Введение в Пи-Исчисление

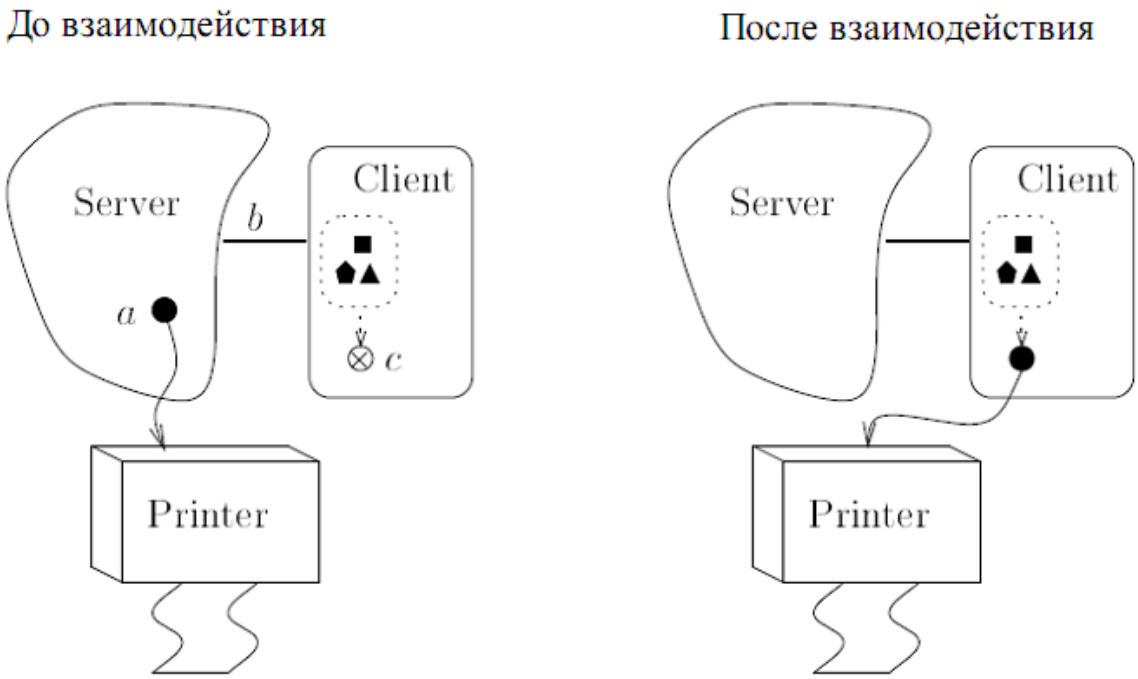
Chapter to appear in Handbook of Process Algebra, ed. Bergstra, Ponse and Smolka,
Elsevier
Joachim Parrow,
Dep. Teleinformatics,
Royal Institute of Technology, Stockholm

Пи-исчисление является алгеброй процессов, где последние взаимодействуют, посылая каналы связи друг другу. Эта статья представляет собой обзор и введение в основную теорию. Рассматривается синтаксис, семантика, эквивалентность аксиоматизация самых распространенных типов.

1. Введение

Пи-исчисление - математическая модель процессов, взаимосвязи которых изменяются, параллельно их взаимодействию. Основной вычислительный шаг - передача канала связи между двумя процессами; после чего получатель может использовать канал для дальнейшего взаимодействия с другими участвующими сторонами. Именно эта особенность исчисления делает его крайне удобным для моделирования систем, где доступные ресурсы изменяются в течение времени. Это также обеспечивает значительную выразительную мощность, так как понятия доступ и ресурс лежат в основе большей части теории параллельного вычисления, аналогично тому как более абстрактное и математическое понятие функции лежит в основе функционального вычисления. Это введение в пи-исчисление предназначено для читателей имеющих представление об общих принципах алгебры процессов и желающих изучить основные принципы исчисления и его самые общие и устойчивые типы.

Сначала рассмотрим пример. Предположим, что сервер управляет доступом к принтеру и клиент желает использовать его. В исходном состоянии только сам сервер имеет доступ к принтеру, что представлено в виде канала связи a. После взаимодействия с клиентом по другому каналу b, этот доступ к принтеру был передан.



В пи-исчислении это выражается следующим образом: сервер, который посылает a по b задается $\overline{b}\,a.S$; клиент, который получает некоторый канал по b и потом использует его, чтобы послать данные задается следующим образом $b(c).\overline{c}\,d.P$. Взаимодействие, изображенное выше, выражается следующей формулой

$$\overline{\boldsymbol{b}}\,\boldsymbol{a}.\boldsymbol{S} \mid \boldsymbol{b}(\boldsymbol{c}).\overline{\boldsymbol{c}}\,\boldsymbol{d}.\boldsymbol{P} \xrightarrow{\tau} \boldsymbol{S} \mid \overline{\boldsymbol{a}}\,\boldsymbol{d}.\boldsymbol{P}$$

Мы здесь видим, что a играет две различные роли. При взаимодействии сервера с клиентом это объект, переданный от одного другому. При дальнейшем взаимодействии клиента и принтера это имя канала связи. Идея, что имена каналов принадлежат той же категории что и переданные объекты, является одним из краеугольных камней исчисле-

ния и одним из принципиальных отличий от других типов алгебр процессов. В примере *a; b; c; d;* просто имена, которые интуитивно выражают права доступа : *a* имеет доступ к принтеру, *b* - к серверу, *d* - к каким-то данным, *c*- место где храниться доступ, полученный по *a.* Если *a* единственный способ доступа к принтеру, тогда можем сказать что принтер перешел к клиенту, так как после взаимодействия ничто не имеет доступа к принтеру. По этой причине пи-исчисление назвали исчислением мобильных процессов. Но исчисление более общее, чем здесь рассмотрено. У принтера может быть много каналов, по которым он может выполнять разные вещи и сервер может посылать эти каналы разным клиентам, чтобы те устанавливали различные возможности доступа к общему ресурсу.

На первый взгляд может показаться, что пи-исчисление - специализированная форма алгебры процессов передачи по значению, где значения - каналы. При этом сопоставлении исчисление может выглядеть довольно "жалким", так как в нем нет никаких типов данных и функций, определяемых для имен; передаваемые объекты простые атомарные единицы без какой-либо внутренней структуры. Причина того, что, несмотря на все это, пи-исчисление считается более выразительным, заключается в том, что оно допускает переходящие локальные области. Этот важный пункт заслуживает объяснения.

Большинство видов алгебр процессов имеют возможность объявить канал связи локальным для ряда процессов. Например в CSS тот факт, что P и Q совместно используют частный порт *a* записывается (P|Q)\a , где оператор *a* называется ограничением на *a*. Смысл в том, что никакой другой процесс не может использовать локальный канал a, так как если бы он был бы именем, отличным от всех остальных имен во всех процессах.

В пи-исчислении это ограничение записывается как (νa)(P|Q). Это похоже на то, что никакой другой процесс не может непосредственно использовать a как ссылку на P или Q. Различие в том, что имя a является также и передаваемым объектом, и как таковой может быть передан процессами P или Q другому процессу, который потом может использовать этот ограниченный канал. Возвращаясь к рассмотренному выше примеру предположим что a это локальный канал между сервером и принтером. Представив принтер как R, это можно записать следующим образом $(\nu a)(\bar{b}\,a.S|R)$. При этом сервер все еще может свободно посылать a через b клиенту. Результатом будет частный канал, разделенный между тремя процессами, но все еще имеющий имя, отличное от любого другого имени в любом другом процессе, и, следовательно переход может быть записан как

$$(\boldsymbol{\nu a})(\overline{\boldsymbol{b}}\,\boldsymbol{a}.\boldsymbol{S} \mid \boldsymbol{R}) \mid \boldsymbol{b}(\boldsymbol{c}).\overline{\boldsymbol{c}}\,\boldsymbol{d}.\boldsymbol{P} \xrightarrow{\boldsymbol{\tau}} (\boldsymbol{\nu a})(\boldsymbol{S} \mid \boldsymbol{R} \mid \overline{\boldsymbol{a}}\,\boldsymbol{d}.\boldsymbol{P})$$

Таким образом, хотя и передаваемые объекты простые атомарные вещи, они тоже могут быть объявлены локальными с определенной областью, и именно таким образом исчисление переступает пределы обычной алгебры процессов передачи по значению. Это также является основным источником трудностей в разработке теории, потому что область объекта, как представлено операндами его ограничения, должна перемещаться с объектом, когда тот передается между процессами.

Пи-исчисление далеко от единой хорошо определенной части работы. Центральная концепция – процессо алгебраическое определение передачи каналов, было расширено в нескольких направлениях чтобы вмещать специальные приложения или определять цели различных семантик. Быстрое увеличение - конечно хороший знак для любой научной области, несмотря на то, что это создает проблемы тем , кто хочет получить быстрый обзор. Возможно, некоторые читатели, для которых пи-исчисление в новинку, будут удовлетворены компактным представлением одной версии, в то время как другие могут быть заинтересованы спектром разновидностей.

Эта статья стремиться удовлетворить эти нужды. Четно пронумерованные разделы раскрывают отдельную сторону исчисления. Раздел 2 представляет синтаксис и приводит некоторые маленькие примеры использования. В Разделе 4 мы переходим к семантике в

ее наиболее стандартной форме как система помеченного прехода (Labelled Transition System). В Разделе 6 мы рассматриваем одно из основных определений двойного моделирования (Bisimulation) и конгруэнтность, которое оно вызывает, в Разделе 8 мы смотрим на их аксиоматизации через синтаксические равенства агентов. Эти разделы не зависят от разделов с нечетными номерами и могут быть рассмотрены как основной курс исчисления. Там будут полные определения и формулировки центральных результатов и наброски, которые объясняют идеи и структуру доказательств.

Каждый раздел с нечетным номером представляет изменения предыдущего материала. Таким образом, в Разделе 3 мы изучаем различные версии исчисления, такие как эффект изменения операторов, асинхронное, полиадическое, исчисление высокого порядка. Раздел 5 рассматривает альтернативные пути определения семантики с различными версиями помеченных и непомеченных переходов. Раздел 7 определяет другие общие эквивалентности двойного моделирования (пи-исчисление, как и любая алгебра процессов, имеет большое разнообразие эквивалентностей, но в этой работе мы концентрируемся на специфических аспектах пи-исчисления), и их аксиоматизации обработаны в Разделе 9. В этих разделах мы не всегда получаем полную формальную часть, но мы надеемся, что объяснений достаточно, чтобы читатель получил понимание основных идей . Наконец, Раздел 10 содержит ссылки на другие работы. Мы даем краткое изложение того, как исчисление развивалось и упоминаем другие обзоры и введения. Мы также указываем на источники материалов рассмотренных в этой работе.

Нужно подчеркнуть, что есть некоторые аспекты пи-исчисления, которые мы не рассмотрели вообще, такие как модальные логики, алгоритмы анализа, реализации и пути использования исчисления для моделирования параллельных систем и языков. Также различные виды могут быть объединены разными способами, давая начало большому разнообразию исчислений. Я надеюсь, что после этого введения читатель может исследовать поле с некоторой уверенностью.

2. Пи-исчисление

Мы начинаем с последовательности определений и соглашений. Читатель, который дойдет до Раздела 2.3, будет вознагражден маленькими, но информативными примерами.

2.1. Основные определения

Представим потенциально бесконечное множество имен N, элементы которого обозначаются через *a,b, ..., z,* которые будут функционировать как все коммуникационные порты, переменные и значения данных, и ряд (агент) идентификаторов, которых обозначим через A, каждый с фиксированным неотрицательным числом операндов. Агенты, обозначаемые через P, Q, ... определены в Таблице 1. Из этой таблицы видим, что агенты могут иметь следующие формы:

1. Пустой агент 0, который не может выполнить действия.
2. Префикс Вывода (Output Prefix) $\bar{a}x.P$. Смысл в том, что имя x послано вдоль имени a, и после этого агент продолжается как P. Так можно представить $\bar{a}$ как порт вывода и x как величину, посланную из этого порта.

3. Префикс Ввода (Input Prefix) $\boldsymbol{ax.P}$. Смысл в том, что имя получено вдоль имени a, и x носит значение полученного имени. После ввода агент продолжится как P, но с недавно полученным именем, которое заменяет x. Так a можно представить как входной порт и x как переменную, которая получит ее значение от ввода вдоль a.
4. Тихий Префикс (Silent Prefix) $\boldsymbol{\tau.P}$, который представляет агента, который может развиться в P без взаимодействия со средой. Мы используем α, β для обозначения $\boldsymbol{ax}, \boldsymbol{\bar{a}x}$, $\boldsymbol{\tau}$ и называем их Префиксами, и мы говорим что: P – Префиксная форма, или иногда только Префикс, когда это не вызове путаницу.
5. Сумма (Sum) **P + Q** представляет агента, который может вводить в действие или P или Q.
6. Параллельная Композиция (Parallel Composition) **P | Q**, которая представляет объединенное поведение P и Q, выполняющиеся параллельно. Компоненты P и Q могут действовать самостоятельно, и могут также общаться, если один выполнит вывод а другой ввод вдоль того же самого порта.
7. Соответствие (Match) **if x=y then P** . Как ожидалось, этот агент будет вести себя как P, если x и y будут одним и тем же именем, иначе он ничего не делает.
8. Несоответствие (Mismatch) **if x≠y then P** . Этот агент будет вести себя как P, если x и y не будут одним и тем же именем, иначе он ничего не делает.
9. Ограничение (Restriction) **(νx)P.** Этот агент ведет себя как P, но имя x является локальным, что означает, что оно не может непосредственно использоваться как порт для коммуникации с P и ее средой. Однако, оно может использоваться для коммуникации между компонентами в пределах P.

Prefixes	α ::=	$\overline{a}x$	Output
		$a(x)$	Input
		τ	Silent
Agents	P ::=	$\mathbf{0}$	Nil
		$\alpha\,.\,P$	Prefix
		$P + P$	Sum
		$P \mid P$	Parallel
		$\mathtt{if}\ x = y\ \mathtt{then}\ P$	Match
		$\mathtt{if}\ x \neq y\ \mathtt{then}\ P$	Mismatch
		$(\boldsymbol{\nu} x)P$	Restriction
		$A(y_1, \ldots, y_n)$	Identifier
Definitions		$A(x_1, \ldots, x_n) \stackrel{\text{def}}{=} P$	(where $i \neq j \Rightarrow x_i \neq x_j$)

Table 1: The syntax of the π-calculus.

10. *Идентификатор (Identifier)* $\boldsymbol{A(y_1, \ldots, y_n)}$, где n - число операндов A. У Каждого идентификатора есть *Определение (Definition)* $A(x_1, \ldots, x_n) \stackrel{\text{def}}{=} P$, где x_i должны быть попарно различными, и интуитивно $A(y_1, \ldots, y_n)$ ведет себя как P с y_i подставленными вместо x_i для каждого i. Таким образом мы можем рассматривать *Определение (Definition)* как объявление процесса, $x_1, \ldots, x_n$ как формальные параметры, и *Идентификатор (Identifier)* $A(y_1, \ldots, y_n)$ как вызов с фактическими параметрами $y_1, \ldots, y_n$.

Операторы хорошо знакомы из других алгебр процессов, поэтому мы далее будем концентрироваться на некоторых важных аспектах пи-исчисления, оставляя читателю изучение более общих принципов.

Формы Nil,Sum и Parallel имеют тот же смысл и применение, что и в других алгебрах процессов, и Префиксные формы такие же как и в алгебрах, которые допускают передачу по значению (value-passing). Такие конструкции if-a как Соответствие (Match) и Несоответствие (Mismatch) могут казаться ограниченными по сравнению с *value-passing* алгебрами, которые обычно допускают произвольные булевы выражения. Но при более близком рассмотрении очевидно, что комбинации Соответствия и Несоответствия единственные возможные проверки, которые могут быть выполнены в пи-исчислении: передаваемые объекты - просто имена, и они не имеют структуры, и никакие операторы не определены на них, таким образом, единственное, что мы можем делать - сравнение. Мы можем объединить такие проверки конъюнктивно, вкладывая их, например в

if x=y then if u≠v then P

что ведет себя как P, если оба условия x = y и u≠v выполняются**.** Мы можем объединить их дизъюнктивно, используя Сумму, например

if x=y then P+ if u≠v then P

что ведет себя как P, если по крайней мере одно из условий x = y или u≠v выполняется**.** Иногда мы будем использовать двойное условное выражение

if x=y then P else Q

как сокращенную запись выражения

if x=y then P+ if x≠y then Q

Как в других алгебрах мы говорим, что P защищен(guarded) в Q, если P- надлежащий подтерм Префиксной формы в Q. Кроме того, Префикс Ввода (Input Prefix) $ax.P$, связывает x в P,и вхождения x в P,тогда называют связанным. В отличие от этого Префикс вывода (Output Prefix) $\bar{a}x.P$ не связывает x. Говорят, что эти префиксы имеют субъект a и объект x, где объект называется свободным в Префиксе вывода и связанным во Префиксе Ввода. У тихого Префикса (Silent Prefix) нет ни субъекта, ни объекта.

Оператор Ограничения (Restriction) (νx)P также связывает x в P. Он имеет тот же эффект, что и в других алгебрах, (где это пишется \x в CCS и δ_x в ACP), но с одной существенной разницей. В обычных алгебрах процессов, вещи, на которые ставится ограничение, являются имена портов и они не могут быть переданы между агентами. Поэтому ограничение статично в том смысле, что область видимости (*scope*) ограниченного имени не должно меняться когда выполняется агент. В пи-исчислении нет разницы между именами портов и значениями, и, имя, которое представляет порт может передаваться между агентами. Если оно ограничено, область ограничения должна измениться, как мы увидим, и действительно почти вся сложность и выразительность пи-исчисления по сравнению с *алгебрами процессов, в которых возможна передача по значению* (*value-passing algebras*), исходит из того, что ограниченные вещи перемещаются. Читатель может думать о (νx)P как о "new x in P", по аналогии с объектно-ориентированным использованием слова "new", так как эта конструкция может быть использована для объявления нового до настоящего времени неиспользованного имени, представленного через x в пользу P.

Обобщая, скажем, что Префикс Ввода и Ограничение связывают имена, и мы можем определить связанные имена bn(P) как те, которые имеют связанное вхождение в P и свободные имена fn(P) как те, у которых нет связанных вхождений, и аналогично для bn(α) и fn(α) для Префикса α. Мы иногда пишем fn(P,Q), имея ввиду fn(P) ∪ fn(Q) , и только α для fn(α) ∪ fn(α), когда, очевидно что это представляет множество имен, как “x∈α ” . В *Определении* $A(x_1, \ldots, x_n) = P$ мы

предполагаем, что верно следующее fn(P)⊆{ $x_1, \dots, x_n$} . В некоторых примерах мы будем игнорировать параметры *Определений* и *Идентификаторов* , когда они будут незначительны или могут быть выведены из контекста.

Подстановка (substitution) – функция, которая отображает имена в имена. Мы пишем {x/y} для замены, которая отображает y в x и является тождественной для всех других имен, и в общем случае $\{x_1, \dots, x_n/y_1, \dots, y_n\}$, где y_i попарно различные, для функции, которая отображает каждый y_i в x_i. Мы используем σ, чтобы обозначать замены, и иногда будем писать $\tilde{x}$ для последовательности имен, когда длина незначительна или может быть выведена из контекста. Агент P_σ - это P, где все свободные имена x заменены на σ(x), с альфа-преобразованием везде, где необходимо избежать связывания. Это означает, что связанные имена переименованы таким образом, что всякий раз когда x заменяется на σ(x) вхождение последнего свободно. Например,

$$(a(x).(\nu b)\bar{x}b.\bar{c}y.0)\{xb/yc\} \text{ есть } a(z).(\nu d)\bar{z}d.\bar{b}x.0$$

Поклонник алгебры процессов возможно заметил, что в пи-исчислении отсутствует один распространенный оператор – оператор перемаркировки (relabelling) (в CCS пишется как [a/b]). Первичное использование перемаркировки должно ограничивать, экземпляры агентов от других агентов (define instances of agents from other agents) , например, если B - буфер с портами $\boldsymbol{i}$ и $\boldsymbol{o}$ тогда B[$\boldsymbol{i'/i}$, $\boldsymbol{o'/o}$] буфер с портами $\boldsymbol{i'}$ и o'. В пи- исчислении мы вместо этого определим экземпляры через параметры Идентификаторов, таким образом например буфер с портами $\boldsymbol{i}$ и $\boldsymbol{o}$ будет B(i, o), и с портами $\boldsymbol{i'}$ и $\boldsymbol{o'}$ это - B(i', o') . Для инъективных перемаркировок (injective relabellings) это только другой стиль спецификации, который позволяет нам экономить на одном операторе. (Читатель, знакомый с перемаркировкой унифицированного списка команд (CCS relabelling), должен быть предупрежден, что она имеет тот же самый эффект как замена порта, только если инъективна. Вообще они различаются.)

Наконец некоторые письменные соглашения: сумма нескольких агентов $P_1 + \cdots + P_n$ записывается как $\sum_{i=1}^{n} P_i$ или только $\sum_j P_j$, когда n незначителен или очевиден, и мы здесь позволяем случай n = 0, когда сумма означает 0. Последовательность отличных Ограничений $(\nu x_1), \dots (\nu x_1)P$ часто сокращается к $(\nu x_1 \dots x_n)P$. В префиксе мы иногда игнорируем объект, если это не важно, таким образом $\boldsymbol{a.P}$ означает $\boldsymbol{a}(\boldsymbol{x}).\boldsymbol{P}$,где x -имя, которое не используется, и аналогично для вывода. И мы иногда игнорируем перемещение 0, пишем $\boldsymbol{a}$ для агента $\boldsymbol{a.0}$, там где это не приведет к путанице. Приоритет унарных операторов выше чем у бинарных, приоритет | выше чем у +. Поэтому имеем:

(vx) P | Q + R тоже что **(((vx) P) | Q) + R**.

2.2 Структурная Конгруэнтность

Синтаксис агентов в каком-то смысле слишком конкретный. Например, агенты $a(x).\bar{b}x$ и $a(y).\bar{b}y$ синтаксически различны, хотя они отличаются выбором связанного имени, зато интуитивно представляют одинаковое поведение: агент, который вводит что-то a и затем посылает это по $\bar{b}$. Как другой пример агенты P|Q и Q|P представляют одно и тоже: параллельную композицию агентов P и Q. По нашему представлению параллельная композиция по сути является неупорядоченной, и мы вынуждены синтаксически различать P|Q и Q|P, так как наш язык линеен.

Поэтому мы вводим понятие структурной конгруэнтности, чтобы отождествлять те агенты, которые представляют одно и тоже. Необходимо подчеркнуть, что это не имеет ничего общего с традиционными поведенческими эквивалентностями в алгебрах процессов, которые определены с точки зрения поведения, проявленного агентом в условиях некоторой операционной семантики. Кроме того, мы должны определить семантику и структурную конгруэнтность, которая отождествляет только тех агентов, из структуры которых сразу становится очевидным, что они одинаковы.

Структурная конгруэнтность $\equiv$ определяется как наименьшее отношение конгруэнтности, которое удовлетворяет следующим законам:

1. Если P и Q – варианты (variants) альфа-преобразования тогда $P \equiv Q$.
2. Законы абелевого моноида для параллельных агентов:
   - Коммутативность $P|Q \equiv Q|P$
   - Ассоциативность $(P|Q)|R \equiv P|(Q|R)$
   - Существование нейтрального элемента: здесь 0 - нейтральный элемент и $P|0 \equiv P$

   и те же самые законы для Суммы (Sum).
3. Закон развертывания (unfolding law)
   $A(\tilde{y}) \equiv P\{\tilde{y}/\tilde{x}\}$ if $A(\tilde{x}) \stackrel{\text{def}}{=} P$.
4. Законы расширения области видимости (scope extension laws)
   - $(\nu x)0 \equiv 0$
   - $(\nu x)(P|Q) \equiv P|(\nu x)Q$ if $x \notin$ fn(P)
   - $(\nu x)(P + Q) \equiv P + (\nu x)Q$ if $x \notin$ fn(P)
   - (νx) if u=v then P $\equiv$ if u=v then $(\nu x)P$ if $x \neq$ u and $x \neq$ v
   - (νx) if u≠v then P $\equiv$ if u≠v then $(\nu x)P$ if $x \neq$ u and $x \neq$ v
   - $(\nu x)(\nu y)P \equiv (\nu y)(\nu x)P$

Читатель здесь правильно возразит, что "представляют ту же самую вещь", и "немедленно очевидный" не являются формально определенными понятиями, и действительно, в литературе могут быть найдены несколько различных вариантов структурной конгруэнтности; нет канонического определения, и у каждого есть различные достоинства. В Разделе 5.1 мы рассмотрим некоторые из них и исследуем их значения. До тех пор мы принимаем частную структурную конгруэнтность. Определение дано выше. Кратко прокомментируем пункты определения.

1. Альфа-преобразование, то есть, выбор связанных имен, отождествляющий таких агентов, как $a(x).\bar{b}x$ и $a(y).\bar{b}y$.
2. Законы абелевого моноида означают, что Параллель и Сумма неупорядочены. Например, когда мы думаем о композиции трех агентов P,Q,R не имеет значения как писать (P|Q)|R или (R|Q)|P.Аналогично для Суммы. Тот факт, что 0-нейтральный элемент, значит, что $P|0 \equiv P$ и $P + 0 \equiv P$, интуитивно это верно, так как 0 пустой агент и поэтому ничего не вносит в Параллельную композицию или Сумму.
3. Развертывание только говорит, что идентификатор то же определение, со соответствующей реализацией параметров.
4. Законы расширения области видимости исходят из нашей интуиции, что (vx)P только говорит, что x - новое уникальное имя в P; об этом можно думать как маркировка вхождений x в P специальным цветом, говоря, что это локальное имя. Тогда не имеет значения, куда помещены символы "(vx)" до тех пор, пока они отмечают одни и те же вхождения. Например, в 0 нет никаких вхождений, поэтому Ограничение может быть удалено по желанию. В Параллельной композиции, если все вхождения находятся в одной компоненте, тогда не имеет значения, только эта компонента или все выражение находится под ограничением.

Надо отметить, что мы не имеем $(\nu x)(P|Q) \equiv (\nu x)P|(\nu x)Q$. Одни и те же вхождения находятся под ограничением в обоих агентах, но в (vx) (P | Q), они ограничены одной и той же связкой (или если хотите, они окрашены в одинаковый цвет), имея в виду, что P и Q могут взаимодействовать используя x, в отличие от ситуации $(\nu x)P|(\nu x)Q$.

Через комбинацию этих законов мы получаем, что $(\nu x)P \equiv P$, если $x \notin$ fn(P) :

$$P \equiv P|0 \equiv P|(\nu x)0 \equiv (\nu x)(P|0) \equiv (\nu x)P$$

Таким образом, как особый случай мы получаем $(\nu x)(\nu x)P \equiv (\nu x)P$ для всех P.

Другим ключевым моментом является то, что все незащищенные ограничения могут быть перенесены на верхний уровень агента.

Предложение 1. Пусть P агент, в котором $(\nu x)Q$ незащищенный подтерм. Тогда P структурно конгруэнтен агенту $(\nu x')P'$, где P' получен из P заменой $(\nu x)Q$ на $Q\{x'/x\}$, для некоторого имени x' , которое не входит в P.

Доказательство проводится альфа - преобразованием всех связанных имен так, чтобы они стали синтаксически различными, после чего применяется расширение области видимости (справа налево) для переноса ограничения в самый внешний уровень. Смысл в том, что вместо того, чтобы что-то объявить локальным, ему можно дать синтаксически отличное от других имя: эффект одинаковый в том, что ничто иное не имеет доступа к этому имени.

Наши законы расширения области видимости выбраны так, чтобы поддерживалось предложение 1. К примеру, мы не определили никаких законов расширения области для Префиксов и можем только выбрасывать незащищенные ограничения. Читатель возможно предполагает выполнение правила $(\nu x)\alpha.P \equiv \alpha.(\nu x)P$ для $x \notin \alpha$. Действительно, это правило удовлетворяет нашей интуиции и не нарушает никаких результатов, полученных в этой статье, и мы его придержим для поведенческих эквивалентностей, которые будут изучаться позже в разделах 6 и 7. Но это правило нам пока не нужно для доказательства Предложения 1.

Структурная конгруэнтность сильнее, так как определяет меньше агентов чем любая другая поведенческая эквивалентность. Структурная конгруэнтность используется в определении операционной семантики, которая в свою очередь используется для определения поведенческих эквивалентностей. Основной причиной выбора нами данного пути является то, что последующие определения и пояснения становятся проще, и мы получаем единообразное обращение с разновидностями исчисления, которые фактически требуют структурной конгруэнтности. В Разделе 5.1 мы комментируем возможность определения исчисления без структурной конгруэнтности.