UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

GLEISON SAMUEL DO NASCIMENTO

Identificação de Nomes Ativos em Agentes-π Baseada em Tipos

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Álvaro Freitas Moreira
Orientador

Prof. Dr. Luis da Cunha Lamb
Co-orientador

Porto Alegre, março de 2005

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Nascimento, Gleison Samuel do

Identificação de Nomes Ativos em Agentes-π Baseada em Tipos / Gleison Samuel do Nascimento. – Porto Alegre: PPGC da UFRGS, 2005.

61 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2005. Orientador: Álvaro Freitas Moreira; Co-orientador: Luis da Cunha Lamb.

1. Cálculo-π. 2. Sistemas de Tipos. 3. Nomes Ativos. 4. Equivalência Comportamental. 5. Análise Estática Baseada em Tipos. I. Moreira, Álvaro Freitas. II. Lamb, Luis da Cunha. III. Título.

“Faça as coisas o mais simples que puder,
porém não as mais simples.”
— Sir Albert Einsten

SUMÁRIO

LISTA DE FIGURAS

RESUMO

Na última década muitos esforços têm sido feitos em verificação formal de propriedades de agentes do cálculo-π. Uma dessas propriedades é a equivalência observacional, que serve para determinar se um processo é equivalente a sua especificação.

Contudo, a verificação de equivalência observacional não é um problema trivial. A maioria dos algoritmos destinados a verificação de equivalência são baseados na construção de sistemas de transições rotuladas (π-autômatos). O principal problema com essa abordagem é o grande número de estados envolvidos podendo chegar a um número infinito.

Montanari e Pistore mostram que é possível gerar π-autômatos finitos para agentes-π e é possível reduzir a quantidade de estados desses π-autômatos, através da identificação dos nomes ativos. Um nome é semanticamente ativo em um agente se ele pode ser executado de forma observável por ele.

Este é um trabalho de análise estática, que tem por objetivo coletar os possíveis nomes ativos contidos em expressões-π, utilizando para isso um sistema de tipos.

A vantagem da utilização de sistemas de tipos em relação a outras formas de análise estática é que sistemas de tipos são sistemas lógicos, logo as técnicas de prova da lógica podem ser aproveitadas no estudo de propriedades de sistemas de tipos. Além disso sistemas de tipos são definidos através da estrutura sintática de expressões, facilitando assim as provas por indução estrutural. Assim a principal contribuição deste trabalho é a elaboração do *Active-Base*-π, um sistema de tipos para a coleta de nomes ativos de expressões-π.

Palavras-chave: Cálculo-π, Sistemas de Tipos, Nomes Ativos, Equivalência Comportamental, Análise Estática Baseada em Tipos.

A Type-Based Approach for Identifying Active Names in π-Calculus

ABSTRACT

In the last decade many efforts have been made in the formal verification of properties of π-calculus agents. One of these properties is called behavioral equivalence which is used for checking whether a process is equivalent to its specification.

However, the verification of behavioral equivalence is not a trivial problem. Most of the algorithms build a labeled transitions system (π-automata) and the main problem is the large number of states involved. In certain situations this number can even be infinite.

Montanari and Pistore show that it is possible to generate finite π-automata for π-agents and it is also possible to reduce the quantity of states of these automata, through the identification of active names. A name is semantically active in an agent if it can be executed in an observable form by it.

This is a work on static analysis, which aims to collect the possible active names contained in π-expressions, using a type system. The advantage of using type systems, instead of other forms of static analysis, is that type systems are logical systems, thus, proof techniques of logic can be used in the study of the desirable properties. Moreover, type systems are defined on the syntactic structure of expressions, facilitating the proofs by structural induction.

The main contribution of this work is the elaboration of *Active-Base-π*, a type system to collect active names of π-expressions.

Keywords: π-Calculus, Type Systems, Active Names, Behavioral Equivalence, Type-Based Static Analysis.

1 INTRODUÇÃO

Linguagens de descrição de processos são muito usadas para especificar e estudar sistemas distribuídos concorrentes. Dentre estas linguagens, destaca-se o cálculo-π, introduzido por Milner, Parrow e Walker (MILNER; PARROW, 1992), como uma linguagem para descrever sistemas concorrentes com características tais como mobilidade, dependência de parâmetros e reconfiguração dinâmica, que não podem ser expressos em CCS (MILNER, 1989). Basicamente, o cálculo-π estende o CCS, apenas acrescentando a ele facilidades para gerar novos nomes de canais de comunicação, e para trocar nomes de canais entre processos concorrentes. Nos últimos anos, o cálculo-π vem sendo utilizado com sucesso em várias aplicações, desde a modelagem de protocolos de telecomunicações, até a modelagem de linguagens orientadas à objetos.

Na última década muitos esforços têm sido feitos para desenvolver verificação formal de propriedades de agentes do cálculo-π. Neste trabalho nosso interesse está na verificação de equivalência observacional entre processos do cálculo-π. A equivalência observacional (MILNER, 1989) (ou bisimilaridade fraca) é uma noção natural de equivalência, onde dois processos são observacionalmente equivalentes se eles são capazes mutuamente, de simular o comportamento um do outro (independentemente do número de passos internos que cada um executa). A verificação da equivalência observacional é muito utilizada na prática, para mostrar que um determinado processo é equivalente a sua especificação.

Uma forma de verificar equivalência é baseada em sistemas de transições rotuladas, onde se deve montar uma relação entre os processos, mostrando que eles podem mutuamente simular um ao outro. Porém aqui o principal problema encontrado é o grande número de estados envolvidos nos processos, podendo chegar a um número infinito de transições.

Contudo Montanari e Pistore (MONTANARI; PISTORE, 1995) provaram que é possível construir autômatos finitos para agentes do cálculo-π, possibilitanto assim o uso de técnicas de verificação finita de equivalência empregadas em CCS. Eles mostram que é possível reduzir o tamanho dos autômatos através da identificação dos nomes ativos no agente-π, facilitando assim a verificação de equivalência. Intuitivamente esses nomes ativos são aqueles que podem ser executados no agente, ou seja, os nomes que podem participar de uma ação observável.

Desta forma fica claro a necessidade de realizar análise estática em agentes-π, para coletar as informações sobre nomes ativos antes de verificar a equivalência. Com base nestas informações é possível realizar uma redução significativa da quantidade de estados nos autômatos construídos para agentes-π, ou até mesmo reduzir o tamanho de uma expressão-π.

1.1 Trabalhos Relacionados

1.1.1 Coleta de Nomes Ativos

O primeiro trabalho apresentado para a identificação de nomes ativos foi o trabalho de Montanari e Pistore (MONTANARI; PISTORE, 1995) como mencionado anteriormente. A partir dos nomes livres do agente eles geram um autômato representando o comportamento deste agente, e sobre este autômato realizam a coleta dos nomes ativos. Após a coleta dos nomes ativos eles podem gerar um novo autômato minimizado usando somente os nomes ativos para realizar transições.

Outro trabalho relacionado a coleta dos nomes ativos foi proposto por Ana Melo (MELO, 2003). Neste trabalho Ana Melo propõem um algoritmo para identificação de nomes ativos, baseado na caracterização sintática dos nomes ativos sobre expressões do cálculo-π. Ou seja, ao invés de gerar autômatos (semântica) e sobre estes calcular os nomes ativos, como proposto por (MONTANARI; PISTORE, 1995), o algoritmo proposto por Ana Melo calcula os nomes ativos através da estrutura sintática de expressões do cálculo-π.

O presente trabalho, baseado nas idéias pioneiras de (MONTANARI; PISTORE, 1995) e (MELO, 2003) para a identificação de nomes ativos, propõe realizar a coleta dos nomes ativos contidos em uma expressão do cálculo-π, baseada em um sistema de tipos.

1.1.2 Sistemas de Tipos para o Cálculo-π

Este é um trabalho de análise estática, técnica bastante difundida em compiladores (AHO; SETHI; ULLMAN, 1995) para a coleta de informações sobre o comportamento de programas, visando principalmente otimização de código. Existem diversas abordagens para a realização de análise estática, e uma delas é baseada em sistema de tipos para linguagens (PIERCE, 2002).

Um sistema de tipos é um mecanismo para classificar expressões de uma linguagem, em nosso caso o cálculo-π. Sistemas de tipos são bastante utilizados para detectar erros de programação estaticamente e para extrair informações que são usadas para raciocinar sobre o comportamento de programas (PIERCE, 2002).

A principal vantagem de utilização de sistemas de tipos em relação a outras formas de análise estática é que sistemas de tipos são sistemas lógicos, logo todas as técnicas de prova da lógica podem ser aproveitadas no estudo de propriedades de sistemas de tipos (PALSBERG, 2001). Além disso sistemas de tipos são definidos através da estrutura sintática de expressões, facilitando assim provas por indução estrutural.

Assim um grande número de sistemas de tipos vem sendo propostos para analisar várias propriedades de especificações escritas em cálculo-π, tais como:

- Frequência com que o cada canal é usado (KOBAYASHI; PIERCE; TURNER, 1999): se um determinado canal de comunicação entre dois processos não interfere com a comunicação de outros processos;

- *Deadlock* (KOBAYASHI, 1998): este sistema de tipos garante que a comunicação por canais compartilhados nunca causa *deadlock* e também que certas comunicações nunca introduzem o não determinismo;

- *Livelock* (KOBAYASHI, 2002): já este sistema de tipos garante que certas comunicações eventualmente acontecerão durante a execução do processo.

Sistemas de tipos para o cálculo-π são apresentados em (SANGIORGI; WALKER, 2001).

Neste trabalho é apresentado o sistema de tipos *Active-Base*-π que é capaz de realizar a identificação de nomes *potencialmente* ativos em uma expressão-π. É importante enfatizar que não existe nenhum algoritmo, que dada uma expressão-π, seja capaz de identificar exatamente o conjunto de nomes que de fato serão executados. A identificação de nomes ativos é portanto, um problema indecidível [1]. O que procuramos fazer aqui é coletar um conjunto de nomes potencialmente ativos, porém é possível que neste conjunto existam nomes que não serão de fato executados. Para melhorar a qualidade das otimizações a serem feitas, é desejável que esse conjunto de nomes potencialmente ativos seja o menor possível. Mas é essencial que a coleta seja segura, ou seja, nenhum nome observável pode ficar de fora desse conjunto. De agora em diante, faremos referência aos nomes potencialmente ativos, simplesmente por nomes ativos para simplificar o entendimento deste texto.

1.2 Estrutura do Trabalho

O restante deste trabalho está agrupado em quatro capítulos como segue:

- **Capítulo 2** - Neste capítulo introduzimos os conceitos básicos do cálculo-π e do cálculo-π tipado. Para um melhor entendimento dividimos este capítulo em duas partes. A primeira parte é destinada para leitores que não estão familiarizados com os conceitos do cálculo-π. Aqui apresentamos as noções básicas do cálculo como, sintaxe, semântica operacional, congruência estrutural e equivalência de processos. Caso o leitor já esteja a vontade com estes conceitos iniciais, poderá partir direto para a segunda parte deste capítulo. Na segunda parte descrevemos, em detalhes, os conceitos básicos do cálculo-π tipado. Apresentamos as noções básicas e principalmente o sistema de tipos *Base*-π que é utilizado no restante do trabalho para definirmos um novo sistema de tipos capaz de coletar as informações de nomes ativos.

- **Capítulo 3** - Aqui apresentamos a caracterização semântica de nomes ativos em agentes do cálculo-π. Também introduzimos a construção de π-autômatos que são estruturas muito utilizadas na verificação de equivalências.No final deste capítulo apresenta-se duas abordagens utilizadas para detecção de nomes ativos que serviram de inspiração para a realização deste trabalho.

- **Capítulo 4** - Neste capítulo apresentamos o *Active-Base*-π sistema de tipos desenvolvido neste trabalho para a coleta dos nomes ativos em expressões-π. Mostramos exemplos da utilização do sistema para a coleta dos nomes ativos e procuramos discutir propriedades e possíveis transformações das expressões-π, utilizando as informações coletadas pelo *Active-Base*-π. O conteúdo deste capítulo é a principal contribuição deste trabalho.

- **Capítulo 5** - Por fim, no último capítulo, aponta-se caminhos futuros a serem seguidos pela pesquisa.

[1] De fato isso é verdade para toda análise estática visando a coleta de informações sobre o comportamento de execução de programas.

2 O CÁLCULO$-\pi$

O cálculo-π (MILNER; PARROW, 1992) é um cálculo para processos que é capaz de descrever mudanças na estrutura de processos concorrentes. O cálculo-π é uma extensão do CCS com a idéia de mobilidade de canais de comunicação, preservando todas as propriedades algébricas do CCS.

O passo computacional básico do cálculo-π é a transferência de um canal de comunicação entre dois processos. Assim o processo receptor poderá usar o novo canal para futuras interações com outros processos, aos quais não teria acesso antes de receber o novo nome.

Neste trabalho utiliza-se como referência para a introdução do cálculo-π o tutorial de Joachim Parrow (PARROW, 2001). Outras fontes para pesquisa sobre o cálculo-π são os primeiros tutoriais escritos por Milner e Parrow em 92 (MILNER; PARROW, 1992), além dos livros de Milner (MILNER, 1999) e de Sangiorgi e Walker (SANGIORGI; WALKER, 2001) que são ótimas obras de refências para quem quiser conhecer melhor a teoria do cálculo-π.

2.1 Noções Básicas

Esta seção é destinada a descrever as noções básicas do cálculo-π: sua sintaxe, semântica operacional, conceitos de congruência estrutural e equivalência entre processos. Caso o leitor já esteja a vontade com estes conceitos iniciais, poderá partir direto para a segunda seção deste capítulo, que descreve o cálculo-π tipado que será utilizado no restante deste trabalho.

2.1.1 Sintaxe

Assume-se aqui um conjunto infinito e contável N de nomes $a, b, x, y, ...$, representando os canais de comunicação, variáveis e valores do cálculo, e também um conjunto de identificadores de agentes representados por $A, B...$. Os agentes do cálculo-π são definidos conforme a gramática da Figura 2.1 e possuem o seguinte significado intuitivo (as metavariáveis $P, Q, ...$ são usadas para representar agentes):

1. O agente nulo **0**, representa um agente vazio e que não pode realizar nenhuma ação.

2. Um agente $\alpha.P$, é um agente na forma prefixa. Neste caso existem três tipos de agentes prefixos, conforme definição de prefixos α dada na Figura 2.1:

   - no agente $\overline{a}b.P$ o nome b pode ser enviado através do nome a e logo depois do envio, o agente continua como P. Assim $\overline{a}$ pode ser visto como uma porta de

Prefixos	$\alpha ::= \overline{a}b$	Saída
	$a(x)$	Entrada
	τ	Silencioso
Agentes	$P, Q ::= 0$	Nulo
	$\alpha.P$	Prefixo
	$P + Q$	Soma de Processos
	$P \mid Q$	Composição Paralela
	$[a = b]P$	Igualdade
	$(\nu x)P$	Restrição
	$!P$	Replicação
	$A(y_1, ..., y_n)$	Identificador
Definições	$A(x_1, ..., x_n) \stackrel{def}{=} P$	(onde $x_1, ..., x_n$ são diferentes entre si)

Figura 2.1: Sintaxe do cálculo-π.

saída e o nome b como um valor enviado para fora do agente através da porta de saída a.

- já o agente $a(x).P$ representa o recebimento de um valor através do nome a. Após a entrada o agente continua como P, porém aqui todas as ocorrências do nome x que estiverem em P devem ser substituídas pelo valor recebido. Assim a pode ser visto como uma porta de entrada e x como uma variável que receberá o seu valor através da porta a.
  Por exemplo, considere o agente $a(x).\overline{x}c.0$. Agora considere que um nome b é recebido pelo agente através de a. Desta forma o agente passará a se comportar como $\overline{b}c.0$, pois após executar a entrada, todas as ocorrências de x são substituídas pelo valor recebido b.
- por fim temos o agente $\tau.P$, que significa que o agente pode passar a agir como P sem nenhum tipo de interação com o ambiente, ou seja, τ representa uma ação silenciosa.

3. O agente $P + Q$ representa uma escolha não determinística, ou seja, o agente pode agir ou como P ou como Q.

4. A composição paralela $P|Q$ representa a combinação comportamental de P e Q executando em paralelo. Neste caso os componentes P e Q podem agir independentemente, ou podem se comunicar entre si quando um deles executar uma entrada e o outro executar uma saída através da mesma porta de comunicação.
   Por exemplo considere o agente $\overline{a}b.P|a(x).Q$. Aqui os agentes podem vir a se comunicar entre si, pois o primeiro pode executar um prefixo de saída e o segundo pode executar um prefixo de entrada, ambos através do canal de comunicação a.

5. A forma $[a = b]P$ é chamada de um processo igualdade ou *matching*. Neste caso o agente irá se comportar como P se a e b são os mesmos nomes, caso contrário não acontece nada.

6. A restrição $(\nu x)P$ diz que o agente comporta-se como P porém o nome x tem escopo local, ou seja, ele não pode ser usado como um canal de comunicação entre

P e o ambiente externo. Contudo, este nome poderá ser usado para a comunicação entre componentes dentro do agente P.

7. Se P é um agente então $!P$ é a replicação de P dada pela definição $!P \stackrel{def}{=} P|!P$. Em outras palavras, $!P$ representa um número não limitado de cópias de P.

8. A forma $A(y_1, ..., y_n)$ é chamada de um identificador de processos, onde $n \geq 0$ representa a aridade de A. Todos os identificadores tem uma definição $A(x_1, ..., x_n) \stackrel{def}{=} P$ onde os nomes x_i devem ser distintos. A intuição aqui é que $A(y_1, ..., y_n)$ comporta-se como P com y_i substituindo x_i para cada i. Desta forma uma definição pode ser vista como uma declaração de processo, onde $x_1, ..., x_n$ são os parâmetros formais da declaração. E consequentemente, $A(y_1, ..., y_n)$ é uma chamada do processo tendo como parâmetros atuais $y_1, ..., y_n$.

Os operadores são familares a outros cálculos de processos, assim vamos nos concentrar em alguns aspectos do cálculo-π que são importantes para o restante do trabalho.

Nos agentes prefixos $\overline{a}b.P$ e $a(x).P$, define-se que o nome a é o sujeito do prefixo, enquanto os nomes b e x são os objetos dos respectivos prefixos. Assim em $a(x).P$, o objeto x liga o nome x à P, ou seja, o nome x está sendo declarado neste ponto, e portanto todas as ocorrências de x em P são chamadas de ocorrências ligadas. Já em $\overline{a}b.P$ o objeto b não liga o nome b à P. Assim pode-se dizer que o objeto do prefixo será *livre* para um prefixo de saída e *ligado* para um prefixo de entrada.

O operador de restrição $(\nu x)P$ declara um novo nome representado por x. Desta forma, o operador de restrição também liga x à P, da mesma forma que um prefixo de entrada ($a(x)$). Seu efeito aqui é similar a outros cálculos de processos, porém com uma significativa diferença a ser explicada na Seção 2.1.2.

Em resumo, tanto os prefixos de entrada quanto a restrição ligam nomes. Dessa forma o conjunto $bn(P)$ dos nomes ligados de P é definido indutivamente na estrutura sintática dos agentes-π definida na Figura 2.1 como segue:

Definição 1 *Seja P um agente-π então:*

- $bn(0) = \{\}$
- $bn(a(x).P) = \{x\} \cup bn(P)$
- $bn(\overline{a}b.P) = bn(P)$
- $bn(\tau.P) = bn(P)$
- $bn(P + Q) = bn(P) \cup bn(Q)$
- $bn(P|Q) = bn(P) \cup bn(Q)$
- $bn([a = b]P) = bn(P)$
- $bn((\nu x)P) = \{x\} \cup bn(P)$
- $bn(!P) = bn(P)$
- $bn(A(y_1, ..., y_n)) = bn(P)$ *se* $A(x_1, ..., x_n) \stackrel{def}{=} P$

■

Por outro lado, todos os nomes que não aparecem ligados em P, são ditos nomes livres de P. De forma similar a $bn(P)$, podemos definir o conjunto $fn(P)$ dos nomes livres de P, indutivamente na estrutura sintática dos agentes-π.

Definição 2 *Seja P um agente-π então:*

- $fn(0) = \{\}$
- $fn(a(x).P) = (\{a\} \cup fn(P)) - \{x\}$
- $fn(\overline{a}b.P) = \{a, b\} \cup fn(P)$
- $fn(\tau.P) = fn(P)$
- $fn(P + Q) = fn(P) \cup fn(Q)$
- $fn(P|Q) = fn(P) \cup fn(Q)$
- $fn([a = b]P) = \{a, b\} \cup fn(P)$
- $fn((\nu x)P) = fn(P) - \{x\}$
- $fn(!P) = fn(P)$
- $fn(A(y_1, ..., y_n)) = \{y_1, ..., y_n\}$ *se* $A(x_1, ..., x_n) \stackrel{def}{=} P$

■

Aqui também é importante ressaltar que na definição $A(x_1, ..., x_n) \stackrel{def}{=} P$ assume-se que $fn(P) = \{x_1, ..., x_n\}$.

Exemplo 1 *Sendo $P = (\nu y)a(x).\overline{x}b.\overline{y}c.0$, obtemos os seguintes resultados:*

$$\begin{aligned}
bn(P) &= bn((\nu y)a(x).\overline{x}b.\overline{y}c.0) \\
&= \{y\} \cup bn(a(x).\overline{x}b.\overline{y}c.0) \\
&= \{y\} \cup (\{x\} \cup bn(\overline{x}b.\overline{y}c.0)) \\
&= \{y\} \cup (\{x\} \cup bn(\overline{y}c.0)) \\
&= \{y\} \cup (\{x\} \cup bn(0)) \\
&= \{y\} \cup (\{x\} \cup \{\}) \\
&= \{y, x\}
\end{aligned}$$

$$\begin{aligned}
fn(P) &= \{a, b, c\} \cup fn(a(x).\overline{x}b.\overline{y}c.0) - \{y\} \\
&= \{a, b, c\} \cup (\{a\} \cup fn(\overline{x}b.\overline{y}c.0)) - \{x\} - \{y\} \\
&= \{a, b, c\} \cup (\{a\} \cup (\{x, b\} \cup fn(\overline{y}c.0))) - \{x, y\} \\
&= \{a, b, c\} \cup (\{a\} \cup (\{x, b\} \cup (\{y, c\} \cup fn(0)))) - \{x, y\} \\
&= \{a, b, c\} \cup (\{a\} \cup (\{x, b\} \cup (\{y, c\} \cup \{}))) - \{x, y\} \\
&= \{a, b, c\} \cup \{a, b, c\} \\
&= \{a, b, c\}
\end{aligned}$$

■

Outros dois pontos importantes em cálculo-π são os conceitos de α-conversão e de substituição. Dois processos P e Q são α-conversíveis se Q pode ser obtido de P por um número finito de renomeações dos nomes ligados. Uma renomeação de nomes ligados em um processo P nada mais é do que a troca de um subtermo $a(y).Q$ de P por $a(x).Q\{x/y\}$, ou a troca de um subtermo $(\nu y)Q$ de P por $(\nu x)Q\{x/y\}$, onde nos dois casos x não ocorre em Q.

Uma substituição é uma função de nomes para nomes, assim, escreve-se $\{x/y\}$ para dizer que o nome y será substituído por x, e generalizando escreve-se $\{x_1, ..., x_n/y_1, ..., y_n\}$ para dizer que cada y_i será substituído por x_i. Usa-se ϕ para variar sobre substituições e portanto $P\phi$ será o agente P onde todos os nomes livres x são substituídos por $\phi(x)$, com α-conversões quando necessário. Com a α-conversão, os nomes ligados são renomeados de tal forma que sempre que x for substituído por $\phi(x)$, as ocorrências de $\phi(x)$ são sempre livres. Por exemplo, $((\nu y)a(x).\overline{x}b.\overline{y}c.0)\{y/b\}$ é igual à $(\nu z)a(x).\overline{x}y.\overline{z}c.0$. Isto porque o nome b que é livre, será substituído por y que já é um nome ligado em $((\nu \mathbf{y})a(x).\overline{x}b.\overline{\mathbf{y}}c.0)$. Assim o nome ligado y pode ser renomeado por α-conversão para z e a substituição $\{y/b\}$ poderá ser feita normalmente obtendo $(\nu z)a(x).\overline{x}y.\overline{z}c.0$.

2.1.2 Extrusão de Escopo

Em outros cálculos de processos os nomes de canais restritos não podem ser transmitidos entre os agentes. Além disso a restrição é estática pois o escopo de um nome restrito não muda quando o agente executa este nome.

Já no cálculo-π não há diferenças entre nomes e valores, portanto, um nome que representa um canal pode ser transmitido entre agentes. Assim, se em uma comunicação o nome sendo transmitido for restrito, o escopo de sua restrição deve mudar.

Considere como exemplo o agente $(\nu a)(a(x).P|(\nu b)\overline{a}b.Q)$. Note que o agente pode se comunicar através do canal de comunicação a, porém aqui o segundo componente da composição ($(\nu b)\overline{a}b.Q$) envia através de a o nome b que é restrito apenas no escopo de $\overline{a}b.Q$. Como não há distinção entre nomes e valores no cálculo-π esta é uma comunicação válida, porém ao executar a comunicação, o escopo do nome b deve ser expandido para compreender também o agente P na restrição. O resultado da comunicação será $(\nu a, b)(P\{b/x\}|Q)$, onde a restrição de b compreende agora toda a composição paralela.

A noção de extrusão de escopo será muito importante para o restante do trabalho, na definição do sistema de tipos para a coleta dos nomes ativos.

2.1.3 Congruência Estrutural

Existem casos em que agentes são sintaticamente diferentes mas possuem o mesmo comportamento. Por exemplo, os agentes $P|Q$ e $Q|P$ são sintaticamente diferentes mas comportam-se da mesma forma. Nestes casos é definida uma congruência estrutural para identificar os agentes que, intuitivamente, representam a mesma coisa.

Aqui é importante enfatizar a diferença entre congruência estrutural e equivalência observacional. A equivalência observacional é definida em termos do comportamento exibido por um agente sobre alguma semântica operacional. Já a congruência estrutural identifica somente alguns agentes em que a estrutura sintática torna óbvio que eles representam a mesma coisa. Abordaremos na próxima seção as propriedades relativas a semântica operacional e a equivalência observacional.

Existem várias versões de congruência estrutural na literatura, pois não existe uma definição canônica. Para este trabalho adotamos a versão citada em (PARROW, 2001). A congruência estrutural é representada pelo símbolo $\equiv$, e é definida como a menor con-

gruência satisfazendo as leis mostradas na Figura 2.2.

1. Se P e Q são variantes de α-conversão então $P \equiv Q$.

2. As leis para composição paralela e soma de processos:

   - Comutatividade - $P|Q \equiv Q|P$ e $P + Q \equiv Q + P$.
   - Associatividade - $(P|Q)|R \equiv P|(Q|R)$ e $(P + Q) + R \equiv P + (Q + R)$.
   - 0 como identidade - $P|0 \equiv P$ e $P + 0 \equiv P$.

3. A lei de *unfolding* $A(y_1, ..., y_n) \equiv P\{y_1, ..., y_n/x_1, ..., x_n\}$ se $A(x_1, ..., x_n) \stackrel{def}{=} P$

4. Leis de extensão de escopo:

a.	$(\nu x)0$	$\equiv$	0		
b.	$(\nu x)(P\|Q)$	$\equiv$	$P\|(\nu x)Q$	se	$x \notin fn(P)$
c.	$(\nu x)(P + Q)$	$\equiv$	$P + (\nu x)Q$	se	$x \notin fn(P)$
d.	$(\nu x)[a = b]P$	$\equiv$	$[a = b](\nu x)P$	se	$x \neq a \wedge x \neq b$
e.	$(\nu x)(\nu y)P$	$\equiv$	$(\nu y)(\nu x)P$		

Figura 2.2: Definições de congruência estrutural.

A primeira lei diz respeito a α-conversão. Por essa lei, agentes que diferem somente na escolha de nomes ligados representam o mesmo comportamento. Por exemplo, os agentes $a(x).\bar{b}x$ e $a(y).\bar{b}y$ diferem somente nos nomes ligados mas os dois representam o mesmo comportamento, ou seja, o agente recebe alguma informação por a e envia esta informação através de b.

Na segunda lei é definida a comutatividade e associatividade para a soma e composição paralela de processos, isto significa que a ordem dos componentes nestes processos não interfere no comportamento. Também temos aqui a definição de que a composição e a soma de processos de qualquer agente P com 0 é igual a P, pois 0 é o agente vazio e não contribui em nada para uma composição paralela ou soma.

O *unfolding*, definido na terceira lei, diz que um identificador é o mesmo que sua definição, com a apropriada instanciação de parâmetros. Ou seja, uma instância do agente A é o mesmo que a definição do agente substituíndo todos os parâmetros formais $x_1, ..., x_n$, pelos parâmetros atuais da instância $y_1, ..., y_n$.

Por último, temos as leis de extensão de escopo que mostram que não importa onde a restrição é colocada no agente, desde que todas as ocorrências do nome restrito estejam dentro do escopo do símbolo (νx). Por exemplo, em um agente 0 não há nenhuma ocorrência de nomes assim a restrição pode ser removida. Já em uma composição paralela, se todas as ocorrências de x estão em um dos componentes então não importa se a restrição é colocada cobrindo todos componentes da composição ou cobrindo somente o componente onde aparecem as ocorrências.

Note que não temos $(\nu x)(P|Q) \equiv (\nu x)P|(\nu x)Q$. Neste caso as mesmas ocorrências são restritas em ambos os agentes, porém em $(\nu x)(P|Q)$ eles são restritos usando o mesmo *ligador*. Isto significa que P e Q podem interagir usando x, em contraste com o agente $(\nu x)P|(\nu x)Q$ que usa *ligadores* diferentes.

A principal razão para definirmos primeiro a congruência estrutural, é que as definições da semântica operacional e da equivalência observacional tornam-se mais simples,

pois a congruência estrutural é usada para definição de uma semântica operacional, que por sua vez é usada para definir a equivalência observacional.

Exemplo 2 *Considere os seguintes processos:*

- $P = (\nu y)(\nu z)(\overline{a}b.y(x).\overline{x}c.0 \mid \overline{y}z.d(u).u(w).0)$
- $Q = (\nu y)((\nu z)\overline{y}z.d(u).u(v).0 \mid \overline{a}b.y(x).\overline{x}c.0)$

Estes dois processos são estruturalmente congruentes, conforme a aplicação das seguintes leis da Figura 2.2 sobre o processo Q:

$$\begin{array}{rcll} Q & \equiv & (\nu y)(\overline{a}b.y(x).\overline{x}c.0 \mid (\nu z)\overline{y}z.d(u).u(v).0) & \textit{(Comutatividade sob Q)} \\ & \equiv & (\nu y)(\nu z)(\overline{a}b.y(x).\overline{x}c.0 \mid \overline{y}z.d(u).u(v).0) & \textit{(Extensão de Escopo)} \\ & \equiv & (\nu y)(\nu z)(\overline{a}b.y(x).\overline{x}c.0 \mid \overline{y}z.d(u).u(w).0)\{w/v\} & \textit{(}\alpha\textit{-conversão)} \\ & \equiv & (\nu y)(\nu z)(\overline{a}b.y(x).\overline{x}c.0 \mid \overline{y}z.d(u).u(w).0) & \\ & \equiv & P & \end{array}$$

■

2.1.4 Semântica Operacional

A semântica operacional para o cálculo-π é dada através de um sistema de transições rotuladas, onde as transições são do tipo $P \xrightarrow{\alpha} Q$ para algum conjunto de ações representadas por α. Assim, para um agente $\alpha.P$, acontecerá uma transição onde o rótulo será dado pela ação α. Essas ações podem ser de 4 tipos:

- Silenciosa (τ): em uma transição $P \xrightarrow{\tau} Q$, o agente P evolui para Q sem interação com o ambiente externo. Essa ação pode ter origem de um agente $\tau.P$ ou de uma comunicação interna do agente.
  Por exemplo considere o agente $a(x).P|\overline{a}b.Q$, neste caso pode ocorrer uma comunicação interna através do canal de comunicação a. Caso isso aconteça a transição será do tipo silenciosa e terá a seguinte forma: $a(x).P|\overline{a}b.Q \xrightarrow{\tau} P\{b/x\}|Q$. Note que após a transição deve ocorrer a substituição de x pelo nome y que foi enviado pelo segundo componente da composição paralela. Também observe que este tipo de comunicação só pode ocorrer quando em ambos os componentes da composição os prefixos a serem executados possuem o mesmo canal de comunicação, e um dos componentes executa uma entrada e o outro uma saída.

- Entrada (ab): na transição $P \xrightarrow{ab} Q$, o agente P recebe o nome b através do canal de comunicação a e então evolui para Q. Note que todas as ocorrências de x em Q devem ser substituídas pelo nome b que foi recebido na transição. Esse tipo de ação origina-se de agentes na forma $a(x).P$. Por exemplo, no agente $a(x).P$ ocorre uma transição na forma $a(x).P \xrightarrow{ab} P\{b/x\}$ e então o agente passa a comporta-se como $P\{b/x\}$, onde todas as ocorrências de x em P são substituídas por b.

- Saída livre ($\overline{a}b$): em uma transição $P \xrightarrow{\overline{a}b} Q$, o agente P envia um nome livre b através do canal de comunicação a, passando a se comportar como Q. Essas ações têm origem de agentes na forma $\overline{a}b.P$.

- Saída ligada ($\overline{a}\nu x$): Similar a uma transição de saída livre, a transição $P \stackrel{\overline{a}\nu x}{\rightarrow} Q$, representa a saída de um nome local de P, ou seja x é restrito em P e indica onde este nome ocorre em Q. Este tipo de ação ocorre em agentes que enviam nomes para fora do escopo, e tem a forma $(\nu x)\overline{a}x.P$.

Antes de prosseguir é importante observar que, agentes da forma $(\nu a)a(x).P$ ou $(\nu a)\overline{a}b.P$ não podem evoluir, pois o sujeito da ação está restrito. Estes agentes são ditos estar em *deadlock*. Porém note que agentes na forma $(\nu a)(a(x).P|\overline{a}b.Q)$ não estão em *deadlock* pois aqui pode ocorrer uma comunicação silenciosa τ. Se o agente fosse $(\nu a)(a(x).P|\overline{b}c.Q)$ então diriamos que somente o segundo componente da composição poderia evoluir, pois o nome a do primeiro componente está restrito e por isso não pode executar.

Na Figura 2.3 é mostrada a semântica operacional das transições rotuladas conforme (PARROW, 2001). Note que a regra STRUCT torna explícita a intuição de que agentes congruentes são iguais para as finalidades da semântica. Isto simplifica o sistema de regras de transições. Por exemplo, se a regra STRUCT fosse eliminada seria necessário criar duas regras para a soma de processos. Um regra SUM como definido na Figura 2.3, e uma segunda regra SUM_2 para o caso que o agente evolui para Q. Porém a segunda regra pode ser inferida através das regras SUM e STRUCT, tornando-se assim redundante e podendo ser eliminada da semântica.

$$\frac{P' \equiv P \qquad P \stackrel{\alpha}{\rightarrow} Q \qquad Q \equiv Q'}{P' \stackrel{\alpha}{\rightarrow} Q'} \text{(STRUCT)}$$

$$\frac{}{\overline{a}b.P \stackrel{\overline{a}b}{\rightarrow} P} \text{(OUT)} \qquad \frac{}{a(x).P \stackrel{ab}{\rightarrow} P\{b/x\}} \text{(INP)}$$

$$\frac{}{\tau.P \stackrel{\tau}{\rightarrow} P} \text{(TAU)} \qquad \frac{P \stackrel{\alpha}{\rightarrow} P'}{[x=x]P \stackrel{\alpha}{\rightarrow} P'} \text{(MATCH)}$$

$$\frac{P \stackrel{\alpha}{\rightarrow} P'}{P+Q \stackrel{\alpha}{\rightarrow} P'} \text{(SUM)} \qquad \frac{P \stackrel{\alpha}{\rightarrow} P' \qquad bn(\alpha) \bigcap fn(Q) = \emptyset}{P|Q \stackrel{\alpha}{\rightarrow} P'|Q} \text{(PAR)}$$

$$\frac{P \stackrel{ab}{\rightarrow} P\{b/x\} \qquad Q \stackrel{\overline{a}b}{\rightarrow} Q'}{P|Q \stackrel{\tau}{\rightarrow} P\{b/x\}|Q'} \text{(COM)} \qquad \frac{P \stackrel{\alpha}{\rightarrow} P' \qquad x \notin n(\alpha)}{(\nu x)P \stackrel{\alpha}{\rightarrow} (\nu x)P'} \text{(RES)}$$

$$\frac{P \stackrel{\overline{a}x}{\rightarrow} P' \qquad a \neq x}{(\nu x)P \stackrel{\overline{a}\nu x}{\rightarrow} P'} \text{(OPEN)} \qquad \frac{P \stackrel{\alpha}{\rightarrow} P'}{!P \stackrel{\alpha}{\rightarrow} P'|!P} \text{(REP)}$$

Figura 2.3: Semântica Operacional.

Na regra PAR existe uma condição extra que Q não pode ter um nome ligado em α, isto porque na conclusão $P|Q \stackrel{\alpha}{\rightarrow} P'|Q$ a ação não deve alterar nenhuma ocorrência de nomes livres em Q.

A regra OPEN é a regra gerada para saídas ligadas. É interessante notar que ações de saída ligada não aparecem na regra COM e além disso não podem interagir diretamente com entradas. Assim estas interações são inferidas pelo uso de congruência estrutural para puxar a restrição para a parte externa dos agentes.

Na literatura pode-se encontrar duas formas de representar a semântica operacional, uma chamada de semântica *early* e a outra chamada de semântica *late*.

A principal diferença entre a semântica *early* e a semântica *late* esta na regra INP. Na semântica *early* a transição de entrada $a(x).P \stackrel{ab}{\rightarrow} P\{b/x\}$ significa que P recebe o nome b e continua como $P\{b/x\}$. Já na semântica *late* a transição é representa como $a(x).P \stackrel{a(x)}{\rightarrow} P$, e significa que a ação de entranda $a(x)$ é executada e algum nome é recebido para substituir x em P, ou seja, o nome que será recebido na transição é resolvido somente depois da transição. Consequentemente a regra COM também será alterada na semântica *late*. Mais informações sobre as diferenças entre as formas de representar a semântica operacional, podem ser encontradas em (PARROW, 2001).

Neste trabalho é adotada a semântica *early*, isto porque o sistema de tipos *Base-*π que utiliza-se na sequência foi definido utilizando a semântica *early*.

2.1.5 Bissimilaridade

No cálculo-π, assim como na maioria das álgebras de processos, as relações de equivalência de agentes são baseadas em bissimulações. A definição genérica de bissimulação é que ela é uma relação binária e simétrica $\mathcal{R}$ entre agentes satisfazendo:

$$P\mathcal{R}Q \wedge P \stackrel{\alpha}{\rightarrow} P' \Rightarrow \exists Q' : Q \stackrel{\alpha}{\rightarrow} Q' \wedge P'\mathcal{R}Q'$$

Isso quer dizer que se P pode fazer uma ação então Q pode fazer a mesma ação e vice-versa. Dois agentes são bissimilares se eles estão relacionados por alguma bissimulação, ou seja, eles podem indefinidamente simular as transições um do outro.

No cálculo-π é necessário tomar um cuidado especial com as ações de objetos ligados. Para verificar esta característica considere que $P = a(u).0$ e $Q = a(x).(\nu v)\overline{v}u.0$. Intuitivamente estes agentes representam o mesmo comportamento: eles podem executar uma entrada através de a e então não executam mais nenhuma ação. Contudo, Q tem o nome u livre onde P não tem. Além disso, x em Q não pode ser α-convertido para u, e a ação $a(u)$ não pode ser simulada por Q. Tal diferença entre P e Q não é importante desde que, se P tem uma ação $a(u)$, então por α-conversão ele também tem uma ação derivada $a(w)$ para muitos w. Claramente é suficiente que um agente simule somente ações ligadas onde o nome ligado não ocorra livre no próprio agente. Este argumento aplica-se tanto para ações de entrada , quanto para ações de saída ligada.

Note também que em transições de entrada do tipo $a(x).P \stackrel{ab}{\rightarrow} P'$, o comportamento de P' deve ser considerado sob todas as substituições {b/x}, assim para cada uma destas substituições Q' deve estar relacionado à P', ou seja, eles devem estar relacionados para cada um dos nomes recebidos.

Existem diversos modos de definir equivalência conforme (PARROW, 2001). Neste trabalho utilizamos as definições de equivalência forte (ou bissimilaridade forte) e equivalência fraca (ou bissimilaridade fraca). Para a definição que segue, o uso de *"bn(α) é livre"* significa que o nome em *bn(α)* é diferente de qualquer nome livre que ocorre nos agentes.

Definição 3 *Uma bissimulação forte é uma relação binária e simétrica $\mathcal{R}$ em agentes satisfazendo: $P\mathcal{R}Q \wedge P \stackrel{\alpha}{\rightarrow} P'$ onde $bn(\alpha)$ é livre implica que*

1. *Se α é uma entrada ($\alpha = ab$) então* $\exists Q' : Q \xrightarrow{ab} Q' \wedge \forall b : P'\{b/x\}\mathcal{R}Q'\{b/x\}$.

2. *Se α não é uma entrada então* $\exists Q' : Q \xrightarrow{\alpha} Q' \wedge P'\mathcal{R}Q'$.

Dizemos que P e Q são fortemente bissimilares, escrevendo $P \sim Q$, se eles estão relacionados por uma bissimulação. Também dizemos que $P \equiv Q$ implica que $P \sim Q$, em virtude da regra STRUCT mostrada na semântica operacional.

Já a idéia principal de bissimilaridade fraca é que as transições cuja a ação é τ são consideradas como não observáveis. Assim são definidos:

- $\Rightarrow$, que significa $(\xrightarrow{\tau})^*$, isto é, zero ou mais transições τ;
- $\stackrel{\alpha}{\Rightarrow}$, que significa $\Rightarrow \xrightarrow{\alpha} \Rightarrow$;
- $\stackrel{\hat{\alpha}}{\Rightarrow}$, que signigica $\stackrel{\alpha}{\Rightarrow}$ se $\alpha \neq \tau$ e $\Rightarrow$ se $\alpha = \tau$.

Definição 4 *Uma bissimulação fraca é uma relação binária e simétrica $\mathcal{R}$ entre agentes que satisfaz o seguinte $P\mathcal{R}Q$ e $P \xrightarrow{\alpha} P'$ onde $bn(\alpha)$ é livre implica que:*

1. *Se α é uma entrada ($\alpha = ab$) então* $\exists Q" : Q \Rightarrow \xrightarrow{ab} Q'' \wedge \forall b \exists Q' : Q''\{b/x\} \Rightarrow Q' \wedge P'\{b/x\}\mathcal{R}Q'$

2. *Se α não é uma entrada então* $\exists Q' : Q \stackrel{\hat{\alpha}}{\Rightarrow} Q' \wedge P'\mathcal{R}Q'$

Assim dizemos que P e Q são fracamente bissimilares se eles estão relacionados por uma bissimulação fraca, e é escrito $P \sim Q$ para representar a bissimulação fraca.

2.2 Cálculo-π Tipado

Sistemas de tipos são ótimos mecanismos para classificar expressões de uma linguagem. Eles são usados por linguagens de programação, sequenciais ou concorrentes, por uma série de motivos, como por exemplo, para detectar erros de programação, extrair informações para raciocinar sobre o comportamento de programas ou até mesmo para tornar os programas mais fáceis de entender.

Tipos em cálculo-π são uma parte importante da teoria e são uma das mais importantes diferenças entre o cálculo-π e outras linguages de processos concorrentes (como o CCS por exemplo). A maioria dos sistemas de tipos estudados para o cálculo-π são sistemas bem conhecidos em outras linguages sequenciais, especialmente o cálculo-λ. Porém nos últimos anos vem sendo propostos um grande número de sistemas de tipos específicos para processos, tais como sistema de tipos para detectar *deadlocks* (KOBAYASHI, 1998), *livelocks* (KOBAYASHI, 2002), detectar modos de I/O (SANGIORGI; WALKER, 2001), etc.

Esta sessão apresenta tipos para o cálculo-π, através da descrição do *Base*-π que é o mais simples sistema de tipos elaborado para o cálculo introduzido por (SANGIORGI; WALKER, 2001). Posteriomente o *Base*-π será nossa base para a definição de um sistema de tipos capaz de detectar os nomes ativos em processos do cálculo-π. Antes de iniciarmos a descrição do *Base*-π faremos uma breve introdução a noções básicas de sistemas de tipos necessárias para a compreensão do cálculo-π tipado.

2.2.1 Noções Básicas

O objetivo desta seção é familiarizar o leitor com a terminologia empregada em sistemas de tipos para o cálculo-π, assim como introduzir algumas noções básicas de seu funcionamento, facilitando assim a leitura e compreensão do restante do trabalho.

No cálculo-π tipado cada nome livre contido em um processo precisa receber um apontamento de tipo. O apontamento de um tipo para o nome será sempre da forma $a : T$, onde a é um nome, e T é um tipo (lê-se a é do tipo T). Desta forma definimos um ambiente de tipos como:

Definição 5 *Um ambiente de tipos é um conjunto finito de apontamentos de tipos para nomes, onde os nomes nos apontamentos são todos diferentes.* ■

Por exemplo, o conjunto de apontamentos $\{a : T, b : S, c : T\}$ é um ambiente de tipos válido.

Um ambiente de tipos pode ser visto como uma função finita de nomes para tipos, o que possibilita escrever $\Gamma(a)$ para representar o tipo apontado para a no ambiente Γ. Também pode-se dizer que os nomes dos apontamentos em Γ são os nomes para os quais Γ está definido. A notação $\Gamma, x : T$ representa o ambiente de tipo $\Gamma \cup \{x : T\}$.

Outra noção importante para entender o cálculo-π tipado são os julgamentos de tipos. Os julgamentos de tipos serão sempre na forma $\Gamma \vdash E : T$, onde E pode ser um processo ou um nome e T é o seu tipo. Se E for um processo então T será obrigatoriamente $\diamond$ que é o tipo comportamento. Um julgamento de tipo $\Gamma \vdash P : \diamond$ diz que o processo P é bem tipado em relação ao ambiente de tipos Γ; já um julgamento de tipo $\Gamma \vdash a : T$ diz que, o nome a tem o tipo T no ambiente Γ.

Dado um sistema de tipos, um julgamento de tipo válido é aquele que pode ser provado através dos axiomas e regras de inferência do sistema de tipos. Uma expressão E é bem tipada sob Γ, se existir um tipo T tal que o julgamento $\Gamma \vdash E : T$ seja válido.

Em cálculos tipados em geral (como o CCS tipado), os processos são formados por sujeitos e objetos. Os sujeitos são os canais de comunicação entre os processos e são do tipo *link*, já os objetos são os valores que são trocados entre os processos e que são do tipo *valor* (estes valores podem ser números, por exemplo). No cálculo-π tipado esta relação também é mantida porém com uma pequena diferença. No cálculo-π os sujeitos, que são do tipo *link*, também estão incluídos no tipo *valor*, isto é, no cálculo-π tipado o tipo *link* é um subtipo de *valor*. Para melhor entendimento considere o exemplo abaixo.

Exemplo 3 *Se $P = a(x).\overline{x}c.\overline{b}d.0$ então, considerando apenas o escopo de P teremos os seguintes apontamentos de tipos:*

- *a e b são sujeitos e são do tipo link. Eles servem apenas como canal de comunicação.*
- *c e d são objetos e são do tipo valor. Eles não servem como canal de comunicação, podem ser simplemente valores númericos ou qualquer outro tipo.*
- *x é um sujeito e é do tipo valor, pois ele é usado como uma referência para substituições. Note que no prefixo $\overline{x}c$ ele é usado como um canal de comunicação, por este motivo dizemos que no cálculo-π o tipo link é um subtipo de valor.*

■

Note que esta característica torna possível a mobilidade entre os processos, pois como o tipo *link* é um subtipo de *valor*, isto possibilita a troca de nomes entre os processos.

2.2.2 Base-π

O *Base-*π, apresentado em (SANGIORGI; WALKER, 2001), é a mais simples definição de um cálculo-π tipado. Suas regras de tipo e operacionais foram construídas de forma simples, possibilitando assim sua extensão para sistemas de tipos mais elaborados. Neste trabalho nós utilizamos o *Base-*π como base para elaborar um sistema de tipos capaz de detectar nomes ativos em agentes-π (Capítulo 4).

Tipos

S, T	::=	V	tipo *valor*
	$\mid$	L	tipo *link*
	$\mid$	$\diamond$	tipo *comportamento*
V	::=	B	tipo *básico*
	$\mid$	L	tipo *link*
L	::=	$\#V$	tipo *conexão*

Ambiente de Tipos

Γ	::=	$\Gamma, x : L$	
	$\mid$	$\Gamma, x : V$	
	$\mid$	$\emptyset$	

Valores

a, b	::=	x	nome
	$\mid$	$basval$	valor básico

Processos

P, Q	::=	$(\nu x : L)P$	restrição
	$\mid$	...	*

Prefixos

α	::=	$a(x)$	entrada
	$\mid$	$\overline{a}b$	saída
	$\mid$	τ	silencioso

* Os demais processos são definidos tais como os construtores definidos na Figura 2.1

Figura 2.4: *Base-*π - Sintaxe de tipos.

A Figura 2.4 mostra a sintaxe e a linguagem de tipos do *Base-*π. Note que no *Base-*π a distinção sintática entre tipo *valor* e tipo *link* é feita pelo uso das metavariáveis V e L, para agrupar os tipos valores e os tipos links respectivamente. Contudo para a definição das regras de tipos são utilizadas as metavariáveis S e T que significam tipos arbitrários. Assim, em um julgamento de tipo escreve-se $\Gamma \vdash a : T$ ao invés de $\Gamma \vdash a : V$, pois a sintaxe de tipos garante que T pode ser um tipo *valor*.

Em *Base-*π há somente um construtor para o tipo *link*, que é o construtor *conexão* ($\#$). Desta forma um apontamento na forma $a : \#T$ significa que a pode ser usado como um canal de comunicação para carregar valores do tipo T. Note que é o fato do tipo *link* ser subtipo do tipo *valor*, que torna possível a mobilidade entre processos, pois isso possibilita um apontamento na forma $a : \#\#T$ por exemplo, o que significa que a é um canal de comunicação e pode carregar um outro canal de comunicação, que por sua vez pode carregar um valor do tipo T.

O tipo *valor* também pode ser um tipo básico especificado pela metavariável B. Os valores deste tipo são os *valores básicos* ($basval$) e são valores que podem ser trocados entre os processos, mas que não podem ser usados como canal de comunicação. O *Base-*π não especifica o conjunto B (que poderia ser formado por inteiros, booleanos, etc.), pois não há necessidade de um detalhamento maior para o sistema de tipos proposto. Por

exemplo, se B fosse definido como um tipo para inteiros, então haveria a necessidade de criar operações que manipulasem valores inteiros.

A sintaxe dos agentes para o *Base*-π da Figura 2.4, é semelhante a sintaxe apresentada na Figura 2.1. A única diferença em relação a sintaxe da Figura 2.1, é que no construtor de restrição aponta-se um tipo para o nome a ser restrito.

Regras para Valores

$$\frac{}{\Gamma \vdash basval : B}\ \text{(TV-BASE)} \qquad \frac{}{\Gamma, x : T \vdash x : T}\ \text{(TV-NAME)}$$

Regras para Processos

$$\frac{}{\Gamma \vdash 0 : \diamond}\ \text{(T-NIL)} \qquad \frac{\Gamma \vdash P : \diamond}{\Gamma \vdash \tau.P : \diamond}\ \text{(T-TAU)}$$

$$\frac{\Gamma \vdash a : \#T \qquad \Gamma, x : T \vdash P : \diamond}{\Gamma \vdash a(x).P : \diamond}\ \text{(T-INP)}$$

$$\frac{\Gamma \vdash a : \#T \qquad \Gamma \vdash b : T \qquad \Gamma \vdash P : \diamond}{\Gamma \vdash \overline{a}b.P : \diamond}\ \text{(T-OUT)}$$

$$\frac{\Gamma \vdash P : \diamond \qquad \Gamma \vdash Q : \diamond}{\Gamma \vdash P + Q : \diamond}\ \text{(T-SUM)} \qquad \frac{\Gamma \vdash P : \diamond \qquad \Gamma \vdash Q : \diamond}{\Gamma \vdash P \mid Q : \diamond}\ \text{(T-PAR)}$$

$$\frac{\Gamma \vdash a : \#T \qquad \Gamma \vdash b : \#T \qquad \Gamma \vdash P : \diamond}{\Gamma \vdash [a = b]P : \diamond}\ \text{(T-MAT)}$$

$$\frac{\Gamma, x : L \vdash P : \diamond}{\Gamma \vdash (\nu x : L)P : \diamond}\ \text{(T-RES)} \qquad \frac{\Gamma \vdash P : \diamond}{\Gamma \vdash !P : \diamond}\ \text{(T-REP)}$$

Figura 2.5: *Base*-π - Regras de tipos para o *Base*-π.

As regras de tipos do *Base*-π são definidas na Figura 2.5. Abaixo segue uma breve explicação do significado de cada regra:

- **TV-BASE** - um valor básico $basval$ será sempre bem tipado.
- **TV-NAME** - de forma similar a regra TV-BASE esta regra diz que um nome é do tipo T, se ele estiver no ambiente de tipos Γ associado ao tipo T.
- **T-NILL** - esta regra diz que o processo 0 (vazio) é bem tipado em qualquer ambiente de tipos.
- **T-TAU** - se um processo P qualquer é bem tipado no ambiente Γ, então pode-se afirmar que um processo na forma $\tau.P$ também será bem tipado em Γ.

- **T-INP** - para uma entrada $a(x).P$, o sujeito a deve ser do tipo *link* e consequentemente um nome. Além disso um tipo para x é determinado e então o corpo P deve ser bem tipado sobre o resultado da união de Γ com o apontamento escolhido para x. Se estas premissas forem respeitadas pode-se dizer que $a(x).P$ é um processo bem tipado.

- **T-OUT** - no caso de uma saída $\overline{a}b.P$, o sujeito a tem que ser do tipo *link* e este tipo deve ser compatível com o tipo de b. Isto quer dizer que o nome a deve ser um link capaz de carregar um nome do tipo apontado para b. Além disso o corpo P deve ser bem tipado sobre Γ.

- **T-SUM** - esta regra diz respeito a soma de processos. Se dois processos P e Q são bem tipados sob Γ, então pode-se afirmar que a soma destes dois processos, também será bem tipada em Γ.

- **T-PAR** - esta regra tem a mesma intuição que a soma de processos, porém aqui se dois processos P e Q são bem tipados sob Γ, então afirma-se que a composição $P|Q$ é bem tipada em Γ.

- **T-MAT** - esta regra verifica se um processo com operador de igualdade (matching) é bem tipado. Para isso é necessário verificar se dois nomes a e b são bem tipados e se eles são do mesmo tipo *link*. Caso estas premissas sejam verdadeiras e se P é bem tipado sobre Γ, então o processo $[a = b]P$ também será bem tipado.

- **T-RES** - um processo com construtor de restrição será bem tipado, se seu corpo P observar todas as conformidades impostas pelo ambiente de tipos Γ e também pelo tipo do novo nome declarado na restrição. A intuição aqui é que o operador de restrição declara um novo nome que deve ser adicionado ao ambiente Γ. Por este motivo, se P é bem tipado sobre Γ unido ao novo nome, então o processo com construtor de restrição também será bem tipado.

- **T-REP** - sempre que uma única cópia de P for bem tipada sobre Γ, então pode-se afirmar que a replicação $!P$ também será bem tipada sobre Γ.

A seguir mostramos alguns exemplos da aplicação das regras para verificar se um processo é bem tipado sobre um determinado ambiente de tipos contendo apontamentos de tipos para os nomes livres dos processos.

Exemplo 4 *Demonstração de que o processo* $P = a(x).\overline{x}b.0$ *é bem tipado sobre o ambiente* $\Gamma = \{a : \#\#T, b : T\}$.

$$\dfrac{\dfrac{}{\Gamma \vdash a : \#\#T}\ \textit{TV-NAME} \qquad \dfrac{\dfrac{}{\Gamma' \vdash x : \#T}\ \textit{TV-NAME} \quad \dfrac{}{\Gamma' \vdash b : T}\ \textit{TV-NAME} \quad \dfrac{}{\Gamma' \vdash 0 : \diamond}\ \textit{T-NILL}}{\Gamma, x : \#T \vdash \overline{x}b.0 : \diamond}\ \textit{T-OUT}}{\Gamma \vdash a(x).\overline{x}b.0 : \diamond}\ \textit{T-INP}$$

$Onde\ \Gamma' = \Gamma, x : \#T$

■

Para um melhor entendimento a leitura da inferência deve ser feita de baixo para cima. Note que queremos provar que o processo $a(x).\overline{x}b.0$, que é um processo com prefixo de entrada, é bem tipado sob Γ. Portanto a regra para este tipo de processo é T-INP que tem duas premissas.

Na primeira premissa de T-INP temos que verificar se o nome a é bem tipado sobre Γ. Assim verificamos se o apontamento $a : \#\#T$ pertence a Γ (o que é verdadeiro). Na segunda premissa de T-INP deve-se mostrar que o subprocesso $\overline{x}b.0$ é bem tipado sob o ambiente Γ unido ao nome x que é o objeto do prefixo $a(x)$. Note que adicionamos o nome x a Γ apontando para ele um tipo coerente com o tipo de a, ou seja, a está declarado para carregar um valor do tipo $\#T$, portanto x deverá ser adicionado com este tipo em Γ.

Para verificar se o subprocesso $\overline{x}b.0$ é bem tipado deve-se aplicar a regra T-OUT. Esta regra possui três premissas que são verificadas sob o ambiente Γ', que é o resultado da união de Γ com o apontamento $x : \#T$. Nas primeiras duas premissas é verificado se os nomes x e b estão declarados corretamente em Γ'. Já a última premissa verifica se o processo 0 é bem tipado sob Γ'. Para o processo 0 é aplicada a regra T-NIL que diz que um processo vazio 0 é sempre bem tipado.

A seguir mostramos mais dois exemplos onde utilizamos as regras T-RES e T-PAR.

Exemplo 5 *Prova de que o processo* $P = (\nu b : \#T)a(x).\overline{x}b.\overline{b}c.0$ *é bem tipado sobre o ambiente* $\Gamma = \{a : \#\#\#T, c : T\}$.
Derivação 1:

$$\dfrac{\dfrac{\dfrac{}{\Gamma' \vdash a : \#\#\#T}\ \textit{TV-NAME} \quad \dfrac{\dfrac{}{\Gamma'' \vdash x : \#\#T}\ \textit{TV-NAME} \quad \dfrac{}{\Gamma'' \vdash b : \#T}\ \textit{TV-NAME} \quad \textit{Derivação 2}}{\Gamma', x : \#\#T \vdash \overline{x}b.\overline{b}c.0 : \diamond}\ \textit{T-OUT}}{\Gamma, b : \#T \vdash a(x).\overline{x}b.\overline{b}c.0 : \diamond}\ \textit{T-INP}}{\Gamma \vdash (\nu b : \#T)a(x).\overline{x}b.\overline{b}c.0 : \diamond}\ \textit{T-RES}$$

Derivação 2:

$$\dfrac{\dfrac{}{\Gamma'' \vdash b : \#T}\ \textit{TV-NAME} \quad \dfrac{}{\Gamma'' \vdash c : T}\ \textit{TV-NAME} \quad \dfrac{}{\Gamma'' \vdash 0 : \diamond}\ \textit{T-NIL}}{\Gamma'' \vdash \overline{b}c.0 : \diamond}\ \textit{T-OUT}$$

$$\begin{aligned} Onde \quad : \quad & \Gamma' = \Gamma, b : \#T \\ & \Gamma'' = \Gamma', x : \#\#T \end{aligned}$$

■

Exemplo 6 *Prova de que o processo* $P = (\nu y : \#\#T)(\nu z : \#T)(\overline{a}b.y(x).\overline{x}c.0 | \overline{y}z.d(u).u(w).0)$ *é bem tipado sobre o ambiente* $\Gamma = \{a : \#T, b : T, c : T, d : \#\#T\}$.
Derivação 1:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma'' \vdash a : \#T, b : T}\ \textit{TV-NAME} \quad \textit{Derivação 2}}{\Gamma'' \vdash \overline{a}b.y(x).\overline{x}c.0 : \diamond}\ \textit{T-OUT} \quad \dfrac{\dfrac{}{\Gamma'' \vdash y : \#\#T, z : \#T}\ \textit{TV-NAME} \quad \textit{Derivação 3}}{\Gamma'' \vdash \overline{y}z.d(u).u(w).0 : \diamond}\ \textit{T-OUT}}{\Gamma', z : \#T \vdash (\overline{a}b.y(x).\overline{x}c.0 \mid \overline{y}z.d(u).u(w).0) : \diamond}\ \textit{T-PAR}}{\Gamma, y : \#\#T \vdash (\nu z : \#T)(\overline{a}b.y(x).\overline{x}c.0 \mid \overline{y}z.d(u).u(w).0) : \diamond}\ \textit{T-RES}}{\Gamma \vdash (\nu y : \#\#T)(\nu z : \#T)(\overline{a}b.y(x).\overline{x}c.0 \mid \overline{y}z.d(u).u(w).0) : \diamond}\ \textit{T-RES}$$

Derivação 2:

$$\dfrac{\dfrac{}{\Gamma'' \vdash y : \#\#T}\ \textit{TV-NAME} \quad \dfrac{\dfrac{}{\Gamma''' \vdash x : \#T}\ \textit{TV-NAME} \quad \dfrac{}{\Gamma''' \vdash c : T}\ \textit{TV-NAME} \quad \dfrac{}{\Gamma''' \vdash 0 : \diamond}\ \textit{T-NIL}}{\Gamma'', x : \#T \vdash \overline{x}c.0 : \diamond}\ \textit{T-OUT}}{\Gamma'' \vdash y(x).\overline{x}c.0 : \diamond}\ \textit{T-INP}$$

Derivação 3:

$$\dfrac{\dfrac{}{\Gamma'' \vdash d : \#\#T}\ \textit{TV-NAME} \quad \dfrac{\dfrac{}{\Gamma'''' \vdash u : \#T}\ \textit{TV-NAME} \quad \dfrac{}{\Gamma'''', w : T \vdash 0 : \diamond}\ \textit{T-NIL}}{\Gamma'', u : \#T \vdash u(w).0 : \diamond}\ \textit{T-INP}}{\Gamma'' \vdash d(u).u(w).0 : \diamond}\ \textit{T-INP}$$

$$\begin{aligned} Onde \ : \ & \Gamma' = \Gamma, y : \#\#T \\ & \Gamma'' = \Gamma', z : \#T \\ & \Gamma''' = \Gamma'', x : \#T \\ & \Gamma'''' = \Gamma'', u : \#T \end{aligned}$$

■

Note no Exemplo 6 que simplificamos as regras T-OUT da derivação 1 para diminuir o tamanho da derivação. Ao invés de utilizarmos duas regras TV-NAME para avaliar as duas primeiras premissas da regra T-OUT, unimos estas premissas em uma única regra TV-NAME.

Além das regras de tipos apresentadas na Figura 2.5, o *Base*-π também possue regras operacionais, que são baseadas na semântica operacional do cálculo-π. Em nosso trabalho não abordaremos estas regras, pois nosso interesse aqui está na estrutura sintática dos processos, que são verificadas nas regras de tipos. Desta forma maiores informações sobre as regras operacionais do *Base*-π podem ser encontradas em (SANGIORGI; WALKER, 2001).

3 NOMES ATIVOS

Este capítulo apresenta a caracterização semântica de nomes ativos em agentes do cálculo-π e porque a sua detecção pode ajudar na verificação de equivalência entre agentes-π. Também apresenta ao final, duas abordagens utilizadas para detecção de nomes ativos que serviram de inspiração para a realização deste trabalho.

3.1 π-autômatos

Para o restante do trabalho é importante que o leitor conheça e saiba como funciona a construção dos π-autômatos que são muito utilizados por algoritmos de verificação de equivalência. Por este motivo antes de iniciarmos a caracterização dos nomes ativos vamos realizar uma pequena descrição de como montar os π-autômatos.

Segundo Dam (DAM, 1997) à partir de uma agente-π com um conjunto de nomes finito, é possível construir um autômato finito capaz de representar o comportamento deste agente. Estes autômatos são conhecidos como π-autômatos:

Definição 6 *Um π-autômato é uma tupla $\langle Q, q_0, L, T\rangle$ onde:*

- *Q é um conjunto contável de estados, onde os estados são expressões do cálculo-π.*
- *q_0 é o estado inicial.*
- *L é um conjunto contável de rótulos. Os rótulos do π-autômato são as ações de entrada, de saída livre, de saída ligada e silenciosa do cálculo-π.*
- *$T \subseteq Q \times L \times Q$ é um conjunto de transições. Para uma transição $\langle q, l, q'\rangle \in T$ nós escrevemos $q \xrightarrow{l} q'$. Estas transições são definidas pela semântica operacional do cálculo-π mostrada na Figura 2.3.*

■

Nós dizemos que a partir de dois π-autômatos $A \stackrel{def}{=} \langle Q_A, q_0^A, L, T\rangle$ e $B \stackrel{def}{=} \langle Q_B, q_0^B, L, T\rangle$, uma relação $\mathcal{R} \subset Q_A \times Q_B$ será uma simulação se:

- para cada relação $a\mathcal{R}b$ e transição $a \xrightarrow{l} a'$, existirá uma transição $b \xrightarrow{l} b'$ de tal forma que $a'\mathcal{R}b'$.

E assim podemos dizer que uma relação $\mathcal{R} \subset Q_A \times Q_B$ será uma bisimulação, se e somente se, $\mathcal{R}$ e $\mathcal{R}^{-1}$ são simulações, ou seja, se o π-autômato A simula B e vice-versa.

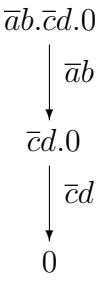
Para construir um π-autômato devemos considerar o comportamento do agente e criar seus estados e transições conforme a semântica operacional definida no Capítulo 2. Para um melhor entendimento considere o exemplo abaixo.

Exemplo 7 *Considere o agente* $P \stackrel{def}{=} \overline{a}b.\overline{c}d.0$*. Este agente executa uma ação de saída* $\overline{a}b$ *e evolui para* $\overline{c}d.0$*. Pela semântica operacional de uma transição de saída (regra OUT da Figura 2.3) devemos ter uma transição de um estado* $\overline{a}b.\overline{c}d.0$ *para o estado* $\overline{c}d.0$ *rotulado pela ação* $\overline{a}b$*.*

A partir do estado $\overline{c}d.0$ *o agente pode executar a ação de saída* $\overline{c}d$ *evoluindo para* 0*. Assim o* π*-autômato correspondente ao processo* P *é definido pelos seguintes conjuntos:*

- $Q = \{\overline{a}b.\overline{c}d.0, \overline{c}d.0, 0\}$
- $L = \{\overline{a}b, \overline{c}d\}$
- $T = \{\overline{a}b.\overline{c}d.0 \stackrel{\overline{a}b}{\rightarrow} \overline{c}d.0, \overline{c}d.0 \stackrel{\overline{c}d}{\rightarrow} 0\}$
- $q_0 = \overline{a}b.\overline{c}d.0$

Também podemos representar este π*-autômato graficamente como demonstrado abaixo:*



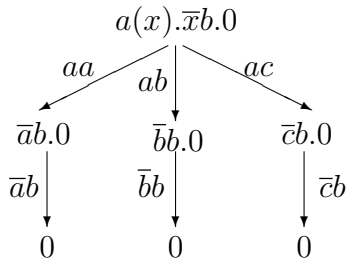
■

Note que na regra INP da semântica operacional da Figura 2.3, o nome x é substituído por um nome que o agente está recebendo por a. Qualquer nome pode ser recebido nesta transição, portanto é necessário utilizar alguma estratégia para escolher estes nomes, pois caso contrário não conseguiriamos construir autômatos finitos.

Pensando assim Montanari e Pistore propõem utilizar os nomes livres do estado corrente para realizar uma transição de entrada, mais um novo nome ainda não utilizado no agente, que representa o restante dos nomes que poderiam ser utilizados como entrada no agente. Assim para um estado $a(x).\overline{x}b.0$ existem 3 transições. Duas utilizando os nomes livres do estado $\{a, b\}$ e uma usando um novo nome ainda não utilizado no estado, por exemplo c. O Exemplo 8 mostra a construção completa deste π-autômato.

Exemplo 8 *Considere o agente* $P = a(x).\overline{x}b.0$*. Para este agente teremos a seguinte definição para o* π*-autômato:*

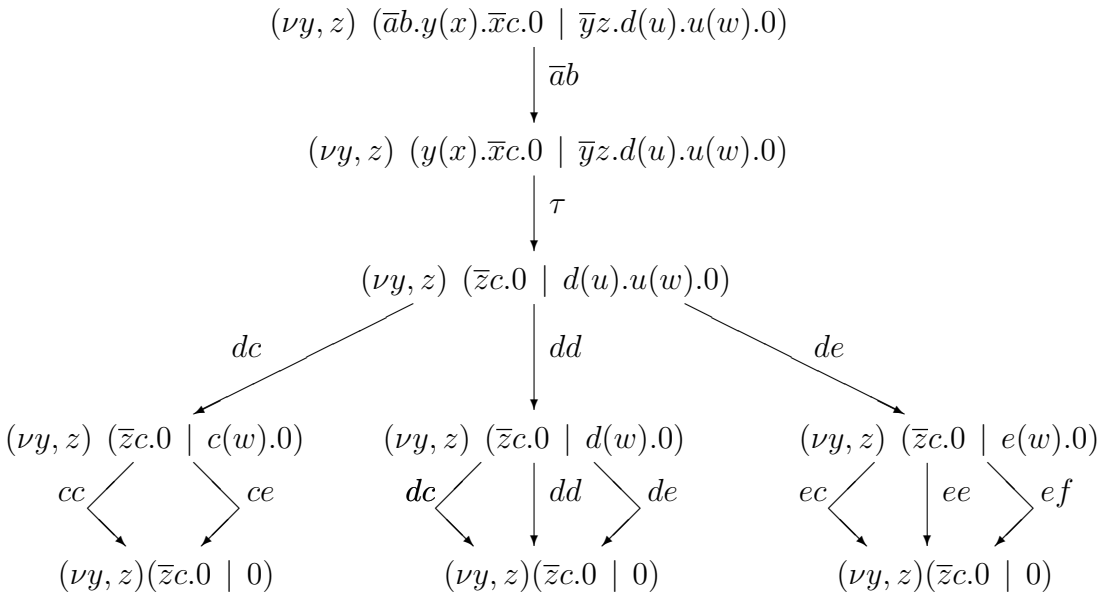
- $Q = \{a(x).\overline{x}b.0\ ,\ \overline{a}b.0\ ,\ \overline{b}b.0\ ,\ \overline{c}b.0\ ,\ 0\}$
- $L = \{aa, ab, ac, \overline{a}b, \overline{b}b, \overline{c}b\}$
- $T = \{a(x).\overline{x}b.0 \stackrel{aa}{\rightarrow} \overline{a}b.0\ ,\ a(x).\overline{x}b.0 \stackrel{ab}{\rightarrow} \overline{b}b.0\ ,\ a(x).\overline{x}b.0 \stackrel{ac}{\rightarrow} \overline{c}b.0\ ,\ \overline{a}b.0 \stackrel{\overline{a}b}{\rightarrow} 0\ ,$ $\overline{b}b.0 \stackrel{\overline{b}b}{\rightarrow} 0\ ,\ \overline{c}b.0 \stackrel{\overline{c}b}{\rightarrow} 0\}$
- $q_0 = a(x).\overline{x}b.0$



■

Note neste exemplo que no estado inicial $a(x).\overline{x}b.0$ temos os nomes a e b livres, e estes são utilizados na transição de entrada $a(x)$ como substitutos de x. Também perceba que existe uma terceira transição à partir do estado inicial (transição ac), onde utiliza-se um novo nome ainda não usado no π-autômato (neste caso c), para representar o restante dos nomes que podem ser utilizados por uma transição de entrada.

Exemplo 9 *Para o agente* $P = (\nu y, z)(\overline{a}b.y(x).\overline{x}c.0 \mid \overline{y}z.d(u).u(w).0)$*, obtemos o seguinte autômato:*



■

Observe neste exemplo que, à partir do estado $(\nu y, z)(\overline{z}c.0|0)$, não é possivel executar a ação $\overline{z}c$. Neste caso o nome b está restrito e portanto não pode ser executado terminando aqui a evolução do autômato.

Agora estamos prontos para caracterizar os nomes ativos e para isso utilizaremos nos próximos exemplos somente a representação gráfica dos π-autômatos. Além disso para simplificar a construção dos π-autômatos vamos usar apenas os nomes livres para realizar as transições de entrada. Por este motivo nos próximos exemplos deve-se levar em consideração, que ao se encontrar uma ação de entrada, haverá uma outra transição com um novo nome não usado no agente, além das transições com os nomes livres descritas no π-autômato. Esta simplificação ficará mais clara nos exemplos mostrados na próxima Seção.

3.2 Caracterização de Nomes Ativos

A proposta de coleta de nomes ativos para agentes do cálculo-π foi apresentada por Montari e Pistore (MONTANARI; PISTORE, 1995) baseada na idéia de **nomes usados**, introduzida por (JONSSON; PARROW, 1992) para o CCS com passagem de valor.

Montanari e Pistore (MONTANARI; PISTORE, 1995) provam em seu trabalho que agentes bisimilares tem o mesmo conjunto de nomes ativos. Desta forma ações compostas por nomes inativos, aqui chamadas de ações inativas, poderiam ser eliminadas do agente sem interferir no seu comportamento, possibilitando assim uma diminuição no conjunto de estados necessários para a prova da equivalência. Após este trabalho um algoritmo de refinamento de partição, baseado na coleta de nomes ativos, foi proposto por Pistore e Sangiorgi (PISTORE; SANGIORGI, 1996), (SANGIORGI, 1998), para a verificação de equivalência entre agentes-π.

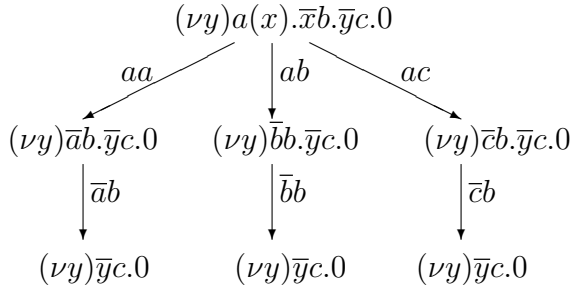
Para Montanari e Pistore um nome é semanticamente ativo em um agente P se ele pode ser executado de forma observável por P. Note assim que as ações com um canal de comunicação restrito não podem ser executadas (a não ser que ocorra uma comunicação interna), e portanto todos os nomes envolvidos na ação não interferem no comportamento observável do agente. Por exemplo, o nome a é ativo no agente $a(b).0$, mas não é ativo no agente $(\nu b)\overline{b}a.0$, pois não pode ser executado já que o nome b está restrito e esta ação não pode ocorrer isoladamente.

Definição 7 *Um nome a é ativo para um agente P se e somente se $P \nsim (\nu a)P$ (MONTANARI; PISTORE, 1995).* ■

Conforme a Definição acima um nome será ativo se, ao restringirmos ele no agente, obtém-se um novo agente que tem um comportamento diferente do agente original sem a restrição, ou seja, o novo agente não é bisimilar ao original. Portanto fica claro que os nomes ativos de um agente formam o menor subconjunto de nomes livres que afetam o comportamento do agente, pois somente nomes livres podem ser executados e podem interferir no comportamento. Já um nome inativo não é capaz de mudar o comportamento do agente do *ponto de vista externo*.

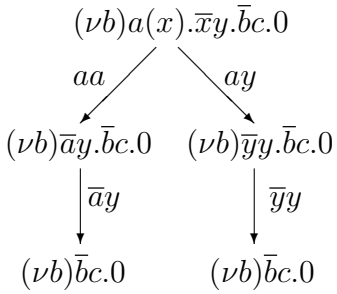
A seguir mostramos um exemplo simples da caracterização de nomes ativos sobre agentes-π e como a coleta de tais nomes pode ajudar na construção dos π-autômatos mostrados na Seção 3.1.

Exemplo 10 *Para o agente $P = (\nu y)a(x).\overline{x}b.\overline{y}c.0$ obtemos o seguinte π-automato, considerando $fn(P) = \{a, b, c\}$:*



Aqui, intuitivamente podemos perceber que o nome c, apesar de ser um nome livre, não pode ser executado em P pois o nome y é um canal restrito, e por isso a ação $\overline{y}c$ nunca será executada. Desta forma dizemos que c é um nome inativo e assim o conjunto

de nomes ativos de P é {a,b}. Sendo assim podemos construir o π-autômato excluindo todas as transições onde o nome c é utilizado, diminuindo o número de estados do π-autômato, como é mostrado abaixo:



■

Note no Exemplo 10 que à partir do estado inicial $(\nu b)a(x).\overline{x}y.\overline{b}c.0$ deveria ser adicionada mais uma transição com um novo nome ainda não usado no agente, por exemplo, uma transição rotulada como ad levando ao estado $(\nu b)\overline{d}y.\overline{b}c.0$. Aqui para simplificar o π-autômato descartamos esta transição dos dois autômatos mostrados no exemplo. Isto porque a coleta dos nomes ativos irá afetar transições de entrada que usam os nomes livres. Assim preferimos eliminar as transições com novos nomes para mostrar melhor a simplificação no π-autômato gerada pela coleta dos nomes ativos.

Neste trabalho o objetivo principal é detectar os nomes ativos de *processos* do cálculo-π. Para isso é necessário primeiro caracterizar os nomes ativos e inativos de ações que são executadas por processos. A Figura 3.1 apresenta os nomes ativos e inativos de cada uma das possíveis ações executadas por processos-π.

Ação	Ativos	Inativos
τ	{}	{}
$\overline{a}b$	{a, b}	{}
$\overline{a}\nu b$	{a}	{b}
ab	{a}	{b}

Figura 3.1: Caracterização de Nomes Ativos de Ações.

Note que a ação $\overline{a}\nu b$ da Figura 3.1 não faz parte da sintaxe do cálculo-π definida no Capítulo 2. Ela representa a ação associada ao prefixo de saída $\overline{a}b$ onde o nome b enviado é restrito ao escopo do agente ($(\nu b)\overline{a}b$), caracterizando assim a extrusão do escopo de b.

Intuitivamente as ações mostradas na Figura 3.1 representam os quatro tipos ações que podem acontecer na execução de um agente-π, ou seja, um agente pode executar uma ação de entrada ab, ou executar uma ação de saída $\overline{a}b$, ou uma ação de saída ligada $\overline{a}\nu b$ ou ainda executar uma comunicação interna τ.

Note nas ações da Figura 3.1 que todos os nomes ligados são inativos, isto porque, somente os nomes livres devem ser levados em consideração na verificação de equivalência. Por isso em agentes da forma $a(b).P$ e $(\nu b)\overline{a}b.P$, as ocorrências de b são inativas. Note também que nos agentes onde o canal de comunicação é restrito (por exemplo, $(\nu a)a(b).P$), este nome também será inativo pois será ligado.

A seguir mostraremos as abordagens para coleta de nomes ativos de (MONTANARI; PISTORE, 1995) e (MELO, 2003), que foram utilizadas como base para a elaboração deste trabalho.

3.3 Abordagem de Montanari e Pistore

Montanari e Pistore definiram um algoritmo para identificar o conjunto de nomes ativos utilizando π-autômatos. Primeiramente o algoritmo identifica todos os nomes livres contidos no agente-π, e então cria um π-*autômato* (como mostrado na Seção 3.1) utilizando somente estes nomes livres. Porém aqui eles adicionam ao rótulo das transições a função de substituição σ e o rótulo da transição permanece com os mesmos nomes do prefixo. A Figura 3.2, mostra o π-autômato gerado pelo algoritmo para o agente P do Exemplo 10.

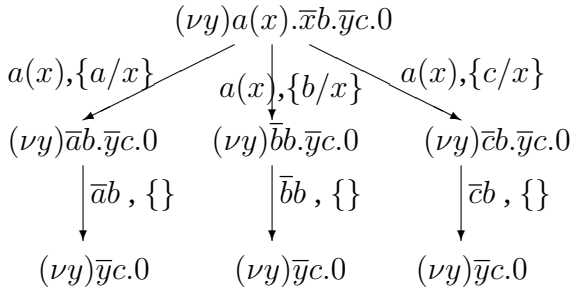


Figura 3.2: π-**autômato** com nomes livres gerado por Montanari e Pistore.

O segundo passo é identificar os nomes ativos para cada transição do π-*autômato,* utilizando a caracterização de nomes ativos da Figura 3.1. O algoritmo define que um nome será ativo sempre que ele aparecer ativo na transição atual, ou sempre que ele se torna ativo em um estado seguinte do π-autômato e não aparece como inativo na transição atual. Assim o conjunto de ativos do agente P na Figura 3.2 será a união dos ativos encontrados nas três transições que partem do primeiro estado do π-autômato.

- Na primeira transição $a(x), \{a/x\}$ temos a como um nome ativo pela Figura 3.1 e x como inativo. Na transição seguinte $\overline{a}b, \{\}$ os dois nomes serão ativos pela Figura 3.1 e como a substituição $\{a/x\}$ não influência no nome b, pode-se dizer que os nomes ativos da transição $a(x), \{a/x\}$ serão a e b que foi herdado do estado seguinte da transição.

- A segunda transição $a(x), \{b/x\}$ como na primeira tem o nome a como ativo e na transição seguinte $\overline{b}b, \{\}$ encontra-se o nome b como ativo. Porém aqui b não entra no conjunto de ativos da transição $a(x), \{b/x\}$, pois deve-se levar em consideração a substituição da transição, ou seja, o nome b substitui o nome x que é inativo na transição $a(x), \{b/x\}$. Assim nesta transição somente o nome a será ativo.

- Na última transição $a(x), \{c/x\}$ o nome a é ativo e na transição seguinte $\overline{c}b, \{\}$ encontra-se os nomes c e b como ativos. Porém aqui o nome c substitui o nome x na transição $a(x), \{c/x\}$ que é inativo, por isso c deve ser excluído do conjunto de ativos desta transição. Desta forma na transição $a(x), \{c/x\}$ encontra-se os nomes a e b como ativos, e unindo o resultado das três transições que partem do primeiro estado do π-autômato obtem-se o conjunto de ativos {a,b}.

Após encontrar os nomes ativos do agente o algoritmo de Montanari e Pistore gera um novo autômato que pode ser utilizado por um algoritmo de verificação de equivalência.

Neste novo autômato, em cada estado será aplicada a substituição σ e os nomes inativos serão restritos. Este autômato é conhecido como *unwinding-automata*. Ilustramos na Figura 3.3 o *unwinding-automata* gerado à partir do π-autômato da Figura 3.2.

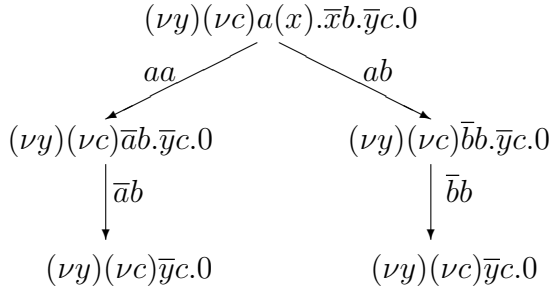


Figura 3.3: **unwinding-automata** gerado por Montanari e Pistore.

Observe a redução do número de estados do autômato da Figura 3.2 para o autômato da Figura 3.3. De acordo com Montanari e Pistore as provas de propriedades relativas a comportamento observável do processo $(\nu y)a(x).\overline{x}b.\overline{y}c.0$ podem ser feitas à partir do processo $(\nu y)(\nu c)a(x).\overline{x}b.\overline{y}c.0$, pois ambos os processos são observacionalmente equivalentes.

3.4 Abordagem de Ana Melo

O algoritmo apresentando em (MELO, 2003) propõem a identificação de nomes ativos baseada na caracterização sintática de nomes ativos sobre expressões de agentes-π. Ao invés de gerar um π-autômato (semântica), e sobre este calcular os nomes ativos, como proposto por Montanari e Pistore, o algoritmo proposto calcula os nomes ativos através da análise da estrutura sintática de expressões de agentes-π (sintaxe). Para isso é definida uma série de equações para calcular os nomes ativos, levando em consideração a estrutura sintática dos agentes-π (conforme a Figura 2.1) e a caracterização sintática de nomes ativos da Figura 3.1.

Por exemplo, o agente P do Exemplo 10 é um agente na forma prefixa e com operador de restrição. Para este tipo de agentes Ana Melo define a seguinte equação para coleta dos nomes ativos:

Equação 1 $Q \stackrel{def}{=} (\nu\ \vec{x})\alpha.P$

$$an(Q) = \begin{cases} \{\} & se\ ch(\alpha) \in \vec{x} \\ fn(\alpha) \cup (an((\nu\ \vec{x} - \{b\})P) - bn(\alpha)) & se\ \alpha = \overline{a}\nu b\ \wedge\ a \notin \vec{x} \\ fn(\alpha) \cup (an((\nu\ \vec{x})P) - bn(\alpha)) & \textit{caso contrário} \end{cases}$$

Onde $an(Q)$ significa o conjunto de nomes ativos de Q e $ch(\alpha)$ representa o sujeito da ação, isto é, o canal de comunicação de α. Já $fn(\alpha)$ e $bn(\alpha)$ siginificam os nomes livres e ligados da ação como definido no Capítulo 2. Também observe aqui que $\vec{x}$ representa o conjunto de nomes que está restrito no agente, e a ação $\overline{a}\nu b$ da segunda linha siginifica uma ação de saída ligada, onde o nome b que está sendo enviado é restrito.

Desta forma para coletar os nomes ativos do agente no Exemplo 10 teríamos que aplicar esta equação até obtermos o conjunto de ativos final, como mostramos abaixo:

Exemplo 11 $P = (\nu y)a(x).\overline{x}b.\overline{y}c.0$

$$\begin{aligned} an(P) &= fn(a(x)) \cup (an((\nu y)\overline{x}b.\overline{y}c.0) - bn(a(x))) \\ &= \{a\} \cup (an((\nu y)\overline{x}b.\overline{y}c.0) - \{x\}) \\ &= \{a\} \cup ((fn(\overline{x}b) \cup (an((\nu y)\overline{y}c.0) - bn(\overline{x}b))) - \{x\}) \\ &= \{a\} \cup ((\{x,b\} \cup (an((\nu y)\overline{y}c.0) - \{\})) - \{x\}) \\ &= \{a\} \cup ((\{x,b\} \cup (\{\} - \{\})) - \{x\}) \\ &= \{a\} \cup (\{x,b\} - \{x\}) \\ &= \{a,b\} \end{aligned}$$

No Exemplo 11 a primeira ação $a(x)$ satisfaz a terceira opção da Equação 1, pois o nome a não é restrito e a ação não é uma saída ligada. A partir deste ponto verificamos que o processo $an((\nu y)\overline{x}b.\overline{y}c.0)$ é da forma prefixa com operador de restrição, assim aplica-se a mesma Equação 1 e novamente a terceira opção satisfaz a ação $\overline{x}b$. Por último verificamos que o processo $an((\nu y)\overline{y}c.0)$ também tem forma prefixa com operador de restrição. Porém aqui sujeito da ação $\overline{y}c$ é restrito, satisfazendo assim a primeira opção da Equação 1. Ao final obtemos o conjunto de nomes ativos do agente P.

A principal vantagem do algoritmo proposto em (MELO, 2003) é que ele pode ser aplicado tanto para abordagens sintáticas, como para abordagens semânticas de verificação de equivalência, pois a identificação é realizada sobre a estrutura das expressões-π. No caso de sistemas de re-escrita (abordagem sintática), o algoritmo de coleta de nomes ativos permitiria reduzir o tamanho das expressões dos agentes, e no caso de sistemas de transições rotuladas (abordagem semântica), não existe a necessidade de gerar um autômato com todos os nomes livres, ou seja, o autômato já seria gerado somente com os nomes ativos. Já o algoritmo proposto por (MONTANARI; PISTORE, 1995) não pode ser usado para sistemas de re-escrita, pois o algoritmo calcula os nomes ativos sobre o π-autômato de nomes livres gerado no primeiro passo. O trabalho de Melo contudo, apenas identifica o conjunto de nomes ativos, e não propõem nenhum forma de transformar a expressão original, em uma expressão sem os nomes inativos.

4 COLETA DE NOMES ATIVOS UTILIZANDO SISTEMA DE TIPOS

Como já mencionamos na introdução este é um trabalho de análise estática, que tem por objetivo coletar os possíveis nomes ativos contidos em expressões-π, utilizando para isso um sistema de tipos.

A principal vantagem de utilização de sistemas de tipos em relação a outras formas de análise estática é que sistemas de tipos são sistemas lógicos, logo todas as técnicas de prova da lógica podem ser aproveitadas no estudo de propriedades de sistemas de tipos (PALSBERG, 2001). Além disso sistemas de tipos são definidos através da estrutura sintática de expressões, facilitando assim as provas por indução estrutural.

Assim, um grande número de sistemas de tipos vem sendo propostos para analisar várias propriedades, tais como frequência com a qual canais são usados (KOBAYASHI; PIERCE; TURNER, 1999), *deadlock* (KOBAYASHI, 1998) e *livelock* (KOBAYASHI, 2002). Sistemas de tipos para o cálculo-π são apresentados em (SANGIORGI; WALKER, 2001).

Apresentamos neste capítulo o *Active-Base-*π, um o sistema de tipos para realizar a coleta de nomes ativos em expressões-π. Este sistema de tipos foi construído tomando como ponto de partida o *Base-*π apresentado na Seção 2.2 do Capítulo 2.

4.1 Active-Base-π

O *Active-Base-*π é responsável por verificar a validade de uma expressão-π e por coletar o conjunto de nomes ativos. Para construir este sistema de tipos foi utilizado como base o sistema *Base-*π apresentado na Seção 2.2.2. O *Base-*π foi escolhido para a construção do *Active-Base-*π por ser o sistema de tipos mais simples apresentado para o cálculo-π.

O primeiro passo para estender o *Base-*π é adicionar o construtor de tipo $@$ para representar nomes restritos (metavariável R). Este construtor é necessário para a identificação dos nomes restritos durante a coleta dos nomes ativos. Desta forma, um nome restrito que no *Base-*π teria um apontamento na forma $a : \#T$, como mostrado na Figura 2.4, agora, no *Active-Base-*π terá um apontamento na forma $a : @T$. Assim a notação $\#T$ no *Active-Base-*π representa um *link não restrito* capaz de carregar um valor do tipo T.

Na Figura 4.1 apresenta-se a sintaxe básica do *Active-Base-*π. Note que o construtor de restrição na sintaxe dos processos, é alterado para a forma $(\nu x : R).P$, onde x é o nome a ser restrito e R é um tipo restrito.

Veja também que tipos restritos (representados pela metavariável R) foram adicionados na gramática de tipos para valores (V), pois desta forma podemos construir agentes-π

Tipos

S, T	::=	V	tipo *valor*
	$\mid$	L	tipo *link*
	$\mid$	R	tipo *link restrito*
	$\mid$	$\diamond$	tipo *comportamento*
V	::=	B	tipo *básico*
	$\mid$	L	tipo *link*
	$\mid$	R	tipo *link restrito*
L	::=	$\#V$	tipo *link não restrito*
R	::=	$@V$	tipo *link restrito*

Ambiente de Tipos

Γ	::=	$\Gamma, x : L$
	$\mid$	$\Gamma, x : R$
	$\mid$	$\Gamma, x : V$
	$\mid$	$\emptyset$

Valores

v, w	::=	x	nome
	$\mid$	$basval$	valor básico

Processos

P, Q	::=	0	Nulo
	$\mid$	$\alpha.P$	Prefixo
	$\mid$	$P + Q$	Soma
	$\mid$	$P \mid Q$	Composição
	$\mid$	$(\nu x : R)P$	Restrição
	$\mid$	$!P$	Replicação

Prefixos

α	::=	$v(x)$	entrada
	$\mid$	$\overline{v}w$	saída
	$\mid$	τ	silencioso

Figura 4.1: *Active-Base-*π - Sintaxe de tipos.

capazes de enviar e receber nomes restritos. Esta característica, como no *Base-*π, mostra que o sistema de tipos garante a mobilidade de nomes do cálculo-π.

Note que a sintaxe básica do *Base-*π é mantida apenas acrescentando o novo tipo R, por este motivo todos os conceitos discutidos para o *Base-*π na Seção 2.2.2 são mantidos também para o *Active-Base-*π. Assim a definição do ambiente de tipos do *Active-Base-*π continua tal qual a sua definição dada no *Base-*π, apenas acrescentando agora, que o ambiente pode conter um apontamento de tipo restrito.

A próxima etapa é definir as regras de tipos para os processos. As regras definidas nas Figuras 4.2 e 4.3 seguem a mesma lógica das regras definidas para o *Base-*π na Figura 2.5. A diferença aqui é o acréscimo da coleta dos nomes ativos de um agente-π. Para isso anota-se, ao lado do julgamento de tipo de um processo, o conjunto de nomes ativos correspondente ao processo. Sendo assim um julgamento de tipo de processos passará a ser da forma:

$$\Gamma \vdash P : \diamond, A_P$$

onde $\Gamma \vdash P : \diamond$ é julgamento de tipo do processo assim como mostrado na Seção 2.2, e A_P significa o conjunto de nomes ativos do processo P.

A seguir detalhamos cada uma das regras de tipos para o *Active-Base-*π mostradas nas Figuras 4.2, para processos sem comunicação interna, e 4.3, para processos com comunicação interna. Procuramos aqui enfatizar mais a questão da coleta de nomes ativos, já que a verificação da validade das expressões permanece de forma similar a verificação do *Base-*π, já explicada na Seção 2.2.2. Começamos por explicar as regras da Figura 4.2.

Regras para Valores

$$\frac{}{\Gamma \vdash basval : B}\ \text{(TV-BASE)} \qquad \frac{}{\Gamma, x : T \vdash x : T}\ \text{(TV-NAME)}$$

Regras para Processos

$$\frac{}{\Gamma \vdash 0 : \diamond, \{\}}\ \text{(T-NIL)} \qquad \frac{\Gamma \vdash P : \diamond, A_P}{\Gamma \vdash \tau.P : \diamond, A_P}\ \text{(T-TAU)}$$

$$\frac{\Gamma \vdash a : \#T \qquad \Gamma, x : T \vdash P : \diamond, A_P}{\Gamma \vdash a(x).P : \diamond, (\{a\} \cup A_P) - \{x\}}\ \text{(T-INP}_1\text{)}$$

$$\frac{\Gamma \vdash a : @T \qquad \Gamma, x : T \vdash P : \diamond, A_P}{\Gamma \vdash a(x).P : \diamond, \{\}}\ \text{(T-INP}_2\text{)}$$

$$\frac{\Gamma \vdash a : \#T \qquad \Gamma \vdash b : T \qquad \Gamma \vdash P : \diamond, A_P \qquad \textit{T não restrito}}{\Gamma \vdash \overline{a}b.P : \diamond, \{a, b\} \cup A_P}\ \text{(T-OUT}_1\text{)}$$

$$\frac{\Gamma \vdash a : \#T \qquad \Gamma \vdash b : @T \qquad \Gamma \Downarrow b : \#T \vdash P : \diamond, A_P}{\Gamma \vdash \overline{a}b.P : \diamond, (\{a\} \cup A_P) - \{b\}}\ \text{(T-OUT}_2\text{)}$$

$$\frac{\Gamma \vdash a : @T \qquad \Gamma \vdash b : T \qquad \Gamma \vdash P : \diamond, A_P}{\Gamma \vdash \overline{a}b.P : \diamond, \{\}}\ \text{(T-OUT}_3\text{)}$$

$$\frac{\Gamma, x : R \vdash P : \diamond, A_P}{\Gamma \vdash (\nu x : R)P : \diamond, A_P}\ \text{(T-RES)} \qquad \frac{\Gamma \vdash P : \diamond, A_P \qquad \Gamma \vdash Q : \diamond, A_Q}{\Gamma \vdash P + Q : \diamond, A_P \cup A_Q}\ \text{(T-SUM)}$$

Figura 4.2: Regras de Tipos para o *Active-Base-*π.

Regra T-Nil: Como no *Base-*π um processo vazio (0) será bem tipado sobre qualquer ambiente de tipos e seu conjunto de nomes ativos será vazio, pois o processo não possui nenhum nome a coletar.

Regra T-Tau: No caso de um agente com prefixo silencioso $\tau.P$, se o processo P for bem tipado e tiver um conjunto de nomes ativos A_P, então pode-se dizer que o conjunto de nomes ativos de $\tau.P$ será o conjunto de nomes ativos correspondente ao processo P.

Regras T-Inp$_1$ e T-Inp$_2$: Estas regras são derivadas da regra T-INP definida na Figura 2.5 para o *Base-*π, e que diz respeito à processos com prefixo de entrada ($a(x).P$). No *Active-Base-*π dividimos esta regra em duas regras distintas (T-INP$_1$ e T-INP$_2$) para possibilitar a coleta dos nomes ativos.

A regra T-INP$_1$ coleta os nomes ativos quando o sujeito do prefixo de entrada é um nome livre e por este motivo o nome deverá ser do tipo *link* (premissa $\Gamma \vdash a : \#T$). Assim dizemos que o conjunto de nomes ativos de um processo $a(x).P$ será a união do nome a com conjunto de nomes ativos do processo P (A_P), menos o nome x que é o objeto do prefixo de entrada e portanto um nome ligado. Note que, pela caracterização

dos nomes ativos da Figura 3.1, em um prefixo de entrada o sujeito é ativo e o objeto é inativo, ou seja, pela caracterização dos nomes ativos o objeto de um prefixo de entrada deve ser sempre retirado do conjunto de nomes ativos, e é exatamente isso que é feito ao excluir do conjunto o nome x.

Já na regra T-INP$_2$ coleta-se os nomes ativos quando o sujeito do prefixo de entrada é um nome de tipo *restrito* (premissa $\Gamma \vdash a : @T$), ou seja, um nome restrito ao escopo do agente. Esta regra justifica a necessidade da adição do tipo restrito ($@T$), pois se o canal de comunicação é restrito, então a entrada não pode ser executada e o agente não consegue evoluir à partir daqui. A única forma do agente evoluir seria através de uma comunicação interna, mas para este caso definimos a regra T-PAR$_1$ que veremos mais a frente. Assim se o agente não puder executar uma comunicação interna dizemos que o agente encontra-se em deadlock e todo o restante do agente pode ser descartado na coleta dos nomes ativos, portanto pode-se afirmar que o conjunto de ativos para $a(x).P$ (onde $a : @T$) é vazio.

Regras T-Out$_1$, T-Out$_2$ e T-Out$_3$: Assim como as regras anteriores estas 3 regras são derivadas da regra T-OUT definida na Figura 2.5 para o *Base-*π. Estas regras dizem respeito à processos com prefixo de saída ($\overline{a}b.P$), e foram divididas em três regras pois, novamente a coleta dos nomes ativos dependerá do tipo associado ao canal de comunicação (a).

A regra T-OUT$_1$ coleta os nomes ativos quando tanto o canal de comunicação quanto o nome sendo enviado pelo prefixo de saída são nomes livres. Desta forma o canal de comunicação do prefixo deve estar associado a um tipo *link* (premissa $\Gamma \vdash a : \#T$), e o nome sendo enviado deve ser de um tipo T (premissa $\Gamma \vdash b : T$). Porém, note que T é uma metavariável sobre qualquer tipo, até mesmo restrito, por este motivo adicionamos a premissa *T não restrito*, para evitar qualquer confusão com a regra T-OUT$_2$. Note também que, da mesma forma que no *Base-*π, deve haver uma correspondência entre os tipos dos dois nomes, ou seja, o nome a deve ser de um tipo capaz de carregar um valor do tipo apontado para o nome b.

Seguindo a caracterização de nomes ativos da Figura 3.1, em prefixos de saída onde todos os nomes são livres temos todos os nomes como ativos também. Portanto na regra T-OUT$_1$ a coleta dos nomes ativos para um agente $\overline{a}b.P$, obtém um conjunto de nomes ativos que é a união de a, b e o conjunto de ativos correspondente ao processo P (A_P).

Já na regra T-OUT$_2$ são coletados os nomes ativos quando o canal de comunicação é de um tipo *link* ($\#T$) e o nome enviado é restrito (premissa $\Gamma \vdash b : @T$). Aqui o nome enviado deve ser retirado do conjunto de ativos do agente, conforme a caracterização de nomes ativos da Figura 3.1 (ação $\overline{a}\nu b$). Além disso, ao enviar um nome restrito para fora do agente, deve-se considerar a extrusão de escopo do nome, pois ao enviar o nome para fora do escopo do agente outro agente externo pode vir a se comunicar através deste nome enviado. Assim para o restante da coleta o tipo de b deve ser modificado no ambiente de tipos alterando-o de restrito ($@T$) para um tipo *link* ($\#T$). Essa alteração é representada por $\Gamma \Downarrow b : \#T$ na premissa $\Gamma \Downarrow b : \#T \vdash P : \diamond, A_P$.

Por exemplo, considere o agente $(\nu b : @T)\overline{a}b.b(x).\overline{x}c.0$, neste caso se não considerarmos a extrusão de escopo de b, ao avaliar a ação $b(x)$ chegariamos a conclusão que o agente está em *deadlock* pois b é um nome de tipo *restrito* ($b : @T$). Porém, como o nome foi enviado para outro agente na ação $\overline{a}b$, este agente pode vir a se comunicar através do canal b, o que torna possível que a ação $b(x)$ aconteça. Assim ao avaliar o ação $\overline{a}b$ trocamos o tipo do nome b no ambiente de tipos para um tipo link, desta forma ao avaliar $b(x)$ o nome agora será do tipo *link* e o agente não se encontra mais em $deadlock$.

Por fim temos a regra T-OUT$_3$ que coleta os nomes ativos quando o canal de comunicação é restrito. De forma similar com o que acontece na regra T-INP$_2$ esta ação nunca acontecerá e o conjunto de ativos será vazio.

Regra T-Res: Um processo com construtor restrição é bem tipado caso seu corpo (P) observa as restrições impostas pelo ambiente de tipos Γ e pelos novos nomes declarados de tipo *restrito* ($x : @T$). Se esta condição for respeitada então o agente com restrição será bem tipado e o conjunto de nomes ativos será o conjunto de ativos coletados para o processo P. Não há necessidade aqui de retirar os nomes restritos do conjunto de ativos, pois isto é feito nas demais regras T-INP, T-OUT e T-PAR, verificando apenas se o tipo do nome é restrito ($@T$).

Regra T-Sum: Como a soma de processos é não determinística, o conjunto de nomes ativos será a união do conjunto de nomes ativos de P e Q, e assim como no *Base-*π a regra T-SUM diz que $P + Q$ é bem tipado se ambos P e Q são bem tipados em Γ ($\Gamma \vdash P : \diamond$ e $\Gamma \vdash Q : \diamond$).

A seguir detalhamos as regras da Figura 4.3 que dizem respeito a composição paralela. Note que estas regras foram definidas de tal forma que possibilitam a coleta de nomes ativos, considerando a comunicação interna entre agentes.

Regras T-Par$_1$ à T-Par$_9$: No *Base-*π a regra de composição de processos é similar a regra de soma de processos, ou seja, também é necessário verificar se os dois processos P e Q são bem tipados sobre Γ. Esta abordagem poderia ser utilizada para a coleta de nomes ativos, desde que não fosse considerada a comunicação interna entre os agentes, ou seja, desconsiderando que os agentes envolvidos na composição podem reagir entre si.

Porém como estamos interessados em definir regras para coletar nomes ativos de processos que também se comuniquem entre si, dividimos a regra T-PAR do *Base-*π em outras 9 regras, trabalhando com processos na forma prefixa para verificar se pode ou não existir uma comunicação entre os processos da composição.

As regras T-PAR$_1$ até T-PAR$_8$ foram definidas para considerar a composição de apenas dois agentes. Estes agentes por sua vez não devem ser uma composição, o que caracterizaria a composição paralela de mais de dois agentes. Para este último caso definimos a regra T-PAR$_9$ que possibilita a coleta de nomes ativos em uma composição paralela de mais de dois agentes.

A regra T-PAR$_1$ leva em consideração que os agentes em paralelo podem reagir entre si através de um canal de comunicação restrito a. Para este caso a coleta dos nomes ativos deve levar em consideração a comunicação dos processos, por este motivo o conjunto de nomes ativos coletado será o o conjunto de ativos de $P\{b/x\}|Q$ menos os nomes ligados a e x envolvidos na comunicação e que são inativos. O nome b será o substituto de x em P por isso não é necessário levá-lo em conta nesse momento, pois ele será avaliado na tipagem $P\{b/x\}$. A verificação da validade do processo é feita por três premissas onde é verificada a tipagem dos nomes envolvidos e se a composição de $P\{b/x\}|Q$ é bem tipada. Note que se a composição $P\{b/x\}|Q$ é bem tipada, então isso implica que P e Q também são bem tipados. Sendo assim se as premissas forem verdadeiras pode-se dizer que o agente $a(x).P|\overline{a}b.Q$ é bem tipado sobre Γ.

A regra T-PAR$_2$ coleta os nomes ativos quando os processos da composição não podem reagir entre si, mas pode ser executada uma ação de entrada em um dos agentes da composição, ou seja, em um dos agentes existe um prefixo de entrada cuja o canal de comunicação não é restrito. Esta regra corresponde a aplicação da regra T-INP$_1$ em uma composição de processos. Desta maneira o conjunto de nomes ativos será a união do nome a com o conjunto de ativos da composição $P|Q$, menos o nome x que é inativo

Regras para Processos - Continuação

$$\frac{\Gamma \vdash a : @T \qquad \Gamma \vdash b : T \qquad \Gamma, x : T \vdash P\{b/x\}|Q : \diamond, A_{P\{b/x\}|Q}}{\Gamma \vdash a(x).P|\overline{a}b.Q : \diamond, A_{P\{b/x\}|Q} - \{a, x\}} \quad \text{(T-PAR}_1\text{)}$$

$$\frac{\Gamma \vdash a : \#T \qquad \Gamma, x : T \vdash P|Q : \diamond, A_{P|Q}}{\Gamma \vdash a(x).P|Q : \diamond, (\{a\} \cup A_{P|Q}) - \{x\}} \quad \text{(T-PAR}_2\text{)}$$

$$\frac{\Gamma \vdash a : \#T \qquad \Gamma \vdash b : T \qquad \Gamma \vdash P|Q : \diamond, A_{P|Q} \qquad \textit{T não restrito}}{\Gamma \vdash \overline{a}b.P|Q : \diamond, \{a, b\} \cup A_{P|Q}} \quad \text{(T-PAR}_3\text{)}$$

$$\frac{\Gamma \vdash a : \#T \qquad \Gamma \vdash b : @T \qquad \Gamma \Downarrow b : \#T \vdash P|Q : \diamond, A_{P|Q}}{\Gamma \vdash \overline{a}b.P|Q : \diamond, (\{a\} \cup A_{P|Q}) - \{b\}} \quad \text{(T-PAR}_4\text{)}$$

$$\frac{\Gamma \vdash a : @T \qquad \Gamma, x : T \vdash P : \diamond, A_P \qquad \Gamma \vdash c : @T' \qquad \Gamma \vdash b : T' \qquad \Gamma \vdash Q : \diamond, A_Q}{\Gamma \vdash a(x).P|\overline{c}b.Q : \diamond, \{\}} \quad \text{(T-PAR}_5\text{)}$$

$$\frac{\Gamma \vdash a : @T \qquad \Gamma, x : T \vdash P : \diamond, A_P \qquad \Gamma \vdash c : @T' \qquad \Gamma, y : T' \vdash Q : \diamond, A_Q}{\Gamma \vdash a(x).P|c(y).Q : \diamond, \{\}} \quad \text{(T-PAR}_6\text{)}$$

$$\frac{\Gamma \vdash a : @T \qquad \Gamma \vdash b : T \qquad \Gamma \vdash P : \diamond, A_P \qquad \Gamma \vdash c : @T' \qquad \Gamma \vdash d : T' \qquad \Gamma \vdash Q : \diamond, A_Q}{\Gamma \vdash \overline{a}b.P|\overline{c}d.Q : \diamond, \{\}} \quad \text{(T-PAR}_7\text{)}$$

$$\frac{\Gamma \vdash P : \diamond, A_P}{\Gamma \vdash 0|P : \diamond, A_P} \quad \text{(T-PAR}_8\text{)}$$

$$\frac{\Gamma \vdash P_1|P_2 : \diamond, A_{P_1|P_2} \qquad ... \qquad \Gamma \vdash P_2|P_n : \diamond, A_{P_2|P_n} \qquad ... \qquad \Gamma \vdash P_{n-1}|P_n : \diamond, A_{P_{n-1}|P_n}}{\Gamma \vdash P_1|P_2|...|P_n : \diamond,\ A_{P_1|P_2} \cup ... \cup A_{P_2|P_n} \cup ... \cup A_{P_{n-1}|P_n}} \quad \text{(T-PAR}_9\text{)}$$

$$\frac{\Gamma \vdash P : \diamond, A_P \qquad \Gamma \vdash P|P : \diamond, A_{P|P}}{\Gamma \vdash !P : \diamond, A_P \cup A_{P|P}} \quad \text{(T-REP)}$$

Figura 4.3: Regras de Tipos para o Active-Base-π (continuação).

conforme a Figura 3.1. A diferença para a regra T-INP$_1$ é que aqui o restante dos nomes ativos são coletados sobre a composição $P|Q$, isto porque é necessário verificar se P e Q não podem reagir entre si em ações futuras. A verificação da tipagem também é feita de forma semelhante a regra T-INP$_1$ com a diferença que deve-se verificar a tipagem da composição $P|Q$ e não de apenas um processo. Esta verificação implica que P e Q também são bem tipados.

É importante enfatizar aqui que a regra T-PAR$_2$, assim como as regras T-PAR$_3$ e T-PAR$_4$, são escolhidas quando o agente pode evoluir executando uma ação de entrada ou saída em um dos componentes da composição paralela. Note aqui que agentes na forma $a(x).P|\overline{a}b.Q$, onde o nome a *não é restrito*, poderá executar a ação $a(x)$ (regra T-PAR$_2$), ou executar a ação $\overline{a}b$ (regra T-PAR$_3$), ou ainda executar uma comunicação interna τ através do nome a. Porém aqui não definimos nenhuma regra para comunicação interna, pois a regra T-PAR$_1$ diz que o canal de comunicação a deve ser do tipo restrito. Assim coleta-se os nomes ativos deste agente considerando apenas a execução da ação $a(x)$ e $\overline{a}b$ em separado. Note que a escolha da execução destas regras não interfere no conjunto final de nomes ativos, pois o conjunto obtido será maior ou igual ao conjunto de nomes ativos que seria obtido se fosse considerada a possibilidade de uma comunicação interna, já que neste caso o canal de comunicação é livre.

As regras T-PAR$_3$ e T-PAR$_4$ coletam os nomes para a composição de processos que não podem se comunicar entre si, mas podem executar uma ação de saída em um dos agentes. Estas regras tem a mesma função que as regras T-OUT$_1$ e T-OUT$_2$, porém são utilizadas para composição de processos. Desta forma utilizamos a mesma idéia de correspondência que usamos na regra anterior para uma ação de entrada, apenas observando que T-PAR$_3$ é usada para coletar os nomes ativos de saídas com o nome b livre e a regra T-PAR$_4$ é usada para coletar os nomes ativos de saídas com b restrito, e portanto observando a extrusão de escopo do nome b.

Nas regras T-PAR$_5$, T-PAR$_6$ e T-PAR$_7$ são coletados os nomes ativos quando ambos os agentes da composição possuem prefixos com o canal de comunicação restrito, mas não podem reagir entre si, ou seja, não é possível ocorrer uma comunicação interna. Neste caso pode-se dizer que o agente encontra-se em *deadlock*, pois nenhuma ação pode ser executada, e assim o conjunto de ativos será vazio como acontece nas regras T-INP$_2$ e T-OUT$_3$. Foram definidas três regras para este caso devido a combinação dos prefixos entre os agentes. Em T-PAR$_5$ considera-se que um dos agentes possui um prefixo de entrada e no outro um prefixo de saída, já em T-PAR$_6$ ambos os agentes possuem prefixos de entrada e por fim em T-PAR$_7$ ambos os agentes possuem prefixos de saída.

A regra T-PAR$_8$ coleta os nomes quando um dos lados da composição é o mesmo que 0. E neste caso o conjunto de ativos será apenas a avaliação do lado que sobrou na composição como um agente simples.

Para completar a regra T-PAR$_9$ coleta os nomes ativos na composição paralela de mais de dois processos. Neste caso para coletar os nomes ativos é necessário dividir a composição em várias outras sub-composições contendo apenas dois processos. Desta forma é possível coletar os nomes ativos de cada uma destas sub-composições usando as regras de T-PAR$_1$ até T-PAR$_8$ que são definidas sobre a composição paralela de apenas dois processos.

A divisão do processo principal deve ser realizada considerando todas as possíveis combinações de processos que podem ocorrer, e o conjunto de nomes ativos final será a união de todos os conjuntos de nomes ativos encontrados nas sub-composições. Por exemplo para o agente $P_1|P_2|P_3$, o conjunto de nomes ativos será $A_{P_1|P_2} \cup A_{P_1|P_3} \cup A_{P_2|P_3}$.

T-Rep: Na regra de tipo para um processo de replicação verificamos se o processo é válido e se a composição paralela de duas cópias dele também é válida. O conjunto de nomes ativos será a união do conjunto de nomes ativos encontrado para uma única cópia, com o conjunto de nomes ativos encontrado para a composição paralela de duas cópias do processo. Desta forma estamos considerando tanto a coleta dos nomes ativos para agentes onde não há comunicação interna, quanto também para agentes onde pode haver comunicação interna entre as cópias do processo.

4.1.1 Exemplos

Nesta seção mostramos alguns exemplos de aplicação das regras propostas acima para a coleta dos nomes ativos. Os exemplos a seguir seguem a mesma idéia dos exemplos utilizados na apresentação do *Base-*π na Seção 2.2.2.

Exemplo 12 *Coleta dos nomes ativos para o agente* $P = (\nu y : @T)\ a(x).\overline{x}b.\overline{y}c.0$ *com* $\Gamma = \{a : \#\#S, b : S, c : T\}$*, onde* S *não é tipo restrito.*

Derivação 1:

$$\dfrac{\dfrac{\dfrac{\overline{\Gamma \vdash a : \#\#S}^{\text{TV-Name}} \quad \dfrac{\overline{\Gamma'' \vdash x : \#S}^{\text{TV-Name}} \quad \overline{\Gamma'' \vdash b : S}^{\text{TV-Name}} \quad \textit{Derivação 2}}{\Gamma', x : \#S \vdash \overline{x}b.\overline{y}c.0 : \diamond, \{x, b\} \cup A_{\overline{y}c.0}}\ \text{T-Out}_1}{\Gamma, y : @T \vdash a(x).\overline{x}b.\overline{y}c.0 : \diamond, (\{a\} \cup A_{\overline{x}b.\overline{y}c.0}) - \{x\}}\ \text{T-Inp}_1}{\Gamma \vdash (\nu y : @T)\ a(x).\overline{x}b.\overline{y}c.0 : \diamond, A_{a(x).\overline{x}b.\overline{y}c.0}}\ \text{T-Res}$$

Derivação 2:

$$\dfrac{\overline{\Gamma'' \vdash y : @T}^{\text{TV-Name}} \quad \overline{\Gamma'' \vdash c : T}^{\text{TV-Name}} \quad \overline{\Gamma'' \vdash 0 : \diamond, \{\}}^{\text{T-Nil}}}{\Gamma'' \vdash \overline{y}c.0 : \diamond, \{\}}\ \text{T-Out}_3$$

$$\begin{array}{rcl} Onde & : & \Gamma' = \Gamma, y : @T \\ & & \Gamma'' = \Gamma', x : \#S \end{array}$$

Conclusão:

$$\begin{array}{rcl} A_{a(x).\overline{x}b.\overline{y}c.0} & = & (\{a\} \cup A_{\overline{x}b.\overline{y}c.0}) - \{x\} \\ & = & (\{a\} \cup (\{x, b\} \cup A_{\overline{y}c.0})) - \{x\} \\ & = & (\{a\} \cup (\{x, b\} \cup \{\})) - \{x\} = \{a, b\} \end{array}$$

■

A leitura do exemplo, da mesma forma que na Seção 2.2.2, é feita de baixo para cima para melhor entendimento. Veja que, pela conclusão da regra T-RES, o conjunto de nomes ativos para um agente $(\nu x : @T)P$ será o conjunto de nomes ativos encontrado para P sem considerar a restrição $(\nu x : @T)$. Portanto aqui no exemplo o conjunto de nomes ativos para $(\nu y : @T)a(x).\overline{x}b.\overline{y}c.0$ será o conjunto de ativos encontrados para $a(x).\overline{x}b.\overline{y}c.0$. A partir daqui basta ir montando o cojunto de baixo para cima, pois o resultado de $A_{a(x).\overline{x}b.\overline{y}c.0}$ é $(\{a\} \cup A_{\overline{x}b.\overline{y}c.0}) - \{x\}$ que se encontra na premissa da regra T-RES. Agora temos que encontrar o resultado de $A_{\overline{x}b.\overline{y}c.0}$ que se encontra na premissa

da T-INP$_1$ e assim sucessivamente até chegarmos ao fim da derivação como mostra a conclusão do exemplo.

O exemplo a seguir ilustra como a extrusão de escopo é tratada pelo sistema de tipos (veja T-OUT$_2$).

Exemplo 13 *Coleta dos nomes ativos para o agente* $P = (\nu b : @T)\ a(x).\overline{x}b.\overline{b}y.0$ *quando* $\Gamma = \{a : \#\#T, y : T\}$

Derivação 1:

$$\dfrac{\dfrac{\dfrac{}{\Gamma' \vdash a : \#\#T}\ \textit{TV-Name} \quad \dfrac{\dfrac{}{\Gamma'' \vdash x : \#T}\ \textit{TV-Name} \quad \dfrac{}{\Gamma'' \vdash b : @T}\ \textit{TV-Name} \quad \textit{Derivação 2}}{\Gamma', x : \#T \vdash \overline{x}b.\overline{b}y.0 : \diamond, (\{x\} \cup A_{\overline{b}y.0}) - \{b\}}\ \textit{T-Out}_2}{\Gamma, b : @T \vdash a(x).\overline{x}b.\overline{b}y.0 : \diamond, (\{a\} \cup A_{\overline{x}b.\overline{b}y.0}) - \{x\}}\ \textit{T-Inp}_1}{\Gamma \vdash (\nu b : @T)\ a(x).\overline{x}b.\overline{b}y.0 : \diamond, A_{a(x).\overline{x}b.\overline{b}y.0}}\ \textit{T-Res}$$

Derivação 2:

$$\dfrac{\dfrac{}{\Gamma''' \vdash b : \#T}\ \textit{TV-Name} \quad \dfrac{}{\Gamma''' \vdash y : T}\ \textit{TV-Name} \quad \dfrac{}{\Gamma''' \vdash 0 : \diamond, \{\}}\ \textit{T-Nil}}{\Gamma'' \Downarrow b : \#T \vdash \overline{b}y.0 : \diamond, \{b, y\} \cup A_0}\ \textit{T-Out}_1$$

$$\begin{array}{rcl} \textit{Onde} & : & \Gamma' = \Gamma, b : @T \\ & & \Gamma'' = \Gamma', x : \#T \\ & & \Gamma''' = \Gamma'' \Downarrow b : \#T \end{array}$$

Conclusão:

$$\begin{array}{rcl} A_{a(x).\overline{x}b.\overline{b}y.0} & = & (\{a\} \cup A_{\overline{x}b.\overline{b}y.0}) - \{x\} \\ & = & (\{a\} \cup (\{x\} \cup A_{\overline{b}y.0}) - \{b\}) - \{x\} \\ & = & (\{a\} \cup (\{x\} \cup (\{b, y\} \cup A_0)) - \{b\}) - \{x\} \\ & = & (\{a\} \cup (\{x\} \cup (\{b, y\} \cup \{\})) - \{b\}) - \{x\} = \{a, y\} \end{array}$$

■

Com este exemplo procuramos mostrar a extrusão de escopo do nome b na ação $\overline{x}b$. Note que se o tipo de b não fosse modificado em $\Gamma'' \Downarrow b : \#T \vdash \overline{b}y.0 : \diamond,$ a regra T-OUT$_1$ iria falhar, pois o nome b seria do tipo $@T$ e não do tipo $\#T$. Isso significa que a regra a ser aplicada seria T-OUT$_3$ e nome y seria excluído do conjunto de nomes ativos pois este nome não poderia ser alcançado no cáculo dos nomes ativos. Porém aqui o nome y deve ser considerado ativo, pois o nome b que é restrito é enviado para fora do escopo do agente P. Assim um outro agente que recebe este nome b pode vir a se comunicar com o agente P através da ação $\overline{b}y$, possibilitando que o nome y seja alcançado.

O exemplo abaixo mostra a coleta de nomes ativos em um agente na forma de soma de processos.

Exemplo 14 *Coleta dos nomes ativos do agente* $P = (\nu y : @T)(a(x).\overline{x}b.0 + \overline{c}d.\overline{y}e.\overline{e}g.0)$ *com* $\Gamma = \{a : \#\#S, b : S, c : \#S', d : S', e : \#T, g : T\}$*, onde S, S' e T não são tipos restritos.*

Derivação 1:

$$\dfrac{\dfrac{\textit{Derivação 2} \qquad \textit{Derivação 3}}{\Gamma, y : @T \vdash a(x).\overline{x}b.0 \ + \ \overline{c}d.\overline{y}e.\overline{e}g.0 : \diamond, A_{a(x).\overline{x}b.0} \cup A_{\overline{c}d.\overline{y}e.\overline{e}g.0}}\ \textit{T-Sum}}{\Gamma \vdash (\nu y : @T)(a(x).\overline{x}b.0 \ + \ \overline{c}d.\overline{y}e.\overline{e}g.0) : \diamond, A_{(a(x).\overline{x}b.0 \ + \ \overline{c}d.\overline{y}e.\overline{e}g.0)}}\ \textit{T-Res}$$

Derivação 2:

$$\dfrac{\dfrac{}{\Gamma' \vdash a : \#\#S}\ \textit{TV-Name} \qquad \dfrac{\dfrac{}{\Gamma'' \vdash x : \#S}\ \textit{TV-Name} \quad \dfrac{}{\Gamma'' \vdash b : S}\ \textit{TV-Base} \quad \dfrac{}{\Gamma'' \vdash 0 : \diamond, \{\}}\ \textit{T-Nil}}{\Gamma', x : \#S \vdash \overline{x}b.0 : \diamond, \{x, b\} \cup A_0}\ \textit{T-Out}_1}{\Gamma' \vdash a(x).\overline{x}b.0 : \diamond, (\{a\} \cup A_{\overline{x}b.0}) - \{x\}}\ \textit{T-Inp}_1$$

Derivação 3:

$$\dfrac{\dfrac{}{\Gamma' \vdash c : \#S'}\ \textit{TV-Name} \quad \dfrac{}{\Gamma' \vdash d : S'}\ \textit{TV-Base} \qquad \dfrac{\dfrac{}{\Gamma' \vdash y : @T}\ \textit{TV-Name} \quad \dfrac{}{\Gamma' \vdash e : \#T}\ \textit{TV-Name} \qquad \textit{Derivação 4}}{\Gamma' \vdash \overline{y}e.\overline{e}g.0 : \diamond, \{\}}\ \textit{T-Out}_1}{\Gamma' \vdash \overline{c}d.\overline{y}e.\overline{e}g.0 : \diamond, \{c, d\} \cup A_{\overline{y}e.\overline{e}g.0}}\ \textit{T-Out}_3$$

Derivação 4:

$$\dfrac{\dfrac{}{\Gamma' \vdash e : \#T}\ \textit{TV-Name} \quad \dfrac{}{\Gamma' \vdash g : T}\ \textit{TV-Base} \quad \dfrac{}{\Gamma' \vdash 0 : \diamond, \{\}}\ \textit{T-Nil}}{\Gamma' \vdash \overline{e}g.0 : \diamond, \{e, g\} \cup A_0}\ \textit{T-Out}_1$$

$$\begin{array}{rcl} \textit{Onde} & : & \Gamma' = \Gamma, y : @T \\ & & \Gamma'' = \Gamma', x : \#T \end{array}$$

Conclusão:

$$\begin{array}{rcl} A_{(a(x).\overline{x}b.0 \ + \ \overline{c}d.\overline{y}e.\overline{e}g.0)} & = & A_{a(x).\overline{x}b.0} \cup A_{\overline{c}d.\overline{y}e.\overline{e}g.0} \\ & = & ((\{a\} \cup A_{\overline{x}b.0}) - \{x\}) \cup (\{c, d\} \cup A_{\overline{y}e.\overline{e}g.0}) \\ & = & ((\{a\} \cup (\{x, b\} \cup A_0)) - \{x\}) \cup (\{c, d\} \cup \{\}) \\ & = & ((\{a\} \cup (\{x, b\} \cup \{\})) - \{x\}) \cup (\{c, d\} \cup \{\}) \\ & = & (\{a, x, b\} - \{x\}) \cup \{c, d\} \\ & = & \{a, b, c, d\} \end{array}$$

■

Note neste exemplo que, no segundo componente da soma ($\overline{c}d.\overline{y}e.\overline{e}g.0$), o nome y é restrito e não é realizada a extrusão do nome y. Portanto ao analisar o prefixo $\overline{y}e$ concluimos que ele esta restrito e portanto os demais nomes que seguem após este prefixo podem ser eliminados do conjunto de ativos, pois as ações seguintes não podem acontecer, considerando apenas o escopo do agente P.

Os próximos dois exemplos ilustram o uso de regras de tipos para processos que usam composição paralela.

Exemplo 15
Coleta dos nomes ativos para $P = (\nu y : @@T)(\nu z : @T)(\overline{a}b.y(x).\overline{x}c.0 \mid \overline{y}z.d(u).u(w).0)$ *com* $\Gamma = \{a : \#S, b : S, c : T, d : \#\#S'\}$*, onde T não é tipo restrito.*

Derivação 1:

$$\dfrac{\dfrac{\dfrac{\overline{\Gamma'' \vdash a : \#S}^{\text{TV-Name}} \quad \overline{\Gamma'' \vdash b : S}^{\text{TV-Name}} \quad \textit{Derivação 2}}{\Gamma', z : @T \vdash \overline{a}b.y(x).\overline{x}c.0 \mid \overline{y}z.d(u).u(w).0 : \diamond, \{a, b\} \cup A_{y(x).\overline{x}c.0 \mid \overline{y}z.d(u).u(w).0}}\ \text{T-Par}_3}{\Gamma, y : @@T \vdash (\nu z : @T)(\overline{a}b.y(x).\overline{x}c.0 \mid \overline{y}z.d(u).u(w).0) : \diamond, A_{\overline{a}b.y(x).\overline{x}c.0 \mid \overline{y}z.d(u).u(w).0}}\ \text{T-Res}}{\Gamma \vdash (\nu y : @@T)(\nu z : @T)(\overline{a}b.y(x).\overline{x}c.0 \mid \overline{y}z.d(u).u(w).0) : \diamond, A_{(\nu z:@T)(\overline{a}b.y(x).\overline{x}c.0 \mid \overline{y}z.d(u).u(w).0)}}\ \text{T-Res}$$

$$\begin{aligned} Onde \quad : \quad & \Gamma' = \Gamma, y : @@T \\ & \Gamma'' = \Gamma', z : @T \end{aligned}$$

Derivação 2:

$$\dfrac{\overline{\Gamma'' \vdash y : @@T}^{\text{TV-Name}} \quad \overline{\Gamma'' \vdash z : @T}^{\text{TV-Name}} \quad \textit{Derivação 3}}{\Gamma'' \vdash y(x).\overline{x}c.0 \mid \overline{y}z.d(u).u(w).0 : \diamond, A_{\overline{z}c.0 \mid d(u).u(w).0} - \{y, x\}}\ \text{T-Par}_1$$

Derivação 3:

$$\dfrac{\dfrac{\overline{\Gamma''' \vdash d : \#\#S'}^{\text{TV-Name}} \quad \dfrac{\overline{\Gamma'''' \vdash u : \#S'}^{\text{TV-Name}} \quad \textit{Derivação 4}}{\Gamma''', u : \#S' \vdash \overline{z}c.0 \mid u(w).0 : \diamond, (\{u\} \cup A_{\overline{z}c.0 \mid 0}) - \{w\}}\ \text{T-Par}_2}{\Gamma'', x : @T \vdash \overline{z}c.0 \mid d(u).u(w).0 : \diamond, (\{d\} \cup A_{\overline{z}c.0 \mid u(w).0}) - \{u\}}\ \text{T-Par}_2$$

$$\begin{aligned} Onde \quad : \quad & \Gamma''' = \Gamma'', x : @T \\ & \Gamma'''' = \Gamma''', u : \#S' \end{aligned}$$

Derivação 4:

$$\dfrac{\dfrac{\overline{\Gamma''''' \vdash z : @T}^{\text{TV-Name}} \quad \overline{\Gamma''''' \vdash c : T}^{\text{TV-Name}} \quad \overline{\Gamma''''' \vdash 0 : \diamond, \{\}}^{\text{T-Nil}}}{\Gamma''''' \vdash \overline{z}c.0 \ : \diamond, \{\}}\ \text{T-Inp}_2}{\Gamma'''', w : S' \vdash \overline{z}c.0 \mid 0 : \diamond, A_{\overline{z}c.0}}\ \text{T-Par}_8$$

$$Onde \quad : \quad \Gamma''''' = \Gamma'''', w : S'$$

Conclusão:

$$\begin{aligned} A_{(\nu y:@@T)(\nu z:@T)(\overline{a}b.y(x).\overline{x}c.0 \mid \overline{y}z.d(u).u(w).0)} &= A_{(\nu z:@T)(\overline{a}b.y(x).\overline{x}c.0 \mid \overline{y}z.d(u).u(w).0)} \\ &= A_{\overline{a}b.y(x).\overline{x}c.0 \mid \overline{y}z.d(u).u(w).0} \\ &= \{a, b\} \cup A_{y(x).\overline{x}c.0 \mid \overline{y}z.d(u).u(w).0} \\ &= \{a, b\} \cup (A_{\overline{z}c.0 \mid d(u).u(w).0} - \{y, x\}) \\ &= \{a, b\} \cup ((\{d\} \cup A_{\overline{z}c.0 \mid u(w).0}) - \{u, y, x\}) \\ &= \{a, b\} \cup ((\{d\} \cup \{u\}) - \{u, y, x\}) \\ &= \{a, b, d\} \end{aligned}$$

■

Este exemplo mostra a aplicação das regras de composição paralela apresentadas na Figura 4.3. Note que na derivação 1 aplica-se a regra T-PAR$_3$, pois não é possível acontecer uma comunicação interna em $\overline{a}b.y(x).\overline{x}c.0 \mid \overline{y}z.d(u).u(w).0$ e a ação $\overline{y}z$ não pode acontecer já que y é um nome restrito. Assim somente pode acontecer a ação $\overline{a}b$, que é uma ação de saída e portanto a regra T-PAR$_3$ é escolhida.

Também observe que ao aplicar-se a regra T-PAR$_1$ na derivação 2, não verificamos se o nome z é ativo ou não. Esta verificação é feita no momento que ele é novamente utilizado. Neste caso na derivação 4 ao aplicarmos a regra T-INP$_2$ verificamos que o nome b é restrito e portanto não deve ser incluido no conjunto de ativos.

Exemplo 16
Coleta dos nomes ativos para $P = (\nu y : @@\#T)(\nu z : @\#T)(\overline{a}b.y(x).\overline{x}c.0 \mid \overline{y}z.z(u).\overline{u}w.0)$ *onde* $\Gamma = \{a : \#S, b : S, c : \#T, w : T\}$*, onde* T *não é um tipo restrito.*

Derivação 1:

$$\dfrac{\dfrac{\dfrac{\overline{\Gamma'' \vdash a : \#S}^{\text{TV-Name}} \quad \overline{\Gamma'' \vdash b : S}^{\text{TV-Base}} \quad \textit{Derivação 2}}{\Gamma', z : @\#T \vdash \overline{a}b.y(x).\overline{x}c.0 \mid \overline{y}z.z(u).\overline{u}w.0 : \diamond, \{a,b\} \cup A_{y(x).\overline{x}c.0 \mid \overline{y}z.z(u).\overline{u}w.0}}\ \text{T-Par}_3}{\Gamma, y : @@\#T \vdash (\nu z : @\#T)(\overline{a}b.y(x).\overline{x}c.0 \mid \overline{y}z.z(u).\overline{u}w.0) : \diamond, A_{(\overline{a}b.y(x).\overline{x}c.0 \mid \overline{y}z.z(u).\overline{u}w.0)}}\ \text{T-Res}}{\Gamma \vdash (\nu y : @@\#T)(\nu z : @\#T)(\overline{a}b.y(x).\overline{x}c.0 | \overline{y}z.z(u).\overline{u}w.0) : \diamond, A_{(\nu z:@\#T)(\overline{a}b.y(x).\overline{x}c.0 | \overline{y}z.z(u).\overline{u}w.0)}}\ \text{T-Res}$$

$$Onde \quad : \quad \Gamma' = \Gamma, y : @@\#T$$
$$\Gamma'' = \Gamma', z : @\#T$$

Derivação 2:

$$\dfrac{\overline{\Gamma'' \vdash y : @@\#T}^{\text{TV-Name}} \quad \overline{\Gamma'' \vdash z : @\#T}^{\text{TV-Name}} \quad \textit{Derivação 3}}{\Gamma'' \vdash y(x).\overline{x}c.0 \mid \overline{y}z.z(u).\overline{u}w.0 : \diamond, A_{\overline{z}c.0 \mid z(u).\overline{u}w.0} - \{y, x\}}\ \text{T-Par}_1$$

Derivação 3:

$$\dfrac{\overline{\Gamma''' \vdash z : @\#T}^{\text{TV-Name}} \quad \overline{\Gamma''' \vdash c : \#T}^{\text{TV-Name}} \quad \textit{Derivação 4}}{\Gamma'', x : @\#T \vdash \overline{z}c.0 \mid z(u).\overline{u}w.0 : \diamond, A_{0 \mid \overline{c}w.0} - \{z, u\}}\ \text{T-Par}_1$$

$$Onde \quad : \quad \Gamma''' = \Gamma'', x : @\#T$$

Derivação 4:

$$\dfrac{\dfrac{\overline{\Gamma'''' \vdash c : \#T}^{\text{TV-Name}} \quad \overline{\Gamma'''' \vdash w : T}^{\text{TV-Name}} \quad \overline{\Gamma'''' \vdash 0 : \diamond, \{\}}^{\text{T-Nil}}}{\Gamma'''' \vdash \overline{c}w.0 : \diamond, \{c, w\} \cup A_0}\ \text{T-Out}_1}{\Gamma''', u : \#T \vdash 0 \mid \overline{c}w.0 : \diamond, A_{\overline{c}w.0}}\ \text{T-Par}_8$$

$$Onde \quad : \quad \Gamma'''' = \Gamma''', u : \#T$$

Conclusão:

$$\begin{aligned}
A_{(\nu y:@@\#T)(\nu z:@\#T)(\overline{a}b.y(x).\overline{x}c.0|\overline{y}z.z(u).\overline{u}w.0)} &= A_{(\nu z:@\#T)(\overline{a}b.y(x).\overline{x}c.0|\overline{y}z.z(u).\overline{u}w.0)} \\
&= A_{(\overline{a}b.y(x).\overline{x}c.0|\overline{y}z.z(u).\overline{u}w.0)} \\
&= \{a,b\} \cup A_{y(x).\overline{x}c.0 \mid \overline{y}z.z(u).\overline{u}w.0} \\
&= \{a,b\} \cup (A_{\overline{z}c.0 \mid z(u).\overline{u}w.0} - \{y,x\}) \\
&= \{a,b\} \cup ((A_{0 \mid \overline{c}w.0} - \{z,u\}) - \{y,x\}) \\
&= \{a,b\} \cup ((A_{\overline{c}w.0} - \{z,u\}) - \{y,x\}) \\
&= \{a,b\} \cup ((((\{c,w\} \cup A_0) - \{z,u\}) - \{y,x\}) \\
&= \{a,b\} \cup ((((\{c,w\} \cup \{\}) - \{z,u\}) - \{y,x\}) \\
&= \{a,b\} \cup (\{c,w\} - \{y,x\}) \\
&= \{a,b,c,w\}
\end{aligned}$$

■

Veja neste exemplo que o nome y é do tipo restrito e pode carregar um nome do mesmo tipo de z. Por sua vez o nome z também é restrito e pode carregar um nome do tipo *link* ($\#T$). Caso o nome z fosse do tipo $@T$ teriamos um erro ao avaliar a ação $\overline{u}w$, pois u é um nome ligado declarado em $z(u)$ e como z recebe um valor do tipo T, não poderíamos usar u como uma referência para um canal de comunicação.

4.2 Propriedades do Active-Base-π

Nesta seção definimos algumas propriedades exclusivas para o *Active-Base-*π. Nas propriedades que seguem $\vdash_{AB}$ é usado em julgamentos de tipos do *Active-Base-*π, e $\vdash_B$ é usado para julgamentos de tipos do *Base-*π.

A propriedade abaixo mostra que o conjunto de nomes ativos coletado pelo *Active-Base-*π é um subconjunto dos nomes livres do agente P.

Proposição 1 *Se* $\Gamma \vdash_{AB} P : \diamond, A_P$ *então* $A_P \subseteq fn(P)$.

A prova segue por indução na altura da árvore de derivação para $\Gamma \vdash P : A_P$. Para esta prova deve-se considerar as formas de agentes-π apresentadas na Figura 2.1, e também a definição de nomes livres dada na Seção 2.1.1. Abaixo apresentamos está prova para três formas de agentes:

- Agente 0 : Para agentes na forma 0 aplica-se a regra T-NIL do *Active-Base-*π para a coleta dos nomes ativos. O conjunto de ativos obtido será {} que é igual ao conjunto de nomes livres obtido na Definição 2 para agentes na forma 0.

- Agente $\tau.P$: Neste caso aplica-se a regra T-TAU para a coleta dos nomes ativos. Por essa regra $A_{\tau.P} = A_P$ e, pela hipótese indutiva $A_P \subseteq fn(P)$. Assim pela Definição 2 obtem-se que $fn(\tau.P) = fn(P)$ e portanto, $A_{\tau.P} \subseteq fn(\tau.P)$.

- Agente $a(x).P$: Aqui queremos provar que $A_{a(x).P} \subseteq fn(a(x).P)$ e, para um agente com prefixo de entrada existem duas regras que podem ser aplicadas (T-INP$_1$ ou T-INP$_2$). Pela conclusão da regra T-INP$_1$ obtem-se que $A_{a(x).P} = (\{a\} \cup A_P) - \{x\}$. Já pela hipótese indutiva e pela segunda premissa de T-INP$_1$, temos que

$A_P \subseteq fn(P)$ e logo $(\{a\} \cup A_P) - \{x\} \subseteq (\{a\} \cup fn(P)) - \{x\}$. Como $fn(a(x).P) = (\{a\} \cup fn(P)) - \{x\}$ (Definição 2) conclui-se que $A_{a(x).P} \subseteq fn(a(x).P)$ como desejado.
Já para a regra T-INP$_2$ obtemos que $A_{a(x).P} = \{\}$ e portanto é um subconjunto de $fn(a(x).P)$, pois {} é subconjunto de qualquer outro conjunto.

A prova para as demais formas de processo, segue de modo semelhante a prova das três formas de processos mostradas acima.

Nas próximas propriedades queremos mostrar que se o agente P é bem tipado sob o *Active-Base-*π, então ele será bem tipado sob o *Base-*π e vice-versa.

Proposição 2 *Se* $\Gamma \vdash_{AB} P : \diamond, A_P$ *então* $\overline{\Gamma} \vdash_B \overline{P} : \diamond$*, onde* $\overline{\Gamma}$ *e* $\overline{P}$ *são o mesmo que* Γ *e* P *respectivamente com as ocorrências de tipo* $@T$ *substituídas por* $\#T$.

Para que a propriedade 2 seja correta, deve-se considerar que todas as ocorrências de nomes restritos no *Active-Base-*π (nomes de tipo $@T$), são substituídas por *links* no *Base-*π, ou seja, no *Base-*π os nomes restritos são do tipo *link* ($\#T$). Esta mesma propriedade pode ser definida de forma inversa como mostramos abaixo.

Proposição 3 *Se* $\Gamma \vdash_B \overline{P} : \diamond$ *então* $\Gamma \vdash_{AB} P : \diamond, A_P$ *para algum conjunto* A_P *de nomes, onde as ocorrências de* $(\nu a : \#T)$ *em* $\overline{P}$ *são substituídas por* $(\nu a : @T)$ *em P.*

Esta propriedade diz que se um agente $\overline{P}$ é bem tipado no *Base-*π então ele será bem tipado no *Active-Base-*π, desde que todas as ocorrências do construtor de restrição em $\overline{P}$ sejam substituídas pelo construtor de restrição do *Active-Base-*π em P.

Note na propriedade 3 que o ambiente de tipos Γ usado nos dois julgamentos é o mesmo. Isto porque os nomes restritos contidos em Γ para o *Base-*π possuem um apontamento na forma $a : \#T$, ou seja, o mesmo tipo de apontamento que nomes não restritos. Assim não podemos distinguir nomes restitos de nomes não restritos para realizar uma tradução para o ambiente usado no *Active-Base-*π. Porém isso não afeta a verificação da validade de uma expressão, pois os nomes restritos no *Active-Base-*π apenas influenciam na coleta dos nomes ativos, e os novos nomes que venham a ser restritos em P serão identificados pelo construtor de restrição que é substituído de $\overline{P}$ para P.

Já na propriedade 2 esta característica não ocorre, pois em Γ utilizado em julgamentos para o *Active-Base-*π encontramos as ocorrências de nomes restritos na forma $a : @T$, e estas devem ser substitídas em $\overline{\Gamma}$ por $a : \#T$ para o *Base-*π, caso contrário teríamos uma apontamento de tipos inválido para o *Base-*π.

Outra propriedade que podemos definir diz respeito a restrição dos nomes inativos a um escopo local dos agentes-π. Desta forma pode-se inibir a influência dos nomes inativos na geração do π-autômato. Esta idéia é baseada na noção semântica de nomes ativos mostrada por Montanari e Pistore, apresentada na Seção 3.2 (7). Pela definição, um nome a é ativo para um agente P, se e somente se $(\nu a)P$ *não* é bisimilar à P. Desta forma se restringirmos um nome inativo em P, obtemos um novo agente P' que é bissimilar ao agente original P. Assim o que fazemos aqui é restringir P com respeito à todos os nomes inativos obtendo um novo agente P' bissimilar a P, onde todos os nomes livres serão ativos.

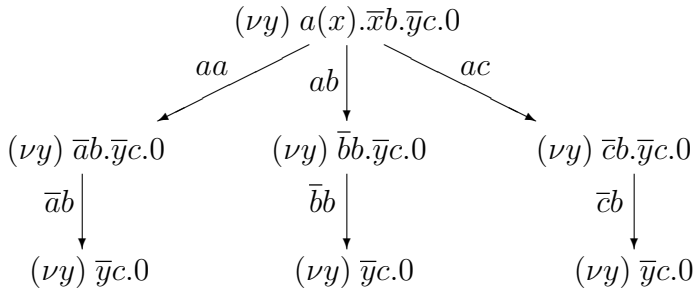
Definimos I_P como o conjunto de nomes inativos de P dado por $I_P = fn(P) - A_P$ (onde $fn(P)$ é o conjunto de nomes livres do agente P e A_P é o conjunto de nomes ativos do agente P).

Proposição 4 *Se* $\Gamma \vdash P : \diamond, A_P$ *então* $P \sim (\nu I)P$, *onde* $I = fn(P) - A_P$.

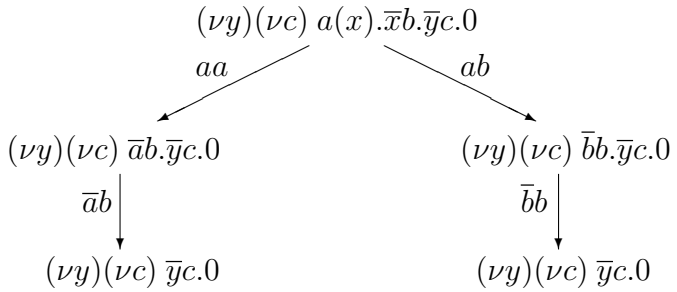
O novo agente com os nomes inativos restritos gera um π-autômato equivalente ao *unwinding-autômato* gerado por Montanari e Pistore (MONTANARI; PISTORE, 1995) e apresentado na Seção 3.3.

Para melhor entendimento desta redução mostramos abaixo dois exemplos de π-*autômatos* gerados à partir do conjunto de nomes ativos encontrados nos Exemplos 12 e 15 da Seção 4.1.1. Primeiro montamos o autômato considerando apenas os nomes livres sem levar em consideração os nomes inativos, e logo em seguida é gerado o autômato restringindo os nomes inativos conforme o seu conjunto de nomes ativos.

Exemplo 17 *Montar os* π*-autômatos para o agente* P *do exemplo 12, utilizando o conjunto de nomes ativos* $A_P = \{a, b\}$ *coletado pelo sistema de tipos.*



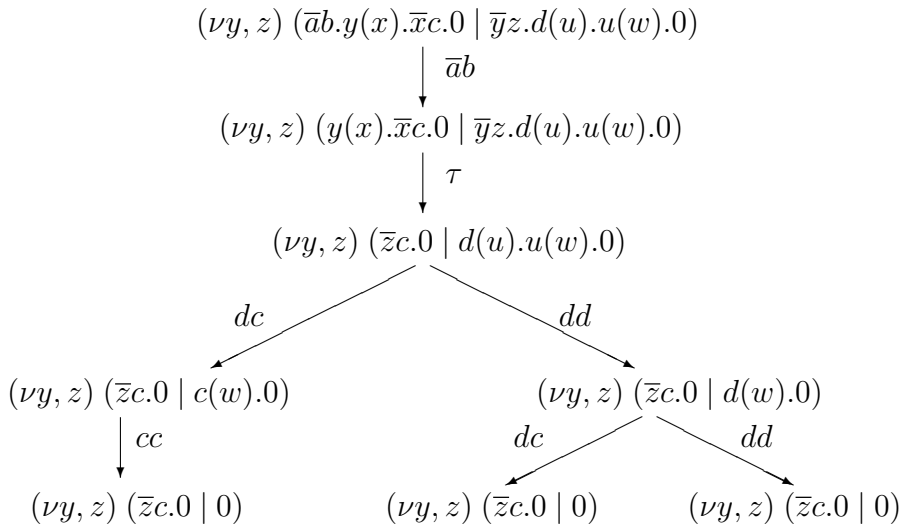
No π*-autômato acima o nome* c *é livre mas não pode ser executado portanto é inativo. A seguir mostramos a montagem do* π*-autômato quando o nome* c *é restrito. Note que tal autômato é equivalente ao unwinding-automata da Figura 3.3, apresentado quando discutimos a abordagem do Montanari e Pistore na Seção 3.3.*



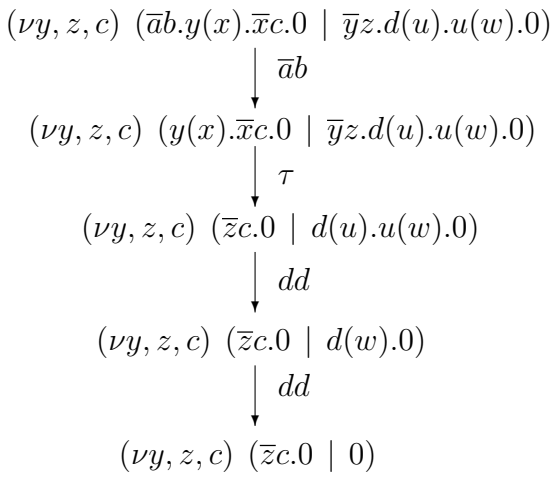
■

Note nestes exemplos que nas transições de entrada além da utilização dos nomes livres, também deve existir uma transição usando um novo nome ainda não utilizado agente, como citamos na Seção 3.1. Porém para nos π-autômatos não colocamos estas transições para reduzir o tamanho do autômato e torna-lo mais simples de ser entendido.

Exemplo 18 *Montar os π-autômatos para o agente P do exemplo 15, utilizando o conjunto de nomes ativos $A_P = \{a, b, d\}$ coletado no referido exemplo pelo sistema de tipos.*



Aqui o nome y é inativo e quando restrito reduz três transições do autômato como mostrado no π-autômato abaixo:



■

4.3 Transformações de Agentes-π

Também trabalhamos com a idéia de construir um novo agente P' à partir do conjunto de nomes ativos encontrado pelo Active-Base-π. Com base no conjunto de nomes ativos podemos gerar um novo agente-π onde é possível eliminar as ações inativas.

Segundo Montanari e Pistore (MONTANARI; PISTORE, 1995) uma ação que contém nomes inativos é considerada uma ação inativa, ou seja, é uma ação que não pode ser executada, não pode ser alcançada.

Pensando assim nesta seção queremos discutir a possibilidade de realizar uma transformação sintática no agente-π eliminando todas as ações consideradas inativas. Desta forma definimos abaixo uma transformação onde retiramos do agente todas as ações inativas.

Uma ação inativa é toda ação que possui nomes inativos. Um nome é inativo se ele está no conjunto $I = fn(P) - A_P$ (onde A_P é o conjunto de nomes ativos de P), ou seja, todos os nomes livres que não são ativos. Note que esse é o mesmo conjunto que definimos na seção anterior.

Definição 8 *Dado um processo P e um conjunto I de nomes inativos de P, a transformação TR(P,I) é definida sobre a estrutura sintática de P como segue:*

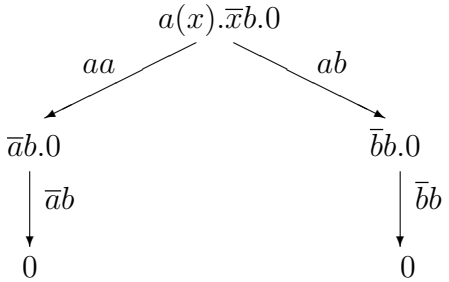
$$\begin{aligned}
TR(0, I) &= 0 \\
TR(\alpha.P, I) &= \begin{cases} \alpha.TR(P, I) & \textit{se} \quad n(\alpha) \cap I = \emptyset \\ 0 & \textit{se} \quad n(\alpha) \cap I \neq \emptyset \end{cases} \\
TR(P + Q, I) &= TR(P, I) + TR(Q, I) \\
TR(P|Q, I) &= TR(P, I)|TR(Q, I) \\
TR((\nu x)P, I) &= (\nu x)TR(P, I) \\
TR(!P, I) &= !TR(P, I)
\end{aligned}$$

Na definição acima $n(\alpha)$ denota o conjunto de nomes que formam o prefixo α (por exemplo, prefixo $\alpha = a(x)$ então $n(a(x)) = \{a, x\}$). Note que ao encontrarmos uma ação inativa todo o restante do agente pode ser eliminado, pois os prefixos que seguem de um inativo nunca poderam ser alcançados e podem ser eliminados do agente. A seguir mostramos alguns exemplos de aplicação da transformação e da construção do π-autômato gerado pelo novo agente.

Exemplo 19 *Transformação do agente $P = (\nu y)\ a(x).\overline{x}b.\overline{y}c.0$ a partir do conjunto de nomes ativos $A_P = \{a, b\}$ (coletado no Exemplo 12).*

- $I = fn(P) - A_P = \{a, b, c\} - \{a, b\} = \{c\}$

$$\begin{aligned}
P' &= TR(P, I) \\
&= TR((\nu y)\ a(x).\overline{x}b.\overline{y}c.0, \{c\}) \\
&= (\nu y)\ TR(a(x).\overline{x}b.\overline{y}c.0, \{c\}) \\
&= (\nu y)\ a(x).TR(\overline{x}b.\overline{y}c.0, \{c\}) \\
&= (\nu y)\ a(x).\overline{x}b.TR(\overline{y}c.0, \{c\}) \\
&= (\nu y)\ a(x).\overline{x}b.0
\end{aligned}$$



■

Note neste exemplo que ao eliminarmos o prefixo $\overline{b}c$, eliminamos todas as ocorrências do b e por isso a restrição (νb) não é mais necessária, portanto podemos aplicar as regras de congruência estrutural, mostradas na Seção 2.1.3, e ficamos com $P' = a(x).\overline{x}b.0$.

Exemplo 20 *Transformação do agente* $P = (\nu y, z)\ (\overline{a}b.y(x).\overline{x}c.0 \mid \overline{y}z.d(u).u(w).0)$ *a partir do conjunto de nomes ativos* $A_P = \{a, b, d\}$ *(coletado no Exemplo 15).*

- $I = fn(P) - A_P \Rightarrow \{a, b, c, d\} - \{a, b, d\} = \{c\}$

$$
\begin{aligned}
P' &= TR(P, I) \\
&= TR((\nu y, z)\ (\overline{a}b.y(x).\overline{x}c.0 \mid \overline{y}z.d(u).u(w).0), \{c\}) \\
&= (\nu y, z)\ TR((\overline{a}b.y(x).\overline{x}c.0 \mid \overline{y}z.d(u).u(w).0), \{c\}) \\
&= (\nu y, z)\ (TR(\overline{a}b.y(x).\overline{x}c.0, \{c\}) \mid TR(\overline{y}z.d(u).u(w).0, \{c\})) \\
&= (\nu y, z)\ (\overline{a}b.TR(y(x).\overline{x}c.0, \{c\}) \mid \overline{y}z.TR(d(u).u(w).0, \{c\})) \\
&= (\nu y, z)\ (\overline{a}b.y(x).TR(\overline{x}c.0, \{c\}) \mid (\overline{y}z.d(u).TR(u(w).0), \{c\})) \\
&= (\nu y, z)\ (\overline{a}b.y(x).0 \mid \overline{y}z.d(u).u(w).TR(0, \{c\})) \\
&= (\nu y, z)\ (\overline{a}b.y(x).0 \mid \overline{y}z.d(u).u(w).0)
\end{aligned}
$$

$$
\begin{array}{c}
(\nu y, z)\ (\overline{a}b.y(x).0 \mid \overline{y}z.d(u).u(w).0) \\
\downarrow\ \overline{a}b \\
(\nu y, z)\ (y(x).0 \mid \overline{y}z.d(u).u(w).0) \\
\downarrow\ \tau \\
(\nu y, z)\ (0 \mid d(u).u(w).0) \\
\downarrow\ dd \\
(\nu y, z)\ (0 \mid d(w).0) \\
\downarrow\ dd \\
(\nu y, z)\ (0 \mid 0)
\end{array}
$$

■

Por fim podemos definir duas propriedades para a transformação proposta. A primeira, diz que o agente obtido através da coleta dos nomes ativos e da tranformação proposta é bissimilar ao agente original.

Proposição 5 *Se* $\Gamma \vdash P : \diamond, A_P$ *então* $P \sim TR(P, I)$*, onde* $I = fn(P) - A_P$*.*

Já a segunda propriedade, diz que o novo agente gerado a partir da transformação proposta, também será bem tipado sobre o *Active-Base-*π, e o conjunto de nomes ativos do novo agente será igual os nomes ativos do agente original.

Proposição 6 *Se* $\Gamma \vdash P : \diamond, A_P$ *e* $P' = Tr(P, I)$ *onde* $I = fn(P) - A_P$ *então* $\Gamma \vdash P' : A_{P'}$ *e* $A_P = A_{P'}$*.*

4.4 Relação com outras Abordagens

Nesta seção mostramos uma relação da coleta de nomes ativos proposta neste trabalho, com as abordagens propostas nos trabalhos de (MONTANARI; PISTORE, 1995) e (MELO, 2003).

4.4.1 Relação com a Abordagem de (MONTANARI; PISTORE, 1995)

Como já mencionamos na Seção 3.3 Montanari e Pistore definem um algoritmo de dois passos para gerar um π-autômato levando em consideração apenas os nomes ativos do agente-π. No primeiro passo é gerado um π-*autômato* à partir dos nomes livres do agente, assim como mostrado na Figura 3.2. Esta operação é representada por π-*aut(P)*, onde P é um agente-π.

O segundo passo identifica os nomes ativos sobre o π-*autômato* e gera um novo autômato chamado de *unwindig-automata*, onde em cada estado os nomes inativos são restritos. Esta operação realizada pelo algoritmo é representada por *unw(π-aut(P))*. O *unwindig-automata* gerado corresponde ao π-*autômato* da Figura 3.3.

Desta forma, se nossa abordagem para coleta de nomes ativos encontrar o mesmo conjunto de nomes ativos que a abordagem de Montanari e Pistore, podemos concluir que o π-*autômato* gerado pelo agente resultante da restrição dos nomes inativos que mostramos na Seção 4.2, é igual ao *unwindig-autômato* gerado pela proposta de (MONTANARI; PISTORE, 1995).

Proposição 7 *Se* $\Gamma \vdash P : \diamond, A_P$ *e* $I = fn(P) - A_P$, *então* π*-aut((*ν *I)P) = unw(*π*-aut(P)).*

Em (MONTANARI; PISTORE, 1995) os nomes ativos são coletados sobre o π-*autômato* gerado no primeiro passo. Esta coleta é realizada em cada transição considerando os nomes ativos e inativos da transição, mais os nomes ativos encontrados no estado resultante da transição. Formalmente, um nome x qualquer é ativo em uma transição, se e somente se, $x \in an(\mu) \cup (\sigma(an(P')) - \overline{an}(\mu))$, onde σ é a substituição que aconteceu na transição, μ é a ação executada e $an(\mu)$ e $\overline{an}(\mu)$ retornam os nomes ativos e inativos da ação μ, conforme definido pela Figura 3.1.

Por exemplo considere a transição $(\nu y, z)(\overline{z}c.0 \mid d(u).u(w).0) \xrightarrow{dc} (\nu y, z)(\overline{z}c.0 \mid c(w).0)$ do primeiro π-autômato mostrado no Exemplo 15. Os nomes ativos do estado $(\nu y, z)(\overline{z}c.0 \mid d(u).u(w).0)$ serão coletados da seguinte forma conforme o algoritmo de Montanari e Pistore:

$$
\begin{aligned}
an(P) &= an(d(u)) \cup (\sigma(an((\nu y, z)(\overline{z}c|c(w)))) - \overline{an}(d(u))) \\
&= \{d\} \cup (\sigma(an(c(w)) \cup (\sigma(an((\nu y, z)(\overline{z}c|0)) - \overline{an}(c(w)))) - \{u\}) \\
&= \{d\} \cup (\sigma(an(c(w)) \cup (\sigma(\{\}) - \overline{an}(c(w)))) - \{u\}) \\
&= \{d\} \cup (\sigma(\{c\} \cup (\{\} - \{w\})) - \{u\}) \\
&= \{d\} \cup (\sigma(\{c\} \cup \{\}) - \{u\}) \\
&= \{d\} \cup (\sigma(\{c\}) - \{u\}) \\
&= \{d\} \cup (\{u\} - \{u\}) = \{d\} \cup \{\} = \{d\}
\end{aligned}
$$

Note que σ é a substituição presente na transição, assim ao executar o prefixo $d(u)$ o nome u é substituido no estado seguinte por c, e após coletar os nomes ativos deste estado esta substituição deve ser levada em consideração, por este motivo $\sigma(c) = u$. Com estas

informações podemos traçar um paralelo entre a abordagem de Montanari e Pistore e a abordagem apresentada neste trabalho, considerando para isso cada tipo de transição que pode acontecer no π-autômato.

4.4.1.1 Transições de Saída

Primeiro considere uma transição de saída onde os nomes envolvidos na ação são livres, neste caso pela abordagem de Montanari e Pistore no π-autômato existe uma transição do tipo $\overline{a}b$ sem nenhuma substituição, e pela caracterização dos nomes ativos da Figura 3.1 os dois serão ativos. Estes nomes serão unidos aos ativos do estado seguinte, pois como não existe nenhuma substituição e portanto σ não altera o conjunto obtido no estado seguinte. Em nossa abordagem este tipo de transição é representada pela regra T-OUT$_1$, ou seja, ela verifica se os dois nomes são bem tipados e nenhum é restrito, então os dois nomes serão ativos e serão unidos aos ativos do restante do agente.

Quando o nome b da transição de saída for restrito, então deve-se considerar a extrusão de escopo do nome. Neste caso em (MONTANARI; PISTORE, 1995) o nome a seria ativo e o nome b inativo conforme a caracterização dos nomes ativos, ou seja, o conjunto seria a unido aos ativos do estado seguinte do π-autômato. Neste trabalho esta transição é representada por T-OUT$_2$, que verifica se b é restrito e se a é um link e então diz que os nomes ativos serão a unido ao restante dos nomes ativos no agente.

Na abordagem que propomos existe ainda a regra T-OUT$_3$, que verifica se o nome a da transição é restrito, neste caso esta ação não poderia acontecer, ou seja, o agente encontra-se em *deadlock*. No π-autômato esta ação não gera transição nenhuma como pode ser visto no estado $(\nu y, z)\ (\overline{z}c.0 \mid 0)$ do primeiro π-autômato do Exemplo 15. Esta transição só poderia acontecer em uma comunicação interna do agente e isso é descrito no conjunto de regras T-PAR. Por este motivo o conjunto de ativos ao aplicar a regra T-OUT$_3$ é vázio.

4.4.1.2 Transições de Entrada

Em uma transição de entrada no algoritmo de Montanari e Pistore deve-se levar em consideração o prefixo da ação mais uma substituição σ, por exemplo, o prefixo executado $a(x)$ mais a substituição {b/x}. Neste caso pela caracterização de nomes ativos, a será ativo e o nome x inativo, portanto o conjunto de ativos da transição será a unido ao conjunto de ativos do estado seguinte (P') menos o nome x. Porém o conjunto de nomes ativos de P' sofrerá a influência da substituição σ, de tal forma que se o nome b, que foi o substituto de x em P', aparecer no conjunto de ativos de P', então este será substituído por x ao aplicar σ, assim os nomes envolvidos na substituição são eliminados do conjunto de ativos final. Desta forma podemos dizer que o objeto de uma transição de entrada vai ser sempre excluído do conjunto de ativos do agente, assim a regra T-INP$_1$ de nosso sistema de tipos obtém o mesmo resultado da abordagem de Montanari e Pistore, verificando a tipagem do processo e eliminando do conjunto de ativos o nome que é o objeto da ação (no exemplo x).

Aqui como na transição de saída temos uma regra adicional T-INP$_2$ para o caso do canal de comunicação da ação ser restrito. Neste caso esta ação não acontece, pois o agente encontra-se em *deadlock* e o conjunto de ativos é vázio.

Note que no caso de composição de processos ($P|Q$) os prefixos de entrada e saída são avaliados pelas regras T-PAR$_2$, T-PAR$_3$ e T-PAR$_4$ respectivamente. Nestas regras o conjunto de ativos coletado segue as mesmas idéias expostas acima para as regras T-INP$_1$, T-OUT$_1$ e T-OUT$_2$, mantendo assim a relação com as transições geradas no π-*autômato*

e possibilitando a verificação de comunicação interna nos agentes (ocorrência de um τ).

De forma análoga podemos traçar um paralelo com a abordagem de Montanari e Pistore para as transições silenciosas que são avalidas por T-TAU e T-PAR$_1$. Com isso mostramos que o conjunto de nomes ativos encontrados com a abordagem proposta neste trabalho, é igual ao conjunto de nomes ativos encontrados em Montanari e Pistore, portanto podemos afirmar que o π-*autômato* (mostrado na Seção 4.2) resultante de nossa abordagem equivale ao *unwinding-automata* encontrado em (MONTANARI; PISTORE, 1995).

4.4.2 Relação com a abordagem de (MELO, 2003)

Da mesma forma que a abordagem de Montanari e Pistore podemos elaborar a seguinte relação com a abordagem proposta em (MELO, 2003).

Proposição 8 *Se* $\Gamma \vdash P : \diamond, A_P$ *então* $an(\nu\overrightarrow{x}.\overline{P}) = A_P$*, onde* $\overline{P}$ *é igual a* P *sem as anotações de tipo e,* $\overrightarrow{x}$ *são os nomes livres de* $\overline{P}$ *mapeados para o tipo restrito em* Γ.

A seguir mostramos um esboço da prova da proposição acima conforme duas equações apresentadas no trabalho de Ana Melo.

Equação 2 *(MELO, 2003)* $an(\alpha.P) = fn(\alpha) \cup (an(P) - bn(\alpha))$

Nesta equação α pode ser uma das quatro ações prefixas definidas na Figura 3.1 de caracterização dos nomes ativos, ax, $\overline{a}\nu b$, $\overline{a}b$ ou τ, lembrando que $\overline{a}\nu b$ representa uma ação se saída com extrusão de escopo do nome b. Caso seja uma ação de entrada ax, tem-se a como um nome livre e x como um nome ligado, portanto a regra T-INP$_1$ coleta o mesmo conjunto de ativos da equação acima, ou seja a será ativo e x será eliminado do conjunto.

No caso de uma ação de saída $\overline{a}b$ os dois nomes serão livres e portanto a regra T-OUT$_1$ obtém o mesmo resultado da equação acima. Da mesma forma para a ação $\overline{a}\nu b$ a regra correspondente em nosso sistema será T-OUT$_2$, que elimina dos ativos o nome b que é restrito e portanto ligado conforme a equação. E por fim, para uma ação τ a equação retorna o mesmo resultado da regra T-TAU em nosso sistema.

Equação 3 *(MELO, 2003)*

$$an\big((\nu\overrightarrow{x})\alpha.P\big) = \begin{cases} \{\} & \textit{se} \quad ch(\alpha) \in \overrightarrow{x} \\ fn(\alpha) \cup (an((v\overrightarrow{x} - \{w\})P) - bn(\alpha)) & \textit{se} \quad \alpha = \overline{v}(w) \wedge v \notin \overrightarrow{x} \\ fn(\alpha) \cup (an((v\overrightarrow{x})P) - bn(\alpha)) & \textit{caso contrário} \end{cases}$$

Aqui como na equação anterior α representa uma das quatro ações da Figura 3.1 de caracterização de nomes ativos. Em nosso sistema de tipos a primeira regra aplicada em um agente com restrição é *T-Res*, que coloca no ambiente de tipos os apontamentos de nomes com o tipo restrito. Assim podemos aplicar as demais regras sabendo que os nomes restritos estão com um tipo especial no ambiente de tipos. Esta operação equivale a propagação do conjunto de restrições na verificação do restante do agente na equação 3.

Sendo assim se a ação a ser verificada tiver o canal de comunicação restrito, então o agente encontra-se em *deadlock* e o cojunto de ativos será vazio, isso é verificado na primeira sentença da equação de (MELO, 2003). Em nosso sistema esta situação é verificada nas regras T-INP$_2$ e T-OUT$_3$ que verificam no ambiente de tipos se o canal de comunicação é do tipo restrito, determinando assim que o conjunto de ativos é vazio.

Para o segundo caso da equação a ação é de saída com o canal de comunicação livre e o nome sendo enviado restrito, assim deve-se considerar a extrusão de escopo do nome e por isso o nome enviado é retirado do conjunto de restrições para o restante do agente. Em nosso sistema isso é verificado na regra *T-Out*$_2$ que verifica a tipagem do restante do processo sob um ambiente de tipos, onde o nome enviado é modificado para um tipo link ($\Gamma \Downarrow b : \#T \vdash P : \diamond, A_P$). Assim o restante do agente é verificado sob um novo ambiente onde o nome não é mais do tipo restrito, e portanto o resultado desta regra equivale ao resultado obtido na equação.

Para a última sentença da equação as regras T-INP$_1$, T-OUT$_1$ e T-TAU são aplicadas conforme o tipo da ação, o que retorna como resultado o mesmos conjunto de ativos.

As demais equações do trabalho de (MELO, 2003) podem ser verificadas da mesma forma, encontrando para cada caso uma regra correspondente no sistema de tipos proposto por este trabalho.

5 CONCLUSÃO

Nos últimos anos muitos pesquisadores têm se esforçado para desenvolver técnicas de verificação formal para o cálculo-π. Entre elas destaca-se a verificação de equivalência observacional, por ser muito utilizada na prática para verificar se um determinado processo é equivalente a sua especificação.

Contudo a verificação de equivalência observacional não é um problema trivial. A maioria dos algoritmos destinados a verificação de equivalência são baseados na construção de sistemas de transições rotuladas (π-autômatos), onde o principal problema é o grande número de estados envolvidos na execução do processo, podendo gerar π-autômatos infinitos.

Porém Montanari e Pistore (MONTANARI; PISTORE, 1995) mostram que é possível gerar π-autômatos mais otimizados, através do conjunto de nomes livres e da identificação dos nomes ativos de agentes-π, como mostrado no Capítulo 3.

Pensando assim, apresenta-se neste trabalho uma abordagem para a coleta de nomes ativos baseada em um sistema de tipos. Aqui a coleta dos nomes ativos é realizada através da análise sintática de expressões-π, ao contrário do que é feito no trabalho de (MONTANARI; PISTORE, 1995), que coleta os nomes ativos através de uma estrutura semântica (π-autômatos).

A principal vantagem de se coletar os nomes ativos sobre expressões-π, é que não há necessidade de gerar um π-autômato utilizando os nomes livres não ativos, como mostrado no trabalho de (MONTANARI; PISTORE, 1995). Ou seja, podemos detectar os nomes ativos sobre a expressão e gerar um π-autômato somente com os nomes livres ativos, como mostramos na Seção 4.2.

A caracterização sintática dos nomes ativos, possibilita encontrar o conjunto de nomes inativos do agente-π. Desta forma é possível descobrir partes da expressão que nunca serão alcançadas, pois são precedidas de ações inativas, ou seja, são precedidas de ações contendo nomes inativos. Assim podemos utilizar a coleta dos nomes ativos para reduzir o tamanho de expressões-π como mostrado na Seção 4.3.

A utilização de um sistema de tipos para a coleta de nomes ativos, facilita a elaboração e prova de propriedades essenciais para raciocinar sobre tal coleta de informações. Como sistemas de tipos são sistemas lógicos, técnicas de provas da lógica podem ser aproveitadas para o estudo de propriedades em sistemas de tipos. Além disso os sistemas de tipos são definidos sobre a estrutura sintática de expressões, tornando mais simples o entendimento da coleta e facilitando as provas por indução estrutural.

O maior problema na utilização de sistemas tipos para o cálculo-π atualmente, é a falta de ferramentas que trabalhem com cálculo-π tipado. A maioria das ferramentas usadas hoje para verificação formal, são baseadas em cálculo-π não tipado (Mobility Workbech (VICTOR; MOLLER, 1994), por exemplo), o que torna difícil a implementação de um

algoritmo para realizar somente a coleta de nomes ativos. Para implementar tal algoritmo seria necessário criar uma ferramenta capaz de interpretar agentes escritos em cálculo-π tipado.

5.1 Trabalhos Futuros

O sistema de tipos proposto neste trabalho para coletar os nomes ativos em expressões-π, ainda necessita de uma prova formal para garantir a segunça da coleta realizada, ou seja, é necessário realizar a prova de corretude do *Active-Base-*π (Proposição 4 da Seção 4.2). Com esta prova pode-se afirmar que o sistema de tipos não irá eliminar nenhum possível nome ativo do conjunto final de nomes ativos coletado.

Além desta prova também é necessário formalizar melhor a idéia de tranformações de agentes mostrada na Seção 4.3. Tal transformação foi adicionada neste trabalho, porém ainda necessita de estudos mais detalhados e principalmente da prova da seguinte propriedade:

Proposição 9 *Se* $\Gamma \vdash P : \Diamond, A_P$ *então* $P \sim TR(P, I)$, *onde* $I = fn(P) - A_P$.

Com a formalização desta transformação podermos utilizar a coleta de nomes ativos, aliada a regras de congruência estrutural, para provar a bissimilaridade entre dois agentes-π. Pois como Montanari e Pistore mostram em (MONTANARI; PISTORE, 1995) agentes bissimilares possuem o mesmo conjunto de nomes ativos.

Outro trabalho a se realizar é a construção de um algoritmo de inferência de tipos, que, dado um determinado processo em cálculo-π tipado, verifica se este processo é bem tipado e coleta o conjunto de seus nomes ativos. É importante também verificar a complexidade deste algoritmo, para que ele possa ser integrado de forma efetiva em uma ferramenta automatizada.

REFERÊNCIAS

AHO, A. V.; SETHI, R.; ULLMAN, J. D. **Compiladores:** Princípios, Técnicas e Ferramentas. Rio de Janeiro: LTC, 1995.

DAM, M. On the decidability of process equivalences for the π-calculus. **Theoretical Computer Science**, Amsterdam, v.183, n.2, p.215-228, 1997.

HIRSCHKOFF, D. Automatically Proving up to Bisimulation. In: INTERNATIONAL SYMPOSIUM ON MATHEMATICAL FOUNDATIONS OF COMPUTER SCIENCE, MFCS, 23., 1998, Brno. **Mathematical Foundations of Computer Science 1998:** proceedings. Berlin: Springer-Verlag, 1998. (Lecture Notes in Computer Science, v. 1450).

JONSSON, B.; PARROW, J. Deciding bisimulation equivalences for a class of Non-Finite-State programs. **Information and Computation**, Cambridge, v.107, n.2, p.272-302, 1993.

KOBAYASHI, N. A partial deadlock-free typed process calculus. **ACM Transactions on Programming Languages and Systems**, New York, v.20, n.2, p.436-482, 1998.

KOBAYASHI, N. A type system for lock-free processes. **Information and Computation**. Duluth, MN, USA: Academic Press, v.177, n.2, p.122-159, 2002.

KOBAYASHI, N.; PIERCE B. C.; TURNER D. N. Linearity and the π-calculus. **ACM Transactions on Programming Languages and Systems**. New York, v.21, n.5, p.914-947, 1999.

MELO, A. A study on the Potential Active Names of π-Agents. In: WORKSHOP ON FORMAL METHODS, 6., 2003. **Proceedings.**, Campina Grande: Universidade Federal de Campina Grande, p.198-211, 2003.

MILNER, R. **Comunication and Concurrency**. Englewood Cliffs, New Jersey: Prentice-Hall, 1989.

MILNER, R. **Comunicating and mobile systems :** the π-calculus. Cambridge: Cambridge University, Cambridge, 1999.

MILNER, R.; PARROW, J. A Calculus for Mobile Processes I e II. **Information and Computation**, Edinburgh: University of Edinburgh, v.100, p.1-77, 1992.

MONTANARI, U.; PISTORE, M. Checking bisimilarity for finitary π-calculus. In: INTERNATIONAL CONFERENCE ON CONCURRENCY THEORY, CONCUR, 6., 1995, Philadelphia. **Concurrency Theory:** proceedings. Berlin: Springer-Verlag, 1995. p.42-56. (Lecture Notes in Computer Science, v. 962).

PALSBERG, J. Type-Based Analysis and Applications. In ACM SIGPLAN-SIGSOFT WORKSHOP ON PROGRAM ANALYSIS FOR SOFTWARE TOOLS AND ENGINEERING, PASTE, 3., 2001, Snowbird. **Proceedings.** New York: ACM Press, p.20-27, 2001.

PARROW, J. An Introduction to the π-Calculus. In: **Handbook of Process Algebra**. Amsterdan: Elsevier, 2001. p.476-543.

PIERCE, R. **Types and Programming Languages**. Cambridge: Cambridge University Press, 2002.

PISTORE, M; SANGIORGI, D. A partition refinament algorithm for the π-calculus. In: INTERNATIONAL CONFERENCE ON COMPUTER AIDED VERIFICATION, CAV, 8., 1996, New Brunswick, New Jersey, USA. **Computer Aided Verification:** proceedings. London: Springer-Verlag, 1996, p.38-49. (Lecture Notes In Computer Science, v.1102).

SANGIORGI, D. On the bisimulation proof method. In: **Laboratory for Foundations of Computer Science**, Technical Reports. Disnponível em: <http://www.lfcs.inf.ed.ac.uk/reports/>. Acesso em: 10/10/2004..

SANGIORGI, D.; WALKER, D. **The π-calculus:** a theory of Mobile Processes. Cambridge, UK: Cambridge University Press, 2001. p.517-532.

VICTOR, B.; MOLLER, F. The Mobility Workbench - A Tool for the π-Calculus. In: INTERNATIONAL CONFERENCE ON COMPUTER AIDED VERIFICATION, CAV, 6., 1994, Palo Alto, California, USA. **Computer Aided Verification:** proceedings. London: Springer-Verlag, 1994, p.428-440. (Lecture Notes In Computer Science, v.818).