При передаче этого кадра не было искажений.

  В этом случае все равенства (8.15) будут верны.

- При передаче этого кадра произошло инвертирование контрольного бита с кортежным номером $(0 \ldots 1 \ldots 0)$ (единица в j–й позиции).

  Нетрудно видеть, что в этом случае среди равенств (8.15) будет неверно только равенство номер j.

- При передаче этого кадра произошло инвертирование информационного бита с кортежным номером, содержащим единицы в позициях j_1, …, j_l.

  Нетрудно видеть, что в этом случае среди равенств (8.15) будут неверны только равенства с номерами j_1, …, j_l.

Таким образом, во всех случаях мы можем

- определить, было ли искажение кадра при передаче, и
- если искажение было – то вычислить номер искажённого бита.

8.4.3 Методы обнаружения искажений в кадрах

Другой класс методов анализа искажений в кадрах связан с установлением самого факта искажения.

Задача определения номеров искажённых битов имеет высокую сложность, и поэтому, если вероятность искажения кадров при передаче невысока (что имеет место при использовании медных или оптоволоконных каналов связи), то более эффективным с точки зрения сложности является повторная посылка искажённых кадров: если получатель кадра обнаруживает искажение в этом кадре, он просит отправителя послать этот кадр ещё раз.

Для сравнения сложности задач исправления искажений и обнаружения искажений рассмотрим следующий пример. Пусть известно, что в кадре может быть искажено не более одного бита. Если размер кадра $n = 1000$, то

- для *исправления* такого искажения нужно 10 контрольных битов,
- а для *обнаружения* такого искажения нужен лишь 1 контрольный бит, значение которого полагается равным чётности количества единиц в информационных битах.

Один из методов кодирования с целью обнаружения искажений заключается в следующем:

- кадр делится на k частей, и
- в каждой части назначается один контрольный бит, значение которого полагается равным чётности количества единиц в остальных битах этой части.

Если при передаче этого кадра биты искажаются равновероятно и независимо, то для каждой такой части кадра вероятность того, что

- эта часть кадра искажена, и
- тем не менее, её чётность верна (т.е. мы считаем её неискажённой)

меньше $1/2$, поэтому вероятность необнаружения искажения меньше 2^{-k}.

Другим методом кодирования с целью обнаружения искажений является **полиномиальный код** (Cyclic Redundancy Check, CRC).

Данный метод основан на рассмотрении битовых строк как многочленов над полем $\mathbf{Z}_2 = \{0, 1\}$: битовая строка вида

$$(b_k, b_{k-1}, \ldots, b_1, b_0)$$

рассматривается как многочлен

$$b_k \cdot x^k + b_{k-1} \cdot x^{k-1} + \ldots + b_1 \cdot x + b_0$$

Пусть необходимо передавать кадры размера $m+1$. Каждый такой кадр рассматривается как многочлен $M(x)$ степени $\leq m$.

Для кодирования этих кадров выбираются

- число $r < m$, и
- многочлен G степени r, имеющий вид

$$x^r + \ldots + 1$$

Многочлен G называется **образующим многочленом**.

Для каждого кадра $M(x)$ его код $T(x)$ вычисляется следующим образом. Многочлен $x^r \cdot M(x)$ делится с остатком на $G(x)$:

$$x^r \cdot M(x) = G(x) \cdot Q(x) + R(x)$$

где $R(x)$ – остаток, т.е. многочлен степени $< r$.

Кодом кадра $M(x)$ является многочлен

$$T(x) \stackrel{\text{def}}{=} G(x) \cdot Q(x)$$

Нетрудно видеть, что размер $T(x)$ больше размера $M(x)$ на r.

Обнаружение искажений при передаче кадра $T(x)$ производится путём деления полученного кадра $T'(x)$ на $G(x)$: считается, что кадр $T(x)$ был передан без искажений (т.е. полученный кадр $T'(x)$ совпадает с $T(x)$), если $T'(x)$ делится без остатка на $G(x)$.

Если кадр $T(x)$ был передан без искажений, то исходный кадр $M(x)$ можно восстановить путём представления $T(x)$ в виде суммы

$$T(x) = x^r \cdot M(x) + R(x)$$

где $R(x)$ состоит из всех мономов в $T(x)$ степени $< r$.

Если кадр $T(x)$ был передан с искажениями, то связь между $T(x)$ и $T'(x)$ можно представить в виде соотношения

$$T'(x) = T(x) + E(x)$$

где многочлен $E(x)$ называется **многочленом искажений**, и соответствует битовой строке, каждая компонента которой равна

- 1, если соответствующий бит кадра $T(x)$ был искажён, и
- 0, иначе.

Таким образом,

- если $T(x)$ был искажён в одном бите, то $E(x) = x^i$

- если $T(x)$ был искажён в двух битах, то $E(x) = x^i + x^j$,
- и т.д.

Из определений $T'(x)$ и $E(x)$ следует, что $T'(x)$ делится на $G(x)$ без остатка тогда и только тогда, когда $E(x)$ делится на $G(x)$ без остатка. Поэтому искажение, соответствующее многочлену $E(x)$, обнаруживается в том и только том случае, когда остаток от деления $E(x)$ на $G(x)$ не равен нулю.

Рассмотрим подробнее вопрос о том, какие виды искажений могут быть обнаружены при помощи данного метода.

1. Однобитные искажения обнаруживаются всегда, т.к. многочлен $E(x) = x^i$ не делится на $G(x)$ без остатка.

2. Двухбитное искажение может не обнаруживаться в том случае, когда соответствующий многочлен

$$E(x) = x^i + x^j = x^j \cdot (x^{i-j} + 1)$$

делится на G без остатка (мы предполагаем, что $i > j$):

$$x^j \cdot (x^{i-j} + 1) = G(x) \cdot Q(x) \tag{8.16}$$

Из теоремы о единственности разложения на множители многочленов над полем следует, что соотношение (8.16) влечёт соотношение

$$x^{i-j} + 1 = G(x) \cdot Q_1(x) \tag{8.17}$$

Имеет место следующий факт: если

$$G(x) = x^{15} + x^{14} + 1 \tag{8.18}$$

то для каждого $k = 1, \ldots, 32768$ многочлен $x^k + 1$ не делится на $G(x)$ без остатка. Поэтому образующий многочлен (8.18) позволяет обнаружить двухбитные искажения в кадрах длины ≤ 32768.

3. Представим многочлен искажений $E(x)$ в виде

$$E(x) = x^j \cdot (x^{k-1} + \ldots + 1) \tag{8.19}$$

Число k в этой записи называют **размером пакета ошибок**. Очевидно, что k равно размеру подстроки строки искажений (т.е. той строки, которой соответствует многочлен $E(x)$), ограниченной левым и правым единичными битами.

Обозначим знакосочетанием $E_1(x)$ второй множитель в (8.19).

Из теоремы о единственности разложения на множители многочленов над полем следует, что искажение, соответствующее многочлену (8.19), не обнаруживается в том и только том случае, когда $E_1(x)$ делится без остатка на $G(x)$. Это невозможно, если степень $E_1(x)$ меньше степени $G(x)$, т.е. если $k-1 < r$ (или $k \leq r$).

Таким образом, можно обнаружить такие искажения, размер пакета ошибок в которых $\leq r$.

4. Если размер пакета ошибок $k = r+1$, то многочлен $E_1(x)$ (см. предыдущий пункт) делится без остатка на $G(x)$ в том и только том случае, когда $E_1(x) = G(x)$.

   Вероятность такого совпадения равна $2^{-(r-1)}$.

   Таким образом, если размер пакета ошибок равен $r+1$, то вероятность необнаружения такого искажения равна $2^{-(r-1)}$.

5. Можно доказать, что если размер пакета ошибок $k > r+1$, то вероятность необнаружения такого искажения $< 2^{-r}$.

6. Если

   - искажено нечётное количество битов, т.е. $E(x)$ состоит из нечётного количества мономов, и
   - $G = (x+1) \cdot G_1$

   то такое искажение обнаруживается, т.к. если бы было верно равенство

   $$E(x) = G(x) \cdot Q(x)$$

   для некоторого многочлена $Q(x)$, то, в частности было бы верно равенство

   $$E(1) = G(1) \cdot Q(1) \tag{8.20}$$

   чего не может быть, так как левая часть в (8.20) равна 1, а правая – 0.

В заключение отметим, что в стандарте IEEE 802 используется образующий многочлен $G(x)$ вида

$$\begin{aligned} G(x) = \; & x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + \\ & + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1 \end{aligned}$$

он обнаруживает искажения, в которых размер пакета ошибок ≤ 32, или искажено нечётное количество битов.

8.4.4 Пример протокола

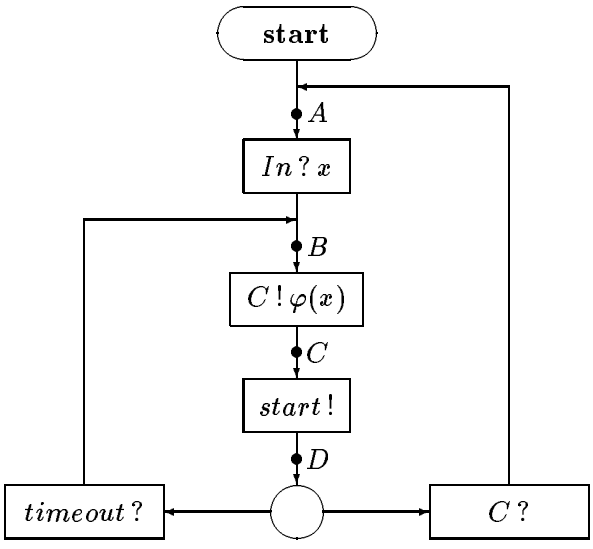
Рассмотрим простой пример протокола, который состоит из двух агентов: **отправителя и получателя**. Задачей протокола является организация доставки кадров от отправителя получателю через ненадёжный канал связи (который может искажать и терять передаваемые кадры).

Работа протокола происходит следующим образом.

1. **Отправитель** получает сообщения от процесса, не входящего в протокол, который называется **сетевым уровнем (СУ) отправителя**. Эти сообщения называются **пакетами**. Задача отправителя заключается в периодическом выполнении следующих действий:

   - получить очередной пакет от своего СУ
   - сформировать из этого пакета кадр
   - послать этот кадр в канал связи и включить таймер
   - если таймер пришлёт сигнал timeout (который означает, что время ожидания подтверждения посланного кадра закончилось, и, видимо, этот кадр до получателя не дошёл), то послать этот кадр ещё раз
   - если придёт подтверждение от получателя, то это означает, что текущий кадр дошёл до него успешно, и можно
     - получать следующий пакет от своего СУ,
     - формировать из него кадр,
     - и т.д.

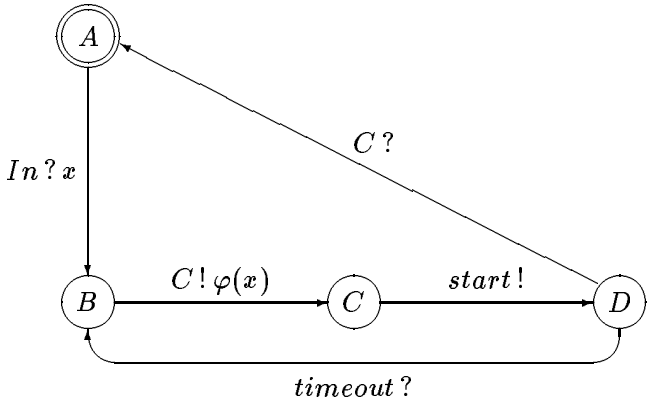
   Блок-схема, представляющая описанное выше поведение, выглядит следующим образом:



Операторы, входящие в эту блок-схему, имеют следующий смысл.

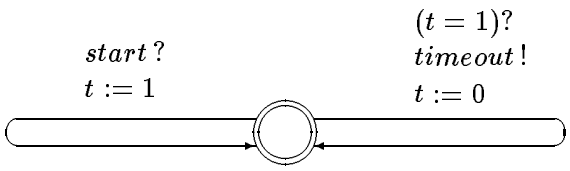
- $In\,?\,x$ – получение отправителем очередного пакета от своего СУ, и запись этого пакета в переменную x
- $C\,!\,\varphi(x)$ – отправление в канал связи кадра $\varphi(x)$
- $start\,!$ – включение таймера
- $timeout\,?$ – получение от таймера сигнала timeout
- $C\,?$ – получение из канала связи сигнала подтверждения

Процесс, представляемый этой блок-схемой, обозначается знакосочетанием Sender и имеет следующий вид:



Процесс отправителя является параллельной композицией (с ограничением)

- процесса Sender, и
- процесса Timer, представляющего поведение таймера, и имеющего вид



(8.21)

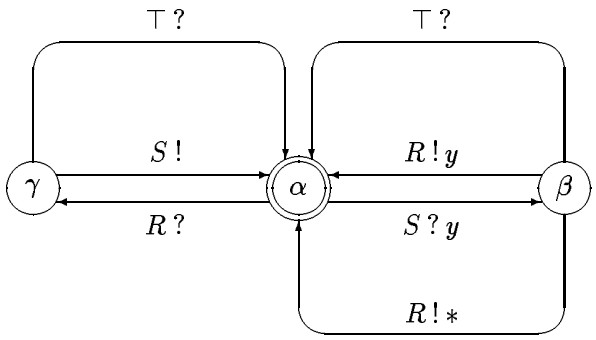
Начальное условие процесса Timer: $t = 0$.

В этой модели мы не детализируем величину промежутка времени между запуском таймера (действие $start$) и посылкой им сигнала $timeout$.

2. **Канал связи** (называемый ниже просто **каналом**) в каждый момент времени может содержать не более одного кадра или сигнала. Он может выполнять следующие действия:

   - получение кадра от отправителя, и
     - пересылка этого кадра получателю, или
     - пересылка получателю искажённого кадра, или
     - потеря кадра
   - получение сигнала подтверждения от получателя, и
     - пересылка этого сигнала отправителю, или
     - потеря сигнала.

   Поведение канала представляется следующим процессом:



(8.22)

   В этом процессе мы используем следующую абстракцию: символ '$*$' означает "искажённый кадр". Мы не уточняем, как именно искажаются кадры в канале. Каждый кадр, поступивший в канал

   - либо передаётся из канала в неизменном виде получателю,
   - либо пребразуется в абстрактное значение '$*$', и это значение передаётся из канала получателю
   - либо пропадает, что выражается переходом процесса (8.22) с меткой $\top?$

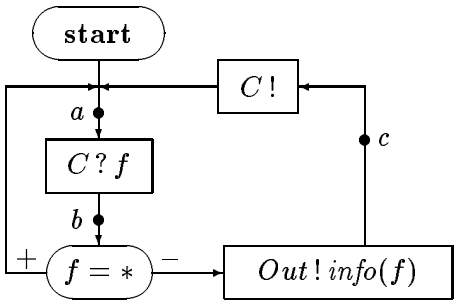
3. **Получатель** периодически выполняет следующие действия:

   - получение из канала очередного кадра
   - проверка наличия искажений в кадре
   - если кадр не искажён, то
     - извлечение из кадра пакета,

– передача этого пакета процессу, называемому **сетевой уровень (СУ) получателя** (этот процесс не входит в протокол)
– посылка отправителю через канал сигнала подтверждения

- если кадр искажён, то получатель его игнорирует (предполагая, что отправитель, не дождавшись подтверждения, пошлёт это кадр ещё раз)

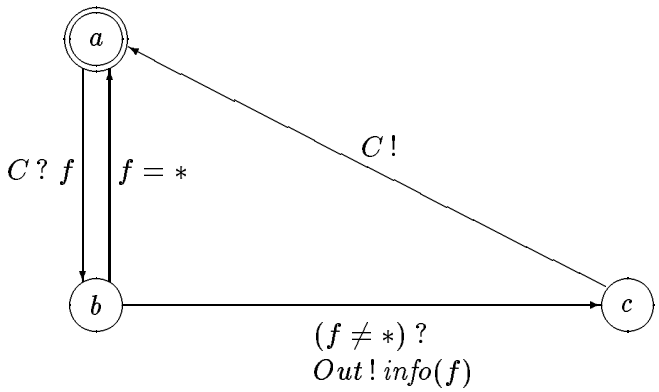
Блок-схема, представляющая описанное выше поведение, выглядит следующим образом:



Операторы, входящие в эту блок-схему, имеют следующий смысл.

- $C\,?\,f$ – получение из канала очередного кадра, и запись его в переменную f
- $(f = *)$ – проверка наличия искажения в кадре f
- $Out\,!\,info(f)$ – отправление пакета $info(f)$, извлечённого из кадра f, своему СУ
- $C\,!$ – посылка сигнала подтверждения

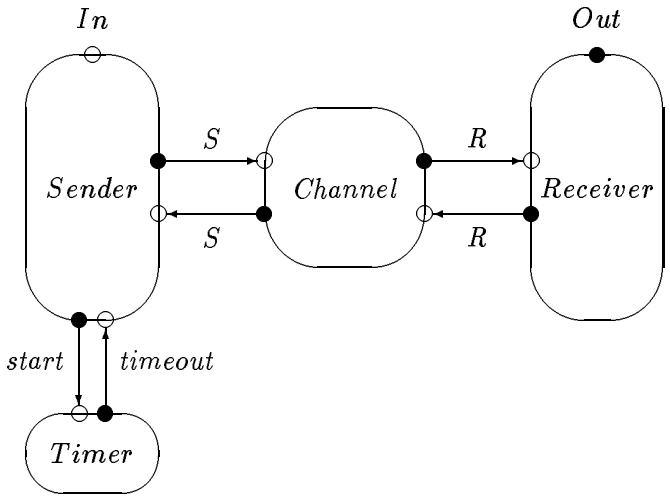
Процесс, представляемый этой блок-схемой, обозначается знакосочетанием Receiver и имеет следующий вид:



Процесс *Protocol*, соответствующий всему протоколу, определяется как параллельная композиция (с ограничением) вышеперечисленных процессов:

$$Protocol \stackrel{\text{def}}{=} \left(\begin{array}{l} Sender\,[S/C]\,| \\ Timer\,| \\ Channel\,| \\ Receiver\,[R/C] \end{array} \right) \setminus \{S, R, start, timeout\} \tag{8.23}$$

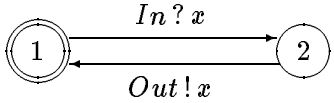
Потоковый граф этого процесса имеет следующий вид:



(8.24)

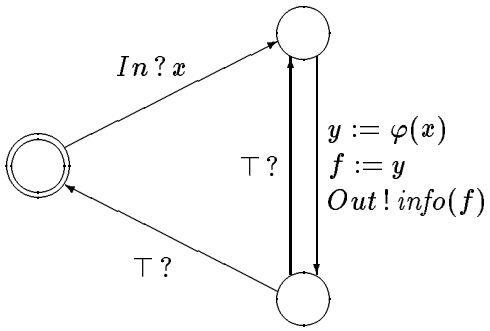
Для того, чтобы можно было анализировать корректность этого протокола, необходимо точно определить спецификацию, которой он должен соответствовать.

Если мы хотим специфицировать только свойства внешних действий, выполняемых этим протоколом (т.е. действий вида $In\,?\,v$ и $Out\,!\,v$), то спецификация может выглядеть следующим образом: поведение данного протокола совпадает с поведением буфера размера 1, т.е. процесс *Protocol* наблюдаемо эквивалентен процессу *Buf*, который имеет вид

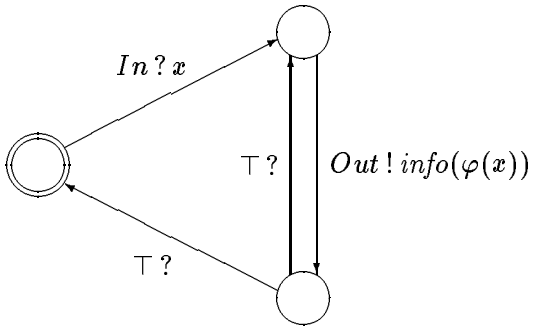


(8.25)

При построении процесса с СО, соответствующего выражению (8.23), и его редукции, получается диаграмма



которая наблюдаемо эквивалентна диаграмме

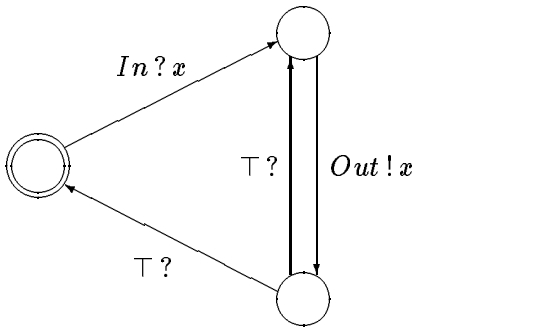


(8.26)

Если мы предположим, что функция *info* извлечения пакетов из кадров является обратной к функции φ формирования кадров, т.е. для каждого пакета x имеет место соотношение

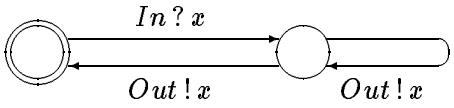
$$info(\varphi(x)) = x$$

то диаграмму (8.26) можно перерисовать следующим образом:



(8.27)

Процесс (8.27) можно редуцировать, в результате чего получается процесс



(8.28)

При сопоставлении процессов (8.28) и (8.25) можно заключить, что данные процессы не могут быть эквивалентными ни в каком приемлемом смысле. Например,

- процесс (8.25) после получения пакета x может только
  - передать этот пакет СУ получателя, и
  - перейти в состояние ожидания следующего пакета
- в то время как процесс (8.28) может после получения пакета x передать этот пакет СУ получателя несколько раз.

Такая повторная передача может происходить, например, при следующем варианте работы протокола.

- Первый кадр, посланный отправителем, доходит до получателя успешно.
- Получатель
  - пересылает пакет, извлечённый из этого кадра, своему СУ, и
  - посылает отправителю через канал подтверждение.
- Это подтверждение пропадает в канале.
- Отправитель, не дождавшись подтверждения, посылает этот кадр ещё раз, и этот кадр снова доходит успешно.
- Получатель воспринимает этот кадр как новый. Он
  - пересылает пакет, извлечённый из этого кадра, своему СУ, и
  - посылает отправителю через канал подтверждение, которое опять пропадает канале.
- и т.д.

Эта ситуация может возникнуть потому, что в данном протоколе отсутствует механизм, с помощью которого получатель мог бы отличить новый кадр от переданного повторно.

В следующем параграфе мы приводим пример протокола, в котором этот механизм присутствует. Для такого протокола уже можно формально доказать его соответствие спецификации (8.25).

8.4.5 Протокол с чередующимися битами

Протокол с чередующимися битами (называемый в англоязычной литературе словосочетанием **Alternating Bit Protocol**, или, сокращённо, **ABP**) предназначен для решения той же задачи, что и предыдущий протокол: доставка кадров от отправителя получателю через ненадёжный канал связи (который может искажать и терять передаваемые кадры).

Механизм, с помощью которого получатель может отличить новый кадр от переданного повторно, реализован в данном протоколе следующим образом: среди переменных отправителя и получателя присутствуют булевы переменные s и r соответственно, значения которых имеют следующий смысл:

- значение переменной s равно чётности номера очередного кадра, которого пытается послать отправитель, и
- значение переменной r равно чётности номера очередного кадра, которого ожидает получатель.

В начальный момент значения s и r равны 0 (первый кадр имеет номер 0).

Как и в предыдущем протоколе, в этом протоколе используется абстрактное значение '$*$', обозначающее искажённый кадр.

Работа протокола происходит следующим образом.

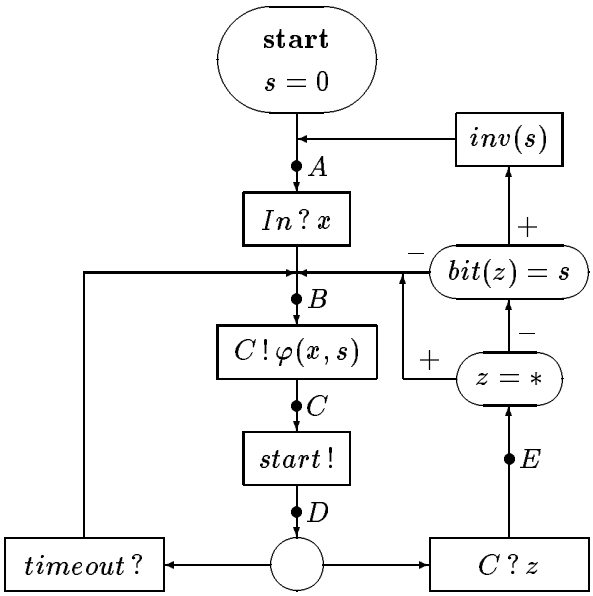
1. **Отправитель**, получив очередной пакет от своего СУ,
   - записывает его в переменную x,
   - формирует кадр, который представляет собой значение некоторой кодирующей функции φ на паре (x, s),
   - посылает этот кадр в канал,
   - запускает таймер
   - после чего он ожидает подтверждение посланного кадра.

   Если у отправителя истекает время ожидания, и он не получает подтверждения от получателя, то он повторно посылает уже посланный кадр.

   Если же он получает подтверждение, которое представляет собой неискажённый кадр, содержащий бит, то он анализирует значение этого бита: если оно совпадает с текущим значением s, то отправитель
   - инвертирует значение переменной s (используя для этого функцию $inv(x) = 1 - x$), и
   - начинает новый цикл своей работы.

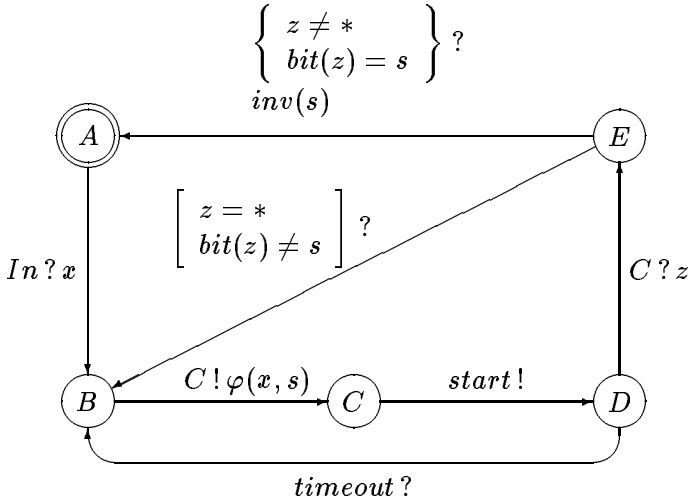
В противном случае он опять посылает уже посланный кадр.

Блок-схема, представляющая такое поведение, выглядит следующим образом:



Процесс, представляемый этой блок-схемой, обозначается знакосочетанием Sender и имеет следующий вид:

$Init = (s = 0)$

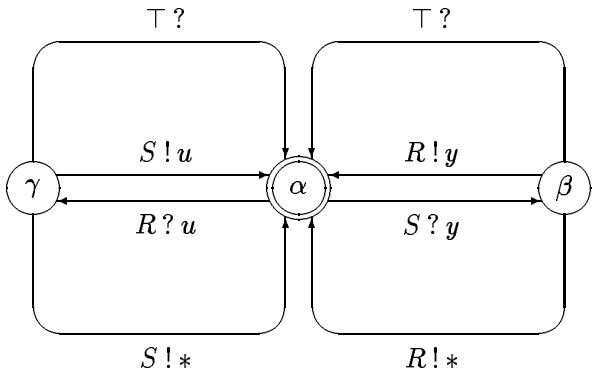


Процесс отправителя является параллельной композицией (с ограничением) процесса Sender, и процесса Timer.

2. **Канал** в каждый момент времени может содержать не более одного кадра. Он может выполнять следующие действия:

   - получение кадра от отправителя, и
     - пересылка этого кадра получателю, или
     - пересылка получателю искажённого кадра, или
     - потеря кадра
   - получение кадра с подтверждением от получателя, и
     - пересылка этого кадра отправителю, или
     - пересылка отправителю искажённого кадра, или
     - потеря кадра.

   Поведение канала представляется следующим процессом:



(8.29)

3. **Получатель** при получении очередного кадра из канала

   - проверяет, не искажён ли этот кадр,
   - и если не искажён, то извлекает из него пакет и связанный с ним бит при помощи функций *info* и *bit*, обладающих следующими свойствами:

$$info(\varphi(x,b)) = x, \quad bit(\varphi(x,b)) = b$$

   Получатель проверяет, совпадает ли значение бита, извлечённого из кадра, с ожидаемым значением, которое содержится в переменной r.

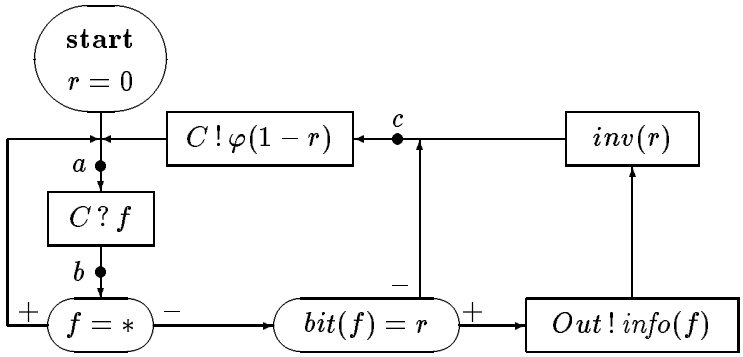
   Если проверка дала положительный результат, то получатель

   - передаёт пакет, извлечённый из этого кадра, своему СУ
   - инвертирует значение переменной r, и
   - посылает отправителю через канал подтверждение.

Если проверка дала отрицательный результат, то получатель посылает кадр подтверждения с неверным битом (что заставит отправителя послать свой кадр ещё раз).

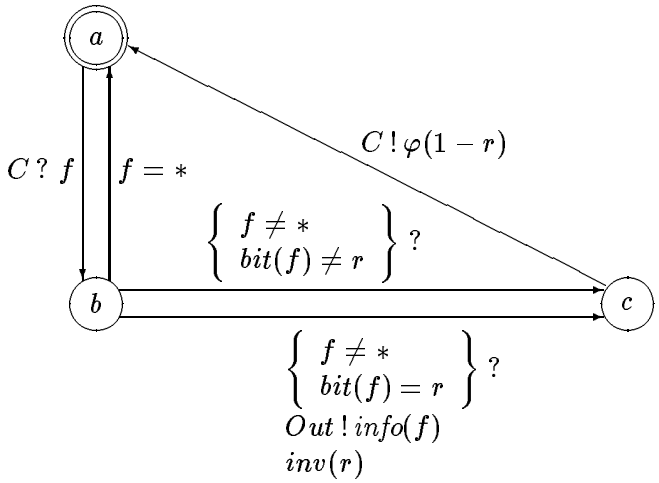
Если же кадр искажён, то получатель его игнорирует (предполагая, что отправитель, не дождавшись подтверждения, пошлёт это кадр ещё раз)

Блок-схема, представляющая описанное выше поведение, выглядит следующим образом:



Процесс, представляемый этой блок-схемой, обозначается знакосочетанием Receiver и имеет следующий вид:

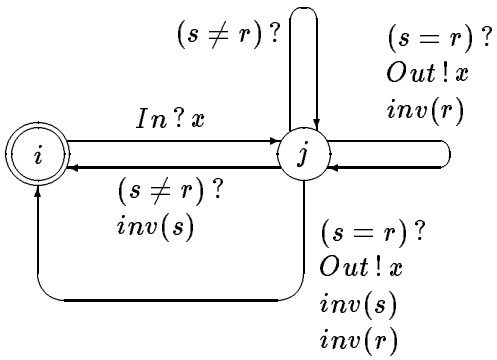
$Init = (r = 0)$



Процесс *Protocol*, соответствующий всему протоколу, определяется так же, как и в предыдущем пункте, выражением (8.23). Потоковый граф данного процесса имеет вид (8.24).

Спецификация данного протокола тоже имеет такой же вид, т.е. представляет собой процесс (8.25).

При построении процесса с СО, соответствующего данному протоколу, и его последующей редукции, получается следующая диаграмма:



(8.30)

Доказать наблюдаемую эквивалентность процессов (8.25) и (8.30) можно, например, при помощи теоремы 34, определив функцию μ вида

$$\mu : \{1,2\} \times \{i,j\} \to Fm$$

следующим образом:

$$\begin{cases} \mu(1,i) \stackrel{\text{def}}{=} (s = r) \\ \mu(2,i) \stackrel{\text{def}}{=} \bot \\ \mu(1,j) \stackrel{\text{def}}{=} (s \neq r) \\ \mu(2,j) \stackrel{\text{def}}{=} (s = r) \end{cases}$$

8.4.6 Двунаправленная передача

Рассмотренные выше протоколы относятся к классу **симплексных** протоколов: они реализуют однонаправленную передачу информационных кадров (т.е. кадров, содержащих пакеты) от одного агента к другому.

В большинстве ситуаций пересылки данных между двумя агентами требуется **двунаправленная передача**, т.е. передача информационных кадров в обоих направлениях. В данном случае каждый из агентов выступает как в роли отправителя, так и в роли получателя. Протоколы, реализующие двунаправленную передачу, называются **дуплексными** протоколами.

В дуплексных протоколах посылка подтверждений может быть совмещена с посылкой информационных кадров: если агент B успешно принял информационный кадр f от агента A, он может послать своё подтверждение получения кадра f не отдельным кадром, а в составе своего информационного кадра.

Ниже мы излагаем примеры некоторых дуплексных протоколов.

8.4.7 Дуплексный протокол с чередующимися битами

Простейшим дуплексным протоколом является излагаемый в этом параграфе **дуплексный протокол с чередующимися битами**. Данный протокол является обобщением протокола ABP.

В этом протоколе тоже участвуют два агента, но, в отличие от протокола ABP, поведение каждого из агентов описывается одним и тем же процессом, который совмещает в себе процессы отправителя и получателя из протокола ABP.

Каждый кадр f, пересылаемый каким-либо из этих агентов, содержит пакет x и два бита: s и r, где

- s имеет тот же смысл, что и в протоколе ABP: это бит, сопоставленный пакету x,
- r является битом подтверждения последнего полученного неискажённого кадра.

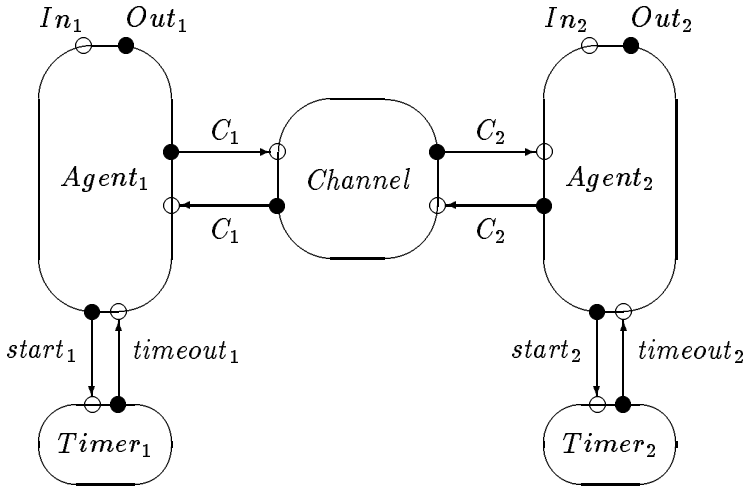
Для формирования кадров используется функция φ. Для извлечения пакетов и битов из кадров используются функции $info$, seq и ack, обладающие следующими свойствами:

$$\begin{aligned} info(\varphi(x,s,r)) &= x \\ seq(\varphi(x,s,r)) &= s \\ ack(\varphi(x,s,r)) &= r \end{aligned}$$

Также агенты используют функцию inv для инвертирования значений битовых переменных.

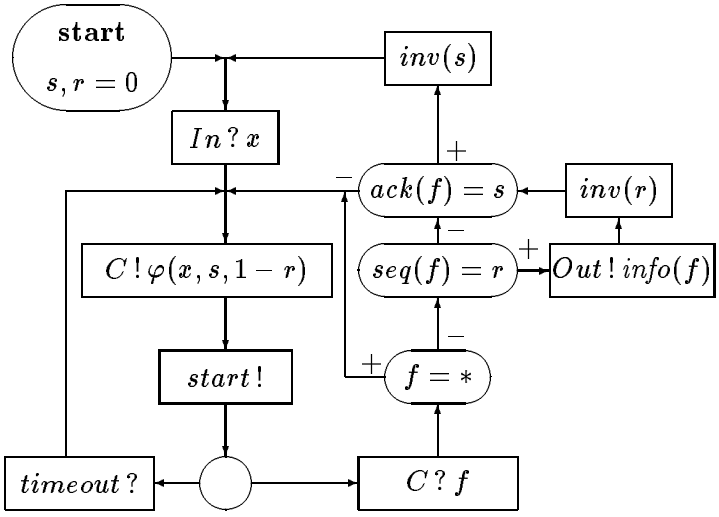
С каждым из агентов связан свой таймер, поведение которого описывается процессом $Timer$, представленным диаграммой (8.21).

Потоковый граф этого протокола имеет следующий вид:



(8.31)

Процесс, описывающий поведение каждого из агентов, представляется в виде следующей блок-схемы:



Блок-схема, описывающая поведение конкретного агента, получается из этой блок-схемы приписыванием соответствующего индекса к переменным и именам, входящим в эту блок-схему.

Поведение канала представляется процессом (8.29), к которому применено переименование

$$[\,C_1/S,\ C_2/R\,]$$

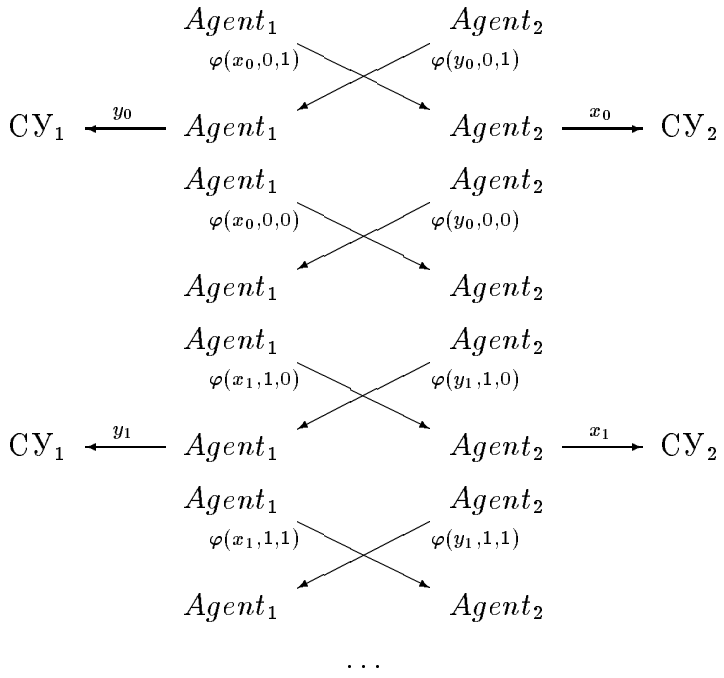
Таким образом, каждый агент в этом протоколе посылает свой следующий пакет только после получения подтверждения своего предыдущего пакета.

Нормальную работу данного протокола можно изобразить следующей диаграммой:

$$\begin{array}{cccccc}
 & & Agent_1 & \xrightarrow{\varphi(x_0,0,1)} & Agent_2 & \xrightarrow{x_0} \text{СУ}_2 \\
\text{СУ}_1 & \xleftarrow{y_0} & Agent_1 & \xleftarrow{\varphi(y_0,0,0)} & Agent_2 & \\
 & & Agent_1 & \xrightarrow{\varphi(x_1,1,0)} & Agent_2 & \xrightarrow{x_1} \text{СУ}_2 \\
\text{СУ}_1 & \xleftarrow{y_1} & Agent_1 & \xleftarrow{\varphi(y_1,1,1)} & Agent_2 & \\
 & & Agent_1 & \xrightarrow{\varphi(x_2,0,1)} & Agent_2 & \xrightarrow{x_2} \text{СУ}_2 \\
\text{СУ}_1 & \xleftarrow{y_2} & Agent_1 & \xleftarrow{\varphi(y_2,0,0)} & Agent_2 & \\
 & & \multicolumn{4}{c}{\ldots}
\end{array}$$

Однако, если оба агента одновременно вышлют друг другу начальные кадры, то работа протокола будет не вполне нормальной (хотя, тем не менее, передача пакетов в данном случае является корректной), и может быть

изображена следующей диаграммой:



...

Читателю предлагается самостоятельно

- – определить процесс $Spec$, являющийся спецификацией этого протокола, и
  – доказать соответствие этого протокола спецификации $Spec$,
- – модифицировать определённый в этом параграфе протокол таким образом, чтобы при любом варианте работы модифицированного протокола не возникало аномальных эффектов, аналогичных приведённому выше, и
  – доказать корректность (т.е. соответствие спецификации $Spec$) модифицированного протокола.

8.4.8 Двунаправленная конвейерная передача

Дуплексный протокол с чередующимися битами является практически приемлемым только в том случае, когда длительность пересылки кадров по каналу пренебрежимо мала.

Если же время прохождения кадра по каналу большое, то лучше использовать **конвейерную** передачу, при которой отправитель может послать несколько кадров подряд, не дожидаясь подтверждений. Ниже мы рассматриваем два протокола двунаправленной конвейерной передачи, называемые **протоколами скользящего окна (ПСО)** (sliding window).

Данные протоколы являются развитием дуплексного протокола с чередующимися битами, изложенного в параграфе 8.4.7.

В этих протоколах

- тоже участвуют два агента, поведение каждого из которых описывается одним и тем же процессом, совмещающим в себе функции отправителя и получателя,
- аналогом бита, связанного с передаваемым кадром, является элемент множества вычетов $\mathbf{Z}_n = \{0, \ldots, n-1\}$, где n – некоторое фиксированное число вида 2^k.

Элемент множества $\mathbf{Z}_n$, связанный с кадром, называется **номером** этого кадра. Отметим, что номер кадра и порядковый номер кадра – это разные понятия: порядковые номера уникальны, а номера циклически повторяются.

8.4.9 Протокол скользящего окна с возвратом

Первый из рассматриваемых нами ПСО называется **ПСО с возвратом** (или **ПСО с повторной передачей**).

Процесс, описывающий поведение агента этого ПСО, содержит среди своих переменных массив $x[n]$, в компонентах которого могут содержаться отправленные, но ещё не подтверждённые пакеты. Совокупность компонентов массива x, в которых содержатся такие пакеты в текущий момент времени, называется **окном**. С окном связаны три переменные этого процесса:

- b – нижняя граница окна,
- s – верхняя граница окна, и
- w – количество пакетов в окне.

Значения переменных b, s и w принадлежат множеству $\mathbf{Z}_n$.

В начальный момент времени окно является пустым, и значения переменных b, s и w равны 0.

Добавление нового пакета к окну происходит путём выполнения следующих операций:

- данный пакет записывается в компоненту $x[s]$, и число s считается номером этого пакета
- верхняя граница окна s увеличивается на 1 по модулю n, т.е. новое значение s полагается равным
  - $s + 1$, если $s < n - 1$, и
  - 0, если $s = n - 1$,

- количество пакетов в окне w увеличивается на 1.

Удаление пакета из окна происходит следующим образом:

- нижняя граница окна b увеличивается на 1 по модулю n, и
- количество пакетов в окне w уменьшается на 1

т.е. удаляется тот пакет, номер которого равен нижней границе окна.

Для упрощения понимания операций работы с окном можно использовать следующую образную аналогию:

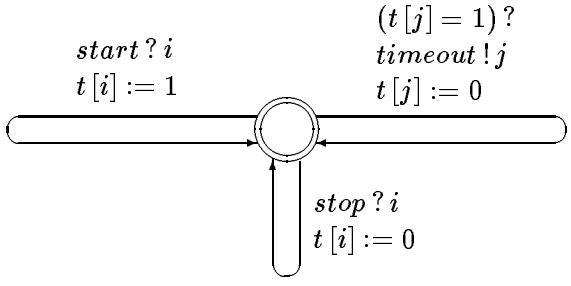
- совокупность компонентов массива x можно рассматривать как "свёрнутую в кольцо" (после компоненты $x[n-1]$ идёт компонента $x[0]$)
- в каждый момент времени окно представляет собой связное подмножество этого кольца,
- во время работы процесса окно перемещается по этому кольцу в одном и том же направлении.

Если окно достигает максимального размера $(n-1)$, то агент не принимает новые пакеты от своего СУ до тех пор, пока размер окна не уменьшится. Возможность приёма новых пакетов определяется булевой переменной $enable$: если её значение равно 1, то агент может принимать новые пакеты от своего СУ, а если 0, то не может.

Когда агент получает подтверждение пакета, номер которого равен нижней границе окна, этот пакет удаляется из окна.

С каждой компонентой массива x связан таймер, при помощи которого определяется время ожидания подтверждения получения пакета, содержащегося в этой компоненте. Совокупность всех этих таймеров рассматривается как один процесс $Timers$, который имеет массив $t\,[n]$ булевых переменных. Данный процесс определяется следующим образом:

$Init = (t = (0, \ldots, 0))$



(8.32)

Правую стрелку в этой диаграмме следует понимать как сокращённое обозначение для совокупности из n переходов, метка каждого из которых получается заменой в метке этой стрелки символа j на любое из значений из множества $\mathbf{Z}_n$.

Отметим, что в данном процессе наряду с операторами запуска таймеров и посылки ими сигнала тайм-аута присутствует также оператор $stop\,?\,i$, выполнение которого досрочно завершает работу соответствующего таймера.

ПСО с возвратом имеет следующие особенности.

- Когда заканчивается лимит времени ожидания подтверждения какого-либо пакета, агент начинает передавать повторно все пакеты из своего окна.
- Когда поступает подтверждение получения какого-либо пакета, все предыдущие пакеты в окне тоже считаются подтвержденными (даже если их подтверждения не дошли).

Каждый кадр f, пересылаемый каким-либо из агентов этого протокола, содержит пакет x и два целых числа: s и r, где

- s – номер, сопоставленный пакету x (который по определению равен номеру кадра f),
- r – номер последнего полученного неискажённого кадра.

Для формирования кадров используется функция φ. Для извлечения пакетов и номеров из кадров используются функции $info$, seq и ack, обладающие следующими свойствами:

$$\begin{aligned} info(\varphi(x,s,r)) &= x \\ seq(\varphi(x,s,r)) &= s \\ ack(\varphi(x,s,r)) &= r \end{aligned}$$

Описание процесса, представляющего поведение агента данного ПСО, мы даём в блок-схемном виде, по которому несложно построить блок–схему этого процесса.

В данном описании мы используем следующие обозначения.

- Символы $\underset{n}{+}$ и $\underset{n}{-}$ обозначают сложение и вычитание по модулю n.
- Символ r обозначает переменную с множеством значений $\mathbf{Z}_n$. Значение переменной r равно номеру ожидаемого кадра.

  Агент посылает своему СУ пакет, извлечённый только из такого кадра f, у которого номер $seq(f)$ совпадает со значением этой переменной r.

  Пакеты из кадров с другими значениями $seq(f)$ игнорируются, у таких кадров учитывается лишь компонента $ack(f)$.

- Знакосочетание $send$ является сокращённым обозначением следующей группы операторов:

$$send = \left\{ \begin{array}{l} C \,!\, \varphi(x[s], s, r \underset{n}{-} 1) \\ start \,!\, s \\ s := s \underset{n}{+} 1 \end{array} \right\}$$

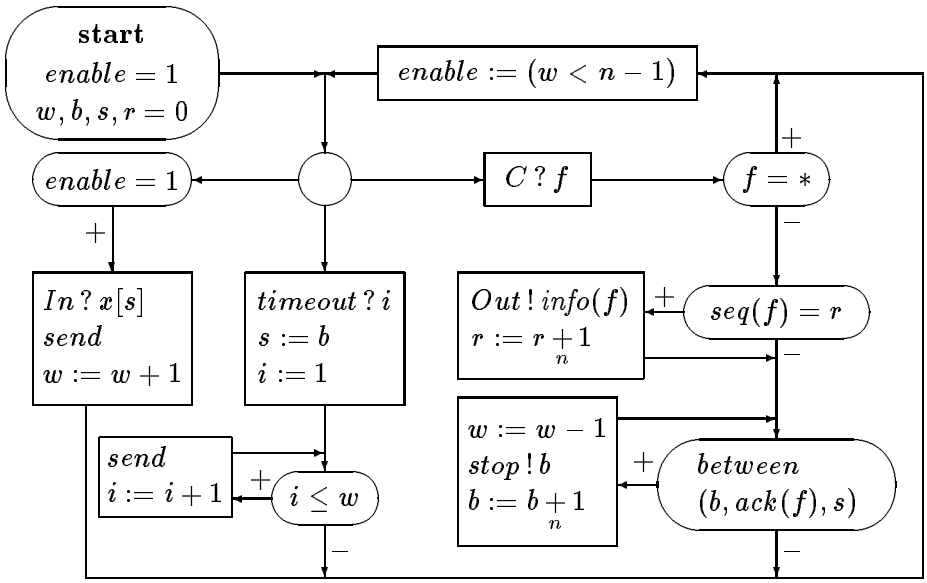
- Знакосочетание $between(a, b, c)$ является сокращённым обозначением формулы

$$\Big(\; a \le b < c \; \Big) \vee \Big(\; c < a \le b \; \Big) \vee \Big(\; b < c < a \; \Big) \qquad (8.33)$$

- Знакосочетание $enable := (w < n - 1)$ является сокращённой записью оператора

$$\textbf{if} \;\; (w < n - 1) \;\; \textbf{then} \;\; enable := 1 \;\; \textbf{else} \;\; enable := 0$$

Процесс, представляющий поведение агента данного ПСО, имеет следующий вид:



Читателю предлагается самостоятельно

- определить процесс “канал” для этого протокола
(канал может содержать несколько кадров, которые могут не только искажаться и пропадать, но ещё и переупорядочиваться)

- определить спецификацию $Spec$ этого протокола

- доказать соответствие этого протокола спецификации $Spec$
- исследовать этот протокол на наличие аномальных эффектов, аналогичных тому эффекту, который был изложен в параграфе 8.4.7
  (если аномальные эффекты присутствуют, то модифицировать этот протокол так, чтобы таких эффектов не было, и доказать корректность модифицированного протокола)

В заключение отметим, что данный ПСО неэффективен при большом количестве ошибок при передаче кадров.

8.4.10 Протокол скользящего окна с выборочным повтором

Второй ПСО отличается от предыдущего тем, что у агента этого ПСО имеется не одно, а два окна.

1. Первое окно имеет те же функции, которые имеет окно первого ПСО (данное окно называется **посылающим окном**).

   Максимальный размер посылающего окна равен $m \stackrel{\text{def}}{=} n/2$, где число n имеет тот же статус, который описан в параграфе 8.4.8 (в частности, номера кадров являются элементами $\mathbf{Z}_n$).

2. Второе окно (называемое **принимающим окном**) предназначено для размещения пакетов, поступивших от другого агента, которые пока не могут быть переданы СУ, потому что ещё не пришли некоторые пакеты с меньшими номерами.
   (агент-получатель должен передавать принятые пакеты своему СУ в том же самом порядке, в котором они поступили к агенту-отправителю от его СУ)

   Принимающее окно имеет фиксированный размер $m = n/2$.

Каждый кадр f, пересылаемый каким-либо из агентов этого протокола, содержит 4 компоненты:

1. k – вид кадра,
   данная компонента может принимать одно из трёх значений:
   - $data$ (информационный кадр)
   - ack (кадр, содержащий лишь подтверждение)
   - nak (кадр, содержащий запрос на повторную передачу)
2. x – пакет

3. s – номер этого кадра

4. r – номер последнего полученного неискажённого кадра.

Если значение первой компоненты кадра равно ack или nak, то вторая и третья компоненты этого кадра имеют фиктивный характер.

Для формирования кадров используется функция φ.

Для извлечения компонентов из кадров используются функции $kind$, $info$, seq и ack, обладающие следующими свойствами:

$$\begin{aligned} kind(\varphi(k,x,s,r)) &= k \\ info(\varphi(k,x,s,r)) &= x \\ seq(\varphi(k,x,s,r)) &= s \\ ack(\varphi(k,x,s,r)) &= r \end{aligned}$$

Процесс, описывающий поведение агента данного ПСО, имеет следующие переменные.

1. Массивы $x[m]$ и $y[m]$, предназначенные для размещения посылающего окна и принимающего окна соответственно.

2. Переменные $enable, b, s, w$, имеющие

   - те же множества значений, и
   - тот же смысл

   которые они имеют в предыдущем протоколе.

3. Переменные, связанные с принимающим окном:

   - r – нижняя граница принимающего окна
   - u – верхняя граница принимающего окна

   значения переменных r и u принадлежат множеству $\mathbf{Z}_n$.

   Если в принимающем окне присутствует пакет, номер которого равным нижней границе принимающего окна (r), то агент

   - передаёт этот пакет своему СУ, и
   - увеличивает на 1 (по модулю n) значения r и u.

4. Булевский массив $arrived[m]$, компоненты которого имеют следующий смысл: $arrived[i] = 1$ тогда и только тогда, когда в i–й компоненте принимающего окна содержится пакет, пока ещё не переданный СУ.

5. Булева переменная no_nak, используемая в следующих целях.

   Если агент получает

- искажённый кадр, или
- кадр, номер которого отличен от нижней границы принимающего окна (r)

то он посылает своему коллеге запрос на повторную передачу кадра, номер которого равен r.

Данный запрос называется NAK (Negative Acknowledgement).

Булева переменная no_nak используется для того, чтобы избежать нескольких запросов на повторную передачу одного и того же кадра: эта переменная имеет значение 1, если NAK для кадра с номером r еще не был послан.

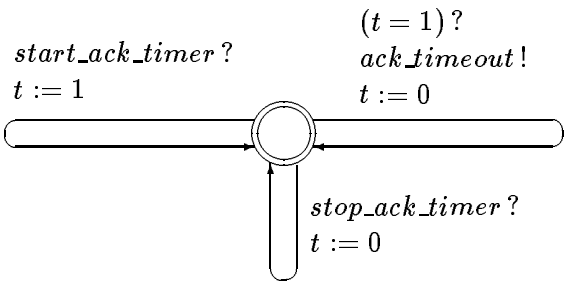
Когда агент получает неискажённый кадр f вида $data$, он выполняет следующие действия.

- Если номер $seq(f)$ попадает в прнинимающее окно, т.е. истинна формула
$$between(r, seq(f), u)$$
где предикатный символ $between$ имеет тот же смысл, что и в предыдущем протоколе (см. (8.33)), то агент
  - извлекает из этого кадра пакет, и
  - помещает этот пакет в своё принимающее окно.
- Если условие из предыдущего пункта не выполнено (т.е. номер $seq(f)$ кадра f не попал в принимающее окно), то
  - пакет в таком кадре игнорируется, и
  - у такого кадра учитывается лишь компонента $ack(f)$.

В данном протоколе участвуют следующие таймеры.

1. Массив из m таймеров, поведение которых представляется процессом $Timers$ (см. (8.32), с заменой n на m), каждый таймер из этого массива предназначен для оповещения агента о том, что
   - время ожидания подтверждения пакета с соответствующим номером из посылающего окна закончилось, и
   - необходимо послать кадр с этим пакетом ещё раз
2. Дополнительный таймер, поведение которого представляется следующим процессом:

   $Init = (t = 0)$



Этот таймер используется со следующей целью.

Посылка агентом подтверждений кадров, поступивших от другого агента, может производиться двумя способами:

(a) подтверждение посылается в составе информационного кадра (т.е. кадра вида $data$), или

(b) подтверждение посылается отдельным кадром вида ack.

Когда агенту необходимо послать такое подтверждение a, он

- включает вспомогательный таймер (действие $start\_ack\_timer$!),
- если до сигнала тайм-аута от вспомогательного таймера агент получил новый пакет от своего СУ, он
  - формирует кадр вида $data$ с этим пакетом,
  - включает в этот кадр подтверждение a как компоненту ack этого кадра, и
  - посылает этот кадр своему коллеге
- если после истечения работы вспомогательного таймера (т.е. после получения от него сигнала $ack\_timeout$) агент так и не получил новый пакет от своего СУ, он посылает подтверждение a отдельным кадром вида ack.

Описание процесса, представляющего поведение агента данного ПСО, мы даём в блок-схемном виде, по которому несложно построить блок–схему этого процесса.

В данном описании мы используем следующие обозначения и соглашения.

1. Если i – целое число, то знакосочетание $i\%m$ обозначает остаток от деления i на m.

2. Если

   - $mass$ – имя массива из m компонентов (т.е. x, y, $arrived$, и т.д.), и
   - i – целое число,

то знакосочетание $mass[i]$ следует понимать как $mass[i\%m]$.

3. Знакосочетание вида $send(kind, i)$ является сокращённым обозначением следующей группы операторов:

$$send(kind, i) = \left\{ \begin{array}{l} C\,!\,\underset{n}{\varphi}(kind, x[i], i, r-1) \\ \textbf{if}\ \ (kind = nak)\ \ \textbf{then}\ \ no\_nak := 0 \\ \textbf{if}\ \ (kind = data)\ \ \textbf{then}\ \ start\,!\,(i\%m) \\ stop\_ack\_timer\,! \end{array} \right\}$$

4. Знакосочетания

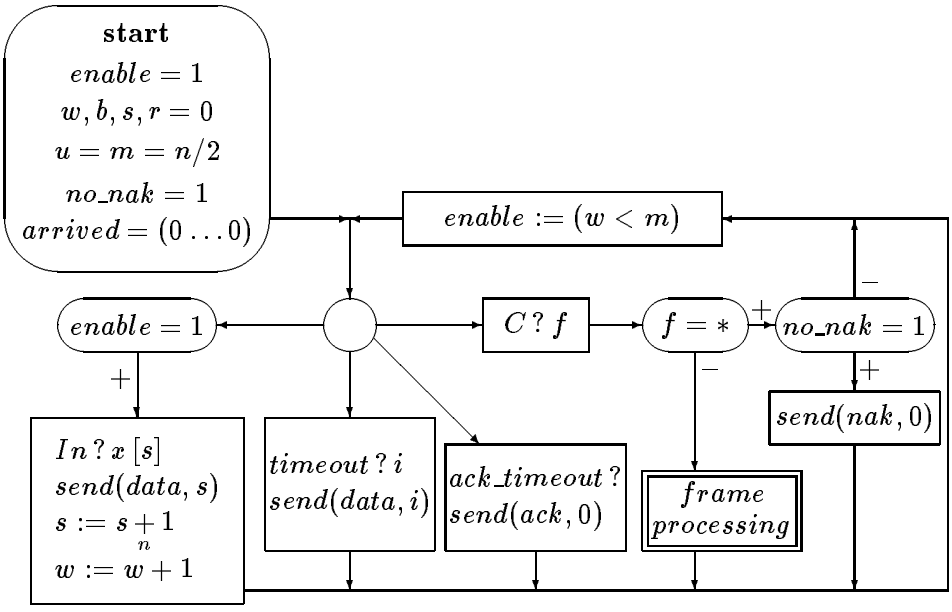
$$between(a, b, c) \quad \text{и} \quad enable := (w < m)$$

имеют тот же смысл, что и в описании предыдущего протокола.

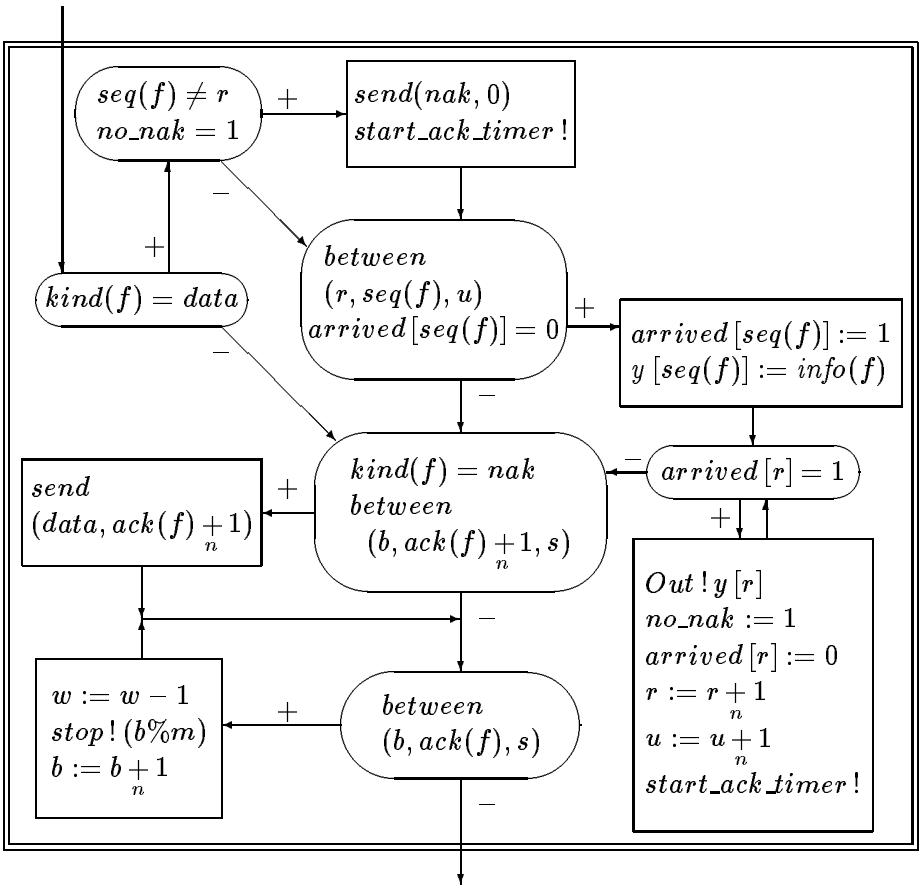
5. Если в каком-либо овале присутствуют не одна, а несколько формул, то мы предполагаем, что эти формулы связаны символом конъюнкции.

6. В целях экономии места некоторые выражения вида $f(e_1, \ldots, e_n)$ написаны в две строки (в первой строке – f, а во второй – список $(e_1, \ldots, e_n)$).

Процесс, представляющий поведение агента данного ПСО, имеет следующий вид:



Фрагмент *frame processing* в этой диаграмме выглядит следующим образом.



Читателю предлагается самостоятельно исследовать для данного протокола все вопросы, которые были перечислены в конце предыдущего параграфа.

8.5 Криптографические протоколы

Важным примером процессов с передачей сообщений являются **криптографические протоколы**.

8.5.1 Понятие криптографического протокола

Криптографический протокол (КП) – это распределённый алгоритм (т.е. совокупность нескольких взаимодействующих процессов), предназначенный для безопасной передачи, обработки и хранения информации или материальных объектов в небезопасной среде.

В работе КП принимают участие несколько **агентов**, которыми могут быть люди, вычислительные процессы, компьютеры, сети связи, базы данных, банковские карточки, банкоматы, и т.д.

Каждый из агентов КП функционирует в соответствии с некоторым последовательным алгоритмом.

Действия, исполняемые агентами, могут иметь следующий вид.

- **Посылка сообщения** другому агенту или группе агентов.

  В качестве сообщения в КП может выступать как информационный, так и материальный объект. Например это может быть набор банкнот или товар.

- **Приём сообщения** от другого агента.

- **Внутреннее действие**, которое заключается в преобразовании агентом имеющихся у него информационных и материальных объектов.

Главное отличие КП от остальных распределённых алгоритмов заключается в том, что

- при функционировании КП допускается высокая вероятность некоторых угроз безопасности сообщений, передаваемых агентами этого КП, и
- КП должен содержать механизмы противодействия этим угрозам.

Под **угрозой безопасности** сообщения понимается возможность выполнения с этим сообщением какой-либо из следующих операций.

1. Нарушение **конфиденциальности** сообщения, т.е., например, перехват этого сообщения (т.е. получение его копии), и
   - полное или частичное ознакомление с содержанием этого сообщения субъектами, не имеющими для этого соответствующих полномочий, или
   - извлечение из сообщения новой информации, например,
     - извлечение из шифртекстов фрагмента криптографического ключа, используемого при создании этих шифртекстов, или
     - извлечение из сообщений о покупках какого-либо лица предположений о его финансовом положении или о его предпочтениях.
2. Нарушение **целостности** сообщения, т.е., например, искажение передаваемого текста в процессе передачи.
3. Кража сообщения (например, кража денег или товара в процессе их доставки от одного агента другому агенту).
4. Подмена сообщения, т.е.

- изъятие сообщения, которое агент A послал агенту B, и
- посылка агенту B другого сообщения вместо изъятого.

Ниже под угрозами безопасности мы будем понимать угрозы безопасности сообщений, передаваемых между агентами какого-либо КП.

Угрозы безопасности могут быть внешними и внутренними.

1. **Внешние угрозы** связаны с вторжением в работу агентов КП других агентов, не входящих в этот КП (их называют **противниками**).

   Противники делятся на следующие два класса.

   (a) **Пассивные противники**, которые могут лишь перехватывать сообщения, пересылаемые другими агентами, и анализировать их.

   (b) **Активные противники**, которые могут

   - перехватывать сообщения, пересылаемые другими агентами, и анализировать их
   - модифицировать или удалять пересылаемые сообщения
   - создавать новые сообщения и посылать их другим агентам
   - и т.д.

2. **Внутренние угрозы** связаны с наличием среди агентов КП нечестных агентов (их называют **мошенниками**), которые могут

   - неправильно выполнять действия, которые им полагается выполнять в ходе работы КП,
   - выдавать себя за других агентов,
   - объединяться в коалиции и совместно использовать сообщения, получаемые ими в ходе работы КП, в злонамеренных целях
   - и т.д.

Свойства безопасности КП представляют собой описание видов угроз безопасности, которым может быть оказано противодействие во время функционирования этого КП.

Наиболее надёжными считаются такие КП, для которых имеется математическое доказательство утверждения о том, что реализованные в этих КП механизмы противодействия угрозам безопасности действительно выполняют поставленные перед ними задачи.

8.5.2 Шифрование сообщений

Для противодействия угрозам нарушения конфиденциальности некоторые из пересылаемых сообщений могут передаваться от одних агентов КП другим агентам в зашифрованном виде.

Шифрование сообщения m представляет собой применение некоторого алгоритма к паре (K, m), где K – объект, называемый **ключом шифрования**. Результат применения алгоритма шифрования к паре (K, m) обозначается знакосочетанием $K(m)$.

Операция извлечения исходного сообщения из зашифрованного сообщения m' называется **дешифрованием**. Данная операция тоже представляет собой применение некоторого алгоритма к паре (K', m'), где K' – объект, называемый **ключом дешифрования**. Результат применения алгоритма дешифрования к паре (K', m') обозначается знакосочетанием $K'(m')$. Ключи шифрования K и дешифрования K' должны быть связаны соотношением

$$\text{для каждого сообщения } m \qquad K'(K(m)) = m \tag{8.34}$$

Если символ K обозначает некоторый ключ шифрования, то знакосочетание K^{-1} обозначает ключ дешифрования K', удовлетворяющий условию (8.34).

Системой шифрования (СШ) называется пара, состоящая из алгоритма шифрования и алгоритма дешифрования. Используемые в настоящее время на практике СШ подразделяются на два класса:

- **симметричные СШ**, в которых каждый ключ шифрования K совпадает с соответствующим ему ключом дешифрования K^{-1}, и
- **СШ с открытым ключом**, в которых ключи шифрования и дешифрования сопоставлены конкретным агентам: если символ A обозначает некоторого агента, использующего СШ с открытым ключом, то знакосочетания K_A^+ и K_A^- обозначают сопоставленные ему ключи шифрования и дешифрования в этой СШ, причём
  - ключ шифрования K_A^+ (называемый **открытым ключом (ОК)** агента A) известен всем агентам
  - ключ дешифрования K_A^- (называемый **закрытым ключом (ЗК)** агента A) известен только агенту A
    (т.е. если какой-либо агент докажет, что он умеет дешифровать сообщения вида $K_A^+(m)$, то он совпадает с A)

Главные различия между СШ с открытым ключом и симметричными СШ заключаются в следующем:

- по сравнению с симметричными СШ, СШ с открытым ключом являются существенно более стойкими к атакам, связанным с нарушением конфиденциальности зашифрованных сообщений путём криптоанализа,
- однако сложность выполнения операций шифрования и дешифрования в СШ с открытым ключом примерно на три порядка выше сложности этих операций в симметричных СШ.

Поэтому во многих КП использование этих СШ носит комбинированный характер:

- Симметричные СШ используются для основной связи, но ключи, используемые в этих СШ, регулярно обновляются.

  Это делается для того, чтобы у противника, перехватывающего сообщения, не было большого количества перехваченных сообщений, зашифрованных на одном и том же ключе, по которым он мог бы

  – вычислить используемый ключ,
  – или иным образом нарушить конфиденциальность

  путём криптоанализа перехваченных сообщений.
- Новые ключи для симметричных СШ пересылаются использующим их агентам в зашифрованном виде, причём шифрование этих ключей происходит с использованием СШ с открытым ключом.

8.5.3 Формальное описание КП

Одним из простейших видов формального описания КП является задание списка Σ действий, каждое из которых имеет вид

$$A \to B : m \tag{8.35}$$

где A и B – имена агентов, и m – выражение, значением которого является сообщение или неопределённый объект. Выполнение действия (8.35) происходит следующим образом. Если значение выражения m определено, то

- A посылает B сообщение, равное значению выражения m, и
- B принимает это сообщение.

Если значение выражения m не определено, то это действие пропускается.

Действия, входящие в список Σ, выполняются в соответствии с тем порядком, в котором они входят в этот список. Если текущее действие не относится к какому-либо из агентов, то этот агент не функционирует в момент исполнения этого действия.

В описание КП могут также входить условия, выражающие, например, меру доверия одних агентов другим агентам.

8.5.4 Примеры КП

КП продажи компьютера

У A есть компьютер, а у B есть деньги. B хочет купить у A компьютер, для чего B должен передать A деньги, а A должен передать B компьютер.

A и B не верят друг другу, поэтому

- A хочет сначала получить деньги, и после этого передать B компьютер
- B хочет сначала получить компьютер, и после этого отдать A деньги.

Таким образом, решить проблему продажи компьютера силами лишь A и B невозможно.

Данная проблема может быть решена при помощи КП, в котором кроме A и B принимает участие **доверенный посредник** T, относительно которого у A и B имеется уверенность в том, что T будет точно выполнять все действия, предписанные ему этим КП.

Искомый КП может иметь следующий вид:

- $A \to T$: компьютер
- $B \to A$: деньги
- $A \to T$: подтверждение или опровержение того, что полученная от B сумма соответствует стоимости компьютера
  (A посылает опровержение вместе с деньгами B)
- $T \to B$: (от A поступило подтверждение) ? компьютер
- $T \to A$: (от A поступило опровержение) ? компьютер

В этом КП значение выражения вида $b?m$, где b – условие, и m – сообщение

- равно m, если b истинно, и
- не определено, если b ложно.

Читателю предлагается самостоятельно

- разработать язык спецификаций, в терминах которого можно было бы
  - выразить условие того, что A и B доверяют посреднику T, и
  - сформулировать спецификацию $Spec$, выражающую свойство корректности этого КП
- проанализировать соответствие этого КП спецификации $Spec$
  - с учётом условия, что A и B доверяют T, и
  - без учёта этого условия.

Обедающие криптографы

За круглым столом сидят три криптографа и обедают. После того, как они пообедали, официант сообщает им, что их обед полностью оплачен, но не уточняет, кто именно платил.

Возможен один из двух вариантов:

- обед оплатил один из криптографов,
- за обед заплатила ФСБ.

Криптографы хотят выяснить, какой именно из вариантов имеет место, причём, если имеет место первый вариант, то те из них, которые не платили, не должны узнать, кто же конкретно заплатил.

Для решения этой задачи предлагается следующий КП.

Поскольку криптографы сидят за круглым столом, то каждая пара соседей может подбрасывать монету между собой, так, чтобы результат был известен только им двоим.

После того, как все три пары подбросили монету, каждый из них знает результаты двух подбрасываний (решка или орёл), которые могут быть

- либо одинаковыми (оба раза была решка, или оба раза был орёл),
- либо разными (один раз была решка, а другой - орёл).

Каждый криптограф говорит другим "одинаково" или "по разному", причём тот, кто заплатил, говорит противоположное (т.е. если надо сказать "одинаково" то он говорит "по-разному", и наоборот).

Если число ответов "по разному" чётно, то это значит, что обед оплатила ФСБ, иначе - один из них.

Подтверждение приёма

Имеется группа агентов, которые используют одну и ту же СШ с открытым ключом, обладающую следующим свойством: алгоритм дешифрования этой СШ может использоваться также и для шифрования, причём имеет место условие:

$$\text{для каждого агента } A \text{ и каждого сообщения } m \qquad K_A^+(K_A^-(m)) = m \quad (8.36)$$

(большинство СШ с открытым ключом обладает этим свойством).

Члены этой группы хотят обмениваться друг с другом зашифрованными сообщениями.

Если, например, агент A хочет послать агенту B сообщение m, то для этого A и B предполагают использовать следующий КП из двух действий:

$$\begin{cases} A \to B : (A, K_B^+(K_A^-(m))) \\ B \to A : (B, K_A^+(K_B^-(m))) \end{cases}$$

т.е. A посылает B пару, состоящую из

- своего имени (A), чтобы B знал, от кого ему пришло сообщение, и
- шифрованного сообщения $K_B^+(K_A^-(m))$.

Получив сообщение $K_B^+(K_A^-(m))$, агент B при помощи своего ЗК K_B^- извлекает из него сообщение $K_A^-(m)$, из которого он, используя ОК K_A^+, может извлечь m (это возможно на основании (8.36)).

Второе действие в этом КП представляет собой подтверждение агентом B получения сообщения от A: оно посылается отправителю для того, чтобы он был уверен, что его сообщение m дошло до B в неискажённом виде.

К сожалению, данный КП не содержит механизма противодействия угрозам, связанным с мошенничеством. Если некоторый агент M из этой группы перехватил сообщение

$$(A, K_B^+(K_A^-(m)))$$

то он может извлечь из него сообщение m, например, следующим способом.

- M посылает B сообщение $(M, K_B^+(K_A^-(m)))$
- B обрабатывает его в соответствии с КП, т.е. извлекает из него вторую компоненту и вычисляет сообщение

$$K_M^+(K_B^-(K_B^+(K_A^-(m)))) \;\; = K_M^+(K_A^-(m))$$

- Получившийся результат не содержит никакого смысла, но, согласно используемому КП, B обязан послать M подтверждение

$$(B, K_M^+(K_B^-(K_M^+(K_A^-(m)))))$$

- Из полученного сообщения агент M, используя свой ЗК K_M^- и известные ему ОК K_B^+ и K_A^+, извлекает m.

КП двусторонней аутентификации

Имеется группа агентов, которые используют

- одну и ту же СШ с открытым ключом, и
- одну и ту же симметричную СШ.

Члены этой группы хотят обмениваться друг с другом зашифрованными сообщениями.

Каждая пара агентов A, B из этой группы использует для основной связи симметричную СШ, и в процессе своего взаимодействия A и B предполагают регулярно обновлять ключи шифрования для этой СШ, т.е.

- некоторый промежуток времени (который называется **сеансом взаимодействия**) агенты A и B используют для шифрования своих сообщений один ключ (называемый **сеансовым ключом**),
- затем при помощи СШ с открытым ключом этот ключ обновляется, и для шифрования сообщений в следующем сеансе взаимодействия A и B используется новый сеансовый ключ
- и т.д.

В целях противодействия возможному мошенничеству, члены группы хотят производить обновление сеансовых ключей при помощи КП, который должен решать следующие задачи:

- агенты A и B, желающие создать новый сеансовый ключ, должны принимать равноправное участие в создании этого ключа
- перед тем, как начать очередной сеанс взаимодействия, они хотят удостовериться в подлинности друг друга, т.е.
  - агент A намерен удостовериться, что тот агент, совместно с которым он генерирует новый сеансовый ключ, это действительно B, а не мошенник, выступающий от имени B, и
  - агент B имеет аналогичное намерение.

Процедура взаимодействия агентов, целью которой является проверка подлинности кого-либо из них, называется **аутентификацией** этого агента. Если в результате взаимодействия A и B они доказывают свою подлинность друг другу, то такая аутентификация называется **двусторонней**.

Агенты A и B собираются решить задачи

- двусторонней аутентификации, и
- генерации нового сеансового ключа K_{AB}

при помощи следующего КП из трёх действий (данный КП называется **КП Нидхема - Шредера** (Needham - Schroeder, 1979), мы будем сокращённо обозначать его знакосочетанием NS):

$$\begin{cases} A \to B : K_B^+(A, N_A) \\ B \to A : K_A^+(N_A, N_B) \\ A \to B : K_B^+(N_B) \end{cases}$$

где

- символ A в первом сообщении обозначает имя агента A,

- символы N_A и N_B обозначают битовые строки, длина которых равна длине создаваемого сеансового ключа K_{AB}. Данные строки называются **нонсами**. Как правило, нонс представляет собой псевдослучайную последовательность битов.

Нонсы предназначены для решения следующих двух задач.

1. Каждый из нонсов является вкладом сгенерировавшего его агента в формирование ключа K_{AB}: данный ключ по определению полагается равным побитовой сумме по модулю 2 нонсов N_A и N_B.

   Нетрудно видеть, что условие равноправного участия агентов A и B в формировании ключа K_{AB} выполнено.

2. Нонсы решают задачу аутентификации следующим образом.

   - Когда B посылает A сообщение $K_A^+(N_A, N_B)$, он тем самым демонстрирует свою способность извлечь из полученного им зашифрованного сообщения $K_B^+(A, N_A)$ нонс N_A (т.к., по предположению, B не мог получить нонс N_A никаким другим способом).

     Это убеждает A в том, что тот агент, с которым он взаимодействует, знает ЗК K_B^- (т.к., по предположению, не зная K_B^-, извлечь сообщение m из сообщения вида $K_B^+(m)$ невозможно), т.е. этот агент действительно есть B.
   - Когда A посылает B сообщение $K_B^+(N_B)$, он тем самым демонстрирует свою способность извлечь из полученного им зашифрованного сообщения $K_A^+(N_A, N_B)$ нонс N_B (т.к., по предположению, A не мог получить нонс N_B никаким другим способом).

     Это убеждает B в том, что тот агент, с которым он взаимодействует, знает ЗК K_A^- (т.к., по предположению, не зная K_A^-, извлечь сообщение m из шифртекста вида $K_A^+(m)$ невозможно), т.е. этот агент действительно есть A.

Одна из уязвимостей данного КП связана с тем, что один из агентов (например, B) может мошенничать, выдавая себя за другого агента. Работа КП NS в этом случае может иметь следующий вид.

1. После того, как агент A выразит желание взаимодействовать с B по КП NS, и пришлёт ему первое сообщение (т.е. $K_B^+(A, N_A)$), агент B может использовать это сообщение для того, чтобы

   - взаимодействовать по КП NS с ещё одним агентом C, и
   - в этом взаимодействии B будет выдавать себя за агента A.

   Для этого B

- договаривается с C о взаимодействии с ним по КП NS, и
- посылает ему сообщение $K_C^+(A, N_A)$.

Получив это сообщение и расшифровав его, C делает вывод, что это сообщение было послано агентом A (о чём говорит ему первая компонента расшифрованного сообщения).

На основании этого, C

- генерирует нонс N_C, и
- посылает агенту B сообщение

$$K_A^+(N_A, N_C) \tag{8.37}$$

(думая, что он посылает его агенту A)

2. B пересылает полученное от C сообщение (8.37) агенту A, эта пересылка представляет собой второе действие в КП NS взаимодействия A и B.

3. A рассматривает извлечённую из полученного сообщения (8.37) компоненту N_C как нонс, сгенерированный агентом B.

   В соответствии с третьим действием КП NS взаимодействия A и B, агент A посылает агенту B сообщение

   $$K_B^+(N_C)$$

   B извлекает N_C, и посылает агенту C сообщение $K_C^+(N_C)$.

   C рассматривает эту пересылку как третье действие в КП NS взаимодействия A и C.

После этого B знает нонсы N_A и N_C, которые позволит ему вычислить сеансовый ключ для шифрованного взаимодействия с C, в котором он будет выступать от имени A.

Читателю предлагается самостоятельно

- разработать язык спецификаций, в терминах которого можно было бы выразить свойство КП противодействовать мошенничеству описанного выше вида, и
- разработать алгоритм проверки соответствия КП спецификациям, выраженным на этом языке.

Глава 9

Представление структур данных в виде процессов

9.1 Понятие структуры данных

Структура данных (СД) – это дискретная динамическая система, каждое состояние которой можно интерпретировать как совокупность **данных** (т.е. некоторых значений), хранящихся в текущий момент в этой системе.

Как правило, каждое состояние СД представляет собой совокупность из нескольких компонентов, каждая из которых является содержимым некоторого ресурса (например, содержимым ячейки памяти).

Одним из способов формального описания СД является представление их в виде процессов.

Если процесс P является представлением некоторой СД, то имена в P можно интерпретировать как точки доступа к данным, содержащимся в этой СД. Объекты, соответствующие этим именам, могут иметь виртуальный характер. Например, в качестве таких объектов могут выступать криптографические ключи.

Представление СД в виде процессов позволяет формализовать понятие активных данных (= знаний), которые развиваются в результате наличия у них неполноты или противоречивости. Активность структур данных заключается в том, что они могут сами инициировать запросы к окружающей среде с целью получения дополнительной информации для достижения согласованности в знаниях, которые содержатся в этих СД.

В этой главе мы приводим несколько примеров описания СД в виде процессов. Каждый из этих процессов задаётся в виде рекурсивного определения.

Ниже мы будем отождествлять процесс, представляющий некоторую СД, с самой этой СД.

9.2 СД "память с 2^k ячейками"

СД "память с 2^k ячейками" (где $k \geq 0$) – это процесс

$$[\![MEM_k(\vec{x})]\!] \tag{9.1}$$

где MEM_k – процессное имя, и $\vec{x}$ – список из 2^k переменных вида

$$\vec{x} = \{x_m \mid m \in \{0,1\}^k\}$$

каждая из которых соответствует некоторой **ячейке памяти**: для каждого $m \in \{0,1\}^k$ значение переменной x_m представляет собой содержимое ячейки памяти с адресом m.

Мы будем предполагать, что процесс $[\![MEM_k(\vec{x})]\!]$ работает следующим образом:

1. сначала, если $k > 0$, то через точку доступа α в процесс $[\![MEM_k(\vec{x})]\!]$ поступает адрес m,
2. затем через точку доступа β в процесс $[\![MEM_k(\vec{x})]\!]$ поступает значение v, которое надо записать в ячейке по адресу m
3. после чего через точку доступа γ процесс $[\![MEM_k(\vec{x})]\!]$ выдаёт значение, которое содержалось в ячейке по адресу m до выполнения предыдущего шага.

Процессы, соответствующие процессным выражениям вида $MEM_k(\vec{x})$, определяются следующим РО:

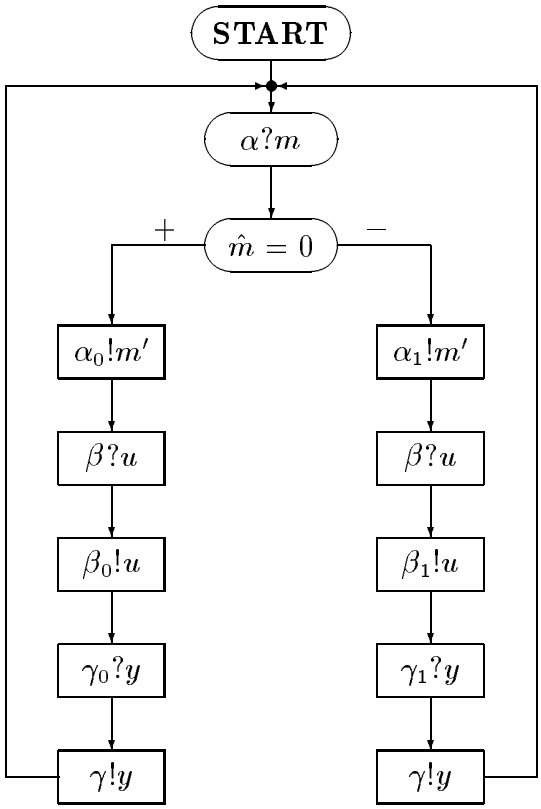
- $MEM_0(x) = \beta?y \,.\, \gamma!x \,.\, MEM_0(y)$
  (процесс $[\![MEM_0(x)]\!]$ – это - ячейка памяти, хранящая значение x)
- для каждого $k \geq 0$

$$MEM_{k+1}(\vec{x} \cdot \vec{y}) = \left(\begin{array}{l} SWITCH \mid \\ MEM_k(\vec{x})\,[\,\alpha_0/\alpha,\ \beta_0/\beta,\ \gamma_0/\gamma\,] \mid \\ MEM_k(\vec{y})\,[\,\alpha_1/\alpha,\ \beta_1/\beta,\ \gamma_1/\gamma\,] \end{array} \right) \setminus \left\{ \begin{array}{l} \alpha_0, \beta_0, \gamma_0, \\ \alpha_1, \beta_1, \gamma_1 \end{array} \right\}$$

  где

  – $\vec{x} \cdot \vec{y}$ – список из 2^{k+1} различных переменных, представленный в виде конкатенации двух списков из 2^k переменных

– $SWITCH$ - это процесс, называемый **переключателем**, и представляемый следующей блок-схемой:

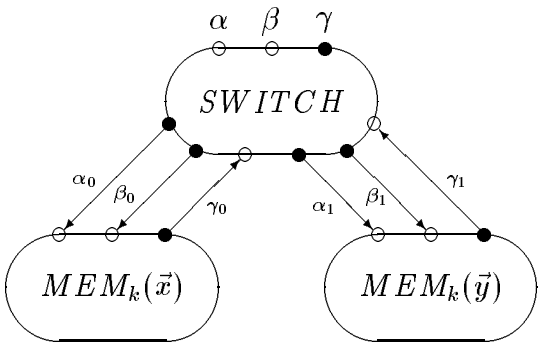


где

– $\hat{m}$ = первый бит в битовой строке m, и

– m' = строка из $k-1$ битов, получаемая из m удалением первого бита.

Процесс $[\![MEM_{k+1}(\vec{x} \cdot \vec{y})]\!]$ изображается следующим потоковым графом:



Читателю предлагается самостоятельно доказать, что процесс $[\![MEM_k(\vec{x})]\!]$ удовлетворяет следующей спецификации: если $k > 0$, то для каждого процесса P

$$\left(\begin{array}{l} MEM_k(\vec{x}) \mid \\ \alpha!m \,.\, \beta!u \,.\, \gamma?y \,.\, P \end{array} \right) \setminus \{\alpha,\beta,\gamma\} \stackrel{+}{\approx} \left(\begin{array}{l} (x_m := u) \,.\, MEM_k(\vec{x}) \mid \\ (y := x_m) \,.\, P \end{array} \right) \setminus \{\alpha,\beta,\gamma\}$$

Операция

- чтения значения, содержащегося в ячейке процесса $[\![MEM_k(\vec{x})]\!]$ по адресу m (без изменения содержимого этой ячейки), и
- занесения этого значения в переменную y процесса P

представляется процессным выражением

$$\left(\begin{array}{l} MEM_k(\vec{x}) \mid \\ \alpha!m \,.\, \beta!0 \,.\, \gamma?y \,.\, \alpha!m \,.\, \beta!y \,.\, \gamma?z \,.\, P \end{array} \right) \setminus \{\alpha,\beta,\gamma\} \qquad (\text{где } z \notin P) \qquad (9.2)$$

Читателю предлагается самостоятельно доказать, что

$$(9.2) \stackrel{+}{\approx} \left(\begin{array}{l} MEM_k(\vec{x}) \mid \\ (y := x_m) \,.\, P \end{array} \right) \setminus \{\alpha,\beta,\gamma\}$$

Можно изменить определение процесса $[\![MEM_k(\vec{x})]\!]$ так, чтобы новый процесс удовлетворял спецификации

$$MEM_k(\vec{x}) \stackrel{+}{\approx} \alpha?m. \left(\beta?y \,.\, (x_m := y) \,.\, MEM_k(\vec{x}) \;+\; \gamma!x_m \,.\, MEM_k(\vec{x}) \right)$$

т.е.

- сначала в процесс $[\![MEM_k(\vec{x})]\!]$ вводится адрес m,
- а потом, в зависимости от выбора окружающей среды,
  - либо в $[\![MEM_k(\vec{x})]\!]$ записывается новое значение по адресу m
  - либо читается значение, хранящееся в $[\![MEM_k(\vec{x})]\!]$ по адресу m

Для этого надо по-другому определить процесс $[\![MEM_0(x)]\!]$:

$$MEM_0(x) = \beta?y \,.\, MEM_0(y) \;+\; \gamma!x \,.\, MEM_0(x)$$

Работа с новым процессом $[\![MEM_k(\vec{x})]\!]$ происходит следующим образом.

Операция записи значения выражения e в ячейку по адресу m представляется процессным выражением

$$\left(\; MEM_k(\vec{x}) \mid \alpha!m \,.\, \beta!e \,.\, \mathbf{0} \; \right)$$

Операция чтения процессом P значения, записанного в ячейке процесса $[\![MEM_k(\vec{x})]\!]$ по адресу m, и занесения этого значения в переменную y процесса P, представляется процессным выражением

$$\left(\; MEM_k(\vec{x}) \mid \alpha!m \,.\, \gamma?y \,.\, P \; \right)$$

9.3 СД "стек"

Стек – это СД с точками доступа α и γ, с которой можно выполнять следующие операции:

1. положить значение v в стек (действие $\alpha\,!\,v$)
2. если стек непуст, то взять головной элемент стека (действие $\gamma\,?\,x$).

СД "стек" представляется совокупностью процессных выражений

$$\{PUSH_n(x_1,\ldots,x_n) \mid n \geq 0\} \tag{9.3}$$

где для каждого $n \geq 0$ $\quad PUSH_n$ – процессное имя, и ПВ

$$PUSH_n(x_1,\ldots,x_n)$$

представляет процесс "стек, хранящий n значений".

Процессы, соответствующие ПВ из совокупности (9.3) определяются следующим РО:

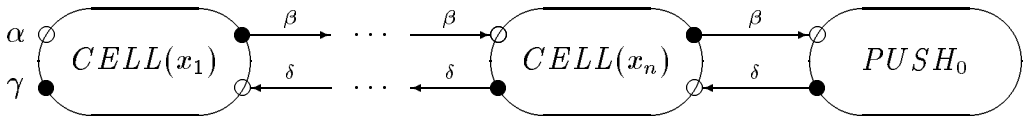
1. $PUSH_0 = \alpha?x\,.\,PUSH_1(x)\;+\;\gamma!\$\,.\,\mathbf{0}$
2. для каждого $n \geq 0$

$$PUSH_{n+1}(x,x_1,\ldots,x_n) = \left(\begin{array}{l} CELL(x)\,| \\ PUSH_n(x_1,\ldots,x_n)[\beta/\alpha,\delta/\gamma] \end{array}\right) \setminus \{\beta,\delta\}$$

где ПВ $CELL(x)$ представляет процесс с точками доступа α, β, γ, δ, называемый **ячейкой памяти**, и определяемый следующим образом:

$$CELL(x) = \left(\begin{array}{l} \alpha?y\,.\,\beta!x\,.\,CELL(y)\,+ \\ \gamma!x\,.\,\delta?y\,.\,\left(\;(y=\$)\;?\;PUSH_0\;+\;(y\neq\$)\;?\;CELL(y)\;\right) \end{array}\right)$$

Потоковый граф процесса $[\![PUSH_n(x_1,\ldots,x_n)]\!]$ имеет следующий вид:



Читателю предлагается самостоятельно доказать, что совокупность процессов, соответствующих ПВ из (9.3), удовлетворяет следующей спецификации: для каждого $n \geq 0$

$$PUSH_{n+1}(x,x_1,\ldots,x_n) \overset{+}{\approx} \left(\begin{array}{l} \alpha?y\,.\,PUSH_{n+2}(y,x,x_1,\ldots,x_n)+ \\ \gamma!x\,.\,PUSH_n(x_1,\ldots,x_n) \end{array}\right)$$

9.4 СД "очередь"

Очередь – это СД с точками доступа α и γ, с которой можно выполнять следующие операции:

1. добавить значение v в конец очереди (действие $\alpha\,!\,v$)
2. если очередь непуста, то взять первый элемент очереди (действие $\gamma\,?\,x$).

СД "очередь" представляется совокупностью процессных выражений

$$\{QUEUE_n(x_1,\ldots,x_n) \mid n \geq 0\} \tag{9.4}$$

где для каждого $n \geq 0$ $\quad QUEUE_n$ – процессное имя, и ПВ

$$QUEUE_n(x_1,\ldots,x_n)$$

представляет процесс "очередь, хранящая n значений".

Процессы, соответствующие ПВ из совокупности (9.4) определяются следующим РО:

1. $QUEUE_0 = \alpha?x\,.\,QUEUE_1(x)\ +\ \gamma!\$\,.\,\mathbf{0}$
2. для каждого $n \geq 0$

$$QUEUE_{n+1}(x,x_1,\ldots,x_n) = \left(\begin{array}{l} CELL(x)\,| \\ QUEUE_n(x_1,\ldots,x_n)[\beta/\alpha,\delta/\gamma] \end{array}\right) \setminus \{\beta,\delta\}$$

где ПВ $CELL(x)$ представляет процесс с точками доступа α, β, γ, δ, называемый **ячейкой памяти**, и определяемый следующим образом:

$$CELL(x) = \left(\begin{array}{l} \alpha?y\,.\,\Big(\,\beta!x\,.\,CELL(y) + \gamma!x\,.\,CELL(y)\,\Big) + \\ \gamma!x\,.\,\delta?y\,.\,\Big(\,(y=\$)\,?\,QUEUE_0 + (y\neq\$)\,?\,CELL(y)\,\Big) \end{array}\right)$$

Потоковый граф процесса $[\![QUEUE_n(x_1,\ldots,x_n)]\!]$ имеет такой же вид, как и потоковый граф в пункте 9.3.

Читателю предлагается самостоятельно доказать, что совокупность процессов, соответствующих ПВ из (9.4), удовлетворяет следующей спецификации: для каждого $n \geq 0$

$$QUEUE_{n+1}(x,x_1,\ldots,x_n) \stackrel{+}{\approx} \left(\begin{array}{l} \alpha?y\,.\,QUEUE_{n+2}(x,x_1,\ldots,x_n,y) + \\ \gamma!x\,.\,QUEUE_n(x_1,\ldots,x_n) \end{array}\right)$$

Глава 10

Семантика языка параллельного программирования

Семантика языка программирования представляет собой правило, сопоставляющее каждой конструкции этого языка некоторый математический объект. В качестве такого объекта может выступать, например, логическая формула или процессное выражение.

Главной целью определения семантики языка программирования является сведение задач анализа свойств программ на этом языке к задачам анализа математических утверждений, соответствующих этим свойствам.

В этой главе мы рассмотрим в качестве примера определение семантики простейшего языка параллельного программирования в терминах теории процессов.

10.1 Описание языка параллельного программирования

В этом параграфе мы описываем простейший язык параллельного программирования, который мы будем обозначать символом $\mathcal{L}$.

10.1.1 Конструкции языка $\mathcal{L}$

Конструкции языка $\mathcal{L}$ делятся на следующие три класса.

1. **Выражения**.

   Понятие выражения языка $\mathcal{L}$ совпадает с аналогичным понятием, определённым в параграфе 7.2.3. Выражения строятся из переменных, констант и ФС. Каждому выражению e сопоставлен некоторый тип $type(e)$.

2. **Декларации**.

   Каждая декларация D имеет один из следующих трёх видов:

   (a) **объявление локальной переменной:**

   $$\texttt{VAR}\ x \tag{10.1}$$

   где x – имя переменной

   (b) **объявление ресурса:**

   $$\texttt{RESOURCE}\ R \tag{10.2}$$

   где R - имя ресурса, который может представлять собой, например, устройство ввода или вывода

   (c) **описание процедуры:**

   $$\texttt{PROCEDURE}\ G(u, v)\ \texttt{IS}\ C \tag{10.3}$$

   где

   - G – имя процедуры,
   - u – переменная, изображающая аргумент процедуры
   - v – переменная, изображающая результат процедуры
   - C – оператор (тело процедуры), в который переменные u и v входят как формальные параметры этой процедуры.

3. **Операторы**.

   Каждый оператор представляет собой описание некоторого алгоритма.

   Ниже

   - операторы обозначаются символом C
   - декларации обозначаются символом D
   - переменные обозначаются символами x, y, z, u, v, ...
   - выражения обозначаются символом e
   - формулы (т.е. выражения типа $bool$) обозначаются символом b

   причём при всех этих символах могут быть индексы.

   Операторы имеют следующий вид.

   (a) **присваивание:**

   $$x := e$$

   где $type(x) = type(e)$

(b) **условный переход:**

$$\texttt{if}\ b\ \texttt{then}\ C_1\ \texttt{else}\ C_2$$

(c) **цикл:**

$$\texttt{while}\ b\ \texttt{do}\ C$$

(d) **ввод:**

$$\texttt{input}\ x$$

(e) **вывод:**

$$\texttt{output}\ e$$

(f) **пустой оператор:**

$$\texttt{skip}$$

(g) **последовательная композиция:**

$$C_1; C_2$$

(h) **параллельная композиция:**

$$C_1\ \texttt{par}\ C_2$$

(i) **блок:**

$$\texttt{begin}\ \{D_1, \ldots, D_k; C\}\ \texttt{end}$$

Блок "связывает" все вхождения объектов, объявленных в декларациях $D_1, \ldots, D_k$: эти объекты являются видимыми только внутри этого блока.

(j) **вызов процедуры:**

$$\texttt{call}\ G(e, z)$$

где

- G – имя вызываемой процедуры,
- e – выражение, значение которого является аргументом процедуры G, и
- z – переменная, в которую будет занесён результат выполнения процедуры G.

(k) **связывание оператора с ресурсом:**

$$\texttt{with}\ R\ \texttt{do}\ C$$

где R – имя некоторого ресурса.

10.1.2 Программы на языке $\mathcal{L}$

Программа на языке $\mathcal{L}$ представляет собой оператор

$$\texttt{begin}\,\{D_1, \ldots, D_k; C\}\,\texttt{end}$$

где все переменные, ресурсы, и процедуры, имена которых входят в C, объявлены

- в декларациях D_1, ..., D_k,
- или в тех декларациях, которые содержатся в C.

Программа может взаимодействовать с окружающей средой только путём выполнения операторов ввода и вывода.

10.2 Семантика языка $\mathcal{L}$

В этом параграфе мы определяем семантику языка $\mathcal{L}$, которая представляет собой правило, сопоставляющее каждой конструкции Q языка $\mathcal{L}$ некоторое ПВ $\langle\!|Q|\!\rangle$, называемое **семантикой конструкции** Q.

При определении этой семантики мы будем использовать следующие обозначения и соглашения.

1. В каждом ПВ B, соответствующем какой-либо конструкции языка $\mathcal{L}$, знакосочетания
$$\delta!, \quad \rho!, \quad i? \quad \text{и} \quad o!$$
обозначают действия, имеющие заранее предопределённый смысл:
   (a) $\delta!$ обозначает сигнал о завершении исполнения процесса, соответствующего процессному выражению B
   (b) $\rho!$ обозначает вывод значения, являющегося результатом работы процесса, соответствующего процессному выражению B
   (c) $i?$ обозначает ввод значения в программу
   (d) $o!$ обозначает вывод значения из программы
2. Символ `done` обозначает ПВ $\delta!\,.\,\mathbf{0}$
3. Для каждой пары ПВ B_1, B_2
   (a) знакосочетание B_1 `before` B_2 обозначает ПВ
$$\Big(\; B_1[\beta/\delta] \,|\, \beta?.B_2 \;\Big) \setminus \{\beta\}$$
   где β не входит в B_1 и B_2

(b) знакосочетание B_1 `result` B_2 обозначает ПВ

$$(B_1 \mid B_2) \setminus \{\rho\}$$

(c) знакосочетание B_1 `par` B_2 обозначает ПВ

$$\left(\begin{array}{l} B_1[\delta_1/\delta] \mid \\ B_2[\delta_2/\delta] \mid \\ \delta_1!.\delta_2!.\mathtt{done} + \delta_2!.\delta_1!.\mathtt{done} \end{array} \right) \setminus \{\delta_1, \delta_2\}$$

где δ_1 и δ_2 не входят в B_1 и B_2

4. Для каждого объекта с именем n, объявленного в некоторой декларации D, символ L_n обозначает подмножество множества $Names$, состоящее из **точек доступа** к этому объекту:

$$L_n \stackrel{\text{def}}{=} \begin{cases} \{\alpha_x, \gamma_x, \pi_x, \varphi_x\}, & \text{если } D = (10.1) \\ \{\pi_R,\ \varphi_R\}, & \text{если } D = (10.2) \\ \{\alpha_G,\ \gamma_G\}, & \text{если } D = (10.3) \end{cases}$$

(если имена n_1 и n_2 таких объектов различны, то $L_{n_1} \cap L_{n_2} = \emptyset$)

10.2.1 Семантика выражений

Каждому выражению e языка $\mathcal{L}$ соответствует ПВ $\langle\!|e|\!\rangle$, определяемое индукцией по построению выражения e:

- для каждой переменной $x \in Var$

$$\langle\!|x|\!\rangle \stackrel{\text{def}}{=} \gamma_x?y.\ \rho!y.\ \mathbf{0}$$

- для каждой константы $c \in Con$

$$\langle\!|c|\!\rangle \stackrel{\text{def}}{=} \rho!c.\ \mathbf{0}$$

- для каждого ФС f

$$\begin{array}{l} \langle\!|f(e_1, \ldots, e_n)|\!\rangle \stackrel{\text{def}}{=} \\ \stackrel{\text{def}}{=} \left(\begin{array}{l} \langle\!|e_1|\!\rangle\ [\rho_1/\rho]\ \mid\ \ldots\ \mid \langle\!|e_n|\!\rangle\ [\rho_n/\rho]\ \mid \\ \rho_1?x_1.\ \ldots\ \rho_n?x_n\,.\,\rho!\ f(x_1, \ldots, x_n)\ .\ \mathbf{0} \end{array} \right) \setminus \{\rho_1, \ldots, \rho_n\} \end{array}$$

10.2.2 Семантика деклараций

При описании семантики деклараций мы будем использовать вспомогательные ПВ, соответствующие процессам, определяемым в виде РО:

- $REG_x(y) = \alpha_x?z\ .\ REG_x(z)\ + \gamma_x!y\ .\ REG_x(y)$
  ($[\![REG_x(y)]\!]$ = регистр для хранения значения переменной x)
- $LOC_x \stackrel{\text{def}}{=} \alpha_x?y\ .\ REG_x(y)$
  (регистр без начального содержания)
- $SEM_x = \pi_x!\ .\ \varphi_x!\ .\ SEM_x$
- $SEM_R = \pi_R!\ .\ \varphi_R!\ .\ SEM_R$

Семантика деклараций имеет следующий вид.

1. Если $D =$ `VAR` x, то
$$\langle\!|D|\!\rangle \stackrel{\text{def}}{=} LOC_x \mid SEM_x$$
2. Если $D =$ `RESOURCE` R, то
$$\langle\!|D|\!\rangle \stackrel{\text{def}}{=} SEM_R$$
3. Если $D =$ `PROCEDURE` $G(u,v)$ `IS` C, то процессному выражению $\langle\!|D|\!\rangle$ соответствует РО
$$\langle\!|D|\!\rangle = \left(\begin{array}{l} LOC_u \mid \\ LOC_v \mid \\ \alpha_G?x\ .\ \alpha_u!x\ .\ ((\langle\!|D|\!\rangle \mid \langle\!|C|\!\rangle) \setminus L_G) \\ \quad \texttt{before}\ \gamma_v?y\ .\ \gamma_G!y\ .\ \langle\!|D|\!\rangle \end{array} \right) \setminus (L_u \cup L_v)$$

Если в РО для $\langle\!|$`PROCEDURE`$\ldots|\!\rangle$ вместо
$$(\langle\!|D|\!\rangle \mid \langle\!|C|\!\rangle) \setminus L_G$$
было бы написано просто $\langle\!|C|\!\rangle$, то неправильно транслировались бы такие процедуры, которые рекурсивно вызывают сами себя, т.е. такие G, в теле которых есть оператор `call` G.

Вышеприведённая семантика деклараций вида (10.3) является неидеальной. Она неправильно сопоставляет ПВ таким операторам, в которых есть параллельные вызовы одной и той же процедуры, например, оператору вида
$$\texttt{call}\ G(6,z)\ \texttt{par}\ \texttt{call}\ G(7,w)$$

Если известно максимальное число n доступных процессоров (т.е. одновременно n процедур могут быть активными), то семантику декларации D вида (10.3) лучше определить в виде РО

$$\left\{\begin{array}{l} \langle\!|D|\!\rangle = \langle\!|D|\!\rangle_1 \mid \ldots \mid \langle\!|D|\!\rangle_n \\ \forall\, i = 1, \ldots, n \\ \qquad \langle\!|D|\!\rangle_i = \alpha_{G,i}?x \;.\; \left(\begin{array}{l} LOC_u \mid \\ LOC_v \mid \\ \alpha_u!x \;.\; ((\langle\!|D|\!\rangle \mid \langle\!|C|\!\rangle) \setminus L_G) \\ \qquad \texttt{before}\ \gamma_v?y \;.\; \gamma_{G,k}!y \;.\; \langle\!|D|\!\rangle_i \end{array}\right) \setminus (L_u \cup L_v) \end{array}\right. \tag{10.4}$$

где $L_G \stackrel{\text{def}}{=} \{\alpha_{G,1}, \ldots, \alpha_{G,n},\ \gamma_{G,1}, \ldots, \gamma_{G,n}\}$.

10.2.3 Семантика операторов

1. $\langle\!|x := e|\!\rangle \stackrel{\text{def}}{=} \pi_x? \;.\; \langle\!|e|\!\rangle\ \texttt{result}\ (\rho?y \;.\; \alpha_x!y \;.\; \varphi_x? \;.\; \texttt{done})$

2. $\langle\!|C_1; C_2|\!\rangle \stackrel{\text{def}}{=} \langle\!|C_1|\!\rangle\ \texttt{before}\ \langle\!|C_2|\!\rangle$

3. $\langle\!|\texttt{if}\ e\ \texttt{then}\ C_1\ \texttt{else}\ C_2|\!\rangle \stackrel{\text{def}}{=}$
   $\stackrel{\text{def}}{=} \langle\!|e|\!\rangle\ \texttt{result}\ \rho?x \;.\; (x\ ?\ \langle\!|C_1|\!\rangle\ +\ \neg x\ ?\ \langle\!|C_2|\!\rangle)$

4. если $C = \texttt{while}\ e\ \texttt{do}\ C'$, то ПВ $\langle\!|C|\!\rangle$ связано со следующим РО:

$$\langle\!|C|\!\rangle = \langle\!|e|\!\rangle\ \texttt{result}\ (\rho?x. \left(\begin{array}{l} x\ ?\ \langle\!|C'|\!\rangle\ \texttt{before}\ \langle\!|C|\!\rangle\ + \\ \neg x\ ?\ \texttt{done} \end{array}\right))$$

5. $\langle\!|\texttt{begin}\ \{D_1, \ldots, D_k; C\}\ \texttt{end}|\!\rangle \stackrel{\text{def}}{=}$
   $\stackrel{\text{def}}{=} (\langle\!|D_1|\!\rangle \mid \ldots \mid \langle\!|D_k|\!\rangle \mid \langle\!|C|\!\rangle) \setminus (L_{D_1} \cup \ldots \cup L_{D_k})$

6. $\langle\!|C_1\ \texttt{par}\ C_2|\!\rangle \stackrel{\text{def}}{=} \langle\!|C_1|\!\rangle\ \texttt{par}\ \langle\!|C_2|\!\rangle$

7. $\langle\!|\texttt{input}\ x|\!\rangle \stackrel{\text{def}}{=} i?y \;.\; \alpha_x!y \;.\; \texttt{done}$

8. $\langle\!|\texttt{output}\ e|\!\rangle \stackrel{\text{def}}{=} \langle\!|e|\!\rangle\ \texttt{result}\ (\rho?x \;.\; o!x \;.\; \texttt{done})$

9. $\langle\!|\texttt{skip}|\!\rangle \stackrel{\text{def}}{=} \texttt{done}$

10. $\langle\!|\texttt{call}\ G(e, z)|\!\rangle \stackrel{\text{def}}{=}$
    $\stackrel{\text{def}}{=} \langle\!|e|\!\rangle\ \texttt{result}\ (\rho?x \;.\; \alpha_G!x \;.\; \gamma_G?z \;.\; \alpha_z!z \;.\; \texttt{done})$

11. $\langle\!|\texttt{with}\ R\ \texttt{do}\ C|\!\rangle \stackrel{\text{def}}{=} \pi_R? \;.\; \langle\!|C|\!\rangle\ \texttt{before}\ (\varphi_R? \;.\; \texttt{done})$

Если известно максимальное число n доступных процессоров (т.е. одновременно n процедур могут быть активными), и семантика деклараций вида (10.3) определяется в соответствии с (10.4), то семантику оператора $\texttt{call}\ G(e,z)$ следует определить следующим образом:

$$\langle\!|\texttt{call}\ G(e,z)|\!\rangle \stackrel{\text{def}}{=}$$
$$\stackrel{\text{def}}{=} \langle\!|e|\!\rangle\ \texttt{result}\ (\rho?x\ .\ \sum_{i=1}^{n}\ \alpha_{G,i}!x\ .\ \gamma_{G,i}?z\ .\ \alpha_z!z\ .\ \texttt{done})$$

Читателю предлагается самостоятельно

- определить семантику каких-либо современных языков параллельного или распределённого программирования (MPI и т.п.) или языков проектирования бизнес-процессов (BEPL и т.п.), и
- разработать на основе этой семантики автоматизированную систему доказательства свойств программ или описаний систем на этих языках.

Глава 11

Исторический обзор и современное состояние дел

Теория процессов объединяет несколько направлений исследований, каждое из которых отражает определённый подход к моделированию и анализу процессов. Ниже мы рассматриваем наиболее крупные из этих направлений.

11.1 Робин Милнер

Наибольший вклад в теорию процессов внёс выдающийся английский математик **Робин Милнер** (см. [1] - [5]).

Робин Милнер родился в 1934 г. в г. Плимуте (Англия) в семье военного офицера. В настоящее время (с 1995 г.) он работает профессором информатики в университете Кембриджа (`http://www.cam.ac.uk`). С января 1996 г. по октябрь 1999 г. Милнер занимал должность руководителя Компьютерной Лаборатории в университете Кембриджа.

В 1971-1973 г. Милнер стажировался в Лаборатории искусственного интеллекта в Стэнфордском университете. С 1973 г. по 1995 г. он работал в отделении информатики (Computer Science Department) Университета Эдинбурга (Шотландия), в котором в 1986 г. он совместно с коллегами основал Лабораторию основ компьютерных наук (Laboratory for Foundation of Computer Science).

С 1971 до 1980, в Стэнфорде и затем в Эдинбурге, он работал в области автоматизации рассуждений. Совместно с коллегами он разработал Логику Вычислимых Функций (Logic for Computable Functions, LCF), развивающую подход Д. Скотта к построению теоретических основ понятия вычислимости, и предназначенную для автоматизации формальных рассуждений. Эта работа послужила основой для прикладных систем, разработанных под руководством Милнера.

В 1975-1990 Милнер руководил группой, которая разрабатывала Standard

ML – широко используемый в индустрии и образовании язык программирования, семантика которого была полностью формализована (ML является сокращением словосочетания "meta-language"). В языке Standard ML был впервые реализован алгоритм вывода полиморфных типов. Главное достоинство Standard ML – возможность оперирования с логическими доказательствами и наличие средств автоматизации построения логических доказательств.

Около 1980 г. Милнер разработал Исчисление Взаимодействующих Систем (Calculus for Communicating Systems, CCS), одно из первых алгебраических исчислений для анализа параллельных процессов.

В конце 1980-х совместно с двумя коллегами он разработал π–исчисление, которое является основной моделью поведения мобильных взаимодействующих систем.

В 1988 г. Милнер был избран членом Королевского Общества. В 1991 г. ему была присуждена высшая награда в области информатики - премия имени А.М.Тьюринга.

Главную цель своей научной деятельности сам Милнер определяет как построение теории, объединяющей понятие вычисления с понятием взаимодействия.

11.2 Исчисление взаимодействующих систем (CCS)

Исчисление Взаимодействующих Систем (Calculus of Communicating Systems, CCS) было впервые опубликовано Милнером в 1980 г. в книге [89]. Стандартным учебником по CCS является книга [92].

В [89] отражены результаты исследований Милнера, проведённые им в период с 1973 по 1980 г.

Основные работы Милнера по моделям параллельных процессов, выполненные в этот период:

- работы [84], [85], в них Милнер исследует денотационную семантику параллельных процессов
- [83], [88], здесь, в частности, введено понятие потокового графа (flow graph) с синхронизированными портами,
- [86], [87], в этих работах появилось современное CCS.

Используемая в CCS модель взаимодействия параллельных процессов, основанная на понятии передачи сообщений, взята Милнером из работы Хоара [71].

В статье [66] исследуются сильная и наблюдаемая эквивалентности и вводится логика Хеннесси - Милнера.

Понятия, введённые в CCS, развивались и в рамках других подходов, среди которых следует отметить в первую очередь

- π–исчисление ([53], [97], [94]), и
- структурную операционную семантику (SOS), это направление было создано Г. Плоткиным, и опубликовано в работе [104].

Более подробная информация исторического характера о CCS может быть найдена в [105].

11.3 Теория взаимодействующих последовательных процессов (CSP)

Теория взаимодействующих последовательных процессов (Communicating sequential processes, CSP) была разработана английским математиком Тони Хоаром (C.A.R. Hoare) (р. в 1934). Эта теория возникла в 1976 г. и была опубликована в [71]. Более полное изложение CSP содержится в книге [73].

В CSP изучается модель взаимодействия параллельных процессов, основанная на передаче сообщений. Рассматривается синхронное взаимодействие между процессами.

Одним из ключевых понятий CSP является понятие охраняемой команды, которое Хоар позаимствовал из работы Дейкстры [52].

В [72] рассматривается модель CSP, основанная на теории трасс. Недостатком этой модели является отсутствие методов изучения дедлокового поведения. Этот недостаток устранён в другой модели CSP (failure model), введённой в [46].

11.4 Алгебра взаимодействующих процессов (ACP)

Ян Бергстра (Jan Bergstra) и Ян Виллем Клоп (Jan Willem Klop) в 1982 г. в работе [37] ввели термин "процессная алгебра" (process algebra), понимая по этим теорию первого порядка с равенством, в которой предметные переменные принимают значения в множестве процессов. Затем разработанные ими подходы привели к созданию нового направления в теории процессов - алгебры взаимодействующих процессов (Algebra of Communicating Processes, ACP), которое изложено в работах [39], [40], [34].

Главным объектом изучения в ACP являются логические теории, в которых испольуются функциональные символы, соответствующие операциям над процессами ($a.$, $+$, и т.д.)

В [30] проводится сравнительный анализ различных точек зрения на понятие процессной алгебры.

11.5 Процессные алгебры

Термин **процессная алгебра (ПА)**, введённый Бергстрой и Клопом, постепенно стал использоваться в двух значениях:

- в первом значении данный термин обозначает произвольную теорию первого порядка с равенством, областью интерпретации которой является некоторое множество процессов, и
- во втором значении данный термин обозначает большой класс направлений, каждое из которых связано с некоторой алгебраической теорией, описывающей свойства процессов, в этом значении данный термин используется, например, в названии книги "Справочная книга по процессной алгебре" (Handbook of Process Algebra) [42]

Ниже мы перечисляем наиболее важные источники, относящиеся к ПА в обоих значениях этого термина.

1. Справочная книга по ПА [42].
2. Краткий обзор основных результатов в ПА: [19].
3. Исторические обзоры: [27], [28], [15].
4. Различные подходы, связанные с понятием эквивалентности процессов: [101], [59], [57], [58], [56].
5. ПА с семантикой частичных порядков: [44].
6. ПА с рекурсией: [91], [47].
7. SOS-модель для ПА: [21], [38].
8. Алгебраические методы верификации: [63].
9. ПА с данными (действия и процессы параметризованы элементами данных)
    - ПА с данными μ-CRL
    - [62] (есть программное средство для верификации на основе излагаемого подхода).
    - PSF [79] (есть программное средство).

- Язык формальных описаний LOTOS [45].

10. ПА с временем (действия и процессы параметризованы моментами времени)

    - ПА с временем на основе CCS: [114], [99].
    - ПА с временем на основе CSP: [107]. Учебник: [109].
    - ПА с временем на основе ACP: [29].
    - Интеграция дискретного и плотного времени, относительного и абсолютного времени: [32].
    - Теория ATP: [100].
    - Учёт времени в БМ: [33].
    - Программное средство UPPAAL [74]
    - Программное средство KRONOS [116] (временные автоматы).
    - μ-CRL с временем: [111] (эквациональные рассуждения).

11. Вероятностные ПА (действия и процессы параметризованы вероятностями).

    Данные ПА предназначены для комбинированного исследования систем, при котором одновременно производится верификация, и анализ производительности.

    - Пионерская работа: [64].
    - Вероятностные ПА, основанные на CSP: [76]
    - Вероятностные ПА, основанные на CCS: [69]
    - Вероятностные ПА, основанные на ACP: [31].
    - ПА TIPP (и связанное с ней програмное средство): [60].
    - ПА EMPA: [43].
    - В работах [50] и [23] рассмотрено одновременное использование обычной и вероятностной альтернативной композиции процессов.
    - В работе [51] рассмотрено понятие аппроксимации вероятностных процессов.

12. Программные средства, относящиеся к ПА

    - Concurrency Workbench [98] (ПА типа CCS).
    - CWB-NC [117].
    - CADP [54].
    - CSP: FDR `http://www.fsel.com/`

11.6 Мобильные процессы

Мобильные процессы описывают поведение распределённых систем, во время функционирования которых может изменяться

- конфигурация связей между их компонентами, и
- структура этих компонентов.

Основные источники:

1. π-исчисление (Milner и др.):
   - старый справочник: [53],
   - стандартный справочник: [97],
   - учебники: [94], [8], [10], [9]
   - страница в Википедии: [14]
   - реализация π-исчисления на распределённой вычислительной системе: [115].
   - приложения π–исчисления к моделированию и верификации криптографических протоколов: [12]
2. The ambient calculus: [48].
3. Action calculus (Milner): [93]
4. Биграфы: [95], [96].
5. Обзор литературы по мобильным процессам: [11].
6. Программное средство: Mobility Workbench [112].
7. Сайт `www.cs.auc.dk/mobility`

Другие источники:

- Лекция Р. Милнера "Computing in Space" [6], которую он прочитал на открытии здания им. Билла Гейтса, построенном для Компьютерной Лаборатории университета Кембриджа 1 мая 2002 г.

  В лекции вводятся понятия "ambient" и "bigraph".
- Лекция Р. Милнера "Turing, Computing and Communication" [7].

11.7 Гибридные системы

Это системы, в которых

- значения одних переменных изменяются дискретно, и
- значения других переменных изменяются непрерывно.

Моделирование поведения таких систем производится с использованием дифференциальных и алгебраических уравнений.

Основные подходы:

- Гибридные процессные алгебры: [41], [49], [113].
- Гибридные автоматы [22] [77].

Для моделирования и верификации гибридных систем разработано программное средство HyTech [68].

11.8 Другие математические теории и программные средства, связанные с моделированием процессов

- Страница в Википедии по теории процессов [13].
- Теория сетей Петри [103].
- Теория частичных порядков [80].
- Темпоральная логика и model checking [106], [118].
- Теория трасс [108].
- Исчисление инвариантов [24].
- Метрический подход (в котором изучается понятие расстояния между процессами): [35], [36].
- SCCS [90].
- CIRCAL [82].
- MEIJE [25].
- Процессная алгебра Hennessy [65].

- Модели процессов с бесконечным числом состояний: [119], [120], [121], [122].
- Синхронно взаимодействующие автоматы: [123], [124], [125].
- Асинхронно взаимодействующие расширенные автоматы: [126]–[130].
- Фрмальные языки SDL [131], Estelle [132], LOTOS [133].
- Формализм Statecharts, введенный D. Harel'ом [134], [135] и входящий в язык проектирования UML.
- Модель взаимодействующих расширенных временных автоматов CETA (Communicating Extended Timed Automata) [136]–[140].
- A Calculus of Broadcasting Systems [17], [18].

11.9 Бизнес-процессы

1. BPEL (Business process execution language) [141].
2. BPML (Business Process Modeling Language) [16], [142].
3. Статья "Does Better Math Lead to Better Business Processes?" [143].
4. Страничка "π-calculus and Business Process Management" [144].
5. Статья "Workflow is just a π-process", Howard Smith and Peter Fingar, October 2003 [145].
6. "Третья волна" в моделировании бизнес-процессов: [146], [147].
7. Статья "Composition of executable business process models by combining business rules and process flows" [148].
8. Web services choreography description language [149].

Библиография

[1] Web-страничка Р.Милнера

http://www.cl.cam.ac.uk/~rm135/

[2] Страничка Р.Милнера в Википедии

http://en.wikipedia.org/wiki/Robin_Milner

[3] Интервью Р. Милнера

http://www.dcs.qmul.ac.uk/~martinb/interviews/milner/

[4] http://www.fairdene.com/picalculus/robinmilner.html

[5] http://www.cs.unibo.it/gorrieri/icalp97/Lauree_milner.html

[6] **R. Milner:** Computing in Space. *May, 2002.*

http://www.fairdene.com/picalculus/milner-computing-in-space.pdf

[7] **R. Milner:** Turing, Computing and Communication. *King's College, October 1997.*

http://www.fairdene.com/picalculus/milner-infomatics.pdf

[8] Учебное пособие по π-исчислению: the pi-calculus, a tutorial.

http://www.fairdene.com/picalculus/pi-c-tutorial.pdf

[9] **J. Parrow:** An introduction to the π-calculus. *[42], p. 479-543.*

[10] **D. Sangiorgi and D. Walker:** The π-calculus: A Theory of Mobile Processes. *ISBN 0521781779.*

http://us.cambridge.org/titles/catalogue.asp?isbn=0521781779

[11] **S. Dal Zilio:** Mobile Processes: a Commented Bibliography.

http://www.fairdene.com/picalculus/mobile-processes-bibliography.pdf

[12] **M. Abadi and A. D. Gordon:** A calculus for cryptographic protocols: The Spi calculus. *Journal of Information and Computation, 143:1-70, 1999.*

[13] Страница в Википедии "Process calculus"

http://en.wikipedia.org/wiki/Process_calculus

[14] Страница в Википедии по π-исчислению

http://en.wikipedia.org/wiki/Pi-calculus

[15] **J.C.M. Baeten:** A brief history of process algebra, *Rapport CSR 04-02, Vakgroep Informatica, Technische Universiteit Eindhoven, 2004*

http://www.win.tue.nl/fm/0402history.pdf

[16] Business Process Modeling Language

http://en.wikipedia.org/wiki/BPML

[17] http://en.wikipedia.org/wiki/Calculus_of_Broadcasting_Systems

[18] **K. V. S. Prasad:** A Calculus of Broadcasting Systems, *Science of Computer Programming, 25, 1995.*

[19] **L. Aceto:** Some of my favorite results in classic process algebra. *Technical Report NS-03-2, BRICS, 2003.*

[20] **L. Aceto, Z.T. Ésik, W.J. Fokkink, and A. Ingólfsdóttir (editors):** Process Algebra: Open Problems and Future Directions. *BRICS Notes Series NS-03-3, 2003.*

[21] **L. Aceto, W.J. Fokkink, and C. Verhoef:** Structural operational semantics. *In [42], pp. 197–292, 2001.*

[22] **R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine:** The algorithmic analysis of hybrid systems. *Theoretical Computer Science, 138:3–34, 1995.*

[23] **S. Andova:** Probabilistic Process Algebra. *PhD thesis, Technische Universiteit Eindhoven, 2002.*

[24] **K.R. Apt, N. Francez, and W.P. de Roever:** A proof system for communicating sequential processes. *TOPLAS, 2:359–385, 1980.*

[25] **D. Austry and G. Boudol:** Algébre de processus et synchronisation. *Theoretical Computer Science, 30:91–131, 1984.*

[26] **J.C.M. Baeten:** The total order assumption. *In S. Purushothaman and A. Zwarico, editors, Proceedings First North American Process Algebra Workshop, Workshops in Computing, pages 231–240. Springer Verlag, 1993.*

[27] **J.C.M. Baeten:** Over 30 years of process algebra: Past, present and future. *In L. Aceto, Z. T. Ésik, W.J. Fokkink, and A. Ingólfsdóttir, editors, Process Algebra: Open Problems and Future Directions, volume NS-03-3 of BRICS Notes Series, pages 7–12, 2003.*

[28] `http://www.win.tue.nl/fm/pubbaeten.html`

[29] **J.C.M. Baeten and J.A. Bergstra:** Real time process algebra. *Formal Aspects of Computing, 3(2):142–188, 1991.*

[30] **J.C.M. Baeten, J.A. Bergstra, C.A.R. Hoare, R. Milner, J. Parrow, and R. de Simone:** The variety of process algebra. *Deliverable ESPRIT Basic Research Action 3006, CONCUR, 1991.*

[31] **J.C.M. Baeten, J.A. Bergstra, and S.A. Smolka:** Axiomatizing probabilistic processes: ACP with generative probabilities. *Information and Computation, 121(2):234–255, 1995.*

[32] **J.C.M. Baeten and C.A. Middelburg:** Process Algebra with Timing. *EATCS Monographs. Springer Verlag, 2002.*

[33] **J.C.M. Baeten, C.A. Middelburg, and M.A. Reniers:** A new equivalence for processes with timing. *Technical Report CSR 02-10, Eindhoven University of Technology, Computer Science Department, 2002.*

[34] **J.C.M. Baeten and W.P. Weijland:** Process Algebra. *Number 18 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1990.*

[35] **J.W. de Bakker and J.I. Zucker:** Denotational semantics of concurrency. *In Proceedings 14th Symposium on Theory of Computing, pages 153–158. ACM, 1982.*

[36] **J.W. de Bakker and J.I. Zucker:** Processes and the denotational semantics of concurrency. *Information and Control, 54:70–120, 1982.*

[37] **J.A. Bergstra and J.W. Klop:** Fixed point semantics in process algebra. *Technical Report IW 208, Mathematical Centre, Amsterdam, 1982.*

[38] **J.A. Bergstra and J.W. Klop:** The algebra of recursively defined processes and the algebra of regular processes. *In J. Paredaens, editor, Proceedings 11th ICALP, number 172 in LNCS, pages 82–95. Springer Verlag, 1984.*

[39] **J.A. Bergstra and J.W. Klop:** Process algebra for synchronous communication. *Information and Control, 60(1/3):109–137, 1984.*

[40] **J.A. Bergstra and J.W. Klop:** A convergence theorem in process algebra. *In J.W. de Bakker and J.J.M.M. Rutten, editors, Ten Years of Concurrency Semantics, pages 164–195. World Scientific, 1992.*

[41] **J.A. Bergstra and C.A. Middelburg:** Process algebra semantics for hybrid systems. *Technical Report CS-R 03/06, Technische Universiteit Eindhoven, Dept. of Comp. Sci., 2003.*

[42] **J.A. Bergstra, A. Ponse, and S.A. Smolka, editors:** Handbook of Process Algebra. *North-Holland, Amsterdam, 2001.*

[43] **M. Bernardo and R. Gorrieri:** A tutorial on EMPA: A theory of concurrent processes with non-determinism, priorities, probabilities and time. *Theoretical Computer Science, 202:1–54, 1998.*

[44] **E. Best, R. Devillers, and M. Koutny:** A unified model for nets and process algebras. *In [42], pp. 945–1045, 2001.*

[45] **E. Brinksma (editor):** Information Processing Systems, Open Systems Interconnection, LOTOS – A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour, *volume IS-8807 of International Standard. ISO, Geneva, 1989.*

[46] **S.D. Brookes, C.A.R. Hoare, and A.W. Roscoe:** A theory of communicating sequential processes. *Journal of the ACM, 31(3):560–599, 1984.*

[47] **O. Burkart, D. Caucal, F. Moller, and B. Steffen:** Verification on infinite structures. *In [42], pp. 545–623, 2001.*

[48] **L. Cardelli and A.D. Gordon:** Mobile ambients. *Theoretical Computer Science, 240:177–213, 2000.*

[49] **P.J.L. Cuijpers and M.A. Reniers:** Hybrid process algebra. *Technical Report CS-R 03/07, Technische Universiteit Eindhoven, Dept. of Comp. Sci., 2003.*

[50] **P.R. D'Argenio:** Algebras and Automata for Timed and Stochastic Systems. *PhD thesis, University of Twente, 1999.*

[51] **J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden:** Metrics for labeled Markov systems. *In J.C.M. Baeten and S. Mauw, editors, Proceedings CONCUR'99, number 1664 in Lecture Notes in Computer Science, pages 258–273. Springer Verlag, 1999.*

[52] **E.W. Dijkstra:** Guarded commands, nondeterminacy, and formal derivation of programs. *Communications of the ACM, 18(8):453– 457, 1975.*

[53] **U. Engberg and M. Nielsen:** A calculus of communicating systems with label passing. *Technical Report DAIMI PB-208, Aarhus University, 1986.*

[54] **J.-C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier, and M. Sighireanu:** CADP (CAESAR/ALDEBARAN development package): A protocol validation and verification toolbox. *In R. Alur and T.A. Henzinger, editors, Proceedings CAV '96, number 1102 in Lecture Notes in Computer Science, pages 437–440. Springer Verlag, 1996.*

[55] **R.W. Floyd:** Assigning meanings to programs. *In J.T. Schwartz, editor, Proceedings Symposium in Applied Mathematics, Mathematical Aspects of Computer Science, pages 19–32. AMS, 1967.*

[56] **R.J. van Glabbeek:** The linear time – branching time spectrum II; the semantics of sequential systems with silent moves. *In E. Best, editor, Proceedings CONCUR '93, number 715 in Lecture Notes in Computer Science, pages 66–81. Springer Verlag, 1993.*

[57] **R.J. van Glabbeek:** What is branching time semantics and why to use it? *In M. Nielsen, editor, The Concurrency Column, pages 190–198. Bulletin of the EATCS 53, 1994.*

[58] **R.J. van Glabbeek:** The linear time – branching time spectrum I. The semantics of concrete, sequential processes. *In [42], pp. 3–100, 2001.*

[59] **R.J. van Glabbeek and W.P. Weijland:** Branching time and abstraction in bisimulation semantics. *Journal of the ACM, 43:555–600, 1996.*

[60] **N. Götz, U. Herzog, and M. Rettelbach:** Multiprocessor and distributed system design: The integration of functional specification and performance analysis using stochastic process algebras. *In L. Donatiello and R. Nelson, editors, Performance Evaluation of Computer and Communication Systems, number 729 in LNCS, pages 121–146. Springer, 1993.*

[61] **J.F. Groote:** Process Algebra and Structured Operational Semantics. *PhD thesis, University of Amsterdam, 1991.*

[62] **J.F. Groote and B. Lisser:** Computer assisted manipulation of algebraic process specifications. *Technical Report SEN-R0117, CWI, Amsterdam, 2001.*

[63] **J.F. Groote and M.A. Reniers:** Algebraic process verification. *In [42], pp. 1151–1208, 2001.*

[64] **H. Hansson:** Time and Probability in Formal Design of Distributed Systems. *PhD thesis, University of Uppsala, 1991.*

[65] **M. Hennessy:** Algebraic Theory of Processes. *MIT Press, 1988.*

[66] **M. Hennessy and R. Milner:** On observing nondeterminism and concurrency. *In J.W. de Bakker and J. van Leeuwen, editors, Proceedings 7th ICALP, number 85 in Lecture Notes in Computer Science, pages 299–309. Springer Verlag, 1980.*

[67] **M. Hennessy and G.D. Plotkin:** Full abstraction for a simple parallel programming language. *In J. Becvar, editor, Proceedings MFCS, number 74 in LNCS, pages 108– 120. Springer Verlag, 1979.*

[68] **T.A. Henzinger, P. Ho, and H. Wong-Toi:** Hy-Tech: The next generation. *In Proceedings RTSS, pages 56–65. IEEE, 1995.*

[69] **J. Hillston:** A Compositional Approach to Performance Modelling. *PhD thesis, Cambridge University Press, 1996.*

[70] **C.A.R. Hoare:** An axiomatic basis for computer programming. *Communications of the ACM, 12:576–580, 1969.*

[71] **C.A.R. Hoare:** Communicating sequential processes. *Communications of the ACM, 21(8):666–677, 1978.*

[72] **C.A.R. Hoare:** A model for communicating sequential processes. *In R.M. McKeag and A.M. Macnaghten, editors, On the Construction of Programs, pages 229–254. Cambridge University Press, 1980.*

[73] **C.A.R. Hoare:** Communicating Sequential Processes. *Prentice Hall, 1985.*

[74] **K.G. Larsen, P. Pettersson, and Wang Yi:** Uppaal in a nutshell. *Journal of Software Tools for Technology Transfer, 1, 1997.*

[75] **P. Linz:** An Introduction to Formal Languages and Automata. *Jones and Bartlett, 2001.*

[76] **G. Lowe:** Probabilities and Priorities in Timed CSP. *PhD thesis, University of Oxford, 1993.*

[77] **N. Lynch, R. Segala, F. Vaandrager, and H.B. Weinberg:** Hybrid I/O automata. *In T. Henzinger, R. Alur, and E. Sontag, editors, Hybrid Systems III, number 1066 in Lecture Notes in Computer Science. Springer Verlag, 1995.*

[78] **S. MacLane and G. Birkhoff:** Algebra. *MacMillan, 1967.*

[79] **S. Mauw:** PSF: a Process Specification Formalism. *PhD thesis, University of Amsterdam, 1991.*

`http://carol.science.uva.nl/~psf/`

[80] **A. Mazurkiewicz:** Concurrent program schemes and their interpretations. *Technical Report DAIMI PB-78, Aarhus University, 1977.*

[81] **J. McCarthy:** A basis for a mathematical theory of computation. *In P. Braffort and D. Hirshberg, editors, Computer Programming and Formal Systems, pages 33–70. North-Holland, Amsterdam, 1963.*

[82] **G.J. Milne:** CIRCAL: A calculus for circuit description. *Integration, 1:121–160, 1983.*

[83] **G.J. Milne and R. Milner:** Concurrent processes and their syntax. *Journal of the ACM, 26(2):302–321, 1979.*

[84] **R. Milner:** An approach to the semantics of parallel programs. *In Proceedings Convegno di informatica Teoretica, pages 285– 301, Pisa, 1973. Instituto di Elaborazione della Informazione.*

[85] **R. Milner:** Processes: A mathematical model of computing agents. *In H.E. Rose and J.C. Shepherdson, editors, Proceedings Logic Colloquium, number 80 in Studies in Logic and the Foundations of Mathematics, pages 157–174. North-Holland, 1975.*

[86] **R. Milner:** Algebras for communicating systems. *In Proc. AFCET/SMF joint colloquium in Applied Mathematics, Paris, 1978.*

[87] **R. Milner:** Synthesis of communicating behaviour. *In J. Winkowski, editor, Proc. 7th MFCS, number 64 in LNCS, pages 71–83, Zakopane, 1978. Springer Verlag.*

[88] **R. Milner:** Flowgraphs and flow algebras. *Journal of the ACM, 26(4):794–818, 1979.*

[89] **R. Milner:** A Calculus of Communicating Systems. *Number 92 in Lecture Notes in Computer Science. Springer Verlag, 1980.*

[90] **R. Milner:** Calculi for synchrony and asynchrony. *Theoretical Computer Science, 25:267–310, 1983.*

[91] **R. Milner:** A complete inference system for a class of regular behaviours. *Journal of Computer System Science, 28:439–466, 1984.*

[92] **R. Milner:** Communication and Concurrency. *Prentice Hall, 1989.*

[93] **R. Milner:** Calculi for interaction. *Acta Informatica, 33:707–737, 1996.*

[94] **R. Milner:** Communicating and Mobile Systems: the π-Calculus. *Cambridge University Press, ISBN 052164320, 1999.*

`http://www.cup.org/titles/catalogue.asp?isbn=0521658691`

[95] **R. Milner:** Bigraphical reactive systems. *In K.G. Larsen and M. Nielsen, editors, Proceedings CONCUR '01, number 2154 in LNCS, pages 16–35. Springer Verlag, 2001.*

[96] **O. Jensen and R. Milner** Bigraphs and Mobile Processes. *Technical report, 570, Computer Laboratory, University of Cambridge, 2003.*

`http://citeseer.ist.psu.edu/jensen03bigraphs.html`

`http://citeseer.ist.psu.edu/668823.html`

[97] **R. Milner, J. lParrow, and D. Walker:** A calculus of mobile processes. *Information and Computation, 100:1–77, 1992.*

[98] **F. Moller and P. Stevens:** Edinburgh Concurrency Workbench user manual (version 7.1).

`http://www.dcs.ed.ac.uk/home/cwb/`

[99] **F. Moller and C. Tofts:** A temporal calculus of communicating systems. *In J.C.M. Baeten and J.W. Klop, editors, Proceedings CONCUR'90, number 458 in LNCS, pages 401–415. Springer Verlag, 1990.*

[100] **X. Nicollin and J. Sifakis:** The algebra of timed processes ATP: Theory and application. *Information and Computation, 114:131– 178, 1994.*

[101] **D.M.R. Park:** Concurrency and automata on infinite sequences. *In P. Deussen, editor, Proceedings 5th GI Conference, number 104 in LNCS, pages 167–183. Springer Verlag, 1981.*

[102] **C.A. Petri:** Kommunikation mit Automaten. *PhD thesis, Institut fuer Instrumentelle Mathematik, Bonn, 1962.*

[103] **C.A. Petri:** Introduction to general net theory. *In W. Brauer, editor, Proc. Advanced Course on General Net Theory, Processes and Systems, number 84 in LNCS, pages 1–20. Springer Verlag, 1980.*

[104] **G.D. Plotkin:** A structural approach to operational semantics. *Technical Report DAIMI FN-19, Aarhus University, 1981.*

[105] **G.D. Plotkin:** The origins of structural operational semantics. *Journal of Logic and Algebraic Programming, Special Issue on Structural Operational Semantics, 2004.*

[106] **A. Pnueli:** The temporal logic of programs. *In Proceedings 19th Symposium on Foundations of Computer Science, pages 46–57. IEEE, 1977.*

[107] **G.M. Reed and A.W. Roscoe:** A timed model for communicating sequential processes. *Theoretical Computer Science, 58:249–261, 1988.*

[108] **M. Rem:** Partially ordered computations, with applications to VLSI design. *In J.W. de Bakker and J. van Leeuwen, editors, Foundations of Computer Science IV, volume 159 of Mathematical Centre Tracts, pages 1–44. Mathematical Centre, Amsterdam, 1983.*

[109] **S.A. Schneider:** Concurrent and Real-Time Systems (the CSP Approach). *Worldwide Series in Computer Science. Wiley, 2000.*

[110] **D.S. Scott and C. Strachey:** Towards a mathematical semantics for computer languages. *In J. Fox, editor, Proceedings Symposium Computers and Automata, pages 19–46. Polytechnic Institute of Brooklyn Press, 1971.*

[111] **Y.S. Usenko:** Linearization in μCRL. *PhD thesis, Technische Universiteit Eindhoven, 2002.*

[112] **B. Victor:** A Verification Tool for the Polyadic π-Calculus. *Licentiate thesis, Department of Computer Systems, Uppsala University, Sweden, May 1994. Report DoCS 94/50.*

[113] **T.A.C. Willemse:** Semantics and Verification in Process Algebras with Data and Timing. *PhD thesis, Technische Universiteit Eindhoven, 2003.*

[114] **Wang Yi:** Real-time behaviour of asynchronous agents. *In J.C.M. Baeten and J.W. Klop, editors, Proceedings CONCUR'90, number 458 in LNCS, pages 502– 520. Springer Verlag, 1990.*

[115] **L. Wischik:** New directions in implementing the π-calculus. *University of Bologna, August 2002.*

`http://www.fairdene.com/picalculus/implementing-pi-c.pdf`

[116] **S. Yovine:** Kronos: A verification tool for real-time systems. *Journal of Software Tools for Technology Transfer, 1:123–133, 1997.*

[117] **D. Zhang, R. Cleaveland, and E. Stark:** The integrated CWB-NC/PIOAtool for functional verification and performance analysis of concurrent systems. *In H. Garavel and J. Hatcliff, editors, Proceedings TACAS'03, number 2619 in Lecture Notes in Computer Science, pages 431–436. Springer-Verlag, 2003.*

[118] **E. Clarke, O. Grumberg, D. Peled:** Model checking. *MIT Press, 2001.*

[119] **J.Esparza:** Decidability of model-checking for infinite-state concurrent systems, *Acta Informatica, 34:85-107, 1997.*

[120] **P.A.Abdulla, A.Annichini, S.Bensalem, A.Bouajjani, P.Habermehl, Y.Lakhnech:** Verification of Infinite-State Systems by Combining Abstraction and Reachability Analysis, *Lecture Notes in Computer Science 1633, pages 146-159, Springer-Verlag, 1999.*

[121] **K.L.McMillan**: Verification of Infinite State Systems by Compositional Model Checking, *Conference on Correct Hardware Design and Verification Methods, pages 219-234, 1999.*

[122] **O.Burkart, D.Caucal, F.Moller, and B.Steffen:** Verification on infinite structures, *In J. Bergstra, A. Ponse and S. Smolka, editors, Handbook of Process Algebra, chapter 9, pages 545-623, Elsevier Science, 2001.*

[123] **D. Lee and M. Yannakakis.** Principles and Methods of Testing Finite State Machines - a Survey. The Proceedings of the IEEE, 84(8), pp 1090-1123, 1996.

[124] **G. Holzmann.** Design and Validation of Computer Protocols. Prentice-Hall, Englewood Cliffs, N.J., first edition, 1991.

[125] **G. Holzmann.** The SPIN Model Checker - Primer and Reference Manual. Addison-Wesley, 2003.

[126] **S. Huang, D. Lee, and M. Staskauskas.** Validation-Based Test Sequence Generation for Networks of Extended Finite State Machines. In Proceedings of FORTE/PSTV, October 1996.

[127] **J. J. Li and M. Segal.** Abstracting Security Specifications in Building Survivable Systems. In Proceedings of 22-тв National Information Systems Security Conference, October 1999, Arlington, Virginia, USA.

[128] **Y.- J. Byun, B. A. Sanders, and C.-S. Keum.** Design Patterns of Communicating Extended Finite State Machines in SDL . In Proceedings of 8-th Conference on Pattern Languages of Programs (PLoP'2001), September 2001, Monticello, Illinois, USA.

[129] **J. J. Li and W. E. Wong.** Automatic Test Generation from Communicating Extended Finite State Machine (CEFSM)-Based Models. In Proceedings of the Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'02), pp. 181-185, 2002.

[130] **S. Chatterjee.** EAI Testing Automation Strategy. In Proceedings of 4-th QAI Annual International Software Testing Conference in India, Pune, India, February 2004

[131] ITU Telecommunication Standardization Sector (ITU-T), Recommendation Z.100, CCITT Specification and Description Language (SDL), Geneva 1994.

[132] Information Processing Systems - Open Systems Interconnection: Estelle, A Formal Description Technique Based on Extended State Transition Model, ISO International Standard 9074, June 1989.

[133] ISOIEC. LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. International Standard 8807, 1988.

[134] **D. Harel.** A Visual Formalism for Complex Systems. Science of Computer Programming, 8:231-274, 1987.

[135] **D. Harel and A. Naamad.** The STATEMATE semantics of statecharts. ACM Transactions on Software Engineering and Methodology (Also available as technical report of Weizmann Institute of Science, CS95-31), 5(4):293-333, Oct 1996.

[136] **M. Bozga, J. C. Fernandez, L. Ghirvu, S. Graf, J. P. Krimm, and L. Mounier.** IF: An intermediate representation and validation environment for timed asynchronous systems. In Proceedings of Symposium on Formal Methods 99, Toulouse, number 1708 in LNCS. Springer Verlag, September 1999.

[137] **M. Bozga, S. Graf, and L. Mounier.** IF-2.0: A validation environment for componentbased real-time systems. In Proceedings of Conference on Computer Aided Verification, CAV'02, Copenhagen, LNCS. Springer Verlag, June 2002.

[138] **M. Bozga, D. Lesens, and L. Mounier.** Model-Checking Ariane-5 Flight Program. In Proceedings of FMICS'01, Paris, France, pages 211-227. INRIA, 2001.

[139] **M. Bozga, S. Graf, and L. Mounier.** Automated validation of distributed software using the IF environment. In 2001 IEEE International Symposium on Network Computing and Applications (NCA 2001). IEEE, October 2001.

[140] **M. Bozga and Y. Lakhnech.** IF-2.0 common language operational semantics. Technical report, 2002. Deliverable of the IST Advance project, available from the authors.

[141] `http://www-128.ibm.com/developerworks/library/specification/ws-bpel/`

[142] http://www.bpml.org

[143] http://www.wfmc.org/standards/docs/
better_maths_better_processes.pdf

[144] http://www.fairdene.com/picalculus/

[145] http://www.bpmi.org/bpmi-library/
2B6EA45491.workflow-is-just-a-pi-process.pdf

[146] http://www.fairdene.com/picalculus/
bpm3-apx-theory.pdf

[147] http://www.bpm3.com

[148] http://portal.acm.org/citation.cfm?id=1223649

[149] http://www.w3.org/TR/ws-cdl-10/