Prentice-Hall International Series In Computer Science

Communicating Sequential Processes

C. A. R. Hoare
Professor of Computation
Oxford University

Prentice-Hall
Englewood Cliffs, New Jersey London Mexico
New Delhi Rio de Janeiro Singapore
Sydney Tokyo Toronto Wellington

Ч. Хоар

Взаимодействующие последовательные процессы

Перевод с английского
А. А. Бульонковой
под редакцией
А. П. Ершова

Москва «Мир» 1989

ББК 22.18
Х68
УДК 681.3

Хоар Ч.
Х68 Взаимодействующие последовательные процессы:
Пер. с англ.– М.: Мир, 1989.– 264 с., ил.
ISBN 5-03-001043-2

Книга известного системного программиста и теоретика информатики (Великобритания), последовательно излагающая теорию взаимодействующих процессов; эта тематика тесно связана с такими реальными понятиями, как операционные системы, мультипроцессорные комплексы и сети ЭВМ. Автор рассматривает параллелизм в языках высокого уровня АДА, Симула 67, Паскаль.

Для специалистов в области системного программирования, теоретической информатики, математической логики, аспирантов и студентов вузов.

1702050000–480
Х——————— 22–88, ч. 1 **ББК 22. 18**
041(01)–89

Редакция литературы по математическим наукам

ISBN 5-03-001043-2 (русск.)
UK, LTD ISBN 0-13-153271-5 (англ.)

От редактора перевода

Эта книга написана выдающимся ученым, лауреатом премии Тьюринга, внесшим большой вклад в теорию и практику программирования. Автор написал к книге подробное предисловие, в котором обстоятельно изложил свой подход к предлагаемому материалу.

Другой, не менее выдающийся ученый, тоже лауреат премии Тьюринга, написал изящное вступление, воздающее должное автору и подогревающее интерес к книге.

В этом контексте намерение писать третье предисловие, хотя бы и по праву редактора перевода, легко может быть воспринято как желание «войти в компанию». Мою решимость поддерживает сознание того, что мне хочется не столько вписать еще пару абзацев в панегирик автору (хотя и тут мне было бы что сказать), сколько побудить читателя к критическому прочтению этой книги.

Здесь уместно сделать одно пояснение. Мы зачастую несколько упрощенно представляем себе, что критиковать – это значит выискивать недостатки. Не желая спорить с этим расхожим представлением, хочу добавить, что критический разбор произведения – это способность приобщиться к мысли и видению автора, сохраняя запас широты восприятия, независимость мышления и самостоятельность собственной отправной позиции.

Предупредить об этом меня побуждает благоприятное смешение стилей, случившееся в этой книге: это монография, написанная с убедительностью учебника. Вспомним словарь иностранных слов: «Монография – научный труд, углубленно разрабатывающий одну тему, ограниченный круг вопросов». Автор «Взаимодействующих последовательных процессов» специально подчеркивает монографический характер книги и свой авторский подход к посылкам, границам и целям исследования. В то же время глубина и скрупулезность проработки, редкая для нашей программистской «лохматости» элегантность обозначений и конструкций, прекрасно подобранные примеры и крепко сбитая структура книги создают комфортабельное ощущение убедительности и своего рода однозначности материала книги.

С этой книгой удобно работать, потому что она написана как отличный учебник. Однако ее нельзя изучать по-школпрски, потому что это монографическое исследование в новом и труднейшем разделе программирования. Изложенный материал не исчерпывает всех проблем на долгом пути жизненного цикла разработки программ, а сам подход к решению этих проблем ограничивается особенностями выбранной математической модели.

Академик С. Л. Соболев как-то сказал нашему студенческому курсу: «Готовиться к экзамену по плохим лекциям даже лучше. Пробелы в конспектах заставят вас их восстановить». Эта книга не дает читателю подобного шанса. Важно, однако, сказать, что законченность изложения и другие неординарные качества книги являются результатом двадцатилетнего труда автора и неоднократной переработки вариантов. Если читатель примет на себя хотя бы малую долю этого труда во время работы над книгой, он сможет удовлетворенно заметить, что не только приобщился к предмету изложения, но и продвинул еще на один шаг развитие теории и методов параллельного программирования.

Академгородок,
январь 1988 *А. П. Ершов*

Предисловие

По многим причинам эту книгу с интересом ждали все, кто знал о планах ее создания; сказать, что их терпение вознаграждено, означало бы не сказать почти ничего.

Причина проста: это первая книга Тони Хоара. Многие знают его по лекциям, с которыми он неустанно выступает по всему свету; гораздо большему числу он знаком как искусный и вдумчивый автор изрядного количества (и разнообразия!), статей, которые становились классикой раньше, чем успевала высохнуть типографская краска. Но книга – это нечто другое: здесь автор свободен от стеснительных ограничений сроков и объема; книга предоставляет ему возможность глубже выразить себя и охватить тему более широко. Эти возможности Тони Хоар использовал лучше, чем мы могли себе представить.

Более серьезная причина кроется в непосредственном содержании книги. Когда примерно четверть века назад сообщество программистов столкнулось с параллелизмом, это вызвало бесконечную путаницу, частично из-за крайнего разнообразия источников параллелизма, частично из-за того, что волею судеб это совпало по времени с началом изучения проблем недетерминизма. Для развязки этой путаницы требовался тяжелый труд зрелого и целеустремленного ученого, которому при некотором везении удалось бы прояснить ситуацию. Тони Хоар посвятил значительную часть своей научной жизни этой работе, и у нас есть тысяча причин быть ему благодарными за это.

Основная причина, однако, наиболее остро была осознана теми, кому довелось увидеть ранние варианты рукописи книги и кто с удивительной ясностью вдруг увидел в новом свете, какой могла бы – или даже должна бы – быть вычислительная наука. Провозгласить или почувствовать, что главная задача специалиста в области информатики состоит в том, чтобы не оказаться погребенным под сложностью своих же собственных конструкций, – это одно, и совсем другое дело – обнаружить и показать, как строгое подчинение совершенно явной и осязаемой элегантности небольшого числа математических законов позволяет достичь этой высокой цели. Именно в этом, смею думать, признательные читатели в наибольшей степени извлекут пользу из научной мудрости, нотационной безупречности и аналитического искусства Чарльза Энтони Ричарда Хоара,

Эдсгер В. Дейкстра

От автора

Эта книга – для ищущего программиста, программиста, который стремится глубже понять и лучше овладеть практическим искусством своей наукоемкой профессии. Прежде всего, книга взывает к естественному любопытству, возникающему благодаря новому подходу к хорошо знакомым вещам. Этот подход иллюстрируется большим числом практических примеров, взятых из самого широкого круга приложений: от торговых автоматов, игр и сказочных историй до операционных систем ЭВМ. В основе изложения лежит математическая теория, описанная в виде систематического набора алгебраических законов. Конечной целью является выработка у читателя особой проницательности, которая даст ему возможность увидеть как текущие, так и будущие проблемы в новом свете, что позволит решать эти проблемы экономнее и надежнее и, что еще важнее, порой избегать их.

Наиболее очевидной сферой применения новых идей служит спецификация, разработка и реализация вычислительных систем, которые непрерывно действуют и взаимодействуют со своим окружением. Основная идея заключается в том, что эти системы без труда можно разложить на параллельно работающие подсистемы, взаимодействующие как друг с другом, так и со своим общим окружением. Параллельная композиция подсистем ничуть не сложнее последовательного сочетания строк или операторов в обычных языках программирования.

Такой подход обладает целым рядом преимуществ. Во-первых, он позволяет избежать многих традиционных для параллельного программирования проблем, таких, как взаимное влияние и взаимное исключение, прерывания, семафоры, многопоточная обработка и т. д. Во-вторых, он включает в себя в виде частных случаев многие из актуальных идей структурирования, которые используются в современных исследованиях по языкам и методологии программирования; мониторы, классы, модули, пакеты, критические участки, конверты, формы и даже заурядные подпрограммы. И, наконец, этот подход является надежной основой для избежания таких ошибок, как расходимость, тупиковые ситуации, зацикливание, а также для доказательства правильности при проектировании и разработке вычислительных систем.

Я стремился изложить свои мысли в определенной логической и психологической последовательности, начиная с простых элементарных операторов и постепенно переходя к более сложным приложениям. Усердный читатель, возможно, прочтет всю книгу целиком, от корки до корки. Но у многих одни разделы вызовут больший интерес, чем другие; в помощь им, чтобы облегчить разумный выбор, материал каждой главы тщательно структурирован.

(1) Каждая новая идея предварена ее неформальным описанием и проиллюстрирована рядом небольших примеров, полезных, вероятно, всем читателям.

(2) Алгебраические законы, описывающие основные свойства различных операций, будут интересны читателям, ценящим в математике красоту. Они окажутся полезными также тем, кто хочет оптимизировать разработку своих систем с помощью преобразований, сохраняющих корректность.

(3) Необычность предложенных способов реализации состоит в том, что в них используется очень простое и чисто функциональное подмножество языка программирования ЛИСП. Это доставит особенное удовольствие тем, кто работает с ЛИСПом, и благодаря этому получит возможность реализовать и испытать свои замыслы.

(4) Определения протоколов и спецификаций заинтересуют специалистов-системщиков, которым приходится специфицировать требования заказчика, прежде чем приступать к исполнению. Эти понятия пригодятся также ведущим программистам, перед которыми стоит задача проектирования системы путем разбиения ее на подсистемы с четко описанными интерфейсами.

(5) Правила доказательств будут интересны тем, кто серьезно относится к стоящей перед ними задаче написания надежных программ по заданной спецификации в установленные сроки и с фиксированными затратами.

(6) И, наконец, эта математическая теория дает строгие определения понятия процесса и способов построения процессов. Эти определения лежат в основе алгебраических законов, реализаций и правил доказательства.

Читатель может, либо, придерживаясь своей системы, либо выборочно, опускать или откладывать на потом те из перечисленных разделов, которые или менее интересны, или вызывают трудности в понимании.

Структура книги позволяет взыскательному читателю либо ограничиться беглым просмотром, либо

определить порядок

чтения и выбор глав. Первые разделы гл. 1 и 2 представляют собой введение, необходимое для всех читателей, тогда как их последующие разделы допускают более поверхностное знакомство или же могут быть отложены до следующего прочтения. Главы 3, 4 и 5 не связаны друг с другом, и знакомство с ними может происходить в любом порядке и комбинации в зависимости от интересов и намерений читателя. Поэтому, если на некотором этапе возникнут трудности с пониманием, желательно продолжить чтение со следующего раздела или даже главы, так как, вполне вероятно, пропущенный материал не потребуется немедленно. Если же потребность в нем возникает, то, как правило, дается явная ссылка на предыдущий материал, которой при необходимости можно воспользоваться. Я надеюсь, что в конце концов все в этой книге окажется интересным и полезным, однако порядок чтения и проработки может быть индивидуальным.

Примеры, выбранные для иллюстрации заключенных в книге идей, покажутся очень простыми. Это сделано сознательно. Первые примеры, иллюстрирующие каждую новую идею, должны быть просты настолько, чтобы сложность и необычность самого примера не могли затемнить идею. Некоторые из последующих примеров отличаются большей тонкостью; стоящие за ними проблемы способны породить немало путаницы и сложностей. Простоту их решения следует отнести за счет мощности используемых понятий и элегантности выражающей их нотации.

Тем не менее, каждый читатель знаком (возможно, до боли знаком) с проблемами куда более сложными, обширными и важными, нежели те, которые служат примерами во вступительной части. Может показаться, что эти проблемы не под силу никакой математической теории. Мой совет – не поддаваться ни гневу, ни отчаянию, а попробовать испытать на этих проблемах новый метод. Начните с самого упрощенного случая какой-нибудь выбранной вами стороны проблемы и постепенно усложняйте ее в меру необходимости. Просто удивительно, как часто изначально переупрощенная модель позволяет увидеть путь к решению всей проблемы в целом. Возможно, ваша модель послужит структурой, на которую в дальнейшем без риска будут накладываться разные усложнения. И главным сюрпризом будет то, что многие из этих усложнений, вероятно окажутся попросту ненужными. В этом случае усилия по освоению нового метода вознаграждаются особенно щедро.

Обозначения часто служат причиной недовольства. Студент, начинающий изучение русского языка, жалуется на трудности запоминания незнакомых букв кириллицы, особенно из-за того, что многие из них имеют непривычное произношение. В порядке утешения скажу, что эта проблема – далеко не главная. Вслед за алфавитом вам предстоит изучить грамматику, накопить словарный запас, затем овладеть знанием идиом и стиля, а после этого – обрести способность свободно выражать на этом языке свои мысли. Все это требует старания, тренировки и времени, и поспешности здесь не место. Так же ив математике. Поначалу символы могут оказаться серьезным препятствием, однако истинная проблема состоит в постижении смысла и свойств этих символов, того, как можно и как нельзя их использовать, в умении свободно обращаться с ними при описании новых задач, решений и доказательств. В конце концов, у вас выработается вкус к хорошему математическому стилю. К тому времени символы станут невидимыми; сквозь них вы будете видеть именно то, что они означают. Большое преимущество математики в том, что правила ее значительно проще, а словарь– гораздо скупее, чем в естественных языках. Вследствие этого, сталкиваясь с незнакомыми вещами, путем логической дедукции или удачного приема вы сами сможете найти решение, не обращаясь к справочникам и специалистам.

Вот почему математика, как и программирование, доставляет порой истинное наслаждение. Но достичь этого бывает нелегко. Даже математик испытывает трудности при изучении новых направлений в своей науке. Теория взаимодействующих процессов – новое направление в математике; программист, приступающий к изучению этой теории, не окажется в невыгодном положении по сравнению с математиком; в конце же он обретет явное преимущество, имея возможность найти полученным знаниям практическое применение.

Материал этой книги неоднократно апробировался как на рабочих семинарах, так и на основных курсах лекций. Сначала он был задуман в качестве семестрового спецкурса по технологии программирования, хотя большая его часть может быть изложена на последнем и даже на втором году общего курса по информатике. Перечень исходных знаний ограничивается некоторым знакомством со школьной алгеброй, понятиями теории множеств и символикой исчисления предикатов. Книга может быть использована также в качестве интенсивного недельного курса для профессиональных программистов. В таком курсе следует делать упор на примеры и определения, оставляя математические обоснования для последующей самостоятельной работы. Материал первых двух глав вполне пригоден и для более компактного курса, и даже часовой обзор книги может быть полезен при условии тщательного отбора материала, достаточного для разбора весьма поучительной истории о пяти обедающих философах.

Надо сказать, что чтение лекций и проведение семинаров по теории взаимодействующих процессов доставляет огромное удовольствие, а рассмотрение примеров дает широкие возможности для развития сценического искусства лектора. Каждый пример – это маленькая драма, которая может быть поставлена с должным учетом эмоций действующих лиц. Особую реакцию аудитории вызывают шуточные сценки на темы дедлока. Однако нужно постоянно предупреждать слушателей об опасности антропоморфизма. Математические формулы сознательно абстрагированы от всех мотивов, предпочтений и эмоциональных реакций, к которым прибегает лектор «для придания художественного правдоподобия повествованию бесцветному и малоубедительному».[1] Так что нужно учиться концентрировать внимание на сухом тексте математических формул и развивать умение восхищаться прелестью их абстрактности. В частности, это относится к некоторым рекурсивно определенным алгоритмам, чья красота в чем-то сродни захватывающей дух красоте фуг И. С. Баха.

[1] Гильберт В., Микадо, Кибернетика, 6, 1969 (пер. А. Ф. Рара),– *Прим. перев.*

Краткое содержание

В гл. 1 вводится общее понятие процесса как математической абстракции взаимодействия системы и ее окружения Показано, как с помощью известного механизма рекурсии можно описывать протяженные во времени и бесконечные процессы. Вначале идея иллюстрируется с помощью примеров и рисунков; более полное истолкование дают алгебраические законы, а также машинная реализация на функциональном языке программирования. Вторая часть главы посвящена тому, как можно представить поведение процесса в виде протокола последовательности его действий. Определены многие полезные операции над протоколами. Еще до реализации процесс может быть специфицирован путем описания свойств его протоколов. Даны правила, помогающие получить реализации процессов, сопровожденные доказательством их соответствия исходным спецификациям.

Вторая глава описывает способы построения из отдельных процессов систем, компоненты которых взаимодействуют друг, с другом и с общим внешним окружением. Введение параллелизма здесь не влечет за собой появления какого бы то ни было недетерминизма. Основным примером в этой главе служит трактовка известной притчи о пяти обедающих философах. Во второй части главы мы показываем, что процесс можно легко использовать в новых целях, изменив названия составляющих его событий. Главу завершает описание математической теории детерминированных процессов; включающей простой очерк теории неподвижной точки.

В третьей главе дается одно из простейших известных решений наболевшей проблемы недетерминизма. Показано, что недетерминизм представляет собой полезный механизм достижения абстрактности, так как он естественным образом возникает из желания не принимать во внимание не интересующие нас аспекты поведения системы. Кроме того, он позволяет сохранить своеобразную симметрию в определении операторов в математической теории. Методы доказательства для недетерминированных процессов несколько сложнее, чем для детерминированных, вследствие необходимости показывать, что в результате любого недетерминированного выбора поведение процесса будет отвечать заданной спецификации. К счастью, существуют прекрасные способы избежать недетерминизма, и они широко используются в гл. 4 и 5, Отсюда следует, что изучение гл. 3 можно отложить непосредственно до гл. 6, где знакомство с недетерминизмом становится необходимым.

В последнем разделе третьей главы дается исчерпывающее определение недетерминированного процесса. Оно предоставляет интерес для чистого математика, который хочет исследовать основы предмета или доказательно проверить справедливость алгебраических законов и других свойств процессов. Прикладные математики (включая программистов) могут воспринять эти законы как самоочевидные или оправданные их практической применимостью и в связи с этим спокойно пропустить наиболее теоретические разделы.

Только в гл. 4 мы, наконец, точно определяем, что такое взаимодействие: это особый способ взаимосвязи двух процессов, один из которых передает сообщение, а другой в то же время его принимает, Это взаимодействие синхронизовано, Если канал требует буферизации, это достигается вставкой буферного процесса между двумя процессами. Важной целью в проектировании параллельных систем является повышение скорости вычислений при решении практических задач. Это иллюстрируется на примере построения некоторых простых систолических (или итеративных) матричных алгоритмов, Простым примером является транспортер, представляющий собой последовательность процессов, в которой каждый процесс принимает сообщения только у своего предшественника и передает только своему преемнику. Транспортеры полезны при реализации протоколов однонаправленной связи, представленных в виде иерархии слоев. Наконец, важное понятие абстрактного типа данных моделируется с помощью подчиненного процесса, каждое вхождение которого взаимодействует только с тем блоком, в котором он описан,

В гл. 5 мы показываем, как совокупность обычных операторов последовательного программирования может быть взята за основу структуры взаимодействующих последовательных процессов. Опытному программисту, возможно, покажется удивительным, что эти операторы обладают такими же красивыми алгебраическими свойствами, что и операторы, знакомые ему из математики, и что способы доказательства соответствия программ своим спецификациям для последовательных и для параллельных программ во многом схожи. Даже внешние прерывания могут быть точно определены и полезно использованы, с соблюдением элегантных законов.

Гл. 6 описывает структуру и способ построения системы, в которой ограниченное число физических ресурсов, таких, как диски и печатающие устройства, разделено между большим количеством процессов

с переменной потребностью в этих ресурсах. Каждый ресурс представлен одним процессом. Как только в нем возникает потребность у некоторого процесса-пользователя, создается новый виртуальный ресурс. Виртуальный ресурс – это процесс, который ведет себя как подчиненный по отношению к процессу-пользователю, а кроме того, при необходимости взаимодействует с реальным ресурсом. Эти взаимодействия чередуются с взаимодействиями с другими одновременно активными виртуальными процессами. Таким образом, реальные и виртуальные процессы играют ту же роль, что и мониторы и конверты в языке PASCAL PLUS. Содержание главы иллюстрируется на примерах построения законченных, но очень простых операционных систем; это самые крупные примеры во всей книге.

В гл. 7 приводится ряд альтернативных подходов к понятиям параллелизма и взаимодействия; поясняются технические, исторические и личные мотивы, которые привели к появлению теории, изложенной в предыдущих главах. Здесь же я воздаю должное другим авторам и даю рекомендации по дальнейшему чтению в этой области.

Благодарности

Мне доставляет огромное удовольствие выразить признательность Робину Милнеру за его глубокую и творческую работу (Milner R. A Calculus of Communicating Systems), заложившую основы исчисления взаимодействующих систем. Его оригинальные идеи, дружеское отношение наряду с профессиональным соперничеством были для меня постоянными источниками вдохновения и поддержки в работе, завершившейся публикацией этой книги.

Последние двадцать лет я занимался проблемами программирования для параллельных вычислений и разработкой языка программирования для решения этих проблем. В течение этого периода мне было в высшей степени полезно сотрудничество со многими учеными, в том числе с П. Б. Хансеном, С. Бруксом, Д. Бастардом, Чжоу Чао Ченем, О.-Й. Далом, Э. Дейкстрой, Д. Элдером, Д. Якобом, И. Хейесом, Д. Кобишем, Д. Кеннауэйем, Т. Й. Конгом, П. Лоэром, М. Маккигом, К. Морганом, Э.-Р. Олдерогом, Р. Райнеке, Б. Роско, А. Теруелом, О. Токером, Д. Уэлшем.

И, наконец, особую благодарность я приношу О.-И. Далу, Э. Дейкстре, Лесли М. Голдшлагеру, Д. Сандерсу и остальным, кто тщательно изучил предыдущие варианты этого текста и указал на ошибки и неточности. Это также относится к слушателям Уоллонгонской летней[1] школы по теоретическому программированию (январь 1983 г.), участникам моего семинара в Школе аспирантов Академии наук Китая (апрель 1983 г.) и студентам кафедры вычислений Оксфордского университета выпусков 1979–1984 гг.

[1] Наблюдательный читатель да не забудет о существовании Южного полушария, – *Прим. ред.*

Глава 1. Процессы

1.1. ВВЕДЕНИЕ

Забудем на время о компьютерах и программировании и вместо этого подумаем о вещах, которые нас окружают. Они действуют и вступают во взаимодействие с нами и друг с другом в соответствии со своими особенностями. Вспомним о часах, телефонах, настольных играх, торговых автоматах. Чтобы описывать поведение (работу) этих объектов, надо решить, какого рода события или действия нас заинтересуют, и выбрать для каждого из них подходящее название, или *имя*. В случае простого торгового автомата существуют два вида событий;

мон – опускание монеты в щель автомата,

шок – появление шоколадки из выдающего устройства.

В случае более сложного торгового автомата различных событий может быть больше:

п1 – опускание 1 пенни,

п2 – опускание монеты в 2 пенни,

мал – появление маленького коржика или булочки,

бол – появление большого коржика или булочки,

сд1–появление 1 пенни сдачи.

Заметим, что имя каждого события обозначает целый класс событий; отдельные вхождения события внутри одного класса разделены во времени. Аналогичное различие между классом и вхождением можно проследить на примере буквы «а», многочисленные вхождения которой в текст этой книги разделены в пространстве.

Множество имен событий, выбранных для конкретного описания объекта, называется его *алфавитом.* Алфавит считается постоянным, заранее определенным свойством объекта. Участвовать в событии, выходящем за рамки его алфавита, для объекта логически невозможно – автомат по продаже шоколадок никогда не выдаст вам неожиданно игрушечный линкор. Обратное, впрочем, не обязательно. Машина, предназначенная для продажи шоколадок, может на самом деле этого не делать – возможно, потому что ее не загрузили, или она сломалась, или же просто никто не хочет шоколада. Но если уж решено, что *«шок»* входит в алфавит машины, то это имя останется там, несмотря на то, что соответствующее событие фактически никогда не происходит.

Обычно выбор алфавита влечет за собой сознательное упрощение: не рассматриваются многие действия и свойства, представляющие меньший интерес. Например, не описываются цвет, вес и форма торгового автомата, равно как и такие весьма важные события в его жизни, как пополнение запаса шоколадок или разгрузка монетоприемника. Возможно, это делается из тех соображений, что эти детали не имеют (или не должны иметь) прямого отношения к пользователям автомата.

Считается, что конкретное событие в жизни объекта происходит мгновенно, т. е. является элементарным действием, не имеющим протяженности во времени. Протяженное, т. е. требующее времени, действие следует рассматривать как пару событий, первое из которых отмечает начало действия, а второе – его завершение. Продолжительность действия определяется интервалом между наступлением его события-начала и наступлением его события-завершения; в течение всего этого времени могут происходить другие события. Два протяженных действия могут перекрываться по времени, если начало каждого из них предшествует завершению другого.

Еще одно обстоятельство, от которого мы сознательно абстрагируемся, – это точная привязка событий ко времени. Преимуществом такого подхода является упрощение построений и рассуждений, которые к тому же можно применить к физическим и вычислительным системам любой скорости и производительности. В тех же случаях, когда временная привязка событий необходима по существу, соответствующее рассмотрение может быть проведено дополнительно к доказательству логической правильности построения. Независимость от времени всегда была решающим условием успешного применения языков программирования высокого уровня.

Следствием исключения времени является то, что мы отказываемся не только от ответов, то даже и от вопросов о том, происходит ли одно событие строго одновременно с другим. Когда совместность событий существенна (например, при синхронизации), мы отмечаем это, сводя их в одно -событие, или же позволяем совместным событиям происходить в любом относительно друг друга порядке.

При выборе алфавита не обязательно делать различия между событиями, инициированными самим объектом (например, *«шок»*) или некоторым внешним по отношению объекту фактором (например, *«мон»*). Неупотребление понятия причинности ведет к значительному упрощению нашей теории и ее

приложений.

Будем отныне употреблять слово процесс для обозначения поведения объекта постольку, поскольку оно может быть описано в терминах ограниченного набора событий, выбранного в качестве его алфавита. Введем следующие соглашения:

1. Имена событий будем обозначать словами, составленными из строчных букв, например:
мон, шок, п1, п2,
а также буквами *a, b, c, d, e.*

2. Имена конкретных процессов будем обозначать словами, составленными из прописных букв, например:
ТАП – простой торговый автомат,
ТАС – сложный торговый автомат,
а буквами *P, Q, R* (в законах) будем обозначать произвольные процессы.

3. Буквы *x, y, z* используются для переменных, обозначающих события.

4. Буквы *A, B, C* используются для обозначения множества событий.

5. Буквы *X, Y* используются для переменных, обозначающих процессы.

6. Алфавит процесса P обозначается αP, например:
αТАП={мон, шок}
αТИС = {п7, п2, мал, бол, сд1}

Процесс с алфавитом *A*, такой, что в нем не происходит ни одно событие из *Л*, назовем $СТОП_A$. Этот процесс описывает поведение сломанного объекта: имея физическую способность участвовать в событиях из *A*, тот никогда ее не использует. Мы различаем объекты с различными алфавитами, даже если эти объекты ничего не делают. Так, $СТОП_{αТАП}$ мог бы выдать шоколадку, тогда как $СТОП_{αТАС}$ никогда бы не выдал шоколадку, а только коржик. Покупатель знает это даже в том случае, когда ему неизвестно, что обе машины сломаны.

В заключение определим простую систему обозначений, которая поможет при описании объектов, уже обладающих способностью к некоторым действиям.

1.1.1. Префиксы

Пусть *x* – событие, а *P* – процесс. Тогда $(x \rightarrow P)$ (читается как *"P за x"*)
описывает объект, который вначале участвует в событии *x*, а затем ведет себя в точности как *P*. Процесс $(x \rightarrow P)$ имеет по определению тот же алфавит, что и *P*; поэтому это обозначение можно использовать только при условии, что *x* принадлежит тому же алфавиту. Более формально:

$\alpha(x \rightarrow P) = \alpha P$, если $x \rightarrow \alpha P$.

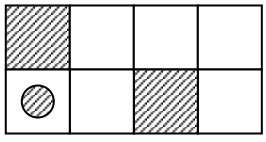
Примеры

X1. Простой торговый автомат, который поглощает монету и ломается:
$(мон \rightarrow СТОП_{αТАП})$

X2. Простой торговый автомат, который благополучно обслуживает двух покупателей и затем ломается:
$(мон \rightarrow (шок \rightarrow (мон \rightarrow (шок \rightarrow (СТОП_{αТАС}))))$
В начальном состоянии автомат позволяет опустить монету, но не позволяет вынуть шоколадку. После же приема монеты щель автомата закрывается до тех пор, пока не будет вынута шоколадка. Таким образом, эта машина не позволяет ни опустить две монеты подряд, ни получить подряд две шоколадки.

В дальнейшем мы будем опускать скобки в случае линейной последовательности событий, как в примере **X2**, условившись, что операция → ассоциативна справа.

X3. Фишка находится в левом нижнем углу поля и может двигаться только вверх или вправо на соседнюю белую клетку:



αФИШ ={вверх, вправо}
$ФИШ = (вправо \rightarrow вверх \rightarrow вправо \rightarrow вправо \rightarrow СТОП_{αФИШ})$

Заметим, что в записи с операцией → справа всегда стоит процесс, а слева – отдельное событие. Если *P* и *Q* – процессы, то запись P → Q синтаксически неверна. Правильный способ описания процесса, который ведет себя сначала как P, а затем как Q, будет дан в гл. 4. Аналогично синтаксически неверной будет запись $x \rightarrow y$, где x и y – события. Такой процесс надо было бы записать как $x \rightarrow (y \rightarrow СТОП)$. Таким образом, мы тщательно разделяем понятия «событие» и «процесс», состоящий из событий – может быть, из многих, а может быть, ни из одного.

1.1.2. Рекурсия

Префиксную запись можно использовать для полного описания поведения процесса, который рано или поздно останавливается. Но было бы чрезвычайно утомительно полностью выписывать поведение торгового автомата, рассчитанного на долгую службу; поэтому нам необходим способ более сжатого описания повторяющихся действий. Было бы желательно, чтобы этот способ не требовал знать заранее срок жизни объекта; это позволило бы нам описывать объекты, которые продолжали бы действовать и взаимодействовать с их окружением до тех пор, пока в них есть необходимость.

Рассмотрим простейший долговечный объект – часы, у которых только одна забота – тикать (их заводом мы сознательно пренебрегаем):

$\alpha ЧАСЫ = \{тик\}$

Теперь рассмотрим объект, который вначале издает единственный «*тик*», а затем ведет себя в точности как *ЧАСЫ*

$(тик \rightarrow ЧАСЫ)$

Поведение этого объекта неотличимо от поведения исходных часов. Следовательно, один и тот же процесс описывает поведение обоих объектов. Эти рассуждения позволяют сформулировать равенство

$ЧАСЫ = (тик \rightarrow ЧАСЫ)$

Его можно рассматривать как неявное задание поведения часов, точно так же как корень квадратный из двух может быть найден как положительное решение для x в уравнении

$x = x^2 + x - 2$

Уравнение для часов имеет некоторые очевидные следствия, которые получаются простой заменой членов на равные:

$ЧАСЫ = (тик \rightarrow ЧАСЫ)$	исходное уравнение
$= (тик \rightarrow (тик \rightarrow ЧАСЫ))$	подстановкой
$= (тик \rightarrow тик \rightarrow тик \rightarrow ЧАСЫ)$	аналогично

Это уравнение можно развертывать столько раз, сколько нужно, при этом возможность для дальнейшего развертывания сохраняется. Мы эффективно описали потенциально бесконечное поведение объекта *ЧАСЫ* как

$тик \rightarrow тик \rightarrow тик \ldots$

точно так же, как корень квадратный из двух можно представить в виде предела последовательности десятичных дробей 1.414… .

Такой метод «самоназывания», или рекурсивное определение процесса, будет правильно работать, только если в правой части уравнения рекурсивному вхождению имени процесса предшествует хотя бы одно событие. Например, рекурсивное «определение» $X = X$ не определяет ничего, так как решением этого уравнения может служить все что угодно. Описание процесса, начинающееся с префикса, называется предваренным. Если $F(X)$ – предваренное выражение, содержащее имя процесса X, а A – алфавит X, то мы утверждаем, что уравнение $X = F(X)$ имеет единственное решение в алфавите A Иногда удобнее обозначать это решение выражением $\mu X : A.F(X)$. Здесь X является локальным именем (связанной переменной) и может произвольно изменяться, поскольку

$\mu X : A.F(X) = \mu Y: A.F(Y)$.

Справедливость этого равенства следует из того, что решение для X уравнения $X = F(X)$ является также решением для Y уравнения $Y = F(Y)$,

В дальнейшем мы будем задавать рекурсивные определения процессов либо уравнениями, либо с помощью μ-оператора в зависимости от того, что удобнее. В записи $\mu X : A.F(X)$ мы часто будем опускать явное упоминание алфавита A, если это понятно из контекста или из содержания процесса.

Примеры

X1. Вечные часы

ЧАСЫ = μX:{тик}. (тик X)

X2. Наконец-то простой торговый автомат, полностью удовлетворяющий спрос на шоколадки:

ТАП = (мон (шок ТАП))

Как уже пояснялось, это уравнение является лишь альтернативой более формального определения

ТАП = μX:{мон, шок}. (мон (шок X))

X3. Автомат по размеру монеты в 5 пенни

αРАЗМ5А = {п5, сд2, сд1} РАЗМ5А = (п5 → сд2 → сд1 → сд2 → РАЗМ5А)

X4. Другой автомат по размену монет с тем же алфавитом}

РАЗМ5В = (п5 → сд1 → сд1 → сд1 → сд2 → РАЗМ5В)

Утверждение о том, что предваренное уравнение имеет решение, и это решение единственное, можно неформально доказать методом подстановки. Всякий раз, когда в правую часть уравнения производится подстановка на место каждого вхождения имени процесса, выражение, определяющее поведение процесса, становится длиннее, а значит, описывает больший начальный отрезок этого поведения. Таким путем можно определить любой конечный отрезок поведения процесса. А так как два объекта, ведущие себя одинаково вплоть до любого момента времени, ведут себя одинаково всегда, то они представляют собой один и тот же процесс. Те, кто считает, что такие рассуждения непонятны или неубедительны, могут принять это утверждение как аксиому; со временем его важность и уместность станут очевидными. Более строгое доказательство невозможно без точного математического определения процесса. Это будет сделано в разд. 2.8.3. Приведенное здесь описание рекурсии существенно опирается на понятие предваренности рекурсивного уравнёния. Смысл не предваренной рекурсии мы обсудим в разд. 3.8.

1.1.3. Выбор

Используя префиксы и рекурсию, можно описывать объекты, обладающие только одной возможной линией поведения. Однако поведение многих объектов зависит от окружающей их обстановки. Например, торговый автомат может иметь различные щели для 1- и 2-пенсовых монет; выбор одного из двух событий в этом случае предоставлен покупателю.

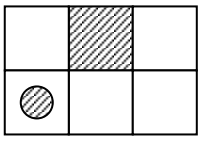
Если *x* и *y* – различные события, то *(x P | y Q)* описывает объект, который сначала участвует в одном из событий *x, y*. Последующее же поведение объекта описывается процессом P, если первым произошло событие *x*, или *Q*, если первым произошло событие *y*. Поскольку *x* и *y* – различные события, то выбор между *P* и *Q* определяется тем, какое из событий в действительности наступило первым. Требование неизменности алфавитов по-прежнему остается в силе, т. е.

α (x P | y Q) = αP

если *{x, y} ⊆ αP* и *αP = αQ*. Вертикальную черту | следует читать как "или":"*P* за *x* или *Q* за *y*".

Примеры

X1. Возможные перемещения фишки на поле описываются процессом



(вверх → СТОП | вправо → вправо → вверх → СТОП)

X2. Автомат с двумя вариантами размена монеты в 5 пенни (ср. с примерами 1.1.2 X3 и X4, где выбора нет):

РАЗМ5С = п5 →(сд1 → сд1 → сд1 → сд2 → РАЗМ5С | сд2 → сд1 → сд2 → РАЗМ5С)

Выбор осуществляется покупателем.

X3. Автомат, предлагающий на выбор шоколадку или ириску;

ТАШИ = μX . мон → (шок → X | ирис → X)

X4. Более сложный автомат, предлагающий различные товары различной стоимости и дающий сдачу:

ТАС = (п2 → (бол → ТАС | мал → сд1 → ТАС) |
п1 → (мал → ТАС | п1 → (бол → ТАС | п1 → СТОП)))

Как и многие сложные механизмы, этот автомат имеет в своей конструкции изъян. Часто бывает проще изменить инструкцию для пользователя, чем вносить исправления в готовую разработку, и поэтому мы снабдим наш автомат предупреждением:

«ВНИМАНИЕ: не опускайте три монеты подряд!»

X5. Автомат, доверяющий покупателю сначала попробовать шоколад, а заплатить потом. Нормальная последовательность событий тоже допускается:

ТАДОВЕР = μX.. (мон шок Z | шок мон X)

X6. Чтобы не возникало убытков, за право пользоваться ТАДОВЕР взимается предварительная плата

ТАП2 = (мон ТАДОВЕР)

Эта машина позволяет опустить последовательно до двух монет, после чего последовательно получить до двух шоколадок; она не выдает ни на шоколадку больше заранее оплаченного количества.

X7. Процесс копирования состоит из следующих событий:

вв.0 – считывание нуля из входного канала,
вв.1 – считывание единицы из входного канала,
выв.0 – запись нуля в выходной канал,
выв.1 – запись единицы в выходной канал.

Поведение процесса состоит из повторяющихся пар событий. На каждом такте он считывает, а затем записывает один бит.

КОПИБИТ = μX.. (вв.0 → выв.0 → X | вв.1 → выв.1 → X)

Заметим, что этот процесс предоставляет своему окружению выбор того, какое значение следует ввести, но сам решает, какое значение вывести. В этом будет состоять основное различие между считыванием и записью в нашей трактовке взаимодействия в гл. 4.

Определение выбора легко обобщить на случай более чем двух альтернатив, например:

(x → P | y → Q | ... | z → R)

Заметим, что символ выбора | не является операцией над процессами; для процессов *P* и *Q* запись *P | Q* будет синтаксически неверной. Мы вводим это правило, чтобы избежать необходимости выяснять смысл записи *(x → P) | (y → Q)*, в которой безуспешно предлагается выбор первого события. Эта проблема решается в разд. 3.3 ценой введения недетерминизма. Между тем, если *x, y, z* – различные события, запись

(x → P | y → Q | ... | z → R)

следует рассматривать как один оператор с тремя аргументами *P, Q, R*. При этом она не будет эквивалентна записи

((x → P) | (y → Q) | ... | (z → R))

которая синтаксически неверна,

В общем случае если *B* – некоторое множество событий, а *P(x)* – выражение, определяющее процесс для всех различных *x* из *B*, то запись

(x:B → P(x))

определяет процесс, который сначала предлагает на выбор любое событие *y* из *B*, а затем ведет себя как *P(y)*. Запись следует читать как «*P* от *x* за *x* из *B*». В этой конструкции *x* играет роль локальной переменной, и поэтому

(x:B →P(x)) = (y:B →P(y))

Множество *B* определяет начальное женю процесса, потому что в нем предлагается набор действий, из которого осуществляется первоначальный выбор.

Пример

X8. Процесс, который в каждый момент времени может участвовать в любом событии из своего алфавита *A*:

$\alpha ИСП_A = A$
$ИСП_A = (x: A \rightarrow ИСП_A)$

В особом случае, когда в начальном меню содержится лишь одно событие *e*,

(x:{e} →P(x)) = (e → P(e))

потому что e является единственным возможным начальным событием. В еще более специальном случае, когда начальное меню пусто, вообще ничего не происходит, и поэтому

(x:{ } → P(x)) = (y:{ } → Q(y)) = СТОП

Двуместный оператор выбора | можно определить и в более общих обозначениях:

(a → P | b → Q) = (x:B → R(x)) где *B={a, b}*, а *R(x) =* **if** *x = a* **then** *P* **else** *Q*.

Аналогичным образом можно выразить выбор из трех и более альтернатив. Итак, мы сумели сформулировать выбор, префиксы и СТОП как частные случаи записи обобщенного оператора выбора. Это нам очень пригодится в разд. 1.3, где будут сформулированы общие законы, которым подчиняются процессы, и в разд. 1.4, посвященном реализации процессов.

1.1.4. Взаимная рекурсия

Рекурсия позволяет определить единственный процесс как решение некоторого единственного уравнения. Эта техника легко обобщается на случай решения систем уравнений с более чем одним неизвестным. Для достижения желаемого результата необходимо, чтобы правые части всех уравнений были предваренными, а каждый неизвестный процесс входил ровно один раз в правую часть одного из уравнений.

Пример

X1. Автомат с газированной водой имеет рукоятку с двумя возможными положениями – *ЛИМОН* и *АПЕЛЬСИН.* Действия по установке рукоятки в соответствующее положение назовем *устлимон* и *устапельсин*, а действия автомата по наливанию напитка – *лимон* и *апельсин*. Вначале рукоятка занимает некоторое нейтральное положение, к которому затем уже не возвращается. Ниже приводятся уравнения, определяющие алфавит и поведение трех процессов:

α*АГАЗ* = αG == α*W* = {*устлимон, устапельсин, мон, лимон, апельсин*}

АГАЗ = (*устлимон* →*G* | *устапельсин* →W)

G = (*мон* →*лимон* →G | *устапельсин* →W)

W = (*мон* →*апельсин* →W | *устлимон* →G)

Неформально, после того как произошло первое событие, поведение автомата описывается одним из двух состояний G и W. В каждом из этих состояний автомат либо наливает соответствующий напиток, либо может быть переключен в другое состояние.

Использование переменных с индексами позволяет описывать бесконечные системы уравнений.

Пример

X2. Объект начинает движение с земли и может двигаться *вверх*. После этого в любой момент времени он может сдвинуться на один шаг *вверх* или *вниз*, за исключением того, что, находясь на земле, он больше не может двигаться вниз. Однако, находясь на земле, объект может совершать движение *вокруг*. Пусть n – некоторое число из натурального ряда {0, 1, 2, ,,.}. Для каждого такого n введем пронумерованное имя CT_n. с помощью которого будем описывать поведение объекта, находящегося в и шагах от земли. Начальное поведение определяется уравнением

*СТ*о = (*вверх*→CT_1 | *вокруг*→CT_0)

а остальные уравнения образуют бесконечную систему

CT_{n+1} = (*вверх*→CT_{n+2} | *вниз*→CT_n)

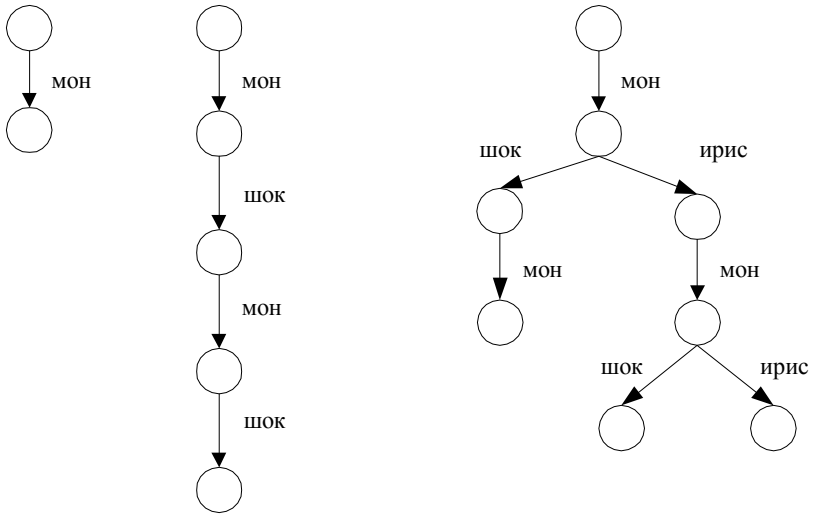
где n пробегает натуральный ряд 0,1,2, ...

Содержательность обычного индуктивного определения основана на том, что индексы, используемые в правой части каждого уравнения, меньше, чем индексы левой части. Здесь же CT_{n+1} определяется в терминах CT_{n+2}, и поэтому нашу систему можно рассматривать только как бесконечный набор взаимно рекурсивных определений, содержательность которых вытекает из того факта, что правая часть каждого уравнения является предваренной.

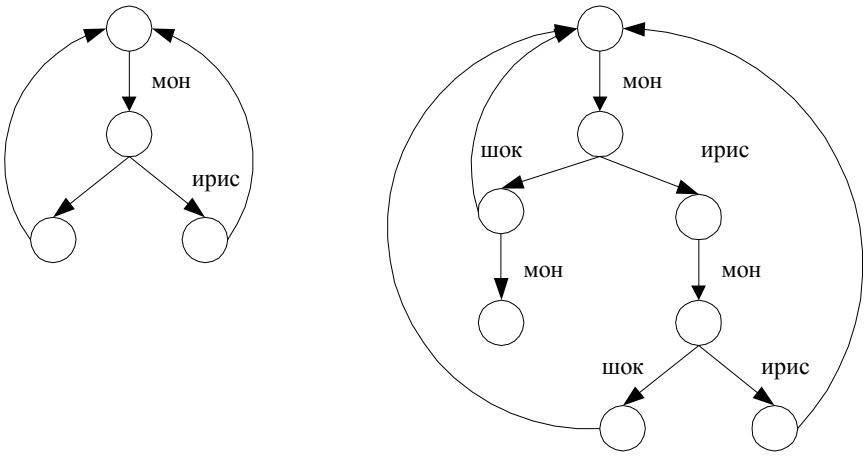
1.2. РИСУНКИ

Иногда процесс полезно представить графически в виде древовидной структуры с помощью кружков-вершин, соединенных стрелками-дугами. Следуя традиционной терминологии машин, обладающих внутренними состояниями, вершины представляют состояния процесса, а дуги – переходы между ними. Корневая вершина дерева (изображаемая обычно вверху страницы) соответствует начальному состоянию; от нее процесс движется вниз по стрелкам. Каждая дуга помечена именем события, которое соответствует данному переходу. Все дуги, ведущие из одной вершины, должны иметь различные метки.

Примеры (1.1.1 **X1, X2;** 1.1.3 **X3**)



В этих трех примерах каждая ветвь каждого дерева оканчивается состоянием *СТОП*, которое изображается вершиной без выходных дуг. Для представления процессов, обладающих неограниченным поведением, необходимо еще одно условное обозначение, а именно: непомеченная дуга, ведущая из висячей вершины назад к некоторой вершине дерева. Условимся, что когда процесс достигает вершины у основания этой дуги, он мгновенно переходит назад к вершине, на которую указывает дуга.

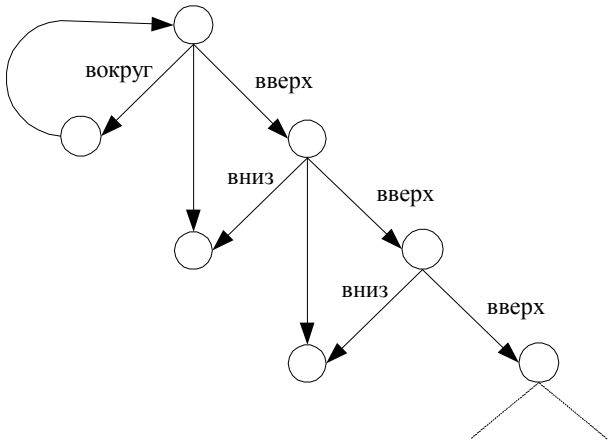


Очевидно, что два этих различных рисунка иллюстрируют в точности один и тот же процесс (1.1.3 X3 [1]). Одна из слабостей графического представления, однако, состоит в том, что доказать такое равенство графически очень трудно.

Другая проблема рисунков – это то, что с их помощью нельзя представить процесс с очень большим или бесконечным числом состояний, например CT_0:

---

[1] Строго говоря, в 1.1.3 X3 нет процесса, в точности соответствующего этому дереву. Фактически это дерево описывает три конечных процесса, каждый из которых является начальным отрезком поведения процесса *ТАШИ. – Прим. ред.*



Чтобы нарисовать эту картину целиком, никогда не хватит места. Да и на рисунок для объекта, обладающего «всего» 65536 состояниями, потребуется немало времени.

1.3. ЗАКОНЫ

Несмотря на ограниченность набора введенных нами обозначений, одно и то же поведение процесса можно описывать по-разному. Так, например, не имеет значения порядок, в котором записаны члены в операторе выбора:

$(x \to P \mid y \to Q) = (y \to Q \mid x \to P)$

С другой стороны, процесс, способный что-либо выполнять, и процесс, который не может делать ничего, – это, очевидно, не одно и то же: $(x \to P) \neq СТОП$. Чтобы правильно понимать и эффективно использовать обозначения, надо научиться узнавать, какие выражения определяют один и тот же объект, а какие – нет, точно так же как каждый, кто знаком с арифметикой, знает, что $(x + y)$ –это то же число, что и $(y + x)$, Тождественность процессов с одинаковыми алфавитами можно устанавливать с помощью алгебраических законов, очень похожих на законы арифметики.

Первый закон (**L1** ниже) касается оператора выбора. Он гласит, что два процесса, определенные с помощью оператора выбора, различны, если на первом шаге они предлагают различные альтернативы или после одинакового первого шага ведут себя по-разному. Если же множества начального выбора оказываются равными и для каждой начальной альтернативы дальнейшее поведение процессов совпадает, то очевидно, что процессы тождественны.

L1. $(x{:}A \to P(x)) = (y{:}B \to Q(y)) = (A = B \;\&\; \forall\, x \in A \,.\, P(x) = Q(x))$

Здесь и далее мы неявно предполагаем, что алфавиты процессов в обоих частях уравнения совпадают. Закон **L1** имеет ряд следствий:

L1A. $СТОП \neq (d \to P)$

Доказательство:

$ЛЧ = (x{:}\{\,\} \to P)$ по определению (конец 1.1.3)

$\neq (x{:}\{d\} \to P)$ так как $\{\,\} \neq \{d\}$

$= ПРЧ$ по определению (конец 1.1.3)

L1B. $(c \to P) \neq (d \to Q)$, если $c \neq d$,

Доказательство: $\{c\} \neq \{d\}$

L1C. $(c \to P \mid d \to Q) = (d \to Q \mid c \to P)$

Доказательство:

Определим $R(x) = P$, если $x = c$,

$= Q$, если $x = d$.

$ЛЧ = (x{:}\{c,d\} \to R(x))$ по определению

$= (x{:}\{d,c\} \to R(x))$ так как $\{c, d\} = \{d, c\}$

$= ПРЧ$ по определению

L1D. $(c \to P) = (c \to Q) \equiv P = Q$

Доказательство: $\{c\} = \{c\}$

С помощью этих законов можно доказывать простые теоремы.

Примеры

X1. (*мон → шок → мон → шок → СТОП) ≠ (мон → СТОП)*

Доказательство: следует из **L1D** и **L1A**.

X2. *μX.(мон → (шок → X | ирис → X)) = μX. (мон → (ирис → X | мон →X))*

Доказательство: следует из **L1C**.

Для доказательства более общих теорем о рекурсивно определенных процессах необходимо ввести закон, гласящий, что всякое должным образом предваренное рекурсивное урав-. нение имеет единственное решение.

L2. Если *F(X)* – предваренное выражение, то *(Y = F(Y)) ≡ (Y = μX.F(X))*.

Как прямое, но важное следствие получаем, что *μX.F(X)* в действительности является решением соответствующего уравнения:

L2A. μX. F(X) = F(μX. f (X))

Пример

X3. Пусть *TA1 = (мон → TA2)*, а *TA2 = (шок →TA1)*. Требуется доказать, что *TA1 = ТАП.*

Доказательство:

TA1 = (мон → TA2) по определению *TA1*

= (мон → (шок → TA2)) по определению *TA2*

Таким образом, *TA1* является решением того же рекурсивного уравнения, что и *ТАП*. Так как это уравнение предваренное, оно имеет единственное решение. Значит, *TA1* и *ТАП* – это просто разные имена одного и того же процесса. Эта теорема может казаться настолько очевидной, что ее формальное доказательство ничего не прибавит к ее убедительности. Единственная цель этого доказательства – показать на примере, что наши законы достаточно мощны для установления фактов подобного рода. В частности, при доказательстве очевидных фактов с помощью менее очевидных законов важно проследить справедливость каждого шага доказательства, чтобы избежать порочного круга.

Закон **L2** можно распространить на случай взаимной рекурсии. Систему рекурсивных уравнений можно записать в общем виде, используя индексы:

$X_i = F(i, X)$ *для всех i из S*,

где *S* – множество индексов, взаимно однозначно соответствующих множеству уравнений, X – массив процессов с индексами из *S*, а *F(i, X)* – предваренное выражение.

Закон **L3** гласит, что при соблюдении этих условий существует единственный массив *X*, элементы которого удовлетворяют всем уравнениям.

L3. При соблюдении описанных выше условий

если $(\forall i: S.(X_i = F(i, X)\ Y_i = F(i, Y)))$, то *X = Y*.

1.4. РЕАЛИЗАЦИЯ ПРОЦЕССОВ

Любой процесс *P*, записанный с помощью уже введенных обозначений, может быть представлен в виде

(x:B → F(x))

где *F* – функция, ставящая в соответствие множеству символов множество процессов, множество *B* может быть пустым (в случае *СТОП*), может содержать только один элемент (в случае префикса) или – более одного элемента (в случае выбора). Если процесс задан рекурсивным уравнением, то мы требуем, чтобы рекурсия была предваренной и, значит, решение уравнения могло быть записано как

μX.(x:B→F(x, X))

Используя закон **L2A**, можно привести эту запись к требуемому виду,

(x: B F(x, μX.(x: B → F(x, X))))

Таким образом, каждый процесс можно рассматривать как функцию *f* с областью определения *B*, выделяющей множество событий, которые могут служить начальными событиями процесса. Если первым произошло событие *x* из *B*, то *F(x)* определяет дальнейшее поведение процесса.

Такой подход позволяет представить любой процесс как функцию в некотором подходящем

функциональном языке программирования, например в ЛИСПе. Каждое событие из алфавита процесса представлено атомом, например *"МОН, "ИРИС*. Процесс – это функция которая может использовать эти символы в качестве аргументов. При этом если символ не может быть начальным событием процесса, то результатом функции будет специальный символ *"BLEEP*, который используется только для этой цели. Например, так как в процессе СТОП не происходит никаких событий, то *"BLEEP* – единственный возможный результат соответствующей функции, и можно определить

СТОП = λx. "BLEEP

Если же фактический аргумент является событием, возможным для процесса, результатом функции будет другая функция, определяющая последующее поведение процесса. Так, процесс (мон СТОП) можно представить функцией

λx. **if** x = "МОН **then** СТОП **else** "BLEEP

В последнем примере мы используем возможность ЛИСПа получать в качестве результата функции снова функцию (в нашем случае СТОП). ЛИСП также позволяет задавать функцию в качестве аргумента – средство, которое мы используем для представления общей операций префиксации **,** *(c → P):*

префикс(c, P) = λx. **if** *x = c* **then** *P*
else *"BLEEP*

Для представления общего двуместного выбора *(c → P | d →Q)* нам потребуется функция с четырьмя параметрами:

выбор2(c, P, d, Q) = λx. **if** *x = c* **then** P
else if *x = d* **then** *Q*
else *"BLEEP*

Для представления рекурсивно определенных процессов можно использовать функцию ЛИСПа *LABEL.* Например, простой торговый автомат *(μX.мон → шок → X)* определяется функцией

LABEL X.префикс("МОН, префикс("ШОК, X))

С помощью *LABEL* можно описать и взаимную рекурсию. Например, *СТ* из примера 1.1.4 **X2** можно рассматривать как функцию, ставящую в соответствие множеству натуральных .чисел множество процессов (которые сами являются функциями – но нас это не должно тревожить). Поэтому можно определить СТ как

СТ = LABEL X. λn.
if *n ≠ 0* **then** *выбор2("ВОКРУГ, X(0), "ВВЕРХ, X(1))*
else *выбор2("ВВЕРХ, X(n+1), "ВВЕРХ, X(n –1))*

Объект, стартующий с земли, описывается процессом

СТ(0).

Если *P* – функция, описывающая процесс, а *x* – список символов, составляющих его алфавит, то ЛИСП–функция *меню(A, P)* выдает список всех символов из *A*, которые могут обозначать первое событие в жизни *P*:

меню(Л, P) = **if** *F = NIL* **then** *NIL*
else if *P(car(A)) = "BLEEP* **then** *меню (cdr(A), P)*
else *cons(car(A), меню(cdr(A), P))*

Если *x* принадлежит *меню (A, P),* то значение функции *P(x)* отлично от *"BLEEP* и, значит, определяет дальнейшее поведение процесса *P* после выполнения события *x*. Поэтому если у принадлежит *меню(A, P(x)),* то *P(x)(y)* описывает последующее поведение *P* после того, как произошли и *x* и *y*. Это наблюдение подсказывает удобный способ исследования поведения процесса. Напишите программу, которая выводит на экран содержимое *меню(A, P),* а затем воспринимает символ, вводимый с клавиатуры. Если символ отсутствует в меню, программа реагирует на это слышимым сигналом *«bleep»*, а символ в дальнейшем игнорирует. В противном случае символ воспринимается, а процесс повторяется, но *P* в нем заменяется на результат применения *P* к указанному символу. Процесс завершается появлением на экране символа "*END*. Таким образом, если *A* – последовательность вводимых с клавиатуры символов, то последовательность выводимых на экран ответов описывается функцией

взаимодействие(A, P, k) = cons(меню(A, P)
if *car(k) = "END* **then** *NIL*
else if *P(car(k)) = "BLEEP* **then**
cons("BLEEP, взаимодействие(A, P ,cdr(k)))
else *взаимодействие(A, P(car(k),cdr(k))*

Обозначения, которые мы использовали для описания ЛИСП–функций, очень нестрогие и нуждаются в трансляции в специальные общепринятые выражения какой-либо конкретной реализации ЛИСПа. В ЛИСПките, например, функция *префикс* запишется как

(префикс

```
lambda
    (a p)
    (lambda (x) (if (eq x a) p (quote BLEEP))))
```

К счастью, мы будем пользоваться только очень небольшим подмножеством чисто функционального ЛИСПа, что сведет к минимуму трудности при трансляции и исполнении наших процессов на множестве диалектов языка на различных машинах.

Если в вашем распоряжении несколько версий ЛИСПа, выберите ту, где должным образом представлено статическое связывание переменных. Наличие в ЛИСПе отложенных вычислений также предпочтительно, поскольку оно делает возможным прямое кодирование рекурсивных уравнений без помощи механизма *LABEL*; так, например,

ТАП = префикс("МОН, префикс("ШОК, ТАП))

Если ввод и вывод реализованы с помощью отложенных вычислений, функцию *взаимодействие* можно вызвать, подставив в качестве третьего параметра символы, вводимые с клавиатуры, и получить на экране меню для процесса *P,* Так, отбирая и вводя символы следующих друг за другом меню, пользователь имеет возможность интерактивно исследовать поведение' процесса Р. В других версиях ЛИСПа для достижения этой цели функцию; *взаимодействие* необходимо переписать, явным образом используя ввод и вывод. После того как это сделано, можно наблюдать машинное исполнение процесса, представленного ЛИСП-функцией. В этом смысле функцию ЛИСПа можно рассматривать как *реализацию* соответствующего процесса. Более того, такую ЛИСП-функцию, как *префикс*, обрабатывающую представленные таким образом процессы, можно рассматривать как реализацию соответствующей операции над процессами.

1.5. ПРОТОКОЛЫ

Протоколом поведения процесса называется конечная последовательность символов, фиксирующая события, в которых процесс участвовал до некоторого момента времени. Представьте себе наблюдателя с блокнотом, который следит за процессом и записывает имя каждого происходящего события. У нас есть все основания не учитывать возможности одновременного наступления двух событий; даже если это случится, наблюдателю все равно придется записать их одно за другим, и порядок, в котором записаны события, не будет иметь значения.

Мы будем обозначать протокол последовательностью символов, разделенной запятыми и заключенной в угловые скобки:

<x, y> состоит из двух событий – *x* и следующего за ним *y*.

<x> состоит из единственного события *x*.

<> пустая последовательность, не содержащая событий.

Примеры

X1. Протокол простого торгового автомата *ТАП (1.1.2 **X2)*** в момент завершения обслуживания первых двух покупателей:

<мон, шок, мон, шок>

X2. Протокол того же автомата перед тем, как второй покупатель вынул свою шоколадку:

<мон, шок, мон>

Понятие завершенной сделки недоступно ни процессу, ни его наблюдателю. Голод ожидающего клиента и готовность автомата его удовлетворить не входит в алфавит процесса и не могут быть зафиксированы и занесены в протокол.

X3. Перед началом работы процесса блокнот наблюдателя пуст. Это изображается пустым протоколом <> – самым коротким из всех возможных протоколов любого процесса.

X4. Сложный торговый автомат *ТАС(1.1.3. **X4)*** имеет семь различных протоколов длины два и менее:

<>

<n2> *<n1>*

<n2, бол > <n2, мал > *<n1, n1> <n1, мал >*

Из четырех протоколов длины два для данного устройства фактически может произойти только один. Его выбор по своему желанию осуществляет первый покупатель.

X5. Протокол *TAC,* если первый покупатель нарушил инструкцию: *<n1, n1, n1>*. Сам протокол не фиксирует поломку автомата. На нее указывает лишь тот факт, что среди всех возможных протоколов этой машины не существует такого, который являлся бы продолжением данного, т. е. не существует такого *x*, что *<n1, n1, n1,x>* является возможным протоколом *TAC*. Покупатель может волноваться, негодовать; наблюдатель– нетерпеливо ожидать, держа карандаш наготове, но отныне не произойдет ни одно событие, и ни один символ не будет записан в блокнот. Окончательное право распорядиться судьбой покупателя и машины не входит в выбранный нами алфавит.

1.6. ОПЕРАЦИИ НАД ПРОТОКОЛАМИ

Протоколам принадлежит основная роль в фиксировании, описании и понимании поведения процессов. В этом разделе мы исследуем некоторые общие свойства протоколов и операций над ними. Введем следующие соглашения:

s, t, u обозначают протоколы,

S, T, U обозначают множества протоколов,

f, g, h обозначают функции.

1.6.1. Конкатенация

Наиболее важной операцией над протоколами является конкатенация, которая строит новый протокол из пары операндов *s* и *t*, просто соединяя их в указанном порядке; результат будем обозначать *s^t*. Например,

(*<мон, шок>^<мон, ирис> = <мон, шок, мон, ирис>*

<n1>^<n1> = <n1, n1>

<n1, n1>^<> = <n1, n1>

Самые важные свойства конкатенации – это ее ассоциативность и то, что пустой протокол <> служит для нее единицей:

L1. *s^ <> = <>^s*

L2. *s^(t ^ u> = (s^t)^u*

Следующие законы и очевидны, и полезны:

L3. *s^t = t^u ≡ t = u*

L4. *s^t = u^t ≡ s = u*

L5. *s^t = <> ≡ s = <> & t = <>*

Пусть *f* – функция, отображающая протоколы в протоколы. Мы говорим, что функция *строгая,* если она отображает пустой протокол в пустой протокол:

f(<>) = <>

Будем говорить, что функция дистрибутивна, если

f(s^t) = f(s)^f(t)

Все дистрибутивные функции являются строгими.

Если *n* – натуральное число, то t^n будет обозначать конкатенацию n копий протокола t. Ее легко определить индукцией по n:

L6. $t^0 = <>$

L7. $t^{n+1} = t^t^n$

Это определение дает нам два полезных закона; и еще два можно доказать как их следствия:

L8. $t^{n+1} = t^n^t$

L9. $(s^t)^{n+1} = s^(t^s)^n^t$

1.6.2. Сужение

Выражение *(t ↾ A)* обозначает протокол t, суженный на множество символов A; он строится из t отбрасыванием всех символов, не принадлежащих A. Например,

<вокруг, вверх, вниз, вокруг > ↾ {вверх, вниз} = *<вверх, вниз>*

Сужение дистрибутивно и поэтому строго.

L1. *<> ↾ A = <>*

L2. *(s^t) ↾ A = (s ↾ A)^(t ↾ A)*

Эффект сужения на одноэлементных последовательностях очевиден:

L3. *<x> ↾ A = <x> если x ∈ A*

L4. *<y> ↾ A = <> если y ∉ A.*

Дистрибутивная функция однозначно определяется своими результатами на одноэлементных последовательностях, поскольку ее значения на последовательностях большей длины могут быть вычислены конкатенацией ее результатов на отдельных элементах последовательности. Например, если $x \neq y$, то

<x, y, x> = (<x>^<y>^<x>) ↾ {x} = (<x> ↾ {x})^(<y> ↾ {x})^(<x> ↾ {x}) по **L2**

= <x>^<>^<x> по **L3, L4**

= <x, x>

Приведенные ниже законы раскрывают взаимосвязь сужения и операций над множествами. От протокола, суженного на пустое множество, не остается ничего, а последовательное сужение на два множества равнозначно одному сужению на пересечение этих множеств. Эти законы можно доказать индукцией по длине s.

L5. *s ↾{} = <>*

L6. *(s ↾A) ↾ B = s ↾ (A ∪ B)* ;

1.6.3. Голова и хвост

Если *s* – непустая последовательность, обозначим ее первый элемент s_0, а результат, полученный после его удаления,– *s'*. Например:

$<x, y, x>_0 = x$

<x, y, x>' = <y, x>

Обе эти операции не определены для пустой последовательности.

L1. $(<x>\hat{}s)_0 = x$

L2. *(<x>^s)' = s*

L3. *s = (<s0>^s')* если $s \neq <>$.

Следующий закон предоставляет удобный метод доказательства равенства или неравенства двух протоколов:

L4. $s = t \equiv (s = t = <> \vee (s_0 = t_0 \,\&\, s' = t'))$

1.6.4. Звездочка

Множество *A**–это набор всех конечных протоколов (включая <>), составленных из элементов множества *A*. После сужения на *A* такие протоколы остаются неизменными, Отсюда следует простое определение:

A = {s | s ↾ A = s}*

Приведенные ниже законы являются следствиями этого определения:

L1. *<> ∈ A**

L2. *<x> ∈ A* ≡ x ∈ A*

L3. *(s^t) ∈ A* ≡ s ∈ A* & t ∈ A**

Они обладают достаточной мощностью, чтобы определить, принадлежит ли протокол множеству A*. Например, если

x ∈ A, а *y ∉ A*, то

<x, y> ∈ A ≡ (<x>^<y>) ∈ A**

≡ (<x> ∈ A) & (<y> ∈ A*)* по **L3**

≡ истина & ложь по **L2**

$\equiv$ *ложь*

Еще один полезный закон может служить рекурсивным определением A^*:

L4. $A^* = \{t \mid t = <> \vee (t0 \in A \;\&\; t' \in A^*)\}$

1.6.5. Порядок

Если s – копия некоторого начального отрезка t, то можно найти такое продолжение u последовательности s, что $s\hat{}u = t$ Поэтому мы определим отношение порядка

$s \leq t = (\exists u.s\hat{} = t)$

и будем говорить, что s является префиксом t. Например:

$<x, y> \leq <x, y, x, w>$

$<x, y> \leq <z, x, y> \equiv x = z$

Отношение $\leq$ является частичным упорядочением и имеет своим наименьшим элементом $<>$. Об этом говорят законы **L1–L4**:

L1. $<> \leq s$ наименьший элемент

L2. $s \leq s$ рефлексивность

L3. $s \leq t \;\&\; t\ s \Rightarrow s = t$ антисимметричность

L4. $s \leq t \;\&\; t \leq u \Rightarrow s \leq u$ транзитивность

Следующий закон вместе с L1 позволяет определить, является ли справедливым отношение $s \leq t$:

L5. $(<x>\hat{}s) \leq t \equiv t \neq <> \;\&\; x = t_0 \;\&\; s \leq t'$

Множество префиксов данной последовательности вполне упорядочено:

L6. $s \leq u \;\&\; t \leq u \Rightarrow s \leq t \vee t \leq s$

Если s является подпоследовательностью t (не обязательно начальной), мы говорим, что s находится **в** t; это можно определить так:

s **в** $t = (\exists u, v.t = u\hat{}s\hat{}v)$

Это отношение также задает частичный порядок в том смысле, что оно удовлетворяет законам **L1–L4**, Кроме того, для него справедливо следующее утверждение:

L7. $(<x>\hat{}s)$ **в** $t = t \equiv t \neq <> \;\&\; ((t_0 = x \;\&\; s \leq t) \vee (<x>\hat{}s)$ **в** $t')$

Будем говорить, что функция f из множества протоколов во множество протоколов *монотонна*, если она сохраняет отношение порядка $\leq$ т. е.

$f(s) \leq f(t)$ всякий раз, когда $s \leq t$

Все дистрибутивные функции монотонны, например;

L8. $s \leq t \Rightarrow (s \restriction A) \leq (t \restriction A)$

Двуместная функция может быть монотонной по каждому из аргументов. Например, конкатенация монотонна по второму (но не по первому) аргументу:

L9. $t \leq u \Rightarrow (s\hat{}t) \leq (s\hat{}u)$

Функция, монотонная по всем аргументам, называется просто монотонной.

1.6.6. Длина

Длину протокола t будем обозначать $\#t$, Например, $\#<x, y, x> = 3$ Следующие законы определяют операцию #;

L1. $\#<> = 0$

L2. $\#<x> = 1$

L3. $\#(s\hat{}t) = (\#s) + (\#t)$

Число вхождений в t символов из A вычисляется выражением $\#(t \restriction A)$.

L4. $\#(t \restriction (A \cup B)) = \#(t \restriction A) + \#(t \restriction B) - \#(t \restriction (A \cap B))$

L5. $s \leq t \Rightarrow \#s \leq \#t$

L6. $\#(t^n) = n(\#t)$

Число вхождений символа x в протокол s определяется как

$s \downarrow x = \#(s \restriction \{x\})$

1.7. РЕАЛИЗАЦИЯ ПРОТОКОЛОВ

Для машинного представления протоколов и реализации операций над ними нам потребуется язык высокого уровня с развитыми средствами работы со списками, К счастью, ЛИСП как нельзя лучше подходит для этой цели. Протоколы в нем очевидным образом представляются списками атомов, соответствующих его событиям:

<> = NIL

<мон>= cons("МОН, NIL)

<мон, шок> = "(МОН, ШОК)

что означает *cons("МОН, cons("ШОК, NIL))*.

Операции над протоколами легко реализовать как функции над списками. Так, например, голова и хвост непустого списка задаются примитивами car и cdr:

$t_0 = car(t)$

t' = cdr(t)

<x>^s = cons(x,s)

Конкатенация в общем виде реализуется с помощью известной функции *append,* которая задается рекурсивно:

s^t = append(s, t)

где *append(s, t)* = **if** *s = NIL* **then** *t*

else *cons(car(s),append(cdr(s, t))*

Корректность этого определения вытекает из законов

<>^t = t

$s^t = \langle s_0 \rangle ^ (s' ^ t)$ когда $s \neq \langle\rangle$.

Завершение ЛИСП–функции *append* гарантируется тем, что при каждом рекурсивном вызове список, подставляемый в качестве первого аргумента, короче, чем на предыдущем уровне рекурсии. Подобное рассуждение позволяет установить корректность и других, приведенных ниже, реализаций. Для реализации сужения представим конечное множество списком его элементов. Проверка ($x \in B$) выполняется вызовом функции

принадлежит(x, B) = **if** *B = NIL* **then** *ложь*

else if *x = car(B)* **then** *истина*

else принадлежит*(x, cdr(B))*

после чего сужение *(s !B)* может быть реализовано функцией

сужение (s, B) = **if** *s = NIL* **then** *NIL*

else if *принадлежит(car(s), B)* **then** *cons(car(s), сужение(cdr(s), B))*

else *сужение(cdr(s),B)*

Проверка ($s \leq t$) реализуется функцией, дающей ответ *истина* или *ложь*; реализация основана на законах 1.6.5 **L1** и **L5:**

префикс_ли(s, t) = **if** *s = NIL* **then** *истина*

else if *t = NIL* **then** *ложь*

else *car(s) = car(t) & префикс_ли (cdr(s), cdr(t))*

1.8. ПРОТОКОЛЫ ПРОЦЕССА

В разд. 1.5 мы ввели понятие протокола как последовательной записи поведения процесса вплоть до некоторого момента времени. До начала процесса неизвестно, какой именно из возможных протоколов будут записан: его выбор зависит от внешних по отношению к процессу факторов. Однако полный набор возможных протоколов процесса *P может* быть известен заранее, и мы введем функцию *протоколы(P)* для обозначения этого множества.

Примеры

X1. Единственным возможным протоколом процесса *СТОП* является <>. Блокнот наблюдателя этого процесса всегда остаётся чистым:

Протоколы(СТОП)= {< >}

X2. Автомат, поглощающий монету, прежде чем сломаться, имеет всего два протокола:

Протоколы(мон→СТОП)={< >,<мон>}

X3. Часы, которые только тикают:

Протоколы(μX.тик→X)={<>,<тик>,<тик, тик>,...}

*={тик}**

Здесь, как и во всех наиболее содержательных процессах, множество протоколов бесконечно, хотя, разумеется, каждый отдельный протокол конечен.

X4. Простой торговый автомат:

Протоколы(μX.мон→шок→X)={s | ∃n.s≤<мон, шок>^{n}}

1.8.1. Законы

В этом разделе мы покажем, как вычислить множество протоколов любого процесса, определенного с помощью уже введенных обозначений. Как отмечалось выше, процесс *СТОП* имеет только один протокол:

L1. *протоколы (СТОП)={t | t = < >} = {< >}*

Протокол процесса *(c→P)* может быть пустым, поскольку <> является протоколом поведения любого процесса до момента наступления его первого события. Каждый непустой протокол этого процесса начинается с *c*, а его хвост должен быть возможным протоколом *P*:

L2. *протоколы(c→P)={t | t = < > ∨ (t_0 = c&t' ∈ протоколы(P))}*

={< > ∪ {<c>^t | t ∈ протоколы(P)}

Протокол процесса, содержащего выбор начального события, должен быть протоколом одной из альтернатив:

L3. *протоколы(c→P | d→Q)={t | t=< > ∨* (t_0 = c & t' ∈ *протоколы(P))* ∨ (t_0 = c&t' ∈ *протоколы(Q))}*

Эти три закона можно объединить в один общий закон, которому подчиняется конструкция выбора:

L4. *протоколы(x:B→P(x))={t | t = < > ∨ (t_0 ∈ B & t' ∈ протоколы(P(t_0))}*

Несколько сложнее найти множество протоколов рекурсивно определенного процесса. Рекурсивно определенный процесс является решением уравнения *X = F(X)*. Сначала определим по индукции итерацию функции F:

$$F^0(X) = X$$
$$F^{n+1}(x) = F(F^n(X))$$
$$= F^n(F(X))$$
$$= \underbrace{F(\ldots(F(F(X)))\ldots)}_{n\ раз}$$

Затем, при условии, что функция *F* – предваренная, можно определить

L5. $протоколы(\mu X{:}A.F(X)) = \bigcup_{n\geq 0} протоколы(F^n(СТОП_A))$

Примеры

X1. Вспомним, что в примере 1.1.3 **X8** мы определили $ИСП_A$ как *μX:A.F(X),* где *F(X)=(x:A→X).* Мы хотим доказать, что *протоколы*($ИСП_A$)=*A**.

Доказательство: $A^* = \bigcup_{n\geq 0} \{s \mid s \in A^* \ \&\ \#s \leq n\}$

Поэтому, согласно **L5**, достаточно для всех *n* доказать, что

протоколы($Fn(СТОП_A)$ = *(s | s ∈* A* *&* $\#s \leq n\}$

Это делается индукцией по n:

(1) Для n = 0

протоколы(СТОП$_A$) = {<>} = {s | s ∈ A & #s ≤ 0}

(2) *протоколы* (F^{n+1}(*СТОП$_A$*)) = *протоколы* (x: A → F^n(*СТОП$_A$*)) = по определению *F,Fn^{+1}*

= {t | t = <> ∨ (t_0 ∈ A & t' ∈ протоколы(F^n(СТОП$_A$)))} = **L4**

= {t | t = <> ∨ (t_0 ∈ A & (t' ∈ A & #t' ≤ n))} =* предположение индукции

= {t | (t = <> ∨ (t_0 ∈ A & t' ∈ A)) & #t ≤ n+1} =* свойство#

= {t | t ∈ A & #t ≤ n+1}* 1.6.4 **L4**

X2. Докажем 1.8 **X4**, т. е.

$$протоколы(ТАП) = \bigcup_{n\geq 0} \{s \mid s \ <мон, шок>^n\}$$

Доказательство:

Согласно предположению индукции *протоколы(Fn(ТАП)) = {t | t ≤ <мон, шок>^{n}}*,

где F(X) = (мон→ шок → X)

(1) протоколы(СТОП) = {<>} = {s | s <мон, шок>^{n} } 1.6.1 **L6**

(2) *протоколы(мон→ шок → Fn(СТОП)) =*

= {<>,<мон>} ∪ { <мон, шок>^t | t ∈ протоколы(Fn(СТОП)} = **L2**

= {<>, <мон>} ∪ {<мон, шок>^ t | t ≤ <мон, шок>^{n}} = предположение индукции

= {s | s = <>∨ s = <мон> ∨ ∃ t. s = <мон, шок>^ t & t ≤ <мон, шок>^{n}} =

= {s | s ≤ <мон, шок>$^{n+1}$}

Заключительная часть следует из **L5**.

Как упоминалось в разд. 1.5, протокол – это последовательность символов, фиксирующих события, в которых процесс *P* участвовал до некоторого момента времени. Отсюда следует, что <> является протоколом любого процесса до момента наступления его первого события. Кроме того, если *(s^t)* – протокол процесса до некоторого момента, то *s* должен быть протоколом того же процесса до некоторого более раннего момента времени. Наконец, каждое происходящее событие должно содержаться в алфавите процесса. Три этих факта находят свое формальное выражение в законах;

L6. <> ∈ *протоколы (P)*

L7. *s^t ∈ протоколы (P) ⇒ s ∈ протоколы(P)*

L8. *протоколы(P) ⊆ (αP)**

Существует тесная взаимосвязь между протоколами процесса и изображением его поведения в виде дерева. Для каждой вершины дерева протоколом процесса до момента достижения этой вершины будет просто последовательность меток,

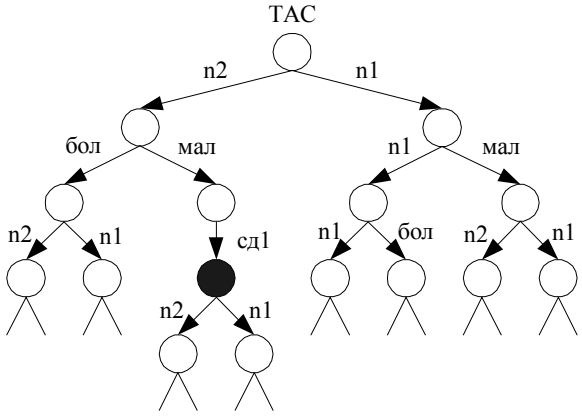


Рис 1.1

встреченных на пути из корня дерева в эту вершину. Например, в дереве для *ТАС*, приведенном на рис. 1.1, пути из корня в черную вершину соответствует протокол *<n2, мон, сд1>*.

Очевидно, что все начальные отрезки путей в дереве сами являются путями в этом дереве; более строго это было сформулировано в законе **L7**, Пустой протокол определяет путь нулевой длины из корня в самого себя, что подтверждает закон **L6**. Протоколы процесса представляют собой множество всех путей, ведущих из корня в различные вершины дерева. И наоборот, так как все дуги, ведущие из каждой вершины, помечены различными событиями, каждый протокол процесса однозначно определяет

путь из корня дерева в некоторую вершину. Таким образом, любое множество протоколов, удовлетворяющих законам **L6** и **L7**, является удобным математическим представлением для дерева без повторяющихся меток на дугах, исходящих из одной вершины.

1.8.2. Реализация

Предположим, что процесс реализован ЛИСП-функцией P, и пусть s – некоторый протокол. Определить, является ли s возможным протоколом P, можно с помощью функции

протокол_ли(s,P) = **if** *s = NIL* **then** *истина*
else if $P(s_0)$ = *"BLEEP* **then** *ложь*
else *протокол_ли(s', $P(s_0)$)*

Так как протокол *s* конечен, то содержащаяся в этом определении рекурсия завершится после исследования лишь конечного по длине начального отрезка поведения процесса *P*. Вследствие того, что мы избегаем бесконечного исследования поведения процесса, можно с уверенностью определить процесс как бесконечный объект, т. е. функцию, результат которой есть функция, результат которой есть функция, результат которой...

1.8.3. После

Если s ∈ *протоколы(P)*,то

P / s (P после *s)* –

это процесс, ведущий себя так, как ведет себя *P* с момента завершения всех действий, записанных в протоколе *s*. Если *s* не является протоколом *P*, то *(P / s)* не определено.

Примеры

X1. *(ТАП / <мон>) = (шок ТАП)*

X2. *(ТАП / <мон, шок>) = ТАП*

X3. *(ТАС / $<n1>^3$) = СТОП*

X4. Во избежание убытка от установки *ТАДОВЕР* (1.1.3 **X5**, **X6**) первую шоколадку владелец автомата решает съесть сам:

(*ТАДОВЕР / <шок>) = ТАП2*

В древовидном представлении *P* (рис. 1.1) конструкции P / s) соответствует все поддерево с корнем в конечной вершине пути, помеченного символами из s. Таким образом, в нашем примере поддерево, исходящее из черной вершины, описывается как *ТАС / (n2, мал, сд1)*

Следующие законы раскрывают значение операции /. Ничего не сделав, процесс остается неизменным:

L1. *P / <> = P*

Поведение процесса *P* после завершения событий из *s^t* совпадает с поведением *(P / s)* после завершения событий из *t*:

L2. *P / (s^t) = (P / s) / t;*

Если процесс участвовал в событии *c*, его дальнейшее поведение определяется именно этим начальным выбором:

L3. *(x : B → P(x)) / <c> = P(c)* при условии, что *c ∈ B*

В качестве следствия мы получаем, что оператор */<c>* является обратным по отношению к оператору префиксации

c →:

L3A. *(c P) / <c> = P*

Протоколы *(P / s)* определяются как

L4. *протоколы(P / s) = (t | s^t ∈ протоколы(P)}* при условии, что *s ∈ протоколы(P)*

Для доказательства того, что процесс P работает бесконечно, достаточно доказать, что

P / s = СТОП для всех *s ∈ протоколы (P)*

Другим желательным свойством процесса является цикличность; по определению процесс P – циклический, если при любых обстоятельствах он может вернуться в начальное состояние, т. е.

∀s : протоколы(P). ∃t.(P / (s^t) = P)

СТОП – это тривиальный циклический процесс; если же любой другой процесс цикличен, он также обладает полезным свойством никогда не останавливаться.

Примеры

X1. Следующие процессы являются циклическими (1.1.3 **X8**, 1.1.2 **X2**, 1.1.3 **X3**, 1.1.4 **X2**):

$ИСП_A$, ТАП, (шок → ТАП), ТАШИ, $СТ_7$

X2. Следующие процессы не являются циклическими, потому что они не могут вернуться в начальное состояние (1.1.2 **X2**, 1.1.3 **X3**, 1.1.3 **X2**):

(мон → ТАП), (шок → ТАШИ), (вокруг → $СТ_7$)

Так, например, в начальном состоянии *(шок → ТАШИ)* можно получить только шоколадку, тогда как в дальнейшем наряду с шоколадкой всегда можно выбрать ириску, следовательно, никакое из этих последующих состояний не эквивалентно начальному.

Предостережение. Использование / в рекурсивно определенных процессах, к сожалению, лишает их свойства предваренности, и поэтому возникает опасность получения неоднозначных решений. Например, уравнение $X = (a \rightarrow (X / \langle a \rangle))$ не является предваренным и имеет решением любом процесс вида $a \rightarrow P$ для любого P,

Доказательство:

$(a \rightarrow ((a \rightarrow) / \langle a \rangle)) = (a \rightarrow P)$ согласно **L3A**

По этой причине мы никогда не будем использовать операцию «/» в рекурсивных определениях процесса.

1.9. ДАЛЬНЕЙШИЕ ОПЕРАЦИИ НАД ПРОТОКОЛАМИ

Этот раздел посвящен некоторым дальнейшим операциям над протоколами. На данной стадии его можно пропустить, потому что в следующих главах, где используются эти операции, будут даны соответствующие ссылки.

1.9.1. Замена символа

Пусть f – функция, отображающая символы из множества A в множество B. С помощью f можно построить новую функцию f^*, отображающую последовательность символов из A^* в последовательность символов из B^*, путем применения f к каждому элементу последовательности. Например, если *удвоить* – функция, удваивающая свой целый аргумент, то

$удвоить^*(\langle 1, 5, 3, 1 \rangle) = \langle 2, 10, 6, 2 \rangle$

Очевидно, что функция со звездочкой является дистрибутивной и, следовательно, строгой.

L1. $f^*(\langle\rangle) = \langle\rangle$

L2. $f^*(\langle x \rangle) = \langle f(x) \rangle$

L3. $f^*(s \hat{} t) = f^*(s) \hat{} f^*(t)$

Остальные законы являются очевидными следствиями:

L4. $f^*(s)_0 = f(s_0)$ если $s \neq \langle\rangle$

L5. $\#f^*(s) = \#s$

Однако следующий очевидный закон, к сожалению, справедлив не всегда;

$f^*(s \upharpoonright A) = f^*(s) \upharpoonright f(A)$, где $f(A) = \{f(x) \mid x \in A\}$

Простейший контрпример представляет собой функция f такая, что

$f(b) = f(c) = c$, *где* $b \neq c$

Так как $b \neq c$, то, следовательно,

$$f^*(\langle b \rangle \upharpoonright \{c\}) = f^*(\langle\rangle)$$
$$= \langle\rangle$$
$$\neq \langle c \rangle$$
$$= \langle c \rangle \upharpoonright \{c\}$$
$$= f^*(\langle c \rangle) \upharpoonright f(\{c\}) \quad \text{так как } f(c) = c$$

Если же функция f взаимно однозначна, то данный закон выполняется:

L6. $f^*(s \upharpoonright A) = f^*(s) \upharpoonright f(A)$, при условии, что f –инъективная функция.

1.9.2. Конкатенация

Пусть *s* – последовательность, каждый элемент которой в свою очередь является последовательностью. Тогда ^/*s* получается из s конкатенацией всех ее элементов в их исходном порядке, например:

^/<<1, 3>, <>, <7> = <1, 3>^<>^<7> = <1, 3, 7>

Эта операция дистрибутивна:

L1. *^/<> = <>*
L2. *^/(s) = s*
L3. *^/(s^t) = (^/s)^(^/t)*

1.9.3. Чередование

Последовательность s является чередованием последовательностей t и u, если ее можно разбить на серию подпоследовательностей, которые, чередуясь, представляют собой подпоследовательности из t и u. Например,

s = <1, 6, 3, 1, 5, 4, 2, 7>

является чередованием t и u, где t = <1, 6, 5, 2, 7>, a u = <3, 1, 4>

Рекурсивное определение чередования можно задать следующими законами:

L1. <> *чередование(t, u)* ≡ *(t = <> & u<>)*
L2. s *чередование(t, u)* ≡ *s чередование(u, t)*
L3. (<x>^s) чередование*(t, u)* ≡

(t <> & t_0 = x & s чередование(t', u)
(u <> & u_0 = x & s чередование(t, u')

1.9.4. Индекс

Если *0 ≤ i ≤ #s*, то мы используем привычную запись *s[i]* для обозначения *i*-го элемента последовательности s, как это описано в законе L1:

L1. *s[0] = s0 & s[i + l] = s'[i]*, при условии, что *s ≠ <>*
L2. *(f*(s))[i] = f(s[i])* для *i < #s*

1.9.5. Обратный порядок

Если *s* – последовательность, то $\bar{s}$ получается из *s* взятием ее элементов в обратном порядке. Например,

$\overline{<3,5,37>}$ = <37, 5, 3>

Полностью обратный порядок задают следующие законы:

L1. $\overline{<>}$ = <>
L2. $\overline{<x>}$ = *x*
L3. $\overline{s^t}$ = $\bar{t}^\bar{s}$

Обратный порядок обладает рядом простых алгебраических свойств, в том числе:

L4. $\bar{\bar{s}}$ = *s*

Исследование остальных свойств мы оставляем читателю. Полезным является тот факт, что $\overline{s_0}$ есть *последний* элемент последовательности, а в общем случае

L5. $\overline{s[i]}$ = *s[#s – i – i]* для *i < #s*

1.9.6. Выборка

Если s – последовательность пар, определим s ↓ x как результат выборки из s всех пар, первый элемент которых есть x, и замены каждой такой пары на ее второй элемент, Первый и второй элементы пары будем разделять точкой. Так, если

s = <a.7, b.9, a.8, c.0 >, mo s ↓ a = <7, 8>, a s ↓ d = <>.

L1. <> ↓ *x* = <>

L2. *(<y, z>^t) ↓ x = t ↓ x* если *y ≠ x*
L3. *(<x.z>^t) ↓ x = <z>^(t ↓ x)*

Если s не является последовательностью пар, то *s ↓ a* обозначает число вхождений *a* в *s* {как это определялось в разд. 1.6.6).

1.9.7. Композиция

Пусть символ √ означает успешное завершение процесса. Значит, этот символ может появиться только в конце протокола. Пусть t – протокол, отражающий последовательность событий с того момента, как s успешно завершился. Композицию s и t обозначим (s; t). Если в s отсутствует символ √, то t не может начаться:
L1. *s; t = s* если ^ *(<√>)* **в** *s)*

Если же символ √ присутствует в конце s, то он удаляется а t присоединяется к результату:
L2. *(s^<√>); t = s^t* если *(<√>* **в** *s)*

Символ √ можно рассматривать как своего рода "клей", склеивающий *s* и *t*; при отсутствии клея *t* не может прилипнуть (**L1**). Если символ √ встречается (что ошибочно) в середине протокола, то из соображений общности условимся, что все символы после его первого вхождения следует считать ошибочными и опустить.
L2A. *(s^<√>^u); t = s^t* если *(<√>)* **в** *s)*

Эта необычная операция композиции обладает рядом обычных алгебраических свойств. Как и конкатенация, она ассоциативна. В отличие от конкатенации она монотонна как по первому, так и по второму аргументу с <√> в качестве левой, единицы.
L3. s; (t; u) = (s; t); u

L4A. s ≤ *t* ⇒ *((u; s) ≤ (u; t))*

L4B. *s ≤ t ⇒ ((s; u) ≤ (t; u))*
L5. *<>; t = t*
L6. *<√>; t = t*

Если символ √ не встречается нигде, кроме конца протокола, то <√> является также и правой единицей:
L7. *s; <> = s* при условии, что *^(<√>* **в** *(s)')*

1.10. СПЕЦИФИКАЦИИ

Спецификация изделия –это описание его предполагаемого поведения. Это описание представляет собой предикат, содержащий свободные переменные, каждая из которых соответствует некоторому обозримому аспекту поведения изделия. Например, спецификация электронного усилителя с входным диапазоном в один вольт и с усилением приблизительно в 10 раз задается предикатом

УСИЛ10 = (0 ≤ v ≤ 1 ⇒ | v' – 10v | ≤ 1)

Из этой спецификации ясно, что *v* обозначает входное, а *v'*– выходное напряжения. Без понимания физического смысла переменных вообще невозможно успешное применение математики в других науках и инженерном деле.

В случае процесса наиболее естественно в качестве результата наблюдения за его поведением рассматривать протокол событий, произошедших вплоть до данного момента времени. Для обозначения произвольного протокола процесса мы будем использовать специальную переменную пр, точно так же как в предыдущем примере оно' используются для произвольных значений данных наблюдений за напряжением.

Примеры

X1. Владелец торгового автомата не желает терпеть убытков от его установки. Поэтому он оговаривает, что число выданных шоколадок не должно превышать числа опущенных монет:

НЕУБЫТ = (#(пр ↾ {шок}) #(пр ↾ {мон}))

В дальнейшем мы будем использовать введенное в разд. 1.6.6 сокращение

пр ↓ с = # (пр ↾ {с})

для обозначения числа вхождений символа *с* в *пр*.

X2. Пользователь автомата хочет быть уверенным в том, что. машина не будет поглощать монеты, пока не выдаст ужа оплаченный шоколад:

ЧЕСТН1 = ((пр ↓ мон) ≤ (пр ↓ шок) +1)

X3. Изготовитель простого торгового автомата должен учесть требования как владельца, так и клиента:

ТАПВЗАИМ = НЕУБЫТ & ЧЕСТН1 = (0 ≤ ((пр ↓ мон) – (пр ↓ шок)) ≤ 1)

X4. Спецификация исправления сложного торгового автомата не позволяет последнему принимать три монеты подряду

*ТАСРЕМ = (^<n1>[3] **в** пр)*

X5. Спецификация исправленной машины

ИСПРТАС = (пр ∈ протоколы(ТАС) & ТАСРЕМ)

X6. Спецификация ТАП2 (1.1.3 **X6**)

0 ≤ ((пр ↓ мон) – (пр ↓ шок)) ≤ 2

1.10.1. Соответствие спецификации

Если P – объект[1] отвечающий спецификации S, мы говорим, что P удозлегеоряег 3, сокращенно

P УД *S*

Это означает, что *S* описывает все возможные результаты наблюдения за поведением *P*, или, другими словами, *S* истинно всякий раз, когда его переменные принимают значения, полученные в результате наблюдения за объектом P, или, более формально, ∀*пр.пр* ∈ *протоколы(P)* ⇒ *S*. В следующей таблице, например, приведены некоторые результаты наблюдений за свойствами усилителя;

	1	2	3	4	5
v	0	0.5	0.5	2	0.1
v’	0	5	4	1	3

Все наблюдения, кроме последнего, описываются УСИЛ10. Вторая и третья колонки иллюстрируют тот факт, что выход усилителя не зависит целиком от его входа. Четвертая колонка показывает, что, если входное напряжение выходит за пределы указанного диапазона, на выходе мы можем получить что угодно, и это не будет нарушением спецификации.

.

(В этом простом примере мы не учитываем того, что слишком высокое входное напряжение может привести к поломке объекта.)

В следующих законах приводятся наиболее общие свойства отношения *удовлетворяет*. Спецификации истина, не накладывающей никаких ограничений на поведение, будут удовлетворять все объекты; такой слабой и нетребовательной спецификации удовлетворяет даже неисправный объект:

L1. *P* **уд** *истина*

Если объект удовлетворяет двум различным спецификациям, он удовлетворяет также и их конъюнкции:

L2A. Если *P* **уд** *S* и *P* **уд** *T*, то *P* **уд** *(S & T)*

Закон **L2A** можно обобщить на бесконечное число конъюнкций, т. е. на предикаты, содержащие квантор всеобщности. Пусть *S(n)*–предикат, содержащий переменную *n*.

L2. Если ∀*n.(P* **уд** *S(n))*, то *P* **уд** (∀*n.S(n))*

при условии что *P* не зависит от *n*.

Если из спецификации *S* логически следует другая спецификация *T*, то всякое наблюдение, описываемое *S*, описывается также и *T*. Следовательно, каждый объект, удовлетворяющий *S*, должен удовлетворять также и более слабой спецификации *T*:

L3. Если *P* **уд** *S* и *S*⇒*T*, то *P* **уд** *T*

В свете этого закона мы будем иногда записывать доказательства в виде цепочки,, т.е., если S ⇒ T, мы запишем

P **уд** *S*

⇒*T*

как сокращение более полного доказательства:

P **уд** *S*

S ⇒ *T*

P **уд** *T* по **L3**

---

[1] В оригинале автор называет специфицируемый предмет продуктом (product); однако использование русского слова <продукт> вносит нежелательные смысловые аберрации. Поэтому в качестве русского эквивалента взяты неспецифические термины «объект» или «изделие», выбор между которыми определяется контекстом. – *Прим. перев.*

Приведенные выше законы и пояснения к ним применимы ко всем видам объектов и ко всем видам спецификаций. В следующем разделе будут приведены дополнительные законы, применимые к процессам.

1.10.2. Доказательства

При проектировании изделия разработчик несет ответственность за то, чтобы оно соответствовало своей спецификации; Эта ответственность может быть реализована посредством

обращения к методам соответствующих разделов математики, например геометрии или дифференциального и интегрального исчисления. В этом разделе мы приводим набор законов, позволяющих с помощью математических рассуждений убедиться в том, что процесс *P* соответствует своей спецификации *S*.

Иногда мы будем записывать спецификацию как *S(пр)*, предполагая, что спецификация обычно содержит *пр* в качестве свободной переменной. В действительности же мы явно записываем *пр*, чтобы указать, что ее можно заменить на более сложное выражение, как, например, в S*(пр")*. Важно отметить, что как *S*, так и *S(пр)* кроме *пр* могут иметь и другие свободные переменные.

Результатом наблюдения за процессом *СТОП* всегда будет пустой протокол, так как этот процесс никогда ничего не делает:

L4A. *СТОП* **уд** *(пр = <>)*

Протокол процесса (с→Р) вначале пуст. Каждый последующий протокол начинается с с, а его хвост является протоколом Р. Следовательно, хвост может быть описан любой спецификацией для Р:

L4B. Если *P* **уд** *S(пр),* то *(с→P)* **уд** *(пр = <> ∨ (пр$_0$ = с & S(пр')))*

В качестве следствия получаем закон для двойной префиксации:

L4C. Если *P* **уд** *S(пр)*, то *(с → d → P)* **уд** *(пр ≤ <с, d> ∨ (пр ≥ <с, d> & S<пр">)))*

Двуместный выбор аналогичен префиксации с той разницей, что протокол может начинаться с любого из двух событий–альтернатив, а хвост должен описываться спецификацией выбранной альтернативы:

L4D. Если *P* **уд** *S(пр),* а *Q* **уд** *T(пр),* то *(с → P) | d → Q)*

уд *(пр = <> ∨ (пр$_0$ = с & S(пр')) ∨ (пр$_0$ = d & T(пр"))*

Все приведенные выше законы являются частными случаями закона для обобщенного оператора, выбора:

L4. Если ∀*x* ∈ *B.(P(x)* **уд** *S(пр, x)),* то *(x: B → P(x))* **уд** *(пр = <> ∨ (пр$_0$ ∈ B & S(пр', пр$_0$)))*

Закон для оператора *после* удивительно прост. Если *пр* – протокол (Р / s), то s^пр является протоколом Р и поэтому должен описываться любой спецификацией, которой удовлетворяет Р:

L5. Если *P* **уд** *S(пр),* а *s* ∈ *протоколы(P),* то *(P / s)* **уд** *S(s^пр)*

И, наконец, нам нужен закон, чтобы устанавливать корректность рекурсивного определяемого процесса.

L6. Если *F(X)* – предваренная, *СТОП* **уд** *S,* а *((X уд S) ⇒ (F(X) уд S))*, то *(μX.F(X))* **уд** *S*

Из левой части этого закона по индукции следует, что

F^n(СТОП) **уд** *S* для всех *n*

Так как F предварена, то F^n(СТОП) полностью описывает как минимум n первых шагов в поведении *μX.F(X)*. Поэтому каждый протокол *μX.F(X)* является протоколом F^n(СТОП) для некоторого *n*. Значит, этот протокол должен удовлетворять той же спецификации, что и F^n(СТОП), которая (для всех *n*) есть *S*. Более строгое доказательство можно дать в терминах математической теории из разд. 2.8.

Пример

X1. Мы хотим доказать (1.1.2 **X2**, 1.10 **X3**), что

ТАП уд *ТАПВЗАИМ*

Доказательство

(1) *СТОП* **уд** *пр = <>* **L4A**

⇒ *0 ≤ (пр ↓ мон – пр ↓ шок) ≤* 1,

так как *(<>↓ мон) = (<>↓ шок)* = 0

Это заключение сделано на основании неявного применения закона L3.

(2) Предположим, что *X* **уд** *(0 ≤ ((пр ↓ мон) – (пр ↓ шок)) ≤* 1*)*.

Тогда *(мон → шок → X)* **уд** *(пр ≤ <мон, шок> ∨ (пр ≥ <мон, шок>*

$$\& \; 0 \le ((пр'' \downarrow мон) - (пр'' \downarrow шок)) \le 1)) \qquad \textbf{L4C}$$

$$\Rightarrow \; 0 \le ((пр \downarrow мон) - (пр \downarrow шок)) \le 1,$$

так как $<> \downarrow мон = <> \downarrow шок = <мон> \downarrow шок = 0$,

а $<мон> \downarrow мон = <мон, шок> \Rightarrow (пр \downarrow мон = пр'' \downarrow мон + 1 \; \& \; пр \downarrow шок = пр'' \downarrow шок + 1)$.
Применяя теперь закон L3, а затем L6, получим требуемый результат.

Тот факт, что процесс *P* удовлетворяет спецификации, еще не означает, что он будет нормально функционировать, Например, так как

$$пр = <> \Rightarrow 0 \le (пр \downarrow мон - пр \downarrow шок) \le 1$$

то с помощью законов L3 и L4 можно доказать, что

$$СТОП \textbf{ уд } 0 \le (пр \downarrow мон - пр \downarrow шок) \le 1$$

Однако *СТОП* вряд ли соответствует требованиям торгового автомата, с точки зрения как владельца, так и клиента. Он, несомненно, не сделает ничего плохого, но только благодаря тому. что он не делает ничего вообще. По этой же причине *СТОП* удовлетворяет любой спецификации, которой может удовлетворять процесс.

К счастью, по независимым соображениям очевидно, что *ТАП* никогда не останавливается. На самом деле, любой процесс, определенный только с помощью префикса, выбора и предваренной рекурсии, никогда не останавливается. Единственный способ записать останавливающийся процесс – это явно включить в него *СТОП* или эквивалентный процесс$(x : B \rightarrow P(x))$, где *B* – пустое множество. Избегая таких элементарных ошибок, можно с гарантией записывать бесконечные процессы. Однако после введения в следующей главе параллелизма такие простые меры предосторожности становятся недостаточными. Более общий способ спецификации и доказательства свойства бесконечности процесса описывается в разд. 3.7.

Глава 2. Параллельные процессы

2.1. ВВЕДЕНИЕ

Процесс определяется полным описанием его потенциального поведения. При этом часто имеется выбор между несколькими различными действиями, например опусканием большой или маленькой монеты в щель торгового автомата ТАС (1.1.3 **X4**). В каждом таком случае выбор того, какое из событий произойдет в действительности, может зависеть от окружения, в котором работает процесс. Так, например, покупатель сам решает, какую монету ему опустить. К счастью, само окружение процесса может быть описано как процесс, поведение которого определяется в уже знакомых нам терминах. Это позволяет исследовать поведение целой системы, состоящей из процесса и его окружения, взаимодействующих по мере их параллельного исполнения. Всю систему также следует рассматривать как процесс, поведение которого определяется в терминах поведения составляющих его процессов; эта система в свою очередь может быть помещена в еще более широкое окружение. На самом деле, лучше всего забыть разницу между процессами, окружениями и системами; все они – всего лишь процессы, поведение которых может быть предписано, описано, зафиксировано и проанализировано простым и единообразным способом.

2.2. ВЗАИМОДЕЙСТВИЕ

Объединяя два процесса для совместного исполнения, мы, как правило, хотим, чтобы они взаимодействовали друг с другом. Эти взаимодействия можно рассматривать как события, требующие одновременного участия обоих процессов. Давайте на некоторое время сосредоточимся лишь на таких событиях и забудем обо всех остальных. Будем поэтому предполагать, что алфавиты обоих процессов совпадают. Следовательно, каждое происходящее событие является допустимым в независимом поведении каждого процесса в отдельности. Шоколадку, например, можно извлечь, только когда этого хочет покупатель и только когда автомат готов, ее выдать. Если P и Q – процессы с одинаковым алфавитом,

обозначим через $P \parallel Q$ процесс, ведущий себя как система, составленная из процессов P и Q, взаимодействие между которыми пошагово синхронизовано, как это описано выше.

Примеры

X1. Жадный покупатель был бы не прочь получить ириску или даже шоколадку бесплатно. Однако, если это не вышло, он с неохотой готов заплатить, но при этом настаивает на получении шоколадки:

ЖАДН = (ирис → ЖАДН | шок → ЖАДН | мон → шок → ЖАДН)

Когда такой покупатель сталкивается с автоматом *ТАШИ* (1.1.3 **X3**), его алчные попытки оказываются несостоятельными, потому что этот торговый автомат не позволяет получить товар прежде, чем он будет оплачен:

(ЖАДН || ТАШИ) = μX.(мон → шок → X)

Этот пример показывает, как можно описать процесс, заданный в виде композиции двух подпроцессов, без использования параллельного оператора ||.

X2. Рассеянный покупатель хочет большой коржик и поэтому опускает монету в торговый автомат *ТАС*. Он не обратил внимания, какую монету он опустил – большую или маленькую, однако настаивает только на большом коржике:

РАС = (п2 → бол → РАС | п1 → бол → РАС)

К сожалению, торговый автомат не предусматривает выдачу большого коржика за маленькую монету:

(РАС || ТАС) = μX. (п2 → бол → X | п1 → СТОП)

Событие *СТОП*, следующее за первым же *п1*, известно как тупиковая ситуация или *дедлок*. Хотя каждый из составляющих процессов готов к некоторым дальнейшим действиям, действия эти различны. И так как процессы не могут прийти к соглашению о том, какое действие будет следующим, ничего в дальнейшем произойти не может.

Сюжеты, сопровождающие эти примеры, являют собой образец полной измены принципу научной абстракции и объективности. Важно помнить, что, согласно нашему предположению, события являются

нейтральными переходами, которые могут наблюдаться и фиксироваться некоторым бесстрастным пришельцем с другой планеты, не ведающим ни радости поедания коржика, ни голода незадачливого покупателя, тщетно пытающегося добыть себе пропитание. При выборе алфавита соответствующих событий мы сознательно не включили в него эти внутренние эмоциональные состояния; при желании же дополнительные события, моделирующие изменения «внутреннего» состояния, могут быть введены, как Доказано в 2.3 **X1.**

2.2.1. Законы

Законы, управляющие поведением (P || Q), исключительно просты и регулярны. Первый закон выражает логическую симметрию между процессом и его окружением:
L1. $P||Q = Q||P$.
Следующий закон показывает, что при совместной работе трех процессов неважно, в каком порядке они объединены оператором параллельной композиции:
L2. $P \parallel (Q||R) = (P||Q) \parallel R$
В-третьих, процесс, находящийся в тупиковой ситуации, приводит к дедлоку всей системы: композиция же с $ИСП_{\alpha P}$ (1.1.3 **X8**) ничего не меняет:
L3A. $P||СТОП_{\alpha P} = СТОП_{\alpha P}$
L3B. $P||ИСП_{\alpha P} = P$
Следующий закон гласит, что пара процессов либо одновременно выполняет одно и то же действие, либо попадает в состояние дедлока, если начальные события процессов не совпадают:
L4A. $(c \to P) \parallel (c \to Q) = (c \to (P||Q))$
L4B. $(c \to P) \parallel (d \to Q) = СТОП$, если $c \neq d$
Эти законы легко обобщить на случаи, когда у одного или обоих процессов имеется выбор начального события: при комбинации процессов остаются возможными лишь те события, которые содержатся во множествах выбора обоих процессов:
L4. $(x{:}A \to P(x)) \parallel (y{:}B \to Q(y)) = (z{:}(A \cap B) \to (P(z) \parallel Q(z))$
Именно этот закон позволяет описать систему, определенную в терминах параллельного исполнения без использования параллелизма.

Примеры
X1. Пусть $P = (a \to b \to P \mid b \to P)$, а $Q = (a \to (b \to Q \mid c \to Q))$
Тогда $(P \parallel Q) = a \to ((b \to P) \parallel (b \to Q \mid c \to Q))$ по **L4A**
$= a \to (b \to (P \parallel Q))$ по **L4A**
$= \mu X.(a \to b \to X)$ так как рекурсия предварена

2.2.2. Реализация

Реализация комбинирующего оператора || очевидным образом основана на **L4**:
взаим(P,Q) = λz.

if *P(z)* = "*BLEEP* ∨ *Q(z)* = "*BLEEP* **then** "*BLEEP*
else *взаим(P(z),Q(z))*

2.2.3. Протоколы

Поскольку каждое действие $(P||Q)$ требует одновременного участия P и Q, каждая последовательность таких действий должна быть возможной для обоих операндов. По этой же причине операция $/s$ дистрибутивна относительно ||.
L1. *протоколы(P||Q) = протоколы(P)* ∩ *протоколы(Q)*
L2. $(P||Q)/s = (P/s) \parallel (Q/s)$

2.3. ПАРАЛЛЕЛИЗМ

Описанный в предыдущем разделе оператор можно обобщить на случай, когда его операнды P и Q имеют различные алфавиты:
$\alpha P \neq \alpha Q$

Когда такие процессы объединяются для совместного исполнения, события, содержащиеся в обоих алфавитах, требуют одновременного участия P и Q (как было описано в предыдущем разделе). События же из алфавита P, не содержащиеся в алфавите Q, не имеют никакого отношения к Q, который физически неспособен ни контролировать, ни даже замечать их. Такие события могут происходить при участии P независимо от Q. Аналогично Q может самостоятельно участвовать в событиях, содержащихся в его алфавите, но отсутствующих в алфавите P. Таким образом, множество всех событий, логически возможных для данной системы, есть просто объединение алфавитов составляющих ее процессов:

$\alpha(P \parallel Q) = \alpha P \cup \alpha Q$

Это редкий пример операции, когда у операндов и результата различные алфавиты. Если же алфавиты операндов совпадают, то этот же алфавит имеет и их результирующая комбинация, и в этом случае у операции $(P \parallel Q)$ в точности тот же смысл, что и у операции, описанной в предыдущем разделе.

Примеры

X1. Пусть *αТАШУМ = {мон, шок, звяк, щелк, ирис}*, где *«звяк»* – звук монеты, попадающей в монетоприемник шумного торгового автомата, а *«щелк»* – щелчок, издаваемый автоматом при завершении каждой торговой операции. Если у шумного торгового автомата кончился запас ирисок, то

ТАШУМ = (мон → звяк → шок → щелк → ТАШУМ)

Покупатель у этого автомата определенно предпочитает ириску: не сумев получить ее, он восклицает «черт», после чего берет шоколадку:

αКЛИЕНТ = {мон, шок, черт, ирис}

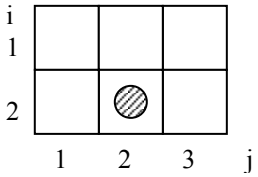
КЛИЕНТ = (мон → (ирис → КЛИЕНТ) | черт → шок → КЛИЕНТ)

Результатом параллельной работы этих двух процессов является

(ТАШУМ || КЛИЕНТ) = μX. (мон → (звяк → черт → шок → щелк → X
| черт → шок → КЛИЕНТ)

Заметим, что событие *звяк* может как предшествовать событию *черт*, так и наоборот. Они могут произойти и одновременно, и порядок, в котором они будут зафиксированы, не имеет значения. Заметим, что настоящая математическая формула никак не отражает того факта, что клиент предпочитает получить ириску, нежели выругаться. Формула есть абстракция реальной действительности, не учитывающая людские эмоции и сосредоточенная лишь на описании возможности наступления или ненаступления событий из алфавита процесса независимо оттого, желательны они или нет.

X2. Фишка стартует со средней нижней клетки поля и может



по нему *вверх, вниз, влево* или *вправо*. Пусть

αP = {вверх, вниз}

P = (вверх → вниз → P)

αQ = {влево, вправо}

Q = (вправо → влево → Q | влево → вправо → Q)

Поведение фишки можно определить как $P \parallel Q$. В этом примере алфавиты αP и αQ не имеют общих событий. Перемещения фишки, следовательно, представляют собой произвольное чередование действий процессов P и Q. Без помощи параллелизма описать такое чередование очень сложно. Обычно требуется взаимная рекурсия с уравнением для каждого состояния системы. Пусть, например, R_{ij} отвечает поведению фишки (**X2**), находящейся в i-строке и j-м столбце поля, где $i \in \{1,2\}$, $j \in \{1,2,3\}$. Тогда $(P \parallel Q) = R_{12}$, где

$R_{21} = (вниз \rightarrow R_{11} \mid вправо \rightarrow R_{22})$

$R_{11} = (вверх \rightarrow R_{21} \mid вправо \rightarrow R_{12})$

$R_{22} = (вниз \rightarrow R_{12} \mid влево \rightarrow R_{21} \mid вправо \rightarrow R_{23})$

$R_{12} = (вверх \rightarrow R_{22} \mid влево \rightarrow R_{11} \mid вправо \rightarrow R_{13})$

$R_{23} = (вниз \rightarrow R_{13} \mid влево \rightarrow R_{22})$

$R_{13} = (вверх \rightarrow R_{23} \mid влево \rightarrow R_{12})$

2.3.1. Законы

Первые три закона для расширенной операции параллельной композиции совпадают с аналогичными законами для оператора взаимодействия (2.2.1).

L1.2. Операция || симметрична и ассоциативна

L3A. $P \parallel СТОП_{\alpha P} = СТОП_{\alpha P}$

L3B. $P \parallel ИСП_{\alpha P} = P$

Пусть $a \in (\alpha P - \alpha Q)$, $b \in (\alpha Q - \alpha P)$, а $\{c, d\} \subseteq (\alpha P \cap \alpha Q)$. Следующие законы показывают, каким образом процесс P один участвует в событии a, Q один участвует в b, а c и d требуют одновременного участия P и Q.

L4A. $(c \to P \parallel c \to Q) = c \to (P \parallel Q)$

L4B. $(c \to P \parallel d \to Q) = СТОП$, если $c \neq d$

L5A. $(a \to P) \parallel (c \to Q) = a \to (P \parallel (c \to Q)$

L5B. $(c \to P) \parallel (b \to Q) = b \ ((c \to P) \parallel Q)$

L6. $(a \to P) \parallel (b \to Q) = (a \to (P \parallel (b \to Q)) \mid b \to ((a \to Q) \parallel Q))$

Эти законы можно обобщить для общего случая оператора выбора:

L7. Пусть $P = (x : A \to P(x))$, $Q = (y : B \to Q(y))$.

Тогда $(P \parallel Q) = (z : C \ (P' \parallel Q'))$,

где $C = (A \cap B) \cup (A - \alpha Q) \cup (B - \alpha P)$,

$P' = P(z)$, если $z \in A$

$= P$ иначе,

а $Q' = Q(z)$, если $z \in B$

$= Q$ иначе.

С помощью этих законов можно переопределить процесс, удалив из его описания параллельный оператор, как это показано в следующем примере.

Пример

X1. Пусть $\alpha P = \{a, c\}$, $\alpha Q = \{b, c\}$, $P = (a \to c \to P)$, $Q = (c \to b \to Q)$.

Тогда $P \parallel Q = (a \to c \to P) \parallel (c \to b \to Q)$ по определению

$= a \to ((c \to P) \parallel (c \to b \to Q))$ по **L5**

$= a \to c \to (P \parallel (b \to Q))$ по **L4**…(1)

$a \to P \parallel (b \to Q) = (a \to (c \to P) \parallel (b \to Q) \mid b \to (P \parallel Q)$ по **L6**

$= (a \to b \to ((c \to P) \parallel Q) \mid b \to (P \parallel Q)$ по **L5**

$= (a \to b \to a \to (P \parallel (b \to Q)) \mid b \to a \to c \to (P \parallel (b \to Q)))$ по **L4** и (1) выше

$= \mu X. (a \to b \to c \to X \mid b \to a \to c \to X)$ в силу предваренности

Отсюда следует, что

$(P \parallel Q) = (a \ c \ \mu X. (a \ b \ c \ X \mid b \ a \ c \ X))$

2.3.2. Реализация

Реализация операции || выводится непосредственно из закона **L7**. Алфавиты операндов представляются конечными списками символов A и B. В проверке на принадлежность используется функция *принадлежит(x, A)* определенная в разд. 1.7.

(P || Q) реализуется вызовом функции

параллельно(P, αP, αQ, q)

определенной как

параллельно(P, A, B, Q) = вспом(P, Q)

где

вспом(P,Q) =

λx. **if** *P = "BLEEP* **or** *Q = "BLEEP* **then** *"BLEEP*

else if *принадлежит(x, A) & принадлежит(x, B)* **then** *вспом(P(x), Q(x))*

else if *принадлежит(x, A)* **then** *вспом(P(x), Q)*

else if *принадлежит(x, A)* **then** *вспом(P, Q(x))*

else *"BLEEP*

2.3.3. Протоколы

Пусть *t* – протокол *(P || Q)*. Тогда все события из t, принадлежащие алфавиту *P,* являлись событиями в жизни *P*. а все события из *t*, не принадлежащие *αP*, происходили без участия *P*. Таким образом, *(t ↾ αP)* – это протокол всех событий, в которых участвовал процесс *P*, и поэтому он является протоколом *P*. По тем же соображениям *(t ↾ αQ)* является протоколом *Q*. Более того, каждое событие из *t* должно содержаться либо в *αP*, либо в *αQ*. Эти рассуждения позволяют сформулировать закон

L1 протоколы((P || Q) = {t | (t ↾ αP) ∈ протоколы(P) & (t ↾ αQ) ∈ протоколы(Q) & t ∈ (αP ∪ αQ)*}

Следующий закон демонстрирует дистрибутивность операции /s относительно оператора параллельной композиции:

L2. (P || Q)/s = (P/(s ↾ αP)) || (Q/(s ↾ αQ))

Если *α*P = *α*Q, то s ↾ *α*P = s ↾ *α*Q = s, и тогда законы **L1**, **L2** совпадают с аналогичными из разд. 2.2.3.

Пример

X1. См. 2.3 **X1**

Пусть *t1 = <мон, звяк, черт>*.

Тогда *t1 ↾ ТАШУМ = <мон, звяк>,* что принадлежит множеству *протоколы(ТАШУМ),*

t1 ↾ КЛИЕНТ = <мон, черт>, что принадлежит множеству *протоколы(КЛИЕНТ).*

Следовательно, *t1 ∈ протоколы(ТАШУМ || КЛИЕНТ).* Аналогичные рассуждения показывают, что *<мон, черт, звяк> ∈ протоколы(ТАШУМ || КЛИЕНТ)*

Отсюда видно, что события мон и звяк могут происходить и фиксироваться в любом порядке. Они могут произойти и одновременно, но мы решили никак не отражать этот факт. В общих словах, протокол *(P || Q)* представляет собой некоторую разновидность чередования протокола *P* с протоколом *Q,* в котором события, содержащиеся в обоих алфавитах, записаны по одному разу. Если *αP ∩ αQ* = {}, то этот протокол в чистом виде является чередованием (разд. 1.9.3), как показано в примере 2.3 **X2**. В другом крайнем случае, когда *αP = αQ*, каждое событие принадлежит обоим алфавитам, и *(P || Q)* имеет в точности тот же смысл, что и взаимодействие, определенное в разд.2.2.

L3A. Если *αP ∩ αQ* = {}, то

протоколы*(P || Q) = {s | ∃t: протоколы(P). ∃u: протоколы(Q). s чередование(t, u)}*

L3B. Если *αP = αQ*, то

протоколы*(P || Q) = протоколы(P) ∩ протоколы(Q*

2.4. РИСУНКИ

Процесс P с алфавитом {a, b, c} изображается прямоугольником с меткой P, из которого исходят линии, помеченные различными событиями из его алфавита (рис. 2.1). Аналогично процесс Q с алфавитом {b, c, d} можно изобразить, как показано на рис. 2.2. Когда эти два процесса объединяются для параллельного исполнения, полученную систему можно

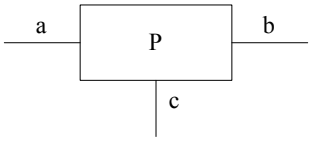


Рис 2.1

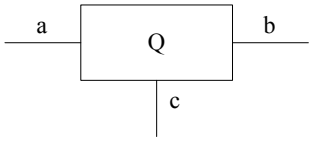


Рис 2.2

изобразить в виде сети, в которой линии с одинаковыми метками соединены, а линии, помеченные событиями, принадлежащими алфавиту только одного процесса, оставлены свободными (рис. 2.3). Сюда можно добавить третий процесс R с алфавитом αR= {c, e}, как показано на рис. 2.4. Из этой диаграммы видно, что событие с требует участия всех трех процессов, b требует участия P и Q, тогда как все остальные события имеют отношение только к отдельным процессам.

Рисунки эти, однако, легко могут ввести в заблуждение. Система, построенная из трех процессов, все же представляет собой один процесс и должна поэтому изображаться одним прямоугольником (рис. 2.5). Число 60 тоже можно получить как произведение трех других чисел, (3 ⨯ 4 ⨯ 5), но после того, как оно было таким образом получено, это всего лишь отдельное число, и способ его получения перестает быть

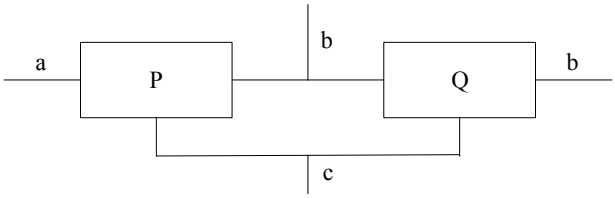


Рис 2.3

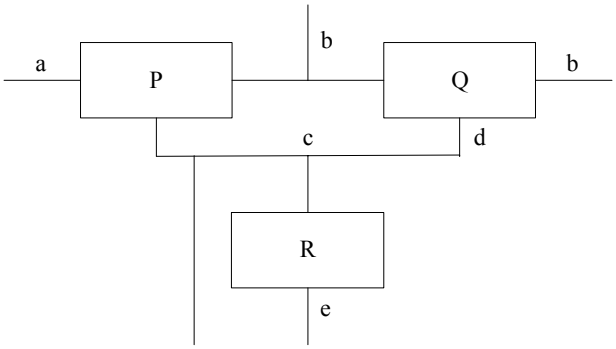


Рис 2.4

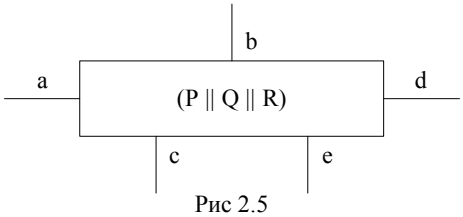


Рис 2.5

достойным внимания.

2.5. ПРИМЕР: ОБЕДАЮЩИЕ ФИЛОСОФЫ

В давние времена один богатый филантроп пожертвовал свой капитал на основание некоего пансиона, чтобы дать пристанище пяти знаменитым философам. У каждого философа была своя комната, где он мог предаваться размышлениям.

Была у них и общая столовая с круглым столом, вокруг которого стояли пять стульев, каждый помеченный именем того философа, которому он предназначался. Звали философов $ФИЛ_0$, $ФИЛ_1$, $ФИЛ_2$, $ФИЛ_3$ и $ФИЛ_4$, и за столом они располагались в этом же порядке против часовой стрелки. Слева от каждого философа лежала золотая вилка, а в центре стола стояла большая миска спагетти, содержимое которой постоянно пополнялось.

Предполагалось, что большую часть времени философ проводил в размышлениях, а почувствовав голод, шел в столовую, садился на свой стул, брал слева от себя вилку и приступал к еде. Но такова уж сложная природа спагетти, что их не донести до рта без помощи второй вилки. Поэтому философу приходилось брать вилку и справа от себя. Закончив трапезу, он клал на место обе вилки, выходил из-за стола и возвращался к своим размышлениям. Разумеется, одной вилкой философы могли пользоваться только по очереди. Если вилка требовалась другому философу, ему приходилось ждать, пока она освободится.

2.5.1. Алфавиты

Теперь построим математическую модель этой системы. Сначала надо выбрать подходящие множества событий. Для $ФИЛ_i$ определим это множество как

$\alpha ФИЛ_i$ = {*i. садится, i. встает, i. берет вилку. i, i. берет вилку*$(i \oplus 1)$,

i. кладет вилку. i, i. кладет вилку$(i \oplus 1)$}

где $\oplus$ –сложение по модулю 5, т. е. $i \oplus 1$ указывает правого соседа i-го философа. Заметим, что алфавиты философов друг с другом не пересекаются. Ни в одном из событий философы не участвуют совместно, и поэтому возможность какого бы то ни было общения или взаимодействия между ними

исключена – весьма реалистическое отражение поведения философов тех дней.

Другие «действующие лица» в нашей маленькой драме – это пять вилок, каждая из которых имеет тот же номер, что и философ, которому она принадлежит. Вилку берет и кладет на место или сам этот философ, или его сосед слева. Алфавит *i*-й вилки определим как

$\alpha ВИЛКА_i = \{i.$ *берет вилку.* $i, (i \ominus 1).$ *берет вилку.* i , $i.$ *кладет вилку.* $i, (i \ominus 1).$ *кладет вилку.* $i\}$

где $\ominus$ обозначает вычитание по модулю 5.

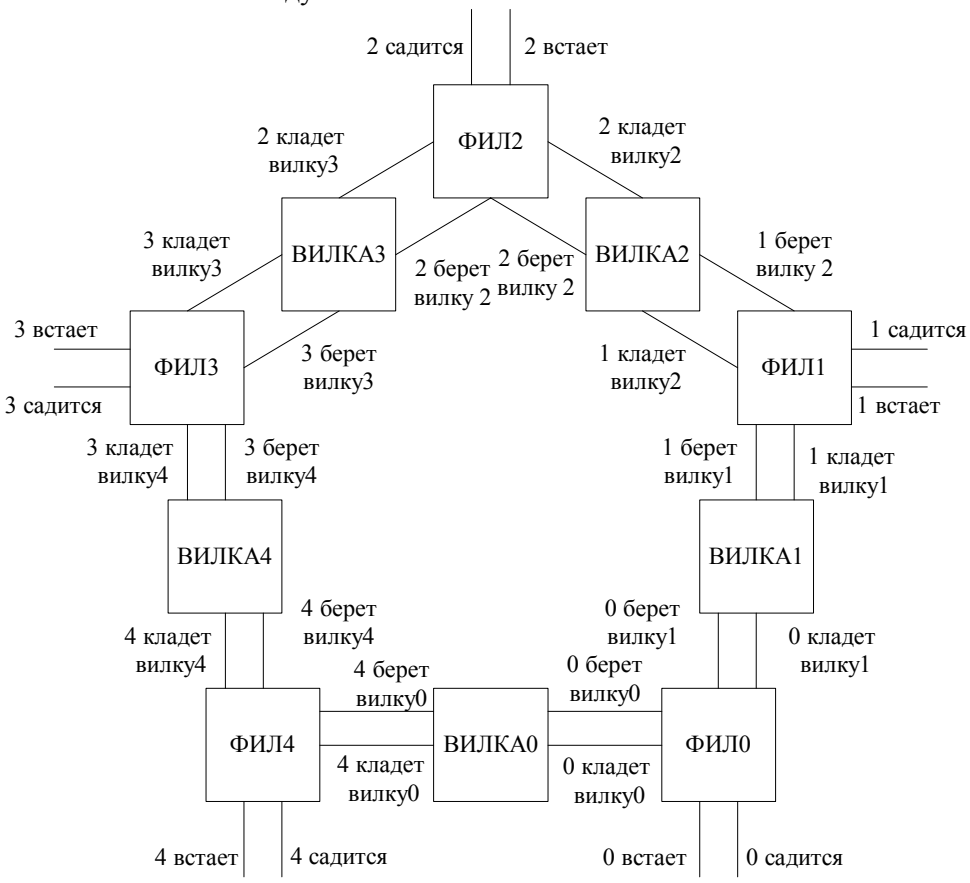


Рис 2.6

Таким образом, каждое событие, кроме *садится* и *встает*, требует участия ровно двух соседних действующих лиц – философа и вилки, как показано на диаграмме связей на рис. 2.6.

2.5.2. Поведение

Помимо процессов размышления и еды, которыми мы решили пренебречь, жизнь каждого философа представляет собой повторение цикла из шести событий:

$ФИЛ_i =$ *(*$i.$ *садится* $\rightarrow i.$ *берет вилку.* $i \rightarrow i.$ *берет вилку*$(i \oplus 1)$

$\rightarrow i.$ *кладет вилку.* $i \rightarrow i.$ *кладет вилку*$(i \oplus 1) \rightarrow i.$ *встает* $\rightarrow ФИЛ_i)$

Роль вилки проста: ее неоднократно берет и кладет на место кто-нибудь из соседних с ней философов:

$ВИЛКА_i =$ *(*$i.$ *берет вилку.* $i \rightarrow i.$ *кладет вилку.* $i \rightarrow ВИЛКА_i$

$|$ *(*$i \ominus 1$*). берет вилку.* $i \rightarrow$ *(*$i \ominus 1$*). кладет вилку.* $i \rightarrow ВИЛКА_i$ *)*

Поведение всего пансиона можно описать как параллельную комбинацию поведения следующих компонент:

ФИЛОСОФЫ=($ФИЛ_0$ || $ФИЛ_1$ || $ФИЛ_2$ || $ФИЛ_3$ || $ФИЛ_4$)

ВИЛКИ = ($ВИЛКА_0$ || $ВИЛКА_1$ || $ВИЛКА_2$ || $ВИЛКА_3$ || $ВИЛКА_4$)

ПАНСИОН = (ФИЛОСОФЫ || ВИЛКИ)

В одном из интересных вариантов этой истории философы могут брать и класть обе вилки в любом порядке. Рассмотрим поведение каждой руки философа в отдельности. Каждая рука может самостоятельно взять соответствующую вилку, но для того, чтобы сесть или встать, требуются обе руки:

$\alpha ЛЕВАЯ_i$ = {i. берет вилку. i, i. кладет вилку. i, i.садится, i. встает}

$\alpha ПРАВАЯ_i$ = {i. берет вилку.i, i. кладет вилку. i, i. садится, i. встает}

$ЛЕВАЯ_i$ = (i. садится → i. берет вилку. i → i. кладет вилку. i → i. встает → $ЛЕВАЯ_i$)

$ПРАВАЯ_i$ = (i. садится → i. берет вилку. (i ⊕ 1) → i. кладет вилку. (i ⊕ 1) → i. встает → $ПРАВАЯ_i$)

$ФИЛ_i = ЛЕВАЯ_i \,||\, ПРАВАЯ_i$

Синхронизацией процессов $ЛЕВАЯ_i$ и $ПРАВАЯ_i$ в момент, когда философ садится или встает, достигается то, что вилка не может быть поднята, если соответствующий философ не сидит за столом. Помимо этого, операции с обеими вилками произвольно чередуются.

В еще одном варианте этой истории философ, находясь за столом, может поднимать и опускать каждую вилку многократно. Для этого придется изменить поведение обеих рук, введя итерацию, например:

$ЛЕВАЯ_i$ = (i. садится → μX.(i. берет вилку. i → i. кладет вилку. i → X
| i. встает → $ЛЕВАЯ_i$)

2.5.3. Дедлок!

Построенная математическая модель позволяет обнаружить серьезную опасность. Предположим, что все философы проголодаются примерно в одно и то же время. Все они сядут, все возьмут в левую руку вилку, все потянутся за второй – которой уже нет на месте. В столь недостойной ситуации всех их неизбежно ждет голодная смерть. Несмотря на то, что каждый из участников способен к дальнейшим действиям, ни одна пара участников не в состоянии договориться о том, какое действие совершить следующим.

Конец нашей истории, однако, не столь печален. Как только опасность была обнаружена, было предложено множество способов ее избежать. Один из философов, например, всегда может брать сначала правую вилку – если бы только они сумели договориться, кто из них будет это делать! По тем же причинам исключается покупка одной дополнительной вилки; покупка же пяти новых вилок обошлась бы слишком дорого.

Окончательным решением проблемы явилось появление слуги, в чьи обязанности входило сопровождать каждого философа за стол и из-за стола. Алфавит его был определен как

$$\bigcup_{i=0}^{4} \{i.\ садится,\ i.\ встает\}$$

Слуге было дано секретное указание следить за тем, чтобы за столом никогда не оказывалось больше четырех философов одновременно. Его поведение проще всего описать с помощью взаимной рекурсии. Пусть

$$B = \bigcup_{i=0}^{4} \{i.\ встает\},\quad C = \bigcup_{i=0}^{4} \{i.\ садится\}$$

$СЛУГА_j$ определяет поведение слуги, когда за столом сидят j философов:

$СЛУГА_0 = (x : C \rightarrow СЛУГА_1)$

$СЛУГА_j = (x : C \rightarrow СЛУГА_{j+1} \mid y : B \rightarrow СЛУГА_{j-1})$ для $j \in \{1, 2, 3\}$

$СЛУГА_4 = (y : B \rightarrow СЛУГА_3)$

Пансион, свободный от дедлока, определяется как

$НОВПАНСИОН = (ПАНСИОН \,||\, СЛУГА_0)$

Эта поучительная история об обедающих философах принадлежит Эдсгеру В. Дейкстре. Слуга появился благодаря Карелу С. Шолтену.

2.5.4. Доказательство беступиковости

В первоначальном *ПАНСИОНе* риск тупиковой ситуации был далеко не очевиден, и поэтому утверждение, что *НОВПАНСИОН* свободен от дедлоков, следует доказывать с известной аккуратностью. То, что мы хотим доказать, строго можно сформулировать как

$(НОВПАНСИОН/s) \neq СТОП$ для всех $s \in$ протоколы(НОВПАНСИОН)

Для доказательства возьмем произвольный протокол s и покажем, что в любом случае найдется хотя бы одно событие, которым можно продолжить s и по-прежнему получить протокол *НОВПАНСИОН*. Сначала определим число сидящих философов:

сидят(s) = #(s ↾ C) – #(s ↾ B), где *C* и *B* определены ранее. Так как, согласно закону 2.3.3,

$s \upharpoonright (B \cup C) \in$ *протоколы*$(СЛУГА_0)$ мы знаем, что

сидят(s) $= s \leq 4$

Если *сидят(s)* ≤ 3, то мы знаем, что еще по крайней мере один философ может сесть и это не приведет к дедлоку. В оставшемся случае, когда сидят *(s)* = 4, рассмотрим число философов, которые едят (т. е. обе их вилки подняты). Если это число не равно нулю, то такой философ всегда может положить левую вилку. Если же никто из философов не ест, рассмотрим число поднятых вилок. Если оно меньше или равно трем, то один из сидящих философов по-прежнему может поднять левую вилку. Если поднято четыре вилки, это значит, что философ, сидящий рядом со свободным местом, уже поднял левую вилку и может взять правую. Если же поднятых вилок пять, это означает, что по крайней мере один из философов уже ест.

Это доказательство состоит из рассмотрения множества различных случаев, неформально описанных в терминах нашего конкретного примера. Рассмотрим другой способ доказательства: составление машинной программы для исследования поведения системы в поисках дедлока. В общем случае мы никогда не сможем узнать, достаточно ли проработала программа, чтобы гарантировать отсутствие тупиковых ситуаций. Но если система имеет конечное число состояний, как в нашем случае *ПАНСИОН*, достаточно рассмотреть только те протоколы, длина которых не превышает известной верхней границы числа состояний. Число состояний *(P || Q)* не превышает произведения числа состояний *P* и числа состояний *Q*. Так как каждый философ имеет шесть, а каждая вилка – три состояния, общее число состояний системы *ПАНСИОН* не превышает $6^5 \times 3^5$, или приблизительно 1,8 млн. Так как алфавит слуги содержится в алфавите *ПАНСИОН*, число состояний *НОВПАНСИОНа* не может превышать числа состояний *ПАНСИОНа*. Поскольку почти каждое состояние содержит два или более возможных события, то число протоколов, которые надо проверить, будет превышать два в степени 1,8 млн. Весьма маловероятно, что компьютер когда-нибудь сумеет исследовать все возможные случаи. Доказательство отсутствия тупиковых ситуаций даже для весьма простых конечных процессов, таким образом, по-прежнему будет входить в обязанности разработчика параллельных систем.

2.5.5. Бесконечный перехват

Помимо дедлока обедающего философа подстерегает другая опасность, а именно опасность бесконечного перехвата инициативы соседом. Предположим, что левая рука у сидящего философа весьма медлительна, в то время как его левый сосед в высшей степени проворен. Прежде чем философ дотянется до своей левой вилки, его левый сосед быстро вступает в дело, садится, хватает обе вилки и долго ест. Рано или поздно наевшись, он кладет обе вилки на стол и покидает свое место. Но стоит ему встать, как он снова ощущает голод, возвращается к столу, садится, мгновенно хватает обе вилки – и все это прежде, чем его медлительный, долгосидящий и изнывающий от голода правый сосед дотянется до их общей вилки. Поскольку такого рода цикл может повторяться неограниченно, может оказаться, что сидящий философ никогда не приступит к еде.

Похоже, что при строгом подходе эта проблема неразрешима, поскольку если каждый философ столь жаден и проворен, как было описано, то неизбежно кто-нибудь (или он, или его сосед) очень долгое время будет оставаться голодным. В такой ситуации не видно эффективного пути к достижению общего удовлетворения, кроме как приобрести достаточное количество вилок и побольше спагетти.

Однако, если все-таки важно гарантировать, чтобы сидящий философ смог в конце концов поесть, нужно изменить поведение слуги: проводив философа до его места, он ждет, пока философ не возьмет обе вилки, и только посла этого позволяет сесть его соседям.

Однако в отношении бесконечного перехвата остается более философская проблема. Предположим, что слуга испытывает иррациональную неприязнь к одному из философов и постоянно затягивает исполнение своей обязанности проводить его к столу, даже если философ к этому готов. Это ситуация, которую мы не можем описать в нашей концептуальной схеме, поскольку она неотличима от той, когда философу, требуется бесконечно длительное время, чтобы проголодаться? Таким образом, возникает проблема, аналогичная проблемам детальной синхронизации, которыми, как вы помните, мы сознательно решили пренебречь или, более точно, отложить на другие стадии проектирования и реализации. Таким образом, добиваться, чтобы любое желаемое и возможное событие обязательно наступило в пределах разумного интервала времени, становится задачей реализатора. Это аналогично требованию, предъявляемому реализатору обычного языка программирования высокого уровня, а именно: не создавать произвольных задержек в выполнении программы, хотя программист не имеет возможности ни обеспечить, ни даже описать это требование.

2.6. ПЕРЕИМЕНОВАНИЕ

В примере из предыдущего раздела мы имели дело с двумя группами процессов – философами и вилками; внутри каждой группы процессы вели себя очень похоже с той разницей, что события, в которых они участвовали, имели различные имена. В настоящем разделе мы описываем удобный способ задания группы процессов, обладающих сходным поведением. Пусть f – взаимно однозначная (инъективная) функция, отображающая алфавит процесса P во множество символов A:

$f : \alpha P \rightarrow A$

Определим f(P) как процесс, участвующий в событии f(c) всякий раз, когда P по определению участвует в событии c. Из определения следует, что

$\alpha f(P) = f(\alpha P)$

протоколы(f(P)) = {f(s) | s ∈ протоколы(P)}*

(Определение *f** см. в разд. 1.9.1.)

Примеры

X1. Как известно, с годами стоимость жизни растет. Функцию f, отражающую последствия инфляции, зададим с помощью уравнений

f(п2) = п10 f(бол) = бол

f(п1) = п5 f(мал) = мал

f(сд1) = сд5

Новый торговый автомат

НОВТАС = f(ТАС)

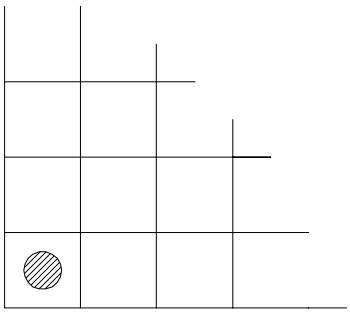
X2. Фишка ведет себя как $СТ_0$ (1.1.4 **X2**),только вместо вверх и вниз она совершает движения вправо и влево.

f(вверх) = вправо, f(вниз) = влево, f(вокруг) = вокруг

$ЛП_0 = f(СТ_0)$

В основном подобное переименование событий делается для. того, чтобы использовать полученные процессы в параллельной комбинации Друг с другом.

X3. Фишка двигается *вверх*, *вниз*, *влево* и *вправо* но бесконечному полю, ограниченному слева и снизу.



Движение начинается из левого нижнего угла поля. Только из этой клетки фишка может двигаться *вокруг*. Как и в примере 2.3 X3, вертикальные и горизонтальные перемещения можно промоделировать независимыми событиями различных процессов, однако движение *вокруг* требует одновременного участия обоих процессов.

$ЛПВН = ЛП_0 \,||\, СТ_0$.

X4. Мы хотим соединить два экземпляра процесса КОПИБИТ (1.1.3 **X7**) в цепь, чтобы каждый выходной бит первого процесса одновременно был входом для второго. Сначала нам придется переименовать события, использующиеся для внутреннего сообщения; введем поэтому два новых события, средн. 0 и средн. 1, и определим функции f и g для изменения выхода одного процесса и входа другого:

f(выв.0) = g(вв.0) = средн.0, f(выв.1) = g(вв.1) = средн.1

f(вв.0) = вв. 0, f(вв.1) = вв.1

g(выв.0) = выв.0, g(выв.1) = выв.1

Требуемый результат получаем как

ЦЕПЬ2 = f(КОПИБИТ) || g(КОПИБИТ)

Заметим, что по определению *f* и *g* каждый вывод 0 или 1 левым операндом и ввод тех же 0 и 1 правым операндом представляют собой единое событие *средн.*0 или *средн.*1. Так моделируется синхронизованный обмен двоичными цифрами по каналу, соединяющему два операнда, как показано на рис. 2.7.

Рис. 2.7

Левый операнд не выбирает значение, которое передается по соединяющему каналу, тогда как правый операнд готов и к событию *средн*.0, и к событию *средн*.1. Поэтому лишь процесс вывода в каждом случае определяет, которое из этих двух событий произойдет. Этот способ взаимодействия между параллельными процессами будет обобщен в гл. 4.

Заметим, что внутренние сообщения *средн*.0 и *средн*.1, оставаясь в алфавите составного процесса, доступны для наблюдения (и, возможно, контроля) со стороны его окружения. Иногда такие внутренние события желают не принимать во внимание или делать недоступными; в общем случае это может приводить к недетерминизму, и поэтому мы отложим этот вопрос до разд. 3.5.

X5. Мы хотим представить поведение логической переменной, использующейся в программе. Ее алфавит содержит события

*присв*0 присваивание переменной значения О,

*прис*в1 присваивание переменной значения 1,

*выб*0 взятие нулевого значения переменной,

*выб*1 взятие значения переменной, равного единице.

Поведение этой переменной удивительно похоже на поведение автомата с газировкой (пример 1.1.4 **X1**), и поэтому определим

ЛОГ = f(АГАЗ),

где определение *f* мы оставляем в качестве элементарного упражнения. Заметим, что значение нашей переменной не может быть считано прежде, чем произойдет первое присваивание этой переменной. Попытка взятия значения неопределенной переменной приведет к тупиковой ситуации – пожалуй, лучшему, что может произойти с неверной программой, ибо простейшая «посмертная» выдача точно укажет на эту ошибку. Древовидное представление *f(P)* можно построить по древовидному представлению *P*, применив функцию *f* к меткам на всех дугах. Так как функция *f* взаимно однозначна, это преобразование сохраняет структуру дерева и существенное

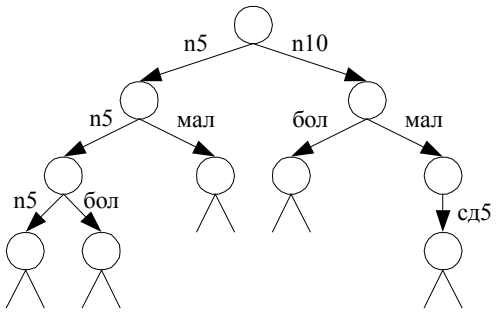


Рис. 2.8

свойство – различие всех меток на дугах, исходящих из одной вершины. Дерево для НОВТАС, к примеру, выглядит, как показано на рис. 2.8.

2.6.1. Законы

Переименование путем применения взаимно однозначной функции не меняет структуры поведения процесса. Это отражает тот факт, что применяемая функция дистрибутивна относительно всех остальных операций, о чем гласят приведенные ниже законы. Будем использовать следующие вспомогательные определения:

$f(B) = \{f(x) \mid x \in B\}$

f^{-1} функция, обратная к f

$f \circ g$ композиция функций f и g

f^* определена в разд. 1.9.1

(использование в законах f^{-1} является важной причиной нашего требования инъективности f),

После переименования *СТОП* по-прежнему не участвует ни в одном из событий своего нового алфавита:

L1. $f(СТОП_A) = СТОП_{f(A)}$

В конструкции выбора меняются символы из множества первоначального выбора, и аналогично изменяется последующее поведение]

L2. $f(x : B \rightarrow P(x)) = (y : f(B) \rightarrow f(P(f^{-1}(y))))$

Могут понадобиться разъяснения по поводу использования в правой части функции f^{-1}. Вспомним, что P – это функция, выдающая процесс в зависимости от выбора некоторого x из множества B. Но переменная y в правой части выбрана из множества $f(B)$. Соответствующим событием для P будет $f^{-1}(y)$ принадлежащее B (так как $y \in f(B)$). После этого события поведение P есть $P(f^{-1}(y))$, и ко всем действиям, которые совершает этот процесс, по-прежнему должна применяться функция f.

Переименование дистрибутивно относительно оператора параллельной композиции:

L3. $f(P \parallel Q) = f(P) \parallel f(Q)$

Дистрибутивность относительно оператора рекурсии осуществляется чуть сложнее; при этом соответственно изменяется алфавит:

L4. $f(\mu X : A.F(X)) = (\mu Y : f(A). f(F(f^{-1}(Y)))$

Опять может вызвать недоумение использование f^{-1} в правой части. Вспомним, что для того, чтобы рекурсивное определение в левой части было корректным, требуется, чтобы аргументом функции F был процесс с алфавитом A, а результатом – процесс с тем же алфавитом. В правой части переменная Y принимает значения процессов с алфавитом $f(A)$ и не может поэтому использоваться в качестве аргумента f, пока ее алфавит не будет заменен снова на A. Это делается применением обратной функции f^{-1}. Теперь $F(f^{-1}(Y))$ имеет алфавит A, и поэтому применение функции f изменит этот алфавит на $f(A)$, обеспечив таким образом справедливость рекурсии в правой части.

Композиция двух переименований определяется как композиция двух переименующих функций:

L5. $f(g(P)) = (f \circ g)(P)$

Протоколы процесса после переименования получаются простой заменой отдельных символов во всех протоколах исходного процесса:

L6. $протоколы(f(P)) = \{f^*(s) \mid s \in протоколы(P)\}$

Пояснение к следующему и последнему закону аналогично пояснению к **L6**.

L7. $f(P)/f^*(s) = f(P/s)$

2.6.2. Помеченные процессы

Переименование особенно полезно при создании групп сходных процессов, которые в режиме параллельной работы предоставляют некоторые идентичные услуги их общему окружению, но никак не взаимодействуют друг с другом. Это означает, что все они должны иметь различные и взаимно непересекающиеся алфавиты. Чтобы этого достичь, снабдим каждый процесс отдельным именем; каждое событие помеченного процесса помечено тем же именем. Помеченное событие – это пара $l.\ x$, где l – метка, а x – имя события.

Процесс P с меткой l обозначают $l : P$. Он участвует в событии $l.\ x$, когда по определению P участвует в x. Процесс $l : P$ задается функцией

$f_l(x) = l.\ x$ для всех x из P,

и пометкой $l : P = f_l(P)$.

Примеры

X1. Два работающих рядом торговых автомата

(лев : ТАП) || (прав : ТАП)

Алфавиты этих процессов не пересекаются, и каждое происходящее событие помечено именем того устройства, на котором оно происходит. Если перед их параллельной установкой устройства не получили имен, каждое событие будет требовать участия их обоих, и тогда пара машин будет неотличима от единственного устройства; это является следствием того, что (ТАП || ТАП)=ТАП.

Пометка позволяет использовать процессы наподобие локальных переменных в языке высокого уровня, описанных в том блоке программы, где они используются.

X2. Поведение логической переменной моделируется процессом ЛОГ (2.6 **X5**). Поведение блока программы представимо процессом ПОЛЬЗ. Этот процесс присваивает и считывает значения двух логических переменных b и c. Поэтому его алфавит αПОЛЬЗ состоит из таких событий, как

b. присв0 присвоить *b* значение 0

c. выб1 выбрать текущее значение *c*, когда оно равно 1

Процесс *ПОЛЬЗ* и обе его логические переменные работают параллельно:

b*: ЛОГ || c: ЛОГ || ПОЛЬЗ*

В программе *ПОЛЬЗ* можно задать следующие действия:

b: = ложь; P – посредством *(b. присв0 → P),*

b := ^c; P – посредством *(c.выб0 → b.присв1 → P | c.выб1 → b.присв0 → P)*

Заметим, что для того, чтобы выяснить текущее значение переменной, используется выбор между событиями *выб*0 и *выб*1; результат этого выбора соответствующим образом влияет на последующее поведение процесса *ПОЛЬЗ.*

В **X2** и последующих примерах было бы удобнее определять эффект одного присваивания, например:

b := ложь,

нежели пары команд

b:= ложь; P

где имеется явное упоминание остатка программы *P*. Это может быть достигнуто с помощью средств, описанных в гл. 5.

X3. Процессу *ПОЛЬЗ* требуются два счетчика – переменные *l* и *m.* Их начальные значения равны 0 и 3 соответственно. Процесс *ПОЛЬЗ* может увеличивать эти переменные с помощью событий *l.вверх* и *m.вверх* и уменьшать (когда они положительны) с помощью *l.вниз* и *m.вниз*. Проверка на нуль осуществляется событиями *l.вокруг* и *m.вокруг*. Отсюда следует, что мы можем использовать процесс *СТ* (1.1.4 **X2**), пометив его соответственно *l* и *m*:

(l : CT$_0$ *|| m : CE*$_3$ *|| ПОЛЬЗ)*

В ходе процесса *ПОЛЬЗ* могут быть осуществлены следующие действия (выраженные в обычных обозначениях):

(m:=m + 1; P) – посредством

(m.вверх → P), **if** *l* = 0 **then** *P* **else** *Q* – применением *(l. вокруг → P | l. вниз → l. вверх → Q)*

Обратите внимание, как работает проверка на нуль: *l.вниз* пытается уменьшить счетчик на единицу, и одновременно производится попытка выполнить *l. вокруг*. Счетчик выбирает одно из этих событий: если его величина равна нулю – событие *l. вокруг*, иначе – другую альтернативу. Но в последнем случае величина счетчика уменьшается на единицу и поэтому должна быть немедленно восстановлена с помощью *l. вверх*. В последующем примере восстановление начального значения более трудоемко.

Конструкция *(m:=m + 1; P)* реализуется рекурсивно определенным процессом *СЛОЖ*:

СЛОЖ = ВНИЗ$_0$

где *ВНИЗ*$_i$ *= (i. вниз → ВНИЗ*$_{i+1}$ *| вокруг → ВВЕРХ*$_i$*)*

ВВЕРХ$_0$ *= P*

а *ВВЕРХ*$_{i+1}$*, = l.вверх → m.вверх → ВВЕРХ*$_i$

Процесс *ВНИЗ*$_i$ выясняет начальное значение *l*, уменьшая его до нуля. Затем процесс *ВВЕРХ*$_i$ прибавляет полученное значение к *m* и к *l* восстанавливая тем самым исходное значение *l* и прибавляя это значение к *m*.

Переменную-массив можно представить набором параллельных процессов, каждый из которых помечен его индексом в массиве.

X4. Задача процесса *НД* состоит в том, чтобы регистрировать, происходило ли ранее событие *b* или нет. При первом наступлении *b* процесс отвечает *нет*, а при всех последующих – *да*.

αНД = {b, нет, да}

НД = b→ нет → μX.(b → да →X)

Массив из этих процессов можно использовать для имитации поведения цифрового множества:

МНОЖ3 = (0 : НД) || (1: НД) || (2: НД) || (3: НД)

Перед использованием весь массив также может быть помечен:

m : МНОЖ3 || ПОЛЬЗ

Каждое событие из алфавита *α(m : МНОЖ3)* представляет собой тройку, например *m. 2. b.* В ходе процесса *ПОЛЬЗ* эффект конструкции

if *2 ∈ m* **then** *P* **else** *(m:= m ∪ {2}; Q)*

может быть достигнут посредством

m. 2. b → (m. 2. да → P | m. 2. нет → Q)

2.6.3. Реализация

Для реализации переименования в общем случае требуется знать функцию g – обратную к

переименующей функции *f*. Кроме того, необходимо обеспечить, чтобы *g* выдавала специальный символ "*BLEEP* в случае, когда ее аргумент выходит за область значений *f*. Реализация основана на законе 2.6.1 **L4**.

переименование(g, P) = λx. **if** *g(x) = "BLEEP* **then** *"BLEEP*
else if *P(g(x)) = "BLEEP* **then** *"BLEEP*
else *переименование(g, P(g(x)))*

Пометка процессов реализуется проще. Составное событие *l.x* представляется парой атомов *cons("l, "x)*. Помеченный процесс *(l : P)* реализуется функцией

пометка(l, P) = λy. **if** *null(y)* $\vee$ *atom(y)* **then** *"BLEEP*
else if *car(y)* $\neq$ *l* **then** *"BLEEP*
else if *P(cdr(y)) = "BLEEP* **then** *"BLEEP*
else *пометка (l, P(cdr(y)))*

2.6.4. Множественная пометка

Определение пометки можно расширить, позволив помечать каждое событие любой меткой *l* из некоторого множества *L*. Если *P*– процесс, определим *(L:P)* как процесс, ведущий себя в точности как *P* с той разницей, что он участвует в событии *l.c (*где $l \in L$, а $c \in \alpha P$*),* если по определению *P* участвует в *c*. Выбор метки *l* каждый раз независимо осуществляется окружением процесса *(L : P)*.

Пример

X1. Лакей – это младший слуга, который имеет одного хозяина, провожает его к столу и из–за стола и прислуживает ему, пока тот ест:

αЛАКЕЙ = {садится, встает}

ЛАКЕЙ = (садится → встает → ЛАКЕЙ)

Чтобы научить лакея обслуживать всех пятерых философов {но только по очереди), определим

L = {0, 1, 2, 3, 4}

ОБЩИЙ ЛАКЕЙ = (L : ЛАКЕЙ)

Общего лакея можно нанимать на период отпуска слуги (2.5.3) для предохранения обедающих философов от тупиковой ситуации. Конечно, в течение этого времени философам придется поголодать, ибо находиться за столом они смогут только по очереди.

Если множество *L* содержит более одной метки, древовидное представление *L: P* похоже на древовидное представление *P*, однако оно гораздо гуще в том смысле, что из каждой вершины исходит гораздо больше дуг. Дерево процесса *ЛАКЕЙ*, например, представляет собой ствол без ветвей(рис. 2.9).

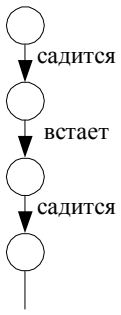


Рис. 2.9

Представлением же процесса {0, 1}*:ЛАКЕЙ* будет полное двоичное дерево (рис. 2.10). Дерево для процесса *ОБЩИЙ ЛАКЕЙ* будет еще гуще.

В общем случае множественную пометку можно использовать для распределения услуг одного процесса между некоторым числом других помеченных процессов при условии, что множество меток известно заранее. Более полно эта техника развивается в гл. 6.

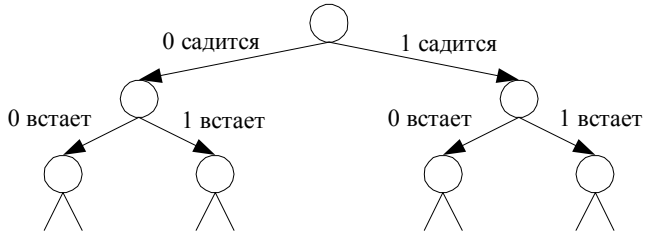


Рис. 2.10

2.7. СПЕЦИФИКАЦИИ

Пусть P и Q – процессы, которые мы хотим исполнять параллельно, и предположим, что мы доказали, что P уд $S(пр)$, а Q уд $T(пр)$. Пусть $пр$ – протокол процесса $(P \parallel Q)$. Из закона 2.3.3 **L1** следует, что $(пр \upharpoonright \alpha P)$ является протоколом P и, следовательно, удовлетворяет S, т. е. $S(пр \upharpoonright \alpha P)$. Аналогично $(пр \upharpoonright \alpha Q)$ является протоколом Q и, значит, $T(пр \upharpoonright \alpha Q)$. Это справедливо для каждого протокола $(P \parallel Q)$. Следовательно, можно заключить, что

$(P \parallel Q)$ **уд** $(S(пр \upharpoonright \alpha P) \;\&\; T(пр \upharpoonright \alpha Q))$

Как следствие этих нестрогих рассуждений можно сформулировать закон

L1. Если P **уд** $S(пр)$, а Q **уд** $T(пр)$, то $(P \parallel Q)$ **уд** $(S(пр \upharpoonright \alpha P) \;\&\; T(пр \upharpoonright \alpha Q))$.

Пример

X1. (см. 2.3.1 **X1**)

Пусть $\alpha P = \{a, c\}$, $\alpha Q = \{b, c\}$, $P = (a \to c \to P)$, $Q = (c \to b \to Q)$. Мы хотим доказать, что

$(P \parallel Q)$ **уд** $(0 \le пр \downarrow a - пр \downarrow b \le 2)$.

Чтобы показать, что

P **уд** $(0 \le пр \downarrow a - пр \downarrow c \le 1)$, а Q **уд** $(0 \le пр \downarrow c - пр \downarrow b \le 1)$,

можно очевидным образом использовать доказательство из примера 1.10.2 **X1** Отсюда по **L1** следует, что

$(P \parallel Q)$ **уд** $(0 \le (пр \upharpoonright \alpha P) \downarrow a - (пр \upharpoonright \alpha P) \downarrow \; c \le 1 \;\&\; 0 \le (пр \upharpoonright \alpha Q) \downarrow c - (пр \upharpoonright \alpha Q) \downarrow b \le 1)$

$\Rightarrow 0 \le пр \downarrow a - пр \downarrow b \le 2$, так как $(пр \upharpoonright A) \downarrow a = пр \downarrow a$, если $a \in A$

Поскольку, согласно законам для отношения **уд**, *СТОП* удовлетворяет любой выполнимой спецификации, рассуждениями, основанными на этих законах, доказать отсутствие дедлока нельзя. В разд. 3.7 будут даны более сильные законы. Пока же единственным способом исключить риск остановки является тщательное доказательство, как и в разд. 2.5.4. Другой способ – это показать, что процесс, определенный с помощью оператора параллельной комбинации, эквивалентен бесконечному процессу, определенному без этого оператора, как делалось в примере 2.3.1 **X1**. Такие доказательства, однако, требуют очень трудоемких алгебраических преобразований. Где это возможно, следует прибегать к помощи какого-либо общего закона, как, например,

L2. Если процессы P и Q бесконечны, а $(\alpha P \cap \alpha Q)$ содержит не более одного события, то процесс $(P \parallel Q)$ бесконечен.

Пример

X2. Процесс $(P \parallel Q)$, определенный в **X1**, никогда не останавливается, потому что

$\alpha P \cap \alpha Q = \{c\}$.

Правило доказательства для переименования имеет вид

L3. Если P **уд** $S(пр)$, то $f(P)$ **уд** $S(f^{-1}*(пр))$

Использование функции $f^{-1}*$ в последующем члене этого закона может потребовать разъяснений. Пусть $пр$ – протокол $f(P)$. Тогда $f^{-1}*(пр)$ – протокол P. Предыдущий член **L3** утверждает, что всякий протокол P удовлетворяет S. Следовательно, $f^{-1}*(пр)$ удовлетворяет S, что в точности соответствует утверждению последующего члена **L3**,

2.8. МАТЕМАТИЧЕСКАЯ ТЕОРИЯ ДЕТЕРМИНИРОВАННЫХ ПРОЦЕССОВ

При описании процессов мы сформулировали множество законов и время от времени использовали их в доказательствах. Если мы и давали этим законам некоторое обоснование, то только в форме нестрогих разъяснений, почему мы предполагаем или хотим, чтобы эти законы выполнялись. Для читателя, обладающего чутьем математика-прикладника или инженера, этого может быть достаточно. Но возникает вопрос: а действительно ли справедливы эти законы? Является ли их набор хотя бы непротиворечивым? Должно ли их быть больше, или они полны в том смысле, что с их помощью можно доказать любой истинный факт о процессах? Можно ли обойтись меньшим числом более простых законов? Ответы на эти вопросы следует искать в более глубоком математическом исследовании.

2.8.1. Основные определения

При создании математической модели некоторой физической системы хороша стратегия, при которой определение основных понятий ведется в терминах свойств и признаков, поддающихся прямому или косвенному наблюдению или измерению. Для детерминированного процесса P нам известны два таких свойства:

αP множество событий, в которых процесс может участвовать;

протоколы(P) множество всех последовательностей событий, в которых процесс действительно может участвовать если потребуется.

На эти два множества налагаются условия, которые уже были сформулированы в законах 1.8.1 **L6, L7, L8.** Рассмотрим теперь произвольную пару множеств *(A, S),* удовлетворяющую этим трем законам. По этой паре можно единственным образом построить процесс P с множеством протоколов S согласно следующим определениям:

Пусть $P^0 = \{x \mid \langle x\rangle \in S\}$, а $P(x)$ – процесс с множеством протоколов $\{t \mid \langle x\rangle \hat{}\, t \in S\}$ для всех x из P^0

Тогда $\alpha P = A$, а $P = (x : P^0 \rightarrow P(x))$.

Кроме того, уравнения

$A = \alpha P$

$S =$ *протоколы*$(x : P0 \rightarrow P(x))$

позволяют по P восстановить пару (A,S). Таким образом, между каждым процессом P и парой множеств $(\alpha P,$ *протоколы(P))* существует взаимно однозначное соответствие. В математике такое использование одного понятия для определения другого служит достаточным основанием для их последующего отождествления.

D0. Детерминированный процесс – это пара (A,S), где A – произвольное множество символов, а S – любое подмножество A, удовлетворяющее двум условиям:

C0. $\langle\rangle \in S$

C1. $\forall\, s,t.s\hat{}\,t \in S \Rightarrow s \in S$

Простейшим примером процесса, отвечающего этому определению, служит *СТОП* – процесс, который ничего не делает:

D1. $СТОП_A = (A,\{\langle\rangle\})$

В другом крайнем случае – это процесс, всегда готовый выполнить любое действие:

D2. $ИСП_A = (A,A^*)$

Теперь мы можем дать формальные определения различных операций над процессами, описав, как по алфавиту и протоколам операндов строятся алфавит и протоколы результата.

D3. $(x:B \rightarrow (A,S(x))) = (A,\{\langle\rangle\} \cup \{\langle x\rangle \hat{}\, s \mid x \in B \,\&\, s \in S(x)\})$ при условии, что $B \subseteq A$;

D4. $(A,S)/s = (A,\{t \mid (s\hat{}\,t) \in S\})$ при условии, что $s \in S$

D5. $\mu X: A.F(x) = (A, \bigcup_{n\geq 0}$ *протоколы* $(F^n(СТОП_A)))$ при условии, что выражение F – предваренное;

D6. $(A,S) \parallel (B,T) = (A \cup B, \{s \mid s \in (A \cup B)^* \,\&\, (s \upharpoonright A) \in S \,\&\, (s \upharpoonright B) \in T\})$

D7. $f(A,S) = (f(A), \{f^*(s) \mid s \in S\}$ при условии, что f взаимно однозначна.

Разумеется, надо доказать, что правые части этих определений действительно являются процессами, т.е. удовлетворяют условиям **C0** и **C1** в определении **D0**. К счастью, сделать это несложно.

В гл. 3 станет очевидно, что **D0** – не вполне достаточное определение понятия процесса, поскольку в нем никак не представлена возможность недетерминированного поведения. следовательно, нам потребуется более общее и более сложное определение. Все законы для недетерминированных процессов справедливы также и для детерминированных. Но детерминированные процессы

подчиняются, кроме того, некоторым дополнительным законам, например $P \parallel P = P$. Во избежание путаницы такие законы мы в книге не помечаем; таким образом, все помеченные законы применимы как к детерминированным, так и к недетерминированным процессам.

2.8.2. Теория неподвижной точки

Целью этого раздела является доказательство в общих чертах основополагающей теоремы о рекурсии, т.е. о том, что рекурсивно определенный процесс (2.8.1. **D5**) в действительности является решением соответствующего рекурсивного уравнения

$\mu X.F(X)=F(\mu X.F(X))$

Наш подход будет основан на теории неподвижной точки по Скотту.

Прежде всего мы должны ввести на множестве процессов отношение порядка:

D1. $(A,S)\sqsubseteq(B,T)=(A=B \;\&\; S\subseteq T)$

При таком упорядочении два процесса сравнимы, если они имеют общий алфавит, и один из них может делать все то же, что и другой, и, возможно, больше. Это отношение является частичным порядком в том смысле, что

L1. $P \sqsubseteq P$

L2. $P \sqsubseteq Q \;\&\; Q \sqsubseteq P \Rightarrow P = Q$

L3. $P \sqsubseteq Q \;\&\; Q \sqsubseteq R \Rightarrow P = R$

Цепью в частичном упорядочении называется бесконечная последовательность элементов $\{P_0, P_1, P2,\ldots\}$. такая, что $P_i \sqsubseteq P_{i+1}$ для всех *i. Предел* (наименьшую верхнюю грань) такой цепи определим как

$$\bigsqcup_{i\geq 0} Pi = (\alpha P_0, \bigcup_{i\geq 0} протоколы(P_i))$$

В дальнейшем мы будем применять предельный оператор $\sqcup$ только к тем последовательностям процессов, которые образуют цепь.

Будем говорить, что частичный порядок *полный*, если в нем имеется наименьший элемент, а все цепи имеют наименьшую верхнюю грань. Множество всех процессов над данным алфавитом A образует полный частичный порядок (п. ч. п.), так как оно удовлетворяет следующим законам:

L4. $СТОП_A \sqsubseteq P$ при условии, что $\alpha P = A$

L5. $P_i \sqsubseteq \bigsqcup_{i\geq 0} P_i$

L6. $(\forall i \geq 0.\ P_i \sqsubseteq Q) \Rightarrow (P_i) \sqsubseteq Q$

Кроме того, в терминах предела можно переформулировать и определение μ-оператора {2.8.1 **D5**);

L7. $\mu X : A,\ F(X) = \bigsqcup_{i\geq 0} F^i(СТОП_A)$

Говорят, что функция F из одного п. ч. п. в другой (или в тот же) *непрерывна*, если она дистрибутивна относительно пределов всех цепей, т. е.

$F(\bigsqcup_{i\geq 0} P_i) = \bigsqcup_{i\geq 0} F(P_i)$, если $\{P_i \mid i \geq 0\}$ образует цепь.

(Все непрерывные функции монотонны в том смысле, что $P \sqsubseteq Q \Rightarrow F(P) \sqsubseteq F(Q)$ для всех P и Q, и поэтому правая часть предыдущего равенства является пределом возрастающей цепи.) По определению функция G от нескольких аргументов непрерывна, если она непрерывна по каждому аргументу в отдельности, например:

$G(\bigsqcup_{i\geq 0} Pi, Q) = \bigsqcup_{i\geq 0} G(Pi, Q)$ для всех Q и

$G(Q, (\bigsqcup_{i\geq 0} Pi)) = \bigsqcup_{i\geq 0} G(Q, Pi)$ для всех Q.

Композиция непрерывных функций также непрерывна; и, конечно же, любое выражение, полученное применением любого числа непрерывных функций к любому числу и комбинации переменных, непрерывно по каждой из этих переменных. Например, если G, F и H непрерывны, то $G(F(X), H(X, Y))$ непрерывна по X, т. е.

$G(F(\bigsqcup_{i\geq 0} Pi), H((\bigsqcup_{i\geq 0} Pi), Y)) = \bigsqcup_{i\geq 0} G(F(Pi), H(Pi, Y))$ для всех Y.

Все определенные в **D3 – D7** операции (кроме /) непрерывны в вышеописанном смысле:

L8. $(x : B \to (\coprod_{i\geq 0} P_i(x))) = \coprod_{i\geq 0} (x : B \to P_i(x))$

L9. $\mu X : A.\ F(X, (\coprod_{i\geq 0} P_i)) = \coprod_{i\geq 0} \mu X : A.\ F(X, P_i)$ при условии, что F непрерывна

L10. $(\coprod_{i\geq 0} P_i) \parallel Q = Q \parallel (\coprod_{i\geq 0} P_i) = \coprod_{i\geq 0} (Q \parallel P_i)$

L11. $f(\coprod_{i\geq 0} P_i) = \coprod_{i\geq 0} f(P_i)$

Таким образом, если выражение $F(X)$ построено в терминах только этих операций, оно будет непрерывным по X. Теперь можно доказать основную теорему о неподвижной точке:

$$
\begin{aligned}
F(\mu X : A.\ F(X)) &= F(\coprod_{i\geq 0} (F^i(СТОП_A)) && \text{по определению } \mu \\
&= \coprod_{i\geq 0} F(F^i(СТОП_A)) && \text{непрерывность } F \\
&= \coprod_{i\geq 0} F^i(СТОП_A) && \text{по определению } F^{i+1} \\
&= \coprod_{i\geq 0} F^i(СТОП_A)\ F \\
&= \mu X : A.\ F(X) && \text{по определению } \mu
\end{aligned}
$$

Это доказательство основано на том, что F непрерывна. Предваренность F необходима лишь для доказательства единственности решения.

2.8.3. Единственность решения

В этом разделе мы несколько формализуем наши рассуждения из разд. 1.1.2, показавшие, что уравнение, задающее процесс с помощью предваренной рекурсии, имеет единственное решение. При этом нам удастся сформулировать более общие условия единственности таких решений. Для простоты мы будем рассматривать процессы, заданные только одним рекурсивным уравнением; эти рассуждения можно легко обобщить на случай систем взаимно рекурсивных уравнений.

Если P – процесс, а n – натуральное число, то определим $(P \upharpoonright n)$ как процесс, который ведет себя как P в течение первых n событий, а затем останавливается; более строго:

$(A, S)\ n = (F, \{s \mid s \in S \ \&\ \#s \leq n\})$

Отсюда следует, что

L1. $P \upharpoonright 0 = СТОП$

L2. $P \upharpoonright n \sqsubseteq P \upharpoonright (n + 1) \sqsubseteq P$

L3. $P = \coprod_{n\geq 0} P \upharpoonright n$

L4. $\coprod_{n\geq 0} P_n = \coprod_{n\geq 0} (P_n \upharpoonright n)$

Пусть F – монотонная функция из множества процессов во множество процессов. Будем говорить, что F конструктивна, если

$F(X) \upharpoonright (n+1) = F(X \upharpoonright n) \upharpoonright (n + 1)$ для всех X

Это означает, что поведение $F(X)$ на $n + 1$ первом шаге определяется поведением X только на n первых шагах; таким образом, если $S \neq <>$, то

$s \in протоколы(F(X))\ \ s \in протоколы(F(X \upharpoonright (\#s - 1)))$

Простейшим примером конструктивной функции служит префиксация, поскольку

$(c \to P) \upharpoonright (n + 1) = (c \to (P \upharpoonright n)) \upharpoonright (n + 1)$

Обобщенный выбор также конструктивен:

$(x : B \to P(x)) \upharpoonright (n + 1) = (x : B \to (P(x) \upharpoonright n)) \upharpoonright (n + 1)$

Тождественная функция I не конструктивна, поскольку

$$
\begin{aligned}
I(c \to P) \upharpoonright 1 &= c \to СТОП \\
&\neq СТОП \\
&= I((c \to P) \upharpoonright 0) \upharpoonright 1
\end{aligned}
$$

Теперь можно сформулировать основную теорему.

L5. Пусть *F* – конструктивная функция. Тогда уравнение *X = F(X)* имеет единственное решение для *X*.

Дозательство. Пусть *X* – произвольное решение. Сначала докажем по индукции лемму о том, что

X ↾ n = Fn(СТОП) ↾ n.

Основание индукции. *X ↾ 0 = СТОП* = 0 = *СТОП ↾ 0= F^0(СТОП) ↾* 0

Шаг индукции.

X ↾ (n + 1) = F(X) ↾ (n + 1)	так как *X = F(X)*
= F(X ↾ n) ↾ (n + 1)	F конструктивна
= F(F^n(СТОП) ↾n) ↾ (n + 1)	предположение индукции
= F(F^n(СТОП)) ↾ (n + 1)	*F* конструктивна
= F^{n+1}(СТОП) ↾(n+ 1)	по определению F^n

Теперь вернемся к основной теореме.

$X = \coprod_{i\geq 0} (X \upharpoonright n)$	**L3**
$= \coprod_{i\geq 0} F^n(СТОП) \upharpoonright n$	согласно лемме
$= \coprod_{i\geq 0} F^n(СТОП)$	**L4**
$= \mu X.\ F(X)$	2.8.2 **L7**

Таким образом, все решения уравнения X = F(X) совпадают и равны *μX. F(X)* другими словами, μX.F(X) является единственным решением уравнения.

Полезное значение этой теоремы сильно возрастет, если мы научимся четко определять, какие функции являются конструктивными, а какие – нет. Назовем функцию G недеструктивной, если она удовлетворяет условию

G(P) ↾ n = G(P ↾ n) ↾ n для всех n и P.

Переименование в этом смысле недеструктивно, поскольку

f(P) ↾ n = f(P ↾ n) ↾ n

Это же справедливо и для тождественной функции. Любая монотонная конструктивная функция является также и недеструктивной. Операция *после*, однако, деструктивна, поскольку

((c → c → СГОЯ) /<c>) ↾ 1 = c → СТОП
≠ СТОП
= (c → СТОП) / <c>
= (((c → c → СТОП) ↾ 1) / <c>) ↾ 1

Любая композиция недеструктивных функций (*G* и *H*) также недеструктивна, потому что

G(H(P)) ↾ n = G(H(P) ↾ n) ↾ n = G(H(P ↾ n) ↾ n) ↾ n = G(H(P ↾ n) ↾ n

Что еще важнее, любая композиция конструктивной функции с недеструктивными функциями является конструктивной. Поэтому если *F, C, ..., H* – недеструктивные функции и хотя бы одна из них конструктивна, то *F(G(...(H(X))...))* – конструктивная функция от *X*.

Приведенные рассуждения легко распространить на функции более чем одного аргумента. Так, например, параллельная композиция недеструктивна (по обоим аргументам), потому что

(P || Q) ↾ n = ((P ↾ n) || (Q ↾ n)) ↾ n

Пусть *E* – выражение, содержащее в качестве переменной процесс *P*. Говорят, что *E* конструктивно по *X*, если к каждому вхождению *X* в j5 применяется некоторая конструктивная функция и не применяется никакая деструктивная функция. Так, следующее выражение является конструктивным по *X:*

(c → X | d F(X || P) | e → (f(x) || Q)) || ((d → X) || R)

Важным следствием этого определения является то, что теперь конструктивность можно синтаксически определить с помощью следующих условий предваренности:

D0. Будем говорить, что выражение сохраняет предваренность, если оно построено только с помощью операторов параллелизма, переименования и обобщенного выбора.

D1. Будем говорить, что выражение, не содержащее *X*, предварено по *X*.

D2. Обобщенный выбор *(x: B → P(X, x))* предварен по *X*, если *P(X, x)* сохраняет предваренность для всех *x*.

D3. Переименование *fP(X))* предварено по *X*, если *P(X)* предварено по *X*.

D4. Параллельная система *P(X)* || *Q(X)* предварена по *X*, если как *P(X),* так и *Q(X*) предварены по *X*.

Окончательно мы заключаем, что

L6. Если *E* предварено по *X*, то уравнение *X* = *E* имеет единственное решение.